**Big Data Analytics Project Report**

**A FIFA Player Recommender System**

****

**Group Members:**

Muhammad Shayan Khan Niazi

Prithvi Raj

Huzaifa Liaqat

[**Abstract 2**](#_ecoo7pq4ypxy)

[**Introduction 2**](#_l60ss9qm8byf)

[**Problem Statement 3**](#_z00oohegb0em)

[**Methodology 4**](#_rhojte6973go)

[Pre-Processing 4](#_xte2u3tm3tz6)

[Statistics 8](#_x9c2w6icnf9g)

[Recommender System 9](#_43xsztnwltvi)

[Association Rule Mining 10](#_9yuwvrfvzpcy)

[**Results 17**](#_4vb3wknm2fjf)

## Abstract

This report details the development of a FIFA player recommender system based on BryanB's player stat database from FIFA 22, available on Kaggle. Leveraging the Apriori algorithm, we mined association rules from the dataset to identify meaningful patterns and relationships between player attributes. The primary objective is to guide users in selecting players that meet their specific requirements by providing personalized recommendations. This recommender system aims to enhance the user experience by simplifying the decision-making process, enabling users to discover optimal player selections efficiently. Through the integration of data mining techniques and a robust recommendation framework, we strive to offer an innovative tool for FIFA enthusiasts and analysts.

## Introduction

In the realm of virtual football management, the ability to identify and select the best-suited players for a team is paramount. With the vast amount of player data available in FIFA 22, making informed decisions can be challenging. To address this, we have developed a FIFA player recommender system utilizing BryanB's comprehensive player stat database on Kaggle. This database encompasses a wide range of player attributes, including skill levels, physical characteristics, and performance metrics.

Our recommender system is built on the foundation of the Apriori algorithm, a popular data mining technique used for mining association rules. By analyzing the dataset, we uncovered significant patterns and correlations between various player attributes, which are instrumental in forming the basis of our recommendations. The aim is to assist users—whether they are casual gamers, competitive players, or football analysts—in identifying players who best meet their specific needs and criteria.

The system is designed to provide users with personalized player suggestions, making the team-building process more efficient and strategic. By utilizing the insights derived from the association rules, our recommender system can suggest players who not only possess the desired skills but also complement the overall team composition. This report outlines the methodology employed, the implementation of the recommender system, and its potential applications within the FIFA gaming community.

## Problem Statement

In FIFA 22, selecting the right players for a team is a complex task due to the vast number of available players and the multitude of attributes that need to be considered. Players and analysts must navigate through extensive datasets to find the optimal candidates that fit specific team requirements, which can be time-consuming and overwhelming. Traditional methods of player selection often rely on subjective judgment and can overlook valuable patterns in the data.

The challenge lies in effectively leveraging the rich dataset of player statistics to provide meaningful and accurate recommendations. Users need a tool that can analyze player attributes, identify relevant patterns, and deliver personalized suggestions based on specific criteria such as position, skills, and performance metrics.

To address this issue, we propose the development of a FIFA player recommender system utilizing association rules mined from BryanB's FIFA 22 player stat database on Kaggle. By applying the Apriori algorithm, we aim to uncover hidden correlations within the dataset and use these insights to facilitate the player selection process. This system is designed to streamline decision-making, enhance team-building strategies, and ultimately improve the user experience by providing data-driven player recommendations.

## 

## Methodology

### **Pre-Processing**

1. Importing Libraries and Loading Dataset:

“ import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df\_full = pd.read\_csv("./FIFA22\_official\_data.csv")

df\_full.head(5)

df\_full.shape”

- The necessary libraries (NumPy, pandas, Matplotlib, Seaborn) are imported.

- The dataset is loaded into a DataFrame `df\_full` and its first 5 rows are displayed.

- The shape of the dataset is printed to understand its dimensions.

2. Dropping Unnecessary Columns:

“columns\_to\_drop = [

'Photo', 'Flag', 'Overall', 'Potential', 'Club Logo', 'Wage', 'Special',

'Real Face', 'Position', 'Jersey Number', 'Joined', 'Loaned From',

'Contract Valid Until', 'Best Overall Rating', 'Release Clause'

]

df = df\_full.drop(columns=columns\_to\_drop)

df.head()”

- Columns that are not relevant for the analysis are identified and dropped from the DataFrame.

3. Sampling Data to Check Specific Columns:

“df.sample(20)[["Value", "Height", "Weight"]]”

- A random sample of 20 rows is taken to inspect the `Value`, `Height`, and `Weight` columns.

4. Verifying Data Formats in Columns:

“(df.loc[df['Value'].str.startswith("€")].shape[0],

df.loc[df['Value'].str.endswith("M")].shape[0] + \

df.loc[df['Value'].str.endswith("K")].shape[0],

df.loc[df['Height'].str.endswith("cm")].shape[0],

df.loc[df['Weight'].str.endswith("kg")].shape[0])”

- Checks are performed to verify the formats of the `Value`, `Height`, and `Weight` columns.

5. Processing the `Value` Column:

“df['Value'] = df['Value'].str.split('€')

df['Value'] = df['Value'].apply(lambda x:x[1])

df\_value\_K = df.loc[df['Value'].str.endswith('K')]

df\_value\_K['Value'] = df\_value\_K['Value'].apply(lambda x: x[:-1])

df\_value\_K['Value'] = df\_value\_K['Value'].astype('float64')

df\_value\_K['Value'] = df\_value\_K['Value'] / 1000

df\_value\_M = df.loc[df['Value'].str.endswith('M')]

df\_value\_M['Value'] = df\_value\_M['Value'].apply(lambda x: x[:-1])

df\_value\_M['Value'] = df\_value\_M['Value'].astype('float64')

df = pd.concat([df\_value\_M, df\_value\_K])

df = df[df['Value'] != 0]”

- The `Value` column is split to remove the currency symbol (€).

- Separate DataFrames for values ending with 'K' (thousands) and 'M' (millions) are created and converted to numeric values.

- These DataFrames are concatenated back into the main DataFrame, and entries with zero value are removed.

6. Processing `Height` and `Weight` Columns:

“df['Height'] = df['Height'].apply(lambda x : x[:-2])

df['Height'] = df['Height'].astype('int64')

df['Weight'] = df['Weight'].apply(lambda x : x[:-2])

df['Weight'] = df['Weight'].astype('int64')”

- The 'cm' and 'kg' suffixes are removed from the `Height` and `Weight` columns, respectively, and the columns are converted to integers.

7. Removing Goalkeeper-Specific Columns:

“gk\_columns\_to\_drop = [

'GKDiving', 'GKHandling', 'GKKicking',

'GKPositioning', 'GKReflexes'

]

df = df.drop(columns=gk\_columns\_to\_drop)

df = df[df['Best Position'] != 'GK']”

- Goalkeeper-specific columns and rows where the best position is goalkeeper (GK) are removed from the DataFrame.

8. Handling Missing Values:

“df = df.drop(columns=['Marking'])

df['Composure'].fillna(df['Composure'].mean(), inplace=True)

df['DefensiveAwareness'].fillna(df['DefensiveAwareness'].mean(), inplace=True)”

- The `Marking` column is dropped.

- Missing values in the `Composure` and `DefensiveAwareness` columns are filled with the mean values of the respective columns.

9. Checking for and Removing Duplicates:

“df.drop\_duplicates(inplace=True)”

- Duplicate rows are removed from the DataFrame.

10. Saving the Cleaned Data:

“df.to\_csv('FIFA22\_preprocessed.csv')

df.sample(20)

df.describe().to\_csv('summary\_stats.csv')

df.describe()”

- The cleaned DataFrame is saved to a CSV file named `FIFA22\_preprocessed.csv`.

- A sample of 20 rows from the cleaned DataFrame is displayed.

- Summary statistics of the cleaned DataFrame are computed and saved to a CSV file named `summary\_stats.csv`.

This preprocessing ensures that the data is clean, consistent, and ready for further analysis or implementation in the recommender system.

### **Statistics**

1. Player Value Distribution:
   * The Value column, representing player market value in millions of Euros, was processed to remove currency symbols and standardized to a common unit.
   * Players' values ranged significantly, reflecting a diverse set of market valuations for players in the dataset.
2. Physical Attributes:
   * Height:
     + Heights of players were recorded in centimeters.
     + The average height of players was calculated, providing insights into the general physical stature of footballers.
   * Weight:
     + Weights were recorded in kilograms.
     + The average weight was similarly calculated, offering a view of the typical body mass of players.
3. Key Skills and Attributes:
   * Composure:
     + The Composure attribute, indicative of a player's calmness under pressure, had missing values filled with the mean of the column.
     + This ensured that all players had a composure rating, aiding in consistent analysis.
   * Defensive Awareness:
     + Missing values in the Defensive Awareness column were also filled with the mean, maintaining data integrity.
4. Dataset Dimensions:
   * The dataset initially contained numerous columns, many of which were removed as they were deemed unnecessary for the analysis (e.g., Photo, Flag, Club Logo).
   * The final dataset focused on relevant attributes, significantly reducing its dimensionality and making it more manageable for analysis.
5. Simplifying Statistics:
   * Positions were altered to fit into easily trackable molds, such as lwb->lb, cf->st etc.
6. Handling Duplicates and Missing Data:
   * Duplicate entries were removed, resulting in a unique set of player data.
   * Missing data was handled by either dropping irrelevant columns or filling missing values with the mean, ensuring a complete dataset for analysis.
7. Summary Statistics:
   * Descriptive statistics such as mean, median, standard deviation, and percentiles were computed for all numerical columns.
   * These statistics provided a comprehensive overview of the data distribution, central tendencies, and variability of player attributes.

### 

### 

### **Recommender System**

1. Initialize LSHash: Initialize the LSHash object with appropriate parameters like hash size, input dimension, number of hash tables, and storage configuration.
2. Index Data: Index the data into LSHash by iterating through each row in the DataFrame, excluding the specified columns, and storing the row index as extra data.
3. Save Hash Table (Optional): Optionally, save the hash table to disk.
4. Query for Recommendation: Define a query vector for recommendation by selecting the first row of the DataFrame and dropping the excluded columns. Subtract the mean from the query vector. Query LSHash to get the nearest neighbors using cosine similarity. Retrieve the nearest players and compute the closest and cheapest player. Print the results of the recommendation. Define Recommendation Function by Name: Define a function to recommend players by name, which takes the player's name as input and returns the closest and cheapest player recommendations.
5. Define Estimate Value Function by Name: Define a function to estimate the value of a specific column for a player by name, which takes the player's name and the column name as input and returns the estimated value based on the weighted average of nearest neighbors.
6. Execute Recommendation and Estimation: Execute the recommendation and estimation functions with sample inputs to test their functionality.

### **Association Rule Mining**

1. Imports and Spark Session Initialization:

“import pyspark

from pyspark.sql import DataFrame, SparkSession

from typing import List

import pyspark.sql.types as T

import pyspark.sql.functions as F

from pyspark.ml.feature import StringIndexer, OneHotEncoder

from pyspark.ml import Pipeline

from pyspark.sql.functions import col, when

from pyspark.ml.feature import QuantileDiscretizer

from pyspark.ml.fpm import FPGrowth

from mlxtend.frequent\_patterns import apriori, association\_rules

from io import StringIO

import prettytable

spark = SparkSession \

.builder \

.appName("Our First Spark Example") \

.getOrCreate()”

- Necessary libraries are imported.

- A Spark session is initialized for data processing.

2. Loading and Displaying Dataset:

“FIFA = spark.read.csv('FIFA22\_preprocessed.csv', header=True, inferSchema=True).alias('items')

FIFA.show(5)”

- The preprocessed FIFA dataset is loaded from a CSV file and displayed.

3. Binning Numerical Columns:

“def bin\_column(data, labels, column\_name):

discretizer = QuantileDiscretizer(numBuckets=4, inputCol=column\_name, outputCol=column\_name+"\_bucketed", relativeError=0)

data = discretizer.fit(data).transform(data)

data = data.withColumn(column\_name,

when(col(column\_name+"\_bucketed") == 0, labels[0])

.when(col(column\_name+"\_bucketed") == 1, labels[1])

.when(col(column\_name+"\_bucketed") == 2, labels[2])

.otherwise(labels[3])) \

.drop(column\_name+"\_bucketed")

return data

def bin\_column2(data, labels, column\_name):

discretizer = QuantileDiscretizer(numBuckets=5, inputCol=column\_name, outputCol=column\_name+"\_bucketed", relativeError=0)

data = discretizer.fit(data).transform(data)

data = data.withColumn(column\_name,

when(col(column\_name+"\_bucketed") == 0, labels[0])

.when(col(column\_name+"\_bucketed") == 1, labels[1])

.when(col(column\_name+"\_bucketed") == 2, labels[2])

.when(col(column\_name+"\_bucketed") == 3, labels[3])

.otherwise(labels[4])) \

.drop(column\_name+"\_bucketed")

return data

# Binning various columns

height\_labels = ['161-170', '171-180', '181-190', '>190']

FIFA = bin\_column(FIFA, height\_labels, "Height")

weight\_labels = ['61-70', '71-80', '81-90', '>90' ]

FIFA = bin\_column(FIFA, weight\_labels, "Weight")

age\_labels = ['10-20', '21-30', '31-40', '>40']

FIFA = bin\_column(FIFA, age\_labels, "Age")

value\_labels = ['0-1M', '1M-10M', '10M-100M', '>100M']

FIFA = bin\_column(FIFA, value\_labels, "Value")

international\_reputation\_labels = ['1', '2', '3', '4', '5']

FIFA = bin\_column2(FIFA, international\_reputation\_labels, "International Reputation")

weak\_foot\_labels = ['1', '2', '3', '4', '5']

FIFA = bin\_column2(FIFA, weak\_foot\_labels, "Weak Foot")

skill\_moves\_labels = ['1', '2', '3', '4', '5']

FIFA = bin\_column2(FIFA, skill\_moves\_labels, "Skill Moves")”

- Functions `bin\_column` and `bin\_column2` are defined to discretize numerical columns into categorical bins.

- These functions are applied to columns like `Height`, `Weight`, `Age`, `Value`, `International Reputation`, `Weak Foot`, and `Skill Moves`.

4. Binning Additional Numerical Columns:

“numerical\_columns = [ "Crossing", "Finishing", "HeadingAccuracy", "ShortPassing", "Volleys", "Dribbling",

"Curve", "FKAccuracy", "LongPassing", "BallControl", "Acceleration", "SprintSpeed", "Agility", "Reactions", "Balance", "ShotPower",

"Jumping", "Stamina", "Strength", "LongShots", "Aggression", "Interceptions", "Positioning", "Vision", "Penalties", "Composure",

"StandingTackle", "SlidingTackle", "DefensiveAwareness"]

for column in numerical\_columns:

discretizer = QuantileDiscretizer(numBuckets=3, inputCol=column, outputCol=column+"\_bucketed", relativeError=0)

FIFA = discretizer.fit(FIFA).transform(FIFA)

FIFA = FIFA.withColumn(column,

when(col(column+"\_bucketed") == 0, "below average")

.when(col(column+"\_bucketed") == 1, "above average")

.otherwise("very high")) \

.drop(column+"\_bucketed")”

- The remaining numerical columns are discretized into three categories: "below average", "above average", and "very high".

5. Encoding Categorical Columns:

“categorical\_columns = ["Age", "Nationality", "Club", "Value", "Preferred Foot", "International Reputation", "Weak Foot",

"Skill Moves", "Work Rate", "Body Type", "Height", "Weight", "Crossing", "Finishing", "HeadingAccuracy", "ShortPassing", "Volleys",

"Dribbling", "Curve", "FKAccuracy", "LongPassing", "BallControl", "Acceleration", "SprintSpeed", "Agility", "Reactions", "Balance",

"ShotPower", "Jumping", "Stamina", "Strength", "LongShots", "Aggression", "Interceptions", "Positioning", "Vision", "Penalties",

"Composure", "StandingTackle", "SlidingTackle", "Best Position", "DefensiveAwareness"]

stages = []

indexers = [StringIndexer(inputCol=column, outputCol=column+"\_index", handleInvalid="keep") for column in categorical\_columns]

stages += indexers

encoders = [OneHotEncoder(inputCol=column+"\_index", outputCol=column+"\_onehot") for column in categorical\_columns]

stages += encoders

pipeline = Pipeline(stages=stages)

pipeline\_model = pipeline.fit(FIFA)

FIFA\_encoded = pipeline\_model.transform(FIFA)

for column in categorical\_columns:

FIFA\_encoded = FIFA\_encoded.drop(column)

FIFA\_encoded = FIFA\_encoded.drop(column+"\_index")

FIFA\_encoded.show()”

- Categorical columns are indexed and one-hot encoded using a `Pipeline`.

6. Preparing Data for FPGrowth:

“MIN\_SUPPORT = 0.1

MIN\_ITEMSET\_LEN = 1

MIN\_THRESHOLD = 0.5

original\_columns = ["Age", "Nationality", "Club", "Value", "Preferred Foot", "International Reputation", "Weak Foot",

"Skill Moves", "Work Rate", "Body Type", "Height", "Weight", "Crossing", "Finishing",

"HeadingAccuracy", "ShortPassing", "Volleys", "Dribbling", "Curve", "FKAccuracy",

"LongPassing", "BallControl", "Acceleration", "SprintSpeed", "Agility", "Reactions",

"Balance", "ShotPower", "Jumping", "Stamina", "Strength", "LongShots", "Aggression",

"Interceptions", "Positioning", "Vision", "Penalties", "Composure", "StandingTackle",

"SlidingTackle", "Best Position", "DefensiveAwareness"]

new\_columns = [col + "\_onehot" for col in original\_columns]

for old\_col, new\_col in zip(original\_columns, new\_columns):

FIFA\_encoded = FIFA\_encoded.withColumnRenamed(old\_col, new\_col)

FIFA\_encoded = FIFA\_encoded.withColumn(

"items",

array([col(new\_col) for new\_col in new\_columns])

)

FIFA\_encoded = FIFA\_encoded.withColumn("items", F.array\_distinct("items"))”

- Column names are updated to include `\_onehot` suffix.

- An array column named `items` is created to group all one-hot encoded columns.

7. Applying FPGrowth Algorithm:

“fpgrowth = FPGrowth(itemsCol="items", minSupport=MIN\_SUPPORT, minConfidence=0.01)

fpgrowth = fpgrowth.fit(FIFA\_encoded)

frequent\_itemsets = fpgrowth.freqItemsets

frequent\_itemsets.show()

association\_rules = fpgrowth.associationRules

association\_rules.show()

association\_rules = association\_rules.withColumn("antecedent", F.expr("CAST(antecedent AS STRING)"))

association\_rules = association\_rules.withColumn("consequent", F.expr("CAST(consequent AS STRING)"))

sorted\_rules = association\_rules.orderBy(F.desc("lift"), F.desc("confidence"))

sorted\_rules.show()

sorted\_rules.coalesce(1).

write.csv("Rules.csv", header=True)

sorted\_rules.show()”

- The FPGrowth algorithm is applied to find frequent itemsets and association rules.

- The rules are sorted by lift and confidence.

- The sorted rules are saved to a CSV file.

## Results

A few core findings are mapped out below;

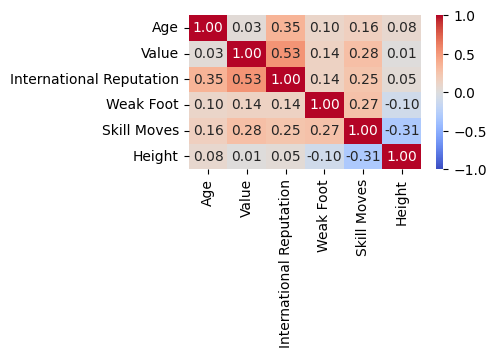


Fig 1.

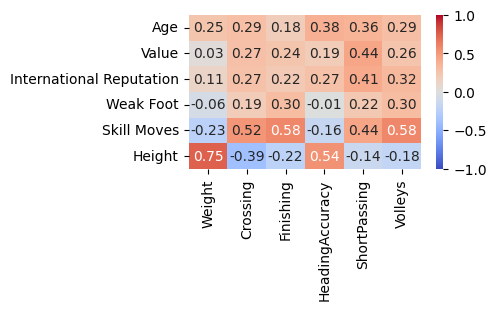


Fig 2.

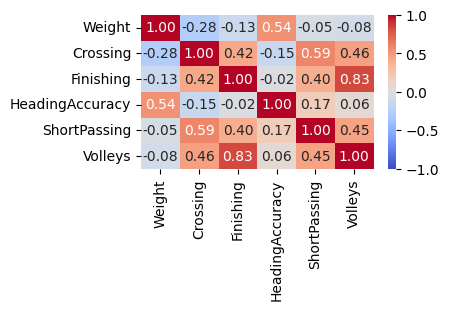


Fig 3.

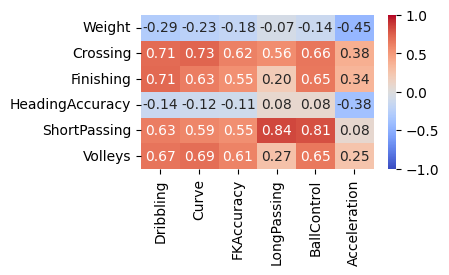


Fig 4.

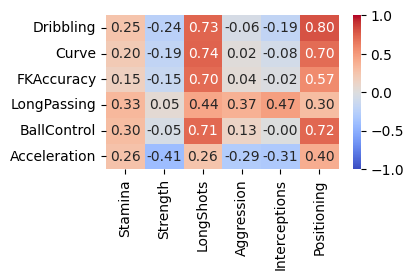


Fig 5.

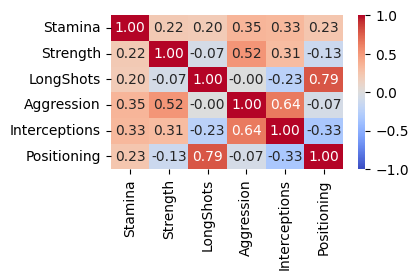


Fig 6.

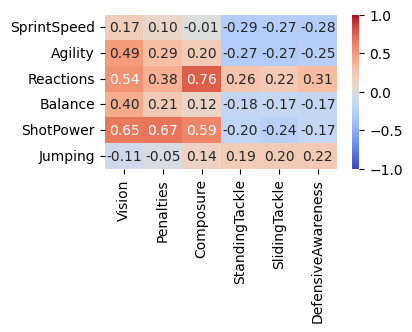


Fig 7.

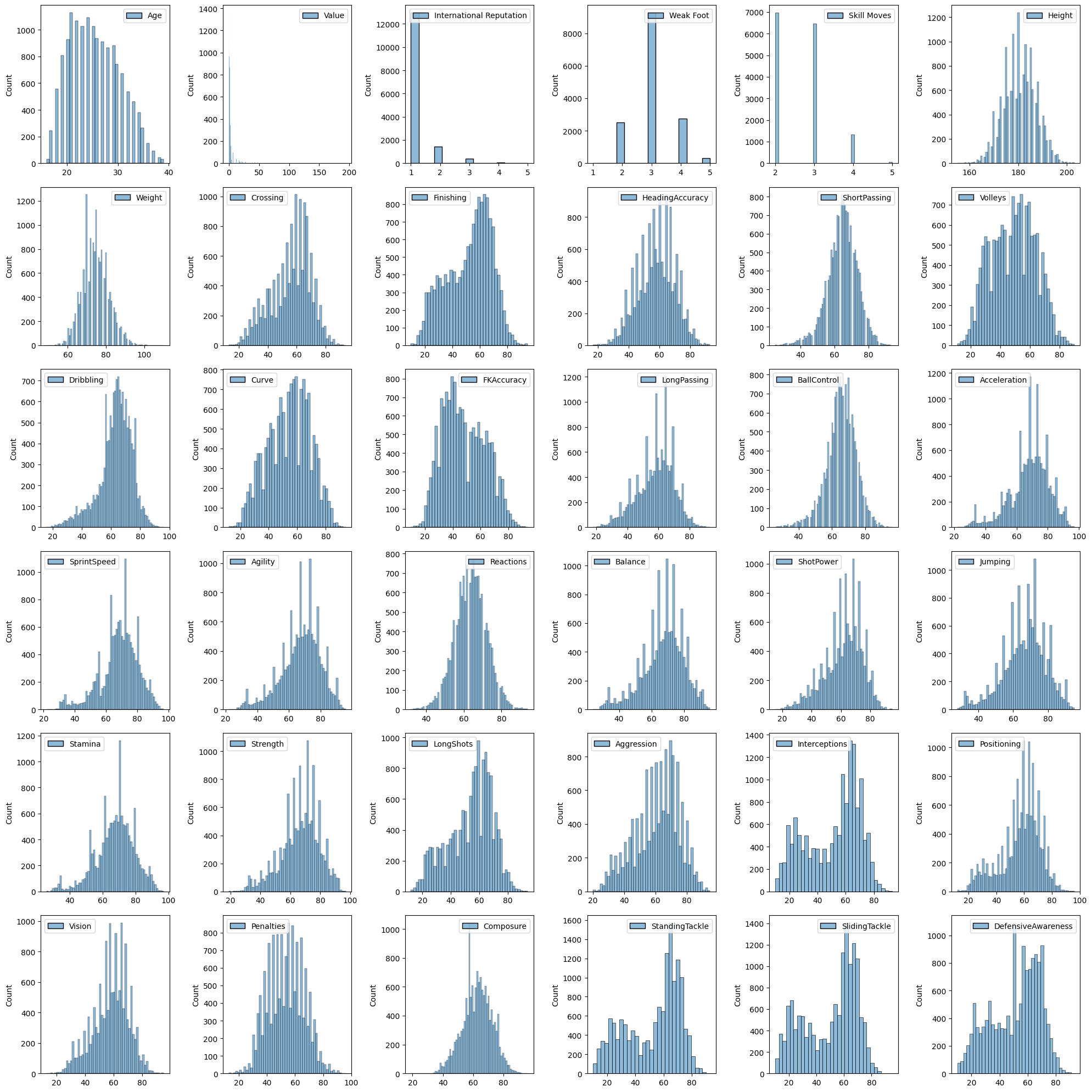


Fig 8.

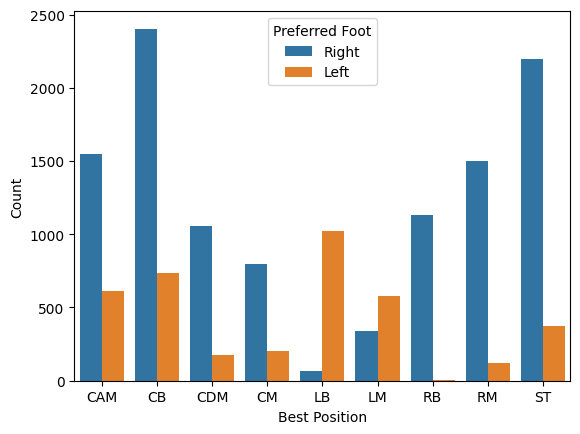


Fig 9.