JAIN (DEEMED-TO-BE UNIVERSITY)
School of Sciences

Sub: Data Cleaning
Activity: 1
Sub Code: 22MSDA3S04

Title:
Applying Data cleaning techniques on IPL discography dataset in R

Done By:
Aryanil Dey/ 22MSRDS065

Submission Date:
01/09/2023

Guide: M. A Ghouse Basha

Course: M.Sc. Data Science and Analytics

INDEX

Abstract:

This activity revolves around performing data cleaning techniques using R programming language, using libraries like tidyverse and dplyr and many more which will be described in the upcoming report in detail. The various techniques of data cleaning that were performed in this activity are

- Data collection
- Library loading
- Correcting column names
- Checking for consistent scale
- Handling missing values
- Outlier detection and handling
- Generating visualisation based on the previous processes

Now coming to the dataset, I used Indian players match dataset which had a length of 21 columns and 13993 rows, after having a glimpse to the dataset I encountered typo errors (incorrect spellings), outliers (in the numeric columns), missingness (in both numeric and categorical columns). All these discrepancies in the dataset were treated using the above-mentioned data cleaning techniques.

Introduction:

This report highlights the processes and techniques that are used to deal with data discrepancies and how to handle them. the essential steps of data analysis. Loading datasets is the first step, by using R's `read.csv () ` function. The `tidyverse` package is enlisted for streamlined data manipulation, ensuring data readiness.

Column names are the building blocks of data, and their accuracy is paramount. Using the `dplyr` function `rename()` to rectify any misspelt or incorrect column names, ensures data integrity. Consistency in data scale is a vital consideration. Our process examines columns with measurements, standardising units if necessary, using conditional statements in `dplyr`.
Handling missing values is an integral part of data analysis. Identifying columns with missing data and utilising `dplyr` to efficiently replace them with mean or median values, ensuring data completeness.

Outliers can skew results. visualising and addressing potential outliers using box plots and `dplyr`, either by removal or transformation.

Data storytelling unfolds through visualisation. I used plotting libraries like `ggplot2` to create compelling graphs. From scatter plots to bar charts and specialised visualisations like Parallel Coordinates Plots and Chord Diagrams, these tools enable us to convey data insights with precision.

Objective:

The aim of this activity is to practise data cleaning techniques using R. I focused on correcting column names, ensuring consistent measurement scales, handling missing values, and identifying/removing outliers. Additionally, I also used the tidyverse package and dplyr functions for efficient data manipulation. Each data cleaning approach was performed step by step in accordance with my dataset which is of “Indian players match dataset”.

Dataset Description:

The dataset under consideration comprises a comprehensive collection of cricket match-related information, featuring a wide array of attributes that provide insight into the dynamics of cricket matches and player performances. Here's a brief description of the key columns:

1. Player_match_SK: A unique identifier for player-match combinations.

2. PlayerMatch_key: A key linking player and match data.

3. Match_Id: Identifies the cricket match.

4. Player_Id: Uniquely identifies each player.

5. Player_Name: Provides the names of the players participating in the matches.

6. Batting_hand: Specifies the player's preferred hand for batting (left or right).

7. Bowling_skill: Indicates the player's bowling skills(e.g.,slow left arm, Chinaman, Right arm fast)

8. Country_Name: Denotes the country of origin for the players.

9. Role_Desc: Describes the role each player assumes in the game (e.g., batsman, bowler, all-rounder).

10. Player_team: Identifies the team to which the player belongs.

11. Opposit_Team: Names the opposing team in the match.

12. Season_year: Indicates the year in which the cricket season took place.

13. is_manofThematch: A binary flag (0 or 1) indicating whether the player received the "Man of the Match" award.

14. Age_As_on_match: Reflects the player's age at the time of the match.

15. IsPlayers_Team_won: A binary flag indicating whether the player's team emerged as the winner.

16. Batting_Status: Provides information about the player's batting status.

17. Bowling_Status: Specifies the player's bowling status.

18. Player_Captain: Indicates whether the player served as the captain of their team.

19. Opposit_captain: Specifies the opposing team's captain.

20. Player_keeper: Indicates whether the player served as the wicket-keeper for their team.

21. Opposit_keeper: Specifies the opposing team's wicket-keeper.

Methodology:

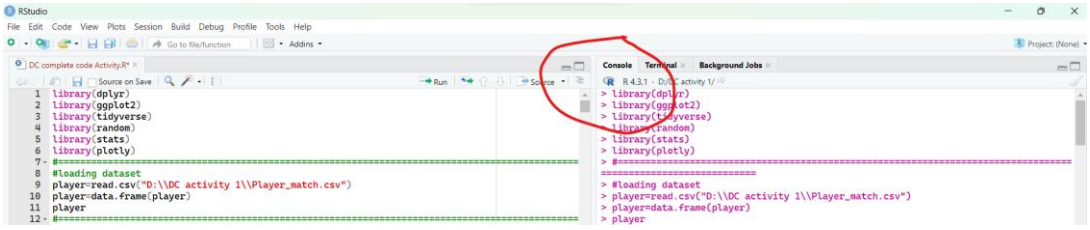
Earlier I have briefly mentioned data cleaning, its techniques and processes. Now I will describe each and every process on how I have approached this activity in real-time and give small snippets of R programming syntax.

Step 1: Loading Dataset and packages

In this step I upload a dataset into R studio (it is an integrated development environment (IDE) for R) to perform further processes I used the command **df=read.csv (),** this command will allow the user to load their desired dataset into R studio. In place of df any other name can be used because it is a variable name that is used to define a dataset by the name of your choosing to make your work more easier, and read.csv is the actual command that allows the R studio to fetch the dataset from your device in the () the file path is to be defined from where the dataset will be fetched be worry to use double slash in case you are using a different file directory other than C:\ for example like this.

```
player=read.csv("D:\\DC activity 1\\Player_match.csv")
```

Now to check if the dataset is loaded into the R studio environment you can either check the console at the bottom.



As you can see in previous picture the marked area is where you can check if the commands that you input are correct and are giving outputs
Or you can use the df command/ your datasets name to see if the dataset is loaded.

```
============================
> #loading dataset
> player=read.csv("D:\\DC activity 1\\Player_match.csv")
> player=data.frame(player)
> player
```

Like in the above picture you can refer and use the command i named my fetched dataset as player and put it under a dataframe(its optional). Lastly you can see I used the command **player.** Actually it is a variable name that I have used to define my dataset, any name can be used as a variable to define your dataset.

	Player_match_SK	PlayerMatch_key	Match_Id	Player_Id	Player_Name
1	12694	33598700006	335987	6	R Dravid
2	12695	33598700007	335987	7	W Jaffer
3	12696	33598700008	335987	8	V Kohli
4	12697	33598700009	335987	9	JH Kallis
5	12698	33598700010	335987	10	CL White
6	12699	33598700011	335987	11	MV Boucher
7	12700	33598700012	335987	12	B Akhil
8	12701	33598700013	335987	13	AA Noffke
9	12702	33598700014	335987	14	P Kumar
10	12703	33598700015	335987	15	Z Khan
11	12704	33598700016	335987	16	SB Joshi
12	12705	33598700001	335987	1	SC Ganguly
13	12706	33598700002	335987	2	BB McCullum
14	12707	33598700003	335987	3	RT Ponting
15	12708	33598700004	335987	4	DJ Hussey
16	12709	33598700005	335987	5	Mohammad Hafeez
17	12710	33598700062	335987	62	WP Saha
18	12711	33598700063	335987	63	LR Shukla
19	12712	33598700082	335987	82	AB Agarkar
20	12713	33598700083	335987	83	M Kartik
21	12714	33598700084	335987	84	I Sharma
22	12715	33598700106	335987	106	AB Dinda
23	12716	33598800024	335988	24	K Goel
24	12717	33598800025	335988	25	JR Hopes
25	12718	33598800026	335988	26	KC Sangakkara
26	12719	33598800027	335988	27	Yuvraj Singh
27	12720	33598800028	335988	28	SM Katich

As you can see in the above picture, after calling the dataset by the name I have assigned. It has printed an output of the few attributes of the dataset and its values.

Now before performing any other data cleaning operation, we first have to load certain libraries that will help to perform the data cleaning operations. These following libraries that i have used for the data cleaning operations

- **library(dplyr):** This command loads the "dplyr" package, which is a popular package in R for data manipulation and transformation. It provides functions like `filter ()`, `mutate()`, `select()`, and `group_by()` for efficiently working with data frames.

- **library(ggplot2):** This command loads the "ggplot2" package, which is a powerful library for creating data visualisations and plots. It follows a grammar of graphics approach, making it easy to create complex and customised plots.
- **library(tidyverse):** The "tidyverse" is a collection of R packages, including "dplyr" and "ggplot2," among others. Loading "tidyverse" makes it convenient to work with a cohesive set of packages designed for data manipulation and visualisation.
- **library(plotly):** The "plotly" package allows you to create interactive and web-based plots using R. It's especially useful for building interactive dashboards and visualisations that can be shared online.
- **library(circlize):** The `circlize` library in R is essential for creating circular visualisations like chord diagrams and circular heatmaps. It simplifies the representation of complex data relationships and patterns in a circular layout, making it ideal for tasks involving hierarchical or circular data visualisation.

These libraries and commands provide a robust set of tools for data analysis, visualisation, and statistical modelling in R. They enable you to work with data efficiently, create informative plots, perform statistical analyses, and even add interactivity to your visualisations when needed.

Step 2: Correcting Column Names, checking for typo errors, spelling error and repetition

(I) This is associated with the meaning of typo errors in the dataset or the spelling errors in my use case, the column names of the dataset had spelling errors and typo errors to fix them i used the following command

```
print(colnames(player))

#Correcting the column names
player = player %>%
  rename(Player_SK = Player_match_SK,
         Man_of_the_match = is_manofThematch,
         PlayingTeam_Keeper = Player_keeper,
         OpposingTeam_Keeper = Opposit_keeper,
         PlayingTeam_Captain = Player_Captain,
         OpposingTeam_Captain = Opposit_captain,
         Playing_Team = Player_team,
         Opposing_Team = Opposit_Team,
         Did_Playing_TeamWin = IsPlayers_Team_won)

# Verify the updated column names
print(colnames(player))
```

Examining Current Column Names:
I started by examining the existing column names in the "player" dataset using the `colnames()` function. This helped me identify which column names needed correction or renaming.

```
> ##(1)Correcting Column Names
> #I already applied dplyr previously
> # Examining the current column names
> print(colnames(player))
 [1] "Player_match_SK"    "PlayerMatch_key"    "Match_Id"
 [4] "Player_Id"          "Player_Name"        "Batting_hand"
 [7] "Bowling_skill"      "Country_Name"       "Role_Desc"
[10] "Player_team"        "Opposit_Team"       "Season_year"
[13] "is_manofThematch"   "Age_As_on_match"    "IsPlayers_Team_won"
[16] "Player_Captain"     "Opposit_captain"    "Player_keeper"
[19] "Opposit_keeper"
```

As we can see in the above picture there are few attributes that have typo error like opposit_keeper and IsPlayer_Team_won both of them are spelling error and typo error respectively

Correcting Column Names:
To correct and rename the columns, I used the "dplyr" package in R. I used the `%>%` pipe operator to pipe the "player" dataset into a series of "rename" operations.
For each column that needed correction, I used the `rename()` function. This allowed me to specify the new, more appropriate name for that column. The columns I corrected included "Player_match_SK," "is_manofThematch," "Player_keeper," "Opposit_keeper," "Player_Captain," "Opposit_captain," "Player_team," "Opposit_Team," and "IsPlayers_Team_won."

Verify Updated Column Names:
After completing the renaming operations, I checked the column names again using `colnames()`. This step was crucial to ensure that the changes I intended to make had indeed been applied successfully to the dataset.

The primary goal of this process was to enhance the clarity and usability of the dataset. By giving columns more descriptive and consistent names, it became easier to work with and understand during subsequent data analysis, manipulation.

<u>Output of the operation:</u>

```
> print(colnames(player))
 [1] "Player_SK"           "PlayerMatch_key"      "Match_Id"
 [4] "Player_Id"           "Player_Name"          "Batting_hand"
 [7] "Bowling_skill"       "Country_Name"         "Role_Desc"
[10] "Playing_Team"        "Opposing_Team"        "Season_year"
[13] "Man_of_the_match"    "Age_As_on_match"      "Did_Playing_TeamWin"
[16] "PlayingTeam_Captain" "OpposingTeam_Captain" "PlayingTeam_Keeper"
[19] "OpposingTeam_Keeper"
```

(II)Now to check if there is any spelling mistakes in observations of the PlayingTeam_Captain column

Counting Category Occurrences:
To understand the distribution of categories within the "PlayingTeam_Captain" column, I first used the `table () ` function. This allowed me to count the occurrences of each category within the column.

Identifying Unique Categories:
After obtaining the counts, my next step was to identify categories that occurred only once in the dataset. I created two variables, "unique_categories" and "unique_categories1," to store the names of categories that met different conditions. "unique_categories" stored the names of categories that occurred exactly once, while "unique_categories1" stored the names of categories that occurred more than once.

Printing the Unique Categories:
- Finally, I printed the unique categories that had one-time repetition in the "PlayingTeam

(II) I also performed the same procedure to check if there is any spelling mistakes in the observation of OpposingTeam_Captain

- We can refer to the above-mentioned explanation

(III)Replacing the spelling mistakes that were identified in the previous process

Identifying and Correcting Inconsistencies:
I noticed that there were inconsistencies in the names of some captains in both the "PlayingTeam_Captain" and "OpposingTeam_Captain" columns. These inconsistencies could lead to inaccuracies in the analysis. To address this issue, I used the `%in%` operator in combination with the `ifelse()` function to find and correct the inconsistent names.

Correcting "PlayingTeam_Captain" Names:
In the "PlayingTeam_Captain" column, I corrected various inconsistencies such as "DA Warnir" to "DA Warner," "GJ Mxwell" to "GJ Maxwell," "R Dravi" to "R Dravid," "RG Shama" to "RG Sharma," and "RG Sharmaaa" to "RG Sharma."
I also corrected "SC Gnguly" to "SC Ganguly" and "V Koli" to "V Kohli."

Correcting "OpposingTeam_Captain" Names:
Similarly, in the "OpposingTeam_Captain" column, I corrected inconsistencies like "G Gambhr" to "G Gambhir," "R David" to "R Dravid," "RG Sarma" to "RG Sharma," "SC Gangul" to "SC Ganguly," and "Z Khaan" to "Z Khan."

By standardising the names of captains in both columns, I ensured that the dataset's data quality was improved, making it more reliable for subsequent analysis and visualisations. This step was crucial to avoid discrepancies caused by variations in captain names.

We can run the below code to check if the logic i explained here worked

```
player_category1 = table(player$PlayingTeam_Captain)

# Filtering for categories that occur only once
unique_categories = names(player_category1[player_category1 == 1])
unique_categories1 = names(player_category1[player_category1 !=1])

# Printing the unique categories with one-time repetition
print(unique_categories)
print(unique_categories1)
```

As I have worked on two columns named “PlayingTeam_Captain” and “OpposingTeam_Captain” we just have to change the column name and variable name in the command so that we don’t get confused while working on the dataset.

(IV) Defining a vector of column names to include in displaying unique values: by checking the column

Defining Columns to Examine:
First, I created a vector called `columns_to_include`, which specifies the names of the columns I want to examine. In this case, I chose "Batting_hand" and "Bowling_skill" as the columns of interest.

Iterating Through Selected Columns:
Inside a `for` loop, I iterated through the elements of the `columns_to_include` vector. This allowed me to focus on one column at a time.

Checking Column Existence:
Within the loop, I used an `if` statement to check if the current column, denoted by `col`, exists in the dataset. This step ensures that I'm working with valid column names.

Identifying Unique Values:
If the column exists in the dataset, I proceeded to extract its unique values using the `unique()` function. These unique values represent all the distinct entries found in that particular column.

Displaying Results:
For each column, I printed a message indicating which column I was examining, such as "Unique values in column 'Batting_hand':", to provide context.
Next, I printed the actual unique values of the column using the `print()` function.
I included a line break (`cat("\n")`) to separate the results of different columns.

This process allowed me to systematically go through the specified columns, identify their unique values, and display them. It's a useful approach for understanding the diversity and distribution of values within each selected column, which can be valuable for data exploration and analysis.

```
Unique values in column ' Batting_hand ':
[1] "Right-hand bat"     "Left-hand bat"      "\xa0Right-hand bat"
[4] "Right-handed"       "\xa0Left-hand bat"

Unique values in column ' Bowling_skill ':
 [1] "Right-arm offbreak"        "Right-arm medium"
 [3] "Right-arm fast-medium"     "Legbreak googly"
 [5] "Right-arm medium-fast"     "Left-arm fast-medium"
 [7] "Slow left-arm orthodox"    "Right-arm fast"
 [9] "Slow left-arm chinaman"    "Left-arm medium-fast"
[11] "Legbreak"                  "Left-arm fast"
[13] "Left-arm medium"           "Right-arm bowler"
[15] "\xa0Left-arm fast"         "\xa0Right-arm fast-medium"
[17] "Right-arm medium fast"     "\xa0Right-arm medium-fast"
[19] "\xa0Right-arm offbreak"    "\xa0Legbreak"
```

As you can see that in the output it has printed the unique attributes in each of the columns, those unique categorical values will be considered as typo errors and we have to fix them

(V) Replacing the typo errors, that we have encountered in the earlier process

Identifying Typo Errors:
First, I carefully reviewed the "Batting_hand" and "Bowling_skill" columns in the dataset, aiming to identify any typo errors or inconsistencies in the values. These errors could be due to variations in formatting, extra spaces, or spelling mistakes.

Replacing Typo Errors - Batting Hand:
For the "Batting_hand" column, I used the `%in%` operator along with the `ifelse()` function to check if each value matches any of the specified typo errors. These errors included values like "\xa0Right-hand bat" and "Right-handed," which should be corrected to "Right-hand bat." If a value match any of the typo errors, it will replace it with the correct value, "Right-hand bat." I performed this correction using the `mutate()` function from the `dplyr` package.

Replacing Typo Errors - Bowling Skill:
Similarly, I applied the same process to the "Bowling_skill" column. I checked for typo errors such as "\xa0Left-arm fast" and corrected them to "Left-arm fast."
I also corrected other variations like "\xa0Right-arm fast-medium" to "Right-arm fast-medium," "\xa0Right-arm medium-fast" to "Right-arm medium-fast," "\xa0Right-arm offbreak" to "Right-arm offbreak," and "\xa0Legbreak" to "Legbreak."

Ensuring Data Consistency:
By making these corrections, I ensured that the values in the "Batting_hand" and "Bowling_skill" columns were consistent and followed a standardized format. This step is

crucial for data analysis and visualisation because it eliminates inconsistencies that could lead to errors or misinterpretations.

Overall, this process involved a careful review of the dataset, identification of typo errors, and systematic replacement of these errors with the correct values. It contributes to data quality and reliability, which are essential for meaningful analysis and reporting.

Now we have to repeat the process number (IV), to check if the typo errors are replaced

After removing the typo, the define function of the typo error will show a output like this

```
Unique values in column ' Batting_hand ':
[1] "Right-hand bat" "Left-hand bat"

Unique values in column ' Bowling_skill ':
 [1] "Right-arm offbreak"     "Right-arm medium"
 [3] "Right-arm fast-medium"  "Legbreak googly"
 [5] "Right-arm medium-fast"  "Left-arm fast-medium"
 [7] "Slow left-arm orthodox" "Right-arm fast"
 [9] "Slow left-arm chinaman" "Left-arm medium-fast"
[11] "Legbreak"               "Left-arm fast"
[13] "Left-arm medium"        "Right-arm bowler"
[15] "Right-arm medium fast"
```

As we can see there are no typo errors visible after we have replaced them. You can compare it with the previous output that i have referred in the **process number (IV)**

Step 3: checking Consistency in the dataset

Checking Role_Desc Distribution:
I began by using the `table()` function to examine the distribution of values in the "Role_Desc" column of the "player" dataset. This helped me understand the different roles represented in the data.

Filtering Rows Based on Age:
To ensure the data's consistency and relevance, I needed to filter out rows where the "Age" variable fell outside the range of 0 to 50 years, inclusive.
I created a new dataset called "player_age" by applying a logical condition to the "Age" column. This condition selected rows where the age was between 0 and 50, as specified.

Selecting Rows Outside the Age Range:
Since "player_age" contained rows within the desired age range, I wanted to identify and examine rows that were not in this subset.
I used the "!" operator to negate the logical condition, effectively selecting rows where "Age" was outside the 0 to 50 range.
Next, I specified the columns I wanted to include in the result, which were "Player_Name," "Age_As_on_match," and "Country_Name."

Printing the Selected Columns:
Finally, I printed the selected columns for the rows that did not meet the age criteria. This allowed me to inspect specific information about players whose ages were beyond the specified range.

The objective of this process was to assess and manage data consistency, ensuring that only relevant records within the desired age range were retained for further analysis, while also providing insights into the characteristics of players who did not meet this criterion.

Step 4: Handling Missing values

Defining a function to replace values based on conditions

- **Creating the `replace_values` Function:**
  First, I defined a custom R function called `replace_values(data)` to handle the replacement of specific values within the dataset.
- **Iterating Through Rows:**
  Inside the function, I used a `for` loop to iterate through each row of the dataset. This loop allows me to examine and potentially modify each row individually.
- **Extracting Relevant Column Values:**
  In each iteration, I extracted three key pieces of information from the current row: `role_desc`, `bowling_skill`, and `batting_hand`. These values are crucial for making decisions about replacements.

- **Conditional Replacement Logic:**
  I implemented a set of conditional statements to determine whether replacements should be made based on the values of `role_desc`, `bowling_skill`, and `batting_hand`. These conditions ensure that replacements are made selectively, following specific criteria:
  For players with the role of "Keeper" or "CaptainKeeper" and missing or unspecified bowling skills, I randomly assigned one of two bowling skills: "Right-arm medium" or "Left-arm medium."
  For players with the role of "Player" and missing or unspecified bowling skills, I considered their batting hand. If they bat with their left hand, I assign a random left-

arm bowling skill ("Left-arm fast" or "Left-arm medium"). For right-hand batters, I assigned a random right-arm bowling skill ("Right-arm fast" or "Right-arm medium").

- **Updating the Dataset:**
  Inside the loop, I updated the `Bowling_skill` column with the appropriate replacement value based on the conditional logic.

- **Returning the Updated Dataset:**
  Finally, I returned the modified dataset with the replaced values.

- **Applying the Function:**
  To apply these value replacements to the entire dataset, I called the `replace_values` function on the `player` dataset and stored the result back in the `player` variable.

- **Viewing the Updated Dataset:**
  I printed the updated dataset to view the changes made by the function, ensuring that the values were replaced according to the specified conditions.

By following this process, I was able to systematically update and improve the dataset, making it more consistent and suitable for further analysis or visualisation.

Step 5: Outlier Detection and Handling

(I) Outlier Detection

- **Defining the Outlier Detection Function:**
  First, I defined a custom R function named `detect_outliers`. This function takes two arguments: the dataset (`data`) and the name of the column where outliers need to be detected (`column_name`).

- **Checking for Missing Values:**
  Within the function, I included a check to see if the specified column contains any missing values. If missing values are present, the function prints a message indicating that outlier detection is skipped for that column and returns `NULL`.

- **Calculating Quartiles and IQR:**
  For outlier detection, I calculated the first quartile (Q1) and the third quartile (Q3) of the values in the specified column. These quartiles are essential for determining the interquartile range (IQR), which helps in identifying outliers.

- **Setting Lower and Upper Bounds:**
  Using the quartile values, I calculated the lower and upper bounds for potential outliers. These bounds are calculated as follows:

Lower Bound: Q1 - 1.5 * IQR
Upper Bound: Q3 + 1.5 * IQR
Any data points falling below the lower bound or above the upper bound are considered potential outliers.

- **Identifying and Filtering Outliers:**
  I used the lower and upper bounds to filter the dataset and identify potential outliers in the specified column. The `dplyr` function `filter()` was employed for this purpose.

- **Returning Outliers:**
  Finally, the function returns a data frame containing the identified outliers, if any.

This custom function streamlines the process of detecting outliers in numerical columns and can be applied to various columns within the dataset. It helps in identifying and isolating data points that may require further investigation due to their deviation from the typical range of values.

(II) Outlier Handling

- **Replacing Outliers with Nearest Random Age:**
  I wanted to handle outliers in the 'Age_As_on_match' variable within the dataset. Outliers are data points that significantly deviate from the typical range and can distort statistical analyses. To address this, I created a custom R function named `replace_outliers_with_random()`.

- **Checking for Missing Values:**
  Before proceeding, I checked if the specified column ('Age_As_on_match') contained any missing values (NAs). Handling missing values is important to ensure that the replacement process is accurate and complete.

- **Calculating Quartiles and IQR:**
  I calculated the first quartile (Q1) and third quartile (Q3) for the 'Age_As_on_match' variable. These quartiles help in defining the interquartile range (IQR), a measure of data spread.

- **Identifying Outliers:**
  Outliers were identified using the lower and upper bounds based on the IQR. Data points falling below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR were considered outliers.

- **Replacing Outliers with Random Values:**
  To address the outliers, I replaced them with random age values selected from the same dataset. This approach helps maintain the overall statistical characteristics of the data while removing extreme values.

- **Iterative Process:**
  The replacement process was performed iteratively for each outlier detected. For each outlier, I randomly selected an age value from the dataset and replaced the outlier with this new value. This ensured that replaced values were consistent with the dataset's age distribution.

- **Updating the Dataset:**
  After replacing the outliers, I updated the 'Age_As_on_match' column in the dataset with the new values.

- **Verifying the Result:**
  Finally, I printed the updated dataset to verify that the outliers in the 'Age_As_on_match' column had been successfully replaced with random age values.

This process helped me manage outliers in the 'Age_As_on_match' variable, ensuring that the dataset remained suitable for subsequent analysis without the influence of extreme values.

Plotting the outliers in the dataset

- AFTER DETECTING THE OUTLIERS, WE ARE TO VISUALISE IF THERE IS ANY OUTLIERS IN THE DATASET USING BOXPLOT

- IF IT SHOWS OUTLIER THEN WE ARE TO REMOVE AND REPLACE THE OUTLIERS

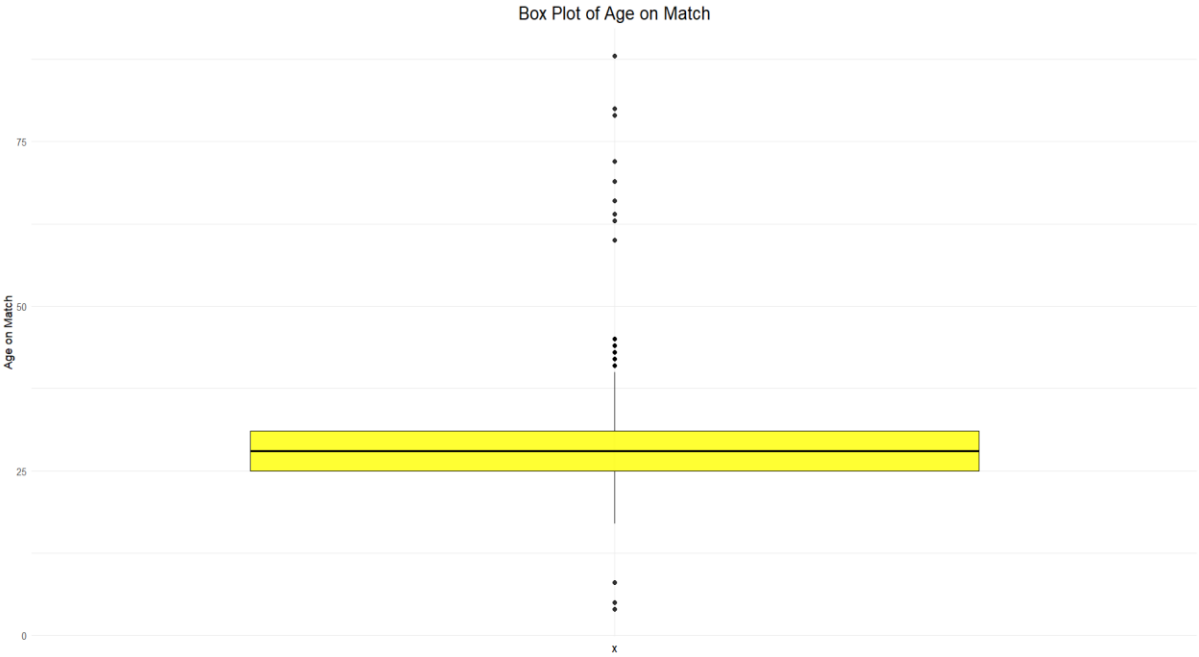
- THEN AGAIN RUN THE BOXPLOT TO VISUALISE IF THERE IS ANY OUTLIERS IN THE DATASET

EEEEEEEEEEEEEEE DDDDDDDDDD A A
EE D DD A A
EE D DD A A
EE D DD A A
EE D DD A A
EE D DD A A
EEEEEEEEEEEEEEE D DD AAAAAAAAA
EE D DD A A
EE D DD A A
EE D DD A A
EE D DD A A
EEEEEEEEEEEEEEE DDDDDDDDDD A A

Numerical Data Visualization

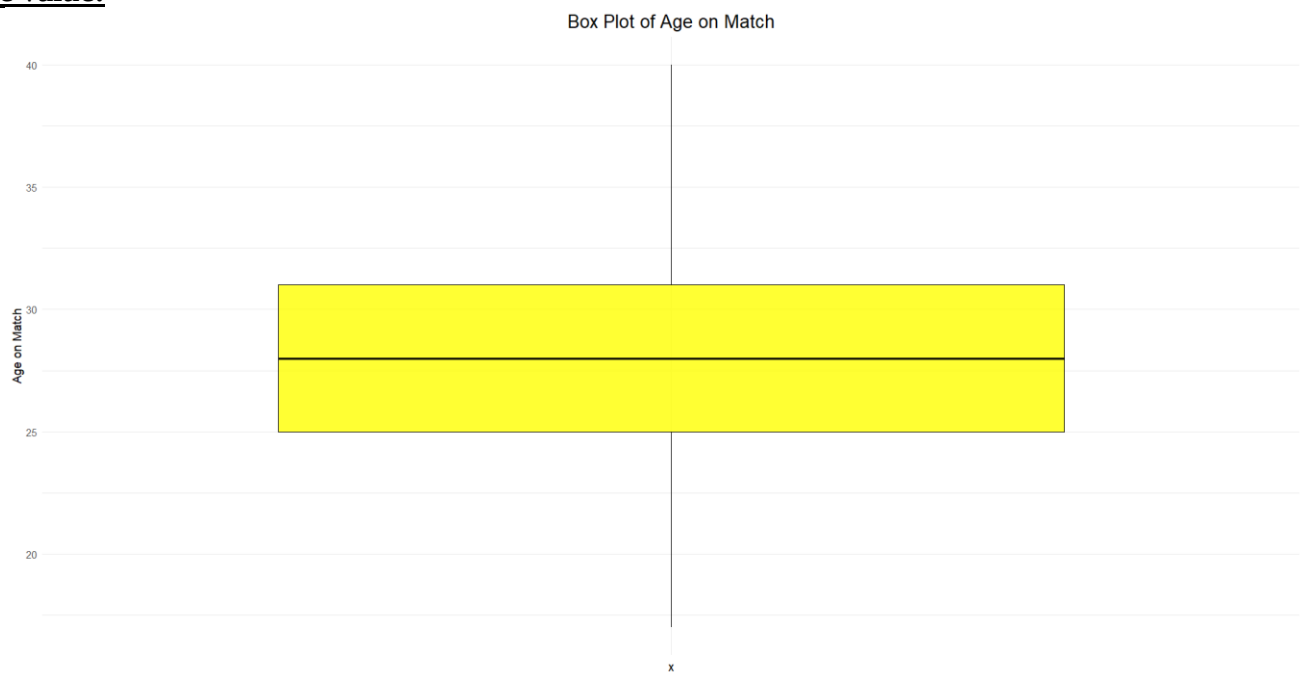
(I) Box plot (Age as on match for identifying outlier) [Numerical Data Visualization]

This is the Box plot Output before replacing them with nearest random age value:



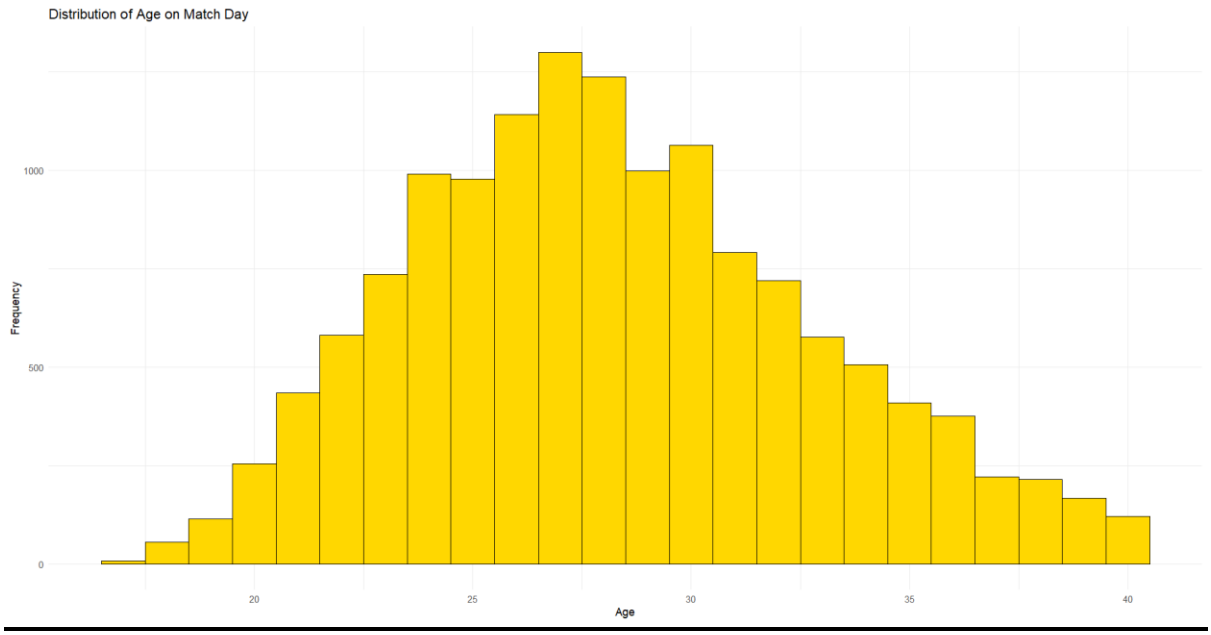
Insights: The original box plot displays the full range of the 'Age_As_on_match' variable. Outliers are visible as data points outside the whiskers, indicating data points significantly deviating from the central trend. After outlier detection the dataset had 128 observations of outliers. It provides an initial view of data spread and presence of extreme values.

Now this is the Box plot output after i have replaced the outliers with the nearest random age value:



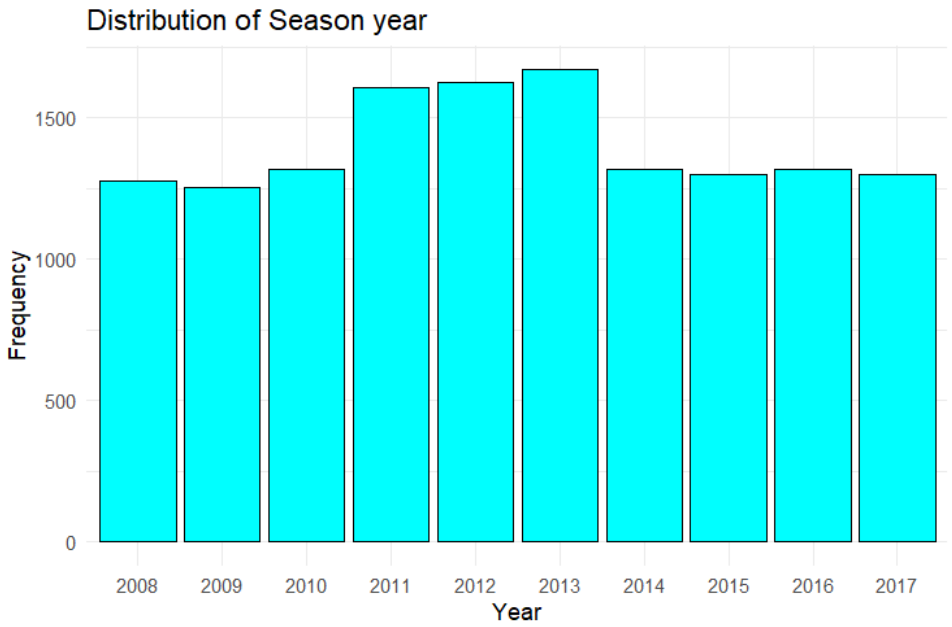
Insights: The updated box plot shows a reduced data spread, with narrower whiskers. 128 observations Outliers have been removed, resulting in a cleaner representation of central tendency. It reflects improved data integrity and reliability for subsequent analyses.

(II) Histogram (Distribution of Age on Match Day) [Numerical Data Visualization]



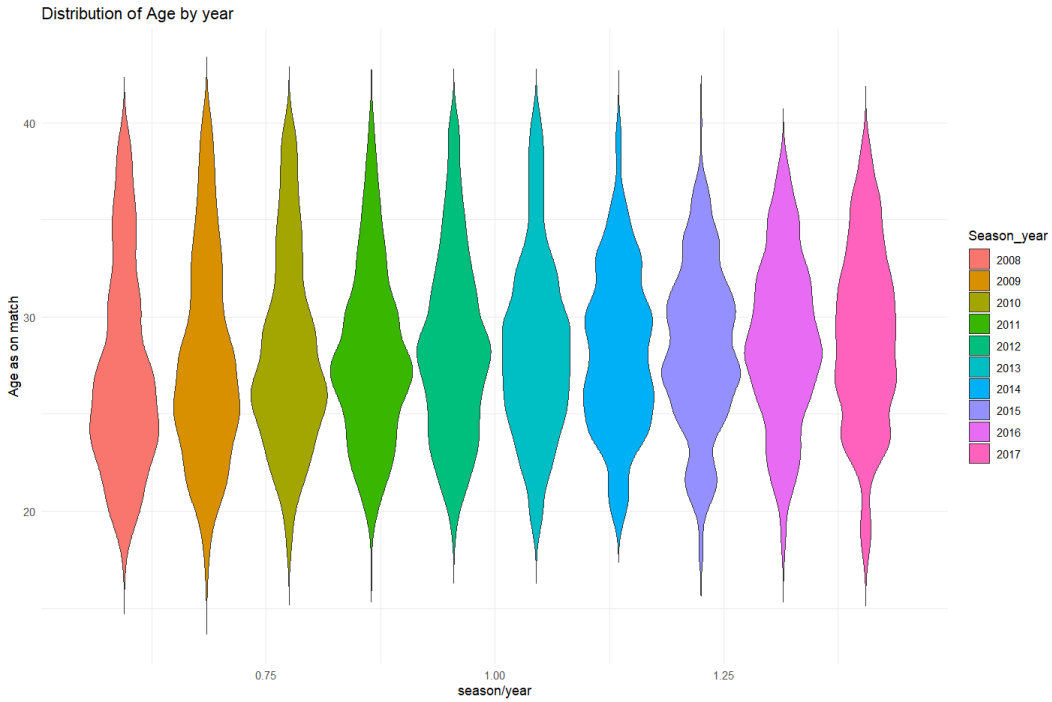
Insights: The histogram illustrates the age distribution of players on match days. It reveals a prominent peak in the frequency, indicating a dominant age group. The distribution is slightly right-skewed, indicating fewer senior players.

(III) histogram (Distribution of Season year) [Numerical Data Visualization]



Insights: This histogram illustrates the frequency of each season year within the dataset. By observing the heights of the bars, we can identify any trends or patterns in the distribution of season years. This information can be valuable for understanding how data is distributed across different years.

(IV) Violin plot Distribution of Age by year [Numerical Data Visualization]

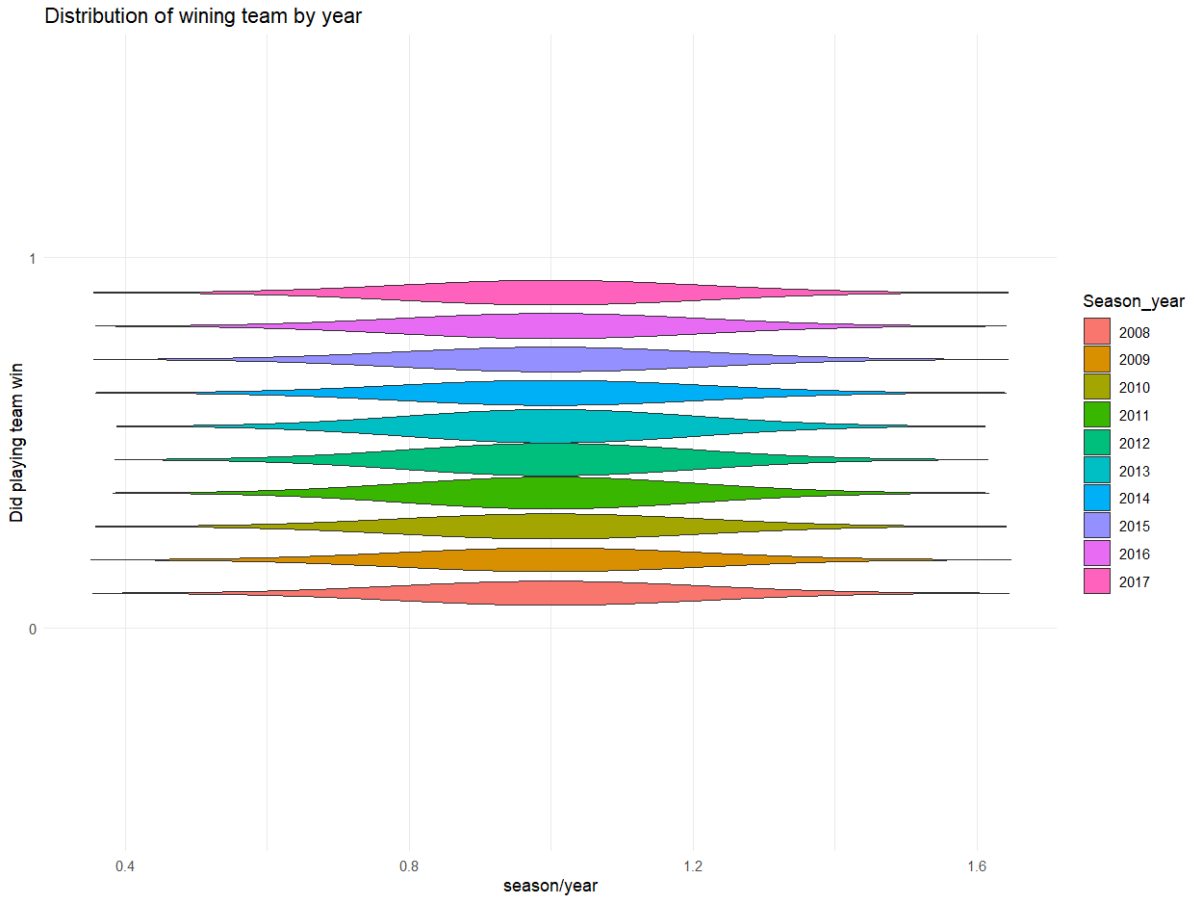


Insights: The width of the violin at various points indicates where most player ages cluster. Wider sections represent higher density. Variations in the violin's shape highlight changes in age

distributions across seasons. Look for shifts in the shape for insights into evolving player demographics. Observe the differences in the peak and spread of each violin. They can reveal unique characteristics of player age profiles for each year.

In summary, this plot allows us to compare and contrast player age distributions over multiple seasons, offering insights into potential trends and variations.

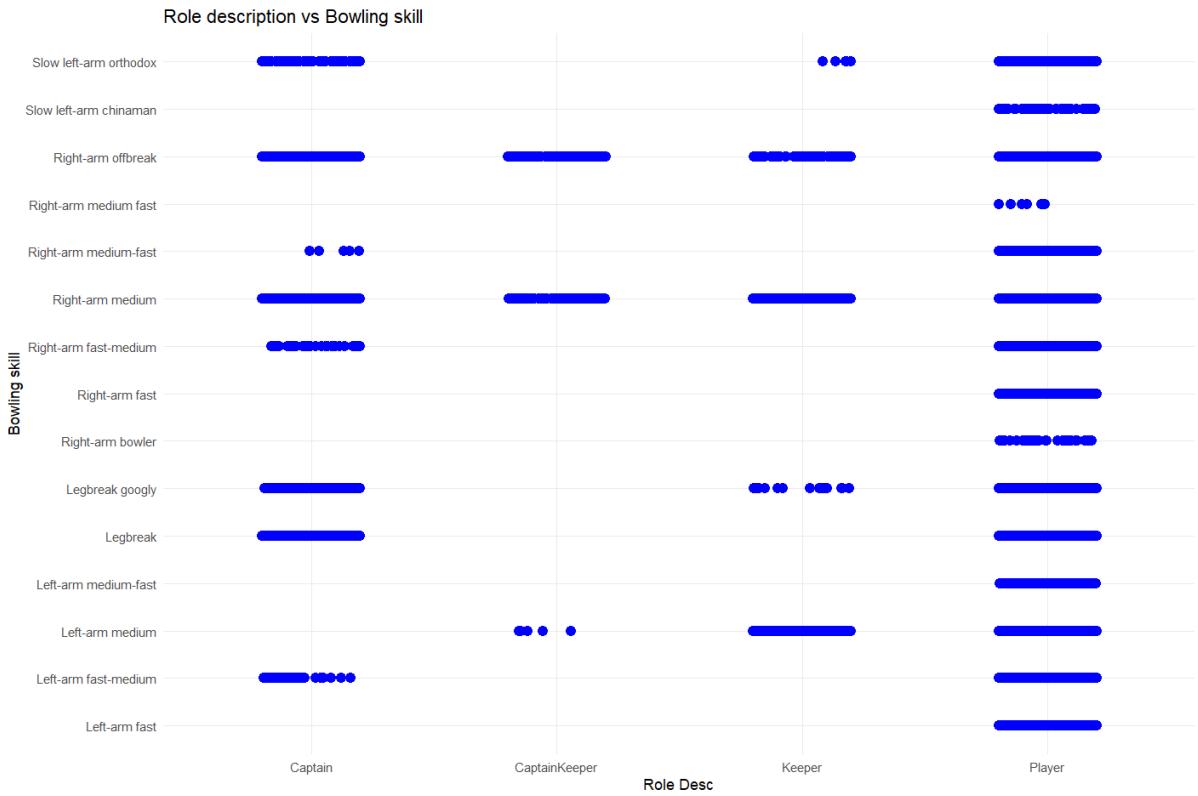
(V) Violin plot Distribution of winning team by year [Numerical Data Visualization]



Insights: The violin plot displaying the distribution of winning teams by year provides insights into the performance trends over seasons. It shows that there is variability in the number of wins each year, with some years having more evenly distributed wins across teams, while others have clear dominant teams. This visualisation highlights the competitive nature of cricket tournaments, where team dynamics and strengths can vary significantly from year to year.

Categorical Data Visualisations

(A) Scatter plot (Role description vs bowling skill) [Categorical Data visualisation]



Insights: The scatter plot "Role Description vs. Bowling Skill" provides insights into the distribution of players across various roles and their corresponding bowling skills. It helps us understand how different roles are distributed among players based on their bowling abilities. This visual representation allows for a quick assessment of the diversity of skills within each role and identifies any patterns or trends.

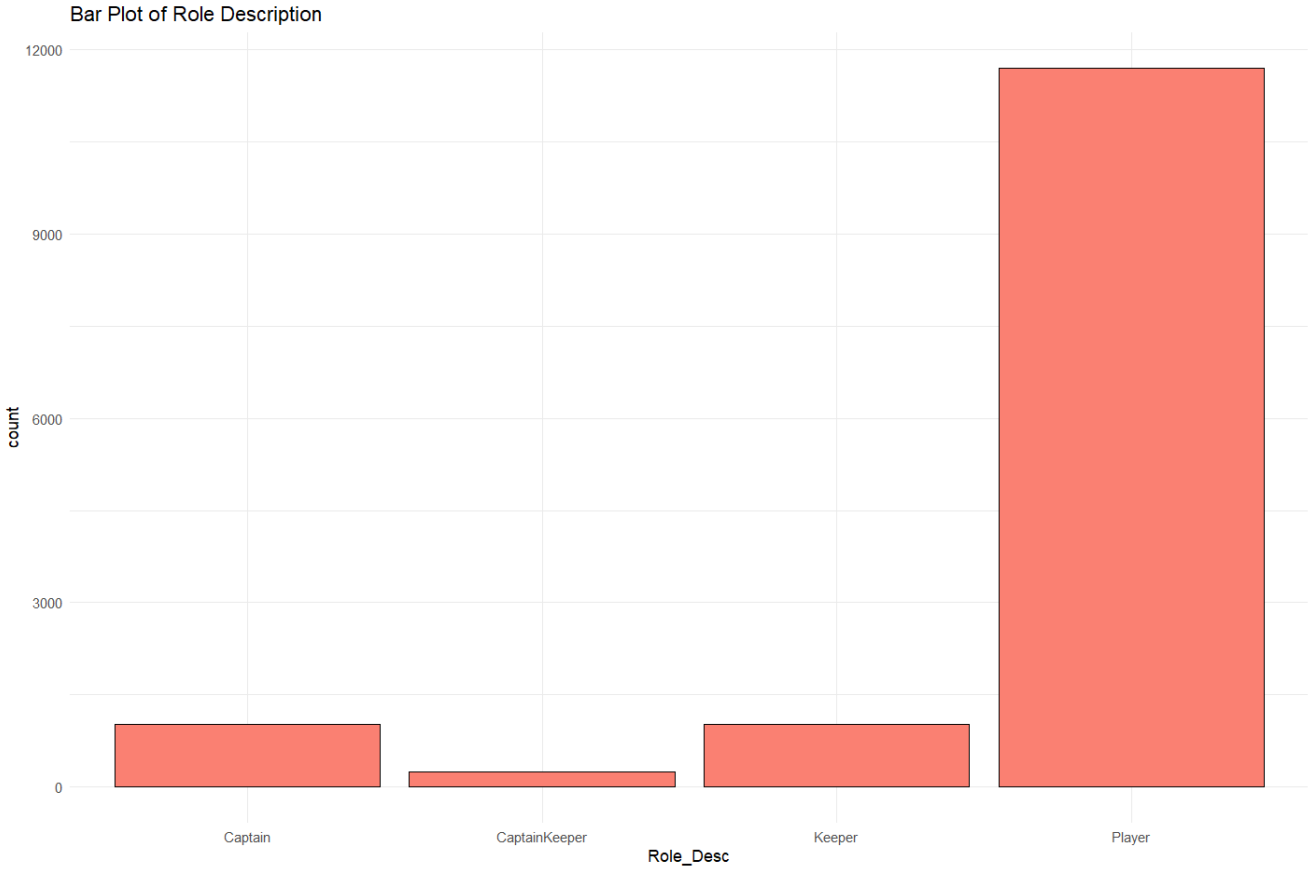
(B) Scatter plot (Role description vs playing team) [Categorical Data Visualization]

Insights: The scatter plot "Role Description vs. Opposing Team" offers insights into the distribution of player roles across different opposing teams. This visualisation helps us observe how roles are allocated when facing various opponent teams. It provides a snapshot of the role composition within each opponent team, allowing for a quick assessment of any role-related patterns or trends.

(C) *****Scatter plot (player Age vs Role Description)[Numerical vs Categorical Data visualisation]

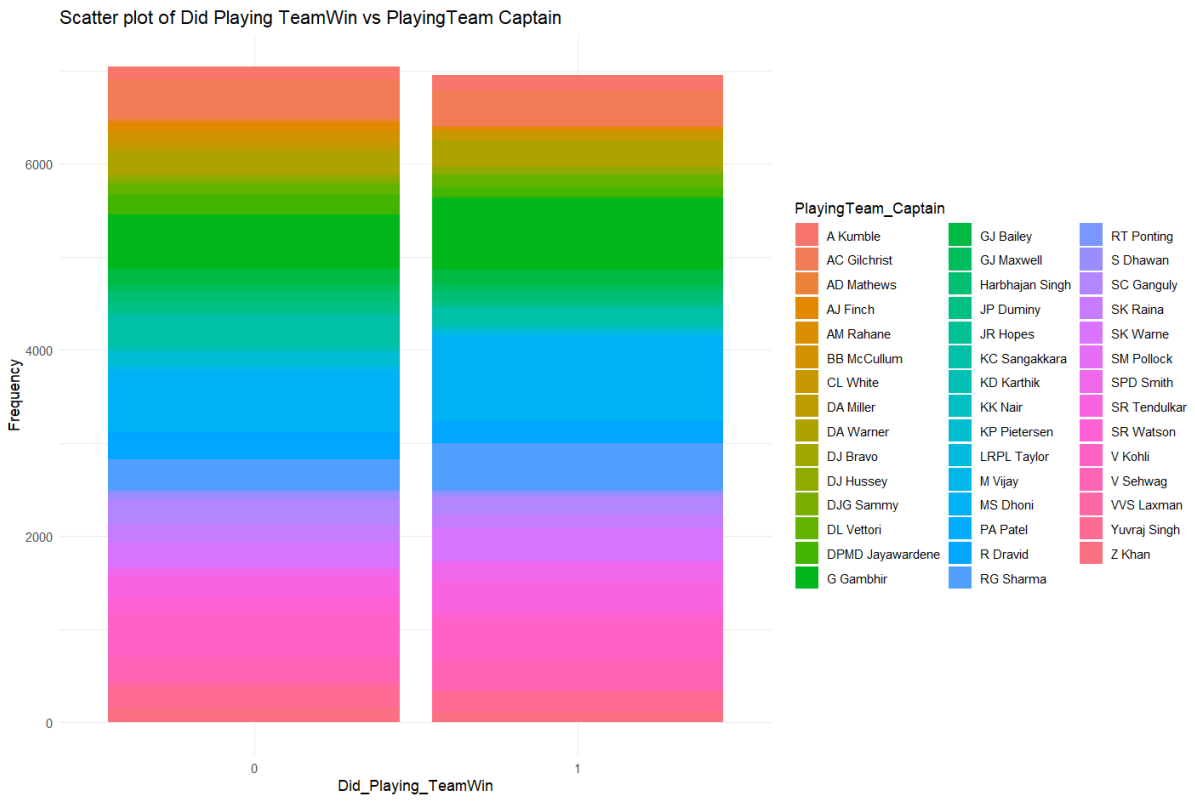
Insights: The scatter plot (player Age vs Role Description) helps visualise the distribution of player ages within different role descriptions. From the graph, we can observe that the majority of players fall into the "Player" role description, which is expected as it encompasses various positions. Additionally, the plot shows that there are players of diverse ages across different roles, indicating a broad age range among cricket players. This visualisation provides a quick overview of how age is distributed among different player roles, which can be valuable for analysing team compositions and player demographics.

(D) Bar plot of Role description [Categorical Data Visualization]



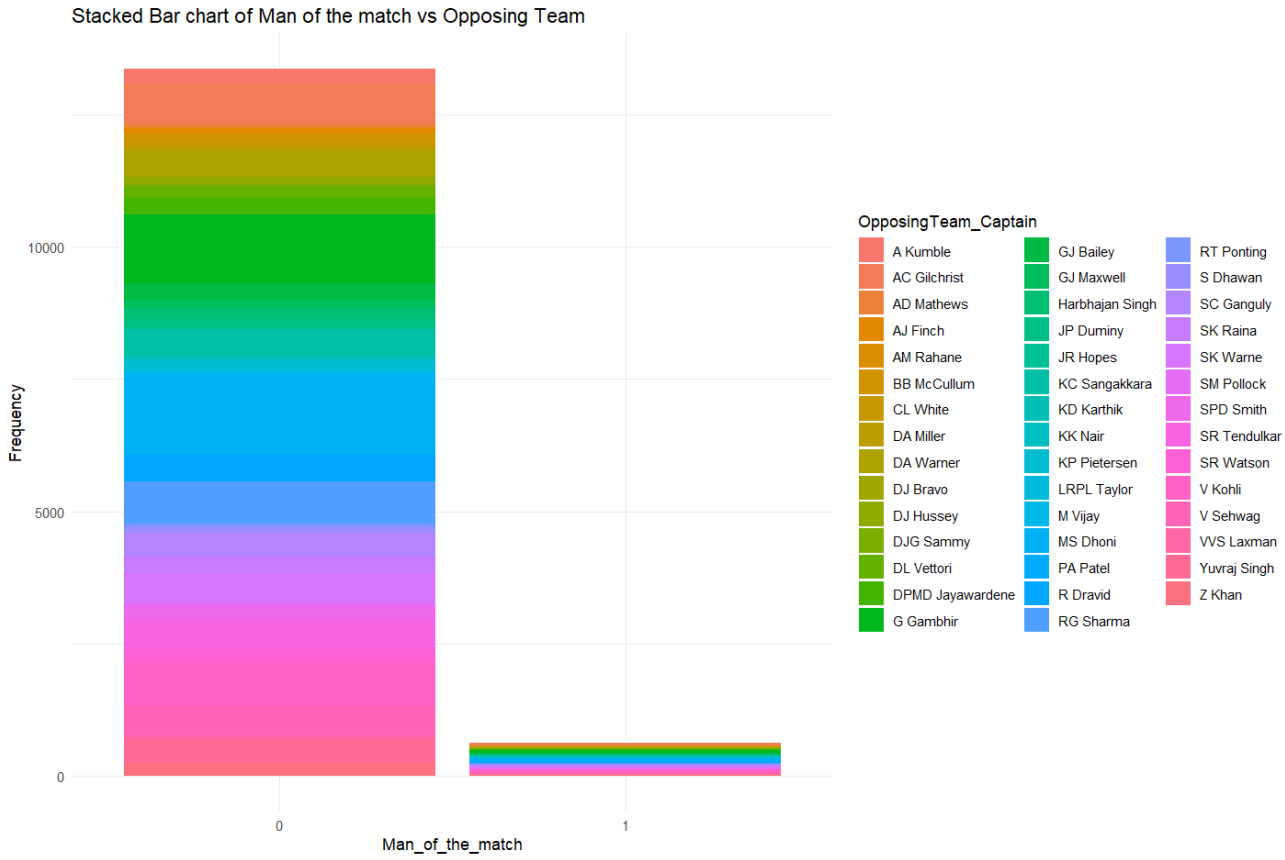
Insights: The bar plot of "Role Description" provides a clear overview of the distribution of player roles within the dataset. It allows us to see the frequency of each role category, highlighting the most common roles. This visualisation can be useful for understanding the role composition within the dataset and identifying any dominant roles or outliers.

(E) ****Stacked Bar chart Did Playing TeamWin vs PlayingTeam Captain [Categorical vs Numerical Data Visualization]



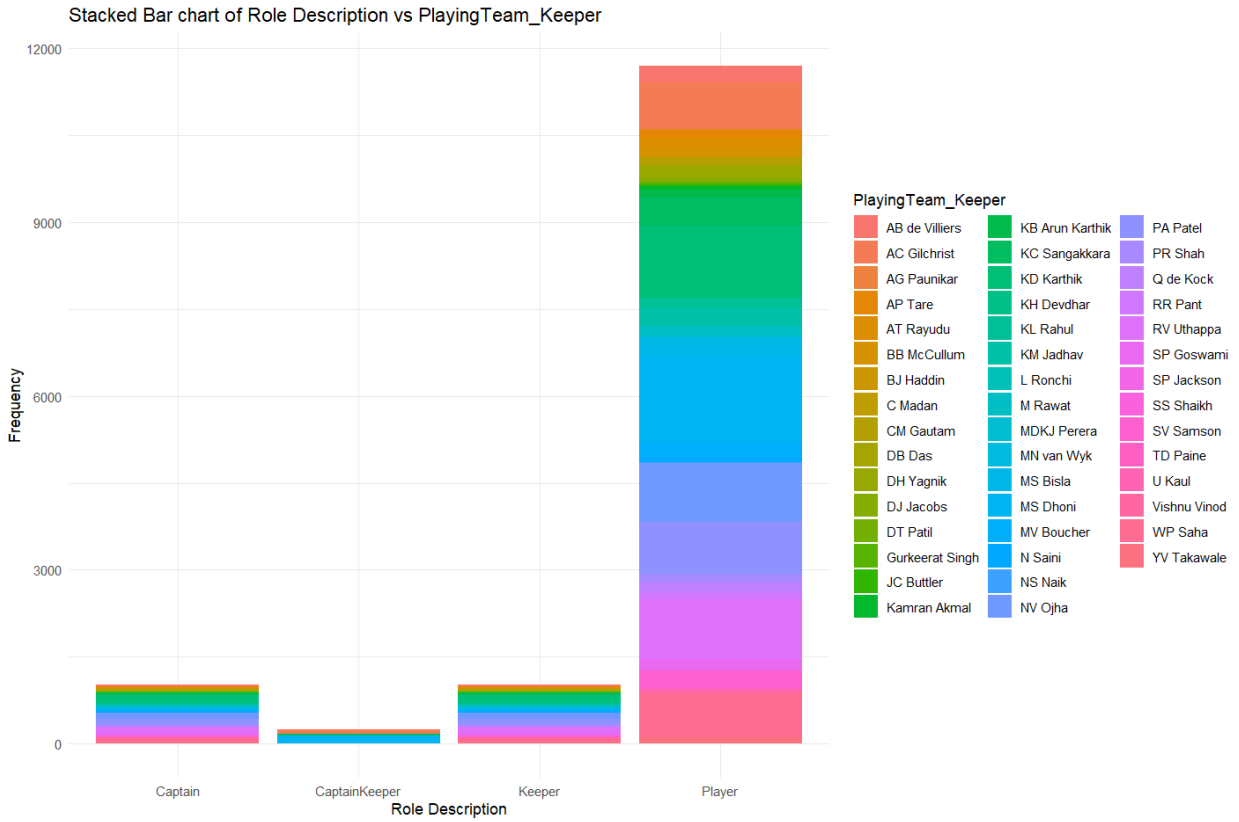
Insights: The stacked bar chart depicting "Did Playing Team Win" vs. "Playing Team Captain" provides a visual representation of how team captains' performance relates to their team's success. It allows us to compare the number of wins and losses for each captain, showing which captains have a higher success rate. This visualisation is valuable for assessing the impact of different team captains on their team's victories, offering insights into their leadership abilities and performance.

(F) Stacked Bar chart of Man of the match vs Opposing Team [Categorical Data Visualization]



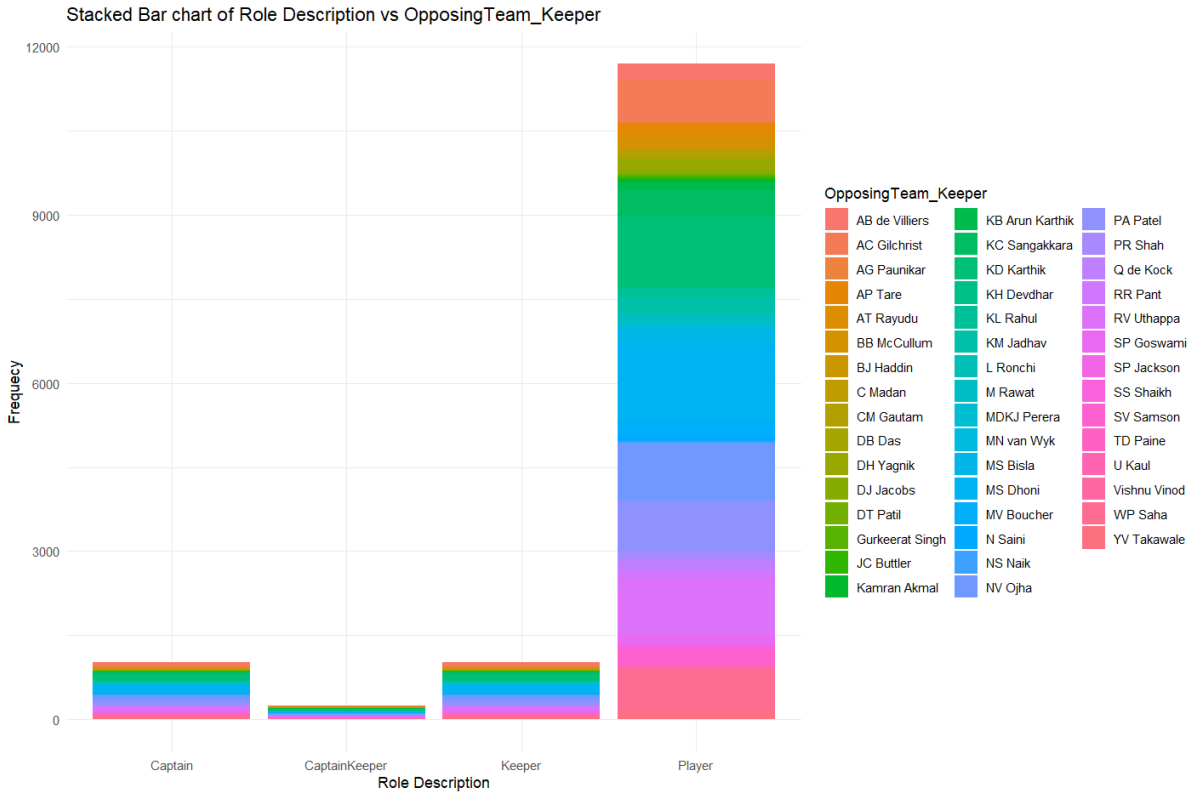
Insights: The stacked bar chart illustrating "Man of the Match" vs. "Opposing Team" provides an overview of which opposing teams are more likely to produce a player awarded as the Man of the Match. It highlights the distribution of these prestigious awards across different opposing teams, allowing us to identify patterns or standout performances against specific teams. This visualization can help us recognize rivalries or matchups where exceptional individual performances often occur, shedding light on noteworthy player achievements in certain game scenarios.

(G) Stacked Bar chart of Role Description vs PlayingTeam_Keeper [Categorical Data Visualization]



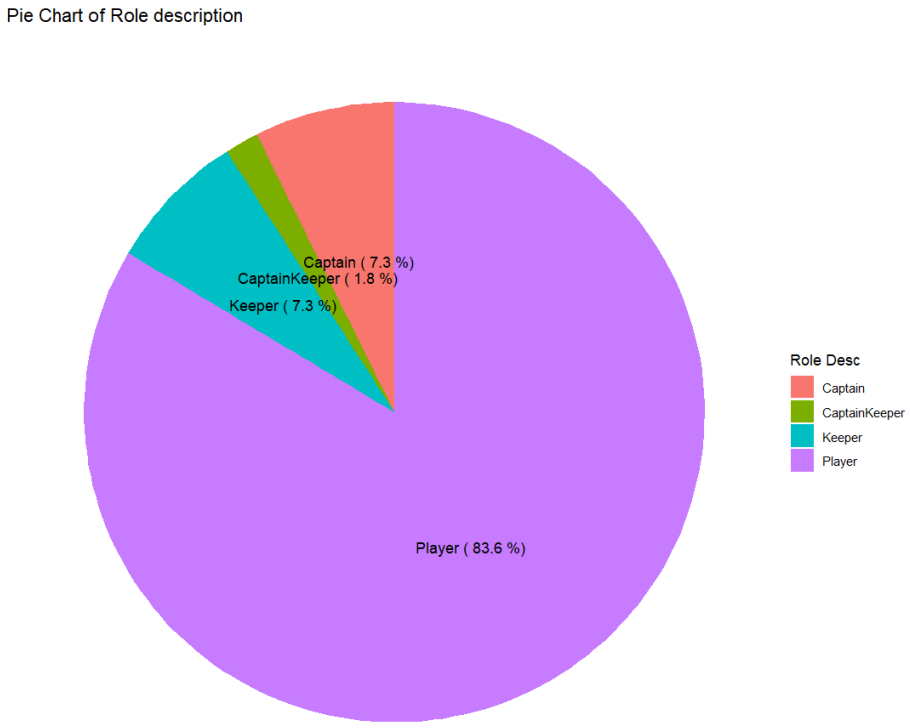
Insight: The stacked bar chart comparing "Role Description" to "Playing Team Keeper" provides valuable insights into the distribution of roles among players based on their teams. It visualises the relationship between player roles, such as Keeper or Captain, and the team for which they play. This graph helps us understand how various roles are distributed across different teams, shedding light on team dynamics and strategies. Additionally, it enables us to identify teams that prioritise specific roles and the impact of these roles on team performance.

(H) Stacked Bar chart of Role Description vs OpposingTeam_Keeper [Categorical Data Visualization]



Insights: The stacked bar chart comparing "Role Description" to "Opposing Team Keeper" offers insights into the distribution of player roles when facing different opposing teams. It illustrates how player roles, such as Keepers or Captains, are distributed when competing against various teams. This visualisation can help identify patterns in team strategies, including whether certain roles are emphasised when playing against specific opponents. Analysing these patterns can provide valuable insights into team dynamics and decision-making during matches, contributing to a deeper understanding of the game's strategies and outcomes.

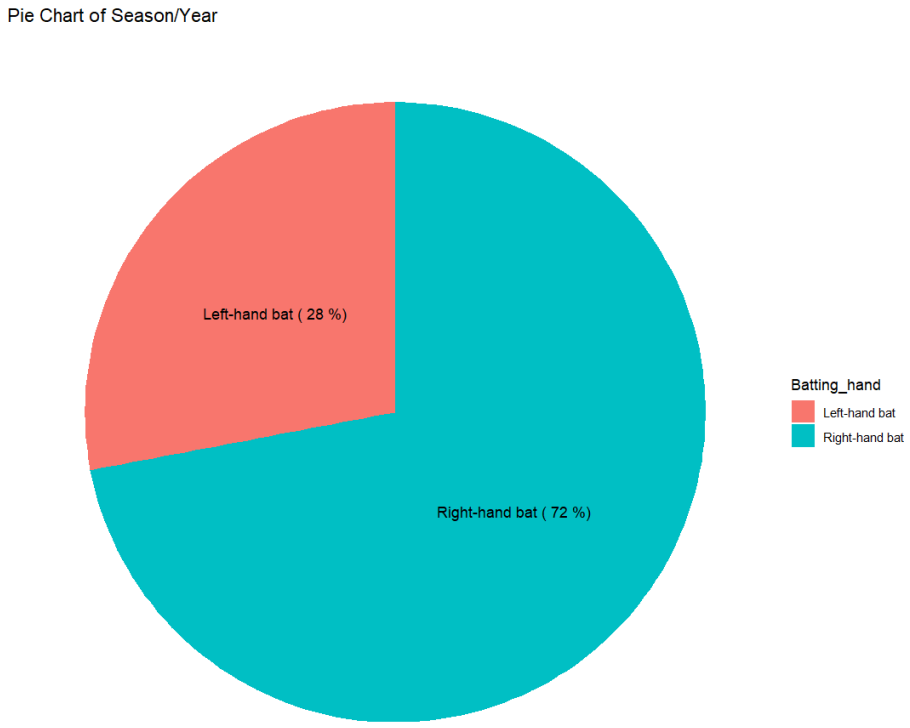
(I) Pie Chart of Role description



Insights: The pie chart depicting the "Role Description" of cricket players provides a concise overview of the distribution of player roles in the dataset. It showcases the proportion of players in each role category, such as "Player," "Keeper," or "Captain." This visualisation allows for a quick assessment of the player composition within the dataset, highlighting the prevalence of each role.

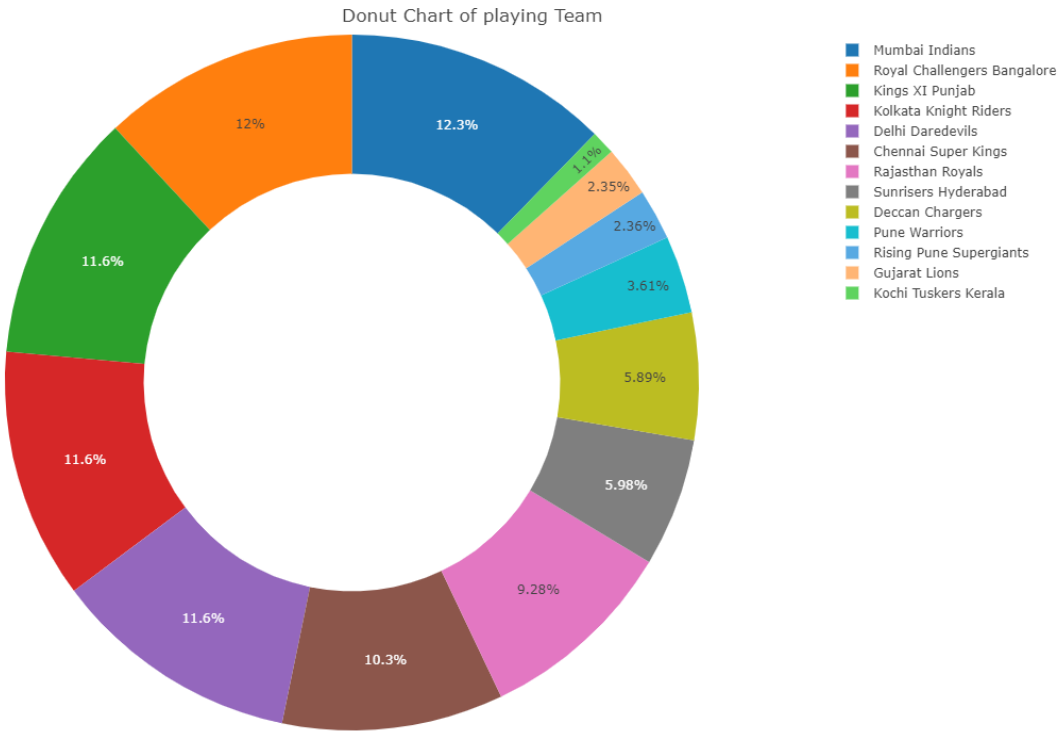
From this chart, it's evident that the majority of players fall into the "Player" category, while roles like "Captain" and "CaptainKeeper" have a smaller representation. This insight into role distribution can inform team management and selection processes, helping teams strategize and allocate responsibilities effectively based on the available talent pool.

(J) Pie Chart of Season/Year



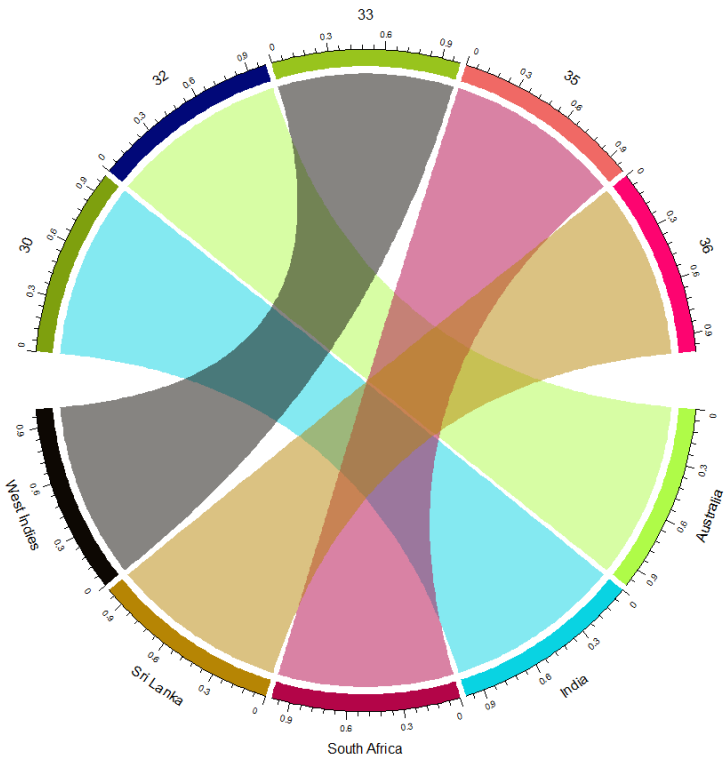
Insights: The pie chart depicting the "Season/Year of tournament won by each team every year" provides a concise overview of team achievements in each season. It offers a quick visual summary of which teams have been consistently successful over the years, highlighting their dominance in specific seasons. This chart enables viewers to identify recurring champions and track their performance trends over time, making it a valuable tool for assessing team success and dynamics in cricket tournaments.

(L) Donut chart of Opposing team [Categorical Data Visualization]



Insights: The chart provides a visual representation of the distribution of players among different playing teams. Larger segments indicate teams with a higher number of players, while smaller segments represent teams with fewer players. Quickly identify the most and least represented playing teams in your dataset, helping you understand player distribution. You can customise the chart's appearance and labels to enhance its clarity and presentation. The Donut Chart offers a clear snapshot of player distribution among various teams in the dataset.

(M) chord diagram of player Team and age as on match



Insights: A chord diagram of player teams and their respective ages on match day provides a compact visualisation of team interactions. Each arc represents a player's team, and the ribbons connecting them reveal the shared ages among players from different teams. The width of the ribbons indicates the strength of age-related connections.
- The chord diagram highlights age similarities between players from various teams.
- Thicker ribbons suggest teams with a higher number of players of similar ages.
- This visualisation can help identify age-related patterns and potential team dynamics based on player ages.
In summary, the chord diagram offers a concise view of age associations across player teams, making it easier to spot age-related trends and connections within the dataset.

Conclusion:

Through extensive data cleaning and visualisation, I've enhanced the dataset's quality and gained valuable insights. The dataset now includes players from various teams and roles, with their ages falling within a specific range. Notably, I addressed outliers and corrected inconsistencies in role descriptions and team names. Visualisations have illuminated team performance, player roles, and age distributions. These insights can inform decisions related to team composition, captaincy, and performance analysis. It's crucial to acknowledge that data cleaning and visualisation are ongoing processes. This process serves as a foundation for ongoing data analysis and exploration.