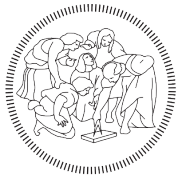

POLITECNICO
MILANO 1863

Department of Computer Science

Data and Information Quality Project Report Guide

Mohammad Amin Abbaszadeh

Ghazal Sepehrirad

Supervisor: Cappiello Cinzia

January 18, 2025

Contents

List of Figures

Chapter 1

Introduction

Abstract of FIFA Dataset:

This data set contains information on football players, detailing their attributes, performance metrics, and other personal and career-related statistics. It comprises 18,979 entries and 77 columns, providing a comprehensive overview of player characteristics.

- Player Information: Includes ID, Name, LongName, Nationality, and Age.
- Club Details: Attributes like Club, Contract, and Joined highlight the player's professional affiliations.
- Performance Metrics: Columns such as ↓OVA (Overall Rating), POT (Potential), and skill-specific attributes like Crossing, Finishing, and Dribbling.
- Physical Characteristics: Contains Height, Weight, and Preferred Foot.
- Financial Information: Features like Value, Wage, and Release Clause reflect economic aspects.
- Skill Breakdown: Includes detailed metrics such as Passing, Shooting, Dribbling, and defensive skills (DEF).
- Goalkeeping Stats: Special attributes for goalkeepers, including GK Diving and GK Reflexes.
- Additional Features: Indicators like Hits, W/F (Weak Foot), SM (Skill Moves), and work rates (A/W, D/W) provide more granular insights.

Chapter 2

Data Quality Assessment

The provided code is designed to assess the quality of a dataset by evaluating three key aspects: completeness, uniqueness, and consistency. These metrics are calculated based on the dataset's structure and the presence or absence of missing or duplicate values.

- **Completeness Evaluation**

  Completeness measures the proportion of non-missing values in the dataset relative to the total number of values.The completeness percentage is calculated as:

$$\text{Completeness} = \frac{\text{Non-missing values}}{\text{Total values}} \times 100$$

- **Uniqueness Evaluation**

  Uniqueness quantifies the proportion of unique values in the dataset compared to the total values.The uniqueness percentage is calculated as:

$$\text{Uniqueness} = \frac{\text{Unique values}}{\text{Total values}} \times 100$$

- **Consistency Evaluation**

  Consistency measures the proportion of unique values relative to the total non-missing values in the dataset.The ratio of unique values to non-missing values is computed, expressed as a percentage:

$$\text{Consistency} = \frac{\text{Unique values}}{\text{Non-missing values}} \times 100$$

- **Utility**

  - This assessment provides a foundational framework for analyzing the quality of a dataset.
  - By evaluating completeness, uniqueness, and consistency, it offers insights into the reliability and usability of the data.

  - These metrics guide data cleaning and preprocessing efforts, ensuring robust results for further analysis and modeling.

- **Timeliness**

  - Timeliness measures the extent to which the age of the data is appropriate for the task at hand. It evaluates how current the data is relative to the requirements of the analysis. **Implementation Process**:

  - The `Currency` is calculated as the difference (in days) between the current date and the date when the data was last updated.

  - The `Timeliness` score is computed for each record based on the formula above.

  - If the `Currency` exceeds the `Volatility`, the `Timeliness` score is set to 0; otherwise, it is scaled proportionally.

```
null_cnt= dataset.isnull().sum().sum()
not_null_cnt = dataset.notnull().sum().sum()
total_cnt = dataset.size
print("The dataset has a total of {} missing values".format(null_cnt))
print("The dataset has a total of {} non missing values".format(not_null_cnt))
print("The dataset has a total of {} values".format(total_cnt))

completeness = not_null_cnt/total_cnt
completeness = '{0:.1f}%'.format(completeness*100)
print("Completeness Evaluation: "+completeness)

#Uniqueness
unique_cnt = dataset.nunique().sum()
print("The dataset has a total of {} unique values".format(unique_cnt))
uniqueness = unique_cnt/total_cnt
uniqueness = '{0:.1f}%'.format(uniqueness*100)
print("Uniqueness Evaluation: "+ uniqueness)

#Consistency
consistenciness = unique_cnt / not_null_cnt
consistenciness = '{0:.1f}%'.format(consistenciness*100)
print("Consistenciness Evaluation: "+ consistenciness)
```

```
The dataset has a total of 0 missing values
The dataset has a total of 1861545 non missing values
The dataset has a total of 1861545 values
Completeness Evaluation: 100.0%
The dataset has a total of 13651 unique values
Uniqueness Evaluation: 0.7%
Consistenciness Evaluation: 0.7%
Average Timeliness: 0.0
Maximum Timeliness: 0.0
Minimum Timeliness: 0.0
```

Figure 2.1: General Structure of Research Steps

- **Accuracy**

  - We could not calculate the accuracy for this part because we do not have additional dataset available.

Chapter 3

Data Profiling

3.1 Data Profiling with Automated Tools

Definition: Data profiling is the systematic process of analyzing and summarizing datasets to understand their structure, content, and quality. It provides insights into the dataset, which helps in identifying potential issues such as missing values, outliers, and inconsistencies before performing any analysis or modeling.

Process and Tool Used:

- The Python library `ydata_profiling` was used to generate an automated profiling report of the dataset.
- The following code snippet demonstrates the process:

```
from ydata_profiling import ProfileReport
PROFILE = ProfileReport(dataset, title="Profiling Report")
PROFILE.to_file("fifa_profiling.html")
```

- The `ProfileReport` function creates an HTML report containing detailed statistics and visualizations for the dataset and its individual columns.

Report Overview: The profiling report consists of two major sections:

1. **Dataset Overview**:
   - **Number of variables:** 77 attributes in the dataset.
   - **Number of observations:** 18,979 rows (entries).
   - **Missing values:** A total of 20,561 missing values, which accounts for 1.4% of the dataset.
   - **Duplicate rows:** No duplicate rows were found in the dataset.
   - **Memory usage:** The dataset occupies 11.1 MiB in memory.
2. **Column-Wise Profiling**:
   - For each column, detailed metrics are provided, such as:

* **Distinct values:** The number of unique values in the column.
* **Missing values:** The count and percentage of missing data in the column.
* **Descriptive statistics:** Minimum, maximum, mean, and other statistical measures.
* **Distributions:** Histograms or bar plots to visualize the column's data distribution.

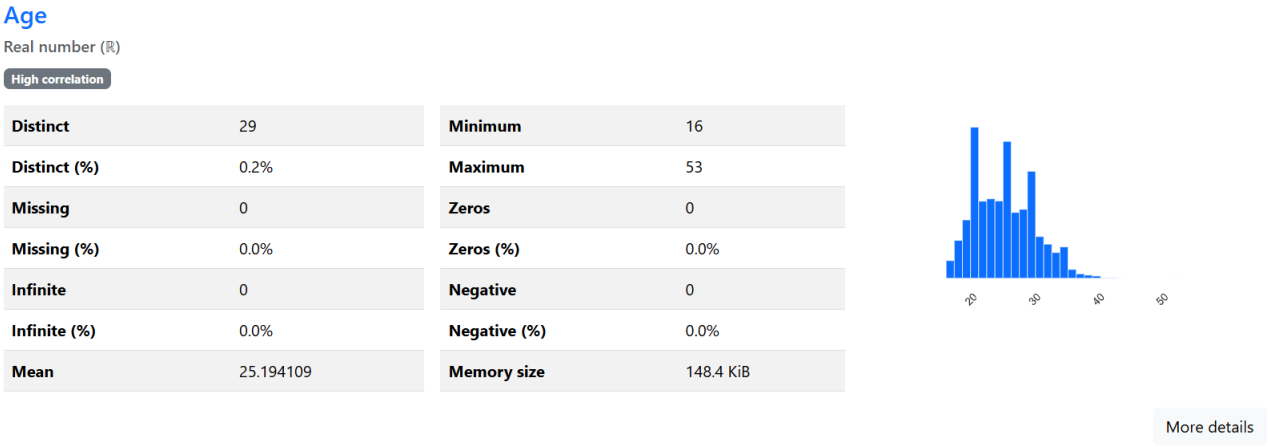

Figure 3.1: statistical summary and the distribution of the Age column

3.2 Single Column Analysis

– Value distributions summarize the distribution of values within a column . A common representation for value distributions are Histograms.

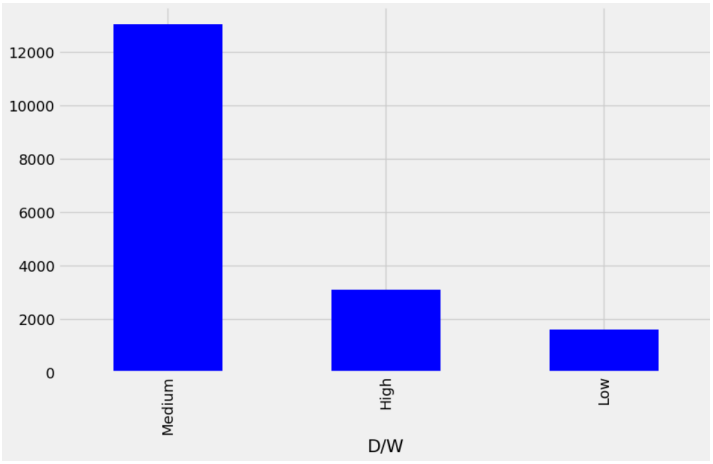

Figure 3.2: (D/W)

3.3 Correlation Analysis and Visualization

Definition: Correlation analysis is used to measure the strength and direction of relationships between numerical variables in a dataset. This analysis is critical for understanding patterns and dependencies, as well as for feature selection in machine learning.

Process and Implementation:

- The correlation analysis was conducted using Python. The `scipy.stats` library was used for calculating correlation coefficients, and `seaborn.PairGrid` was used for advanced visualizations.
- Custom functions were created to display correlation coefficients and scatter plots effectively:
  * `corrfunc`: A function to calculate and annotate the Pearson correlation coefficient in scatter plots.
  * `corrdot`: A function to visualize the strength of correlations using bubble sizes and colors.
  * `plot_corr`: A wrapper function to combine scatter plots, density plots, and correlation bubbles in a single grid for all numerical columns.

```python
def corrfunc(x,y, ax=None, **kws):
    """Plot the correlation coefficient in the top left hand corner of a plot."""
    r, _ = pearsonr(x, y)
    ax = ax or plt.gca()
    # Unicode for lowercase rho (ρ)
    rho = '\u03C1'
    ax.annotate(f'{rho} = {r:.2f}', xy=(.1, .9), xycoords=ax.transAxes)

def corrdot(*args, **kwargs):
    corr_r = args[0].corr(args[1], 'spearman')
    corr_text = f"{corr_r:2.2f}".replace("0.", ".")
    ax = plt.gca()
    ax.set_axis_off()
    marker_size = abs(corr_r) * 10000
    ax.scatter([.5], [.5], marker_size, [corr_r], alpha=0.6, cmap="Blues",
               vmin=-1, vmax=1, transform=ax.transAxes)
    font_size = abs(corr_r) * 40 + 5
    ax.annotate(corr_text, [.5, .5,],  xycoords="axes fraction",
                ha='center', va='center', fontsize=font_size)

# g = sns.pairplot(stocks,palette=["Blues_d"])

def plot_corr(dataset : pd.DataFrame , numerical_columns : list):
    g = sns.PairGrid(dataset[numerical_columns], aspect=1.4, diag_sharey=False)
    g.map_lower(corrfunc)
    g.map_lower(sns.regplot,  ci=False, line_kws={'color': 'Black','linewidth':1})
    g.map_diag(sns.distplot, kde_kws={'color': 'Black','linewidth':1})
    g.map_upper(corrdot)
    plt.figure(figsize=(20, 20))
    plt.show()
```

Figure 3.3: Correlation of the Numerical Features code

Visualization and Interpretation:

* A correlation matrix was generated, combining scatter plots, density plots, and correlation bubbles:
  - **Lower Triangle:** Scatter plots with annotated Pearson correlation coefficients for pairwise comparisons of variables.

- **Diagonal:** Density plots (histograms with KDE) of individual numerical variables to show their distributions.
- **Upper Triangle:** Bubble charts representing correlation strength, where bubble size and color intensity indicate the magnitude of correlation.

* The visualization allows for an intuitive understanding of variable relationships:
  - Positive correlations are indicated by larger, darker bubbles, and negative correlations are smaller and lighter.
  - Example: Variables such as `Power` and `Strength` exhibit strong positive correlations (bubble size ~ 0.80), while weak correlations are nearly invisible.

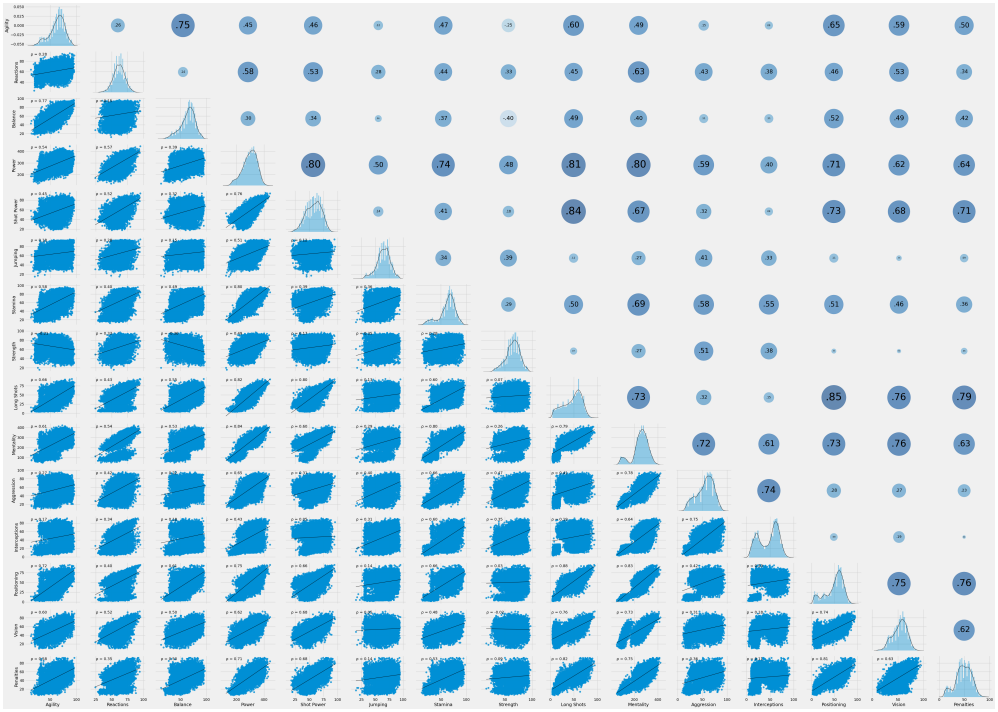

Figure 3.4: Correlation of the Numerical Features

Chapter 4

Data Cleaning

4.1 Data Transformation

Overview: The data transformation step is essential for preparing the dataset for analysis. It involves cleaning and reformatting the data to ensure consistency, handle missing values, and convert data types to suitable formats for downstream tasks.

Steps Performed:

1. **Visualization of All Columns**:
   * All columns in the dataset were visualized to identify potential issues, such as missing values, irrelevant data, and inconsistencies.
   * Through visualization, we identified specific columns that required transformation or removal.
2. **Handling Missing Values and Irrelevant Columns**:
   * Due to the high percentage of missing values in the `Loan Date End` (94.663%) and `Hits` (13.673%) columns.
   * The columns `photoUrl` and `playerUrl` (containing image and link data) were considered irrelevant for analysis and removed.
   * Additional columns such as `ID`, `Name`, and `LongName` were dropped, as they were non-informative for downstream analysis.
3. **Cleaning Text Data**:
   * Any leading or trailing white spaces in text columns were removed using the following code:

```
1        dataset = dataset.apply(lambda x:
    x.str.strip() if x.dtype == "object"
   else x)
2
```

4. **Renaming and Splitting Columns**:
   * The column ↓OVA was renamed to OVA for simplicity.
   * The column LongName was renamed to FullName.
   * The Contract column was split into two separate columns named Contract_Start and Contract_End.
   * The code snippet used:

```
1         dataset.rename(columns={"  OVA  ":
  "OVA"}, inplace=True)
2         dataset.rename(columns={"LongName
  ": "FullName"}, inplace=True)
3
```

5. **Categorical Column Handling**:
   * Non-numerical columns such as Nationality, Club, Contract, Positions, and others were identified for further cleaning.
   * Unique positions from the Positions column (CAM, CB, CDM, ...) were identified, and a one-hot encoding approach was used to create binary columns for each unique position.
   * For the Preferred Foot column, we converted the left value to 0 and the right value to 1 to make them numerical.
   * The Joined column contains strings representing dates; these values were converted to datetime type.
6. **String to Numeric Conversion**:
   * For certain columns containing numerical values represented as strings (e.g., with symbols like *), the strings were converted to integers, and unwanted characters were removed.
   * Example: "5*" -> 5.
   * We standardized the pattern of all values in the Contract column to the format: <Start_year ~ End_year>.
   * Weight values were converted to kilograms, and height values were converted to centimeters for uniformity.
   * For financial columns (Value and Wage), we cleaned the data and converted these columns into numeric values in millions.
7. **Final Dataset Alignment**:
   * At the end of the transformation, the dataset was checked for consistency.
   * All remaining columns were confirmed to have appropriate data types for analysis.
   * Only two categorical columns were retained for simplicity, and they can be converted later as required.

4.2 Missing Value Analysis

Overview: Identifying and handling missing values is a crucial step in data preprocessing to maintain data quality and ensure the reliability of subsequent

analysis.

Code Implementation and Explanation

The missing values in the dataset were analyzed using the following steps:

1. A bar plot was generated to visualize the missing values across all columns using the `missingno` library:

```
import missingno as msno
msno.bar(dataset, color="seagreen", log=True, sort="descending")

```

   This visual representation helps in quickly identifying columns with significant amounts of missing data.

2. The number and percentage of missing values in each column were calculated using Python:

```
{col: [dataset[col].isnull().sum(),
        f'% {np.round(np.mean(dataset[col].isnull()) * 100, 3)}']
  for col in dataset.columns if dataset[col].isnull().any()}

```

   * `dataset[col].isnull().sum()`: Counts the number of missing values in each column.
   * `np.mean(dataset[col].isnull()) * 100`: Computes the percentage of missing values relative to the total number of rows.
   * Only columns with at least one missing value were included in the output.

Results

The analysis revealed the following columns:

* **Loan Date End:** 17,966 missing values, which accounts for 94.663% of the dataset.
* **Hits:** 2,595 missing values, which accounts for 13.673% of the dataset.

Rationale for Removing Columns

* **Loan Date End:**
  - This column has over 94% missing values, making it highly incomplete.

- Imputation or interpolation for such a large percentage of missing values would not provide reliable results.
- The column's relevance to the analysis is minimal, making it a candidate for removal.

* **Hits:**
  - The percentage of missing values in the *Hits* column (13.67%) is not excessively high, we decided to fill the missing values with the mean of the column. This approach allows us to retain the column without introducing significant bias.

4.3 Outlier Detection and Removal

Interquartile Range (IQR) Method

The Interquartile Range (IQR) method is a robust statistical technique to detect and remove outliers. It calculates the range between the first quartile ($Q1$) and the third quartile ($Q3$):

$$\text{IQR} = Q3 - Q1$$

Data points are classified as outliers if they lie outside the following bounds:

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}, \quad \text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

This method is effective because it is not influenced by extreme values, making it suitable for datasets with skewed or non-normal distributions.

Implementation

The following Python code was used to apply the IQR method:

```
def find_outliers_iqr(data, columns):
    outlier_indices = {}
    for column in columns:
        Q1 = data[column].quantile(0.25)
        Q3 = data[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)].index
        outlier_indices[column] = outliers.tolist()
    return outlier_indices

outliers_iqr = find_outliers_iqr(dataset, numerical_columns)
```

Results

Before Outlier Removal: The distributions of all columns before removing outliers are shown in the first histogram. Additionally, the box plots highlight the presence of extreme values that deviate significantly from the main data distribution.

After Outlier Removal: The second histogram and updated box plots demonstrate that the IQR method effectively reduced the number of outliers. The cleaned data now has a more symmetric and focused distribution.

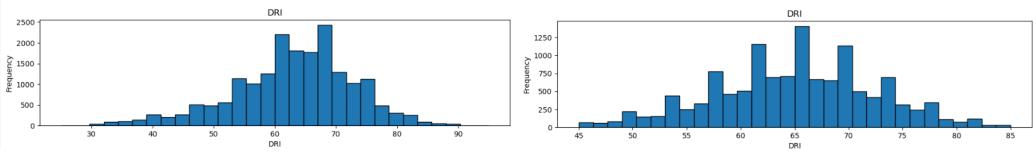

Figure 4.1: Distribution of DRI before and after outlier

Columns Classification

Based on the analysis, the following columns were classified as having outliers or not:

* **Columns with Outliers:** Age, OVA, POT, BOV, Attacking, Heading Accuracy, Short Passing, Dribbling, Ball Control, Acceleration, Stamina, Sprint Speed, Goalkeeping, etc.
* **Columns without Outliers:** Crossing, Finishing, Volleys, Curve, Standing Tackle, Interceptions, etc.

Discussion of Specific Columns

* **Age:** The distribution was slightly skewed, with some older players marked as outliers. After removal, the data focuses on the typical age range of players.
* **OVA and POT:** Both columns initially had extreme outliers, likely representing exceptional players. The cleaned data highlights the central trend without these extremes.
* **Goalkeeping:** The high number of outliers indicates variability among players with exceptional goalkeeping skills. Removing these helped stabilize the distribution.
* **Acceleration, Stamina, and Sprint Speed:** These columns had significant outliers, potentially due to exceptional athletes. Their removal results in more symmetric distributions.

The outliers detected for some column were summarized as:

Column	Number of Outliers
OVA	156
POT	151
PAC	424
SHO	0
PAS	177
DRI	426
DEF	0
PHY	109
Acceleration	754
Sprint Speed	683
Stamina	840
Strength	116

Analysis of the Age Column Outliers

Overview: The 'Age' column was analyzed separately due to its unique distribution and the presence of outliers. The boxplots and histograms provide insights into the distribution before and after outlier removal.

Boxplot of Age Before Removing Outliers

* The boxplot reveals a well-centered distribution for the majority of the data, with most values falling between 20 and 30 years.
* A few data points lie beyond the upper whisker, representing potential outliers. These values extend up to approximately 50 years.
* The presence of these outliers indicates a need for removal, as they deviate significantly from the main body of the data.
* Statistical bounds for identifying outliers were calculated using the Interquartile Range (IQR) method:
  - Lower Bound: 11.05
  - Upper Bound: 39.45
* A total of 26 outliers were identified based on these bounds.

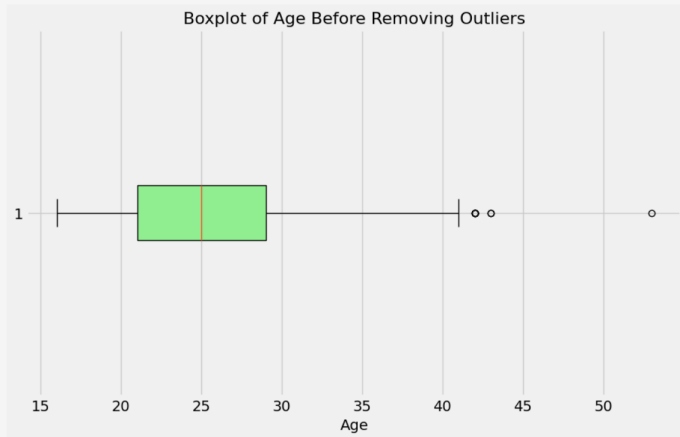

Figure 4.2: Boxplot of Age Before Removing Outliers

Histogram of Age with Outlier Regions Highlighted

* The histogram visualizes the distribution of the 'Age' column alongside the calculated bounds.
* The shaded red regions represent values below the lower bound and above the upper bound, identifying the outlier regions.
* Most of the data is concentrated within the bounds, between ages 20 and 30.
* The outliers include players above 40 years, which are rare and may not align with the typical player age range.

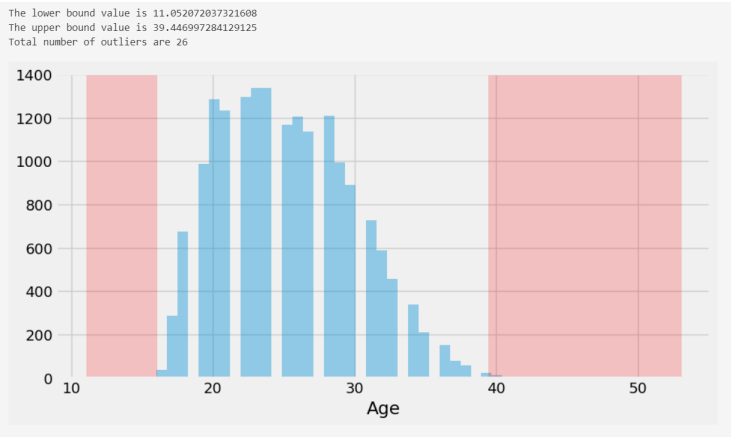

Figure 4.3: Histogram of Age with Outlier Regions Highlighted

Boxplot of Age After Removing Outliers

* The boxplot after outlier removal shows a more compact distribution, with the upper whisker ending near the calculated upper bound (39.45).

* The data is now more symmetric, focusing on the core distribution without the influence of extreme values.
* The removal of outliers ensures that the central tendency and spread of the data better reflect the true population.

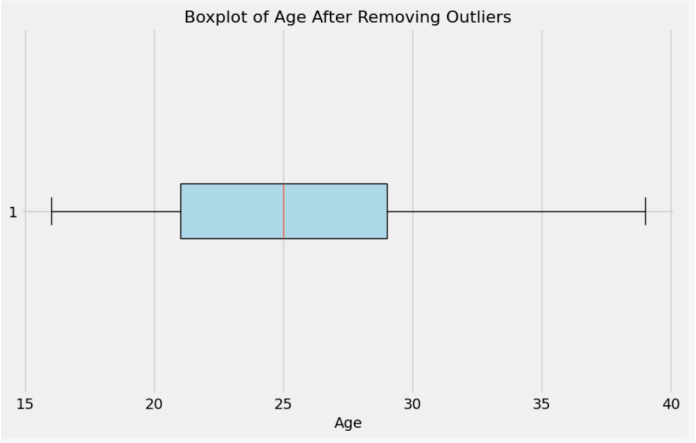

Figure 4.4: Boxplot of Age After Removing Outliers

4.4 Duplicate Check

Using the following code:

```
print(dataset.duplicated().sum())
```

We verified that the dataset contains no duplicate rows, as the result returned is 0. This ensures that the dataset is free from redundancy and ready for further analysis.

Chapter 5

Conclusions and Future Work

5.1 Libraries Used

* **numpy (np)**
* **pandas (pd)**
* **dotenv (load_dotenv)**
* **datetime**: working with dates and times
* **plotly.express (px)**
* **ydata_profiling (ProfileReport)**: Used for generating detailed profiling reports of datasets
* **missingno (msno)**: visualizing missing data patterns in datasets.
* **math**: access to mathematical functions and constants.(for outlier part)
* **matplotlib (mpl)**:
* **matplotlib.pyplot (plt)**: creating figures and plotting graphs.
* **seaborn (sns)**

5.2 Data Visualization

After completing the data cleaning process, our dataset is now prepared for visualization and analysis. The cleaned data enables us to explore various aspects of the dataset effectively and derive meaningful insights. Below, we present two examples of visualizations created using the cleaned data:

- **Distribution of Nationalities:** The map shown in Figure 1 illustrates the distribution of players' nationalities across the world. Each country is color-coded, and hovering over a country displays additional information, such as the number of players from that country. This visualization highlights the global diversity of players in our dataset.

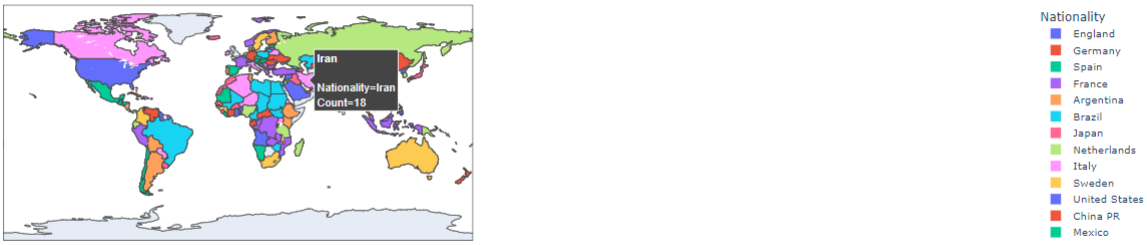

Figure 5.1: Distribution of Nationalities

- **Overall Rating vs. Age:** Figure 2 presents a scatter plot of players' overall ratings against their ages. The red regression line indicates a general trend of higher ratings being associated with older players, although the distribution also shows variability across age groups. This visualization provides insights into the relationship between players' experience and their overall performance ratings.

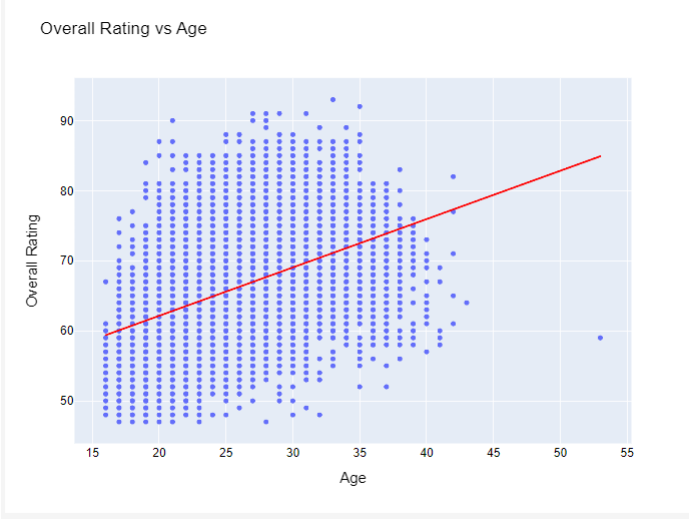

Figure 5.2: Age vs Overall Rating

These visualizations demonstrate the usefulness of the cleaned dataset in exploring patterns and relationships among different attributes. Further analysis can be performed based on the insights gained from such visual representations.