
SUNRISERS
HYDERABAD
CHENNAI SUPER KINGS
BANGALORE
ROYAL CHALLENGERS
DELHI CAPITALS
TATA
IPL
RAJASTHAN ROYALS
GUJARAT TITANS
PUNJAB KINGS
KOLKATA
Knight Riders
MUMBAI INDIANS
LUCKNOW SUPER GIANTS

Tabular Data Visualization Interesting insights into Premier League cricket matches

Rohith Bandari

Overview

1. Introduction
2. Objectives
3. Dataset
4. Exploratory Data Analysis
    - Data Loading
    - Data Cleaning
    - Data Analysis using Descriptive Statistics
5. Data Manipulation
    - Data Aggregation
    - Time Series Technique
6. Data Visualization

Introduction

- The dataset contains information about Indian Premier League (IPL) matches from 2008 to 2019.
- IPL is a professional Twenty20 cricket league formed by the Board of Control for Cricket in India (BCCI).
- The league consists of eight clubs representing different states or cities in India.
- The dataset is in CSV format and has 756 rows and 18 columns.
- The rows signify games played, so there were 756 games played during the time period of the dataset.
- The columns provide information such as match ID, season, city, date, teams, toss winner and decision, match result, winner, margin of victory, outstanding player of the match, venue, and umpires.
- The dataset can be used for data visualization and cleaning to understand the statistics and trends of the IPL matches over the years.
- IPL is extremely popular, with the league's brand value estimated to reach $475 billion (US$6.7 billion) in 2019.
- The league consists of eight clubs, each representing one ofIndia's eight states or cities. It is extremely popular, with the IPL's brand value estimated to reach $475 billion (US$6.7 billion) in 2019. So, let's look at the IPL in terms of statistics.

Objectives – Finding Trends

- To determine Most successful teams.
- To determine Impact of toss decision.
- To determine Winning margin.
- To determine Player of the match.
- To determine Season-wise analysis.
- To determine Venue analysis and many more.

Dataset

- Source is Data world.
- The dataset is in CSV format
- It has 756 rows and 18 columns
- Rows signify games played - so 756 games played.
- https://data.world/coolboipranav/ipl-data/workspace/file?filename=matches.csv

In [257]:
```
import pandas as pd
df = pd.read_csv('C:/Users/rohit/Downloads/EDA Project Data/IPL.csv')
df
```

Out[257]:

	id	Season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	p
0	1	IPL-2017	Hyderabad	05-04-2017	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	
1	2	IPL-2017	Pune	06-04-2017	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	
2	3	IPL-2017	Rajkot	07-04-2017	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0	10	
3	4	IPL-2017	Indore	08-04-2017	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0	6	
4	5	IPL-2017	Bangalore	08-04-2017	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	0	Royal Challengers Bangalore	15	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
751	11347	IPL-2019	Mumbai	05-05-2019	Kolkata Knight Riders	Mumbai Indians	Mumbai Indians	field	normal	0	Mumbai Indians	0	9	
752	11412	IPL-2019	Chennai	07-05-2019	Chennai Super Kings	Mumbai Indians	Chennai Super Kings	bat	normal	0	Mumbai Indians	0	6	
753	11413	IPL-2019	Visakhapatnam	08-05-2019	Sunrisers Hyderabad	Delhi Capitals	Delhi Capitals	field	normal	0	Delhi Capitals	0	2	
754	11414	IPL-2019	Visakhapatnam	10-05-2019	Delhi Capitals	Chennai Super Kings	Chennai Super Kings	field	normal	0	Chennai Super Kings	0	6	
755	11415	IPL-2019	Hyderabad	12-05-2019	Mumbai Indians	Chennai Super Kings	Mumbai Indians	bat	normal	0	Mumbai Indians	1	0	

756 rows × 18 columns

Data Loading

- Importing the Pandas library, which reads the CSV file "IPL.csv"

IPL Dataset Information

- The dataset contains 756 entries with 18 columns
- Data types in the dataset include integers and objects
- Non-null values for each column range from 119 to 756
- 'id', 'Season', 'team1', 'team2', 'toss_winner', 'toss_decision', 'result', 'dl_applied', 'win_by_runs', and 'win_by_wickets' columns have 756 non-null values
- 'city', 'winner', 'player_of_match', 'venue', 'umpire1', and 'umpire2' columns have non-null values ranging from 749 to 754
- The 'umpire3' column has only 119 non-null values.

```
In [258]: df.info()
          # The columns in the dataset have non-null values ranging from 119 to 756.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               756 non-null    int64
 1   Season           756 non-null    object
 2   city             749 non-null    object
 3   date             756 non-null    object
 4   team1            756 non-null    object
 5   team2            756 non-null    object
 6   toss_winner      756 non-null    object
 7   toss_decision    756 non-null    object
 8   result           756 non-null    object
 9   dl_applied       756 non-null    int64
 10  winner           752 non-null    object
 11  win_by_runs      756 non-null    int64
 12  win_by_wickets   756 non-null    int64
 13  player_of_match  752 non-null    object
 14  venue            756 non-null    object
 15  umpire1          754 non-null    object
 16  umpire2          754 non-null    object
 17  umpire3          119 non-null    object
dtypes: int64(4), object(14)
memory usage: 106.4+ KB
```

IPL Dataset Column Description

- **id**: IPL match identification number
- **season**: The season of the IPL match
- **city**: The city where the IPL match was held
- **date**: The date on which the match was held
- **team1**: One of the teams participating in the IPL match
- **team2**: The other team participating in the IPL match
- **toss_winner**: The team that won the toss
- **toss_decision**: The decision taken by the team that won the toss to 'bat' or 'field'
- **result**: The result ('normal', 'tie', 'no result') of the match
- **dl_applied**: A binary indicator of whether the Duckworth-Lewis rule was applied (1) or not (0)
- **winner**: The winning team of the match
- **win_by_runs**: The runs by which the team batting first won
- **win_by_wickets**: The number of wickets by which the team batting second won
- **player_of_match**: The outstanding player of the match
- **venue**: The venue where the match was hosted
- **umpire1**: One of the two on-field umpires who officiate the match
- **umpire2**: One of the two on-field umpires who officiate the match
- **umpire3**: The off-field umpire who officiates the match

```
In [259]: df.columns

Out[259]: Index(['id', 'Season', 'city', 'date', 'team1', 'team2', 'toss_winner',
       'toss_decision', 'result', 'dl_applied', 'winner', 'win_by_runs',
       'win_by_wickets', 'player_of_match', 'venue', 'umpire1', 'umpire2',
       'umpire3'],
      dtype='object')
```

Data Cleaning

Displaying Unique Values of Each Column in a DataFrame

- using the iteritems() method to iterate through each column of a DataFrame, then utilizing the unique() method to extract the unique values present in each column.

```
In [426]: # Iterating over each column of a DataFrame and printing the unique values present in each column,
          # to understand the dataset better .
          for column_name, column_data in df.iteritems():
              unique_values = column_data.unique()
              print(unique_values)

[   1    2    3    4    5    6    7    8    9   10   11   12
   13   14   15   16   17   18   19   20   21   22   23   24
   25   26   27   28   29   30   31   32   33   34   35   36
   37   38   39   40   41   42   43   44   45   46   47   48
   49   50   51   52   53   54   55   56   57   58   59   60
   61   62   63   64   65   66   67   68   69   70   71   72
   73   74   75   76   77   78   79   80   81   82   83   84
   85   86   87   88   89   90   91   92   93   94   95   96
   97   98   99  100  101  102  103  104  105  106  107  108
  109  110  111  112  113  114  115  116  117  118  119  120
  121  122  123  124  125  126  127  128  129  130  131  132
  133  134  135  136  137  138  139  140  141  142  143  144
  145  146  147  148  149  150  151  152  153  154  155  156
  157  158  159  160  161  162  163  164  165  166  167  168
  169  170  171  172  173  174  175  176  177  178  179  180
  181  182  183  184  185  186  187  188  189  190  191  192
  193  194  195  196  197  198  199  200  201  202  203  204
```

Finding all the NaN Values in Dataset

- Identifying all the NaN values present in the dataset, enabling us to locate and assign appropriate values to nullify the NaN values.

```
In [427]: finding_NaN_df = df.loc[df.isna().sum(axis=1) > 0]
          finding_NaN_df.style.highlight_null(null_color='yellow')
```

ngs XI Punjab	Kings XI Punjab	field	normal	0	Deccan Chargers	82	0	S Dhawan	Pradesh Cricket Association Stadium	Asad Rauf	AM Saheba	nan
Pune arriors	Delhi Daredevils	bat	no result	0	nan	0	0	nan	Feroz Shah Kotla	SS Hazare	RJ Tucker	nan
Royal engers galore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	8	CH Gayle	M Chinnaswamy Stadium	K Hariharan	RE Koertzen	nan
umbai ndians	Mumbai Indians	field	normal	0	Mumbai Indians	0	5	JEC Franklin	Eden Gardens	SK Tarapore	SJA Taufel	nan
hennai Super Kings	Chennai Super Kings	field	normal	0	Chennai Super Kings	0	6	SK Raina	Wankhede Stadium	Asad Rauf	SJA Taufel	nan
umbai ndians	Mumbai Indians	field	normal	0	Mumbai Indians	0	4	MM Patel	Wankhede Stadium	Asad Rauf	SJA Taufel	nan

Counting the number of NaN cells in the DataFrame

- Here we can observe that there are total of 656 NaN cells in the dataset we need to clear those to and get it to 0 to complete the Data cleaning operation

```
In [428]: # counting the number of NaN cells in the DataFrame
          n_nans = df.isna().sum().sum()

          print(f'The number of NaN cells in the dataset is {n_nans}')

          The number of NaN cells in the dataset is 656
```

The Column 'Umpire 3' has more 'NaN' values in them, so lets remove it.

- After removing the 'Umpire 3' column from the Pandas DataFrame, we can observe the change in the DataFrame shape using the df.shape attribute. The shape has been updated from 756 rows and 18 columns to 756 rows and 17 columns, providing evidence that the 'Umpire 3' column has been successfully removed.

```
In [429]: # Before removing column Umpire 3
          df.shape

Out[429]: (756, 18)

In [430]: # Removing the column Umpire 3
          df = df.loc[:, df.columns != 'umpire3']

          # After removing the column Umpire 3
          df.shape

          # Here we can observe that number of columns has been changed to 17 from 18

Out[430]: (756, 17)

In [431]: df.head()
          # The column umpire3 has beed removed

Out[431]:
```

1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_of_match	venue	umpire1	umpire2
s d	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	Yuvraj Singh	Rajiv Gandhi International Stadium, Uppal	AY Dandekar	NJ Llong
i s	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	SPD Smith	Maharashtra Cricket Association Stadium	A Nand Kishore	S Ravi
t s	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0	10	CA Lynn	Saurashtra Cricket Association Stadium	Nitin Menon	CK Nandan

Handling of Missing Data

- The number of NaN values has decreased from 656 to 19.

In [432]:
```python
n_nans = df.isna().sum().sum()

print(f'The number of NaN cells in the dataset is {n_nans}')
finding_NaN_df = df.loc[df.isna().sum(axis=1) > 0]
finding_NaN_df.style.highlight_null(null_color='yellow')
```

The number of NaN cells in the dataset is 19

Out[432]:

	id	Season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	pl
4	5	IPL-2017	Bangalore	08-04-2017	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	0	Royal Challengers Bangalore	15	0	
300	301	IPL-2011	Delhi	21-05-2011	Delhi Daredevils	Pune Warriors	Delhi Daredevils	bat	no result	0	nan	0	0	
461	462	IPL-2014	nan	19-04-2014	Mumbai Indians	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	7	
462	463	IPL-2014	nan	19-04-2014	Kolkata Knight Riders	Delhi Daredevils	Kolkata Knight Riders	bat	normal	0	Delhi Daredevils	0	4	
466	467	IPL-2014	nan	23-04-2014	Chennai Super Kings	Rajasthan Royals	Rajasthan Royals	field	normal	0	Chennai Super Kings	7	0	
468	469	IPL-2014	nan	25-04-2014	Sunrisers Hyderabad	Delhi Daredevils	Sunrisers Hyderabad	bat	normal	0	Sunrisers Hyderabad	4	0	

Highlighting Specific Data in a Pandas DataFrame

- Setting the values of the 'city' column to 'Dubai' for specific row indices with the venue as Dubai International Cricket Stadium
- Defining a function called "highlight_data" that takes a DataFrame as an input and returns a copy of it with specific cells highlighted in pink.

In [433]:
```
# Setting the values of the 'city' column to 'Dubai' for the below row indices.
indices = [461, 462, 466, 468, 469, 474, 476]
df.loc[indices, 'city'] = "Dubai"
```

In [434]:
```
def highlight_data(df):
    # Creating a copy of the DataFrame with all cells set to white
    df_copy = df.copy().applymap(lambda x: 'background-color:white')

    # Setting the background color of specific cells to light green
    df_copy.loc[[461, 462, 466, 468, 469, 474, 476], 'city'] = 'background-color:pink'

    return df_copy

# Selecting the rows 461 through 480 of the DataFrame, applying the "highlight_data" function, and format the result.
df_styled = df.loc[461:480].style
df_styled.apply(highlight_data, axis=None)
```

Out[434]:

	id	Season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_o
461	462	IPL-2014	Dubai	19-04-2014	Mumbai Indians	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Royal Challengers Bangalore	0	7	
462	463	IPL-2014	Dubai	19-04-2014	Kolkata Knight Riders	Delhi Daredevils	Kolkata Knight Riders	bat	normal	0	Delhi Daredevils	0	4	JF
463	464	IPL-2014	Sharjah	20-04-2014	Rajasthan Royals	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0	7	GJ

Updating 'no result' Matches

- updating the 'winner' and 'player_of_match' columns in the dataframe 'df' for those rows where the 'result' column has the value 'no result'. It sets the value for both 'winner' and 'player_of_match' as 'No Result'. This is likely done to indicate that the match did not have a clear winner or standout player.

```
In [436]: df.loc[df['result'] == 'no result', ['winner', 'player_of_match']] = 'No Result'
          df
```

Out[436]:

	id	Season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	pl
0	1	IPL-2017	Hyderabad	05-04-2017	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35	0	
1	2	IPL-2017	Pune	06-04-2017	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0	7	
2	3	IPL-2017	Rajkot	07-04-2017	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0	10	
3	4	IPL-2017	Indore	08-04-2017	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0	6	
4	5	IPL-2017	Bangalore	08-04-2017	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	0	Royal Challengers Bangalore	15	0	

Counting the number of NaN cells in the Dataframe.

- At this point the NaN values has been reduced to 4. After all trials we found removing these two rows with Nan vales will be the perfect solution

```
In [404]: # counting the number of NaN cells in the DataFrame
          n_nans = df.isna().sum().sum()

          print(f'The number of NaN cells in the dataset is {n_nans}')

          The number of NaN cells in the dataset is 4
```

Data Cleaning

- Setting the 'id' column as the index of the DataFrame, dropping two rows with specific target IDs, and resetting the index to a sequential integer index.

```
In [405]: df.shape

Out[405]: (756, 17)
```

```
In [437]: df = df.set_index('id') # set 'id' column as the DataFrame index
          df = df.drop([5, 11413], axis=0) # drop rows with the target_id1 and target_id2 values from the DataFrame
          df = df.reset_index() # reset the index to a sequential integer index
```

```
In [438]: df.shape

Out[438]: (754, 17)
```

Finding all the NaN Values in Dataset

- Now that we have confirmed that our dataset has no null values, we can proceed to the next phase of our analysis after completing the data cleansing and preparation steps.

In [456]:
```
n_nans = df.isna().sum().sum()

print(f'The number of NaN cells in the dataset is {n_nans}')
finding_NaN_df = df.loc[df.isna().sum(axis=1) > 0]
finding_NaN_df.style.highlight_null(null_color='yellow')
```

```
The number of NaN cells in the dataset is 0
```

Out[456]:

id	Season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs	win_by_wickets	player_of_match	venue	umpire1

Data Analysis using Descriptive Statistics

1. Statistics of Win Margin in IPL Matches

Finding the statistics of a numerical column

- Describing the numerical column 'win_by_runs' using descriptive statistics to understand the average, variability, and range of win margins in IPL matches.
- Obervation - # We can see that the average win margin in IPL matches is 13.29 runs, with a standard deviation of 23.49 runs. The minimum win margin is 0 runs (indicating a tie or a win by wickets), while the maximum win margin is 146 runs.

```
In [476]: win_by_runs_stats = df['win_by_runs'].describe()
          print(win_by_runs_stats)

count    754.000000
mean      13.298408
std       23.497220
min        0.000000
25%        0.000000
50%        0.000000
75%       19.000000
max      146.000000
Name: win_by_runs, dtype: float64
```

2. Counting the number of matches played in each city

- Visualizing the distribution of IPL matches across different cities using a bar chart or a map.
- Observation - We can see that Mumbai has hosted the most number of IPL matches (101), followed by Kolkata (77) and Delhi (74).

```
In [477]: # Counting the number of matches played in each city
          city_counts = df['city'].value_counts()

          print(city_counts)
```

```
Mumbai           101
Kolkata           77
Delhi             74
Bangalore         65
Hyderabad         64
Chennai           57
Jaipur            47
Chandigarh        46
Pune              38
Durban            15
Bengaluru         14
Ahmedabad         12
Centurion         12
Visakhapatnam     12
```

3. Proportions of Wins for Each Team in the IPL Matches Dataset

- shows the proportion of wins for each team in the IPL matches dataset, with Mumbai Indians having the highest proportion of wins at 14.46% and Kochi Tuskers Kerala having the lowest proportion of wins at 0.8%.

In [521]:
```python
# Calculating the proportion of matches won by each team
team_wins = df['winner'].value_counts(normalize=True)
print(team_wins)
```

```
Mumbai Indians                 0.144562
Chennai Super Kings            0.132626
Kolkata Knight Riders          0.122016
Royal Challengers Bangalore    0.110080
Kings XI Punjab                0.108753
Rajasthan Royals               0.099469
Delhi Daredevils               0.088859
Sunrisers Hyderabad            0.076923
Deccan Chargers                0.038462
Gujarat Lions                  0.017241
Pune Warriors                  0.015915
Rising Pune Supergiant         0.013263
Delhi Capitals                 0.011936
Kochi Tuskers Kerala           0.007958
Rising Pune Supergiants        0.006631
No Result                      0.005305
Name: winner, dtype: float64
```

Summary statistics of IPL match data

- The total number of IPL matches played in the dataset is 754, with match ID ranging from 1 to 11415.
- The average value of the **dl_applied** variable is 0.025, which means that the Duckworth-Lewis rule was applied in only a small proportion of matches.
- The average margin of victory for the team batting first (**win_by_runs**) is 13 runs, with a maximum of 146 runs and a minimum of 0 runs.
- The average number of wickets remaining when the team batting second wins (**win_by_wickets**) is 3, with a maximum of 10 wickets and a minimum of 0 wickets.

In [522]:
```
df.describe()
```

Out[522]:

	id	dl_applied	win_by_runs	win_by_wickets
count	754.000000	754.000000	754.000000	754.000000
mean	1781.789125	0.025199	13.298408	3.356764
std	3450.683492	0.156833	23.497220	3.389898
min	1.000000	0.000000	0.000000	0.000000
25%	190.250000	0.000000	0.000000	0.000000
50%	378.500000	0.000000	0.000000	4.000000
75%	566.750000	0.000000	19.000000	6.000000
max	11415.000000	1.000000	146.000000	10.000000


DATA MANIPULATION

Using Data Aggregation Technique

1. Number of matches per season in IPL

- Grouping the data by season and counting the number of matches played in each season
- Observation: The number of matches played in each season is relatively consistent, with the exception of IPL 2011, which had the most number of matches played (73), and IPL 2013, which had the second-most number of matches played (76). The other seasons had between 57 to 60 matches played.

```
In [530]: matches_per_season = df.groupby('Season')['id'].count()
          print(matches_per_season)

Season
IPL-2008    58
IPL-2009    57
IPL-2010    60
IPL-2011    73
IPL-2012    74
IPL-2013    76
IPL-2014    60
IPL-2015    59
IPL-2016    60
IPL-2017    58
IPL-2018    60
```

2. Grouping data by player_of_match and calculating the count of matches won.

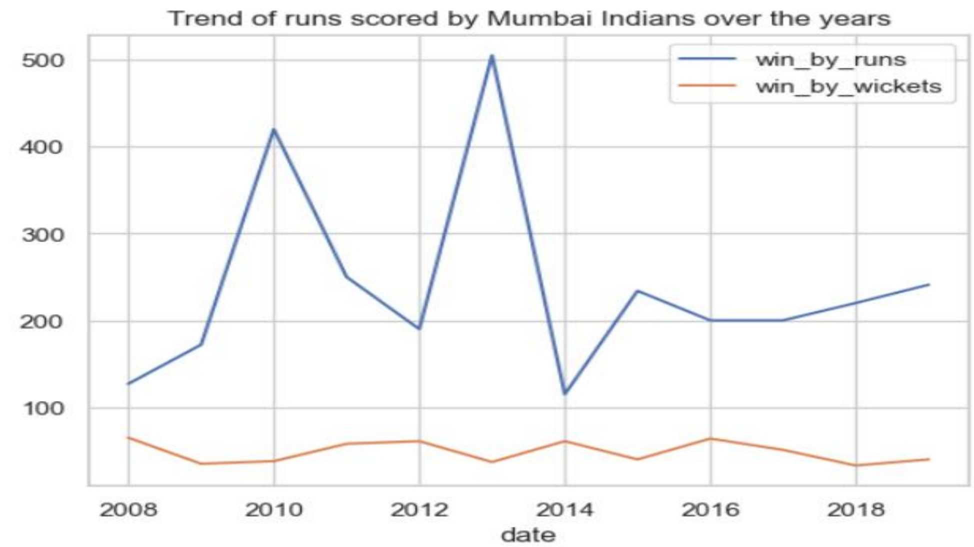
- Calculating the number of times each player has won the "player of the match" award in IPL matches.
- Observation: The output shows the top 10 players who have won the most "player of the match" awards. We can see that Chris Gayle has won the most awards, followed by AB de Villiers, Rohit Sharma, MS Dhoni, DA Warner, YK Pathan, SR Watson, SK Raina, G Gambhir, and AM Rahane.

```
In [555]: player_wins = df.groupby('player_of_match')['winner'].count().sort_values(ascending=False)
          print(player_wins.head(10))

player_of_match
CH Gayle          21
AB de Villiers    20
RG Sharma         17
MS Dhoni          17
DA Warner         17
YK Pathan         16
SR Watson         15
SK Raina          14
G Gambhir         13
AM Rahane         12
Name: winner, dtype: int64
```

Trend of runs scored by Mumbai Indians over the years in the IPL using time series techniques

- The line plot shows that Mumbai Indians have been consistently scoring more runs over the years with some fluctuations

```
In [556]: df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')
          df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')
          df.set_index('date', inplace=True)
          team = 'Mumbai Indians'
          df_team = df[(df['team1']==team) | (df['team2']==team)]
          df_yearly = df_team.groupby(df_team.index.year)[['win_by_runs', 'win_by_wickets']].sum()
          df_yearly.plot(kind='line', title='Trend of runs scored by Mumbai Indians over the years')

Out[556]: <AxesSubplot:title={'center':'Trend of runs scored by Mumbai Indians over the years'}, xlabel='date'>
```



Data Visualization

Data Visualization

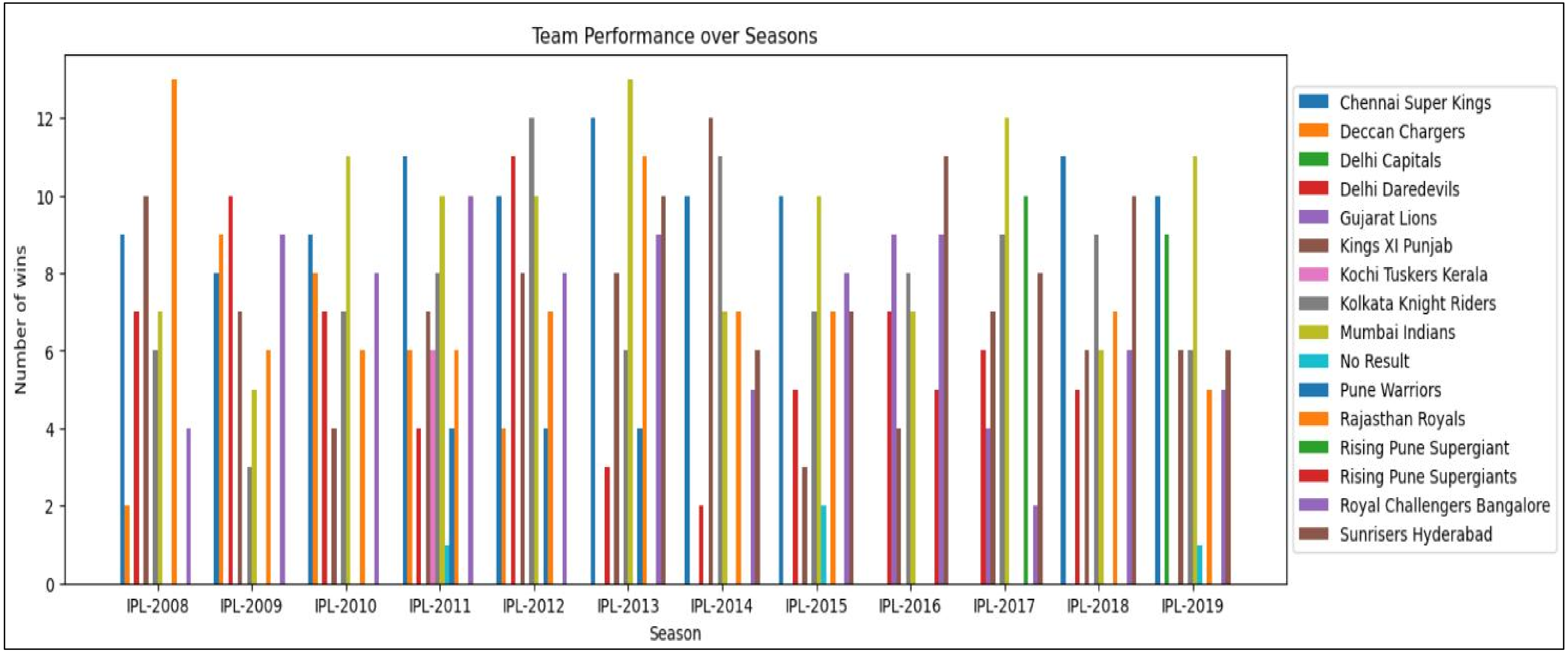
Data visualization is the representation of data in a visual format such as charts, graphs, and plots. In this context, we have used four different types of visualization techniques to represent data:

1. Clustered Bar Chart
2. Pie Chart
3. Stacked Bar Chart
4. Scatter Plot

Comparing the Win/Loss Record of Each Team in IPL Seasons

- Grouping the data by season and team to get the number of wins per team per season. Then, it creates a clustered bar chart to visualize the win/loss record of each team in IPL seasons.
- The visualization shows the number of wins per team per season in a clustered bar chart. We can observe the performance of each team in different seasons and compare their win/loss records.The bar chart also shows a gradual increase in the total number of matches played per season.
- the chart can be used to track the overall trend of the league's competitiveness over time. By observing the number of matches played per season, one can identify the growth of the league and how it has evolved over time.

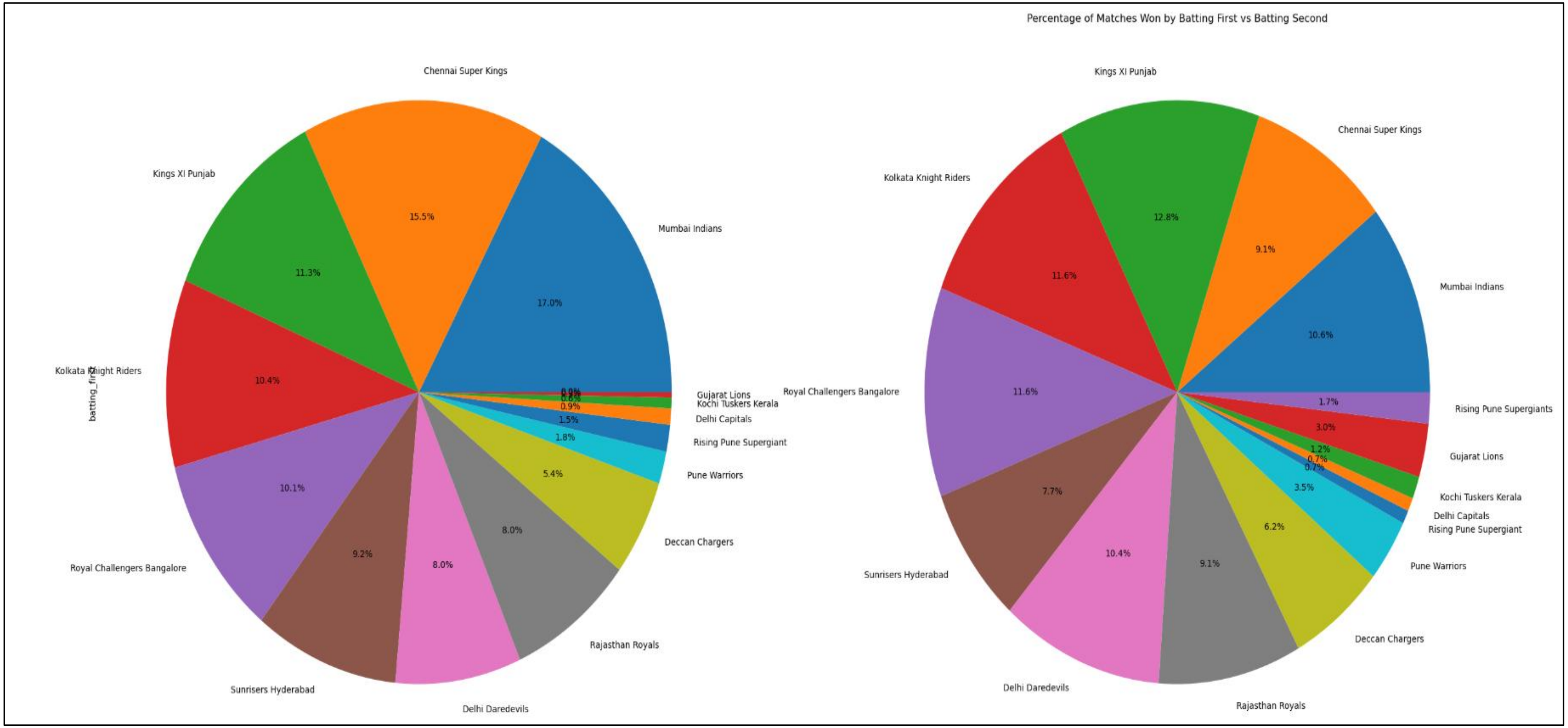
Clustered Bar Chart



Percentage of Matches Won by Batting First vs Batting Second

- A new column 'winning_team' is created to identify the team that won (either team1 or team2). The number of matches won by each team batting first and the number of matches won by each team batting second are counted. The counts are combined into a single dataframe. A pie chart is plotted to show the percentage of matches won by teams batting first vs teams batting second.
- The pie chart shows that in the Indian Premier League, teams batting second have a higher percentage of wins than teams batting first.

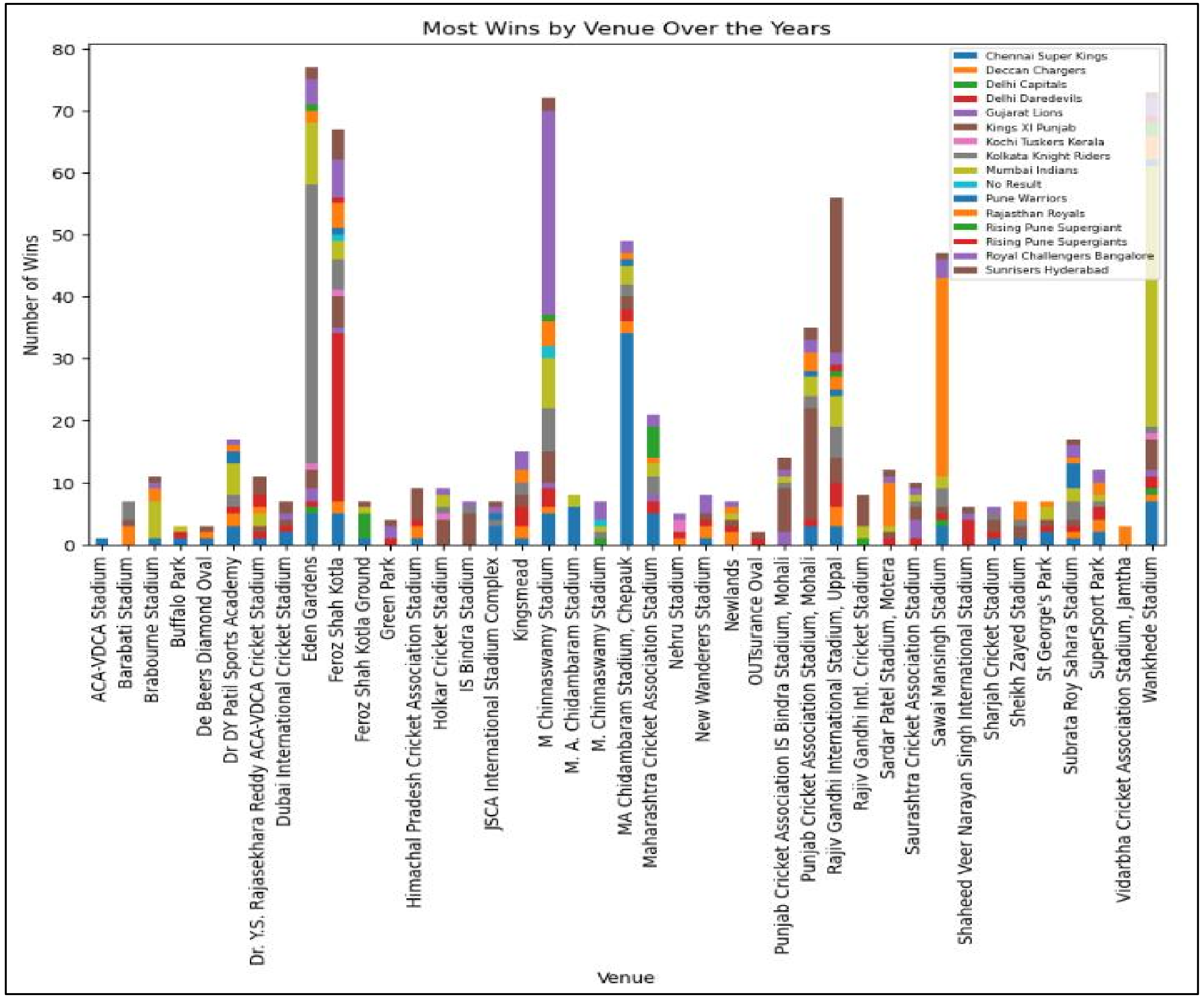
Pie Chart


Chennai Super Kings
Kings XI Punjab
Mumbai Indians
15.5%
11.3%
17.0%
Kolkata Knight Riders
batting_first
10.4%
Gujarat Lions
Kochi Tuskers Kerala
0.9%
Delhi Capitals
1.5%
Rising Pune Supergiant
1.8%
Pune Warriors
5.4%
10.1%
Deccan Chargers
8.0%
Royal Challengers Bangalore
9.2%
8.0%
Rajasthan Royals
Sunrisers Hyderabad
Delhi Daredevils
Percentage of Matches Won by Batting First vs Batting Second
Kings XI Punjab
Chennai Super Kings
Kolkata Knight Riders
12.8%
9.1%
11.6%
Mumbai Indians
10.6%
Royal Challengers Bangalore
11.6%
1.7%
Rising Pune Supergiants
3.0%
Gujarat Lions
1.2%
0.7%
0.7%
Kochi Tuskers Kerala
Delhi Capitals
Rising Pune Supergiant
7.7%
3.5%
6.2%
Pune Warriors
Sunrisers Hyderabad
10.4%
9.1%
Deccan Chargers
Delhi Daredevils
Rajasthan Royals

Most Wins by Venue Over the Years

- Creating a bar chart showing the number of wins for each team at each venue by pivoting the data to create a table of wins for each team at each venue
- we can observe which teams have the most wins at each venue over the years. We can also identify which venues have been most favorable for each team. The chart helps to visualize the data and identify patterns in team performance across different venues.

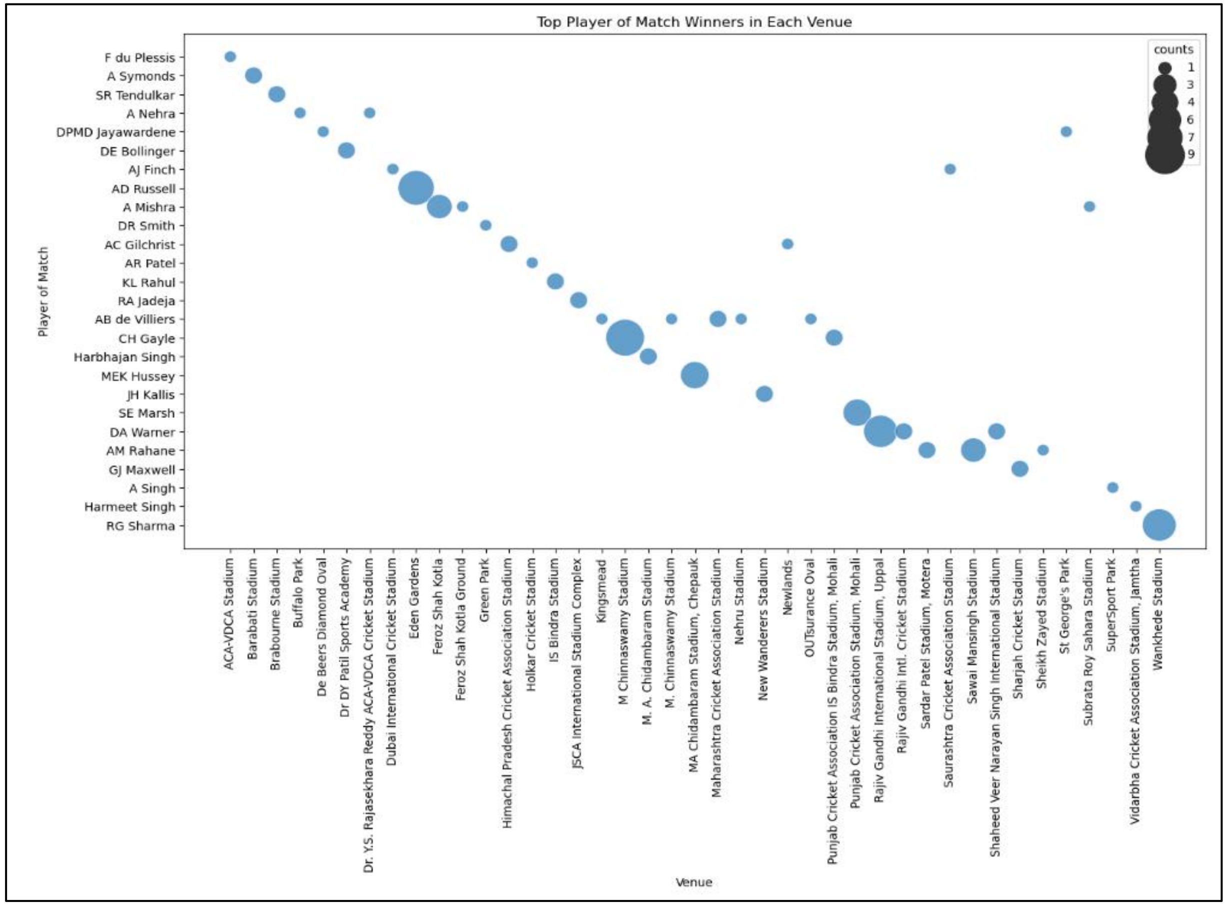
Stacked Bar Chart



Get the top player of match winner in each venue?

- The scatter plot reveals that Chris Gayle has won the player of the match award at the M Chinnaswamy Stadium in Bengaluru more times than any other player. Similarly, AB de Villiers has won the award at the same stadium on the second most occasions.

Scatter Plot



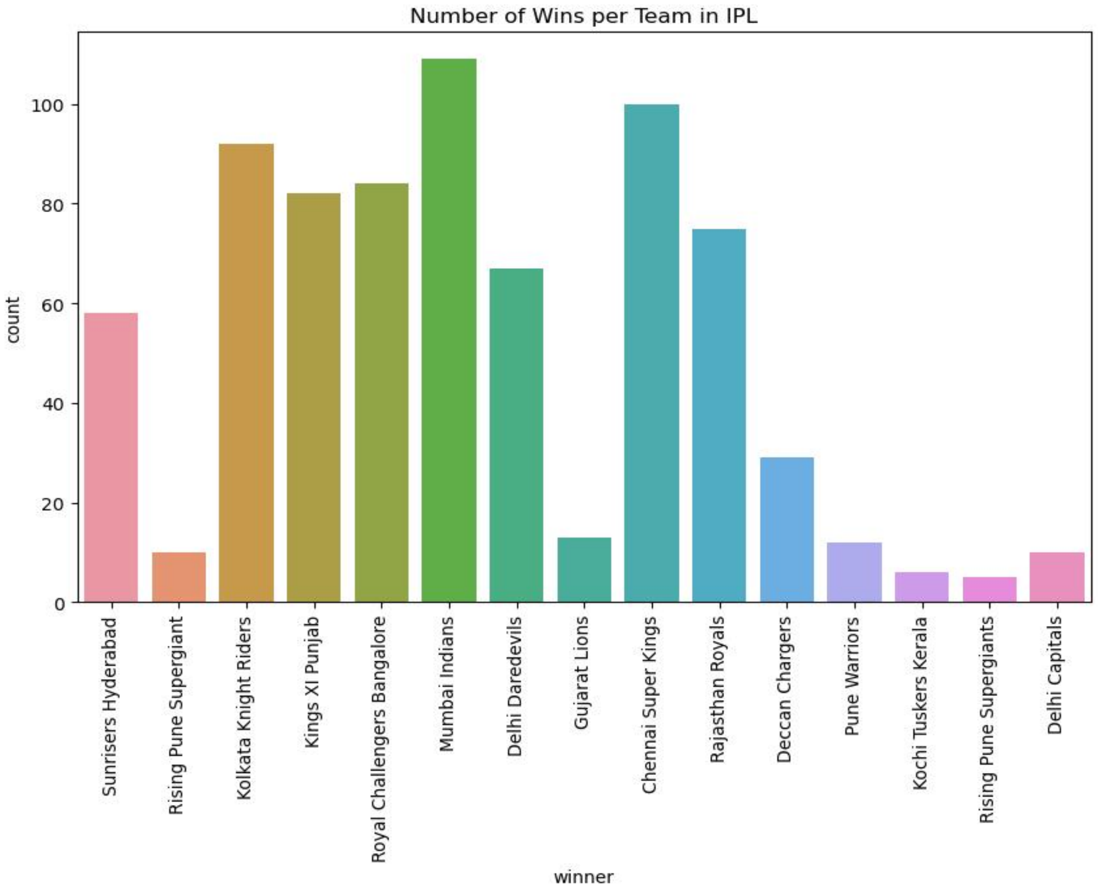
Q&A time

Which team is the most successful team in IPL?

- Mumbai Indians hold the record for winning the most games in the Indian Premier League (IPL). They have won the IPL trophy four times and have secured the most wins in four seasons. This makes Mumbai Indians the most successful team in the history of IPL.

```
# Count the number of wins for each team
wins_by_team = df['winner'].value_counts()

# Print the team with the most wins
print('The most successful team in IPL is', wins_by_team.index[0])
```

```
The most successful team in IPL is Mumbai Indians
```

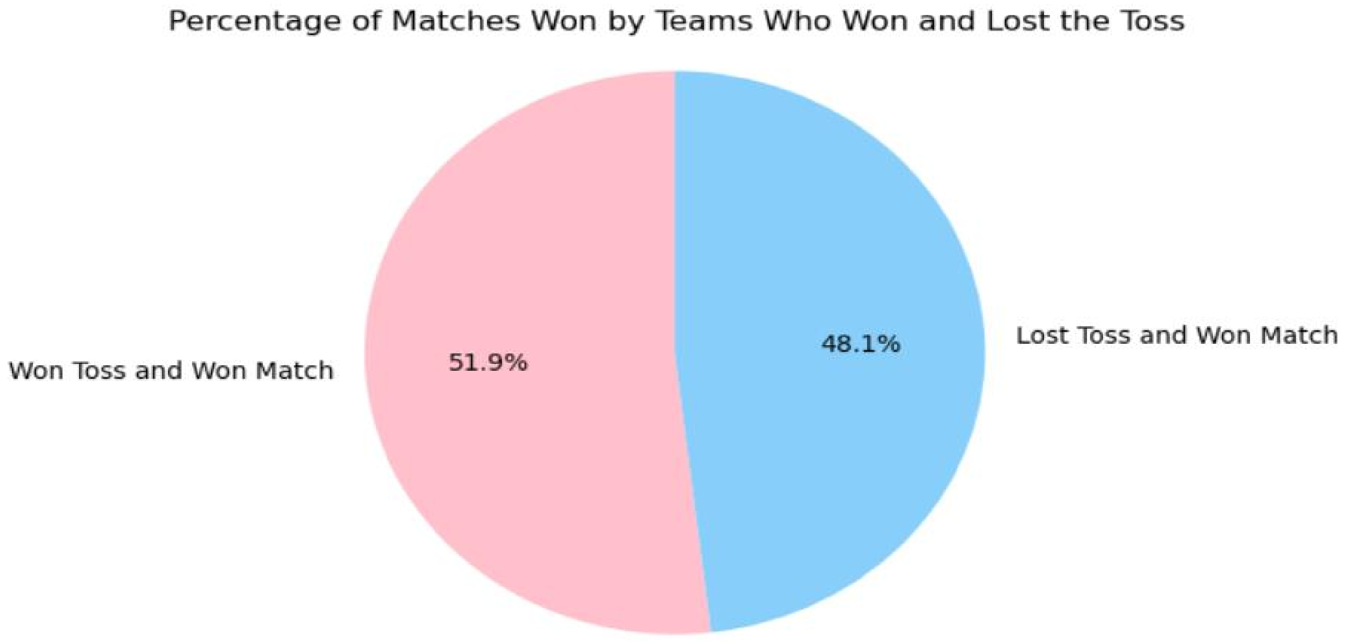


Impact of Winning the Coin Toss on Winning the Match in IPL

- 51% in coin toss wins are favored. It is pretty close, so it is not certain that coin toss decides the winner

```python
# creating a new column 'won_toss_and_won_match' to indicate if the team that won the toss also won the match
df['won_toss_and_won_match'] = (df['toss_winner'] == df['winner'])

# calculating the percentage of matches won by teams who won the toss
toss_win_match_win_pct = df[df['toss_winner']==df['winner']]['winner'].count() / df['winner'].count()

# calculating the percentage of matches won by teams who lost the toss
toss_loss_match_win_pct = df[df['toss_winner']!=df['winner']]['winner'].count() / df['winner'].count()
print("Percentage of Matches Won by Teams Who Won the Toss: ", toss_win_match_win_pct)
print("Percentage of Matches Won by Teams Who Lost the Toss: ", toss_loss_match_win_pct)

labels = ['Won Toss and Won Match', 'Lost Toss and Won Match']
sizes = [toss_win_match_win_pct, toss_loss_match_win_pct]
colors = ['pink', 'lightskyblue']
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
plt.axis('equal')
plt.title('Percentage of Matches Won by Teams Who Won and Lost the Toss')
plt.show()
```

```
Percentage of Matches Won by Teams Who Won the Toss:  0.5185676392572944
Percentage of Matches Won by Teams Who Lost the Toss:  0.48143236074270557
```



Inferences and Conclusion

Key findings from Exploratory Data Analysis:
756 IPL matches were played in 40 venues in 2019.
Winning the toss can’t decide winning of the match.
Mumbai Indians are the most successful IPL team with 4 winning seasons.
Eden Gardens hosted the most IPL matches.
Chris Gayle received the most player of the match awards.

References and Future work

- Indian Premier League official website - https://www.iplt20.com/
- "IPL Stats - IPLT20.com." IPLT20, 22 Mar. 2023, https://www.iplt20.com/stats.
- "IPL 2022: Full schedule, teams, venues, timings, live streaming, tickets, and all you need to know - Sports News." India Today, 22 Mar. 2023, https://www.indiatoday.in/sports/cricket/story/ipl-2022-full-schedule-teams-venues-timings-live-streaming-tickets-and-all-you-need-to-know-1944097-2022-03-11.


Thank you !