IPL DATA ANALYSIS USING APACHE SPARK

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

Submitted by

Nirmal Deep Ponnada	**AP21110011492**
Manohar Srinu Nagarjuna Pyla	**AP21110011520**
Lakhmi Jeevananda Reddy Yellu	**AP21110011527**

Under the Guidance of

Dr. Sriramulu Bojjagani

Asst. Professor, Dept. of. CSE

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[November, 2024]

CERTIFICATE

Date: 03rd November, 2024.

This is to certify that the work present in this Project entitled "**IPL DATA ANALYSIS USING SPARK**" has been carried out by **Nirmal Deep Ponnada, Manohar Srinu Nagarjuna Pyla** and **Lakshmi Jeevananda Reddy Yellu** under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in the **School of Engineering and Sciences**.

Supervisor

Dr. Sriramulu Bojjagani

Asst. Professor, Dept. of. CSE

ACKNOWLEDGMENTS

Our deepest gratitude goes to our esteemed supervisor, **Prof. Sriramalu Bojjagani**, whose expertise and insightful guidance were instrumental in the successful completion of this project. We also extend our appreciation to the faculty members of the Computer Science and Engineering department at SRM University AP for fostering a supportive academic environment that nurtured our research and analytical skills.

We extend our heartfelt appreciation to those who played a pivotal role in the successful execution of this group project. We are immensely grateful to our classmates and friends for their unwavering encouragement and support throughout this project. Their stimulating discussions and constructive feedback have significantly enriched our understanding of the subject matter. We also extend our appreciation to the exceptional library staff at SRM University AP for providing seamless access to the necessary resources and data required for conducting our research. Their cooperation and assistance were invaluable in obtaining accurate and relevant information. Lastly, we would like to extend our heartfelt appreciation to our parents for their great support and encouragement throughout our academic pursuits. Their unwavering love, steadfast understanding and belief in our capabilities have been the bedrock of our accomplishments.

We would like to express my deepest gratitude to SRM University, Andhra Pradesh, for providing me with an exceptional academic environment that has fostered our personal and professional growth. The unwavering support and encouragement from the faculty members in the CSE Department have been instrumental in the successful completion of our project.

Table of Contents

ABSTRACT

This project presents a descriptive analysis of data from the Indian Premier League (IPL) to provide insights into team and player performances over multiple seasons. By examining key statistics, including runs scored, wickets taken, and win-loss records, the study highlights trends and patterns within the league. Various visualizations, such as graphs and charts, are utilized to effectively communicate the findings, offering a clear representation of performance metrics and their implications. The analysis aims to enhance the understanding of IPL dynamics, making the results beneficial for teams, analysts, and cricket fans. This project serves as a foundational exploration of IPL data, paving the way for more in-depth analyses in the future.

CHAPTER 01: INTRODUCTION.

The Indian Premier League (IPL) has revolutionized cricket, transforming it into a global phenomenon since its inception in 2008. With its blend of competitive spirit, entertainment, and star-studded line-ups, the IPL attracts millions of viewers and generates substantial revenue. As the league has evolved, so has the volume of data associated with its matches, players, and performances. Analysing this data provides a valuable opportunity to gain insights into the trends and dynamics that shape the league.

This project focuses on descriptive analysis of IPL data, aiming to unveil the intricate patterns underlying player and team performances. By examining various metrics such as batting averages, strike rates, bowling economies, and match outcomes, we can develop a comprehensive understanding of what contributes to success in the league. Descriptive analysis allows us to summarize and interpret this data, making it accessible for fans, analysts, and teams alike.

The significance of this analysis extends beyond mere statistics; it provides context to the performances and highlights the strategic decisions that impact match results. Through visual representations and detailed examinations of the data, this report seeks to elucidate the strengths and weaknesses of teams and players over the years.

Ultimately, this analysis not only enhances our appreciation of the game but also serves as a foundation for future research in sports analytics. By leveraging historical data, we aim to foster a deeper understanding of the IPL's competitive landscape, paving the way for informed decision-making for teams and stakeholders in the cricketing community.

1.1. MOTIVATION OF THE PROJECT.

The Indian Premier League (IPL) consistently generates large datasets, capturing everything from match outcomes to individual player performances. This project aims to leverage Apache Spark's powerful data processing capabilities to analyse IPL match data, focusing on descriptive metrics like team success rates, toss impact, and win margins across seasons and venues. By exploring these elements, we can identify patterns that may offer strategic insights to teams and enhance fan engagement with the league. Spark's ability to efficiently handle large datasets makes it an ideal choice, allowing us to demonstrate practical applications of big data analytics in the sports domain. Additionally, this analysis serves as an opportunity to develop technical proficiency in handling real-world, complex data structures, contributing to the rapidly growing field of sports analytics and underscoring the potential of data-driven insights in professional sports.

1.2. REQUIRED ENVIRONMENTS.

- **Colab Environment:** Python Data Visualization and Preprocessing.
- **Oracle DM Virtual Box:** To run Hortonworks Dockers Sandbox. (Runs Ambari and SSH client.).

1.3. PROBLEM SURVEY

The explosive growth of data generated by the Indian Premier League (IPL) has brought about new challenges in analysing and extracting meaningful insights from sports data. With each season, the IPL accumulates vast amounts of structured and unstructured data, including match statistics, player performances, venue information, and weather conditions, which require advanced data processing solutions for meaningful interpretation. Traditional analysis methods struggle to handle the complexity and scale of such data, limiting the depth of insights that could benefit teams, coaches, and fans alike.

CHAPTER 02: DATA.

2.1. DATA DESCRIPTION.

The dataset used for this IPL analysis project comprises detailed records of matches played across multiple seasons of the Indian Premier League (IPL). This data provides insights into various aspects of each match, including participating teams, venues, outcomes, and player performances. The key attributes in the dataset are as follows:

- **Team1 & Team2:** The names of the teams that played in each match.
- **match_date:** The date on which the match was held.
- **Season_Year:** The year in which the season took place, allowing trend analysis over multiple seasons.
- **Venue_Name:** The stadium or ground where the match was held, useful for understanding team performances across different locations.
- **City_Name & Country_Name:** The location details, with city and country, to examine any geographical patterns.
- **Toss_Winner & Toss_Name:** Information on the team that won the toss and the decision taken (bat or field), which can impact match strategy.
- **match_winner:** The team that won the match, a primary variable for performance analysis.
- **Win_Type & Win_Margin:** The type of win (by runs or wickets) and the margin, indicating the scale of the victory.
- **Outcome_Type:** The outcome of the match (Result or other scenarios), identifying if the match reached a standard result or ended differently.
- **ManOfMach:** The player awarded "Man of the Match," highlighting standout individual performances.

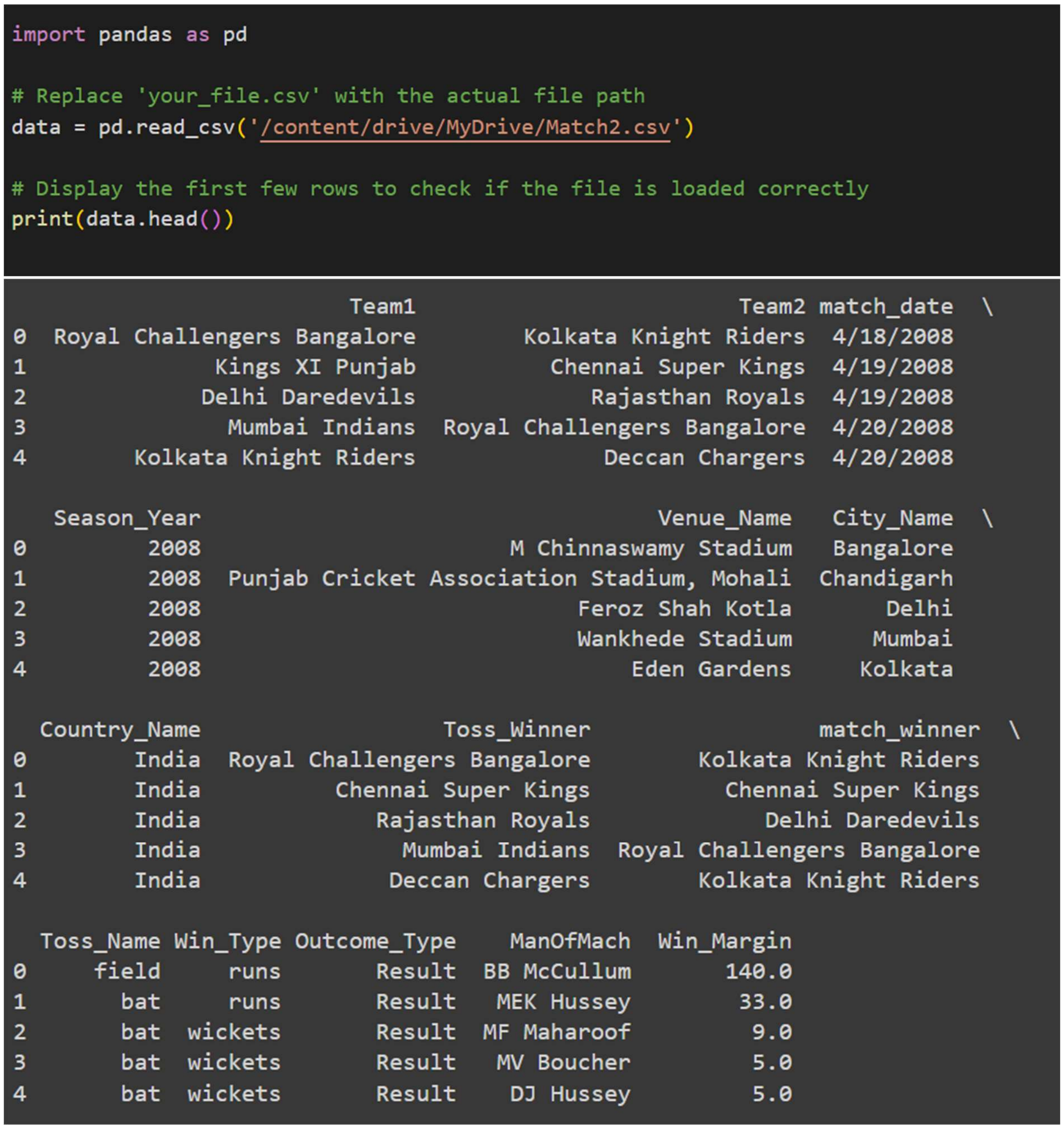
Together, these attributes form a comprehensive dataset for descriptive analysis, enabling insights into match dynamics, team strategies, and seasonal trends.

CHAPTER 03: EXECUTION.

3.1. PREPROCESSING.

- data.head() query gives us the first few rows of the dataset to check if the file loaded correctly.

❖ data.head()

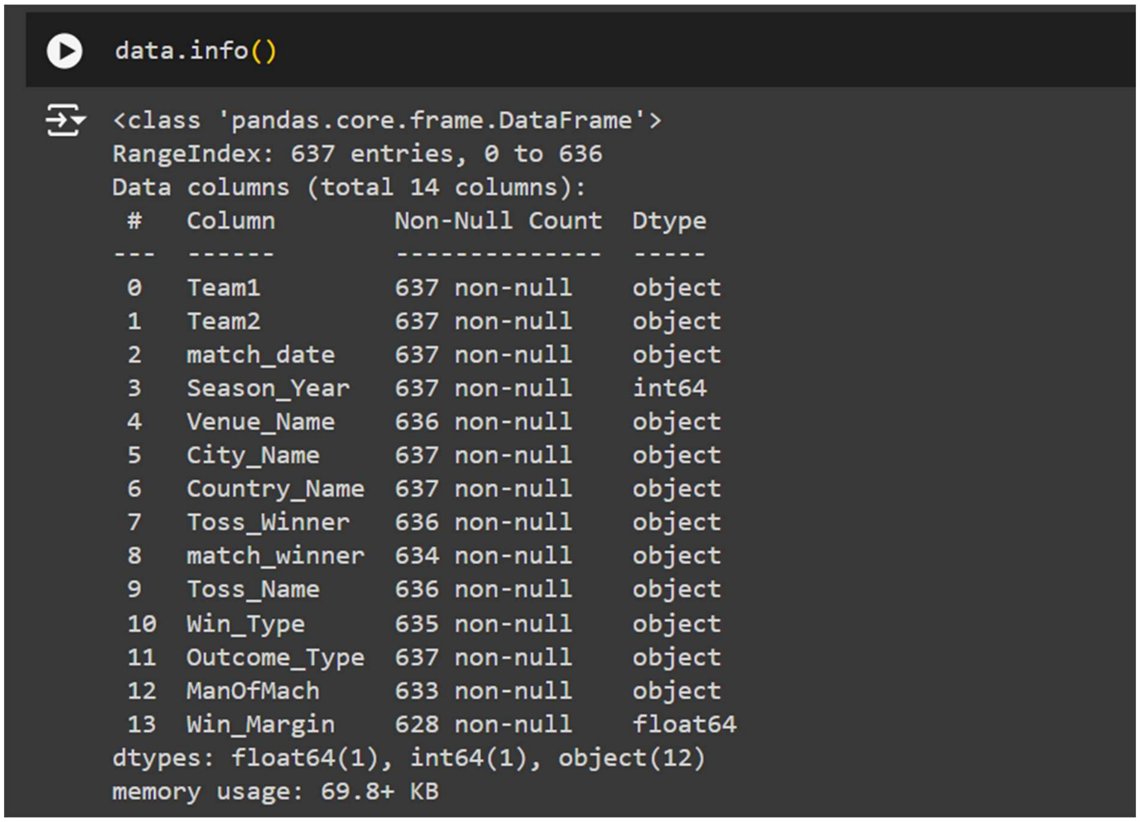


```
import pandas as pd

# Replace 'your_file.csv' with the actual file path
data = pd.read_csv('/content/drive/MyDrive/Match2.csv')

# Display the first few rows to check if the file is loaded correctly
print(data.head())
```

```
                        Team1                        Team2 match_date  \
0  Royal Challengers Bangalore        Kolkata Knight Riders  4/18/2008
1              Kings XI Punjab          Chennai Super Kings  4/19/2008
2             Delhi Daredevils             Rajasthan Royals  4/19/2008
3               Mumbai Indians  Royal Challengers Bangalore  4/20/2008
4        Kolkata Knight Riders              Deccan Chargers  4/20/2008

   Season_Year                               Venue_Name   City_Name  \
0         2008                    M Chinnaswamy Stadium   Bangalore
1         2008  Punjab Cricket Association Stadium, Mohali  Chandigarh
2         2008                         Feroz Shah Kotla       Delhi
3         2008                         Wankhede Stadium      Mumbai
4         2008                             Eden Gardens     Kolkata

  Country_Name                  Toss_Winner                 match_winner  \
0        India  Royal Challengers Bangalore        Kolkata Knight Riders
1        India          Chennai Super Kings          Chennai Super Kings
2        India             Rajasthan Royals             Delhi Daredevils
3        India               Mumbai Indians  Royal Challengers Bangalore
4        India              Deccan Chargers        Kolkata Knight Riders

  Toss_Name Win_Type Outcome_Type    ManOfMach  Win_Margin
0     field     runs       Result  BB McCullum       140.0
1       bat     runs       Result   MEK Hussey        33.0
2       bat  wickets       Result  MF Maharoof         9.0
3       bat  wickets       Result   MV Boucher         5.0
4       bat  wickets       Result    DJ Hussey         5.0
```

❖ **info()** method gives description about the attributes and their corresponding datatypes.

❖ **df.info()**

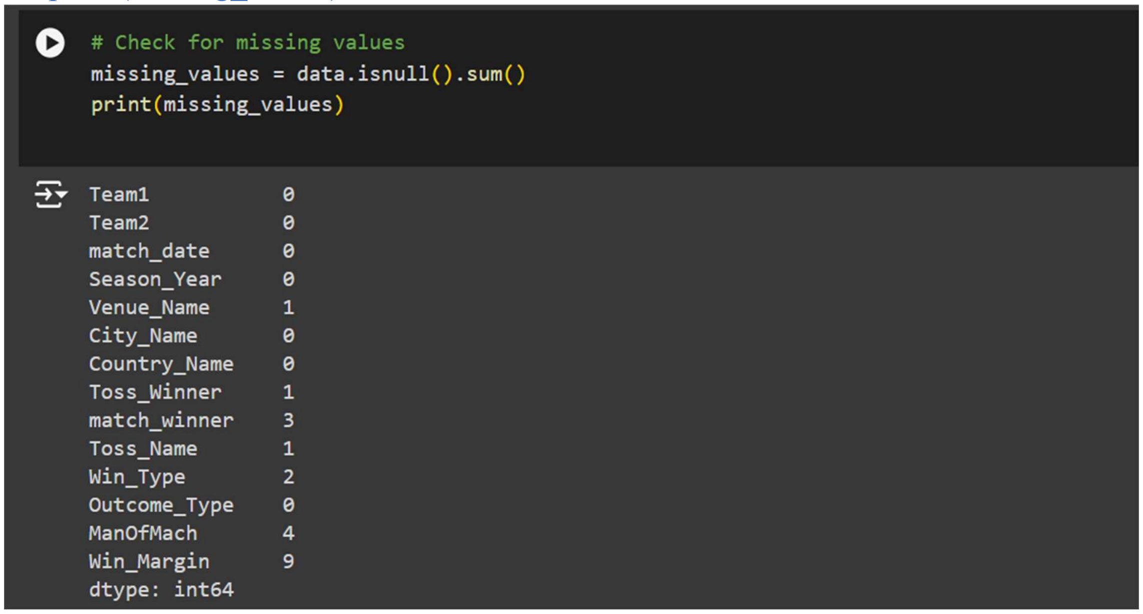


```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 637 entries, 0 to 636
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Team1          637 non-null    object 
 1   Team2          637 non-null    object 
 2   match_date     637 non-null    object 
 3   Season_Year    637 non-null    int64  
 4   Venue_Name     636 non-null    object 
 5   City_Name      637 non-null    object 
 6   Country_Name   637 non-null    object 
 7   Toss_Winner    636 non-null    object 
 8   match_winner   634 non-null    object 
 9   Toss_Name      636 non-null    object 
 10  Win_Type       635 non-null    object 
 11  Outcome_Type   637 non-null    object 
 12  ManOfMach      633 non-null    object 
 13  Win_Margin     628 non-null    float64
dtypes: float64(1), int64(1), object(12)
memory usage: 69.8+ KB
```

❖ **data.isnull().sum()** prints the number of missing values in the all the attributes.

❖ missing_values = data.isnull().sum()
print (missing_values)

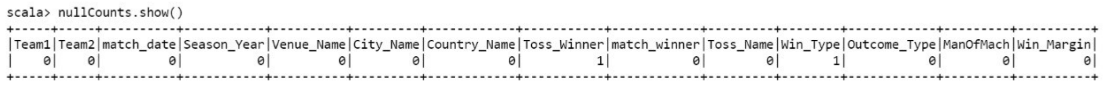


```
# Check for missing values
missing_values = data.isnull().sum()
print(missing_values)
```

```
Team1           0
Team2           0
match_date      0
Season_Year     0
Venue_Name      1
City_Name       0
Country_Name    0
Toss_Winner     1
match_winner    3
Toss_Name       1
Win_Type        2
Outcome_Type    0
ManOfMach       4
Win_Margin      9
dtype: int64
```

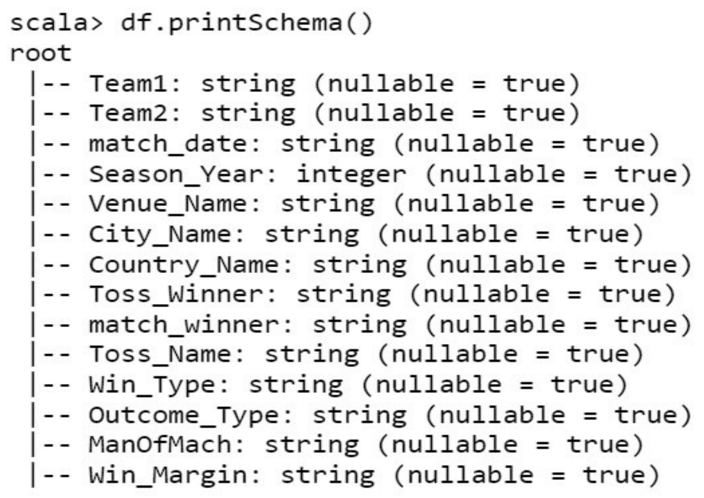
3.2. CODE IMPLEMENTATION.

❖ nullCounts.show(): This command results in displaying the number of null values in every attribute.



```
scala> nullCounts.show()
+-----+-----+----------+-----------+----------+---------+------------+-----------+------------+---------+--------+------------+---------+----------+
|Team1|Team2|match_date|Season_Year|Venue_Name|City_Name|Country_Name|Toss_Winner|match_winner|Toss_Name|Win_Type|Outcome_Type|ManOfMach|Win_Margin|
|    0|    0|         0|          0|         0|        0|           0|          1|           0|        0|       1|           0|        0|         0|
+-----+-----+----------+-----------+----------+---------+------------+-----------+------------+---------+--------+------------+---------+----------+
```

❖ df.printSchema(): This command displays the schema of all the attributes.

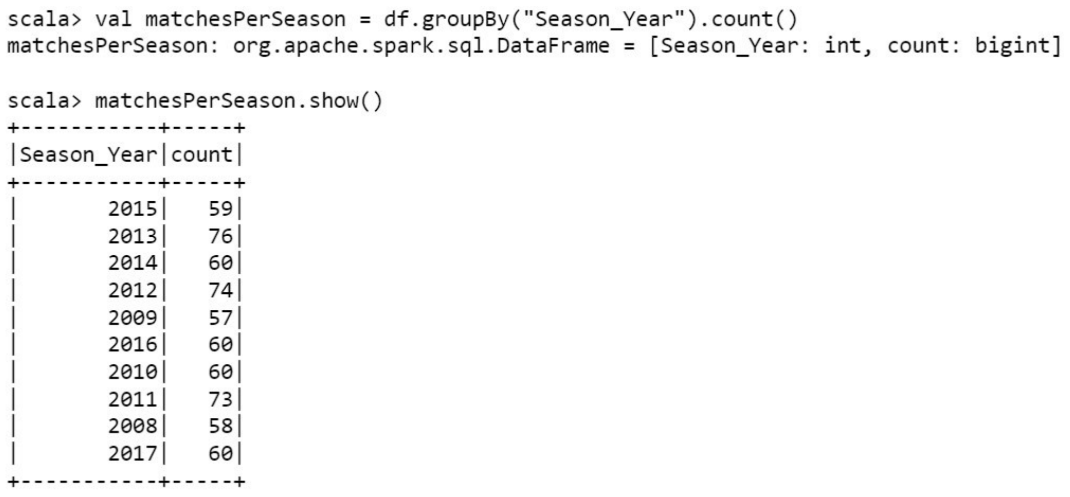


```
scala> df.printSchema()
root
 |-- Team1: string (nullable = true)
 |-- Team2: string (nullable = true)
 |-- match_date: string (nullable = true)
 |-- Season_Year: integer (nullable = true)
 |-- Venue_Name: string (nullable = true)
 |-- City_Name: string (nullable = true)
 |-- Country_Name: string (nullable = true)
 |-- Toss_Winner: string (nullable = true)
 |-- match_winner: string (nullable = true)
 |-- Toss_Name: string (nullable = true)
 |-- Win_Type: string (nullable = true)
 |-- Outcome_Type: string (nullable = true)
 |-- ManOfMach: string (nullable = true)
 |-- Win_Margin: string (nullable = true)
```

❖ This query displays the count of the matches held in respective years.

❖ scala > val matchesPerSeason = df.groupBy("Season_Year").count()
matchesPerSeason: org.apache.spark.sql.DataFrame = [Season_Year: int, count: bigint]

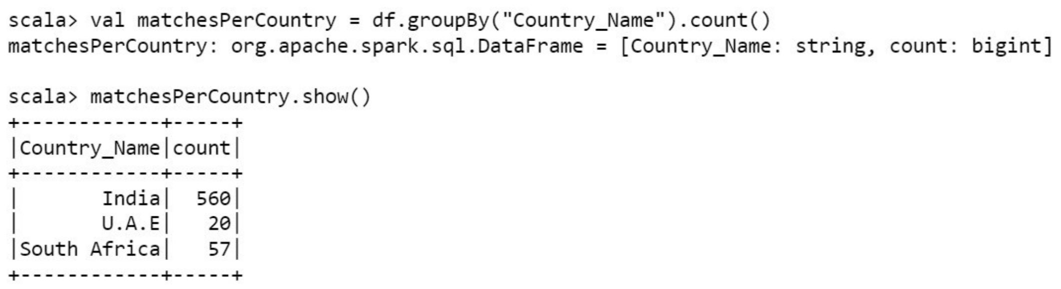
scala > matchesPerSeason.show()



```
scala> val matchesPerSeason = df.groupBy("Season_Year").count()
matchesPerSeason: org.apache.spark.sql.DataFrame = [Season_Year: int, count: bigint]

scala> matchesPerSeason.show()
+-----------+-----+
|Season_Year|count|
+-----------+-----+
|       2015|   59|
|       2013|   76|
|       2014|   60|
|       2012|   74|
|       2009|   57|
|       2016|   60|
|       2010|   60|
|       2011|   73|
|       2008|   58|
|       2017|   60|
+-----------+-----+
```

- This query displays the number of matches played in respective countries.

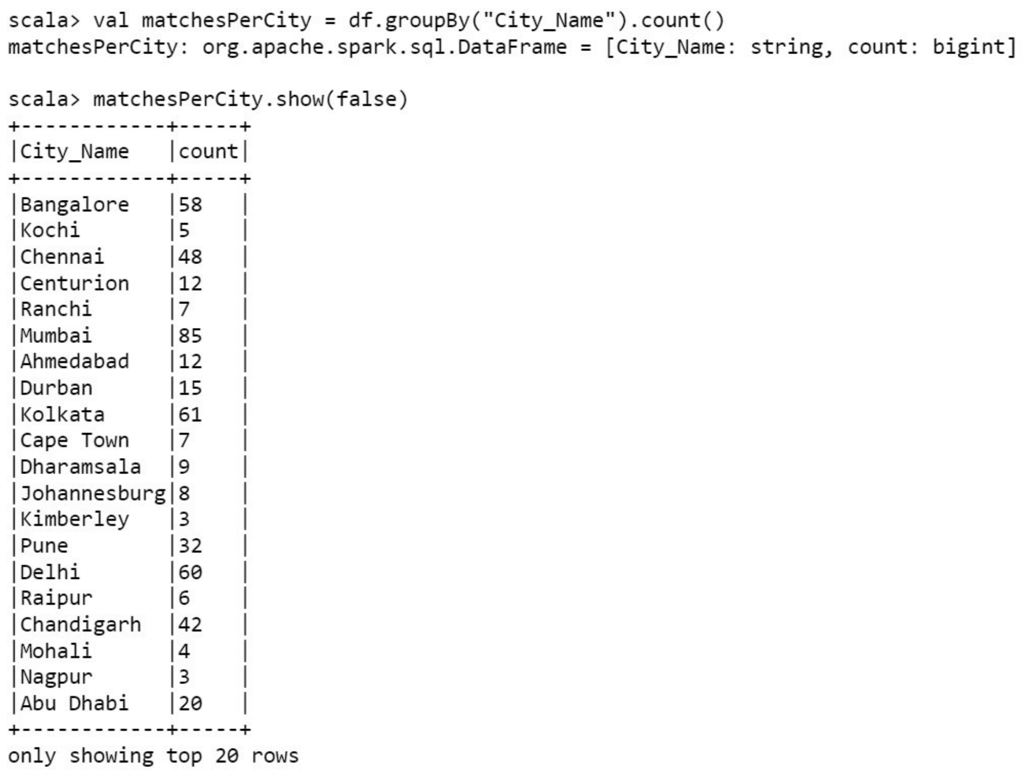
❖ scala> val matchesPerCountry = df.groupBy("Country_Name") . count()
matchesPerCountry: org. apache.spark. sq1.DataFrame = [Country_Name: string, count: bigint]
scala> matchesPerCountry. show()



```
scala> val matchesPerCountry = df.groupBy("Country_Name").count()
matchesPerCountry: org.apache.spark.sql.DataFrame = [Country_Name: string, count: bigint]

scala> matchesPerCountry.show()
+------------+-----+
|Country_Name|count|
+------------+-----+
|       India|  560|
|       U.A.E|   20|
|South Africa|   57|
+------------+-----+
```

- This query displays the number of matches played in respective cities.

❖ scala> val matchesPercity = df.groupBy("City_Name") . count()
matchesPerCity: org.apache. spark. sql.DataFrame = [city_Name: string, count: bigint]
scala> matchesPerCity.show(fa1se)

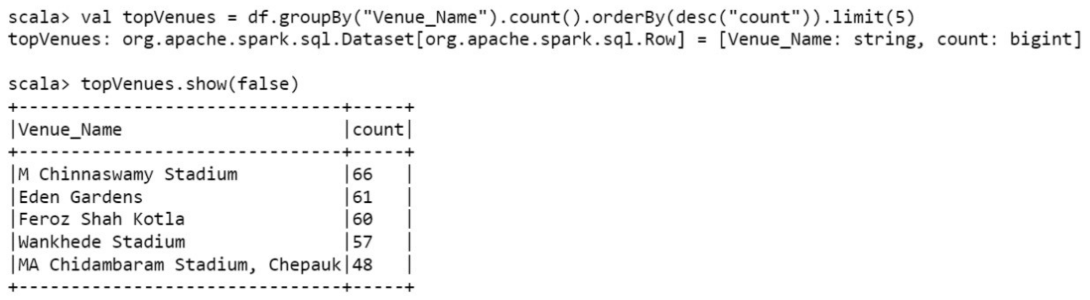


```
scala> val matchesPerCity = df.groupBy("City_Name").count()
matchesPerCity: org.apache.spark.sql.DataFrame = [City_Name: string, count: bigint]

scala> matchesPerCity.show(false)
+------------+-----+
|City_Name   |count|
+------------+-----+
|Bangalore   |58   |
|Kochi       |5    |
|Chennai     |48   |
|Centurion   |12   |
|Ranchi      |7    |
|Mumbai      |85   |
|Ahmedabad   |12   |
|Durban      |15   |
|Kolkata     |61   |
|Cape Town   |7    |
|Dharamsala  |9    |
|Johannesburg|8    |
|Kimberley   |3    |
|Pune        |32   |
|Delhi       |60   |
|Raipur      |6    |
|Chandigarh  |42   |
|Mohali      |4    |
|Nagpur      |3    |
|Abu Dhabi   |20   |
+------------+-----+
only showing top 20 rows
```

- This query displays the top 5 venues (Stadiums) in which highest number of matches took place.

❖ scala> val topVenues = df.groupBy("Venue_Name").count() . limit(5)

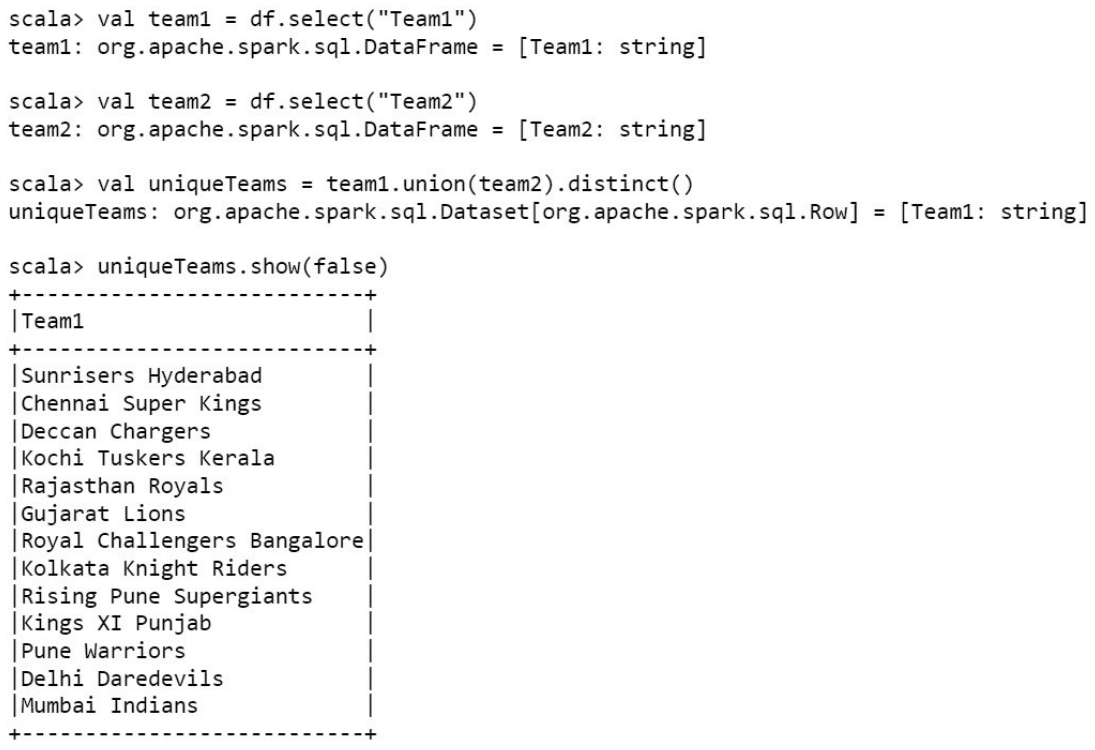
topVenues: org.apache.spark.sql.Dataset[org.apache.spark.sqI.Row] = [Venue_Name: string, count: bigint]
scala> topVenues.show(false)



```
scala> val topVenues = df.groupBy("Venue_Name").count().orderBy(desc("count")).limit(5)
topVenues: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Venue_Name: string, count: bigint]

scala> topVenues.show(false)
+------------------------------+-----+
|Venue_Name                    |count|
+------------------------------+-----+
|M Chinnaswamy Stadium         |66   |
|Eden Gardens                  |61   |
|Feroz Shah Kotla              |60   |
|Wankhede Stadium              |57   |
|MA Chidambaram Stadium, Chepauk|48   |
+------------------------------+-----+
```

- This query displays all the teams who participated in the league.

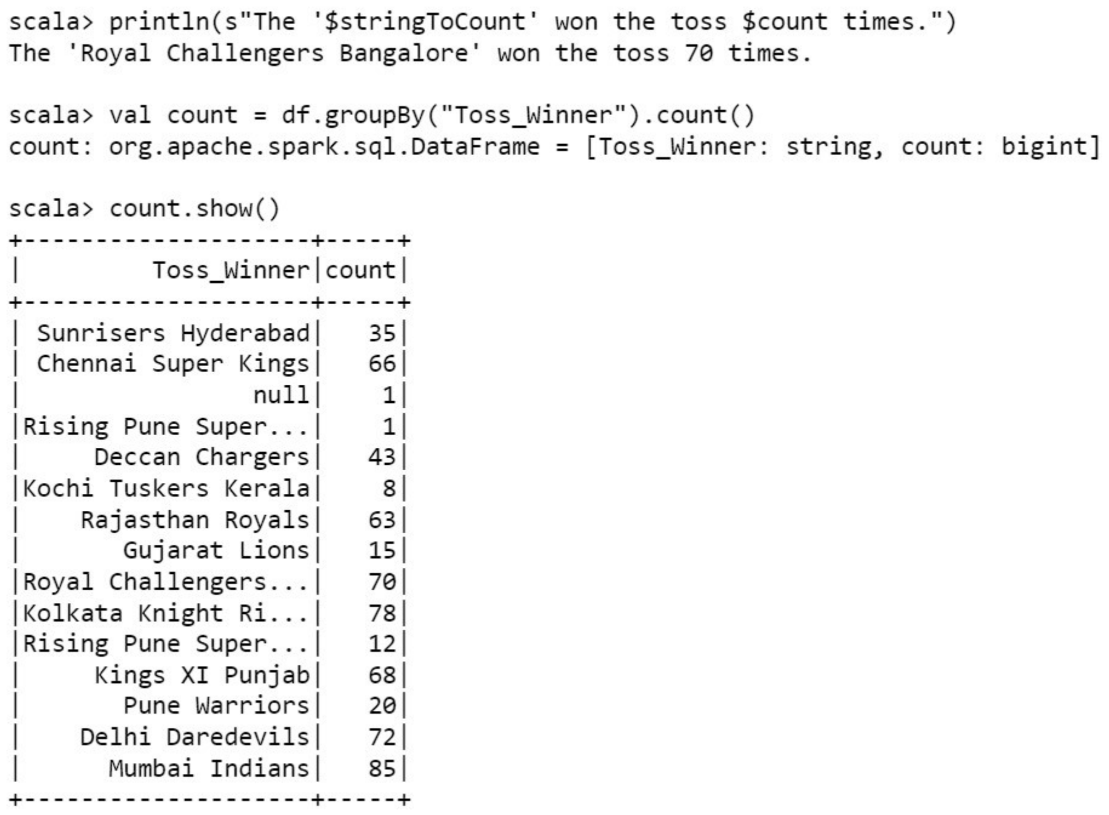
❖ scala> val team1 = df. select("Team1")
team1: org.apache.spark.sql.DataFrame = [Team1: string]
scala> val team2 = df.se1ect("Team2")
team2: org.apache.spark.sq1.DataFrame = [Team2: string]
scala> val uniqueTeams = teaml.union(team2).distinct()
uniqueTeams: org.apache.spark.sq1.Dataset[org.apache.spark.sql.Row] = [Team1: string]
scala> uniqueTeams.show(fa1se)



```
scala> val team1 = df.select("Team1")
team1: org.apache.spark.sql.DataFrame = [Team1: string]

scala> val team2 = df.select("Team2")
team2: org.apache.spark.sql.DataFrame = [Team2: string]

scala> val uniqueTeams = team1.union(team2).distinct()
uniqueTeams: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Team1: string]

scala> uniqueTeams.show(false)
+---------------------------+
|Team1                      |
+---------------------------+
|Sunrisers Hyderabad        |
|Chennai Super Kings        |
|Deccan Chargers            |
|Kochi Tuskers Kerala       |
|Rajasthan Royals           |
|Gujarat Lions              |
|Royal Challengers Bangalore|
|Kolkata Knight Riders      |
|Rising Pune Supergiants    |
|Kings XI Punjab            |
|Pune Warriors              |
|Delhi Daredevils           |
|Mumbai Indians             |
+---------------------------+
```

- This query displays the total number of times a unique team has won the toss.

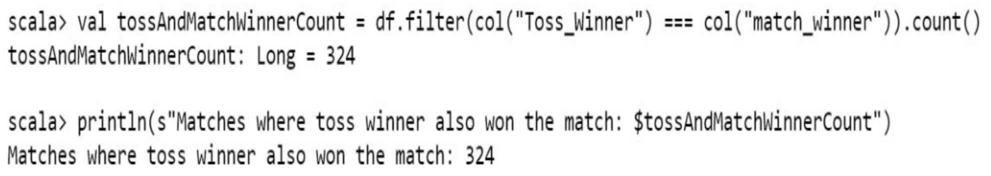
❖ Scala > println(s"The '$stringToCount' won the toss $count times.")
The ' Royal Challengers Bangalore' won the toss 70 times.
Scala > val count = df.groupBy("Toss_Winner").count()
count: org.apache.spark.sql.DataFrame = [Toss_Winner: string, count: bigint]



```
scala> println(s"The '$stringToCount' won the toss $count times.")
The 'Royal Challengers Bangalore' won the toss 70 times.

scala> val count = df.groupBy("Toss_Winner").count()
count: org.apache.spark.sql.DataFrame = [Toss_Winner: string, count: bigint]

scala> count.show()
+--------------------+-----+
|         Toss_Winner|count|
+--------------------+-----+
| Sunrisers Hyderabad|   35|
| Chennai Super Kings|   66|
|                null|    1|
|Rising Pune Super...|    1|
|     Deccan Chargers|   43|
|Kochi Tuskers Kerala|    8|
|    Rajasthan Royals|   63|
|       Gujarat Lions|   15|
|Royal Challengers...|   70|
|Kolkata Knight Ri...|   78|
|Rising Pune Super...|   12|
|     Kings XI Punjab|   68|
|       Pune Warriors|   20|
|    Delhi Daredevils|   72|
|      Mumbai Indians|   85|
+--------------------+-----+
```

- This query displays the Matches where the toss winners also won the match.

❖ Scala > val tossAndMatchWinnerCount = df.filter(col("Toss_Winner"))/count()
tossAndMatchWinnerCount: Long=324
scala > println(s"Matches where toss winner also won the match: $tossAndMatchWinnerCount")

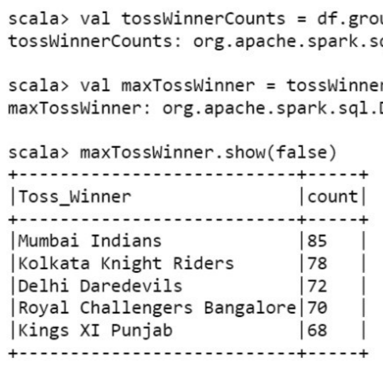


```
scala> val tossAndMatchWinnerCount = df.filter(col("Toss_Winner") === col("match_winner")).count()
tossAndMatchWinnerCount: Long = 324

scala> println(s"Matches where toss winner also won the match: $tossAndMatchWinnerCount")
Matches where toss winner also won the match: 324
```

- This query displays the Maximum Toss winners in all the seasons.

❖ scala> val tossWinnerCounts = df.groupBy("Toss_Winner") . count()

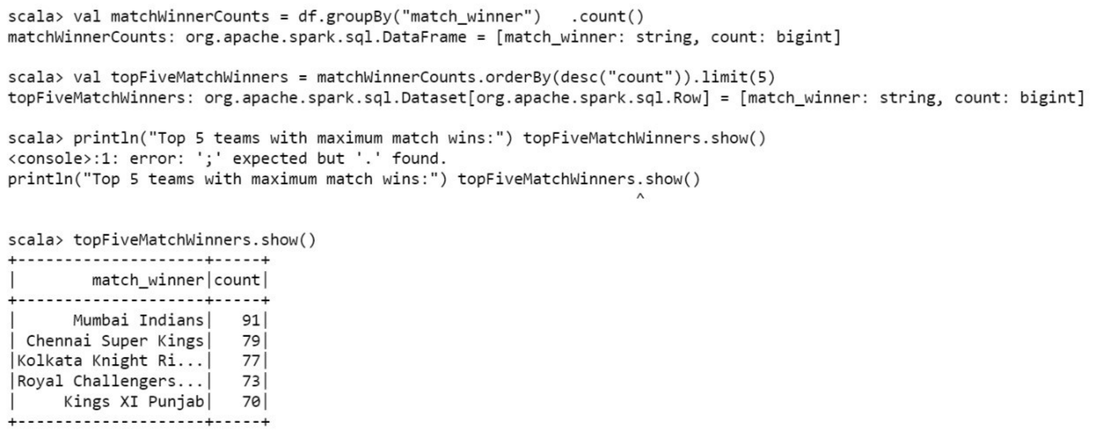
TossWinnerCounts: org.apache@spark.sql.DataFrame = [Toss_Winner: string, count: bigint]
scala> val maxTossWinner = tossWinnerCounts.orderBy(desc("count")).1imit(5)
maxTossWinner: org. apache.spark.sqI.Dataset[org. apache.spark. sqI.Row]
= [Toss_Winner: string, count: bigint]
scala> maxTossWinner. show(false)



```
scala> val tossWinnerCounts = df.groupBy("Toss_Winner").count()
tossWinnerCounts: org.apache.spark.sql.DataFrame = [Toss_Winner: string, count: bigint]

scala> val maxTossWinner = tossWinnerCounts.orderBy(desc("count")).limit(5)
maxTossWinner: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Toss_Winner: string, count: bigint]

scala> maxTossWinner.show(false)
+---------------------------+-----+
|Toss_Winner                |count|
+---------------------------+-----+
|Mumbai Indians             |85   |
|Kolkata Knight Riders      |78   |
|Delhi Daredevils           |72   |
|Royal Challengers Bangalore|70   |
|Kings XI Punjab            |68   |
+---------------------------+-----+
```

- This query displays the top 5 Teams who have the highest wins.

❖ scala> val matchWinnerCounts = df.groupBy("match_winner"). count ( )
matchWinnerCounts: org.apache.spark.sq1.DataFrame = [match_winner: string, count: bigint]
scala> val topFiveMatchWinners = matchWinnerCounts . limit(5)
topFiveMatchWinners: org.apache.spark.sq1.Dataset[org.apache.spark.sq1.Row] = [match_winner: string, count: bigint]
scala> println("Top 5 teams with maximum match wins: ")topFiveMatchWinners.show()
print1n("Top S teams with maximum match wins:") topFiveMatchWinners.show()

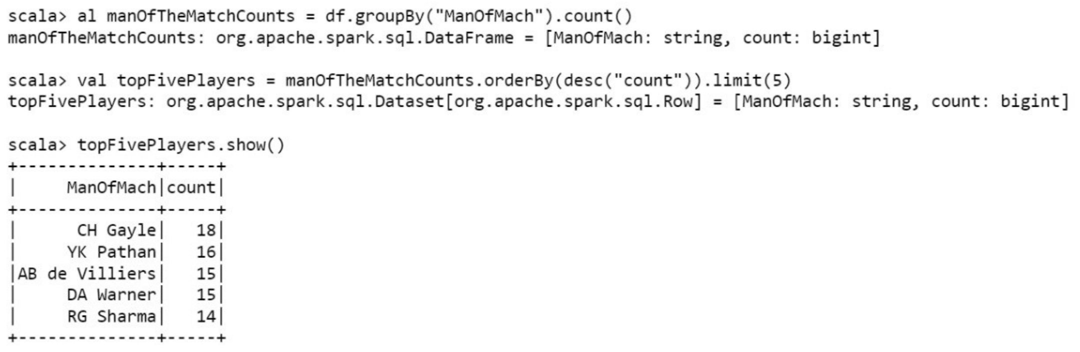


```
scala> val matchWinnerCounts = df.groupBy("match_winner")   .count()
matchWinnerCounts: org.apache.spark.sql.DataFrame = [match_winner: string, count: bigint]

scala> val topFiveMatchWinners = matchWinnerCounts.orderBy(desc("count")).limit(5)
topFiveMatchWinners: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [match_winner: string, count: bigint]

scala> println("Top 5 teams with maximum match wins:") topFiveMatchWinners.show()
<console>:1: error: ';' expected but '.' found.
println("Top 5 teams with maximum match wins:") topFiveMatchWinners.show()
                                                                   ^

scala> topFiveMatchWinners.show()
+--------------------+-----+
|        match_winner|count|
+--------------------+-----+
|      Mumbai Indians|   91|
| Chennai Super Kings|   79|
|Kolkata Knight Ri...|   77|
|Royal Challengers...|   73|
|     Kings XI Punjab|   70|
+--------------------+-----+
```

- This query displays the top 5 best players of the league.

❖ scala> al manOfTheMatchCounts = df.groupBy( ) . count()
manOfTheMatchCounts: org.apache.spark.sqI.DataFrame = [ManOfMach: string, count: bigint]

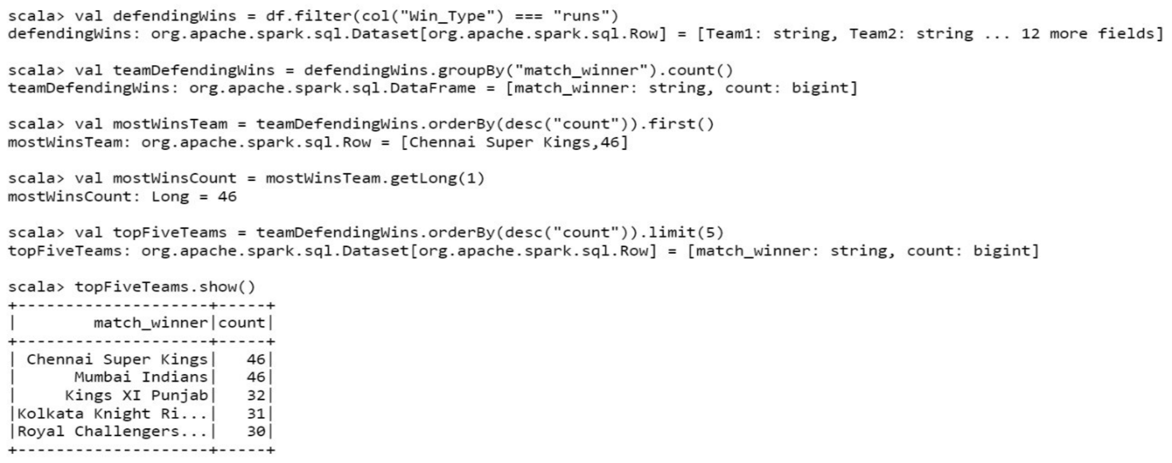
scala> val topFivePIayers = manOfTheMatchCounts . orderBy (desc (" count")) . limit(5)
topFiveP1ayers: org.apache.spark.sq1.Dataset[org.apache.spark.sq1.Row] = [ManOfMach: string, count: bigint]
scala> topFiveP1ayers.show()



```
scala> al manOfTheMatchCounts = df.groupBy("ManOfMach").count()
manOfTheMatchCounts: org.apache.spark.sql.DataFrame = [ManOfMach: string, count: bigint]

scala> val topFivePlayers = manOfTheMatchCounts.orderBy(desc("count")).limit(5)
topFivePlayers: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [ManOfMach: string, count: bigint]

scala> topFivePlayers.show()
+--------------+-----+
|     ManOfMach|count|
+--------------+-----+
|      CH Gayle|   18|
|     YK Pathan|   16|
|AB de Villiers|   15|
|     DA Warner|   15|
|     RG Sharma|   14|
+--------------+-----+
```

- This query displays the top 5 teams who won by choosing Batting in the toss.

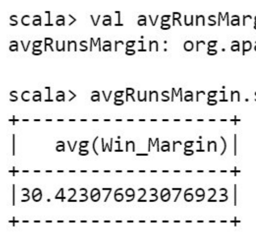
❖ scala> val defendingWins = df.fi1ter(c01("Win_Type") === "runs")defendingWins: org.apache.spark.sq1.Dataset[org.apache.spark.sq1.Row] = [Teaml: string, Team2 :string…12 more fields]
scala> val teamDefendingWins = defendingWins . groupBy("match_winner") . count()
teamDefendingWins: org.apache.spark.sqI.DataFrame =[match_winner: string, count: bigint]
scala> val mostWinsTeam = teamDefendingWins . orderBy(desc (" count")) . first()
mostWinsTeam: org.apache.spark. sql. Row = [Chennai Super Kings,46]
scala> val mostWinsCount
mostWinsCount: Long = 46 = mostWinsTeam.getLong(1)
scala> val topFiveTeams = teamDefendingWins . orderBy(desc(" count")). limit(5)
topFiveTeams: org.apache.spark.sq1.Dataset[org.apache.spark.sql.Row] = [match_winner: string, count: bigint]
scala> topFiveTeams. show()



```
scala> val defendingWins = df.filter(col("Win_Type") === "runs")
defendingWins: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Team1: string, Team2: string ... 12 more fields]

scala> val teamDefendingWins = defendingWins.groupBy("match_winner").count()
teamDefendingWins: org.apache.spark.sql.DataFrame = [match_winner: string, count: bigint]

scala> val mostWinsTeam = teamDefendingWins.orderBy(desc("count")).first()
mostWinsTeam: org.apache.spark.sql.Row = [Chennai Super Kings,46]

scala> val mostWinsCount = mostWinsTeam.getLong(1)
mostWinsCount: Long = 46

scala> val topFiveTeams = teamDefendingWins.orderBy(desc("count")).limit(5)
topFiveTeams: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [match_winner: string, count: bigint]

scala> topFiveTeams.show()
+--------------------+-----+
|        match_winner|count|
+--------------------+-----+
| Chennai Super Kings|   46|
|      Mumbai Indians|   46|
|     Kings XI Punjab|   32|
|Kolkata Knight Ri...|   31|
|Royal Challengers...|   30|
+--------------------+-----+
```

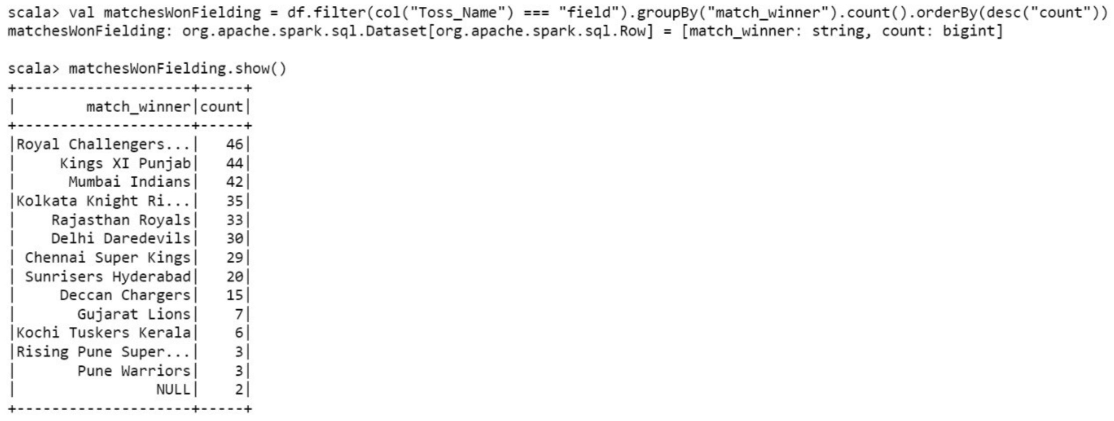
- This query displays the win margin when the win type is runs.

❖ scala> val avgRunsMargin = df.fiIter(col("Win_Type") === "runs"). agg(avg("Win_Margin"))
avgRunsMargin: org. apache.spark. sql .DataFrame = [avg(Win_Margin): double]
scala> avgRunsMargin. show()


```
scala> val avgRunsMargin = df.filter(col("Win_Type") === "runs").agg(avg("Win_Margin"))
avgRunsMargin: org.apache.spark.sql.DataFrame = [avg(Win_Margin): double]

scala> avgRunsMargin.show()
+------------------+
|   avg(Win_Margin)|
+------------------+
|30.423076923076923|
+------------------+
```

- This query displays the list of all teams and the number of matches each team won by choosing fielding in the toss.

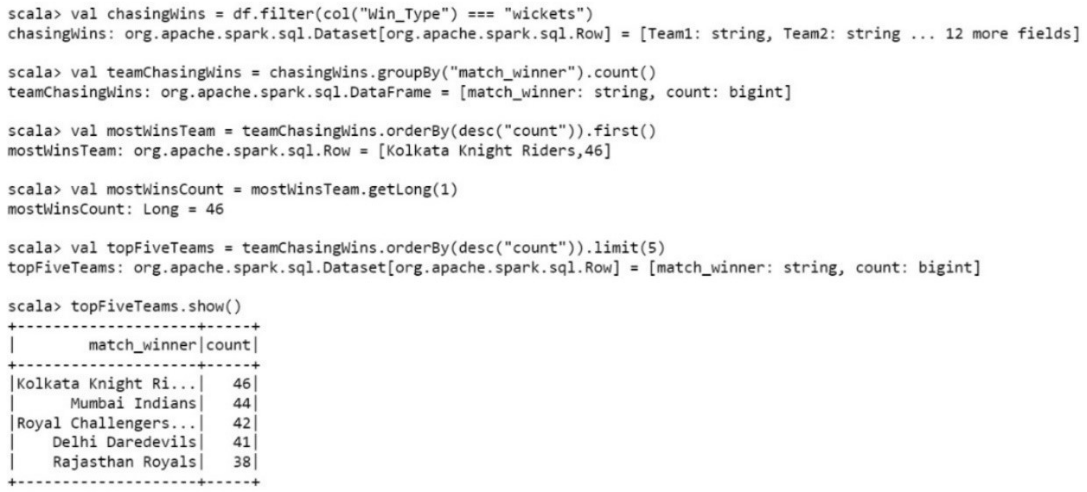
❖ scala> val matchesWonFielding = "Toss_Name") === "field").groupBy("match_winner").count().orderBy(desc("count"))
matcheswonFie1ding: org.apache.spark.sq1.Dataset[org.apache.spark.sq1.Row) = [match_winner: string, count: bigint]


```
scala> val matchesWonFielding = df.filter(col("Toss_Name") === "field").groupBy("match_winner").count().orderBy(desc("count"))
matchesWonFielding: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [match_winner: string, count: bigint]

scala> matchesWonFielding.show()
+-------------------+-----+
|       match_winner|count|
+-------------------+-----+
|Royal Challengers...|   46|
|     Kings XI Punjab|   44|
|      Mumbai Indians|   42|
|Kolkata Knight Ri...|   35|
|    Rajasthan Royals|   33|
|    Delhi Daredevils|   30|
| Chennai Super Kings|   29|
| Sunrisers Hyderabad|   20|
|     Deccan Chargers|   15|
|       Gujarat Lions|    7|
|Kochi Tuskers Kerala|    6|
|Rising Pune Super...|    3|
|       Pune Warriors|    3|
|                NULL|    2|
+-------------------+-----+
```

- This query displays the top 5 teams whose win type is wickets.

❖ scala> val defendingWins = df.fi1ter(c01("Win_Type") === "runs")defendingWins: org.apache.spark.sq1.Dataset[org.apache.spark.sq1.Row] = [Teaml: string, Team2 :string…12 more fields]
scala> val teamDefendingWins = defendingWins . groupBy("match_winner") . count()
teamDefendingWins: org.apache.spark.sqI.DataFrame =[match_winner: string, count: bigint]
scala> val mostWinsTeam = teamDefendingWins . orderBy(desc (" count")) . first()
mostWinsTeam: org.apache.spark. sql. Row = [Kolkata Knight Riders,46]
scala> val mostWinsCount = mostWinsCount.getLong(1)
mostWinsCount: Long=46.
scala> val topFiveTeams = teamDefendingWins . orderBy(desc(" count")). limit(5)

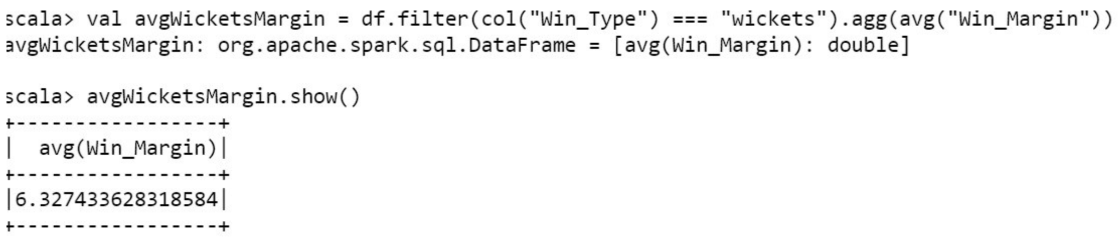
topFiveTeams: org.apache.spark.sq1.Dataset[org.apache.spark.sql.Row] = [match_winner: string, count: bigint]
scala> topFiveTeams. show()



```
scala> val chasingWins = df.filter(col("Win_Type") === "wickets")
chasingWins: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Team1: string, Team2: string ... 12 more fields]

scala> val teamChasingWins = chasingWins.groupBy("match_winner").count()
teamChasingWins: org.apache.spark.sql.DataFrame = [match_winner: string, count: bigint]

scala> val mostWinsTeam = teamChasingWins.orderBy(desc("count")).first()
mostWinsTeam: org.apache.spark.sql.Row = [Kolkata Knight Riders,46]

scala> val mostWinsCount = mostWinsTeam.getLong(1)
mostWinsCount: Long = 46

scala> val topFiveTeams = teamChasingWins.orderBy(desc("count")).limit(5)
topFiveTeams: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [match_winner: string, count: bigint]

scala> topFiveTeams.show()
+--------------------+-----+
|        match_winner|count|
+--------------------+-----+
|Kolkata Knight Ri...|   46|
|      Mumbai Indians|   44|
|Royal Challengers...|   42|
|    Delhi Daredevils|   41|
|    Rajasthan Royals|   38|
+--------------------+-----+
```

- This query displays the average win margin when the win type is wickets.

❖ Scala > val avgWicketsMargin = df.filter(col("Win_Type") === "wickets"). Agg(avg("Win_Margin"))
avgWicketsMargin: org.apache.apark.sql.DataFrame = [avg(Win_Margin): double]
scala > avgWicketsMargin.show()

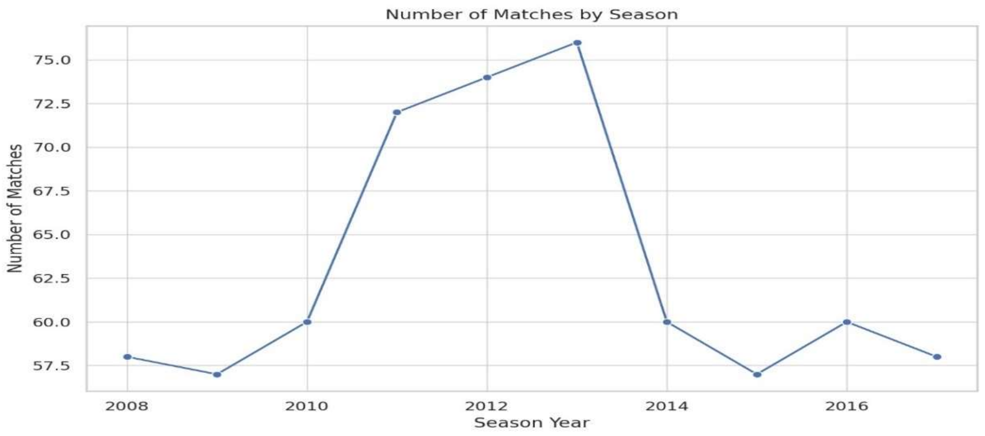


```
scala> val avgWicketsMargin = df.filter(col("Win_Type") === "wickets").agg(avg("Win_Margin"))
avgWicketsMargin: org.apache.spark.sql.DataFrame = [avg(Win_Margin): double]

scala> avgWicketsMargin.show()
+-----------------+
|  avg(Win_Margin)|
+-----------------+
|6.327433628318584|
+-----------------+
```

3.3. VISUALISATION.

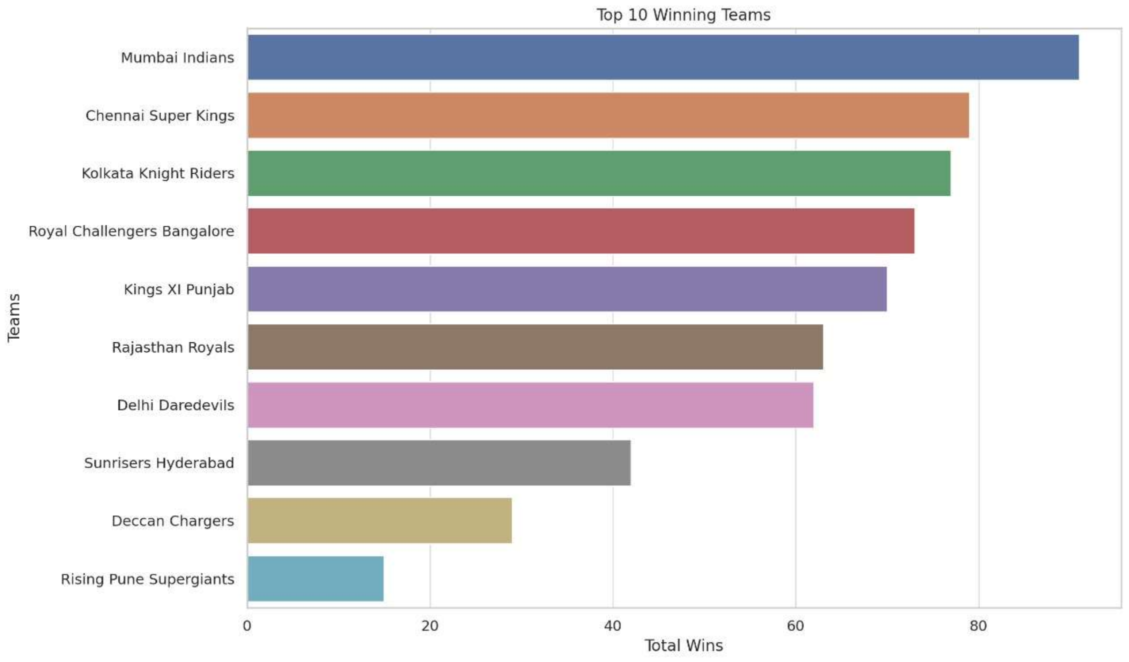
- **Matches by Season - Shows match trends over seasons.**

plt.figure(figsize=(10, 6))
season_counts = data['Season_Year'].value_counts().sort_index()
sns.lineplot(x=season_counts.index, y=season_counts.values, marker="o")
plt.title('Number of Matches by Season')
plt.xlabel('Season Year')
plt.ylabel('Number of Matches')
plt.show()



- **Winning Teams Distribution - Highlights the top 10 teams by number of wins.**

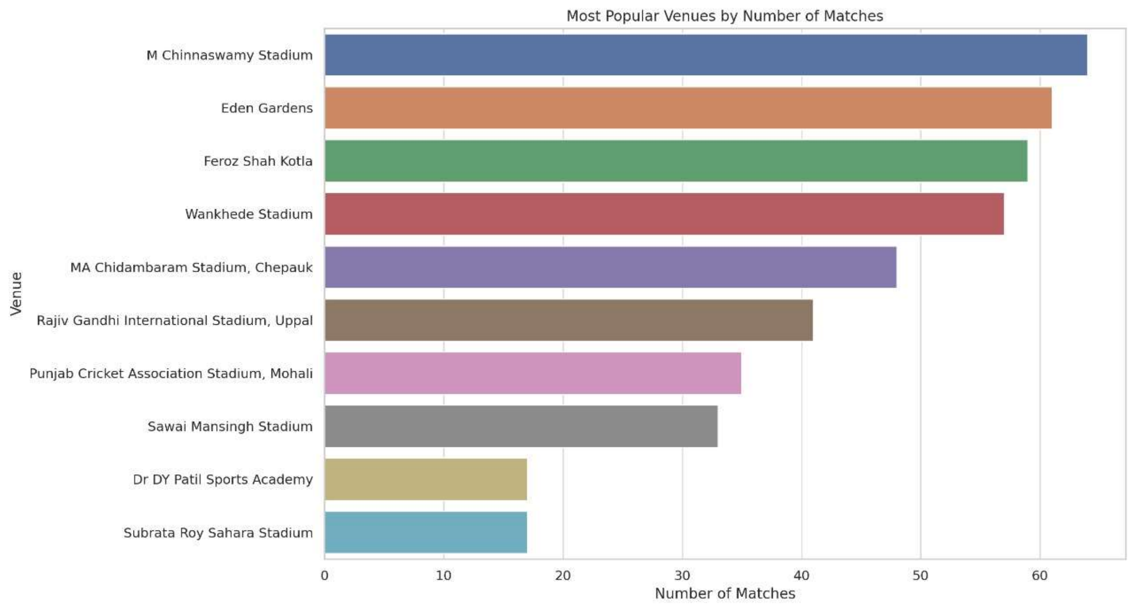
```
plt.figure(figsize=(12, 8))
win_counts = data['match_winner'].value_counts().head(10)  # Top 10 teams for
readability
sns.barplot(x=win_counts.values, y=win_counts.index)
plt.title('Top 10 Winning Teams')
plt.xlabel('Total Wins')
plt.ylabel('Teams')
plt.show()
```



- **Venue Popularity - Identifies the most frequently used venues.**

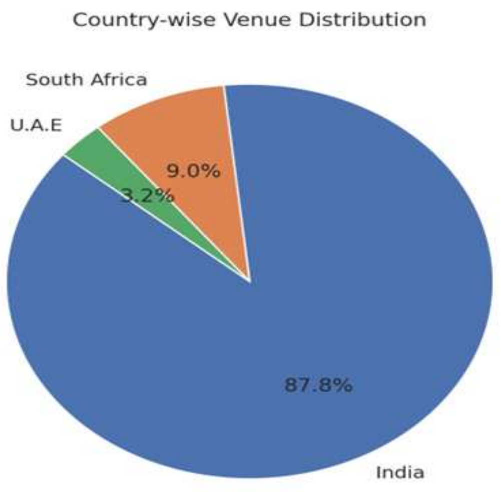
```
plt.figure(figsize=(12, 8))
venue_counts = data['Venue_Name'].value_counts().head(10)  # Top 10 venues for
readability
sns.barplot(x=venue_counts.values, y=venue_counts.index)
```

```
plt.title('Most Popular Venues by Number of Matches')
plt.xlabel('Number of Matches')
plt.ylabel('Venue')
plt.show()
```



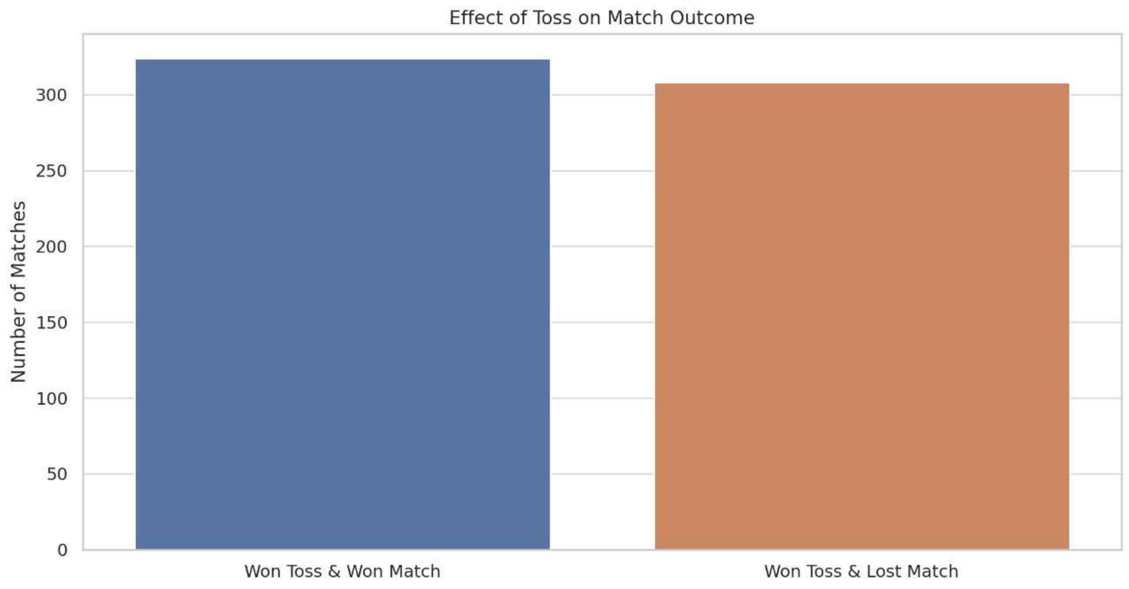
- **Country-wise Venue Distribution - Shows the distribution of venues by country.**

```
plt.figure(figsize=(10, 6))
country_counts = data['Country_Name'].value_counts()
country_counts.plot(kind='pie', autopct='%1.1f%%', startangle=140, legend=False)
plt.title('Country-wise Venue Distribution')
plt.ylabel('')
plt.show()
```



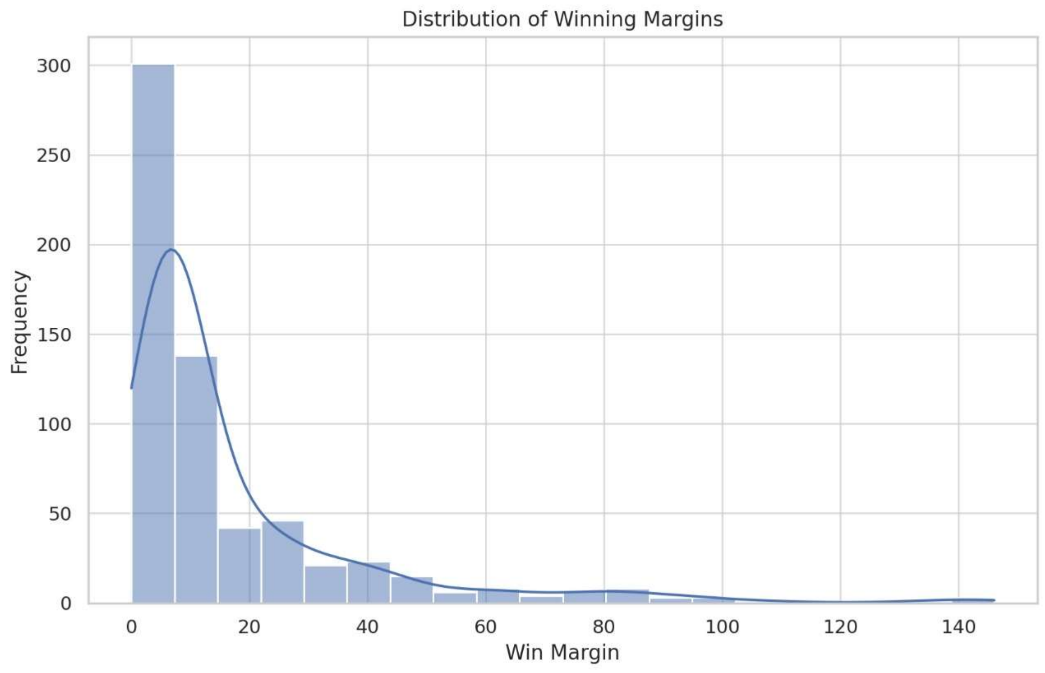
- **Toss Outcome vs. Match Outcome - Illustrates the impact of winning the toss on winning the match.**

```
plt.figure(figsize=(12, 6))
toss_match_relation = data[data['Toss_Winner'] ==
data['match_winner']]['Toss_Winner'].value_counts()
toss_relation_counts = [toss_match_relation.sum(), len(data) -
toss_match_relation.sum()]
sns.barplot(x=['Won Toss & Won Match', 'Won Toss & Lost Match'],
y=toss_relation_counts)
plt.title('Effect of Toss on Match Outcome')
plt.ylabel('Number of Matches')
plt.show()
```



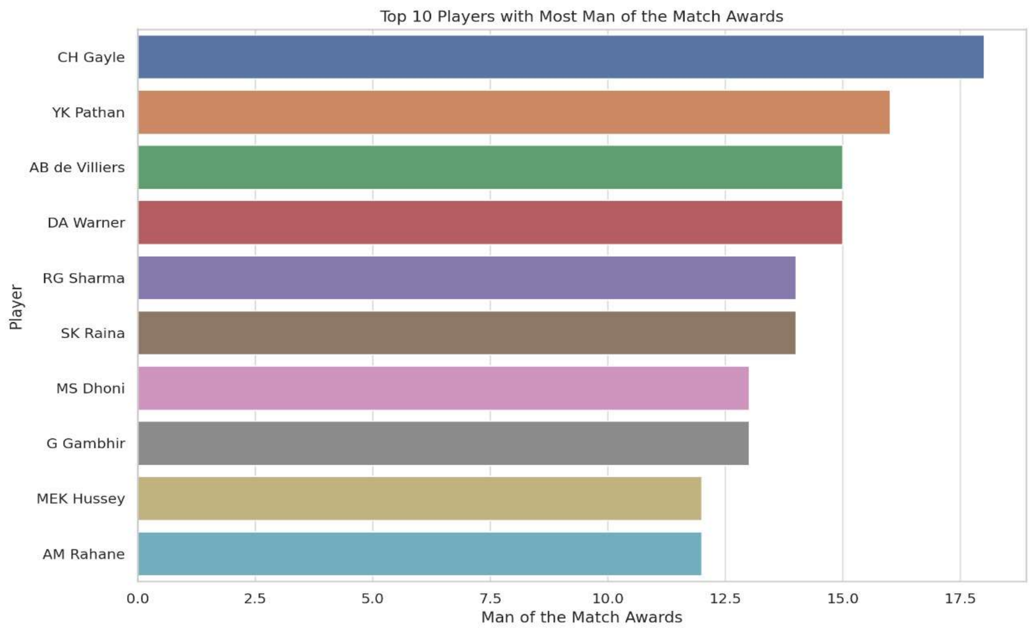
- **Winning Margins Distribution - Depicts how close or decisive games generally are.**

```
plt.figure(figsize=(10, 6))
sns.histplot(data['Win_Margin'], bins=20, kde=True)
plt.title('Distribution of Winning Margins')
plt.xlabel('Win Margin')
plt.ylabel('Frequency')
plt.show()
```



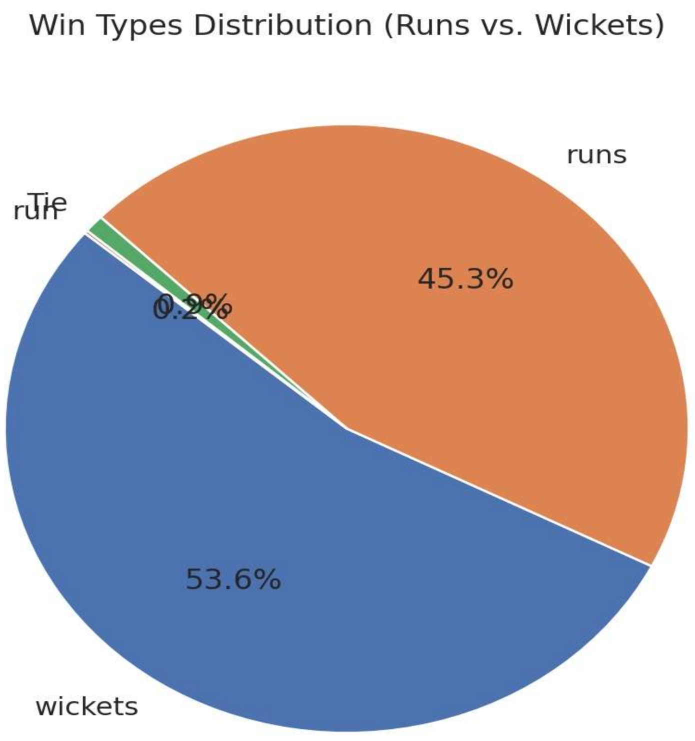
- **Man of the Match Leaders - Highlights the top 10 players with the most awards.**

```
plt.figure(figsize=(12, 8))
mom_counts = data['ManOfMach'].value_counts().head(10)  # Top 10 players for
readability
sns.barplot(x=mom_counts.values, y=mom_counts.index)
plt.title('Top 10 Players with Most Man of the Match Awards')
plt.xlabel('Man of the Match Awards')
plt.ylabel('Player')
plt.show()
```



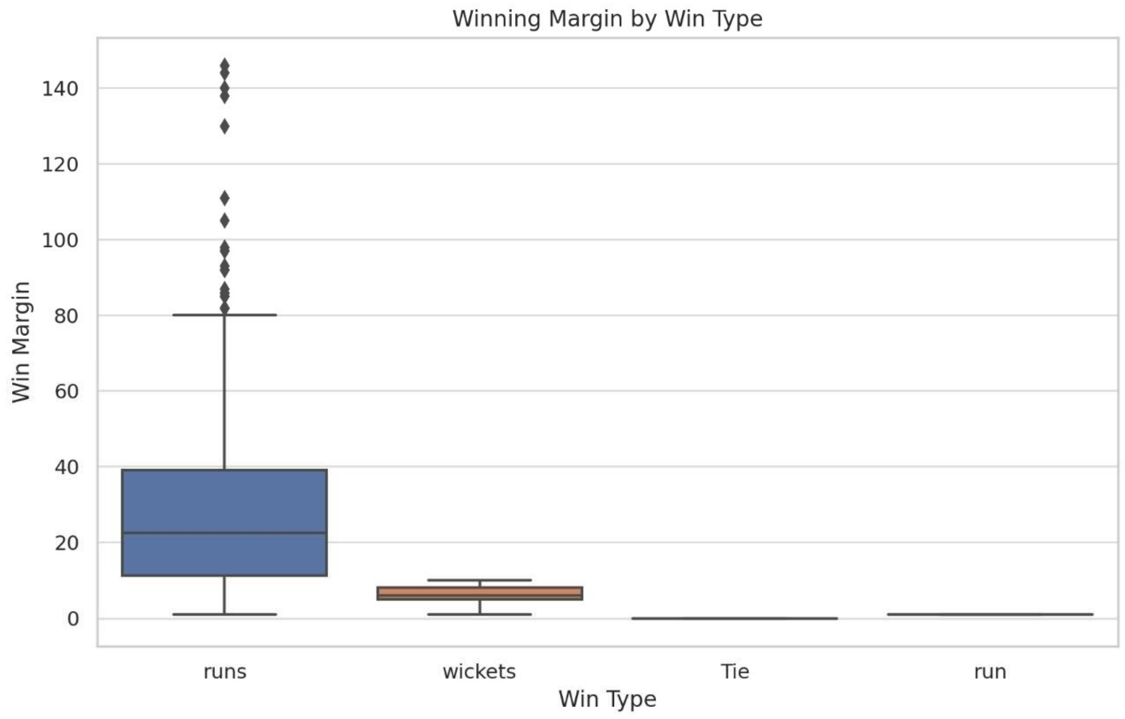
- **Win Types - Shows the proportion of wins by runs vs. wickets.**

```
plt.figure(figsize=(8, 6))
win_type_counts = data['Win_Type'].value_counts()
win_type_counts.plot(kind='pie', autopct='%1.1f%%', startangle=140)
plt.title('Win Types Distribution (Runs vs. Wickets)')
plt.ylabel('')
plt.show()
```



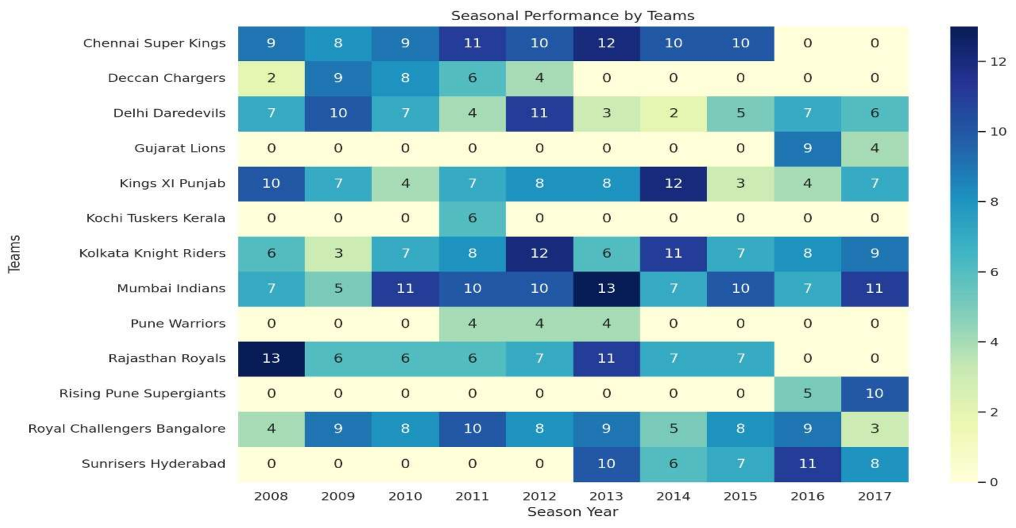
- **Winning Margins by Win Type - Compares margins for wins by runs and wickets.**

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='Win_Type', y='Win_Margin')
plt.title('Winning Margin by Win Type')
plt.xlabel('Win Type')
plt.ylabel('Win Margin')
plt.show()
```



- **Seasonal Performance by Teams - A heatmap of team performance across seasons.**

```
season_team_performance = data.pivot_table(index='match_winner',
columns='Season_Year', aggfunc='size', fill_value=0)
plt.figure(figsize=(12, 8))
sns.heatmap(season_team_performance, annot=True, fmt="d", cmap="YlGnBu",
cbar=True)
plt.title('Seasonal Performance by Teams')
plt.xlabel('Season Year')
plt.ylabel('Teams')
plt.show()
```



CHAPTER 04: CONCLUSIONS.

Analysing IPL data across these metrics reveals insightful trends into team dominance, game strategy, and fan engagement. As matches increase each season, franchises like Mumbai Indians and Chennai Super Kings stand out as consistent top performers. Venue popularity, with stadiums like Wankhede and Eden Gardens frequently in use, underscores fan hotspots and high-capacity locations. Although IPL is primarily based in India, occasional matches abroad have expanded its global reach. Toss outcomes show a strategic advantage, particularly in games where chasing leads to favourable results. Winning margins indicate a mix of close matches and decisive wins, adding to the league's thrill factor, while top players like AB de Villiers and Chris Gayle accumulate "Man of the Match" awards, emphasizing their influence on match outcomes. Analysing win types reveals balanced successes in defending and chasing, with larger margins typically seen in defence. The heatmap of seasonal performances captures the consistency of some franchises and the variability in others, highlighting the impact of strategic adjustments each season. Together, these trends reflect the IPL's blend of consistent team strength, strategic depth, and high-stakes individual contributions, making it one of the most competitive and popular leagues globally.

CHAPTER 05: FUTURE WORKS

Future work for this IPL data analysis project can focus on enhancing predictive modelling capabilities to forecast match outcomes based on historical performance metrics, player statistics, and real-time data. Implementing machine learning algorithms could improve accuracy in predicting winning teams, player performances, and match results. Additionally, incorporating advanced analytics such as player sentiment analysis from social media or fan engagement metrics could provide deeper insights into team dynamics and public perception. Exploring the impact of player injuries and trades on team performance would add another layer of analysis. Visualizations can be enhanced by integrating interactive dashboards, allowing users to explore data trends dynamically. Furthermore, analysing emerging players' performance trends can identify potential future stars in the league. Lastly, expanding the analysis to include off-field factors, such as marketing strategies and sponsorships, could enrich understanding of the IPL's overall impact and commercial success.

CHAPTER 06: REFERENCES.

[1]. Saranya, G., et al. "Ipl data analysis and visualization for team selection and profit strategy." *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2023.

[2]. Manju, M. K., Abin Oommen Philip, and M. U. Sreeja. "Exploratory data analysis of Indian Premier League (IPL)." *AIP Conference Proceedings*. Vol. 2773. No. 1. AIP Publishing, 2023.