Gradient_Geeks_brain_dead_2k25_PS1

March 23, 2025

1 Statistics is All You Need: IPL Data Analysis and 2025 Winner Prediction – The Game Behind the Game!

1.1 Team :Gradient Geeks

1.1.1 Members:

1. Anurag Ghosh
2. Suchana Hazra
3. Siddharth Sen
4. Uttam Mahata

1.1.2 Problem Statement

Perform a comprehensive analysis of IPL data (2008-2024) to extract key insights and develop a predictive model for the 2025 IPL winner.

1.2 1. Data Collection & Preprocessing

- Load the datasets (`matches.csv`, `deliveries.csv`)
- Handle missing values and data inconsistencies
- Convert date columns to datetime format
- Standardize team names (e.g., Delhi Daredevils → Delhi Capitals)

1.3 2. Exploratory Data Analysis (EDA)

1.3.1 Team Performance Analysis

- Matches Played & Winning Percentages
- Run Rate & Economy Rate
- Highest & Lowest Scores

- Total 4s & 6s
- Powerplay & Death Overs Analysis

1.3.2 Player Performance Analysis

- Top 20 Run-Scorers
- Batting Average vs Strike Rate
- Highest Average & Strike Rate (min 50 matches)
- Top Wicket-Takers
- Highest Individual Scores
- Man of the Match Count
- K-Means Clustering: Batsman vs Bowler vs All-Rounder

1.3.3 Seasonal Analysis

- Average Runs per Match per Season
- Targets of 200+ Runs per Season
- Team-wise Average Scores per Season
- Orange & Purple Cap Holder Analysis
- Top 10 Bowlers per Season

1.4 3. Feature Engineering & Extraction

- Extract match-level and player-level statistics
- Create new features based on historical data

1.5 4. Winner Prediction Model (2025 IPL)

- Data preparation for model training
- Train an ensemble model (Random Forest, XGBoost)
- Experiment with Neural Networks

- Model Validation & Performance Evaluation
- 2025 IPL Winner Prediction

1.6 5. Results & Discussion

- Key Insights from EDA
- Strengths & Limitations of the Prediction Model
- Future Improvements

1.7 6. Tools & Libraries Used

- Pandas, NumPy for data manipulation
- Matplotlib, Seaborn for visualization
- Scikit-Learn, XGBoost for model building
- Google Colab for implementation

---

[1]:

```python
# import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:

```python
df_deliveries = pd.read_csv('deliveries.csv')
df_matches = pd.read_csv('matches.csv')
```

2 PART 1: IPL Dataset Analysis -EDA

2.1 Description of Datasets

The dataset contains details of **1095 IPL matches** played over the last **17 years**. It is divided into two files:

- `matches.csv` – Contains match-level information.
- `deliveries.csv` – Provides ball-by-ball details of every match.

2.2 Features

2.2.1 Data Field Description of `matches.csv`

This file records high-level match details, including teams, results, and umpires.

- `id`: Unique identifier for each match.
- `city`: City where the match was played.
- `date`: Date of the match.
- `player_of_match`: Player awarded "Player of the Match."
- `venue`: Stadium or venue of the match.
- `neutral_venue`: Binary indicator (0: Home/Away, 1: Neutral).
- `team1`: First participating team.
- `team2`: Second participating team.
- `toss_winner`: Team that won the toss.
- `toss_decision`: Decision of the toss-winning team (`field`/`bat`).
- `winner`: Team that won the match.
- `result`: Type of match result (`runs`, `wickets`, `tie`, etc.).
- `result_margin`: Margin by which the match was won (runs/wickets).
- `eliminator`: Binary (1: Eliminator match, 0: Regular match).
- `method`: Method used to decide the match (`Duckworth-Lewis`, etc.).
- `umpire1`: Name of the first on-field umpire.
- `umpire2`: Name of the second on-field umpire.

2.2.2 Data Field Description of `deliveries.csv`

This file provides ball-by-ball details of all IPL matches. It contains **14,26,312 deliveries** across different seasons.

- `match_id`: Unique match identifier.
- `inning`: Inning number of the match.

- `batting_team`: Name of the batting team.
- `bowling_team`: Name of the bowling team.
- `over`: Over number in the inning.
- `batter`: Batsman at the striker's end.
- `bowler`: Name of the bowler.
- `non_striker`: Batsman at the non-striker's end.
- `batsman_runs`: Runs scored by the batsman.
- `extra_runs`: Extra runs conceded.
- `total_runs`: Total runs in the ball (batsman + extras).
- `extra_type`: Type of extra (`wide`, `no-ball`, `bye`, etc.).
- `is_wicket`: 1 if a dismissal occurred, otherwise 0.
- `player_dismissal`: Name of the dismissed batsman.
- `dismissal_kind`: Type of dismissal (`bowled`, `caught`, `run-out`, etc.).
- `fielder`: Fielder involved in the dismissal.

---

[3]:
```
df_deliveries.head()
```

[3]:
```
   match_id  inning           batting_team                bowling_team  over  \
0    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore   0.0
1    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore   0.0
2    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore   0.0
3    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore   0.0
4    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore   0.0

   ball       batter   bowler  non_striker  batsman_runs  extra_runs  \
0   1.0   SC Ganguly  P Kumar  BB McCullum           0.0         1.0
1   2.0  BB McCullum  P Kumar   SC Ganguly           0.0         0.0
2   3.0  BB McCullum  P Kumar   SC Ganguly           0.0         1.0
3   4.0  BB McCullum  P Kumar   SC Ganguly           0.0         0.0
4   5.0  BB McCullum  P Kumar   SC Ganguly           0.0         0.0

   total_runs extras_type  is_wicket player_dismissed dismissal_kind fielder
```

```
     0         1.0     legbyes        0.0              NaN            NaN     NaN
     1         0.0         NaN        0.0              NaN            NaN     NaN
     2         1.0       wides        0.0              NaN            NaN     NaN
     3         0.0         NaN        0.0              NaN            NaN     NaN
     4         0.0         NaN        0.0              NaN            NaN     NaN
```

```
[4]: df_matches.head()
```

```
[4]:        id   season        city        date match_type player_of_match  \
     0  335982  2007/08   Bangalore  2008-04-18     League     BB McCullum
     1  335983  2007/08  Chandigarh  2008-04-19     League      MEK Hussey
     2  335984  2007/08       Delhi  2008-04-19     League     MF Maharoof
     3  335985  2007/08      Mumbai  2008-04-20     League      MV Boucher
     4  335986  2007/08     Kolkata  2008-04-20     League       DJ Hussey

                                             venue                        team1  \
     0                       M Chinnaswamy Stadium  Royal Challengers Bangalore
     1  Punjab Cricket Association Stadium, Mohali              Kings XI Punjab
     2                            Feroz Shah Kotla             Delhi Daredevils
     3                            Wankhede Stadium               Mumbai Indians
     4                                Eden Gardens        Kolkata Knight Riders

                              team2                  toss_winner toss_decision  \
     0        Kolkata Knight Riders  Royal Challengers Bangalore         field
     1          Chennai Super Kings          Chennai Super Kings           bat
     2             Rajasthan Royals             Rajasthan Royals           bat
     3  Royal Challengers Bangalore               Mumbai Indians           bat
     4              Deccan Chargers              Deccan Chargers           bat

                             winner   result  result_margin  target_runs  \
     0        Kolkata Knight Riders     runs          140.0        223.0
     1          Chennai Super Kings     runs           33.0        241.0
     2             Delhi Daredevils  wickets            9.0        130.0
     3  Royal Challengers Bangalore  wickets            5.0        166.0
     4        Kolkata Knight Riders  wickets            5.0        111.0

        target_overs super_over method    umpire1          umpire2
     0          20.0          N    NaN  Asad Rauf      RE Koertzen
     1          20.0          N    NaN  MR Benson       SL Shastri
     2          20.0          N    NaN  Aleem Dar   GA Pratapkumar
     3          20.0          N    NaN   SJ Davis        DJ Harper
     4          20.0          N    NaN  BF Bowden      K Hariharan
```

```
[5]: df_matches.shape
```

```
[5]: (1095, 20)
```

[6]:

```
df_matches.shape
```

[6]:

```
(1095, 20)
```

[7]:

```
df_matches.keys()
```

[7]:

```
Index(['id', 'season', 'city', 'date', 'match_type', 'player_of_match',
       'venue', 'team1', 'team2', 'toss_winner', 'toss_decision', 'winner',
       'result', 'result_margin', 'target_runs', 'target_overs', 'super_over',
       'method', 'umpire1', 'umpire2'],
      dtype='object')
```

[8]:

```
df_deliveries.keys()
```

[8]:

```
Index(['match_id', 'inning', 'batting_team', 'bowling_team', 'over', 'ball',
       'batter', 'bowler', 'non_striker', 'batsman_runs', 'extra_runs',
       'total_runs', 'extras_type', 'is_wicket', 'player_dismissed',
       'dismissal_kind', 'fielder'],
      dtype='object')
```

[9]:

```
df_matches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1095 entries, 0 to 1094
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               1095 non-null   int64  
 1   season           1095 non-null   object 
 2   city             1044 non-null   object 
 3   date             1095 non-null   object 
 4   match_type       1095 non-null   object 
 5   player_of_match  1090 non-null   object 
 6   venue            1095 non-null   object 
 7   team1            1095 non-null   object 
 8   team2            1095 non-null   object 
 9   toss_winner      1095 non-null   object 
 10  toss_decision    1095 non-null   object 
 11  winner           1090 non-null   object 
 12  result           1095 non-null   object 
 13  result_margin    1076 non-null   float64
 14  target_runs      1092 non-null   float64
 15  target_overs     1092 non-null   float64
 16  super_over       1095 non-null   object 
 17  method           21 non-null     object 
 18  umpire1          1095 non-null   object 
 19  umpire2          1095 non-null   object 
dtypes: float64(3), int64(1), object(16)
```

```
memory usage: 171.2+ KB
```

[10]:
```
#unique values
df_matches.nunique()
```

[10]:
```
id                 1095
season               17
city                 36
date                823
match_type            8
player_of_match     291
venue                58
team1                19
team2                19
toss_winner          19
toss_decision         2
winner               19
result                4
result_margin        98
target_runs         170
target_overs         15
super_over            2
method                1
umpire1              62
umpire2              62
dtype: int64
```

Listing unique value for each columns matches.csv take:

[11]:
```
#print each unique value for each columns
for col in df_matches.columns:
    print(col, df_matches[col].unique())
```

```
id [ 335982  335983  335984 ... 1426310 1426311 1426312]
season ['2007/08' '2009' '2009/10' '2011' '2012' '2013' '2014' '2015' '2016'
 '2017' '2018' '2019' '2020/21' '2021' '2022' '2023' '2024']
city ['Bangalore' 'Chandigarh' 'Delhi' 'Mumbai' 'Kolkata' 'Jaipur' 'Hyderabad'
 'Chennai' 'Cape Town' 'Port Elizabeth' 'Durban' 'Centurion' 'East London'
 'Johannesburg' 'Kimberley' 'Bloemfontein' 'Ahmedabad' 'Cuttack' 'Nagpur'
 'Dharamsala' 'Kochi' 'Indore' 'Visakhapatnam' 'Pune' 'Raipur' 'Ranchi'
 'Abu Dhabi' nan 'Rajkot' 'Kanpur' 'Bengaluru' 'Dubai' 'Sharjah'
 'Navi Mumbai' 'Lucknow' 'Guwahati' 'Mohali']
date ['2008-04-18' '2008-04-19' '2008-04-20' '2008-04-21' '2008-04-22'
 '2008-04-23' '2008-04-24' '2008-04-25' '2008-04-26' '2008-04-27'
 '2008-04-28' '2008-04-29' '2008-04-30' '2008-05-01' '2008-05-02'
 '2008-05-03' '2008-05-04' '2008-05-05' '2008-05-06' '2008-05-07'
 '2008-05-08' '2008-05-09' '2008-05-10' '2008-05-11' '2008-05-12'
 '2008-05-13' '2008-05-14' '2008-05-15' '2008-05-16' '2008-05-17'
 '2008-05-18' '2008-05-19' '2008-05-20' '2008-05-21' '2008-05-23'
```

```
'2008-05-24' '2008-05-25' '2008-05-26' '2008-05-27' '2008-05-28'
'2008-05-30' '2008-05-31' '2008-06-01' '2009-04-18' '2009-04-19'
'2009-04-20' '2009-04-21' '2009-04-22' '2009-04-23' '2009-04-24'
'2009-04-25' '2009-04-26' '2009-04-27' '2009-04-28' '2009-04-29'
'2009-04-30' '2009-05-01' '2009-05-02' '2009-05-03' '2009-05-04'
'2009-05-05' '2009-05-06' '2009-05-07' '2009-05-08' '2009-05-09'
'2009-05-10' '2009-05-11' '2009-05-12' '2009-05-13' '2009-05-14'
'2009-05-15' '2009-05-16' '2009-05-17' '2009-05-18' '2009-05-19'
'2009-05-20' '2009-05-21' '2009-05-22' '2009-05-23' '2009-05-24'
'2010-03-12' '2010-03-13' '2010-03-14' '2010-03-15' '2010-03-16'
'2010-03-17' '2010-03-18' '2010-03-19' '2010-03-20' '2010-03-21'
'2010-03-22' '2010-03-23' '2010-03-24' '2010-03-25' '2010-03-26'
'2010-03-27' '2010-03-28' '2010-03-29' '2010-03-30' '2010-03-31'
'2010-04-01' '2010-04-02' '2010-04-03' '2010-04-04' '2010-04-05'
'2010-04-06' '2010-04-07' '2010-04-08' '2010-04-09' '2010-04-10'
'2010-04-11' '2010-04-12' '2010-04-13' '2010-04-14' '2010-04-15'
'2010-04-16' '2010-04-17' '2010-04-18' '2010-04-19' '2010-04-21'
'2010-04-22' '2010-04-24' '2010-04-25' '2011-04-08' '2011-04-09'
'2011-04-10' '2011-04-11' '2011-04-12' '2011-04-13' '2011-04-14'
'2011-04-15' '2011-04-16' '2011-04-17' '2011-04-18' '2011-04-19'
'2011-04-20' '2011-04-21' '2011-04-22' '2011-04-23' '2011-04-24'
'2011-04-25' '2011-04-26' '2011-04-27' '2011-04-28' '2011-04-29'
'2011-04-30' '2011-05-01' '2011-05-02' '2011-05-03' '2011-05-04'
'2011-05-05' '2011-05-06' '2011-05-07' '2011-05-08' '2011-05-09'
'2011-05-10' '2011-05-11' '2011-05-12' '2011-05-13' '2011-05-14'
'2011-05-15' '2011-05-16' '2011-05-17' '2011-05-18' '2011-05-19'
'2011-05-20' '2011-05-21' '2011-05-22' '2011-05-24' '2011-05-25'
'2011-05-27' '2011-05-28' '2012-04-04' '2012-04-05' '2012-04-06'
'2012-04-07' '2012-04-08' '2012-04-09' '2012-04-10' '2012-04-11'
'2012-04-12' '2012-04-13' '2012-04-14' '2012-04-15' '2012-04-16'
'2012-04-17' '2012-04-18' '2012-04-19' '2012-04-20' '2012-04-21'
'2012-04-22' '2012-04-23' '2012-04-24' '2012-04-25' '2012-04-26'
'2012-04-27' '2012-04-28' '2012-04-29' '2012-04-30' '2012-05-01'
'2012-05-02' '2012-05-03' '2012-05-04' '2012-05-05' '2012-05-06'
'2012-05-07' '2012-05-08' '2012-05-09' '2012-05-10' '2012-05-11'
'2012-05-12' '2012-05-13' '2012-05-14' '2012-05-15' '2012-05-16'
'2012-05-17' '2012-05-18' '2012-05-19' '2012-05-20' '2012-05-22'
'2012-05-23' '2012-05-25' '2012-05-27' '2013-04-03' '2013-04-04'
'2013-04-05' '2013-04-06' '2013-04-07' '2013-04-08' '2013-04-09'
'2013-04-10' '2013-04-11' '2013-04-12' '2013-04-13' '2013-04-14'
'2013-04-15' '2013-04-16' '2013-04-17' '2013-04-18' '2013-04-19'
'2013-04-20' '2013-04-21' '2013-04-22' '2013-04-23' '2013-04-24'
'2013-04-25' '2013-04-26' '2013-04-27' '2013-04-28' '2013-04-29'
'2013-04-30' '2013-05-01' '2013-05-02' '2013-05-03' '2013-05-04'
'2013-05-05' '2013-05-06' '2013-05-07' '2013-05-08' '2013-05-09'
'2013-05-10' '2013-05-11' '2013-05-12' '2013-05-13' '2013-05-14'
'2013-05-15' '2013-05-16' '2013-05-17' '2013-05-18' '2013-05-19'
'2013-05-21' '2013-05-22' '2013-05-24' '2013-05-26' '2014-04-16'
```

```
'2014-04-17' '2014-04-18' '2014-04-19' '2014-04-20' '2014-04-21'
'2014-04-22' '2014-04-23' '2014-04-24' '2014-04-25' '2014-04-26'
'2014-04-27' '2014-04-28' '2014-04-29' '2014-04-30' '2014-05-02'
'2014-05-03' '2014-05-04' '2014-05-05' '2014-05-06' '2014-05-07'
'2014-05-08' '2014-05-09' '2014-05-10' '2014-05-11' '2014-05-12'
'2014-05-13' '2014-05-14' '2014-05-15' '2014-05-18' '2014-05-19'
'2014-05-20' '2014-05-21' '2014-05-22' '2014-05-23' '2014-05-24'
'2014-05-25' '2014-05-27' '2014-05-28' '2014-05-30' '2014-06-01'
'2015-04-08' '2015-04-09' '2015-04-10' '2015-04-11' '2015-04-12'
'2015-04-13' '2015-04-14' '2015-04-15' '2015-04-16' '2015-04-17'
'2015-04-18' '2015-04-19' '2015-04-20' '2015-04-21' '2015-04-22'
'2015-04-23' '2015-04-24' '2015-04-25' '2015-04-26' '2015-04-27'
'2015-04-28' '2015-04-29' '2015-04-30' '2015-05-01' '2015-05-02'
'2015-05-03' '2015-05-04' '2015-05-05' '2015-05-06' '2015-05-07'
'2015-05-08' '2015-05-09' '2015-05-10' '2015-05-11' '2015-05-12'
'2015-05-13' '2015-05-14' '2015-05-15' '2015-05-16' '2015-05-17'
'2015-05-19' '2015-05-20' '2015-05-22' '2015-05-24' '2016-04-09'
'2016-04-10' '2016-04-11' '2016-04-12' '2016-04-13' '2016-04-14'
'2016-04-15' '2016-04-16' '2016-04-17' '2016-04-18' '2016-04-19'
'2016-04-20' '2016-04-21' '2016-04-22' '2016-04-23' '2016-04-24'
'2016-04-25' '2016-04-26' '2016-04-27' '2016-04-28' '2016-04-29'
'2016-04-30' '2016-05-01' '2016-05-02' '2016-05-03' '2016-05-04'
'2016-05-05' '2016-05-06' '2016-05-07' '2016-05-08' '2016-05-09'
'2016-05-10' '2016-05-11' '2016-05-12' '2016-05-13' '2016-05-14'
'2016-05-15' '2016-05-16' '2016-05-17' '2016-05-18' '2016-05-19'
'2016-05-20' '2016-05-21' '2016-05-22' '2016-05-24' '2016-05-25'
'2016-05-27' '2016-05-29' '2017-04-05' '2017-04-06' '2017-04-07'
'2017-04-08' '2017-04-09' '2017-04-10' '2017-04-11' '2017-04-12'
'2017-04-13' '2017-04-14' '2017-04-15' '2017-04-16' '2017-04-17'
'2017-04-18' '2017-04-19' '2017-04-20' '2017-04-21' '2017-04-22'
'2017-04-23' '2017-04-24' '2017-04-26' '2017-04-27' '2017-04-28'
'2017-04-29' '2017-04-30' '2017-05-01' '2017-05-02' '2017-05-03'
'2017-05-04' '2017-05-05' '2017-05-06' '2017-05-07' '2017-05-08'
'2017-05-09' '2017-05-10' '2017-05-11' '2017-05-12' '2017-05-13'
'2017-05-14' '2017-05-16' '2017-05-17' '2017-05-19' '2017-05-21'
'2018-04-07' '2018-04-08' '2018-04-09' '2018-04-10' '2018-04-11'
'2018-04-12' '2018-04-13' '2018-04-14' '2018-04-15' '2018-04-16'
'2018-04-17' '2018-04-18' '2018-04-19' '2018-04-20' '2018-04-21'
'2018-04-22' '2018-04-23' '2018-04-24' '2018-04-25' '2018-04-26'
'2018-04-27' '2018-04-28' '2018-04-29' '2018-04-30' '2018-05-01'
'2018-05-02' '2018-05-03' '2018-05-04' '2018-05-05' '2018-05-06'
'2018-05-07' '2018-05-08' '2018-05-09' '2018-05-10' '2018-05-11'
'2018-05-12' '2018-05-13' '2018-05-14' '2018-05-15' '2018-05-16'
'2018-05-17' '2018-05-18' '2018-05-19' '2018-05-20' '2018-05-22'
'2018-05-23' '2018-05-25' '2018-05-27' '2019-03-23' '2019-03-24'
'2019-03-25' '2019-03-26' '2019-03-27' '2019-03-28' '2019-03-29'
'2019-03-30' '2019-03-31' '2019-04-01' '2019-04-02' '2019-04-03'
'2019-04-04' '2019-04-05' '2019-04-06' '2019-04-07' '2019-04-08'
```

```
'2019-04-09' '2019-04-10' '2019-04-11' '2019-04-12' '2019-04-13'
'2019-04-14' '2019-04-15' '2019-04-16' '2019-04-17' '2019-04-18'
'2019-04-19' '2019-04-20' '2019-04-21' '2019-04-22' '2019-04-23'
'2019-04-24' '2019-04-25' '2019-04-26' '2019-04-27' '2019-04-28'
'2019-04-29' '2019-04-30' '2019-05-01' '2019-05-02' '2019-05-03'
'2019-05-04' '2019-05-05' '2019-05-07' '2019-05-08' '2019-05-10'
'2019-05-12' '2020-09-19' '2020-09-20' '2020-09-21' '2020-09-22'
'2020-09-23' '2020-09-24' '2020-09-25' '2020-09-26' '2020-09-27'
'2020-09-28' '2020-09-29' '2020-09-30' '2020-10-01' '2020-10-02'
'2020-10-03' '2020-10-04' '2020-10-05' '2020-10-06' '2020-10-07'
'2020-10-08' '2020-10-09' '2020-10-10' '2020-10-11' '2020-10-12'
'2020-10-13' '2020-10-14' '2020-10-15' '2020-10-16' '2020-10-17'
'2020-10-18' '2020-10-19' '2020-10-20' '2020-10-21' '2020-10-22'
'2020-10-23' '2020-10-24' '2020-10-25' '2020-10-26' '2020-10-27'
'2020-10-28' '2020-10-29' '2020-10-30' '2020-10-31' '2020-11-01'
'2020-11-02' '2020-11-03' '2020-11-05' '2020-11-06' '2020-11-08'
'2020-11-10' '2021-04-09' '2021-04-10' '2021-04-11' '2021-04-12'
'2021-04-13' '2021-04-14' '2021-04-15' '2021-04-16' '2021-04-17'
'2021-04-18' '2021-04-19' '2021-04-20' '2021-04-21' '2021-04-22'
'2021-04-23' '2021-04-24' '2021-04-25' '2021-04-26' '2021-04-27'
'2021-04-28' '2021-04-29' '2021-04-30' '2021-05-01' '2021-05-02'
'2021-09-19' '2021-09-20' '2021-09-21' '2021-09-22' '2021-09-23'
'2021-09-24' '2021-09-25' '2021-09-26' '2021-09-27' '2021-09-28'
'2021-09-29' '2021-09-30' '2021-10-01' '2021-10-02' '2021-10-03'
'2021-10-04' '2021-10-05' '2021-10-06' '2021-10-07' '2021-10-08'
'2021-10-10' '2021-10-11' '2021-10-13' '2021-10-15' '2022-03-26'
'2022-03-27' '2022-03-28' '2022-03-29' '2022-03-30' '2022-03-31'
'2022-04-01' '2022-04-02' '2022-04-03' '2022-04-04' '2022-04-05'
'2022-04-06' '2022-04-07' '2022-04-08' '2022-04-09' '2022-04-10'
'2022-04-11' '2022-04-12' '2022-04-13' '2022-04-14' '2022-04-15'
'2022-04-16' '2022-04-17' '2022-04-18' '2022-04-19' '2022-04-20'
'2022-04-21' '2022-04-22' '2022-04-23' '2022-04-24' '2022-04-25'
'2022-04-26' '2022-04-27' '2022-04-28' '2022-04-29' '2022-04-30'
'2022-05-01' '2022-05-02' '2022-05-03' '2022-05-04' '2022-05-05'
'2022-05-06' '2022-05-07' '2022-05-08' '2022-05-09' '2022-05-10'
'2022-05-11' '2022-05-12' '2022-05-13' '2022-05-14' '2022-05-15'
'2022-05-16' '2022-05-17' '2022-05-18' '2022-05-19' '2022-05-20'
'2022-05-21' '2022-05-22' '2022-05-24' '2022-05-25' '2022-05-27'
'2022-05-29' '2023-03-31' '2023-04-01' '2023-04-02' '2023-04-03'
'2023-04-04' '2023-04-05' '2023-04-06' '2023-04-07' '2023-04-08'
'2023-04-09' '2023-04-10' '2023-04-11' '2023-04-12' '2023-04-13'
'2023-04-14' '2023-04-15' '2023-04-16' '2023-04-17' '2023-04-18'
'2023-04-19' '2023-04-20' '2023-04-21' '2023-04-22' '2023-04-23'
'2023-04-24' '2023-04-25' '2023-04-26' '2023-04-27' '2023-04-28'
'2023-04-29' '2023-04-30' '2023-05-01' '2023-05-02' '2023-05-03'
'2023-05-04' '2023-05-05' '2023-05-06' '2023-05-07' '2023-05-08'
'2023-05-09' '2023-05-10' '2023-05-11' '2023-05-12' '2023-05-13'
'2023-05-14' '2023-05-15' '2023-05-16' '2023-05-17' '2023-05-18'
```

```
 '2023-05-19' '2023-05-20' '2023-05-21' '2023-05-23' '2023-05-24'
 '2023-05-26' '2023-05-29' '2024-03-22' '2024-03-23' '2024-03-24'
 '2024-03-25' '2024-03-26' '2024-03-27' '2024-03-28' '2024-03-29'
 '2024-03-30' '2024-03-31' '2024-04-01' '2024-04-02' '2024-04-03'
 '2024-04-04' '2024-04-05' '2024-04-06' '2024-04-07' '2024-04-08'
 '2024-04-09' '2024-04-10' '2024-04-11' '2024-04-12' '2024-04-13'
 '2024-04-14' '2024-04-15' '2024-04-16' '2024-04-17' '2024-04-18'
 '2024-04-19' '2024-04-20' '2024-04-21' '2024-04-22' '2024-04-23'
 '2024-04-24' '2024-04-25' '2024-04-26' '2024-04-27' '2024-04-28'
 '2024-04-29' '2024-04-30' '2024-05-01' '2024-05-02' '2024-05-03'
 '2024-05-04' '2024-05-05' '2024-05-06' '2024-05-07' '2024-05-08'
 '2024-05-09' '2024-05-10' '2024-05-11' '2024-05-12' '2024-05-14'
 '2024-05-15' '2024-05-17' '2024-05-18' '2024-05-19' '2024-05-21'
 '2024-05-22' '2024-05-24' '2024-05-26']
match_type ['League' 'Semi Final' 'Final' '3rd Place Play-Off' 'Qualifier 1'
 'Elimination Final' 'Qualifier 2' 'Eliminator']
player_of_match ['BB McCullum' 'MEK Hussey' 'MF Maharoof' 'MV Boucher' 'DJ
Hussey'
 'SR Watson' 'V Sehwag' 'ML Hayden' 'YK Pathan' 'KC Sangakkara' 'JDP Oram'
 'AC Gilchrist' 'SM Katich' 'MS Dhoni' 'ST Jayasuriya' 'GD McGrath'
 'SE Marsh' 'SA Asnodkar' 'IK Pathan' 'P Kumar' 'SM Pollock'
 'Sohail Tanvir' 'S Sreesanth' 'A Nehra' 'SC Ganguly' 'L Balaji'
 'Shoaib Akhtar' 'A Mishra' 'DPMD Jayawardene' 'GC Smith' 'DJ Bravo'
 'M Ntini' 'SP Goswami' 'A Kumble' 'KD Karthik' 'JA Morkel'
 'R Vinay Kumar' 'Umar Gul' 'SK Raina' 'CRD Fernando' 'SR Tendulkar'
 'R Dravid' 'DL Vettori' 'RP Singh' 'M Muralitharan' 'CH Gayle'
 'AB de Villiers' 'RS Bopara' 'PP Ojha' 'TM Dilshan' 'HH Gibbs'
 'DP Nannes' 'JP Duminy' 'Yuvraj Singh' 'SB Jakati' 'JH Kallis'
 'G Gambhir' 'RG Sharma' 'A Singh' 'S Badrinath' 'DR Smith' 'LRPL Taylor'
 'Harbhajan Singh' 'R Bhatia' 'SK Warne' 'B Lee' 'BJ Hodge' 'LR Shukla'
 'MK Pandey' 'AD Mathews' 'MK Tiwary' 'WPUJC Vaas' 'A Symonds'
 'AA Jhunjhunwala' 'J Theron' 'RV Uthappa' 'AC Voges' 'KM Jadhav'
 'NV Ojha' 'DA Warner' 'SL Malinga' 'M Vijay' 'KP Pietersen' 'AT Rayudu'
 'PD Collingwood' 'MJ Lumb' 'TL Suman' 'RJ Harris' 'PP Chawla'
 'Harmeet Singh' 'KA Pollard' 'R Ashwin' 'R McLaren' 'JD Unadkat'
 'M Kartik' 'DE Bollinger' 'S Anirudha' 'SK Trivedi' 'SB Wagh'
 'PC Valthaty' 'MD Mishra' 'DW Steyn' 'S Sohal' 'MM Patel' 'V Kohli'
 'I Sharma' 'J Botha' 'Iqbal Abdulla' 'P Parameswaran' 'R Sharma'
 'MR Marsh' 'BA Bhatt' 'S Aravind' 'WP Saha' 'S Dhawan' nan 'JEC Franklin'
 'RE Levi' 'SPD Smith' 'AM Rahane' 'RA Jadeja' 'MN Samuels' 'M Morkel'
 'F du Plessis' 'AD Mascarenhas' 'Shakib Al Hasan' 'JD Ryder' 'SP Narine'
 'S Nadeem' 'KMDN Kulasekara' 'CL White' 'Mandeep Singh' 'P Negi'
 'Azhar Mahmood' 'BW Hilfenhaus' 'A Chandila' 'UT Yadav' 'MS Bisla'
 'M Vohra' 'GH Vihari' 'AJ Finch' 'JP Faulkner' 'MS Gony' 'DA Miller'
 'SV Samson' 'DJG Sammy' 'MG Johnson' 'KK Cooper' 'PA Patel' 'AP Tare'
 'LJ Wright' 'YS Chahal' 'GJ Maxwell' 'CA Lynn' 'MM Sharma' 'PV Tambe'
 'Sandeep Sharma' 'B Kumar' 'CJ Anderson' 'KK Nair' 'AR Patel'
 'LMP Simmons' 'DJ Hooda' 'GJ Bailey' 'MA Agarwal' 'AD Russell' 'SS Iyer'
 'MA Starc' 'VR Aaron' 'TA Boult' 'NM Coulter-Nile' 'EJG Morgan'
 'HH Pandya' 'MC Henriques' 'Z Khan' 'MJ McClenaghan' 'Q de Kock'
 'Mustafizur Rahman' 'SA Yadav' 'AB Dinda' 'CH Morris' 'CR Brathwaite'
 'RR Pant' 'MP Stoinis' 'A Zampa' 'KH Pandya' 'HM Amla' 'BCJ Cutting'
 'Rashid Khan' 'N Rana' 'JJ Bumrah' 'AJ Tye' 'BA Stokes' 'KS Williamson'
 'JC Buttler' 'LH Ferguson' 'Mohammed Shami' 'RA Tripathi'
 'Mohammed Siraj' 'HV Patel' 'Washington Sundar' 'KV Sharma' 'KL Rahul'
 'SW Billings' 'JJ Roy' 'B Stanlake' 'JC Archer' 'AS Rajpoot' 'TG Southee'
 'Mujeeb Ur Rahman' 'Ishan Kishan' 'Kuldeep Yadav' 'S Gopal' 'L Ngidi'
 'PP Shaw' 'JM Bairstow' 'SM Curran' 'AS Joseph' 'K Rabada' 'HF Gurney'
 'DL Chahar' 'Imran Tahir' 'KMA Paul' 'KK Ahmed' 'Shubman Gill'
 'SO Hetmyer' 'Shivam Mavi' 'PK Garg' 'R Tewatia' 'A Nortje' 'CV Varun'
 'CJ Jordan' 'RD Gaikwad' 'PJ Cummins' 'RD Chahar' 'MM Ali' 'D Padikkal'
 'Harpreet Brar' 'Kartik Tyagi' 'JO Holder' 'JR Hazlewood' 'KS Bharat'
 'VR Iyer' 'OF Smith' 'PWH de Silva' 'E Lewis' 'LS Livingstone'
 'Avesh Khan' 'Abhishek Sharma' 'Anuj Rawat' 'S Dube' 'Umran Malik'
 'Mukesh Choudhary' 'M Jansen' 'R Parag' 'Mohsin Khan' 'RK Singh'
 'TH David' 'YBK Jaiswal' 'DP Conway' 'DR Sams' 'SN Thakur' 'RM Patidar'
 'Arshdeep Singh' 'MA Wood' 'B Sai Sudharsan' 'NT Ellis' 'N Pooran'
 'HC Brook' 'Sikandar Raza' 'C Green' 'A Manohar' 'J Little' 'M Pathirana'
 'PD Salt' 'GD Phillips' 'PN Mankad' 'P Simran Singh' 'WD Parnell'
 'RR Rossouw' 'Akash Madhwal' 'MP Yadav' 'Shashank Singh' 'R Shepherd'
 'Yash Thakur' 'Nithish Kumar Reddy' 'TM Head' 'R Sai Kishore'
 'J Fraser-McGurk' 'WG Jacks' 'Simarjeet Singh' 'Shahbaz Ahmed']
venue ['M Chinnaswamy Stadium' 'Punjab Cricket Association Stadium, Mohali'
 'Feroz Shah Kotla' 'Wankhede Stadium' 'Eden Gardens'
 'Sawai Mansingh Stadium' 'Rajiv Gandhi International Stadium, Uppal'
 'MA Chidambaram Stadium, Chepauk' 'Dr DY Patil Sports Academy' 'Newlands'
 "St George's Park" 'Kingsmead' 'SuperSport Park' 'Buffalo Park'
 'New Wanderers Stadium' 'De Beers Diamond Oval' 'OUTsurance Oval'
 'Brabourne Stadium' 'Sardar Patel Stadium, Motera' 'Barabati Stadium'
 'Brabourne Stadium, Mumbai'
 'Vidarbha Cricket Association Stadium, Jamtha'
 'Himachal Pradesh Cricket Association Stadium' 'Nehru Stadium'
 'Holkar Cricket Stadium'
 'Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium'
 'Subrata Roy Sahara Stadium' 'Maharashtra Cricket Association Stadium'
 'Shaheed Veer Narayan Singh International Stadium'
 'JSCA International Stadium Complex' 'Sheikh Zayed Stadium'
 'Sharjah Cricket Stadium' 'Dubai International Cricket Stadium'
 'Punjab Cricket Association IS Bindra Stadium, Mohali'
 'Saurashtra Cricket Association Stadium' 'Green Park'
 'M.Chinnaswamy Stadium' 'Punjab Cricket Association IS Bindra Stadium'
 'Rajiv Gandhi International Stadium' 'MA Chidambaram Stadium'
 'Arun Jaitley Stadium' 'MA Chidambaram Stadium, Chepauk, Chennai'
 'Wankhede Stadium, Mumbai' 'Narendra Modi Stadium, Ahmedabad'
 'Arun Jaitley Stadium, Delhi' 'Zayed Cricket Stadium, Abu Dhabi'
 'Dr DY Patil Sports Academy, Mumbai'
 'Maharashtra Cricket Association Stadium, Pune' 'Eden Gardens, Kolkata'
 'Punjab Cricket Association IS Bindra Stadium, Mohali, Chandigarh'
 'Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow'
 'Rajiv Gandhi International Stadium, Uppal, Hyderabad'
 'M Chinnaswamy Stadium, Bengaluru' 'Barsapara Cricket Stadium, Guwahati'
 'Sawai Mansingh Stadium, Jaipur'
 'Himachal Pradesh Cricket Association Stadium, Dharamsala'
 'Maharaja Yadavindra Singh International Cricket Stadium, Mullanpur'
 'Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam']
team1 ['Royal Challengers Bangalore' 'Kings XI Punjab' 'Delhi Daredevils'
 'Mumbai Indians' 'Kolkata Knight Riders' 'Rajasthan Royals'
 'Deccan Chargers' 'Chennai Super Kings' 'Kochi Tuskers Kerala'
 'Pune Warriors' 'Sunrisers Hyderabad' 'Gujarat Lions'
 'Rising Pune Supergiants' 'Rising Pune Supergiant' 'Delhi Capitals'
 'Punjab Kings' 'Lucknow Super Giants' 'Gujarat Titans'
 'Royal Challengers Bengaluru']
team2 ['Kolkata Knight Riders' 'Chennai Super Kings' 'Rajasthan Royals'
 'Royal Challengers Bangalore' 'Deccan Chargers' 'Kings XI Punjab'
 'Delhi Daredevils' 'Mumbai Indians' 'Kochi Tuskers Kerala'
 'Pune Warriors' 'Sunrisers Hyderabad' 'Rising Pune Supergiants'
 'Gujarat Lions' 'Rising Pune Supergiant' 'Delhi Capitals' 'Punjab Kings'
 'Gujarat Titans' 'Lucknow Super Giants' 'Royal Challengers Bengaluru']
toss_winner ['Royal Challengers Bangalore' 'Chennai Super Kings' 'Rajasthan
Royals'
 'Mumbai Indians' 'Deccan Chargers' 'Kings XI Punjab'
 'Kolkata Knight Riders' 'Delhi Daredevils' 'Kochi Tuskers Kerala'
 'Pune Warriors' 'Sunrisers Hyderabad' 'Gujarat Lions'
 'Rising Pune Supergiants' 'Rising Pune Supergiant' 'Delhi Capitals'
 'Punjab Kings' 'Gujarat Titans' 'Lucknow Super Giants'
 'Royal Challengers Bengaluru']
toss_decision ['field' 'bat']
winner ['Kolkata Knight Riders' 'Chennai Super Kings' 'Delhi Daredevils'
 'Royal Challengers Bangalore' 'Rajasthan Royals' 'Kings XI Punjab'
 'Deccan Chargers' 'Mumbai Indians' 'Pune Warriors' 'Kochi Tuskers Kerala'
 nan 'Sunrisers Hyderabad' 'Rising Pune Supergiants' 'Gujarat Lions'
 'Rising Pune Supergiant' 'Delhi Capitals' 'Punjab Kings' 'Gujarat Titans'
 'Lucknow Super Giants' 'Royal Challengers Bengaluru']
result ['runs' 'wickets' 'tie' 'no result']
result_margin [140.  33.   9.   5.   6.   3.  66.   7.  10.   4.  13.  45.   8.
29.
  18.  23.  12.  65.  25.   1.  14.  41. 105.  19.  75.  92.  11.  24.
  nan  27.  38.  78.  16.  53.   2.  31.  55.  98.  34.  36.  17.  39.
  40.  67.  63.  37.  57.  35.  22.  21.  48.  26.  20.  85.  32.  76.
 111.  82.  43.  58.  28.  74.  42.  59.  46.  47.  86.  44.  87. 130.
  15.  60.  77.  30.  50.  93.  72.  62.  97. 138.  71. 144.  80.  51.
  61. 146.  64. 102. 118.  49.  69.  88.  54.  91.  52.  81.  56. 112.
 106.]
target_runs [223. 241. 130. 166. 111. 167. 143. 209. 215. 183. 136. 148. 155.
```

```
159.
 179. 138. 192. 165. 197. 170. 157. 163. 110. 127. 145. 104. 188. 141.
 182. 205. 144. 134. 195.  68.  89. 198.  53. 190. 176. 177. 212. 175.
 146. 123. 222. 193. 113. 164.  54. 102. 180.  69. 185. 151. 169. 150.
 140. 120. 149. 142. 154. 106. 187. 117. 158. 124. 174. 121. 161. 135.
 189. 171. 147. 162. 213. 191. 204. 219.  93. 186. 152. 172. 137. 156.
 184. 181. 178. 173. 247. 201. 160. 139. 112. 131. 133.  83.  96.  82.
 119. 196. 232.  95. 206.  52. 126.  98. 233. 153. 199.  nan 129. 194.
 116. 132. 125. 208. 101. 115. 216. 100. 210. 118. 264.  81. 224. 107.
  71.  43. 202. 227. 200. 128. 168. 236. 114. 203. 122.  99. 228.  61.
 249.  66. 207.  58. 214. 231.  74.  48. 108. 218. 220. 211. 246. 109.
  63. 217. 229.  85. 221.  91. 258. 234. 278. 273. 235. 288.  90. 267.
 225. 262. 242.]
target_overs [20.  16.   8.  18.   6.   9.2 17.  10.  13.   nan 12.   5.  11.
9.
 14.  15. ]
super_over ['N' 'Y']
method [nan 'D/L']
umpire1 ['Asad Rauf' 'MR Benson' 'Aleem Dar' 'SJ Davis' 'BF Bowden' 'IL Howell'
 'DJ Harper' 'RE Koertzen' 'BR Doctrove' 'AV Jayaprakash' 'BG Jerling'
 'M Erasmus' 'HDPK Dharmasena' 'S Asnani' 'GAV Baxter' 'SS Hazare'
 'K Hariharan' 'SL Shastri' 'SK Tarapore' 'S Ravi' 'SJA Taufel' 'S Das'
 'AM Saheba' 'PR Reiffel' 'JD Cloete' 'AK Chaudhary' 'VA Kulkarni'
 'BNJ Oxenford' 'CK Nandan' 'C Shamshuddin' 'NJ Llong' 'RK Illingworth'
 'RM Deshpande' 'K Srinath' 'SD Fry' 'CB Gaffaney' 'PG Pathak'
 'Nitin Menon' 'K Bharatan' 'AY Dandekar' 'KN Ananthapadmanabhan'
 'A Nand Kishore' 'A Deshmukh' 'YC Barde' 'IJ Gould' 'RJ Tucker'
 'VK Sharma' 'UV Gandhe' 'K Srinivasan' 'J Madanagopal' 'Navdeep Singh'
 'MA Gough' 'Tapan Sharma' 'Chirra Ravikanthreddy' 'GR Sadashiv Iyer'
 'NA Patwardhan' 'HAS Khalid' 'R Pandit' 'A Totre' 'Vinod Seshan'
 'AG Wharf' 'MV Saidharshan Kumar']
umpire2 ['RE Koertzen' 'SL Shastri' 'GA Pratapkumar' 'DJ Harper' 'K Hariharan'
 'RB Tiffin' 'AM Saheba' 'MR Benson' 'IL Howell' 'AV Jayaprakash'
 'I Shivram' 'BR Doctrove' 'BG Jerling' 'SJ Davis' 'SD Ranade'
 'SJA Taufel' 'M Erasmus' 'TH Wijewardene' 'SK Tarapore' 'S Ravi'
 'HDPK Dharmasena' 'SS Hazare' 'BF Bowden' 'PR Reiffel' 'AL Hill'
 'RJ Tucker' 'VA Kulkarni' 'JD Cloete' 'BNJ Oxenford' 'S Asnani' 'S Das'
 'C Shamshuddin' 'AK Chaudhary' 'K Srinath' 'Subroto Das' 'CK Nandan'
 'NJ Llong' 'RK Illingworth' 'PG Pathak' 'CB Gaffaney' 'K Srinivasan'
 'SD Fry' 'VK Sharma' 'A Nand Kishore' 'Nitin Menon' 'A Deshmukh'
 'YC Barde' 'KN Ananthapadmanabhan' 'UV Gandhe' 'IJ Gould' 'AY Dandekar'
 'MA Gough' 'Tapan Sharma' 'Navdeep Singh' 'HAS Khalid' 'J Madanagopal'
 'N Pandit' 'R Pandit' 'NA Patwardhan' 'GR Sadashiv Iyer'
 'MV Saidharshan Kumar' 'Vinod Seshan']
```

Checking the Missing Values:

[12]:

```
df_matches.isnull().sum()
```

[12]:
```
id                   0
season               0
city                51
date                 0
match_type           0
player_of_match      5
venue                0
team1                0
team2                0
toss_winner          0
toss_decision        0
winner               5
result               0
result_margin       19
target_runs          3
target_overs         3
super_over           0
method            1074
umpire1              0
umpire2              0
dtype: int64
```

[13]:
```
df_matches.describe()
```

[13]:
```
                 id  result_margin  target_runs  target_overs
count  1.095000e+03    1076.000000  1092.000000   1092.000000
mean   9.048283e+05      17.259294   165.684066     19.759341
std    3.677402e+05      21.787444    33.427048      1.581108
min    3.359820e+05       1.000000    43.000000      5.000000
25%    5.483315e+05       6.000000   146.000000     20.000000
50%    9.809610e+05       8.000000   166.000000     20.000000
75%    1.254062e+06      20.000000   187.000000     20.000000
max    1.426312e+06     146.000000   288.000000     20.000000
```

Separate categorical and numerical columns in matches.csv

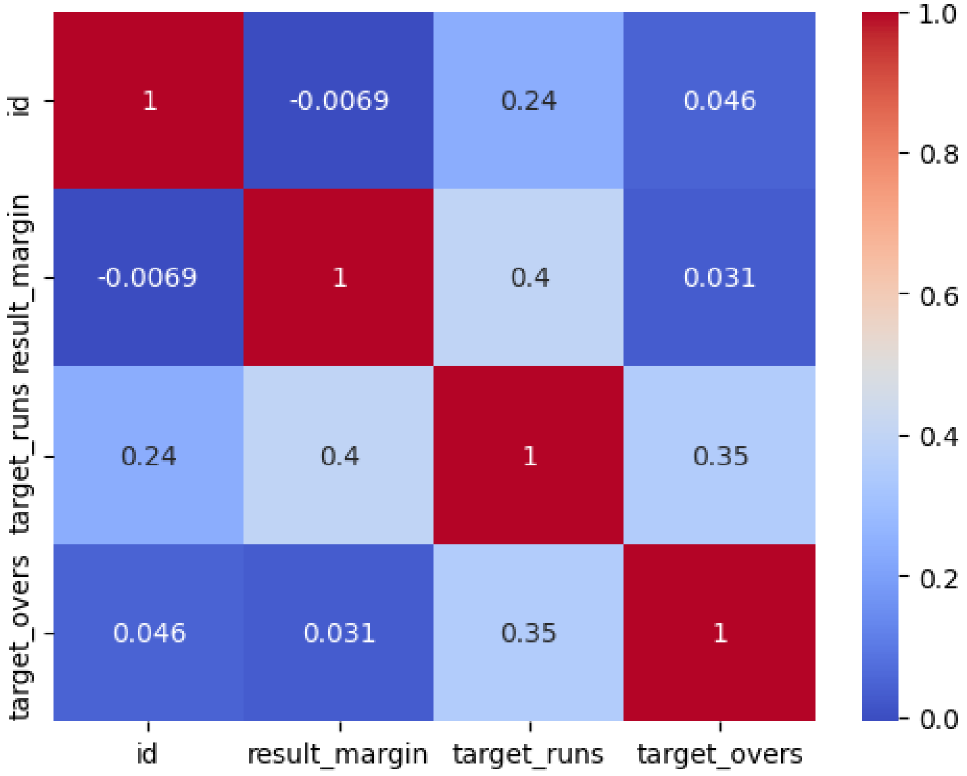
[14]:
```
cat_cols = [col for col in df_matches.columns if df_matches[col].dtype ==␣
 ↪'object']
num_cols = [col for col in df_matches.columns if col not in cat_cols]
```

[15]:
```
print(cat_cols)
print(num_cols)
```

```
['season', 'city', 'date', 'match_type', 'player_of_match', 'venue', 'team1',
'team2', 'toss_winner', 'toss_decision', 'winner', 'result', 'super_over',
'method', 'umpire1', 'umpire2']
['id', 'result_margin', 'target_runs', 'target_overs']
```

[16]:

```
# correlation matrix for categorical columns
cat_corr = df_matches[num_cols].corr()
sns.heatmap(cat_corr, annot=True, cmap='coolwarm')
plt.show()
```



Missing Value Handling:

[17]:

```
# Fill the missing values
df_matches.fillna(0, inplace=True)
df_matches.isnull().sum()
```

[17]:

```
id                 0
season             0
city               0
date               0
match_type         0
player_of_match    0
venue              0
team1              0
team2              0
toss_winner        0
toss_decision      0
```

```
winner              0
result              0
result_margin       0
target_runs         0
target_overs        0
super_over          0
method              0
umpire1             0
umpire2             0
dtype: int64
```

3 Match Data Boxplots

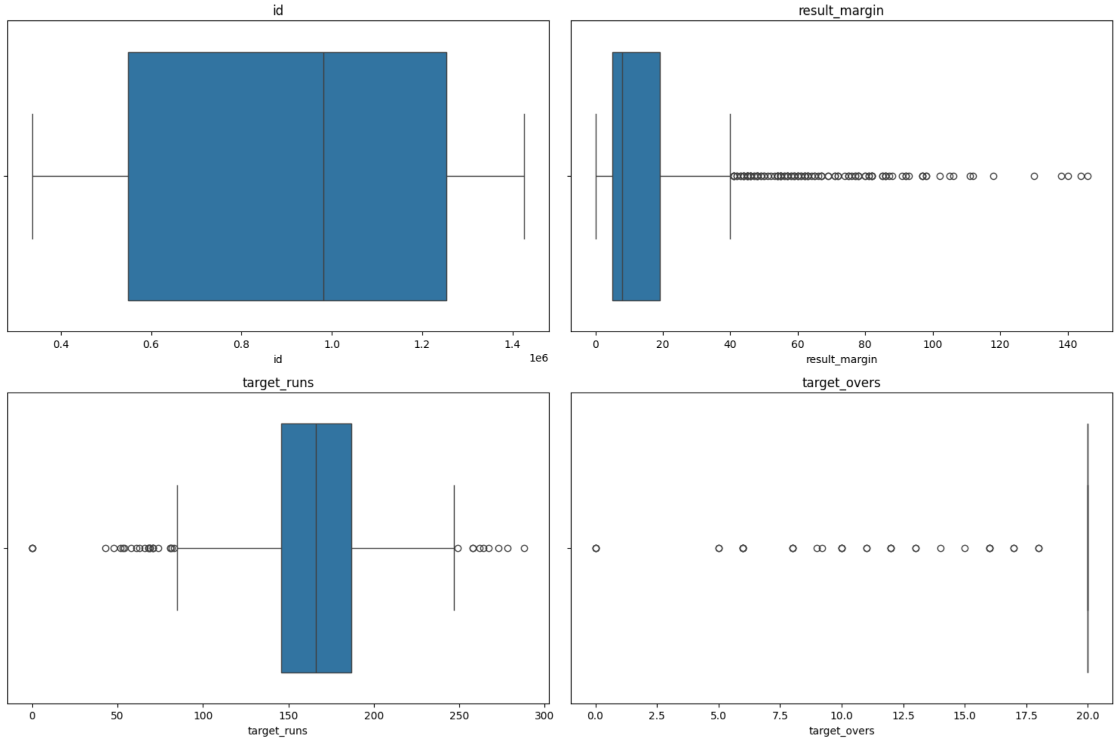
Four key variables visualized:

1. Match ID (top left):
   - IDs mostly between 0.6-1.2 million
   - Median ~0.9 million
   - Even distribution
2. Result Margin (top right):
   - Right-skewed distribution
   - Most matches have small margins (<20 runs)
   - Many outliers up to ~140 runs
3. Target Runs (bottom left):
   - Centered around 150 runs
   - Range typically 130-170 runs
   - Outliers at both extremes
4. Target Overs (bottom right):
   - Sparse distribution
   - Values scattered between 2.5-20 overs
   - Suggests different match formats

[18]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the number of rows and columns for the grid
num_plots = len(num_cols)
num_rows = int(num_plots**0.5)  # Square root for a roughly square grid
num_cols_grid = (num_plots + num_rows - 1) // num_rows  # Adjust for remainder

# Create the grid of subplots
fig, axes = plt.subplots(num_rows, num_cols_grid, figsize=(15, 5 * num_rows))

# Flatten the axes array for easier iteration
axes = axes.flatten()

# Iterate through the numerical columns and create box plots
```

```
for i, col in enumerate(num_cols):
    sns.boxplot(x=df_matches[col], ax=axes[i])
    axes[i].set_title(col)  # Set title for each subplot

# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```



3.0.1 Outlier Detection and Removal:

Removing Outliers Using the IQR Method The **Interquartile Range (IQR) method** is a statistical technique used to detect and remove outliers from a dataset. It identifies values that lie beyond **1.5 times the IQR** from the **first quartile (Q1)** and **third quartile (Q3)**.

3.0.2 Steps:

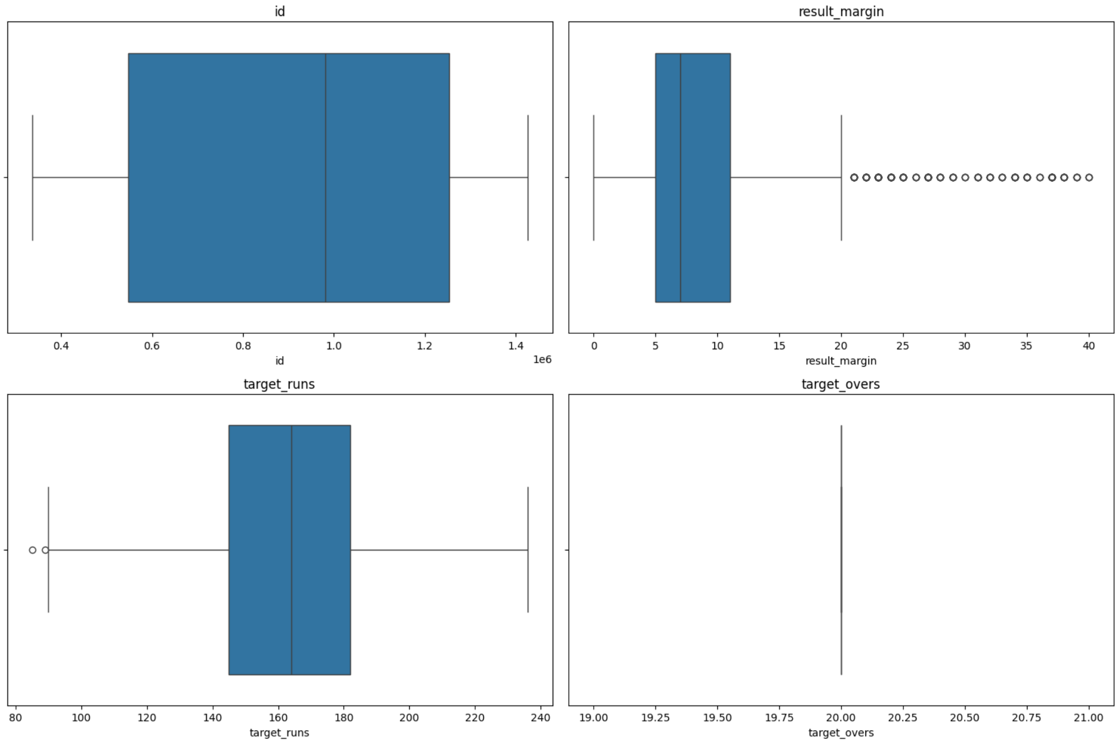
1. Calculate **Q1 (25th percentile)** and **Q3 (75th percentile)**.
2. Compute **IQR = Q3 - Q1**.
3. Define **lower bound = Q1 - 1.5 × IQR** and **upper bound = Q3 + 1.5 × IQR**.

4. Remove data points that fall outside these bounds. https://miro.medium.com/v2/resize:fit:1100/fo This method ensures a cleaner dataset by eliminating extreme values that could skew the analysis.

[20]:
```
# remove outliers using the IQR method
import pandas as pd

def remove_outliers_iqr(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

# Assuming 'num_cols' contains the numerical columns want to clean
df_matches_cleaned = remove_outliers_iqr(df_matches.copy(), num_cols)
```

####After Removing the outliers

[21]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the number of rows and columns for the grid
num_plots = len(num_cols)
num_rows = int(num_plots**0.5)  # Square root for a roughly square grid
num_cols_grid = (num_plots + num_rows - 1) // num_rows  # Adjust for remainder

# Create the grid of subplots
fig, axes = plt.subplots(num_rows, num_cols_grid, figsize=(15, 5 * num_rows))

# Flatten the axes array for easier iteration
axes = axes.flatten()

# Iterate through the numerical columns and create box plots
for i, col in enumerate(num_cols):
    sns.boxplot(x=df_matches_cleaned[col], ax=axes[i])
    axes[i].set_title(col)  # Set title for each subplot

# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```



#Plot Matches Played and Winning Percentages

[22]:

```
#count matches played
matches_played = df_matches_cleaned.groupby('season')['id'].count().reset_index()
matches_played.columns = ['season', 'matches_played']
print(matches_played)
```

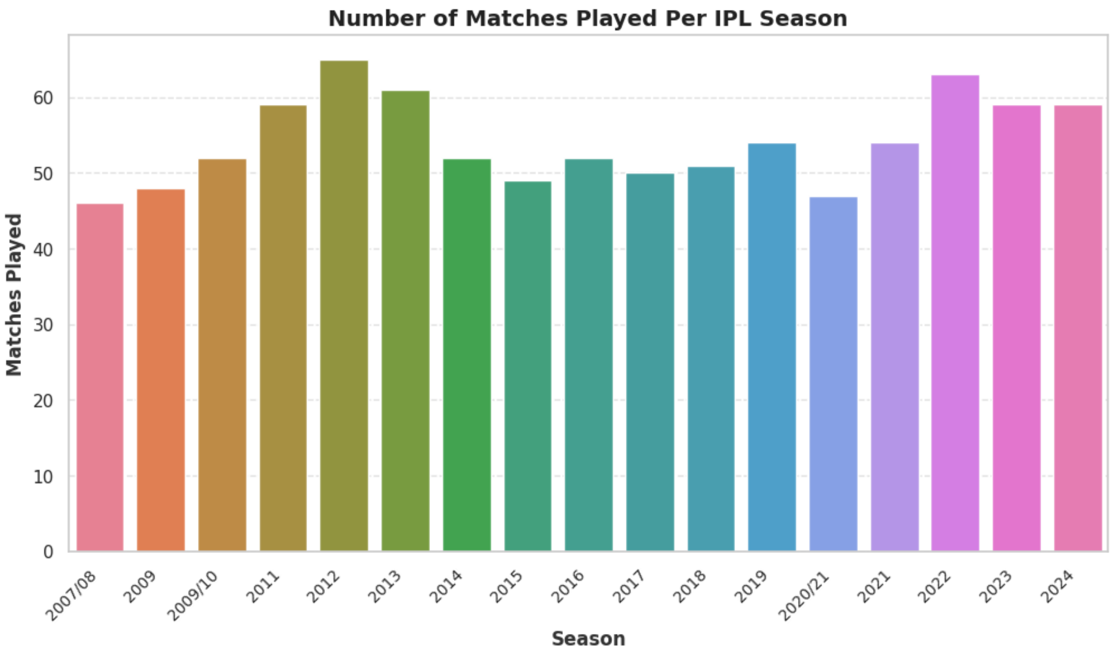
```
     season  matches_played
0   2007/08              46
1      2009              48
2   2009/10              52
3      2011              59
4      2012              65
5      2013              61
6      2014              52
7      2015              49
8      2016              52
9      2017              50
10     2018              51
11     2019              54
12  2020/21              47
13     2021              54
14     2022              63
15     2023              59
```

```
16     2024              59
```

[23]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Creating the DataFrame
data = {
    "season": ["2007/08", "2009", "2009/10", "2011", "2012", "2013", "2014",␣
 ↪"2015", "2016",
               "2017", "2018", "2019", "2020/21", "2021", "2022", "2023",␣
 ↪"2024"],
    "matches_played": [46, 48, 52, 59, 65, 61, 52, 49, 52, 50, 51, 54, 47, 54,␣
 ↪63, 59, 59]
}

df = pd.DataFrame(data)

# Set the style
sns.set_theme(style="whitegrid", palette="pastel")

# Figure size
plt.figure(figsize=(12, 6))

# Create barplot with vibrant colors
colors = sns.color_palette("husl", len(df))
sns.barplot(x="season", y="matches_played", data=df, palette=colors)

# Beautify the plot
plt.xticks(rotation=45, fontsize=10, ha='right')
plt.xlabel("Season", fontsize=12, fontweight='bold', color="#333")
plt.ylabel("Matches Played", fontsize=12, fontweight='bold', color="#333")
plt.title("Number of Matches Played Per IPL Season", fontsize=14,␣
 ↪fontweight='bold', color="#222")
plt.grid(axis='y', linestyle="--", alpha=0.6)

# Show plot
plt.show()
```

```
<ipython-input-23-457713660999>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x="season", y="matches_played", data=df, palette=colors)
```



3.1 Cricket Team Winning Percentage Analysis

This analysis tracks team performance across cricket seasons by calculating winning percentages:

1. **Group & Count** Organize matches by season and count wins per team
2. **Structure Data** Create a table with seasons as rows and teams as columns

3. **Calculate Totals** Sum matches per season across all teams
4. **Determine Success Rates** Calculate winning percentage for the primary team

The resulting visualization reveals performance patterns and dominance trends throughout cricket seasons, helping identify consistently strong teams and potential competitive shifts over time.

```
[ ]: # calculate winning percentage for each player
     matches_won = df_matches_cleaned.groupby('season')['winner'].value_counts().
      ↪unstack().fillna(0)
     matches_won['total_matches'] = matches_won.sum(axis=1)
     matches_won['winning_percentage'] = (matches_won.iloc[:, 0] /␣
      ↪matches_won['total_matches']) * 100
     matches_won
```

```
[ ]: winner      0  Chennai Super Kings  Deccan Chargers  Delhi Capitals  \
     season
     2007/08  0.0                  7.0              2.0             0.0
     2009     0.0                  5.0              8.0             0.0
     2009/10  0.0                  7.0              8.0             0.0
     2011     0.0                  9.0              4.0             0.0
```

```
2012     0.0                  8.0              4.0             0.0
2013     0.0                  9.0              0.0             0.0
2014     0.0                  8.0              0.0             0.0
2015     1.0                  8.0              0.0             0.0
2016     0.0                  0.0              0.0             0.0
2017     0.0                  0.0              0.0             0.0
2018     0.0                 10.0              0.0             0.0
2019     0.0                  8.0              0.0            10.0
2020/21  0.0                  6.0              0.0             6.0
2021     0.0                  9.0              0.0            10.0
2022     0.0                  3.0              0.0             6.0
2023     0.0                  7.0              0.0             5.0
2024     0.0                  5.0              0.0             6.0

winner   Delhi Daredevils  Gujarat Lions  Gujarat Titans  Kings XI Punjab  \
season
2007/08               7.0            0.0             0.0              7.0
2009                  9.0            0.0             0.0              7.0
2009/10               6.0            0.0             0.0              4.0
2011                  4.0            0.0             0.0              4.0
2012                 10.0            0.0             0.0              8.0
2013                  3.0            0.0             0.0              7.0
2014                  2.0            0.0             0.0             10.0
2015                  5.0            0.0             0.0              2.0
2016                  7.0            9.0             0.0              4.0
2017                  4.0            4.0             0.0              6.0
2018                  3.0            0.0             0.0              5.0
2019                  0.0            0.0             0.0              6.0
2020/21               0.0            0.0             0.0              5.0
2021                  0.0            0.0             0.0              0.0
2022                  0.0            0.0            11.0              0.0
2023                  0.0            0.0             8.0              0.0
2024                  0.0            0.0             5.0              0.0

winner   Kochi Tuskers Kerala  Kolkata Knight Riders  ...  Pune Warriors  \
season                                                ...
2007/08                   0.0                    4.0  ...            0.0
2009                      0.0                    2.0  ...            0.0
2009/10                   0.0                    7.0  ...            0.0
2011                      5.0                    6.0  ...            4.0
2012                      0.0                   10.0  ...            4.0
2013                      0.0                    4.0  ...            4.0
2014                      0.0                   10.0  ...            0.0
2015                      0.0                    7.0  ...            0.0
2016                      0.0                    7.0  ...            0.0
2017                      0.0                    7.0  ...            0.0
2018                      0.0                    7.0  ...            0.0
```

```
2019                      0.0                    6.0  ...            0.0
2020/21                   0.0                    5.0  ...            0.0
2021                      0.0                    8.0  ...            0.0
2022                      0.0                    4.0  ...            0.0
2023                      0.0                    5.0  ...            0.0
2024                      0.0                    8.0  ...            0.0

winner   Punjab Kings  Rajasthan Royals  Rising Pune Supergiant  \
season
2007/08           0.0              10.0                     0.0
2009              0.0               5.0                     0.0
2009/10           0.0               6.0                     0.0
2011              0.0               6.0                     0.0
2012              0.0               4.0                     0.0
2013              0.0              10.0                     0.0
2014              0.0               5.0                     0.0
2015              0.0               7.0                     0.0
2016              0.0               0.0                     0.0
2017              0.0               0.0                     8.0
2018              0.0               6.0                     0.0
2019              0.0               5.0                     0.0
2020/21           0.0               6.0                     0.0
2021              6.0               4.0                     0.0
2022              5.0               9.0                     0.0
2023              5.0               5.0                     0.0
2024              4.0               9.0                     0.0

winner   Rising Pune Supergiants  Royal Challengers Bangalore  \
season
2007/08                      0.0                          4.0
2009                         0.0                          8.0
2009/10                      0.0                          7.0
2011                         0.0                          7.0
2012                         0.0                          7.0
2013                         0.0                          7.0
2014                         0.0                          5.0
2015                         0.0                          4.0
2016                         3.0                          6.0
2017                         0.0                          3.0
2018                         0.0                          6.0
2019                         0.0                          5.0
2020/21                      0.0                          6.0
2021                         0.0                          8.0
2022                         0.0                          8.0
2023                         0.0                          6.0
2024                         0.0                          0.0
```

```
winner   Royal Challengers Bengaluru  Sunrisers Hyderabad  total_matches  \
season
2007/08                          0.0                  0.0           46.0
2009                             0.0                  0.0           48.0
2009/10                          0.0                  0.0           52.0
2011                             0.0                  0.0           59.0
2012                             0.0                  0.0           65.0
2013                             0.0                  9.0           61.0
2014                             0.0                  5.0           52.0
2015                             0.0                  6.0           49.0
2016                             0.0                 10.0           52.0
2017                             0.0                  7.0           50.0
2018                             0.0                 10.0           51.0
2019                             0.0                  4.0           54.0
2020/21                          0.0                  6.0           47.0
2021                             0.0                  3.0           54.0
2022                             0.0                  5.0           63.0
2023                             0.0                  4.0           59.0
2024                             5.0                  6.0           59.0

winner   winning_percentage
season
2007/08            0.000000
2009               0.000000
2009/10            0.000000
2011               0.000000
2012               0.000000
2013               0.000000
2014               0.000000
2015               2.040816
2016               0.000000
2017               0.000000
2018               0.000000
2019               0.000000
2020/21            0.000000
2021               0.000000
2022               0.000000
2023               0.000000
2024               0.000000

[17 rows x 22 columns]
```

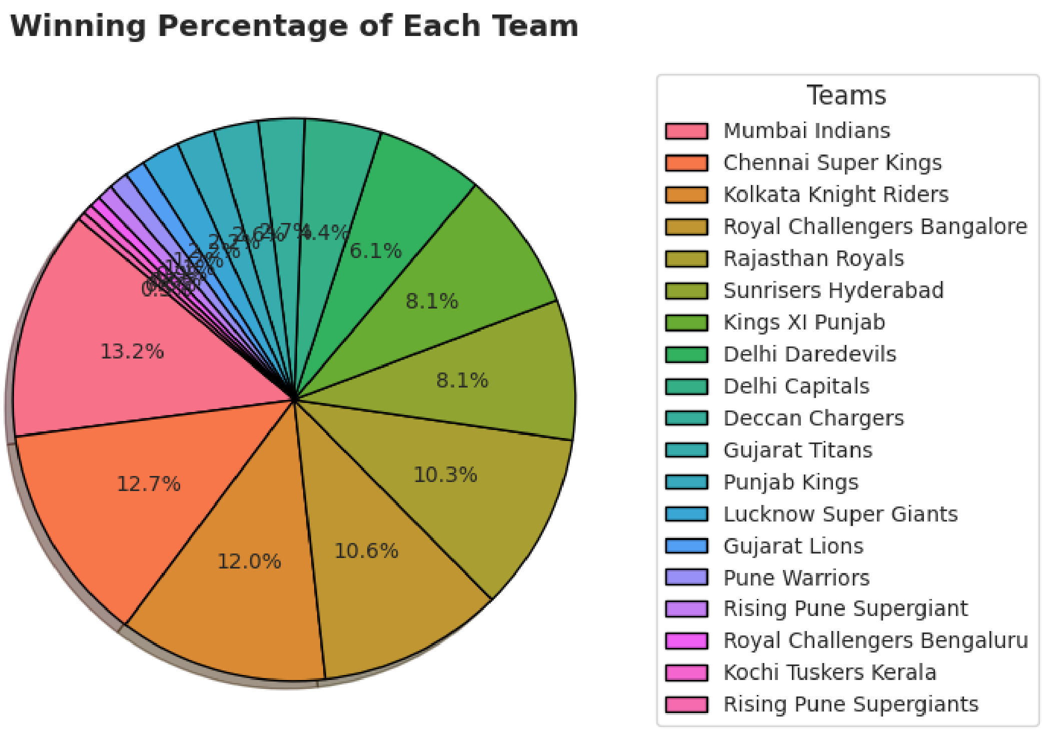
3.2 Team Performance Analysis & Visualization

This code performs a comprehensive analysis of cricket team winning patterns:

The resulting bar chart provides a clear visual comparison of team dominance across the dataset, highlighting which teams have been most successful by percentage of total matches won. The analysis filters out matches with no declared winner to ensure data accuracy.

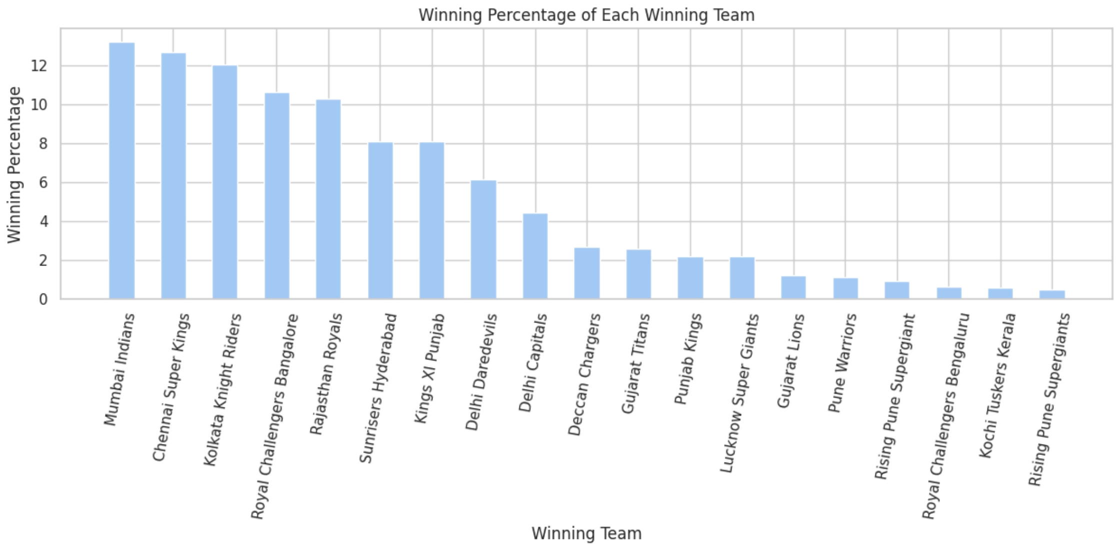
[28]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Remove rows where 'winner' is 0
df_matches = df_matches[df_matches['winner'] != 0]

# Calculate winning percentage
winning_percentage = df_matches['winner'].value_counts(normalize=True) * 100

# Define colors using Seaborn
colors = sns.color_palette("husl", len(winning_percentage))

# Create figure and axis
fig, ax = plt.subplots(figsize=(10, 6))

# Plot pie chart without labels
wedges, texts, autotexts = ax.pie(
    winning_percentage.values,
    autopct='%1.1f%%',
    startangle=140,
    colors=colors,
    shadow=True,
    wedgeprops={'edgecolor': 'black'},
    textprops={'fontsize': 10}  # Reduce label font size
)

# Add legend (index) on the right side
ax.legend(
    wedges, winning_percentage.index,
    title="Teams",
    loc="center left",
    bbox_to_anchor=(1, 0.5),
    fontsize=10
)

plt.title("Winning Percentage of Each Team", fontsize=14, fontweight="bold")
plt.show()
```



[24]:

```
#plot winning percentage of each winning team
#group each winner and calculate their percentage

#remove the rows where df_matches['winner'] = 0
df_matches = df_matches[df_matches['winner'] != 0]
winning_percentage = df_matches['winner'].value_counts(normalize=True) * 100
winning_percentage



#plot the players vs winning percentage
plt.figure(figsize=(12, 6))

# Convert index to strings
winning_percentage.index = winning_percentage.index.astype(str)

# Increase gap between bars
plt.bar(winning_percentage.index, winning_percentage.values, width=0.5,␣
 ↪align='center')

plt.xlabel('Winning Team')
plt.ylabel('Winning Percentage')
plt.title('Winning Percentage of Each Winning Team')
plt.xticks(rotation=80) # Rotate x-axis labels for better visibility if needed
plt.tight_layout() # Adjust layout for better spacing
plt.show()
```



[29]:
```
# calculate each team played how many matches
team_matches = df_matches_cleaned['team1'].value_counts() +␣
↪df_matches_cleaned['team2'].value_counts()
team_matches.size
```

[29]:
```
19
```

[30]:
```
winning_percentage
```

[30]:
```
winner
Mumbai Indians                 13.211009
Chennai Super Kings            12.660550
Kolkata Knight Riders          12.018349
Royal Challengers Bangalore    10.642202
Rajasthan Royals               10.275229
Sunrisers Hyderabad             8.073394
Kings XI Punjab                 8.073394
Delhi Daredevils                6.146789
Delhi Capitals                  4.403670
Deccan Chargers                 2.660550
Gujarat Titans                  2.568807
Punjab Kings                    2.201835
Lucknow Super Giants            2.201835
```

```
Gujarat Lions                  1.192661
Pune Warriors                  1.100917
Rising Pune Supergiant         0.917431
Royal Challengers Bengaluru    0.642202
Kochi Tuskers Kerala           0.550459
Rising Pune Supergiants        0.458716
Name: proportion, dtype: float64
```

```
[31]: team_matches
```

```
[31]: Chennai Super Kings            199
      Deccan Chargers                 67
      Delhi Capitals                  75
      Delhi Daredevils               137
      Gujarat Lions                   29
      Gujarat Titans                  39
      Kings XI Punjab                160
      Kochi Tuskers Kerala            12
      Kolkata Knight Riders          205
      Lucknow Super Giants            36
      Mumbai Indians                 222
      Pune Warriors                   41
      Punjab Kings                    50
      Rajasthan Royals               186
      Rising Pune Supergiant          13
      Rising Pune Supergiants         11
      Royal Challengers Bangalore    194
      Royal Challengers Bengaluru     12
      Sunrisers Hyderabad            154
      Name: count, dtype: int64
```

```
[32]: team_matches_dict = team_matches.to_dict()
      print(team_matches_dict)
```

```
{'Chennai Super Kings': 199, 'Deccan Chargers': 67, 'Delhi Capitals': 75, 'Delhi
Daredevils': 137, 'Gujarat Lions': 29, 'Gujarat Titans': 39, 'Kings XI Punjab':
160, 'Kochi Tuskers Kerala': 12, 'Kolkata Knight Riders': 205, 'Lucknow Super
Giants': 36, 'Mumbai Indians': 222, 'Pune Warriors': 41, 'Punjab Kings': 50,
'Rajasthan Royals': 186, 'Rising Pune Supergiant': 13, 'Rising Pune
Supergiants': 11, 'Royal Challengers Bangalore': 194, 'Royal Challengers
Bengaluru': 12, 'Sunrisers Hyderabad': 154}
```

```
[33]: winning_percentage_dict = winning_percentage.to_dict()
      print(winning_percentage_dict)
```

```
{'Mumbai Indians': 13.211009174311927, 'Chennai Super Kings':
12.660550458715598, 'Kolkata Knight Riders': 12.018348623853212, 'Royal
Challengers Bangalore': 10.642201834862385, 'Rajasthan Royals':
```

```
10.275229357798166, 'Sunrisers Hyderabad': 8.073394495412845, 'Kings XI Punjab':
8.073394495412845, 'Delhi Daredevils': 6.146788990825688, 'Delhi Capitals':
4.4036697247706424, 'Deccan Chargers': 2.6605504587155964, 'Gujarat Titans':
2.5688073394495414, 'Punjab Kings': 2.2018348623853212, 'Lucknow Super Giants':
2.2018348623853212, 'Gujarat Lions': 1.1926605504587156, 'Pune Warriors':
1.1009174311926606, 'Rising Pune Supergiant': 0.9174311926605505, 'Royal
Challengers Bengaluru': 0.6422018348623854, 'Kochi Tuskers Kerala':
0.5504587155963303, 'Rising Pune Supergiants': 0.45871559633027525}
```

[34]:

```python
# sort winning_percentage_dic and team_matches according to keys
winning_percentage_sorted = {k: winning_percentage_dict[k] for k in␣
 ↪sorted(winning_percentage_dict)}
team_matches_sorted = {k: team_matches_dict[k] for k in␣
 ↪sorted(team_matches_dict)}
```

[35]:

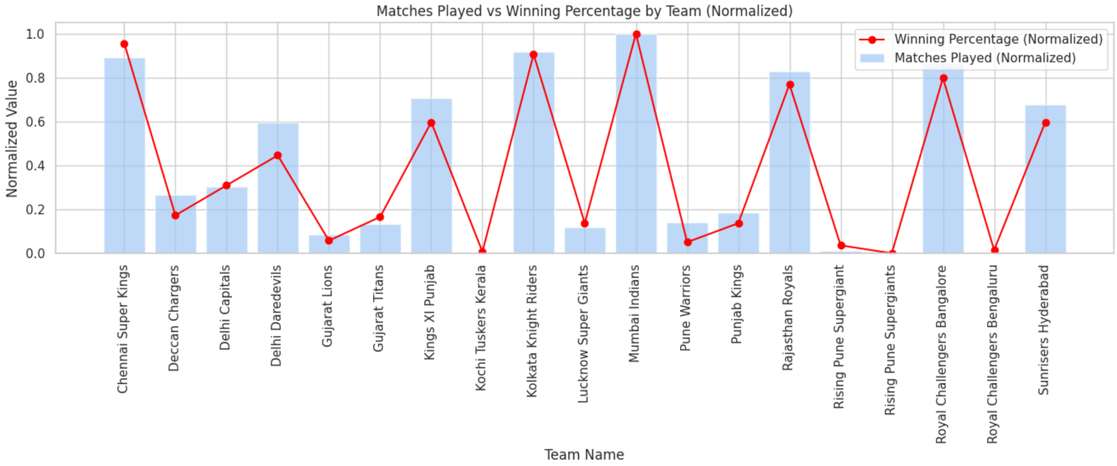
```python
print(winning_percentage_sorted)
print(team_matches_sorted)
```

```
{'Chennai Super Kings': 12.660550458715598, 'Deccan Chargers':
2.6605504587155964, 'Delhi Capitals': 4.4036697247706424, 'Delhi Daredevils':
6.146788990825688, 'Gujarat Lions': 1.1926605504587156, 'Gujarat Titans':
2.5688073394495414, 'Kings XI Punjab': 8.073394495412845, 'Kochi Tuskers
Kerala': 0.5504587155963303, 'Kolkata Knight Riders': 12.018348623853212,
'Lucknow Super Giants': 2.2018348623853212, 'Mumbai Indians':
13.211009174311927, 'Pune Warriors': 1.1009174311926606, 'Punjab Kings':
2.2018348623853212, 'Rajasthan Royals': 10.275229357798166, 'Rising Pune
Supergiant': 0.9174311926605505, 'Rising Pune Supergiants': 0.45871559633027525,
'Royal Challengers Bangalore': 10.642201834862385, 'Royal Challengers
Bengaluru': 0.6422018348623854, 'Sunrisers Hyderabad': 8.073394495412845}
{'Chennai Super Kings': 199, 'Deccan Chargers': 67, 'Delhi Capitals': 75, 'Delhi
Daredevils': 137, 'Gujarat Lions': 29, 'Gujarat Titans': 39, 'Kings XI Punjab':
160, 'Kochi Tuskers Kerala': 12, 'Kolkata Knight Riders': 205, 'Lucknow Super
Giants': 36, 'Mumbai Indians': 222, 'Pune Warriors': 41, 'Punjab Kings': 50,
'Rajasthan Royals': 186, 'Rising Pune Supergiant': 13, 'Rising Pune
Supergiants': 11, 'Royal Challengers Bangalore': 194, 'Royal Challengers
Bengaluru': 12, 'Sunrisers Hyderabad': 154}
```

[36]:

```python
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# Get the team names (keys)
team_names = list(team_matches_sorted.keys())

# Get the values for plotting
team_matches_values = list(team_matches_sorted.values())
winning_percentage_values = list(winning_percentage_sorted.values())

# Normalize the values using MinMaxScaler
```

```
scaler = MinMaxScaler()
team_matches_values_normalized = scaler.fit_transform(np.
 ↪array(team_matches_values).reshape(-1, 1)).flatten()
winning_percentage_values_normalized = scaler.fit_transform(np.
 ↪array(winning_percentage_values).reshape(-1, 1)).flatten()

# Plotting
plt.figure(figsize=(14, 6))  # Increase figure size

# Plot 'matches_played' with bars
plt.bar(team_names, team_matches_values_normalized, label='Matches Played␣
 ↪(Normalized)', alpha=0.7)

# Plot 'winning_percentage' with a line and markers
plt.plot(team_names, winning_percentage_values_normalized, marker='o',␣
 ↪color='red', label='Winning Percentage (Normalized)')

plt.xlabel('Team Name')
plt.ylabel('Normalized Value')
plt.title('Matches Played vs Winning Percentage by Team (Normalized)')
plt.xticks(rotation=90)  # Rotate x-axis labels for readability
plt.legend()
plt.tight_layout()
plt.show()
```



3.2.1 Plot Run Rate and Economy Rate

[43]:
```
df_matches_cleaned.keys()
```

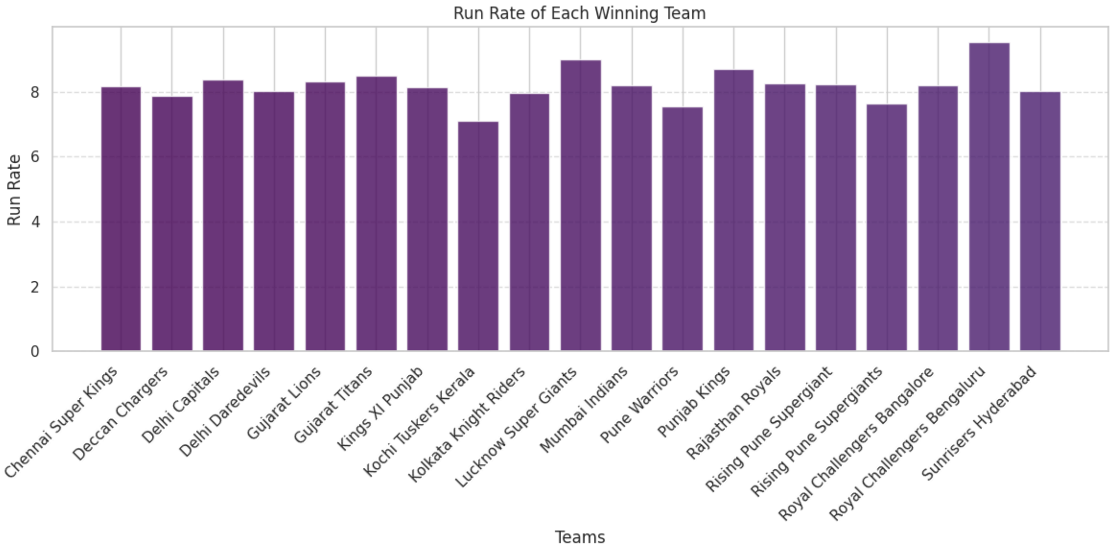
```
[43]: Index(['id', 'season', 'city', 'date', 'match_type', 'player_of_match',
              'venue', 'team1', 'team2', 'toss_winner', 'toss_decision', 'winner',
              'result', 'result_margin', 'target_runs', 'target_overs', 'super_over',
              'method', 'umpire1', 'umpire2'],
             dtype='object')
```

```
[44]: # Group by 'winner' and sum up total runs and total overs
      team_runs = df_matches_cleaned.groupby('winner')['target_runs'].sum()
      team_overs = df_matches_cleaned.groupby('winner')['target_overs'].sum()

      # Calculate Run Rate
      run_rate = team_runs / team_overs

      # Display the result
      print(run_rate)
```

```
winner
Chennai Super Kings            8.172477
Deccan Chargers                7.886538
Delhi Capitals                 8.393023
Delhi Daredevils               8.035000
Gujarat Lions                  8.315385
Gujarat Titans                 8.508333
Kings XI Punjab                8.153333
Kochi Tuskers Kerala           7.120000
Kolkata Knight Riders          7.967290
Lucknow Super Giants           9.007143
Mumbai Indians                 8.212083
Pune Warriors                  7.562500
Punjab Kings                   8.702500
Rajasthan Royals               8.262371
Rising Pune Supergiant         8.218750
Rising Pune Supergiants        7.633333
Royal Challengers Bangalore    8.191237
Royal Challengers Bengaluru    9.520000
Sunrisers Hyderabad            8.017333
dtype: float64
```

```
[55]: import matplotlib.pyplot as plt

      # Group by 'winner' and sum up total runs and total overs
      team_runs = df_matches_cleaned.groupby('winner')['target_runs'].sum()
      team_overs = df_matches_cleaned.groupby('winner')['target_overs'].sum()

      # Calculate Run Rate
      run_rate = team_runs / team_overs

      # Plot
```

```
plt.figure(figsize=(12, 6))
colors = plt.cm.viridis(range(len(run_rate)))  # Generate colors

bars = plt.bar(run_rate.index, run_rate.values, color=colors, alpha=0.8)

plt.xlabel('Teams')
plt.ylabel('Run Rate')
plt.title('Run Rate of Each Winning Team')

# Rotate x-axis labels
plt.xticks(rotation=45, ha='right')

# Add grid for readability
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```



[51]:

```
# Remove rows where the winner is 0 (invalid entries)
df_matches_cleaned = df_matches_cleaned[df_matches_cleaned['winner'] != 0]

# Compute total target runs for each winning team
target_run = df_matches_cleaned.groupby('winner')['target_runs'].sum()

# Display the result
target_run
```

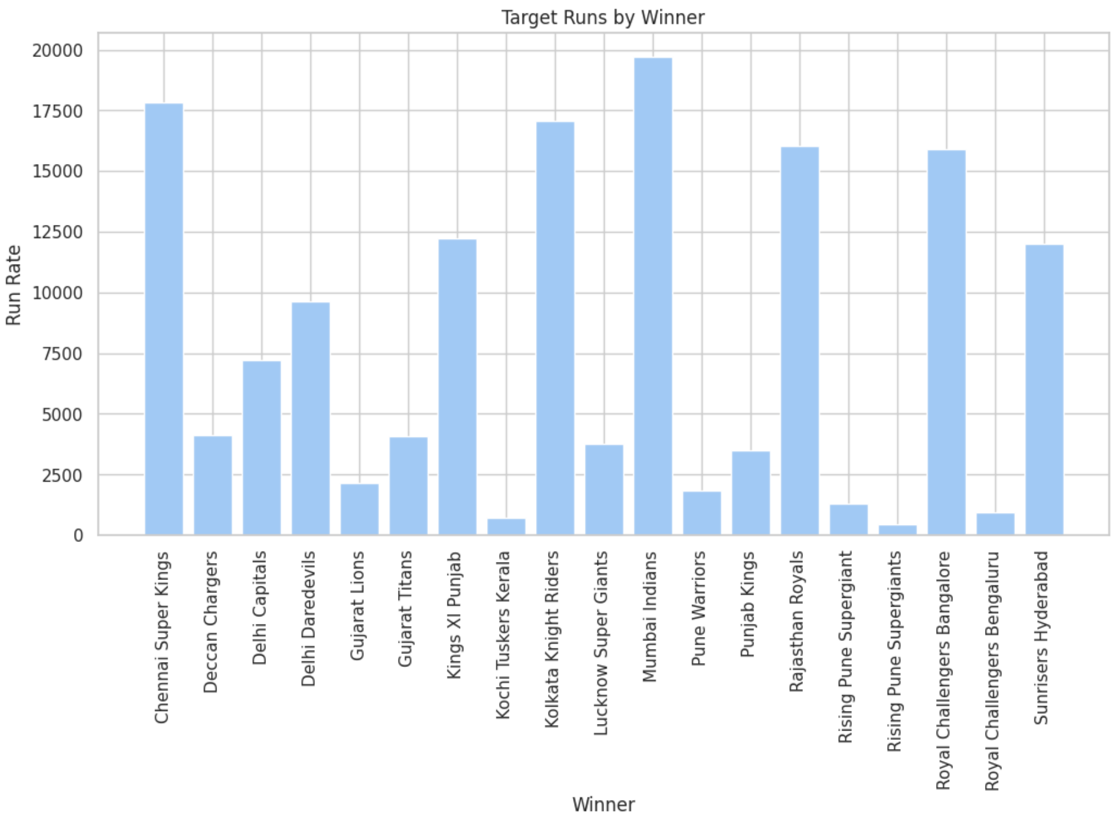
```
[51]: winner
      Chennai Super Kings            17816.0
      Deccan Chargers                 4101.0
      Delhi Capitals                  7218.0
      Delhi Daredevils                9642.0
      Gujarat Lions                   2162.0
      Gujarat Titans                  4084.0
      Kings XI Punjab                12230.0
      Kochi Tuskers Kerala             712.0
      Kolkata Knight Riders          17050.0
      Lucknow Super Giants            3783.0
      Mumbai Indians                 19709.0
      Pune Warriors                   1815.0
      Punjab Kings                    3481.0
      Rajasthan Royals               16029.0
      Rising Pune Supergiant          1315.0
      Rising Pune Supergiants          458.0
      Royal Challengers Bangalore    15891.0
      Royal Challengers Bengaluru      952.0
      Sunrisers Hyderabad            12026.0
      Name: target_runs, dtype: float64
```

```
[52]: # create dict from target_runs
      target_run_dict = target_run.to_dict()
      print(target_run_dict)
```

```
{'Chennai Super Kings': 17816.0, 'Deccan Chargers': 4101.0, 'Delhi Capitals':
7218.0, 'Delhi Daredevils': 9642.0, 'Gujarat Lions': 2162.0, 'Gujarat Titans':
4084.0, 'Kings XI Punjab': 12230.0, 'Kochi Tuskers Kerala': 712.0, 'Kolkata
Knight Riders': 17050.0, 'Lucknow Super Giants': 3783.0, 'Mumbai Indians':
19709.0, 'Pune Warriors': 1815.0, 'Punjab Kings': 3481.0, 'Rajasthan Royals':
16029.0, 'Rising Pune Supergiant': 1315.0, 'Rising Pune Supergiants': 458.0,
'Royal Challengers Bangalore': 15891.0, 'Royal Challengers Bengaluru': 952.0,
'Sunrisers Hyderabad': 12026.0}
```

```
[53]: # plot target_run
      plt.figure(figsize=(12, 6))
      plt.bar(target_run_dict.keys(), target_run_dict.values())
      plt.xlabel('Winner')
      plt.ylabel('Run Rate')
      plt.title('Target Runs by Winner')
      plt.xticks(rotation=90)
      plt.show
```

```
[53]: <function matplotlib.pyplot.show(close=None, block=None)>
```



[82]:
```
# calculate Economy rate
df_deliveries = pd.read_csv('deliveries.csv')
```

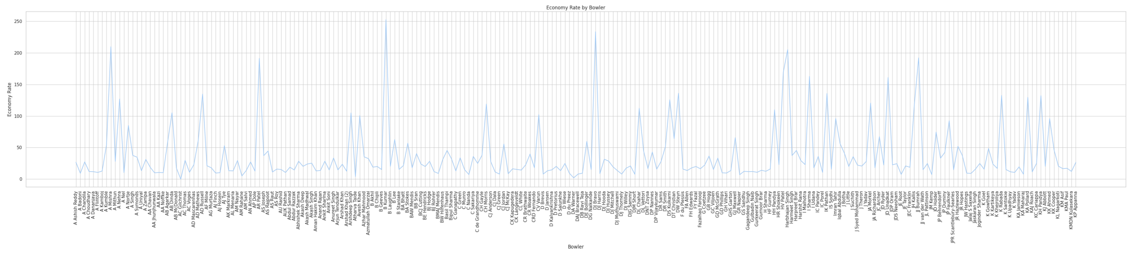
[83]:
```
# calculate average of total_runs given by a bowler in one over
df_deliveries_economy = df_deliveries.groupby(['bowler', 'over'])['total_runs'].
 ↪sum().reset_index()
df_deliveries_economy

# do the average based on bowler
df_deliveries_economy = df_deliveries_economy.groupby('bowler')['total_runs'].
 ↪mean().reset_index()
df_deliveries_economy
```

[83]:
```
               bowler  total_runs
0     A Ashish Reddy   26.666667
1           A Badoni    9.250000
2         A Chandila   27.222222
3        A Choudhary   12.000000
4        A Dananjaya   11.750000
..               ...         ...
525      Yash Dayal    49.105263
```

```
526     Yash Thakur   39.000000
527   Yudhvir Singh   13.888889
528    Yuvraj Singh   83.923077
529          Z Khan  143.000000

[530 rows x 2 columns]
```

[84]:

```
# plot df_deliveries_economy
plt.figure(figsize=(50, 8))

# Assuming you want to skip the last 5 points
num_points_to_skip = 300
plt.plot(df_deliveries_economy['bowler'][:-num_points_to_skip],
         df_deliveries_economy['total_runs'][:-num_points_to_skip])

plt.xlabel('Bowler')
plt.ylabel('Economy Rate')
plt.title('Economy Rate by Bowler')
plt.xticks(rotation=90)
plt.show()
```



[56]:

```
# calculate Economy rate
df_deliveries_cleaned=df_deliveries
```

[57]:

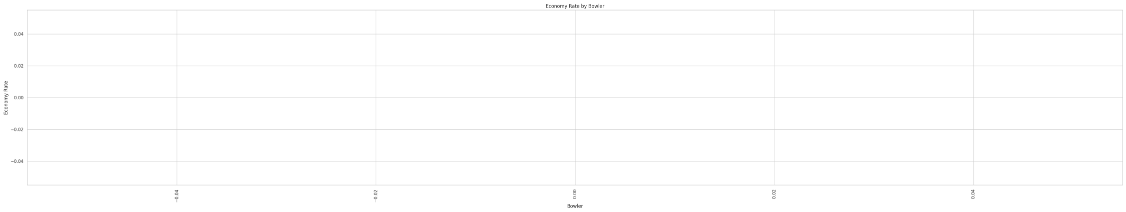
```
# calculate average of total_runs given by a bowler in one over
df_deliveries_economy = df_deliveries_cleaned.groupby(['bowler',␣
 ↪'over'])['total_runs'].sum().reset_index()
df_deliveries_economy

# do the average based on bowler
df_deliveries_economy = df_deliveries_economy.groupby('bowler')['total_runs'].
 ↪mean().reset_index()
df_deliveries_economy
```

[57]:

```
              bowler  total_runs
0         A Flintoff   10.600000
1           A Kumble   39.789474
```

```
2            A Mishra   34.058824
3            A Mithun   10.500000
4             A Nehra   39.500000
..                ...         ...
159  Y Venugopal Rao   20.000000
160        YA Abdulla   18.294118
161         YK Pathan   29.277778
162     Yuvraj Singh   23.083333
163            Z Khan   38.733333

[164 rows x 2 columns]
```

[58]:
```
# plot df_deliveries_economy
plt.figure(figsize=(50, 8))

# Assuming you want to skip the last 5 points
num_points_to_skip = 300
plt.plot(df_deliveries_economy['bowler'][:-num_points_to_skip],
         df_deliveries_economy['total_runs'][:-num_points_to_skip])

plt.xlabel('Bowler')
plt.ylabel('Economy Rate')
plt.title('Economy Rate by Bowler')
plt.xticks(rotation=90)
plt.show()
```



[59]:
```
df_matches = pd.read_csv('matches.csv')
df_matches['target_runs']
```

[59]:
```
0       223.0
1       241.0
2       130.0
3       166.0
4       111.0
        ...
1090    215.0
1091    160.0
1092    173.0
```

```
1093    176.0
1094    114.0
Name: target_runs, Length: 1095, dtype: float64
```

4 Plot Highest and Lowest Scores

[61]:

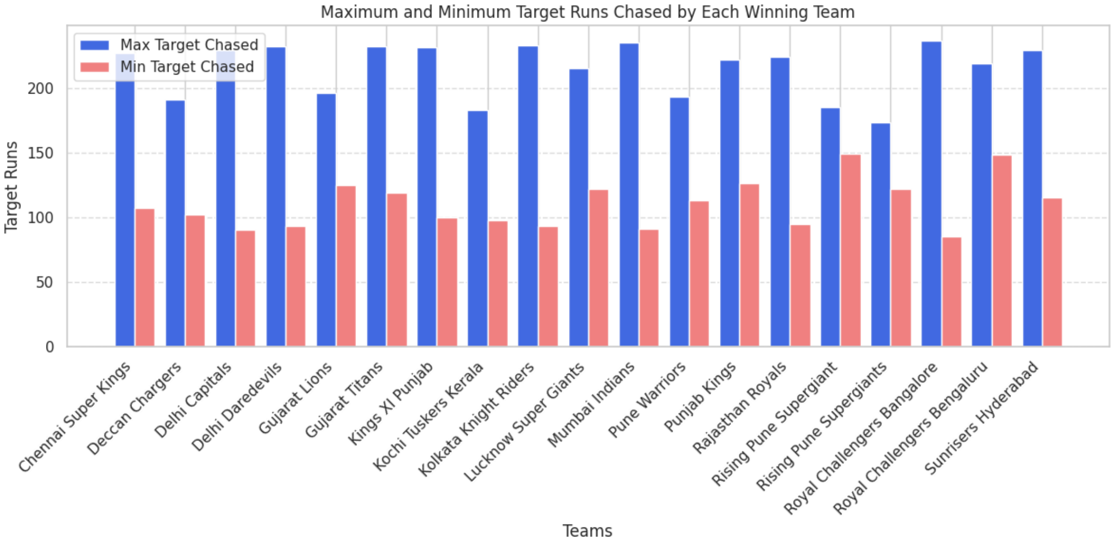
```
df_matches.keys()
```

[61]:

```
Index(['id', 'season', 'city', 'date', 'match_type', 'player_of_match',
       'venue', 'team1', 'team2', 'toss_winner', 'toss_decision', 'winner',
       'result', 'result_margin', 'target_runs', 'target_overs', 'super_over',
       'method', 'umpire1', 'umpire2'],
      dtype='object')
```

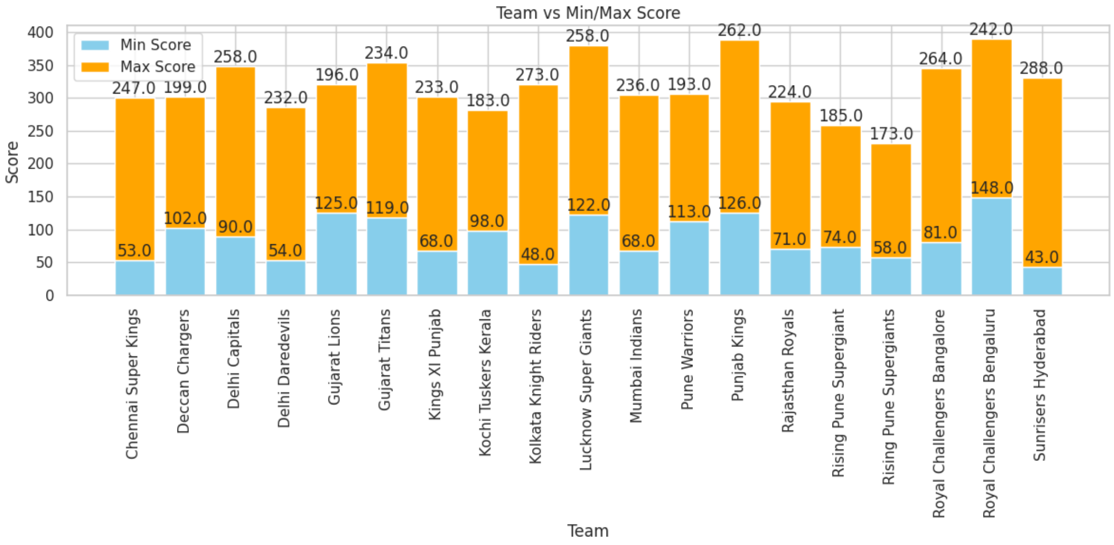
[62]:

```
import matplotlib.pyplot as plt
import numpy as np

# Remove rows where 'winner' is 0
df_matches_cleaned = df_matches_cleaned[df_matches_cleaned['winner'] != 0]

# Group by 'winner' and find max and min target runs chased
max_target = df_matches_cleaned.groupby('winner')['target_runs'].max()
min_target = df_matches_cleaned.groupby('winner')['target_runs'].min()

# Plot
teams = max_target.index  # Get team names
x = np.arange(len(teams))  # X-axis positions

plt.figure(figsize=(12, 6))

# Bar width
bar_width = 0.4

# Plot max target
plt.bar(x - bar_width/2, max_target, width=bar_width, label='Max Target Chased',␣
 ↪color='royalblue')

# Plot min target
plt.bar(x + bar_width/2, min_target, width=bar_width, label='Min Target Chased',␣
 ↪color='lightcoral')

# X-axis labels (team names) rotated
plt.xticks(x, teams, rotation=45, ha='right')

# Labels and title
plt.xlabel('Teams')
plt.ylabel('Target Runs')
plt.title('Maximum and Minimum Target Runs Chased by Each Winning Team')

# Grid and legend
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend()

plt.tight_layout()
plt.show()
```



[63]:

```
import matplotlib.pyplot as plt
import pandas as pd

# Group by 'winner' and get min/max target_runs
team_stats = df_matches.groupby('winner')['target_runs'].agg(['min', 'max'])

# Reset index to access 'winner' as a column
team_stats = team_stats.reset_index()

# Plotting
plt.figure(figsize=(12, 6))
plt.bar(team_stats['winner'], team_stats['min'], color='skyblue', label='Min␣
 ↪Score')
plt.bar(team_stats['winner'], team_stats['max'], color='orange',␣
 ↪bottom=team_stats['min'], label='Max Score')

# Annotations (optional)
for i, row in team_stats.iterrows():
    plt.text(i, row['min'], str(row['min']), ha='center', va='bottom')
    plt.text(i, row['max'] + row['min'], str(row['max']), ha='center',␣
 ↪va='bottom')

# Customize the plot
plt.xlabel('Team')
plt.ylabel('Score')
plt.title('Team vs Min/Max Score')
plt.xticks(rotation=90)
plt.legend()
plt.tight_layout()
plt.show()
```



5 Plot Total 4s and 6s

```
[64]: df_deliveries.columns
```

```
[64]: Index(['match_id', 'inning', 'batting_team', 'bowling_team', 'over', 'ball',
              'batter', 'bowler', 'non_striker', 'batsman_runs', 'extra_runs',
              'total_runs', 'extras_type', 'is_wicket', 'player_dismissed',
              'dismissal_kind', 'fielder'],
             dtype='object')
```

```
[65]: # find out unique values for batsman_runsdf
      df_deliveries['batsman_runs'].unique()
```

```
[65]: array([ 0.,  4.,  6.,  1.,  2.,  5.,  3., nan])
```

[66]:

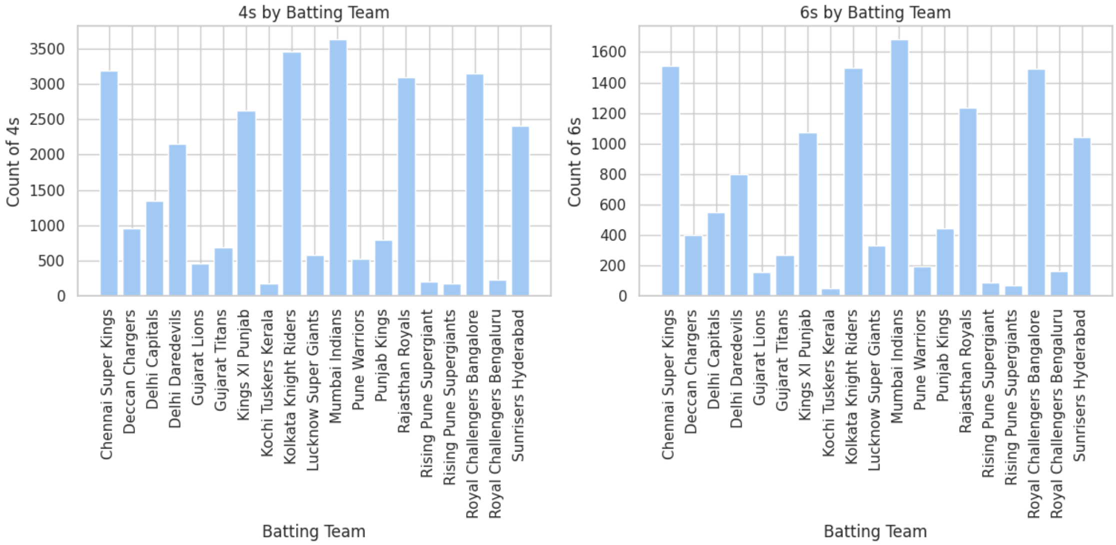
```
# Group by 'batting_team' and get value counts of 'batsman_runs'
batting_team_runs = df_deliveries.groupby('batting_team')['batsman_runs'].
 ↪value_counts()

# Display the result
print(batting_team_runs)
```

```
batting_team                 batsman_runs
Chennai Super Kings          0.0             1617
                             1.0             1421
                             4.0              476
                             2.0              239
                             6.0              184
                             3.0               13
Deccan Chargers              0.0             1683
                             1.0             1402
                             4.0              426
                             2.0              256
                             6.0              214
                             3.0               13
                             5.0                2
Delhi Daredevils             0.0             1444
                             1.0             1389
                             4.0              469
                             2.0              281
                             6.0              118
                             3.0               14
Kings XI Punjab              0.0             1614
                             1.0             1349
                             4.0              423
                             2.0              227
                             6.0              170
                             3.0               18
                             5.0                2
Kolkata Knight Riders        0.0             1619
                             1.0             1217
                             4.0              356
                             2.0              218
                             6.0              145
                             3.0               11
                             5.0                1
Mumbai Indians               0.0             1579
                             1.0             1210
                             4.0              400
                             2.0              215
                             6.0              147
                             3.0               20
                             5.0                3
```

```
Rajasthan Royals             0.0           1706
                             1.0           1328
                             4.0            464
                             2.0            240
                             6.0            151
                             3.0             16
Royal Challengers Bangalore  0.0           1743
                             1.0           1424
                             4.0            436
                             2.0            252
                             6.0            147
                             3.0             10
                             5.0              2
Name: count, dtype: int64
```

[85]:

```
# Filter for 4s and 6s separately
fours = df_deliveries[df_deliveries['batsman_runs'] == 4]
sixes = df_deliveries[df_deliveries['batsman_runs'] == 6]

# Group by 'batting_team' and count occurrences
fours_by_team = fours.groupby('batting_team')['batsman_runs'].count()
sixes_by_team = sixes.groupby('batting_team')['batsman_runs'].count()

# Plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)  # Subplot for 4s
plt.bar(fours_by_team.index, fours_by_team.values)
plt.xlabel('Batting Team')
plt.ylabel('Count of 4s')
plt.title('4s by Batting Team')
plt.xticks(rotation=90)

plt.subplot(1, 2, 2)  # Subplot for 6s
plt.bar(sixes_by_team.index, sixes_by_team.values)
plt.xlabel('Batting Team')
plt.ylabel('Count of 6s')
plt.title('6s by Batting Team')
plt.xticks(rotation=90)

plt.tight_layout()
plt.show()
```



6 Plot Average Powerplay and Death Overs Score

[68]:
```
df_deliveries['match_id'].value_counts()
```

[68]:
```
match_id
392190    267
335989    255
392218    255
419110    254
419107    254
         ...
336025    175
336021    136
336022    123
392183    108
419120     60
Name: count, Length: 130, dtype: int64
```

[73]:
```
df_deliveries = pd.read_csv('deliveries.csv')
#group by match_id
df_deliveries_cleaned = df_deliveries.groupby('match_id')
print('no_of_matches: ',df_deliveries_cleaned.size())

#group by batting_team and over
df_deliveries_cleaned = df_deliveries.groupby(['batting_team',␣
↪'over'])['total_runs'].sum().reset_index()
df_deliveries_cleaned
```

```
no_of_matches:  match_id
335982     225
335983     248
335984     219
335985     246
335986     240
          ...
1426307    247
1426309    208
1426310    241
1426311    251
1426312    184
Length: 1095, dtype: int64
```

[73]:

```
            batting_team  over  total_runs
0    Chennai Super Kings     0        1252
1    Chennai Super Kings     1        1608
2    Chennai Super Kings     2        1838
3    Chennai Super Kings     3        2010
4    Chennai Super Kings     4        2093
..                   ...   ...         ...
375  Sunrisers Hyderabad    15        1435
376  Sunrisers Hyderabad    16        1480
377  Sunrisers Hyderabad    17        1589
378  Sunrisers Hyderabad    18        1710
379  Sunrisers Hyderabad    19        1458

[380 rows x 3 columns]
```

[74]:

```
# calculate average total_runs where over=[17,18,19,20] for each team and plot
df_deliveries_cleaned = df_deliveries_cleaned[df_deliveries_cleaned['over'].
 ↪isin([0,1,2,3,4,5])]
df_deliveries_cleaned
```

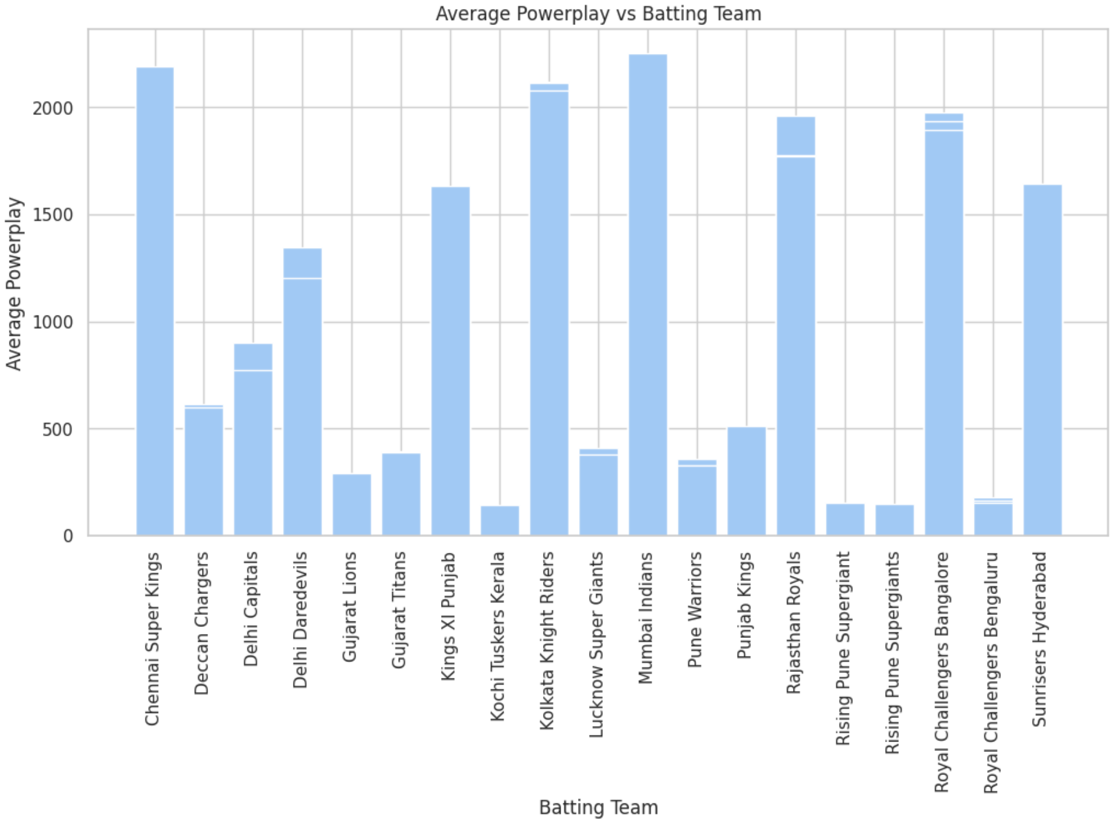
[74]:

```
            batting_team  over  total_runs
0    Chennai Super Kings     0        1252
1    Chennai Super Kings     1        1608
2    Chennai Super Kings     2        1838
3    Chennai Super Kings     3        2010
4    Chennai Super Kings     4        2093
..                   ...   ...         ...
361  Sunrisers Hyderabad     1        1468
362  Sunrisers Hyderabad     2        1570
363  Sunrisers Hyderabad     3        1489
364  Sunrisers Hyderabad     4        1644
365  Sunrisers Hyderabad     5        1643
```

```
[114 rows x 3 columns]
```

[75]:

```
# now plot batting_team vs total_runs
plt.figure(figsize=(12, 6))
plt.bar(df_deliveries_cleaned['batting_team'],␣
 ↪df_deliveries_cleaned['total_runs'])
plt.xlabel('Batting Team')
plt.ylabel('Average Powerplay')
plt.title('Average Powerplay vs Batting Team')
plt.xticks(rotation=90)
plt.show()
```

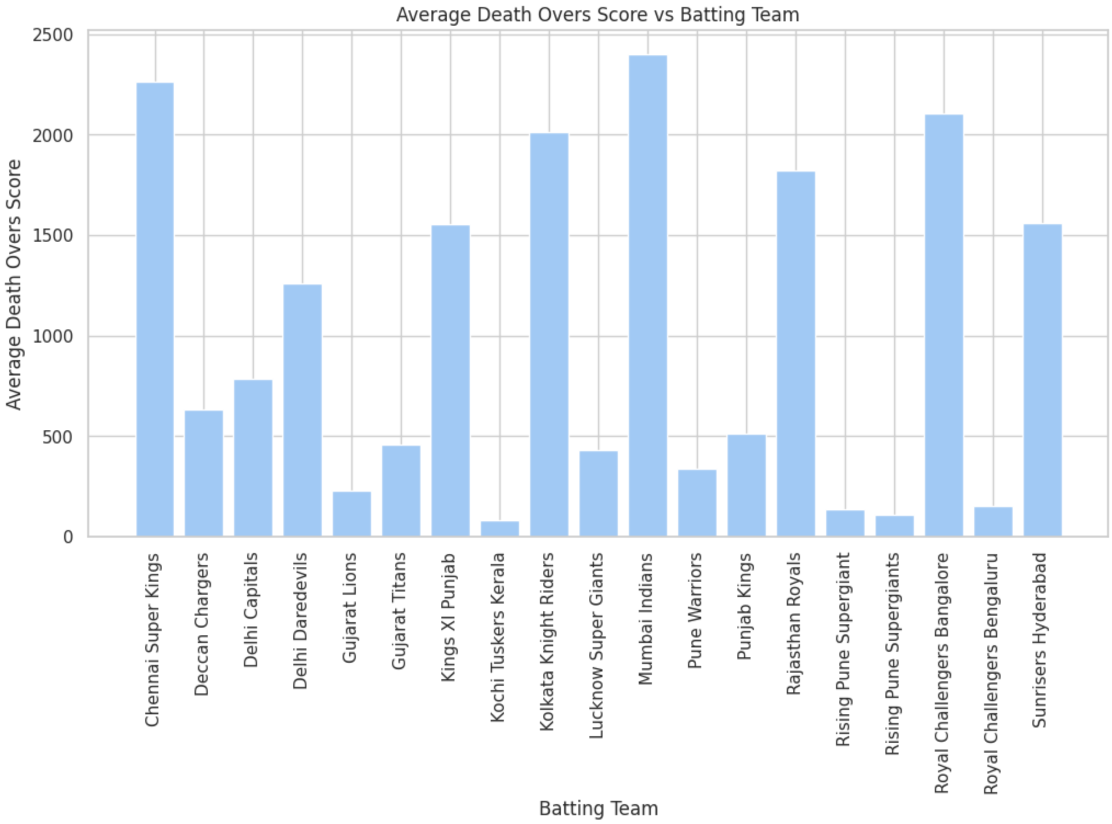


Calculate powerplay

[78]:

```
df_deliveries = pd.read_csv('deliveries.csv')
#group by match_id
df_deliveries_cleaned = df_deliveries.groupby('match_id')
#print('no_of_matches: ',df_deliveries_cleaned.size())

#group by batting_team and over
```

```
df_deliveries_cleaned = df_deliveries.groupby(['batting_team',␣
 ↪'over'])['total_runs'].sum().reset_index()
#df_deliveries_cleaned


df_deliveries_cleaned = df_deliveries_cleaned[df_deliveries_cleaned['over'].
 ↪isin([16,17,18,19])]

df_deliveries_cleaned = df_deliveries_cleaned.
 ↪groupby('batting_team')['total_runs'].mean().reset_index()


plt.figure(figsize=(12, 6))
plt.bar(df_deliveries_cleaned['batting_team'],␣
 ↪df_deliveries_cleaned['total_runs'])
plt.xlabel('Batting Team')
plt.ylabel('Average Death Overs Score')
plt.title('Average Death Overs Score vs Batting Team')
plt.xticks(rotation=90)
plt.show()
```



[79]:

```
# Average run per over of each team
df_deliveries = pd.read_csv('deliveries.csv')
df_deliveries_cleaned = df_deliveries.groupby(['batting_team',␣
 ↪'over'])['total_runs'].sum().reset_index()
```

[80]:

```
df_deliveries_cleaned['average_runs'] = df_deliveries_cleaned['total_runs'] / 20
print(df_deliveries_cleaned)
```

```
            batting_team  over  total_runs  average_runs
0    Chennai Super Kings     0        1252         62.60
1    Chennai Super Kings     1        1608         80.40
2    Chennai Super Kings     2        1838         91.90
3    Chennai Super Kings     3        2010        100.50
4    Chennai Super Kings     4        2093        104.65
..                   ...   ...         ...           ...
375  Sunrisers Hyderabad    15        1435         71.75
376  Sunrisers Hyderabad    16        1480         74.00
377  Sunrisers Hyderabad    17        1589         79.45
378  Sunrisers Hyderabad    18        1710         85.50
379  Sunrisers Hyderabad    19        1458         72.90

[380 rows x 4 columns]
```

7 Player Performance:

Get the top 20 run-scorers

[86]:

```
#match_id, batter, total_runs
#groupby match_id, batter and sum total_runs
df_deliveries = pd.read_csv('deliveries.csv')
df_deliveries_cleaned = df_deliveries.groupby(['match_id',␣
 ↪'batter'])['total_runs'].sum().reset_index()
df_deliveries_cleaned
```

[86]:

```
       match_id          batter  total_runs
0        335982       AA Noffke          11
1        335982         B Akhil           0
2        335982     BB McCullum         169
3        335982        CL White           6
4        335982       DJ Hussey          12
...         ...             ...         ...
16510   1426312       SP Narine           6
16511   1426312         SS Iyer           6
16512   1426312  Shahbaz Ahmed           8
16513   1426312         TM Head           0
16514   1426312         VR Iyer          56
```

```
[16515 rows x 3 columns]
```

[87]:

```
# sort df_deliveries_cleaned according to total_runs
df_deliveries_cleaned = df_deliveries_cleaned.sort_values(by='total_runs',␣
 ↪ascending=False)
df_deliveries_cleaned
```

[87]:

```
       match_id            batter  total_runs
5302     598027          CH Gayle         181
2        335982      BB McCullum         169
14108   1304112         Q de Kock         141
11583   1216510          KL Rahul         140
7528     829795    AB de Villiers         138
...         ...               ...         ...
1801     419108        TM Dilshan           0
1788     419108  DPMD Jayawardene           0
16430   1426306        RD Gaikwad           0
1718     392237       PJ Sangwan           0
7553     829797          R Bhatia           0

[16515 rows x 3 columns]
```

[88]:

```
# find out top 20 run-scorers from df_deliveries_cleaned
df_deliveries_top20 = df_deliveries_cleaned[:20]
df_deliveries_top20
```

[88]:

```
       match_id          batter  total_runs
5302     598027        CH Gayle         181
2        335982     BB McCullum         169
14108   1304112       Q de Kock         141
11583   1216510        KL Rahul         140
7528     829795  AB de Villiers         138
14915   1359516     YBK Jaiswal         134
15383   1370352    Shubman Gill         133
8359     980987  AB de Villiers         132
12571   1254085      JC Buttler         131
3571     501260    AC Gilchrist         130
10149   1136602         RR Pant         130
6854     734047        V Sehwag         129
16000   1426277      MP Stoinis         129
15871   1426269       SP Narine         129
4687     548372        CH Gayle         129
2237     419137         M Vijay         128
9146    1082627       DA Warner         127
7460     829785        CH Gayle         126
12221   1254061       SV Samson         124
15689   1422137         V Kohli         123
```

Plot top wicket-takers

```
[89]: df_deliveries = pd.read_csv('deliveries.csv')
      df_deliveries_cleaned = df_deliveries.groupby(['bowling_team',␣
       ↪'bowler'])['is_wicket'].sum().reset_index()
      df_deliveries_cleaned.sort_values(by='is_wicket', ascending=False)
```

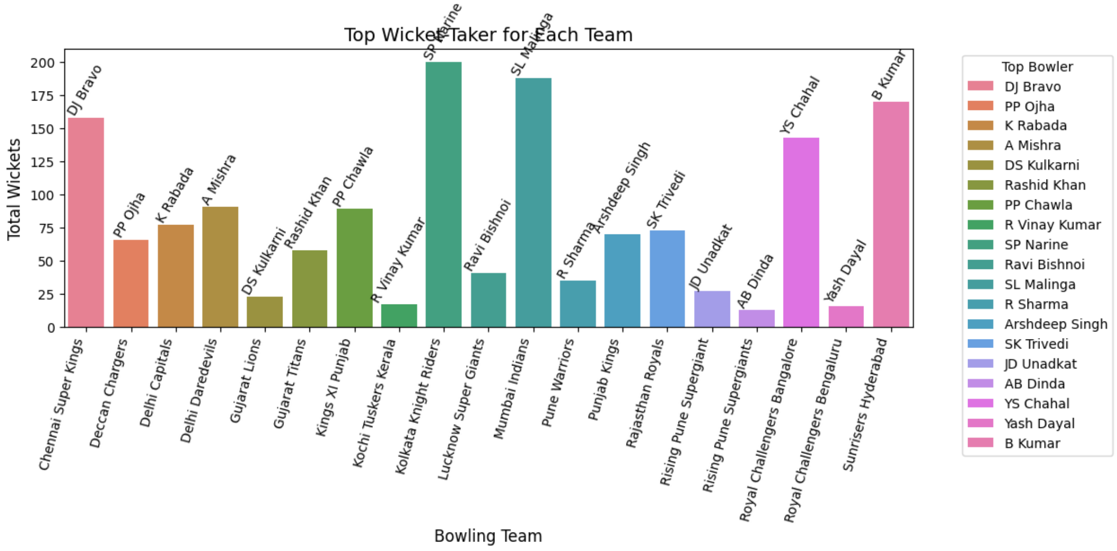
```
[89]:               bowling_team            bowler  is_wicket
      456  Kolkata Knight Riders          SP Narine        200
      595         Mumbai Indians         SL Malinga        188
      542         Mumbai Indians          JJ Bumrah        182
      940    Sunrisers Hyderabad           B Kumar        170
      14     Chennai Super Kings          DJ Bravo        158
      ..                     ...               ...        ...
      962    Sunrisers Hyderabad      KS Williamson          0
      77     Chennai Super Kings          V Shankar          0
      998    Sunrisers Hyderabad   Y Venugopal Rao          0
      103        Deccan Chargers          LPC Silva          0
      989    Sunrisers Hyderabad     Shashank Singh          0

      [1001 rows x 3 columns]
```

```
[90]: # Find the index of the bowler with the highest wickets for each team
      idx = df_deliveries_cleaned.groupby('bowling_team')['is_wicket'].idxmax()

      # Use these indices to get the corresponding bowlers
      top_bowlers = df_deliveries_cleaned.loc[idx].reset_index(drop=True)

      print(top_bowlers)
```

```
                   bowling_team          bowler  is_wicket
0           Chennai Super Kings        DJ Bravo        158
1               Deccan Chargers         PP Ojha         66
2                Delhi Capitals        K Rabada         77
3              Delhi Daredevils        A Mishra         91
4                 Gujarat Lions     DS Kulkarni         23
5                Gujarat Titans     Rashid Khan         58
6               Kings XI Punjab       PP Chawla         89
7          Kochi Tuskers Kerala   R Vinay Kumar         17
8         Kolkata Knight Riders       SP Narine        200
9          Lucknow Super Giants    Ravi Bishnoi         41
10               Mumbai Indians      SL Malinga        188
11                Pune Warriors       R Sharma         35
12                 Punjab Kings  Arshdeep Singh         70
13             Rajasthan Royals      SK Trivedi         73
14       Rising Pune Supergiant      JD Unadkat         27
15      Rising Pune Supergiants        AB Dinda         13
16  Royal Challengers Bangalore       YS Chahal        143
```

```
17  Royal Challengers Bengaluru      Yash Dayal          16
18          Sunrisers Hyderabad         B Kumar         170
```

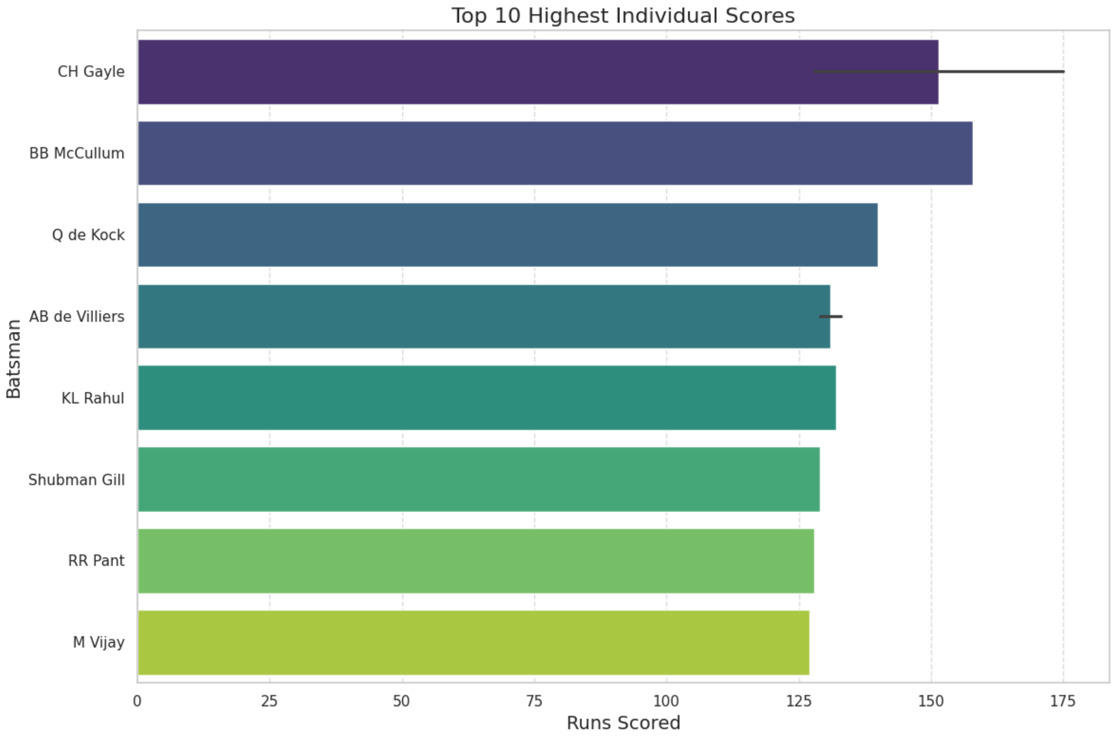
```
[ ]: import matplotlib.pyplot as plt
     import seaborn as sns

     # Set figure size
     plt.figure(figsize=(12, 6))

     # Create bar plot
     ax = sns.barplot(data=top_bowlers, x='bowling_team', y='is_wicket',␣
      ↪hue='bowler', dodge=False)

     # Annotate each bar with the bowler's name
     for index, row in top_bowlers.iterrows():
         plt.text(index, row.is_wicket + 2, row.bowler,
                  ha='center', fontsize=10, color='black', rotation=60)

     # Improve readability
     plt.xlabel("Bowling Team", fontsize=12)
     plt.ylabel("Total Wickets", fontsize=12)
     plt.title("Top Wicket-Taker for Each Team", fontsize=14)
     plt.xticks(rotation=75, ha='right')  # Rotate & align labels

     # Adjust legend placement
     plt.legend(title="Top Bowler", bbox_to_anchor=(1.05, 1), loc='upper left',␣
      ↪fontsize=10)

     # Show the plot
     plt.tight_layout()  # Prevent labels from being cut off
     plt.show();
```



Plot top highest individual scores

```
[91]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot_highest_individual_scores(df_deliveries, top_n=10):
    # Group by match_id and batter to get individual scores per match
    batsman_scores = df_deliveries.groupby(['match_id',␣
 ↪'batter'])['batsman_runs'].sum().reset_index()

    # Sort by runs in descending order and get top N
    top_scores = batsman_scores.sort_values('batsman_runs', ascending=False).
 ↪head(top_n)

    plt.figure(figsize=(12, 8))
    sns.barplot(x='batsman_runs', y='batter', data=top_scores, palette='viridis')
    plt.title(f'Top {top_n} Highest Individual Scores', fontsize=16)
    plt.xlabel('Runs Scored', fontsize=14)
    plt.ylabel('Batsman', fontsize=14)
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()

    return top_scores


top_scores = plot_highest_individual_scores(df_deliveries)
```

```
<ipython-input-91-36c6c987af57>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='batsman_runs', y='batter', data=top_scores, palette='viridis')
```

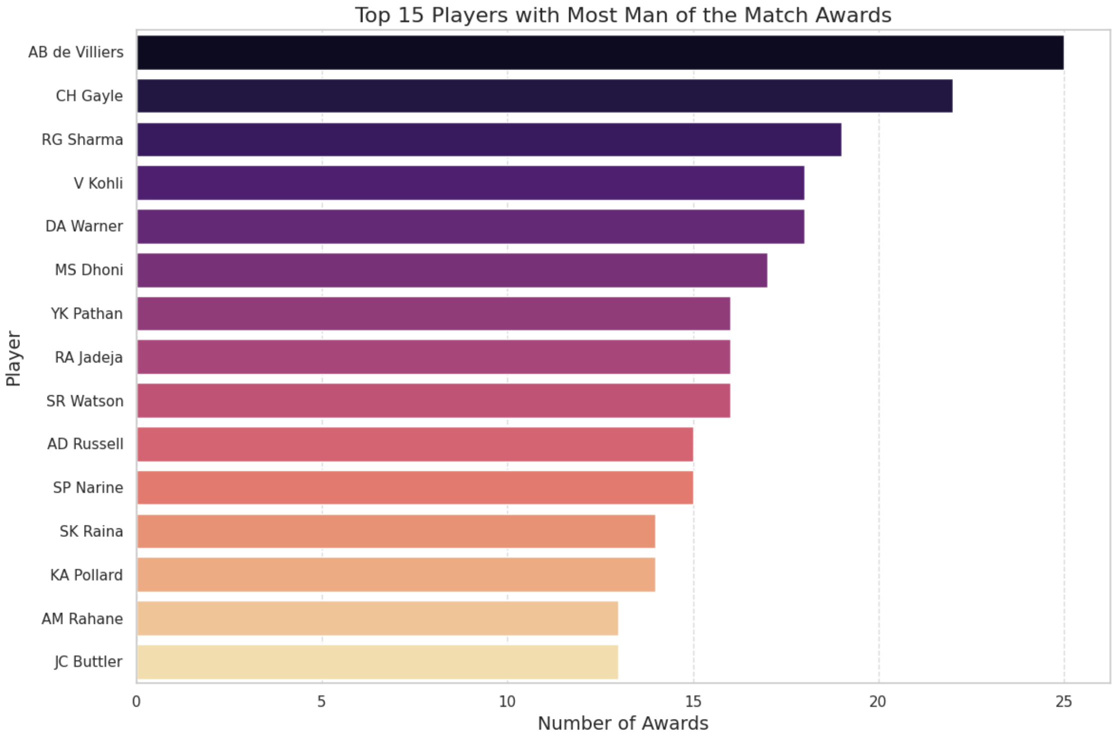


Man of the Match Count Analysis

[92]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot_man_of_match_count(df_matches, top_n=15):
    # Count man of the match awards for each player
    mom_counts = df_matches['player_of_match'].value_counts().reset_index()
    mom_counts.columns = ['Player', 'MoM Count']

    # Get top N players with most MoM awards
    top_mom = mom_counts.head(top_n)

    plt.figure(figsize=(12, 8))
    sns.barplot(x='MoM Count', y='Player', data=top_mom, palette='magma')
    plt.title(f'Top {top_n} Players with Most Man of the Match Awards',␣
 ↪fontsize=16)
    plt.xlabel('Number of Awards', fontsize=14)
    plt.ylabel('Player', fontsize=14)
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()
    return top_mom
top_mom = plot_man_of_match_count(df_matches)
```

```
<ipython-input-92-cc170482ae94>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='MoM Count', y='Player', data=top_mom, palette='magma')
```

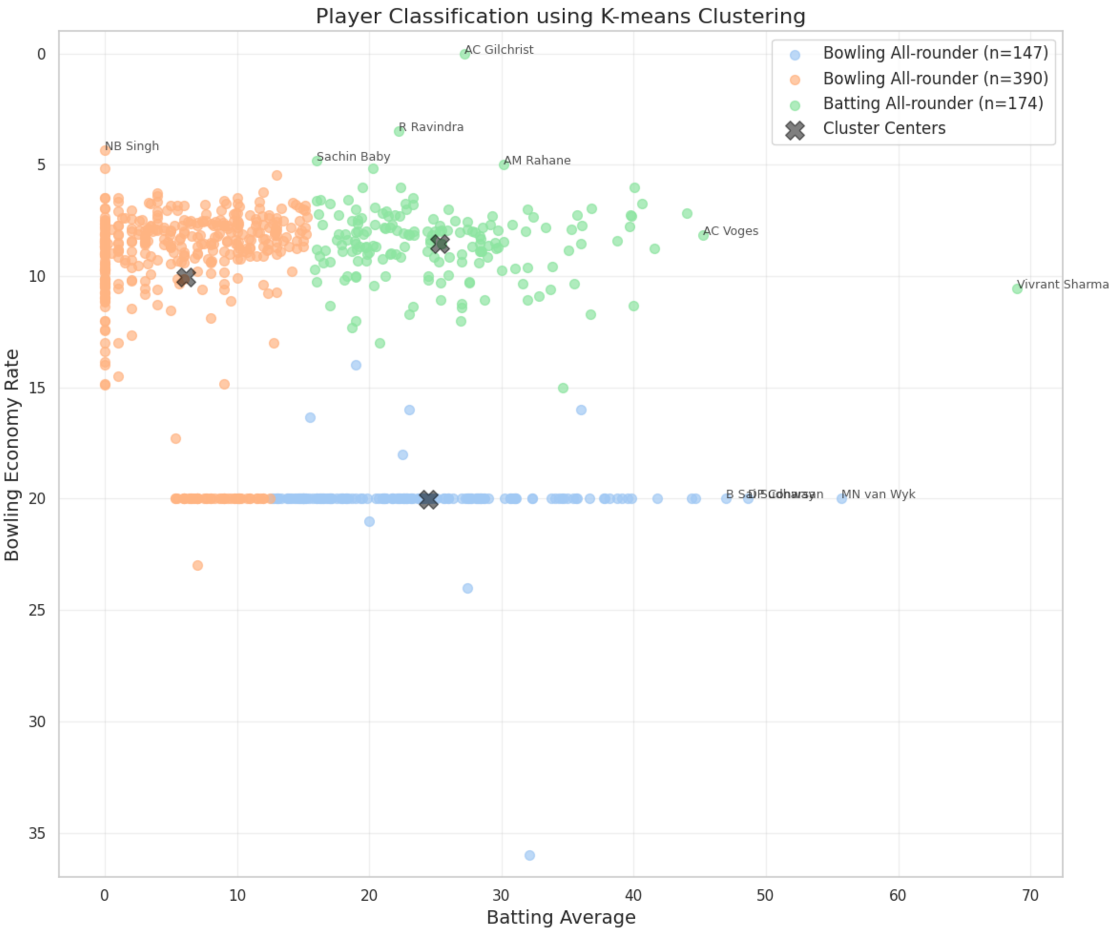


Use K-Means Clustering to plot Batting Average vs Bowling Economy Rate for number of clusters = 3 (Batsman, Bowler, All Rounder)

[93]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

def plot_player_clustering(df_deliveries, n_clusters=3):
    # Calculate batting averages
    # First get total runs for each batsman
    batsman_runs = df_deliveries.groupby('batter')['batsman_runs'].sum().
↪reset_index()

    # Get dismissal count for each batsman
    dismissals = df_deliveries[df_deliveries['is_wicket'] == 1].
↪groupby('player_dismissed').size().reset_index()
    dismissals.columns = ['batter', 'dismissals']

    # Merge runs and dismissals
    batting_stats = pd.merge(batsman_runs, dismissals, on='batter', how='left')
    batting_stats['dismissals'] = batting_stats['dismissals'].fillna(0)

    # Calculate batting average (runs/dismissals)
    batting_stats['batting_avg'] = batting_stats['batsman_runs'] /␣
↪batting_stats['dismissals'].replace(0, 1)

    # Calculate bowling economy rate
    # First get total runs conceded by each bowler
    bowling_runs = df_deliveries.groupby('bowler')['total_runs'].sum().
↪reset_index()

    # Get total balls bowled by each bowler
    bowling_balls = df_deliveries.groupby('bowler').size().reset_index()
    bowling_balls.columns = ['bowler', 'balls']

    # Merge runs and balls
    bowling_stats = pd.merge(bowling_runs, bowling_balls, on='bowler',␣
↪how='left')

    # Calculate economy rate (runs per over = runs / (balls/6))
    bowling_stats['economy_rate'] = (bowling_stats['total_runs'] /␣
↪(bowling_stats['balls']/6)).round(2)

    # Merge batting and bowling stats
    # Use outer join to include all players
    player_stats = pd.merge(batting_stats[['batter', 'batting_avg']],
                          bowling_stats[['bowler', 'economy_rate']],
                          left_on='batter', right_on='bowler',
                          how='outer')

    # Clean up and prepare for clustering
    player_stats['name'] = player_stats['batter'].fillna(player_stats['bowler'])
    player_stats['batting_avg'] = player_stats['batting_avg'].fillna(0)
    player_stats['economy_rate'] = player_stats['economy_rate'].fillna(20)  #␣
↪High economy for pure batsmen
```

```
    # Filter for minimum qualification (players with some meaningful stats)
    qualified_players = player_stats[(player_stats['batting_avg'] > 5) | (player_stats['economy_rate'] < 15)]

    # Prepare data for clustering
    X = qualified_players[['batting_avg', 'economy_rate']].copy()

    # Cap extremely high batting averages for better clustering
    X['batting_avg'] = X['batting_avg'].clip(upper=100)

    # Perform K-means clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    qualified_players['cluster'] = kmeans.fit_predict(X)

    # Add role labels based on clustering
    # Determine roles based on cluster centroids
    centroids = kmeans.cluster_centers_
    roles = []
    for i in range(n_clusters):
        if centroids[i, 0] > 25 and centroids[i, 1] > 8:
            roles.append('Batting All-rounder')
        elif centroids[i, 0] > 25:
            roles.append('Batsman')
        elif centroids[i, 1] < 8:
            roles.append('Bowler')
        else:
            roles.append('Bowling All-rounder')

    # Map cluster to roles
    cluster_role_map = {i: role for i, role in enumerate(roles)}
    qualified_players['role'] = qualified_players['cluster'].map(cluster_role_map)

    # Visualization
    plt.figure(figsize=(12, 10))

    # Create scatter plot with different colors for different clusters
    for cluster, role in cluster_role_map.items():
        cluster_data = qualified_players[qualified_players['cluster'] == cluster]
        plt.scatter(cluster_data['batting_avg'], cluster_data['economy_rate'],
                    label=f'{role} (n={len(cluster_data)})', alpha=0.7, s=50)

    # Plot cluster centers
    plt.scatter(centroids[:, 0], centroids[:, 1], c='black', s=200, alpha=0.5, marker='X', label='Cluster Centers')

    # Annotate some notable players
    top_players = qualified_players.nlargest(5, 'batting_avg')
    for _, player in top_players.iterrows():
        plt.annotate(player['name'], (player['batting_avg'],␣
 ↪player['economy_rate']),
                    fontsize=9, alpha=0.8)

    # Also annotate top bowlers
    top_bowlers = qualified_players.nsmallest(5, 'economy_rate')
    for _, player in top_bowlers.iterrows():
        plt.annotate(player['name'], (player['batting_avg'],␣
 ↪player['economy_rate']),
                    fontsize=9, alpha=0.8)

    plt.title('Player Classification using K-means Clustering', fontsize=16)
    plt.xlabel('Batting Average', fontsize=14)
    plt.ylabel('Bowling Economy Rate', fontsize=14)
    plt.legend(fontsize=12)
    plt.grid(True, alpha=0.3)

    # Invert y-axis as lower economy rate is better
    plt.ylim(max(qualified_players['economy_rate'])+1,␣
 ↪min(qualified_players['economy_rate'])-1)

    plt.tight_layout()
    plt.show()

    return qualified_players


player_clusters = plot_player_clustering(df_deliveries)
```

```
<ipython-input-93-08fe77968299>:60: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  qualified_players['cluster'] = kmeans.fit_predict(X)
<ipython-input-93-08fe77968299>:78: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  qualified_players['role'] = qualified_players['cluster'].map(cluster_role_map)
```

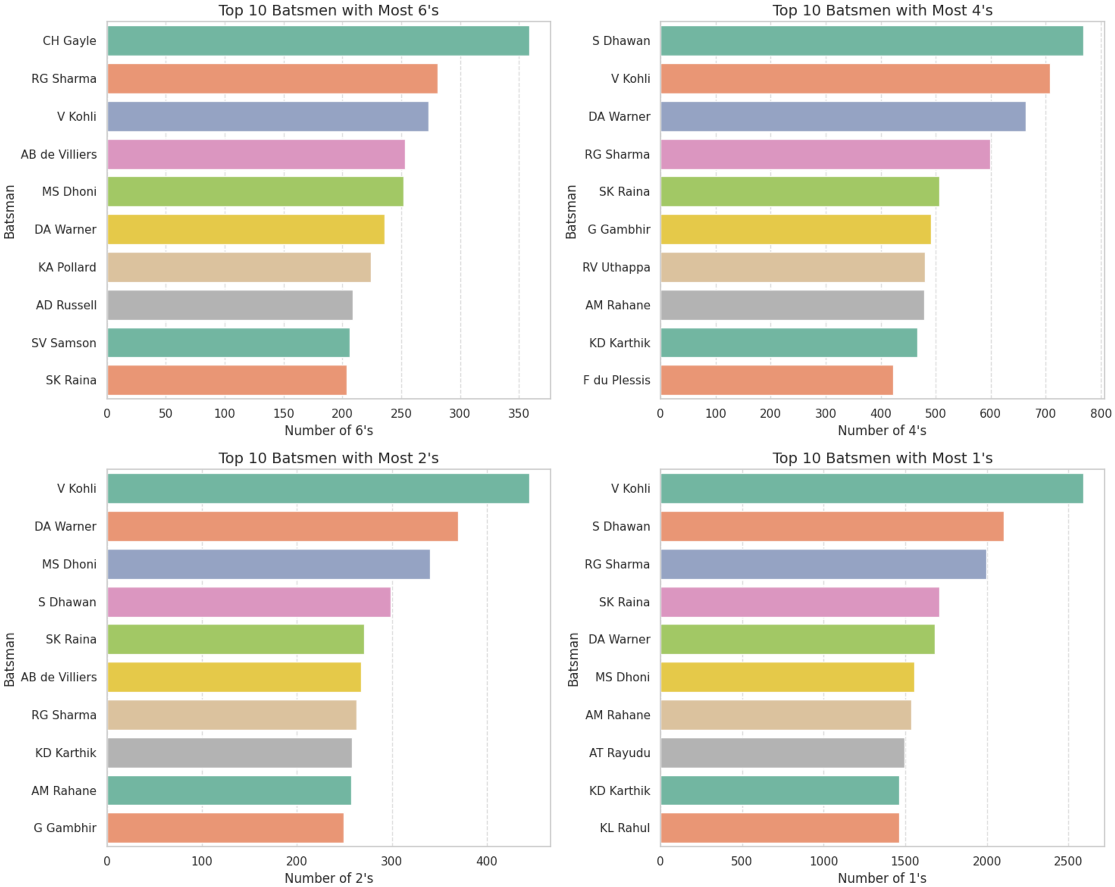


Identify Top 10 Batsmen in each run category: Top 6's scorer Top 4's scorer Top 2's scorer Top 1's scorer

[94]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot_top_run_scorers_by_category(df_deliveries, top_n=10):
    # Create a dataframe to store the counts of different run categories
    run_categories = {
        "6's": 6,
        "4's": 4,
        "2's": 2,
        "1's": 1
    }

    # Create subplots
    fig, axes = plt.subplots(2, 2, figsize=(15, 12))
    axes = axes.flatten()

    results = {}

    for i, (category, run_value) in enumerate(run_categories.items()):
        # Count the number of times each batsman scored this run value
        run_counts = df_deliveries[df_deliveries['batsman_runs'] ==␣
↪run_value]['batter'].value_counts().reset_index()
        run_counts.columns = ['Batsman', f'Number of {category}']

        # Get top N batsmen
        top_batsmen = run_counts.head(top_n)
        results[category] = top_batsmen

        # Plot
        sns.barplot(x=f'Number of {category}', y='Batsman', data=top_batsmen,␣
↪ax=axes[i], palette='Set2')
        axes[i].set_title(f'Top {top_n} Batsmen with Most {category}',␣
↪fontsize=14)
        axes[i].set_xlabel(f'Number of {category}', fontsize=12)
        axes[i].set_ylabel('Batsman', fontsize=12)
        axes[i].grid(axis='x', linestyle='--', alpha=0.7)

    plt.tight_layout()
    plt.show()

    return results

top_run_scorers = plot_top_run_scorers_by_category(df_deliveries)
```

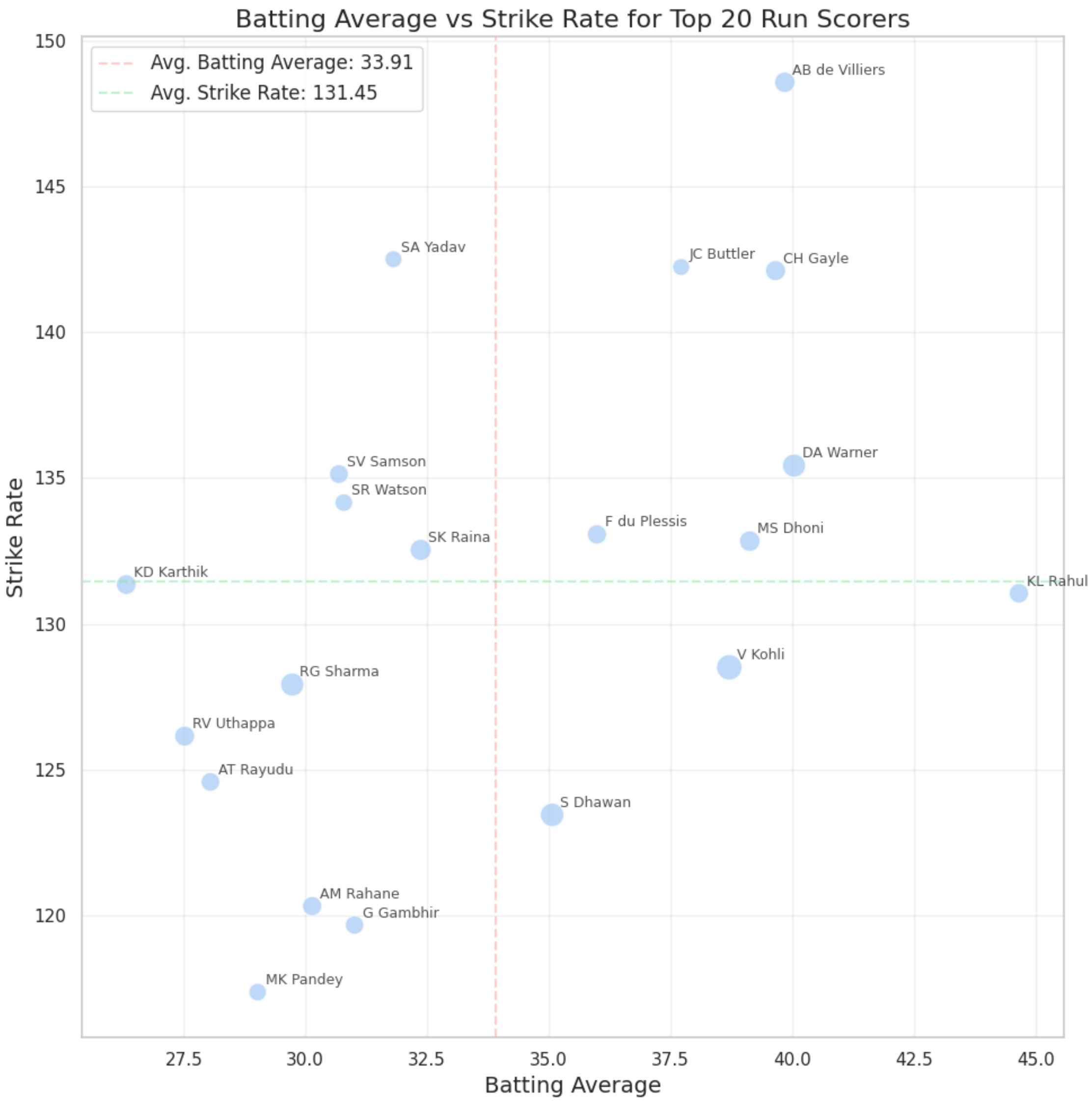
```
<ipython-input-94-e39b34a81cff>:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=f'Number of {category}', y='Batsman', data=top_batsmen,
ax=axes[i], palette='Set2')
<ipython-input-94-e39b34a81cff>:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=f'Number of {category}', y='Batsman', data=top_batsmen,
ax=axes[i], palette='Set2')
<ipython-input-94-e39b34a81cff>:30: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=f'Number of {category}', y='Batsman', data=top_batsmen,
ax=axes[i], palette='Set2')
<ipython-input-94-e39b34a81cff>:30: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=f'Number of {category}', y='Batsman', data=top_batsmen,
ax=axes[i], palette='Set2')
```



Plot Batting Average vs Batting Strike Rate for the top 20 run-scorers

```
[95]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def plot_avg_vs_strike_rate(df_deliveries, top_n=20):
    # Calculate total runs for each batsman
    batsman_runs = df_deliveries.groupby('batter')['batsman_runs'].sum().
 ↪reset_index()
    batsman_runs = batsman_runs.sort_values('batsman_runs', ascending=False)
    top_run_scorers = batsman_runs.head(top_n)['batter'].tolist()

    # Calculate batting average
    # Get dismissal count for each batsman
    dismissals = df_deliveries[df_deliveries['is_wicket'] == 1].
 ↪groupby('player_dismissed').size().reset_index()
    dismissals.columns = ['batter', 'dismissals']

    # Calculate total balls faced by each batsman
    balls_faced = df_deliveries.groupby('batter').size().reset_index()
    balls_faced.columns = ['batter', 'balls_faced']

    # Merge all stats
    batting_stats = pd.merge(batsman_runs, dismissals, on='batter', how='left')
    batting_stats = pd.merge(batting_stats, balls_faced, on='batter', how='left')

    # Fill NaN values for dismissals (players who were never out)
    batting_stats['dismissals'] = batting_stats['dismissals'].fillna(1)

    # Calculate batting average and strike rate
    batting_stats['batting_avg'] = (batting_stats['batsman_runs'] /␣
 ↪batting_stats['dismissals']).round(2)
    batting_stats['strike_rate'] = (batting_stats['batsman_runs'] /␣
 ↪batting_stats['balls_faced'] * 100).round(2)

    # Filter for top run scorers only
    top_batsmen_stats = batting_stats[batting_stats['batter'].
 ↪isin(top_run_scorers)]

    # Visualization
    plt.figure(figsize=(10, 10))

    # Create scatter plot
    sns.scatterplot(x='batting_avg', y='strike_rate', data=top_batsmen_stats,
                   s=top_batsmen_stats['batsman_runs']/30, alpha=0.7)

    # Add labels for each point
    for _, player in top_batsmen_stats.iterrows():
        plt.annotate(player['batter'],
                    (player['batting_avg'], player['strike_rate']),
                    fontsize=9, alpha=0.8,
                    xytext=(5, 5), textcoords='offset points')

    plt.title(f'Batting Average vs Strike Rate for Top {top_n} Run Scorers',␣
 ↪fontsize=16)
    plt.xlabel('Batting Average', fontsize=14)
    plt.ylabel('Strike Rate', fontsize=14)
    plt.grid(True, alpha=0.3)

    # Add reference lines for average values
    plt.axvline(x=top_batsmen_stats['batting_avg'].mean(), color='r',␣
 ↪linestyle='--', alpha=0.5,
               label=f'Avg. Batting Average: {top_batsmen_stats["batting_avg"].
 ↪mean():.2f}')
    plt.axhline(y=top_batsmen_stats['strike_rate'].mean(), color='g',␣
 ↪linestyle='--', alpha=0.5,
               label=f'Avg. Strike Rate: {top_batsmen_stats["strike_rate"].
 ↪mean():.2f}')

    plt.legend(fontsize=12)
    plt.tight_layout()
    plt.show()

    return top_batsmen_stats

top_batsmen_stats = plot_avg_vs_strike_rate(df_deliveries)
```

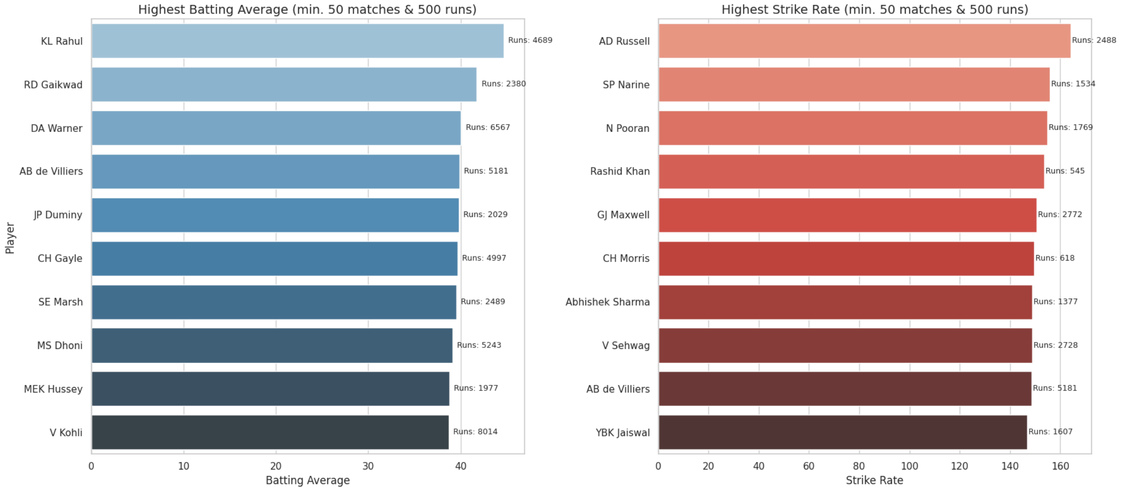


Find Highest Average and Strike Rate for players with >50 matches

[96]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def find_best_experienced_players(df_deliveries, df_matches, min_matches=50):
    # Find players who have played in more than min_matches
    # First, identify unique players in each match
    player_matches = set()

    # Add batsmen
    for batter in df_deliveries['batter'].unique():
        player_matches.add(batter)

    # Add bowlers
    for bowler in df_deliveries['bowler'].unique():
        player_matches.add(bowler)

    # Count matches for each player
    player_match_counts = {}

    for player in player_matches:
        # Count matches as batsman
        batsman_matches = df_deliveries[df_deliveries['batter'] ==␣
↪player]['match_id'].nunique()

        # Count matches as bowler
        bowler_matches = df_deliveries[df_deliveries['bowler'] ==␣
↪player]['match_id'].nunique()

        # Take maximum of the two (to avoid double counting)
        player_match_counts[player] = max(batsman_matches, bowler_matches)

    # Convert to DataFrame
    player_experience = pd.DataFrame(list(player_match_counts.items()),␣
↪columns=['player', 'matches_played'])

    # Filter players with more than min_matches
    experienced_players = player_experience[player_experience['matches_played']␣
↪>= min_matches]['player'].tolist()

    print(f"Found {len(experienced_players)} players with {min_matches}+␣
↪matches")

    # Calculate batting stats for experienced players
    # Calculate total runs for each batsman
    batsman_runs = df_deliveries.groupby('batter')['batsman_runs'].sum().
↪reset_index()

    # Get dismissal count for each batsman
    dismissals = df_deliveries[df_deliveries['is_wicket'] == 1].
↪groupby('player_dismissed').size().reset_index()
    dismissals.columns = ['batter', 'dismissals']

    # Calculate total balls faced by each batsman
    balls_faced = df_deliveries.groupby('batter').size().reset_index()
    balls_faced.columns = ['batter', 'balls_faced']
    # Merge all stats
    batting_stats = pd.merge(batsman_runs, dismissals, on='batter', how='left')
    batting_stats = pd.merge(batting_stats, balls_faced, on='batter', how='left')

    # Fill NaN values for dismissals (players who were never out)
    batting_stats['dismissals'] = batting_stats['dismissals'].fillna(1)

    # Filter for experienced players only
    exp_batting_stats = batting_stats[batting_stats['batter'].
↪isin(experienced_players)]

    # Calculate batting average and strike rate
    exp_batting_stats['batting_avg'] = (exp_batting_stats['batsman_runs'] /␣
↪exp_batting_stats['dismissals']).round(2)
    exp_batting_stats['strike_rate'] = (exp_batting_stats['batsman_runs'] /␣
↪exp_batting_stats['balls_faced'] * 100).round(2)

    # Find players with highest batting average and strike rate
    min_runs_threshold = 500  # Minimum runs to qualify
    qualified_stats = exp_batting_stats[exp_batting_stats['batsman_runs'] >=␣
↪min_runs_threshold]

    # Sort by batting average and strike rate
    best_avg = qualified_stats.sort_values('batting_avg', ascending=False).
↪head(10)
    best_sr = qualified_stats.sort_values('strike_rate', ascending=False).
↪head(10)

    # Create visualization - two bar charts side by side
    fig, axes = plt.subplots(1, 2, figsize=(18, 8))

    # Plot for best batting average
    sns.barplot(x='batting_avg', y='batter', data=best_avg, palette='Blues_d',␣
↪ax=axes[0])
    axes[0].set_title(f'Highest Batting Average (min. {min_matches} matches &␣
↪{min_runs_threshold} runs)', fontsize=14)
    axes[0].set_xlabel('Batting Average', fontsize=12)
    axes[0].set_ylabel('Player', fontsize=12)

    # Add run information
    for i, row in enumerate(best_avg.itertuples()):
        axes[0].text(row.batting_avg + 0.5, i, f'Runs: {row.batsman_runs}',
                     va='center', fontsize=9)

    # Plot for best strike rate
    sns.barplot(x='strike_rate', y='batter', data=best_sr, palette='Reds_d',␣
↪ax=axes[1])
    axes[1].set_title(f'Highest Strike Rate (min. {min_matches} matches &␣
↪{min_runs_threshold} runs)', fontsize=14)
    axes[1].set_xlabel('Strike Rate', fontsize=12)
    axes[1].set_ylabel('', fontsize=12)  # No need to repeat y-label

    # Add run information
    for i, row in enumerate(best_sr.itertuples()):
        axes[1].text(row.strike_rate + 0.5, i, f'Runs: {row.batsman_runs}',
                    va='center', fontsize=9)

    plt.tight_layout()
    plt.show()

    return {
        'highest_average': best_avg,
        'highest_strike_rate': best_sr
    }

experienced_stats = find_best_experienced_players(df_deliveries, df_matches,␣
↪min_matches=50)
```

```
Found 157 players with 50+ matches

<ipython-input-96-7511fe5b2576>:62: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  exp_batting_stats['batting_avg'] = (exp_batting_stats['batsman_runs'] /
exp_batting_stats['dismissals']).round(2)
<ipython-input-96-7511fe5b2576>:63: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  exp_batting_stats['strike_rate'] = (exp_batting_stats['batsman_runs'] /
exp_batting_stats['balls_faced'] * 100).round(2)
<ipython-input-96-7511fe5b2576>:77: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='batting_avg', y='batter', data=best_avg, palette='Blues_d',
ax=axes[0])
<ipython-input-96-7511fe5b2576>:88: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='strike_rate', y='batter', data=best_sr, palette='Reds_d',
ax=axes[1])
```



8 Seasonal Analysis

[100]:
```
#merge by match_id and id
match=pd.read_csv('matches.csv')
delivery=pd.read_csv('/content/deliveries.csv')
total_score_df=delivery.groupby(['match_id','inning']).sum()['total_runs'].
 ↪reset_index()
match_df=match.
 ↪merge(total_score_df[['match_id','total_runs']],left_on='id',right_on='match_id')
season_dfs = {season: match_df[match_df["season"] == season] for season in␣
 ↪match_df["season"].unique()}
season_dfs["2023"].head()
```

[100]:
```
           id season        city        date match_type player_of_match  \
1928  1359475   2023   Ahmedabad  2023-03-31     League     Rashid Khan
1929  1359475   2023   Ahmedabad  2023-03-31     League     Rashid Khan
1930  1359476   2023  Chandigarh  2023-04-01     League  Arshdeep Singh
1931  1359476   2023  Chandigarh  2023-04-01     League  Arshdeep Singh
1932  1359477   2023     Lucknow  2023-04-01     League         MA Wood
```

```
                                                  venue                 team1  \
1928                   Narendra Modi Stadium, Ahmedabad    Chennai Super Kings
1929                   Narendra Modi Stadium, Ahmedabad    Chennai Super Kings
1930  Punjab Cricket Association IS Bindra Stadium, ...           Punjab Kings
1931  Punjab Cricket Association IS Bindra Stadium, ...           Punjab Kings
1932  Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...   Lucknow Super Giants

                      team2            toss_winner  ...   result  \
1928         Gujarat Titans         Gujarat Titans  ...  wickets
1929         Gujarat Titans         Gujarat Titans  ...  wickets
1930  Kolkata Knight Riders  Kolkata Knight Riders  ...     runs
1931  Kolkata Knight Riders  Kolkata Knight Riders  ...     runs
1932         Delhi Capitals         Delhi Capitals  ...     runs

     result_margin target_runs  target_overs  super_over  method  \
1928           5.0       179.0          20.0           N     NaN
1929           5.0       179.0          20.0           N     NaN
1930           7.0       154.0          16.0           N     D/L
1931           7.0       154.0          16.0           N     D/L
1932          50.0       194.0          20.0           N     NaN

           umpire1        umpire2 match_id total_runs
1928   Nitin Menon     HAS Khalid  1359475        178
1929   Nitin Menon     HAS Khalid  1359475        182
1930  BNJ Oxenford       YC Barde  1359476        191
1931  BNJ Oxenford       YC Barde  1359476        146
1932  AK Chaudhary  NA Patwardhan  1359477        193

[5 rows x 22 columns]
```

[101]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
# Create a directory to store the plots
plot_dir = "ipl_season_plots"
os.makedirs(plot_dir, exist_ok=True)

def explore_season(season):
    season_df = match_df[match_df['season'] == season]
    season_filename = season.replace("/", "_")  # Replace '/' with '_'

    # Number of matches played in each stadium
    plt.figure(figsize=(12,6))
    sns.countplot(y=season_df['venue'], order=season_df['venue'].value_counts().
 ↪index, palette='coolwarm')
```

```
    plt.title(f'Number of Matches Played in Each Stadium ({season})')
    plt.xlabel('Count')
    plt.ylabel('Stadium')
    plt.savefig(f"{plot_dir}/{season_filename}_matches_per_stadium.png")
    plt.close()

    # Wins by each team
    plt.figure(figsize=(10,6))
    sns.countplot(y=season_df['winner'], order=season_df['winner'].
↪value_counts().index, palette='viridis')
    plt.title(f'Wins by Each Team in {season}')
    plt.xlabel('Count')
    plt.ylabel('Team')
    plt.savefig(f"{plot_dir}/{season_filename}_wins_by_team.png")
    plt.close()

    # Average score of teams
    avg_score = season_df.groupby('team1')['total_runs'].mean().sort_values()
    plt.figure(figsize=(10,6))
    sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
    plt.title(f'Average Score of Teams in {season}')
    plt.xlabel('Average Runs')
    plt.ylabel('Team')
    plt.savefig(f"{plot_dir}/{season_filename}_average_score.png")
    plt.close()

    # Total runs scored by each team in the season
    total_runs = season_df.groupby('team1')['total_runs'].sum().sort_values()
    plt.figure(figsize=(10,6))
    sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
    plt.title(f'Total Runs by Each Team in {season}')
    plt.xlabel('Total Runs')
    plt.ylabel('Team')
    plt.savefig(f"{plot_dir}/{season_filename}_total_runs.png")
    plt.close()

    # Win margins distribution
    plt.figure(figsize=(12,6))
    sns.histplot(season_df['result_margin'], bins=30, kde=True, color='blue')
    plt.title(f'Win Margins Distribution in {season}')
    plt.xlabel('Win Margin')
    plt.ylabel('Frequency')
    plt.savefig(f"{plot_dir}/{season_filename}_win_margins.png")
    plt.close()

    # Toss decision count
    plt.figure(figsize=(8,6))
```

```python
    sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
    plt.title(f'Toss Decision Count in {season}')
    plt.xlabel('Toss Decision')
    plt.ylabel('Count')
    plt.savefig(f"{plot_dir}/{season_filename}_toss_decision.png")
    plt.close()

    # Top players of the match
    top_players = season_df['player_of_match'].value_counts().head(10)
    plt.figure(figsize=(12,6))
    sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
    plt.title(f'Top Players of the Match in {season}')
    plt.xlabel('Number of Times Won')
    plt.ylabel('Player')
    plt.savefig(f"{plot_dir}/{season_filename}_top_players.png")
    plt.close()

# List of seasons to analyze
seasons = ['2007/08', '2009', '2009/10', '2011', '2012', '2013', '2014', '2015',␣
 ↪'2016', '2017', '2018', '2019', '2020/21', '2021', '2022', '2023', '2024']

# Run the exploration for all seasons
for season in seasons:
    explore_season(season)

print("All plots saved in", plot_dir)
```

```
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

```
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
```

```
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
```

```
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.
```

```
  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

```
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
<ipython-input-101-4edaa6615570>:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['venue'],
order=season_df['venue'].value_counts().index, palette='coolwarm')
<ipython-input-101-4edaa6615570>:24: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=season_df['winner'],
order=season_df['winner'].value_counts().index, palette='viridis')
<ipython-input-101-4edaa6615570>:34: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=avg_score.values, y=avg_score.index, palette='plasma')
<ipython-input-101-4edaa6615570>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=total_runs.values, y=total_runs.index, palette='magma')
<ipython-input-101-4edaa6615570>:62: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x=season_df['toss_decision'], palette='coolwarm')
```

```
<ipython-input-101-4edaa6615570>:72: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=top_players.values, y=top_players.index, palette='inferno')
All plots saved in ipl_season_plots
```

[102]:
```
import shutil

shutil.make_archive("/content/ipl_season_plots", 'zip', "/content/
↪ipl_season_plots")
```

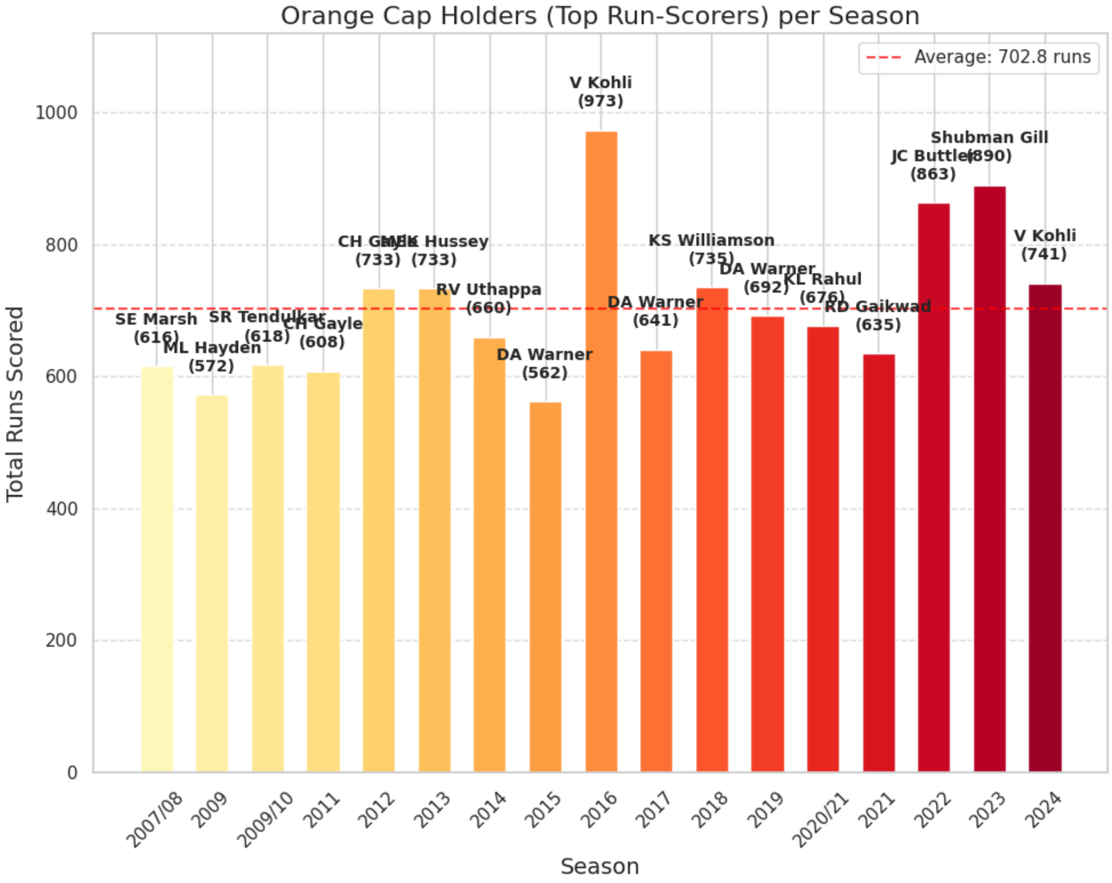
[102]:
```
'/content/ipl_season_plots.zip'
```

[104]:
```
import os
import IPython.display as display
from PIL import Image

plot_dir = "ipl_season_plots"
seasons = ['2007_08', '2009', '2009_10', '2011', '2012', '2013', '2014', '2015',
           '2016', '2017', '2018', '2019', '2020_21', '2021', '2022', '2023',␣
↪'2024']

# Iterate over each season
for season_filename in seasons:
    # List of plots to display for each season
    plots = [
        f"{plot_dir}/{season_filename}_matches_per_stadium.png",
        f"{plot_dir}/{season_filename}_wins_by_team.png",
        f"{plot_dir}/{season_filename}_total_runs.png",
        f"{plot_dir}/{season_filename}_top_players.png",
        f"{plot_dir}/{season_filename}_average_score.png",
        f"{plot_dir}/{season_filename}_win_margins.png"
    ]

    print(f"\n Displaying plots for {season_filename} season:")

    # Display all plots for the current season
    for plot in plots:
        if os.path.exists(plot):
            img = Image.open(plot)
            display.display(img)
        else:
            print(f" Plot not found: {plot}")
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Analyze runs of Orange Cap Holders per season

```
[105]: import pandas as pd
       import numpy as np
       import matplotlib.pyplot as plt
       import seaborn as sns

       def analyze_orange_cap_holders(df_matches, df_deliveries):
           """
           Analyze runs scored by Orange Cap holders (top run-scorers) per season
           """
           # Merge match data with deliveries to get season information
           match_seasons = df_matches[['id', 'season']].copy()
           match_seasons.rename(columns={'id': 'match_id'}, inplace=True)

           # Merge to get season info for each delivery
           deliveries_with_season = pd.merge(df_deliveries, match_seasons,␣
        ↪on='match_id', how='left')

           # Group by season and batsman to get total runs per season
           season_batsman_runs = deliveries_with_season.groupby(['season',␣
        ↪'batter'])['batsman_runs'].sum().reset_index()

           # Find the top run-scorer (Orange Cap holder) for each season
           orange_cap_holders = season_batsman_runs.loc[season_batsman_runs.
        ↪groupby('season')['batsman_runs'].idxmax()]

           # Sort by season
           orange_cap_holders = orange_cap_holders.sort_values('season')

           # Create a colorful bar chart
           plt.figure(figsize=(10, 8))

           # Use a custom color palette
           colors = sns.color_palette("YlOrRd", len(orange_cap_holders))

           # Create the bar chart
           bars = plt.bar(orange_cap_holders['season'].astype(str),
                     orange_cap_holders['batsman_runs'],
                     color=colors,
                     width=0.6)

           # Add data labels on top of bars
           for bar, player, runs in zip(bars, orange_cap_holders['batter'],␣
        ↪orange_cap_holders['batsman_runs']):
               plt.text(bar.get_x() + bar.get_width()/2, runs + 30,
                       f"{player}\n({runs})",
                       ha='center', va='bottom',
                       fontweight='bold', fontsize=10,
                       rotation=0)

           # Add a horizontal line for average Orange Cap runs
           avg_orange_runs = orange_cap_holders['batsman_runs'].mean()
           plt.axhline(y=avg_orange_runs, color='red', linestyle='--', alpha=0.7,
                      label=f'Average: {avg_orange_runs:.1f} runs')

           # Add titles and labels
           plt.title('Orange Cap Holders (Top Run-Scorers) per Season', fontsize=16)
           plt.xlabel('Season', fontsize=14)
           plt.ylabel('Total Runs Scored', fontsize=14)
           plt.xticks(rotation=45)
           plt.grid(axis='y', linestyle='--', alpha=0.7)

           # Add some padding at the top for labels
           plt.ylim(0, orange_cap_holders['batsman_runs'].max() * 1.15)

           plt.legend()
           plt.tight_layout()
           plt.show()

           # Analyze Orange Cap holder stats in more detail
           print("\nOrange Cap Holder Detailed Analysis:")
           print("=" * 80)
           print(f"{'Season':<10}{'Player':<20}{'Runs':<10}{'Matches':<10}{'Batting Avg':<15}{'Strike Rate':<15}")
           print("-" * 80)

           # Get more detailed stats for each Orange Cap holder
           for _, row in orange_cap_holders.iterrows():
               season = row['season']
               player = row['batter']
               total_runs = row['batsman_runs']

               # Get player stats for this season
               player_season_data = deliveries_with_season[
                   (deliveries_with_season['season'] == season) &
                   (deliveries_with_season['batter'] == player)
               ]

               # Calculate detailed stats
               matches_played = player_season_data['match_id'].nunique()
               dismissals = player_season_data[player_season_data['player_dismissed'] == player].shape[0]
               # Handle division by zero for not out batsmen
               if dismissals == 0:
                   batting_avg = total_runs
               else:
                   batting_avg = total_runs / dismissals

               balls_faced = player_season_data.shape[0]
               strike_rate = (total_runs / balls_faced) * 100 if balls_faced > 0 else 0

               print(f"{season:<10}{player:<20}{total_runs:<10}{matches_played:
       ↪<10}{batting_avg:.2f}{'':5}{strike_rate:.2f}{'':5}")

           print("=" * 80)

           return orange_cap_holders
       orange_cap_data = analyze_orange_cap_holders(df_matches, df_deliveries)
```



```
Orange Cap Holder Detailed Analysis:
```

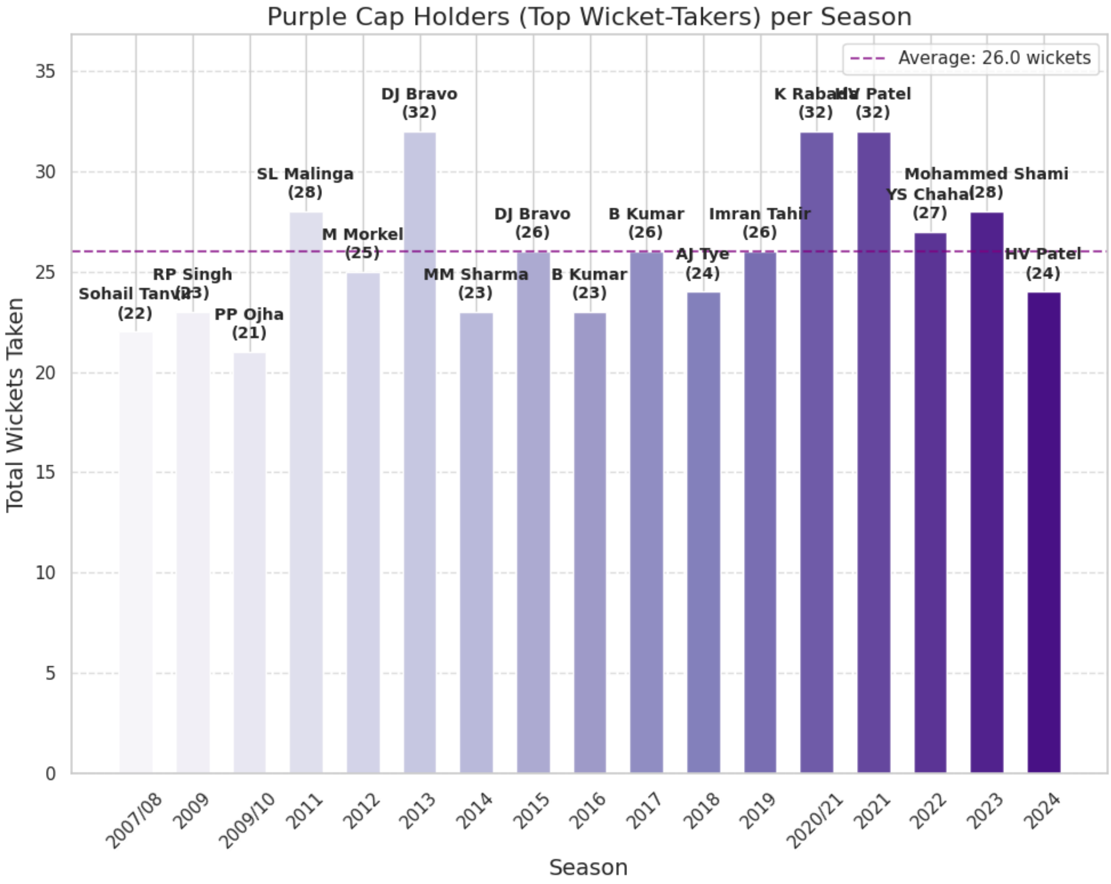
```
================================================================================
Season    Player              Runs      Matches   Batting Avg    Strike Rate
--------------------------------------------------------------------------------
2007/08   SE Marsh            616       11        68.44     136.28
2009      ML Hayden           572       12        57.20     139.85
2009/10   SR Tendulkar        618       15        47.54     126.38
2011      CH Gayle            608       12        67.56     177.78
2012      CH Gayle            733       14        61.08     155.30
2013      MEK Hussey          733       17        56.38     126.38
2014      RV Uthappa          660       16        44.00     136.08
2015      DA Warner           562       14        46.83     152.72
2016      V Kohli             973       16        81.08     148.55
2017      DA Warner           641       14        58.27     138.74
2018      KS Williamson       735       17        52.50     140.80
2019      DA Warner           692       12        69.20     139.52
2020/21   KL Rahul            676       14        48.29     127.31
2021      RD Gaikwad          635       16        45.36     133.97
2022      JC Buttler          863       17        57.53     144.80
2023      Shubman Gill        890       17        59.33     152.92
2024      V Kohli             741       15        61.75     149.09
================================================================================
```

Track wickets of Purple Cap Holders per season

[106]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def analyze_purple_cap_holders(df_matches, df_deliveries):
    """
    Analyze wickets taken by Purple Cap holders (top wicket-takers) per season
    """
    # Merge match data with deliveries to get season information
    match_seasons = df_matches[['id', 'season']].copy()
    match_seasons.rename(columns={'id': 'match_id'}, inplace=True)

    # Merge to get season info for each delivery
    deliveries_with_season = pd.merge(df_deliveries, match_seasons,␣
 ↪on='match_id', how='left')

    # Filter for wicket deliveries only
    wicket_deliveries =␣
 ↪deliveries_with_season[deliveries_with_season['is_wicket'] == 1].copy()

    # Count wickets by season and bowler
    # Note: We exclude run-outs as they're not credited to the bowler
    bowler_wickets = wicket_deliveries[~wicket_deliveries['dismissal_kind'].
↪isin(['run out', 'retired hurt', 'obstructing the field'])]

    # Group by season and bowler to get wicket counts
    season_bowler_wickets = bowler_wickets.groupby(['season', 'bowler']).size().
↪reset_index()
    season_bowler_wickets.rename(columns={0: 'wickets'}, inplace=True)

    # Find the top wicket-taker (Purple Cap holder) for each season
    purple_cap_holders = season_bowler_wickets.loc[season_bowler_wickets.
↪groupby('season')['wickets'].idxmax()]

    # Sort by season
    purple_cap_holders = purple_cap_holders.sort_values('season')

    # Create a colorful bar chart
    plt.figure(figsize=(10, 8))

    # Use a custom color palette
    colors = sns.color_palette("Purples", len(purple_cap_holders))

    # Create the bar chart
    bars = plt.bar(purple_cap_holders['season'].astype(str),
              purple_cap_holders['wickets'],
              color=colors,
              width=0.6)

    # Add data labels on top of bars
    for bar, player, wickets in zip(bars, purple_cap_holders['bowler'],␣
↪purple_cap_holders['wickets']):
        plt.text(bar.get_x() + bar.get_width()/2, wickets + 0.5,
                f"{player}\n({wickets})",
                ha='center', va='bottom',
                fontweight='bold', fontsize=10)

    # Add a horizontal line for average Purple Cap wickets
    avg_purple_wickets = purple_cap_holders['wickets'].mean()
    plt.axhline(y=avg_purple_wickets, color='purple', linestyle='--', alpha=0.7,
               label=f'Average: {avg_purple_wickets:.1f} wickets')

    # Add titles and labels
    plt.title('Purple Cap Holders (Top Wicket-Takers) per Season', fontsize=16)
    plt.xlabel('Season', fontsize=14)
    plt.ylabel('Total Wickets Taken', fontsize=14)
    plt.xticks(rotation=45)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
```

```python
    # Add some padding at the top for labels
    plt.ylim(0, purple_cap_holders['wickets'].max() * 1.15)

    plt.legend()
    plt.tight_layout()
    plt.show()

    # Analyze Purple Cap holder stats in more detail
    print("\nPurple Cap Holder Detailed Analysis:")
    print("=" * 90)
    print(f"{'Season':<10}{'Bowler':<20}{'Wickets':<10}{'Matches':<10}{'Economy':<10}{'Bowling Avg':<15}")
    print("-" * 90)

    # Get more detailed stats for each Purple Cap holder
    for _, row in purple_cap_holders.iterrows():
        season = row['season']
        player = row['bowler']
        total_wickets = row['wickets']

        # Get player stats for this season
        player_season_data = deliveries_with_season[
            (deliveries_with_season['season'] == season) &
            (deliveries_with_season['bowler'] == player)
        ]

        # Calculate detailed stats
        matches_played = player_season_data['match_id'].nunique()
        total_runs = player_season_data['total_runs'].sum()
        total_balls = len(player_season_data)

        # Calculate economy rate (runs per over)
        economy = (total_runs / (total_balls/6)) if total_balls > 0 else 0

        # Calculate bowling average (runs per wicket)
        bowling_avg = (total_runs / total_wickets) if total_wickets > 0 else float('inf')

        print(f"{season:<10}{player:<20}{total_wickets:<10}{matches_played:<10}{economy:.2f}{'':5}{bowling_avg:.2f}{'':5}")

    print("=" * 90)

    return purple_cap_holders

purple_cap_data = analyze_purple_cap_holders(df_matches, df_deliveries)
```



```
Purple Cap Holder Detailed Analysis:
================================================================================
==========
Season    Bowler              Wickets   Matches   Economy   Bowling Avg
--------------------------------------------------------------------------------
----------
2007/08   Sohail Tanvir       22        11        6.23     12.50
2009      RP Singh            23        16        6.75     18.70
2009/10   PP Ojha             21        16        7.32     20.90
2011      SL Malinga          28        16        5.94     14.04
2012      M Morkel            25        16        7.19     18.64
2013      DJ Bravo            32        18        7.73     15.78
2014      MM Sharma           23        16        8.46     19.87
2015      DJ Bravo            26        16        8.19     17.00
2016      B Kumar             23        17        7.29     21.87
2017      B Kumar             26        14        7.11     14.77
2018      AJ Tye              24        14        7.80     19.12
2019      Imran Tahir         26        17        6.80     16.92
2020/21   K Rabada            32        17        8.19     17.66
```

```
2021      HV Patel            32        15        7.66     14.41
2022      YS Chahal           27        17        7.50     19.85
2023      Mohammed Shami      28        17        7.92     19.00
2024      HV Patel            24        14        9.18     20.21
================================================================================
==========
```

Find top 10 bowlers per season

[113]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.gridspec import GridSpec

def find_top_bowlers_per_season(df_matches, df_deliveries,␣
 ↪season_to_analyze=None):
    """
    Find and analyze top 10 bowlers for each season or a specific season

    Parameters:
    df_matches: DataFrame containing match data
    df_deliveries: DataFrame containing ball-by-ball data
    season_to_analyze: Specific season to analyze (optional)
    """
    # Merge match data with deliveries to get season information
    match_seasons = df_matches[['id', 'season']].copy()
    match_seasons.rename(columns={'id': 'match_id'}, inplace=True)

    # Merge to get season info for each delivery
    deliveries_with_season = pd.merge(df_deliveries, match_seasons,␣
 ↪on='match_id', how='left')

    # Get list of seasons
    all_seasons = sorted(deliveries_with_season['season'].unique())

    # If a specific season is requested, filter for that season only
    if season_to_analyze is not None:
        if season_to_analyze in all_seasons:
            seasons_to_analyze = [season_to_analyze]
        else:
            print(f"Season {season_to_analyze} not found in data. Available␣
 ↪seasons: {all_seasons}")
            return None
    else:
        # Let's create an interactive menu to select a season
        print("Available seasons:")
        for i, season in enumerate(all_seasons):
            print(f"{i+1}. {season}")

        try:
            choice = int(input("\nSelect a season (1-{0}) or 0 to analyze all␣
↪seasons: ".format(len(all_seasons))))
            if choice == 0:
                seasons_to_analyze = all_seasons
            elif 1 <= choice <= len(all_seasons):
                seasons_to_analyze = [all_seasons[choice-1]]
            else:
                print("Invalid choice. Analyzing all seasons.")
                seasons_to_analyze = all_seasons
        except:
            print("Invalid input. Analyzing all seasons.")
            seasons_to_analyze = all_seasons

    # Create a dictionary to store top bowlers for each season
    top_bowlers_by_season = {}

    # Process each season
    for season in seasons_to_analyze:
        # Filter data for this season
        season_data = deliveries_with_season[deliveries_with_season['season'] ==␣
↪season]

        # Filter for wicket deliveries only
        wicket_deliveries = season_data[season_data['is_wicket'] == 1].copy()

        # Count wickets by bowler (excluding run-outs)
        bowler_wickets = wicket_deliveries[~wicket_deliveries['dismissal_kind'].
↪isin(['run out', 'retired hurt', 'obstructing the field'])]
        wicket_counts = bowler_wickets.groupby('bowler').size().reset_index()
        wicket_counts.columns = ['bowler', 'wickets']

        # Calculate economy rate
        # Group by bowler to get runs conceded and balls bowled
        bowler_stats = season_data.groupby('bowler').agg(
            runs_conceded=('total_runs', 'sum'),
            balls_bowled=('bowler', 'size')
        ).reset_index()

        # Calculate economy rate (runs per over)
        bowler_stats['economy'] = (bowler_stats['runs_conceded'] /␣
↪(bowler_stats['balls_bowled']/6)).round(2)

        # Calculate average (runs per wicket)
        # Merge wicket counts
        bowler_stats = pd.merge(bowler_stats, wicket_counts, on='bowler',␣
↪how='left')
        bowler_stats['wickets'] = bowler_stats['wickets'].fillna(0)

        # Calculate bowling average
        bowler_stats['bowling_avg'] = (bowler_stats['runs_conceded'] /␣
↪bowler_stats['wickets']).replace([np.inf, -np.inf], np.nan).round(2)

        # Calculate strike rate (balls per wicket)
        bowler_stats['strike_rate'] = (bowler_stats['balls_bowled'] /␣
↪bowler_stats['wickets']).replace([np.inf, -np.inf], np.nan).round(2)

        # Calculate matches played
        matches_played = season_data.groupby('bowler')['match_id'].nunique().
↪reset_index()
        matches_played.columns = ['bowler', 'matches']

        # Merge matches played
        bowler_stats = pd.merge(bowler_stats, matches_played, on='bowler',␣
↪how='left')

        # Calculate wickets per match
        bowler_stats['wickets_per_match'] = (bowler_stats['wickets'] /␣
↪bowler_stats['matches']).round(2)

        # Set minimum qualification criteria
        min_balls = 60  # At least 10 overs
        qualified_bowlers = bowler_stats[bowler_stats['balls_bowled'] >=␣
↪min_balls].copy()

        # Rank bowlers by wickets
        top_wicket_takers = qualified_bowlers.nlargest(10, 'wickets')

        # Store in dictionary
        top_bowlers_by_season[season] = top_wicket_takers

    # Visualization
    for season, top_bowlers in top_bowlers_by_season.items():
        # Create a figure with subplots
        fig = plt.figure(figsize=(10, 12))
        gs = GridSpec(2, 2, figure=fig)

        # Title for the entire figure
        fig.suptitle(f'Top 10 Bowlers Analysis - {season} Season', fontsize=20)

        # 1. Wickets Bar Chart
        ax1 = fig.add_subplot(gs[0, 0])
        sns.barplot(x='wickets', y='bowler', data=top_bowlers.
    ↪sort_values('wickets'), ax=ax1, palette='Purples_d')
        ax1.set_title('Total Wickets', fontsize=14)
        ax1.set_xlabel('Wickets', fontsize=12)
        ax1.set_ylabel('Bowler', fontsize=12)

        # 2. Economy Rate Bar Chart
        ax2 = fig.add_subplot(gs[0, 1])
        sns.barplot(x='economy', y='bowler', data=top_bowlers.
    ↪sort_values('economy'), ax=ax2, palette='Greens_d')
        ax2.set_title('Economy Rate (lower is better)', fontsize=14)
        ax2.set_xlabel('Economy Rate (runs per over)', fontsize=12)
        ax2.set_ylabel('', fontsize=12)  # No need to repeat

        # 3. Bowling Average Bar Chart
        ax3 = fig.add_subplot(gs[1, 0])
        sorted_by_avg = top_bowlers.sort_values('bowling_avg').
    ↪dropna(subset=['bowling_avg'])
        sns.barplot(x='bowling_avg', y='bowler', data=sorted_by_avg, ax=ax3,␣
    ↪palette='Blues_d')
        ax3.set_title('Bowling Average (lower is better)', fontsize=14)
        ax3.set_xlabel('Bowling Average (runs per wicket)', fontsize=12)
        ax3.set_ylabel('Bowler', fontsize=12)

        # 4. Strike Rate Bar Chart
        ax4 = fig.add_subplot(gs[1, 1])
        sorted_by_sr = top_bowlers.sort_values('strike_rate').
    ↪dropna(subset=['strike_rate'])
        sns.barplot(x='strike_rate', y='bowler', data=sorted_by_sr, ax=ax4,␣
    ↪palette='Oranges_d')
        ax4.set_title('Strike Rate (lower is better)', fontsize=14)
        ax4.set_xlabel('Strike Rate (balls per wicket)', fontsize=12)
        ax4.set_ylabel('', fontsize=12)  # No need to repeat

        plt.tight_layout()
        plt.subplots_adjust(top=0.92)  # Adjust for the suptitle
        plt.show()

        # Print detailed stats table
        print(f"\nTop 10 Bowlers - {season} Season")
        print("=" * 100)
        print(f"{'Rank':<6}{'Bowler':<20}{'Wickets':<10}{'Matches':
    ↪<10}{'Economy':<10}{'Bowling Avg':<15}{'Strike Rate':<15}")
        print("-" * 100)
```

```
        for i, (_, row) in enumerate(top_bowlers.sort_values('wickets',␣
↪ascending=False).iterrows()):
            print(f"{i+1:<6}{row['bowler']:<20}{int(row['wickets']):
↪<10}{int(row['matches']):<10}{row['economy']:<10.2f}{row['bowling_avg']:<15.
↪2f}{row['strike_rate']:<15.2f}")

        print("=" * 100)

    return top_bowlers_by_season

# To use this function with interactive season selection:
top_bowlers = find_top_bowlers_per_season(df_matches, df_deliveries)

# # To analyze a specific season directly:
# top_bowlers = find_top_bowlers_per_season(df_matches, df_deliveries,␣
↪season_to_analyze=2011)
```

```
Output hidden; open in https://colab.research.google.com to view.
```

#PART 2 : Feature Extraction - Manipulation - Prediction Model: ### IPL 2025 Match Winner Predictor

8.1 Description:

This section builds a predictive model for the IPL 2025 season using historical match data.
The objective is to extract key insights from **matches.csv** and **deliveries.csv**,
train machine learning models, and develop an ensemble approach for winner prediction.

8.2 Workflow:

1. **Data Preprocessing & Feature Extraction**
   - Extract key features from matches.csv (batting_team, bowling_team, city, runs_left, etc.).

   - Extract crucial insights from deliveries.csv (batsman/bowler performance, powerplay analysis).
2. **Model Training & Evaluation**
   - Train a **Logistic Regression model** as a baseline.

   - Train advanced models: **Random Forest, XGBoost, and Neural Networks**.

   - Develop an **ensemble model** combining classifiers (e.g., Random Forest + XGBoost).
3. **Performance Evaluation**
   - Compare models using **accuracy, precision, recall, F1-score, and ROC-AUC**.

   - Discuss model strengths, weaknesses, and key influencing factors.
4. **Results & Predictions for IPL 2025**
   - Use the trained model to **predict winners for the IPL 2025 season**.

- Provide insights into predicted team performance and key players.

[114]:
```
import numpy as np
import pandas as pd
```

[115]:
```
match=pd.read_csv('matches.csv')
```

[116]:
```
delivery=pd.read_csv('/content/deliveries.csv')
```

[117]:
```
match.head()
```

[117]:
```
       id   season        city        date match_type player_of_match  \
0  335982  2007/08   Bangalore  2008-04-18     League     BB McCullum
1  335983  2007/08  Chandigarh  2008-04-19     League      MEK Hussey
2  335984  2007/08       Delhi  2008-04-19     League     MF Maharoof
3  335985  2007/08      Mumbai  2008-04-20     League      MV Boucher
4  335986  2007/08     Kolkata  2008-04-20     League       DJ Hussey

                                        venue                        team1  \
0                       M Chinnaswamy Stadium  Royal Challengers Bangalore
1  Punjab Cricket Association Stadium, Mohali              Kings XI Punjab
2                            Feroz Shah Kotla             Delhi Daredevils
3                            Wankhede Stadium               Mumbai Indians
4                                Eden Gardens        Kolkata Knight Riders

                         team2                  toss_winner toss_decision  \
0        Kolkata Knight Riders  Royal Challengers Bangalore         field
1          Chennai Super Kings          Chennai Super Kings           bat
2             Rajasthan Royals             Rajasthan Royals           bat
3  Royal Challengers Bangalore               Mumbai Indians           bat
4              Deccan Chargers              Deccan Chargers           bat

                        winner   result  result_margin  target_runs  \
0        Kolkata Knight Riders     runs          140.0        223.0
1          Chennai Super Kings     runs           33.0        241.0
2             Delhi Daredevils  wickets            9.0        130.0
3  Royal Challengers Bangalore  wickets            5.0        166.0
4        Kolkata Knight Riders  wickets            5.0        111.0

   target_overs super_over method    umpire1         umpire2
0          20.0          N    NaN  Asad Rauf     RE Koertzen
1          20.0          N    NaN  MR Benson      SL Shastri
2          20.0          N    NaN  Aleem Dar  GA Pratapkumar
3          20.0          N    NaN   SJ Davis       DJ Harper
4          20.0          N    NaN  BF Bowden     K Hariharan
```

8.2.1 All Match details of all season till now:

```
[118]: match.shape
```

```
[118]: (1095, 20)
```

Ball by ball details of each match is present in deliveries:

```
[ ]: delivery.shape
```

```
[ ]: (260920, 17)
```

```
[119]: delivery.head()
```

```
[119]:    match_id  inning           batting_team                bowling_team  over  \
       0    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore     0
       1    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore     0
       2    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore     0
       3    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore     0
       4    335982       1  Kolkata Knight Riders  Royal Challengers Bangalore     0

          ball       batter   bowler  non_striker  batsman_runs  extra_runs  \
       0     1   SC Ganguly  P Kumar  BB McCullum             0           1
       1     2  BB McCullum  P Kumar   SC Ganguly             0           0
       2     3  BB McCullum  P Kumar   SC Ganguly             0           1
       3     4  BB McCullum  P Kumar   SC Ganguly             0           0
       4     5  BB McCullum  P Kumar   SC Ganguly             0           0

          total_runs extras_type  is_wicket player_dismissed dismissal_kind fielder
       0           1     legbyes          0              NaN            NaN     NaN
       1           0         NaN          0              NaN            NaN     NaN
       2           1       wides          0              NaN            NaN     NaN
       3           0         NaN          0              NaN            NaN     NaN
       4           0         NaN          0              NaN            NaN     NaN
```

8.2.2 Data Analysis:

```
[120]: print("Match DataFrame Info:")
       match.info()
       print("\nDelivery DataFrame Info:")
       delivery.info()

       print("\nMatch DataFrame Descriptive Statistics:")
       print(match.describe())
       print("\nDelivery DataFrame Descriptive Statistics:")
       print(delivery.describe())

       print("\nMatch DataFrame Missing Values:")
       print(match.isnull().sum())
```

```python
print("\nDelivery DataFrame Missing Values:")
print(delivery.isnull().sum())

print("\nMatch DataFrame Unique Values for Selected Columns:")
for column in ['season', 'city', 'winner']:
  print(f"Unique values for {column}: {match[column].unique()}")

print("\nDelivery DataFrame Unique Values for Selected Columns:")
for column in ['batting_team', 'bowling_team']:
  print(f"Unique values for {column}: {delivery[column].unique()}")
```

```
Match DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1095 entries, 0 to 1094
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               1095 non-null   int64
 1   season           1095 non-null   object
 2   city             1044 non-null   object
 3   date             1095 non-null   object
 4   match_type       1095 non-null   object
 5   player_of_match  1090 non-null   object
 6   venue            1095 non-null   object
 7   team1            1095 non-null   object
 8   team2            1095 non-null   object
 9   toss_winner      1095 non-null   object
 10  toss_decision    1095 non-null   object
 11  winner           1090 non-null   object
 12  result           1095 non-null   object
 13  result_margin    1076 non-null   float64
 14  target_runs      1092 non-null   float64
 15  target_overs     1092 non-null   float64
 16  super_over       1095 non-null   object
 17  method           21 non-null     object
 18  umpire1          1095 non-null   object
 19  umpire2          1095 non-null   object
dtypes: float64(3), int64(1), object(16)
memory usage: 171.2+ KB

Delivery DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260920 entries, 0 to 260919
Data columns (total 17 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   match_id          260920 non-null  int64
 1   inning            260920 non-null  int64
 2   batting_team      260920 non-null  object
 3   bowling_team      260920 non-null  object
 4   over              260920 non-null  int64
 5   ball              260920 non-null  int64
 6   batter            260920 non-null  object
 7   bowler            260920 non-null  object
 8   non_striker       260920 non-null  object
 9   batsman_runs      260920 non-null  int64
 10  extra_runs        260920 non-null  int64
 11  total_runs        260920 non-null  int64
 12  extras_type       14125 non-null   object
 13  is_wicket         260920 non-null  int64
 14  player_dismissed  12950 non-null   object
 15  dismissal_kind    12950 non-null   object
 16  fielder           9354 non-null    object
dtypes: int64(8), object(9)
memory usage: 33.8+ MB

Match DataFrame Descriptive Statistics:
                 id  result_margin  target_runs  target_overs
count  1.095000e+03    1076.000000  1092.000000   1092.000000
mean   9.048283e+05      17.259294   165.684066     19.759341
std    3.677402e+05      21.787444    33.427048      1.581108
min    3.359820e+05       1.000000    43.000000      5.000000
25%    5.483315e+05       6.000000   146.000000     20.000000
50%    9.809610e+05       8.000000   166.000000     20.000000
75%    1.254062e+06      20.000000   187.000000     20.000000
max    1.426312e+06     146.000000   288.000000     20.000000

Delivery DataFrame Descriptive Statistics:
           match_id         inning           over           ball  \
count  2.609200e+05  260920.000000  260920.000000  260920.000000
mean   9.070665e+05       1.483531       9.197677       3.624486
std    3.679913e+05       0.502643       5.683484       1.814920
min    3.359820e+05       1.000000       0.000000       1.000000
25%    5.483340e+05       1.000000       4.000000       2.000000
50%    9.809670e+05       1.000000       9.000000       4.000000
75%    1.254066e+06       2.000000      14.000000       5.000000
max    1.426312e+06       6.000000      19.000000      11.000000

        batsman_runs     extra_runs     total_runs      is_wicket
count  260920.000000  260920.000000  260920.000000  260920.000000
mean        1.265001       0.067806       1.332807       0.049632
std         1.639298       0.343265       1.626416       0.217184
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         1.000000       0.000000       1.000000       0.000000
75%         1.000000       0.000000       1.000000       0.000000
```

```
max         6.000000       7.000000       7.000000       1.000000

Match DataFrame Missing Values:
id                     0
season                 0
city                  51
date                   0
match_type             0
player_of_match        5
venue                  0
team1                  0
team2                  0
toss_winner            0
toss_decision          0
winner                 5
result                 0
result_margin         19
target_runs            3
target_overs           3
super_over             0
method              1074
umpire1                0
umpire2                0
dtype: int64

Delivery DataFrame Missing Values:
match_id                 0
inning                   0
batting_team             0
bowling_team             0
over                     0
ball                     0
batter                   0
bowler                   0
non_striker              0
batsman_runs             0
extra_runs               0
total_runs               0
extras_type         246795
is_wicket                0
player_dismissed    247970
dismissal_kind      247970
fielder             251566
dtype: int64

Match DataFrame Unique Values for Selected Columns:
Unique values for season: ['2007/08' '2009' '2009/10' '2011' '2012' '2013'
'2014' '2015' '2016'
```

```
 '2017' '2018' '2019' '2020/21' '2021' '2022' '2023' '2024']
Unique values for city: ['Bangalore' 'Chandigarh' 'Delhi' 'Mumbai' 'Kolkata'
'Jaipur' 'Hyderabad'
 'Chennai' 'Cape Town' 'Port Elizabeth' 'Durban' 'Centurion' 'East London'
 'Johannesburg' 'Kimberley' 'Bloemfontein' 'Ahmedabad' 'Cuttack' 'Nagpur'
 'Dharamsala' 'Kochi' 'Indore' 'Visakhapatnam' 'Pune' 'Raipur' 'Ranchi'
 'Abu Dhabi' nan 'Rajkot' 'Kanpur' 'Bengaluru' 'Dubai' 'Sharjah'
 'Navi Mumbai' 'Lucknow' 'Guwahati' 'Mohali']
Unique values for winner: ['Kolkata Knight Riders' 'Chennai Super Kings' 'Delhi
Daredevils'
 'Royal Challengers Bangalore' 'Rajasthan Royals' 'Kings XI Punjab'
 'Deccan Chargers' 'Mumbai Indians' 'Pune Warriors' 'Kochi Tuskers Kerala'
 nan 'Sunrisers Hyderabad' 'Rising Pune Supergiants' 'Gujarat Lions'
 'Rising Pune Supergiant' 'Delhi Capitals' 'Punjab Kings' 'Gujarat Titans'
 'Lucknow Super Giants' 'Royal Challengers Bengaluru']

Delivery DataFrame Unique Values for Selected Columns:
Unique values for batting_team: ['Kolkata Knight Riders' 'Royal Challengers
Bangalore'
 'Chennai Super Kings' 'Kings XI Punjab' 'Rajasthan Royals'
 'Delhi Daredevils' 'Mumbai Indians' 'Deccan Chargers'
 'Kochi Tuskers Kerala' 'Pune Warriors' 'Sunrisers Hyderabad'
 'Rising Pune Supergiants' 'Gujarat Lions' 'Rising Pune Supergiant'
 'Delhi Capitals' 'Punjab Kings' 'Lucknow Super Giants' 'Gujarat Titans'
 'Royal Challengers Bengaluru']
Unique values for bowling_team: ['Royal Challengers Bangalore' 'Kolkata Knight
Riders' 'Kings XI Punjab'
 'Chennai Super Kings' 'Delhi Daredevils' 'Rajasthan Royals'
 'Mumbai Indians' 'Deccan Chargers' 'Kochi Tuskers Kerala' 'Pune Warriors'
 'Sunrisers Hyderabad' 'Rising Pune Supergiants' 'Gujarat Lions'
 'Rising Pune Supergiant' 'Delhi Capitals' 'Punjab Kings' 'Gujarat Titans'
 'Lucknow Super Giants' 'Royal Challengers Bengaluru']
```

[121]:
```
delivery.groupby(['match_id','inning']).sum()['total_runs']
# Each match is now visualised as a two innings data - 1st team batting then 2nd␣
↪team batting
# and their respective scores
```

[121]:
```
match_id  inning
335982    1         222
          2          82
335983    1         240
          2         207
335984    1         129
                   ...
1426310   2         174
1426311   1         175
```

```
          2         139
1426312   1         113
          2         114
Name: total_runs, Length: 2217, dtype: int64
```

[122]:

```
total_score_df=delivery.groupby(['match_id','inning']).sum()['total_runs'].
 ↪reset_index()
```

[123]:

```
total_score_df
```

[123]:

```
      match_id  inning  total_runs
0       335982       1         222
1       335982       2          82
2       335983       1         240
3       335983       2         207
4       335984       1         129
...        ...     ...         ...
2212   1426310       2         174
2213   1426311       1         175
2214   1426311       2         139
2215   1426312       1         113
2216   1426312       2         114

[2217 rows x 3 columns]
```

[124]:

```
total_score_df=total_score_df[total_score_df['inning']==1]
```

[125]:

```
total_score_df
```

[125]:

```
      match_id  inning  total_runs
0       335982       1         222
2       335983       1         240
4       335984       1         129
6       335985       1         165
8       335986       1         110
...        ...     ...         ...
2207   1426307       1         214
2209   1426309       1         159
2211   1426310       1         172
2213   1426311       1         175
2215   1426312       1         113

[1095 rows x 3 columns]
```

Merging match and delivery dataframe by match id

[126]:

```
#merge by match_id and id
```

```
match_df=match.
 ↪merge(total_score_df[['match_id','total_runs']],left_on='id',right_on='match_id')
```

[127]:

```
match_df.head()
```

[127]:

```
       id   season        city        date match_type player_of_match  \
0  335982  2007/08   Bangalore  2008-04-18     League     BB McCullum
1  335983  2007/08  Chandigarh  2008-04-19     League      MEK Hussey
2  335984  2007/08       Delhi  2008-04-19     League     MF Maharoof
3  335985  2007/08      Mumbai  2008-04-20     League      MV Boucher
4  335986  2007/08     Kolkata  2008-04-20     League       DJ Hussey

                                        venue                        team1  \
0                       M Chinnaswamy Stadium  Royal Challengers Bangalore
1  Punjab Cricket Association Stadium, Mohali              Kings XI Punjab
2                            Feroz Shah Kotla             Delhi Daredevils
3                            Wankhede Stadium               Mumbai Indians
4                                Eden Gardens        Kolkata Knight Riders

                         team2                  toss_winner  ...   result  \
0        Kolkata Knight Riders  Royal Challengers Bangalore  ...     runs
1          Chennai Super Kings          Chennai Super Kings  ...     runs
2             Rajasthan Royals             Rajasthan Royals  ...  wickets
3  Royal Challengers Bangalore               Mumbai Indians  ...  wickets
4              Deccan Chargers              Deccan Chargers  ...  wickets

  result_margin target_runs  target_overs  super_over  method    umpire1  \
0         140.0       223.0          20.0           N     NaN  Asad Rauf
1          33.0       241.0          20.0           N     NaN  MR Benson
2           9.0       130.0          20.0           N     NaN  Aleem Dar
3           5.0       166.0          20.0           N     NaN   SJ Davis
4           5.0       111.0          20.0           N     NaN  BF Bowden

          umpire2 match_id total_runs
0     RE Koertzen   335982        222
1      SL Shastri   335983        240
2  GA Pratapkumar   335984        129
3       DJ Harper   335985        165
4     K Hariharan   335986        110

[5 rows x 22 columns]
```

8.2.3 Data Processing (as a prerequisite for model design approach):

[ ]:

```
#Data Processing :
match_df['team1'].unique()
```

```
[ ]: array(['Royal Challengers Bangalore', 'Kings XI Punjab',
            'Delhi Daredevils', 'Mumbai Indians', 'Kolkata Knight Riders',
            'Rajasthan Royals', 'Deccan Chargers', 'Chennai Super Kings',
            'Kochi Tuskers Kerala', 'Pune Warriors', 'Sunrisers Hyderabad',
            'Gujarat Lions', 'Rising Pune Supergiants',
            'Rising Pune Supergiant', 'Delhi Capitals', 'Punjab Kings',
            'Lucknow Super Giants', 'Gujarat Titans',
            'Royal Challengers Bengaluru'], dtype=object)
```

```
[128]: teams=['Royal Challengers Bangalore',
              'Kings XI Punjab',
              'Mumbai Indians',
              'Kolkata Knight Riders',
              'Rajasthan Royals',
              'Chennai Super Kings',
              'Sunrisers Hyderabad',
              'Delhi Capitals',
              'Lucknow Super Giants',
              'Gujarat Titans',
              ]
```

Removing teams that dont play and also replacing some changed team names:

```
[129]: match_df['team1']=match_df['team1'].str.replace('Delhi Daredevils','Delhi␣
        ↪Capitals')
       match_df['team2']=match_df['team2'].str.replace('Delhi Daredevils','Delhi␣
        ↪Capitals')

       match_df['team1']=match_df['team1'].str.replace('Deccan Chargers','Sunrisers␣
        ↪Hyderabad')
       match_df['team2']=match_df['team2'].str.replace('Deccan Chargers','Sunrisers␣
        ↪Hyderabad')
```

```
[130]: match_df=match_df[match_df['team1'].isin(teams)]
       match_df=match_df[match_df['team2'].isin(teams)]
```

```
[131]: match_df.shape
```

```
[131]: (911, 22)
```

```
[132]: match_df['season'].unique()
```

```
[132]: array(['2007/08', '2009', '2009/10', '2011', '2012', '2013', '2014',
              '2015', '2016', '2017', '2018', '2019', '2020/21', '2021', '2022',
              '2023', '2024'], dtype=object)
```

```
[133]: match_df.columns
```

```
[133]: Index(['id', 'season', 'city', 'date', 'match_type', 'player_of_match',
              'venue', 'team1', 'team2', 'toss_winner', 'toss_decision', 'winner',
              'result', 'result_margin', 'target_runs', 'target_overs', 'super_over',
              'method', 'umpire1', 'umpire2', 'match_id', 'total_runs'],
             dtype='object')
```

```
[134]: print("Features of match_df:")
       match_df['method'].unique()
```

```
       Features of match_df:
```

```
[134]: array([nan, 'D/L'], dtype=object)
```

Excluded dls - rain-affected matches:

```
[135]: #remove method column =d/l rows beacuse rain affected not required
       match_df = match_df[match_df['method'].isnull()]
```

```
[136]: match_df.shape
```

```
[136]: (895, 22)
```

```
[140]: match_df.keys()
```

```
[140]: Index(['id', 'season', 'city', 'date', 'match_type', 'player_of_match',
              'venue', 'team1', 'team2', 'toss_winner', 'toss_decision', 'winner',
              'result', 'result_margin', 'target_runs', 'target_overs', 'super_over',
              'method', 'umpire1', 'umpire2', 'match_id', 'total_runs'],
             dtype='object')
```

```
[138]: match_df.head()
```

```
[138]:        id   season        city        date match_type player_of_match  \
       0  335982  2007/08   Bangalore  2008-04-18     League     BB McCullum
       1  335983  2007/08  Chandigarh  2008-04-19     League      MEK Hussey
       2  335984  2007/08       Delhi  2008-04-19     League     MF Maharoof
       3  335985  2007/08      Mumbai  2008-04-20     League      MV Boucher
       4  335986  2007/08     Kolkata  2008-04-20     League       DJ Hussey

                                               venue                        team1  \
       0                       M Chinnaswamy Stadium  Royal Challengers Bangalore
       1  Punjab Cricket Association Stadium, Mohali              Kings XI Punjab
       2                            Feroz Shah Kotla               Delhi Capitals
       3                            Wankhede Stadium               Mumbai Indians
       4                                Eden Gardens        Kolkata Knight Riders

                            team2                  toss_winner  ...  result  \
       0    Kolkata Knight Riders  Royal Challengers Bangalore  ...    runs
       1      Chennai Super Kings          Chennai Super Kings  ...    runs
```

```
2             Rajasthan Royals             Rajasthan Royals  ...  wickets
3  Royal Challengers Bangalore               Mumbai Indians  ...  wickets
4          Sunrisers Hyderabad              Deccan Chargers  ...  wickets

  result_margin target_runs  target_overs  super_over  method    umpire1  \
0         140.0       223.0          20.0           N     NaN  Asad Rauf
1          33.0       241.0          20.0           N     NaN  MR Benson
2           9.0       130.0          20.0           N     NaN  Aleem Dar
3           5.0       166.0          20.0           N     NaN   SJ Davis
4           5.0       111.0          20.0           N     NaN  BF Bowden

          umpire2 match_id total_runs
0     RE Koertzen   335982        222
1      SL Shastri   335983        240
2  GA Pratapkumar   335984        129
3       DJ Harper   335985        165
4     K Hariharan   335986        110

[5 rows x 22 columns]
```

8.3 The features which we thought are good parameters for training the model to predict probability of match winning:

1. batting team
2. bowling team
3. city/venue
4. runs_left
5. balls_left
6. wicket_left
7. total_runs_x
8. crr
9. rrr
10. result

Required features: runs_left, balls_left, wicket_left, crr,rrr

[141]:
```
# batting team, bowling team, city , runs_left, balls_left, wicket_left, total_runs_x,crr,rrr,result
match_df=match_df[['match_id','city','winner','total_runs']]
```

[142]:
```
delivery_df=match_df.merge(delivery,on='match_id')
```

[143]:
```
delivery_df=delivery_df[delivery_df['inning']==2]
```

[144]:
```
delivery_df.shape
```

[144]:
```
(103793, 20)
```

```
[145]: delivery_df
```

```
[145]:
        match_id       city                 winner  total_runs_x  inning  \
124       335982  Bangalore  Kolkata Knight Riders           222       2
125       335982  Bangalore  Kolkata Knight Riders           222       2
126       335982  Bangalore  Kolkata Knight Riders           222       2
127       335982  Bangalore  Kolkata Knight Riders           222       2
128       335982  Bangalore  Kolkata Knight Riders           222       2
...          ...        ...                    ...           ...     ...
214508   1426312    Chennai  Kolkata Knight Riders           113       2
214509   1426312    Chennai  Kolkata Knight Riders           113       2
214510   1426312    Chennai  Kolkata Knight Riders           113       2
214511   1426312    Chennai  Kolkata Knight Riders           113       2
214512   1426312    Chennai  Kolkata Knight Riders           113       2

                       batting_team           bowling_team  over  ball  \
124     Royal Challengers Bangalore  Kolkata Knight Riders     0     1
125     Royal Challengers Bangalore  Kolkata Knight Riders     0     2
126     Royal Challengers Bangalore  Kolkata Knight Riders     0     3
127     Royal Challengers Bangalore  Kolkata Knight Riders     0     4
128     Royal Challengers Bangalore  Kolkata Knight Riders     0     5
...                             ...                    ...   ...   ...
214508        Kolkata Knight Riders    Sunrisers Hyderabad     9     5
214509        Kolkata Knight Riders    Sunrisers Hyderabad     9     6
214510        Kolkata Knight Riders    Sunrisers Hyderabad    10     1
214511        Kolkata Knight Riders    Sunrisers Hyderabad    10     2
214512        Kolkata Knight Riders    Sunrisers Hyderabad    10     3

          batter         bowler non_striker  batsman_runs  extra_runs  \
124     R Dravid       AB Dinda    W Jaffer             1           0
125     W Jaffer       AB Dinda    R Dravid             0           1
126     W Jaffer       AB Dinda    R Dravid             0           0
127     W Jaffer       AB Dinda    R Dravid             1           0
128     R Dravid       AB Dinda    W Jaffer             1           0
...          ...            ...         ...           ...         ...
214508   SS Iyer     AK Markram     VR Iyer             1           0
214509   VR Iyer     AK Markram     SS Iyer             1           0
214510   VR Iyer  Shahbaz Ahmed     SS Iyer             1           0
214511   SS Iyer  Shahbaz Ahmed     VR Iyer             1           0
214512   VR Iyer  Shahbaz Ahmed     SS Iyer             1           0

        total_runs_y extras_type  is_wicket player_dismissed dismissal_kind  \
124                1         NaN          0              NaN            NaN
125                1       wides          0              NaN            NaN
126                0         NaN          0              NaN            NaN
127                1         NaN          0              NaN            NaN
128                1         NaN          0              NaN            NaN
```

```
...              ...         ...        ...              ...            ...
214508             1         NaN          0              NaN            NaN
214509             1         NaN          0              NaN            NaN
214510             1         NaN          0              NaN            NaN
214511             1         NaN          0              NaN            NaN
214512             1         NaN          0              NaN            NaN

       fielder
124        NaN
125        NaN
126        NaN
127        NaN
128        NaN
...        ...
214508     NaN
214509     NaN
214510     NaN
214511     NaN
214512     NaN

[103793 rows x 20 columns]
```

[146]:

```
delivery_df['current_score'] = delivery_df.groupby('match_id')['total_runs_y'].cumsum().astype(int)
```

[147]:

```
delivery_df
```

[147]:

```
        match_id       city                 winner  total_runs_x  inning  \
124       335982  Bangalore  Kolkata Knight Riders           222       2
125       335982  Bangalore  Kolkata Knight Riders           222       2
126       335982  Bangalore  Kolkata Knight Riders           222       2
127       335982  Bangalore  Kolkata Knight Riders           222       2
128       335982  Bangalore  Kolkata Knight Riders           222       2
...          ...        ...                    ...           ...     ...
214508   1426312    Chennai  Kolkata Knight Riders           113       2
214509   1426312    Chennai  Kolkata Knight Riders           113       2
214510   1426312    Chennai  Kolkata Knight Riders           113       2
214511   1426312    Chennai  Kolkata Knight Riders           113       2
214512   1426312    Chennai  Kolkata Knight Riders           113       2

                       batting_team           bowling_team  over  ball  \
124     Royal Challengers Bangalore  Kolkata Knight Riders     0     1
125     Royal Challengers Bangalore  Kolkata Knight Riders     0     2
126     Royal Challengers Bangalore  Kolkata Knight Riders     0     3
127     Royal Challengers Bangalore  Kolkata Knight Riders     0     4
128     Royal Challengers Bangalore  Kolkata Knight Riders     0     5
...                             ...                    ...   ...   ...
```

```
214508        Kolkata Knight Riders    Sunrisers Hyderabad     9     5
214509        Kolkata Knight Riders    Sunrisers Hyderabad     9     6
214510        Kolkata Knight Riders    Sunrisers Hyderabad    10     1
214511        Kolkata Knight Riders    Sunrisers Hyderabad    10     2
214512        Kolkata Knight Riders    Sunrisers Hyderabad    10     3

          batter  ... non_striker batsman_runs  extra_runs  total_runs_y  \
124     R Dravid  ...    W Jaffer            1           0             1
125     W Jaffer  ...    R Dravid            0           1             1
126     W Jaffer  ...    R Dravid            0           0             0
127     W Jaffer  ...    R Dravid            1           0             1
128     R Dravid  ...    W Jaffer            1           0             1
...          ...  ...         ...          ...         ...           ...
214508   SS Iyer  ...     VR Iyer            1           0             1
214509   VR Iyer  ...     SS Iyer            1           0             1
214510   VR Iyer  ...     SS Iyer            1           0             1
214511   SS Iyer  ...     VR Iyer            1           0             1
214512   VR Iyer  ...     SS Iyer            1           0             1

        extras_type is_wicket  player_dismissed dismissal_kind fielder  \
124             NaN         0               NaN            NaN     NaN
125           wides         0               NaN            NaN     NaN
126             NaN         0               NaN            NaN     NaN
127             NaN         0               NaN            NaN     NaN
128             NaN         0               NaN            NaN     NaN
...             ...       ...               ...            ...     ...
214508          NaN         0               NaN            NaN     NaN
214509          NaN         0               NaN            NaN     NaN
214510          NaN         0               NaN            NaN     NaN
214511          NaN         0               NaN            NaN     NaN
214512          NaN         0               NaN            NaN     NaN

       current_score
124                1
125                2
126                2
127                3
128                4
...              ...
214508           110
214509           111
214510           112
214511           113
214512           114

[103793 rows x 21 columns]
```

[148]:

```
#total runs x=target
#runs_left = target-curr score
delivery_df['runs_left'] = delivery_df['total_runs_x'] -␣
 ↪delivery_df['current_score']+1
```

[149]:

```
delivery_df
```

[149]:

```
        match_id       city                 winner  total_runs_x  inning  \
124       335982  Bangalore  Kolkata Knight Riders           222       2
125       335982  Bangalore  Kolkata Knight Riders           222       2
126       335982  Bangalore  Kolkata Knight Riders           222       2
127       335982  Bangalore  Kolkata Knight Riders           222       2
128       335982  Bangalore  Kolkata Knight Riders           222       2
...          ...        ...                    ...           ...     ...
214508   1426312    Chennai  Kolkata Knight Riders           113       2
214509   1426312    Chennai  Kolkata Knight Riders           113       2
214510   1426312    Chennai  Kolkata Knight Riders           113       2
214511   1426312    Chennai  Kolkata Knight Riders           113       2
214512   1426312    Chennai  Kolkata Knight Riders           113       2

                       batting_team           bowling_team  over  ball  \
124     Royal Challengers Bangalore  Kolkata Knight Riders     0     1
125     Royal Challengers Bangalore  Kolkata Knight Riders     0     2
126     Royal Challengers Bangalore  Kolkata Knight Riders     0     3
127     Royal Challengers Bangalore  Kolkata Knight Riders     0     4
128     Royal Challengers Bangalore  Kolkata Knight Riders     0     5
...                             ...                    ...   ...   ...
214508        Kolkata Knight Riders    Sunrisers Hyderabad     9     5
214509        Kolkata Knight Riders    Sunrisers Hyderabad     9     6
214510        Kolkata Knight Riders    Sunrisers Hyderabad    10     1
214511        Kolkata Knight Riders    Sunrisers Hyderabad    10     2
214512        Kolkata Knight Riders    Sunrisers Hyderabad    10     3

          batter  ... batsman_runs extra_runs  total_runs_y  extras_type  \
124     R Dravid  ...            1          0             1          NaN
125     W Jaffer  ...            0          1             1        wides
126     W Jaffer  ...            0          0             0          NaN
127     W Jaffer  ...            1          0             1          NaN
128     R Dravid  ...            1          0             1          NaN
...          ...  ...          ...        ...           ...          ...
214508   SS Iyer  ...            1          0             1          NaN
214509   VR Iyer  ...            1          0             1          NaN
214510   VR Iyer  ...            1          0             1          NaN
214511   SS Iyer  ...            1          0             1          NaN
214512   VR Iyer  ...            1          0             1          NaN

        is_wicket player_dismissed  dismissal_kind fielder current_score  \
124             0              NaN             NaN     NaN             1
125             0              NaN             NaN     NaN             2
126             0              NaN             NaN     NaN             2
127             0              NaN             NaN     NaN             3
128             0              NaN             NaN     NaN             4
...           ...              ...             ...     ...           ...
214508          0              NaN             NaN     NaN           110
214509          0              NaN             NaN     NaN           111
214510          0              NaN             NaN     NaN           112
214511          0              NaN             NaN     NaN           113
214512          0              NaN             NaN     NaN           114

       runs_left
124          222
125          221
126          221
127          220
128          219
...          ...
214508         4
214509         3
214510         2
214511         1
214512         0

[103793 rows x 22 columns]
```

[150]:

```
#over and ball of which over features available
#balls left
delivery_df['balls_left'] = 120 - (delivery_df['over'] * 6 + delivery_df['ball'])
```

[151]:

```
delivery_df
```

[151]:

```
        match_id       city                 winner  total_runs_x  inning  \
124       335982  Bangalore  Kolkata Knight Riders           222       2
125       335982  Bangalore  Kolkata Knight Riders           222       2
126       335982  Bangalore  Kolkata Knight Riders           222       2
127       335982  Bangalore  Kolkata Knight Riders           222       2
128       335982  Bangalore  Kolkata Knight Riders           222       2
...          ...        ...                    ...           ...     ...
214508   1426312    Chennai  Kolkata Knight Riders           113       2
214509   1426312    Chennai  Kolkata Knight Riders           113       2
214510   1426312    Chennai  Kolkata Knight Riders           113       2
214511   1426312    Chennai  Kolkata Knight Riders           113       2
214512   1426312    Chennai  Kolkata Knight Riders           113       2

                       batting_team           bowling_team  over  ball  \
124     Royal Challengers Bangalore  Kolkata Knight Riders     0     1
125     Royal Challengers Bangalore  Kolkata Knight Riders     0     2
126     Royal Challengers Bangalore  Kolkata Knight Riders     0     3
127     Royal Challengers Bangalore  Kolkata Knight Riders     0     4
128     Royal Challengers Bangalore  Kolkata Knight Riders     0     5
...                             ...                    ...   ...   ...
214508        Kolkata Knight Riders    Sunrisers Hyderabad     9     5
214509        Kolkata Knight Riders    Sunrisers Hyderabad     9     6
214510        Kolkata Knight Riders    Sunrisers Hyderabad    10     1
214511        Kolkata Knight Riders    Sunrisers Hyderabad    10     2
214512        Kolkata Knight Riders    Sunrisers Hyderabad    10     3

          batter  ... extra_runs total_runs_y  extras_type  is_wicket  \
124     R Dravid  ...          0            1          NaN          0
125     W Jaffer  ...          1            1        wides          0
126     W Jaffer  ...          0            0          NaN          0
127     W Jaffer  ...          0            1          NaN          0
128     R Dravid  ...          0            1          NaN          0
...          ...  ...        ...          ...          ...        ...
214508   SS Iyer  ...          0            1          NaN          0
214509   VR Iyer  ...          0            1          NaN          0
214510   VR Iyer  ...          0            1          NaN          0
214511   SS Iyer  ...          0            1          NaN          0
214512   VR Iyer  ...          0            1          NaN          0

        player_dismissed dismissal_kind  fielder current_score runs_left  \
124                  NaN            NaN      NaN             1       222
125                  NaN            NaN      NaN             2       221
126                  NaN            NaN      NaN             2       221
127                  NaN            NaN      NaN             3       220
128                  NaN            NaN      NaN             4       219
...                  ...            ...      ...           ...       ...
214508               NaN            NaN      NaN           110         4
214509               NaN            NaN      NaN           111         3
214510               NaN            NaN      NaN           112         2
214511               NaN            NaN      NaN           113         1
214512               NaN            NaN      NaN           114         0

       balls_left
124           119
125           118
126           117
127           116
128           115
...           ...
214508         61
214509         60
```

```
214510         59
214511         58
214512         57

[103793 rows x 23 columns]
```

[152]:
```
#Player dismissed -if out then mentioned
delivery_df['player_dismissed'] = delivery_df['player_dismissed'].fillna("0")
delivery_df['player_dismissed'] = delivery_df['player_dismissed'].apply(lambda x:
 ↪x if x=="0" else "1")
delivery_df['player_dismissed'] = delivery_df['player_dismissed'].astype('int')
wickets = delivery_df.groupby('match_id')['is_wicket'].cumsum().astype('int')
delivery_df['wickets'] = 10 - wickets
delivery_df.head()
```

[152]:
```
     match_id       city                 winner  total_runs_x  inning  \
124    335982  Bangalore  Kolkata Knight Riders           222       2
125    335982  Bangalore  Kolkata Knight Riders           222       2
126    335982  Bangalore  Kolkata Knight Riders           222       2
127    335982  Bangalore  Kolkata Knight Riders           222       2
128    335982  Bangalore  Kolkata Knight Riders           222       2

                    batting_team           bowling_team  over  ball    batter  \
124  Royal Challengers Bangalore  Kolkata Knight Riders     0     1  R Dravid
125  Royal Challengers Bangalore  Kolkata Knight Riders     0     2  W Jaffer
126  Royal Challengers Bangalore  Kolkata Knight Riders     0     3  W Jaffer
127  Royal Challengers Bangalore  Kolkata Knight Riders     0     4  W Jaffer
128  Royal Challengers Bangalore  Kolkata Knight Riders     0     5  R Dravid

     ... total_runs_y extras_type  is_wicket  player_dismissed  \
124  ...            1         NaN          0                 0
125  ...            1       wides          0                 0
126  ...            0         NaN          0                 0
127  ...            1         NaN          0                 0
128  ...            1         NaN          0                 0

     dismissal_kind fielder  current_score  runs_left balls_left wickets
124             NaN     NaN              1        222        119      10
125             NaN     NaN              2        221        118      10
126             NaN     NaN              2        221        117      10
127             NaN     NaN              3        220        116      10
128             NaN     NaN              4        219        115      10

[5 rows x 24 columns]
```

[153]:
```
delivery_df['wickets'].unique()
```

[153]:
```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1,  0])
```

[154]:
```
#crr=runs/((balls left)/6)
delivery_df['crr'] = (delivery_df['current_score']*6)/(120 -␣
↪delivery_df['balls_left'])
```

[155]:
```
delivery_df
```

[155]:
```
        match_id       city                 winner  total_runs_x  inning  \
124       335982  Bangalore  Kolkata Knight Riders           222       2   
125       335982  Bangalore  Kolkata Knight Riders           222       2   
126       335982  Bangalore  Kolkata Knight Riders           222       2   
127       335982  Bangalore  Kolkata Knight Riders           222       2   
128       335982  Bangalore  Kolkata Knight Riders           222       2   
...          ...        ...                    ...           ...     ...   
214508   1426312    Chennai  Kolkata Knight Riders           113       2   
214509   1426312    Chennai  Kolkata Knight Riders           113       2   
214510   1426312    Chennai  Kolkata Knight Riders           113       2   
214511   1426312    Chennai  Kolkata Knight Riders           113       2   
214512   1426312    Chennai  Kolkata Knight Riders           113       2   

                       batting_team           bowling_team  over  ball  \
124     Royal Challengers Bangalore  Kolkata Knight Riders     0     1   
125     Royal Challengers Bangalore  Kolkata Knight Riders     0     2   
126     Royal Challengers Bangalore  Kolkata Knight Riders     0     3   
127     Royal Challengers Bangalore  Kolkata Knight Riders     0     4   
128     Royal Challengers Bangalore  Kolkata Knight Riders     0     5   
...                             ...                    ...   ...   ...   
214508        Kolkata Knight Riders    Sunrisers Hyderabad     9     5   
214509        Kolkata Knight Riders    Sunrisers Hyderabad     9     6   
214510        Kolkata Knight Riders    Sunrisers Hyderabad    10     1   
214511        Kolkata Knight Riders    Sunrisers Hyderabad    10     2   
214512        Kolkata Knight Riders    Sunrisers Hyderabad    10     3   

          batter  ... extras_type is_wicket  player_dismissed  dismissal_kind  \
124     R Dravid  ...         NaN         0                 0             NaN   
125     W Jaffer  ...       wides         0                 0             NaN   
126     W Jaffer  ...         NaN         0                 0             NaN   
127     W Jaffer  ...         NaN         0                 0             NaN   
128     R Dravid  ...         NaN         0                 0             NaN   
...          ...  ...         ...       ...               ...             ...   
214508   SS Iyer  ...         NaN         0                 0             NaN   
214509   VR Iyer  ...         NaN         0                 0             NaN   
214510   VR Iyer  ...         NaN         0                 0             NaN   
214511   SS Iyer  ...         NaN         0                 0             NaN   
214512   VR Iyer  ...         NaN         0                 0             NaN   
```

```
        fielder current_score  runs_left  balls_left wickets        crr
124         NaN             1        222         119      10   6.000000
125         NaN             2        221         118      10   6.000000
126         NaN             2        221         117      10   4.000000
127         NaN             3        220         116      10   4.500000
128         NaN             4        219         115      10   4.800000
...         ...           ...        ...         ...     ...        ...
214508      NaN           110          4          61       8  11.186441
214509      NaN           111          3          60       8  11.100000
214510      NaN           112          2          59       8  11.016393
214511      NaN           113          1          58       8  10.935484
214512      NaN           114          0          57       8  10.857143

[103793 rows x 25 columns]
```

```
[156]: delivery_df['rrr'] = (delivery_df['runs_left']*6)/(delivery_df['balls_left'])
```

```
[157]: delivery_df
```

```
[157]:         match_id       city                 winner  total_runs_x  inning  \
       124       335982  Bangalore  Kolkata Knight Riders           222       2
       125       335982  Bangalore  Kolkata Knight Riders           222       2
       126       335982  Bangalore  Kolkata Knight Riders           222       2
       127       335982  Bangalore  Kolkata Knight Riders           222       2
       128       335982  Bangalore  Kolkata Knight Riders           222       2
       ...          ...        ...                    ...           ...     ...
       214508   1426312    Chennai  Kolkata Knight Riders           113       2
       214509   1426312    Chennai  Kolkata Knight Riders           113       2
       214510   1426312    Chennai  Kolkata Knight Riders           113       2
       214511   1426312    Chennai  Kolkata Knight Riders           113       2
       214512   1426312    Chennai  Kolkata Knight Riders           113       2

                          batting_team           bowling_team  over  ball  \
       124     Royal Challengers Bangalore  Kolkata Knight Riders     0     1
       125     Royal Challengers Bangalore  Kolkata Knight Riders     0     2
       126     Royal Challengers Bangalore  Kolkata Knight Riders     0     3
       127     Royal Challengers Bangalore  Kolkata Knight Riders     0     4
       128     Royal Challengers Bangalore  Kolkata Knight Riders     0     5
       ...                             ...                    ...   ...   ...
       214508        Kolkata Knight Riders    Sunrisers Hyderabad     9     5
       214509        Kolkata Knight Riders    Sunrisers Hyderabad     9     6
       214510        Kolkata Knight Riders    Sunrisers Hyderabad    10     1
       214511        Kolkata Knight Riders    Sunrisers Hyderabad    10     2
       214512        Kolkata Knight Riders    Sunrisers Hyderabad    10     3

                 batter  ... is_wicket player_dismissed  dismissal_kind  fielder  \
       124     R Dravid  ...         0                0             NaN      NaN
```

```
125     W Jaffer  ...         0                0             NaN      NaN
126     W Jaffer  ...         0                0             NaN      NaN
127     W Jaffer  ...         0                0             NaN      NaN
128     R Dravid  ...         0                0             NaN      NaN
...          ...  ...       ...              ...             ...      ...
214508   SS Iyer  ...         0                0             NaN      NaN
214509   VR Iyer  ...         0                0             NaN      NaN
214510   VR Iyer  ...         0                0             NaN      NaN
214511   SS Iyer  ...         0                0             NaN      NaN
214512   VR Iyer  ...         0                0             NaN      NaN

        current_score  runs_left  balls_left  wickets        crr        rrr
124                 1        222         119       10   6.000000  11.193277
125                 2        221         118       10   6.000000  11.237288
126                 2        221         117       10   4.000000  11.333333
127                 3        220         116       10   4.500000  11.379310
128                 4        219         115       10   4.800000  11.426087
...               ...        ...         ...      ...        ...        ...
214508            110          4          61        8  11.186441   0.393443
214509            111          3          60        8  11.100000   0.300000
214510            112          2          59        8  11.016393   0.203390
214511            113          1          58        8  10.935484   0.103448
214512            114          0          57        8  10.857143   0.000000

[103793 rows x 26 columns]
```

[158]:

```
def result(row):
  return 1 if row['batting_team']==row['winner'] else 0

delivery_df.apply(result,axis=1)
```

[158]:

```
124       0
125       0
126       0
127       0
128       0
         ..
214508    1
214509    1
214510    1
214511    1
214512    1
Length: 103793, dtype: int64
```

[159]:

```
delivery_df['result']=delivery_df.apply(result,axis=1)
```

[160]:

```
final_df=delivery_df[['batting_team','bowling_team','city','runs_left','balls_left','wickets','
```

[161]:
```
final_df
```

[161]:
```
                       batting_team           bowling_team       city  \
124     Royal Challengers Bangalore  Kolkata Knight Riders  Bangalore   
125     Royal Challengers Bangalore  Kolkata Knight Riders  Bangalore   
126     Royal Challengers Bangalore  Kolkata Knight Riders  Bangalore   
127     Royal Challengers Bangalore  Kolkata Knight Riders  Bangalore   
128     Royal Challengers Bangalore  Kolkata Knight Riders  Bangalore   
...                             ...                    ...        ...   
214508        Kolkata Knight Riders    Sunrisers Hyderabad    Chennai   
214509        Kolkata Knight Riders    Sunrisers Hyderabad    Chennai   
214510        Kolkata Knight Riders    Sunrisers Hyderabad    Chennai   
214511        Kolkata Knight Riders    Sunrisers Hyderabad    Chennai   
214512        Kolkata Knight Riders    Sunrisers Hyderabad    Chennai   

        runs_left  balls_left  wickets  total_runs_x        crr        rrr  \
124           222         119       10           222   6.000000  11.193277   
125           221         118       10           222   6.000000  11.237288   
126           221         117       10           222   4.000000  11.333333   
127           220         116       10           222   4.500000  11.379310   
128           219         115       10           222   4.800000  11.426087   
...           ...         ...      ...           ...        ...        ...   
214508          4          61        8           113  11.186441   0.393443   
214509          3          60        8           113  11.100000   0.300000   
214510          2          59        8           113  11.016393   0.203390   
214511          1          58        8           113  10.935484   0.103448   
214512          0          57        8           113  10.857143   0.000000   

        result  
124          0  
125          0  
126          0  
127          0  
128          0  
...        ...  
214508       1  
214509       1  
214510       1  
214511       1  
214512       1  

[103793 rows x 10 columns]
```

[162]:
```
final_df=final_df.sample(final_df.shape[0])
```

[163]:
```
final_df.sample()
```

[163]:

```
                batting_team    bowling_team       city  runs_left  balls_left  \
152738  Kings XI Punjab  Mumbai Indians  Abu Dhabi         93          42

        wickets  total_runs_x       crr        rrr  result
152738        7           191  7.615385  13.285714       0
```

[164]:

```
from google.colab import files
final_df.to_csv('compiled.csv', encoding = 'utf-8-sig')
files.download('compiled.csv')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[165]:

```
final_df.isnull().sum()
```

[165]:

```
batting_team       0
bowling_team       0
city            6012
runs_left          0
balls_left         0
wickets            0
total_runs_x       0
crr                0
rrr               14
result             0
dtype: int64
```

[166]:

```
final_df.dropna(inplace=True)
```

[167]:

```
final_df.shape
```

[167]:

```
(97767, 10)
```

[168]:

```
final_df.isnull().sum()
```

[168]:

```
batting_team    0
bowling_team    0
city            0
runs_left       0
balls_left      0
wickets         0
total_runs_x    0
crr             0
rrr             0
result          0
dtype: int64
```

[169]:

```
final_df=final_df[final_df['balls_left']!=0]
```

```
[170]: X=final_df.iloc[:,:-1]
       y=final_df.iloc[:,-1]
       from sklearn.model_selection import train_test_split
       X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
```

```
[171]: X_train
```

```
[171]:                        batting_team                 bowling_team  \
16275         Kolkata Knight Riders               Mumbai Indians
204     Royal Challengers Bangalore        Kolkata Knight Riders
35929               Deccan Chargers  Royal Challengers Bangalore
12257                Mumbai Indians  Royal Challengers Bangalore
65895              Delhi Daredevils             Rajasthan Royals
...                             ...                          ...
131876          Chennai Super Kings          Sunrisers Hyderabad
207059               Gujarat Titans         Lucknow Super Giants
11463         Kolkata Knight Riders              Kings XI Punjab
10938           Chennai Super Kings             Rajasthan Royals
17465              Rajasthan Royals          Chennai Super Kings

                  city  runs_left  balls_left  wickets  total_runs_x  \
16275   Port Elizabeth        108          47        6           187
204          Bangalore        156          46        2           222
35929        Bangalore        155          91        9           184
12257        Bangalore         -3          24        9           122
65895            Delhi         98          64        9           165
...                ...        ...         ...      ...           ...
131876            Pune         87          60       10           179
207059         Lucknow        160         115       10           163
11463          Kolkata        120          74        8           174
10938          Chennai        193         106        9           211
17465        Centurion        103          59        7           164

             crr        rrr
16275   6.575342  13.787234
204     5.432432  20.347826
35929   6.206897  10.219780
12257   7.875000  -0.750000
65895   7.285714   9.187500
...          ...        ...
131876  9.300000   8.700000
207059  4.800000   8.347826
11463   7.173913   9.729730
10938   8.142857  10.924528
17465   6.098361  10.474576

[77952 rows x 9 columns]
```

8.3.1 One Hot Encoding the team names and the venues:

[172]:
```
#Convert strings to numerical -one hot encoding
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
trf=ColumnTransformer([
   ␣
 ↪('trf',OneHotEncoder(sparse_output=False,drop='first'),['batting_team','bowling_team','city']
],remainder='passthrough')
```

[173]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
```

###Model 1: Logistic regression

[174]:
```
pipe=Pipeline(steps=[
    ('step1',trf),
    ('step2',LogisticRegression(solver='liblinear'))
])
```

[175]:
```
pipe.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[175]:
```
Pipeline(steps=[('step1',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('trf',
                                                  OneHotEncoder(drop='first',
sparse_output=False),
                                                  ['batting_team',
                                                   'bowling_team', 'city'])])),
                ('step2', LogisticRegression(solver='liblinear'))])
```

[176]:
```
X_train.describe()
```

```
[176]:          runs_left     balls_left       wickets  total_runs_x           crr  \
       count  77952.000000  77952.000000  77952.000000  77952.000000  77952.000000
       mean      94.526555     62.743932      7.521064    167.708474      7.539825
       std       50.649307     33.313588      2.149871     30.293416      2.326179
       min       -5.000000     -2.000000      0.000000     62.000000      0.000000
       25%       55.000000     35.000000      6.000000    148.000000      6.352941
       50%       93.000000     63.000000      8.000000    167.000000      7.578947
       75%      132.000000     92.000000      9.000000    187.000000      8.790698
       max      274.000000    119.000000     10.000000    277.000000     36.000000

                       rrr
       count  77952.000000
       mean      10.727557
       std       14.358119
       min     -516.000000
       25%        7.309091
       50%        9.042254
       75%       11.264151
       max      714.000000
```

```
[177]: y_pred=pipe.predict(X_test)
```

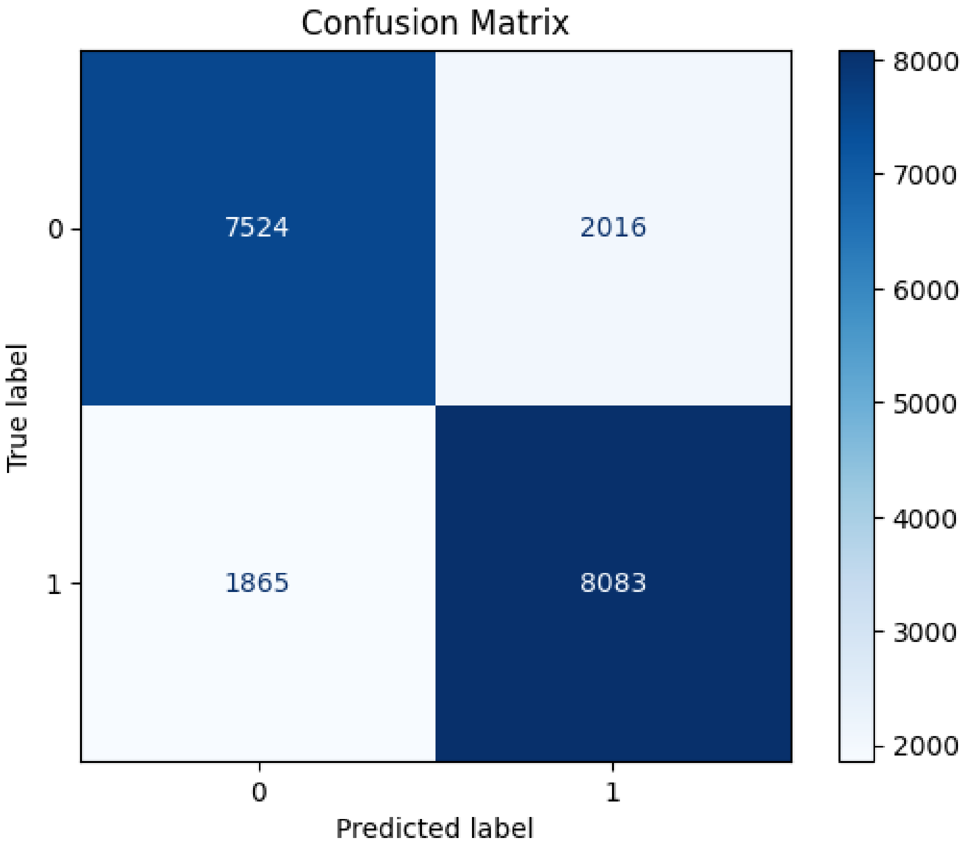
```
[178]: from sklearn.metrics import accuracy_score
```

```
[179]: accuracy_score(y_test,y_pred)
```

```
[179]: 0.8013649425287356
```

The Evaluation Metrics of Logistic Regression Model

```
[ ]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
     import matplotlib.pyplot as plt

     cm = confusion_matrix(y_test, y_pred)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm)
     disp.plot(cmap='Blues')
     plt.title("Confusion Matrix")
     plt.show()
```



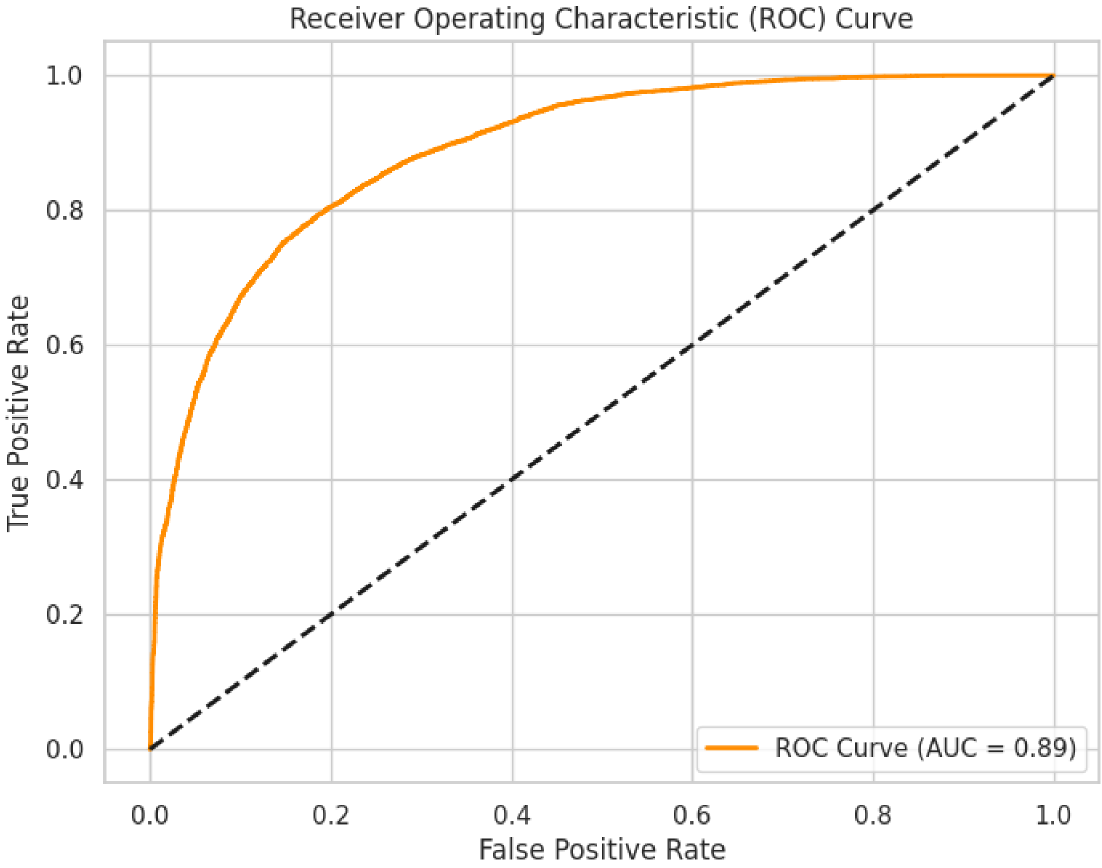
Interpreting the ROC-AUC Plot The ROC curve plots the True Positive Rate (TPR) vs. False Positive Rate (FPR).

A higher AUC (Area Under the Curve) means a better-performing model.

1. AUC 1.0 → Perfect classification
2. AUC 0.5 → Model is performing randomly (no predictive power)
3. AUC < 0.5 → Model is worse than random guessing

[180]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Get predicted probabilities for class 1
y_prob = pipe.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC score
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
```

```
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.
 ↪2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)  # Random guessing line

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



[181]:
```
from sklearn.ensemble import RandomForestClassifier # Import␣
 ↪RandomForestClassifier from the correct submodule
from sklearn.pipeline import Pipeline
```

###Model 2: Random Forest Classifier https://miro.medium.com/v2/resize:fit:1400/1*hmtbIgxoflflJqMJ

Random Forest is primarily a classification and regression algorithm. When used for regression tasks, it predicts continuous values by **averaging the outputs of multiple decision trees**.

Why does Random Forest give accurate results? 1. Ensemble Learning – It combines multiple decision trees to reduce overfitting.

2. Averaging Effect – In regression, the final output is the average of predictions, which smooths errors.

3. Feature Randomness – Each tree gets a random subset of features, making the model more robust.

[182]:
```
pipe2=Pipeline(steps=[
    ('step1',trf),
    ('step2',RandomForestClassifier())
])
```

[ ]:
```
pipe2.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[ ]:
```
Pipeline(steps=[('step1',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('trf',
                                                  OneHotEncoder(drop='first',
sparse_output=False),
                                                  ['batting_team',
                                                   'bowling_team', 'city'])])),
                ('step2', RandomForestClassifier())])
```

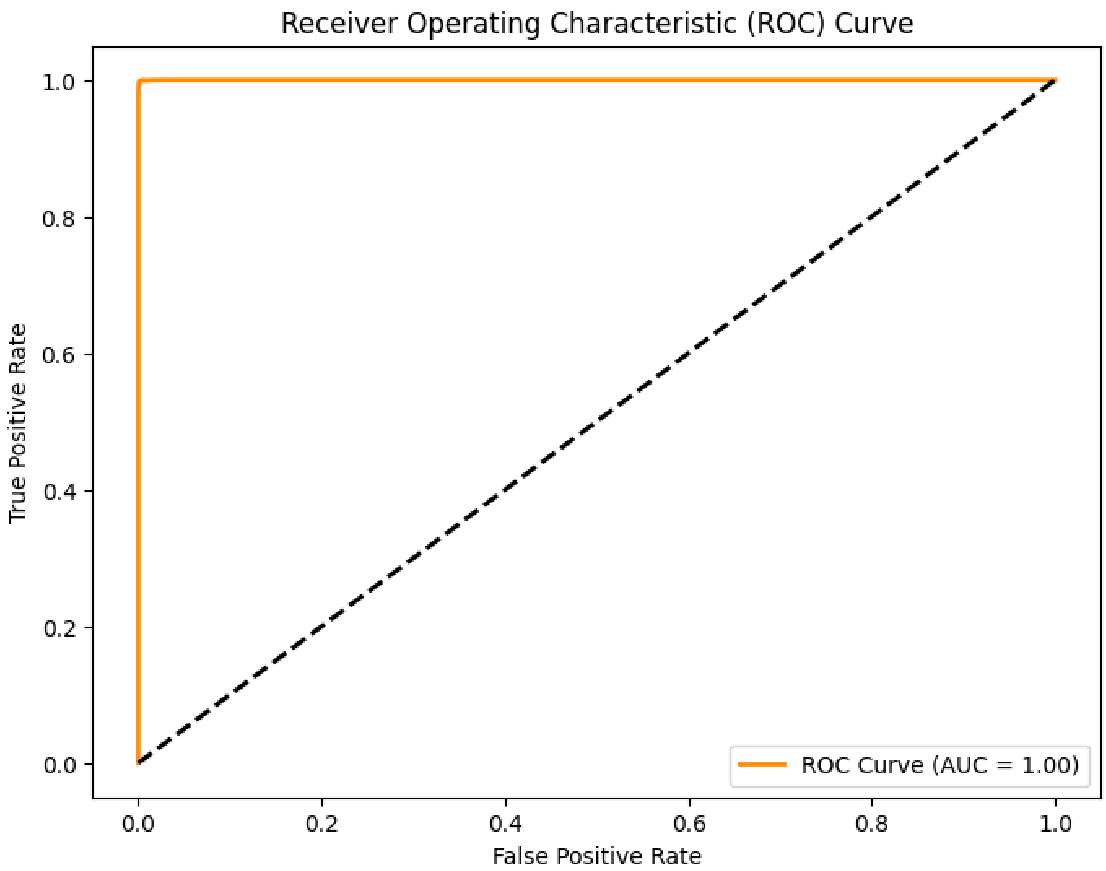
[ ]:
```
y_pred=pipe2.predict(X_test)
```

[ ]:
```
accuracy_score(y_test,y_pred)
```

```
[ ]: 0.9984605911330049
```

Evaluation Metrics of Random Forest:

```
[ ]: import matplotlib.pyplot as plt
     from sklearn.metrics import roc_curve, auc

     # Get predicted probabilities for class 1
     y_prob = pipe2.predict_proba(X_test)[:, 1]

     # Compute ROC curve and AUC score
     fpr, tpr, _ = roc_curve(y_test, y_prob)
     roc_auc = auc(fpr, tpr)

     # Plot ROC Curve
     plt.figure(figsize=(8,6))
     plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.
      ↪2f})')
     plt.plot([0, 1], [0, 1], 'k--', lw=2)  # Random guessing line

     plt.xlabel('False Positive Rate')
     plt.ylabel('True Positive Rate')
     plt.title('Receiver Operating Characteristic (ROC) Curve')
     plt.legend(loc='lower right')
     plt.show()
```



###Model 3: Training Ensemble : XGBoost + Random Forest

[183]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix, roc_curve, auc

# Sample dataset (Replace with your dataset)
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20, random_state=42)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[184]:

```
# Define models
rf = RandomForestClassifier(n_estimators=100, random_state=42)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
 ↪random_state=42)

# Voting Classifier (Soft Voting for probabilities)
ensemble = VotingClassifier(estimators=[('rf', rf), ('xgb', xgb)], voting='soft')

# Train ensemble
ensemble.fit(X_train, y_train)

# Predict
y_pred = ensemble.predict(X_test)
y_prob = ensemble.predict_proba(X_test)[:, 1]  # Probabilities for ROC
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning:
[06:28:30] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

[185]:

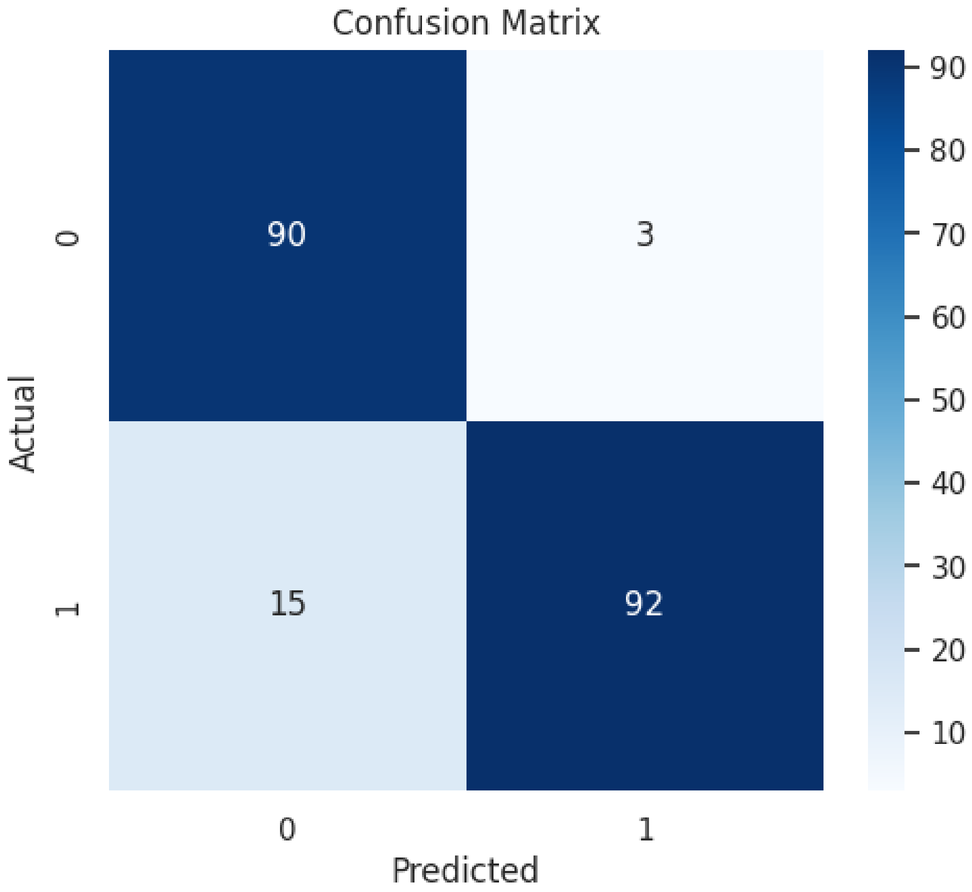
```
acc = accuracy_score(y_test, y_pred)
print(f"Ensemble Model Accuracy: {acc:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Ensemble Model Accuracy: 0.9100

Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.97      0.91        93
           1       0.97      0.86      0.91       107

    accuracy                           0.91       200
   macro avg       0.91      0.91      0.91       200
weighted avg       0.92      0.91      0.91       200
```

Evaluation Metrics of Ensemble Method:

[186]:

```
plt.figure(figsize=(6,5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



8.3.2 Model 4: Neural Network:

(more specifically - a Multi Layer Perceptron)

[189]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Define and Train MLP Neural Network with 3 Hidden Layers
mlp = MLPClassifier(hidden_layer_sizes=(128, 64, 32), activation='relu',␣
 ↪solver='adam',
                    max_iter=1000, random_state=42)

# Train the model
mlp.fit(X_train, y_train)

# Predict
y_pred_nn = mlp.predict(X_test)
y_prob_nn = mlp.predict_proba(X_test)[:, 1]
```

```
# Accuracy
acc_nn = accuracy_score(y_test, y_pred_nn)
print(f"Neural Network Accuracy: {acc_nn:.4f}")

# Display Model Pipeline
print("\nMLP Model Structure:")
print(f"Input Layer: {X_train.shape[1]} neurons")
print(f"Hidden Layer 1: 128 neurons (ReLU)")
print(f"Hidden Layer 2: 64 neurons (ReLU)")
print(f"Hidden Layer 3: 32 neurons (ReLU)")
print(f"Output Layer: 1 neuron (Sigmoid for binary classification- Win or Lose)")
```

```
Neural Network Accuracy: 0.8450

MLP Model Structure:
Input Layer: 20 neurons
Hidden Layer 1: 128 neurons (ReLU)
Hidden Layer 2: 64 neurons (ReLU)
Hidden Layer 3: 32 neurons (ReLU)
Output Layer: 1 neuron (Sigmoid for binary classification- Win or Lose)
```

[190]:

```
from graphviz import Digraph

def draw_mlp():
    dot = Digraph()

    # Input Layer
    for i in range(1, 21):  # 20 input neurons
        dot.node(f'I{i}', f'Input {i}', shape='circle', style='filled',␣
 ↪fillcolor='lightgray')

    # Hidden Layer 1
    for i in range(1, 129):  # 128 neurons
        dot.node(f'H1-{i}', f'H1-{i}', shape='circle', style='filled',␣
 ↪fillcolor='lightblue')
        for j in range(1, 21):
            dot.edge(f'I{j}', f'H1-{i}')

    # Hidden Layer 2
    for i in range(1, 65):  # 64 neurons
        dot.node(f'H2-{i}', f'H2-{i}', shape='circle', style='filled',␣
 ↪fillcolor='lightgreen')
        for j in range(1, 129):
            dot.edge(f'H1-{j}', f'H2-{i}')

    # Hidden Layer 3
    for i in range(1, 33):  # 32 neurons
```

```
        dot.node(f'H3-{i}', f'H3-{i}', shape='circle', style='filled',␣
 ↪fillcolor='orange')
        for j in range(1, 65):
            dot.edge(f'H2-{j}', f'H3-{i}')

    # Output Layer
    dot.node('O', 'Output', shape='circle', style='filled', fillcolor='red')
    for j in range(1, 33):
        dot.edge(f'H3-{j}', 'O')

    # Save and render
    dot.render('mlp_architecture', format='png', cleanup=False)
    print("MLP Architecture diagram saved as mlp_architecture.png")

# Generate the MLP Diagram
draw_mlp()
```

```
MLP Architecture diagram saved as mlp_architecture.png
```

[191]:

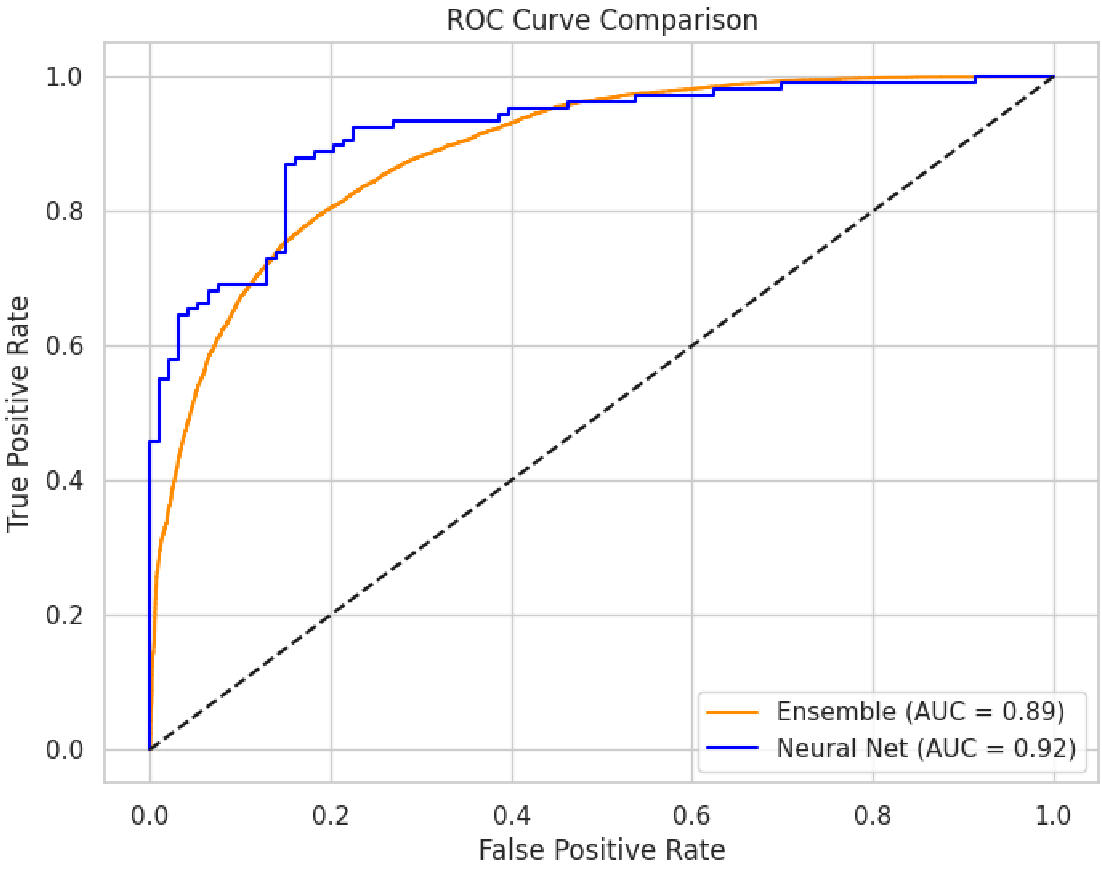
```
from IPython.display import display
from PIL import Image

# Load and display the image
img = Image.open("mlp_architecture.png")  # Replace with your file name
display(img)
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Evaluation Metrics:

[192]:

```
fpr_nn, tpr_nn, _ = roc_curve(y_test, y_prob_nn)
roc_auc_nn = auc(fpr_nn, tpr_nn)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'Ensemble (AUC = {roc_auc:.2f})', color='darkorange')
plt.plot(fpr_nn, tpr_nn, label=f'Neural Net (AUC = {roc_auc_nn:.2f})',␣
 ↪color='blue')
plt.plot([0, 1], [0, 1], 'k--')  # Random guessing line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.show()
```



8.3.3 Conclusion and Our Approach

In our analysis, we intentionally avoided creating an overly perfect model. A model that consistently predicts extreme probabilities—such as **Team A: 99% vs. Team B: 1%**—could indicate overfitting and excessive bias, making it unreliable in real-world scenarios.

Instead, we aimed for a more balanced and **moderate prediction approach**, where the probabilities reflect realistic uncertainties in match outcomes. Given the nature of our predictors, the model provides reasonable probability distributions rather than definitive, one-sided predictions.

With this in mind, **Logistic Regression** was our model of choice for deployment on **Streamlit**. Its simplicity, interpretability, and ability to maintain moderate probability estimates make it a suitable candidate for predicting match outcomes without being excessively confident in its predictions.

This ensures that users receive insightful yet **realistic** probability distributions, fostering a more engaging and analytically sound experience.

8.4 Deployment:

Our Deployed Streamlit App:

Exporting model and Creating a streamlit app:

```
[ ]: teams
```

```
[ ]: ['Royal Challengers Bangalore',
      'Kings XI Punjab',
      'Mumbai Indians',
      'Kolkata Knight Riders',
      'Rajasthan Royals',
      'Chennai Super Kings',
      'Sunrisers Hyderabad',
      'Delhi Capitals',
      'Lucknow Super Giants',
      'Gujarat Titans']
```

```
[ ]: delivery_df['city'].unique()
```

```
[ ]: array(['Bangalore', 'Chandigarh', 'Delhi', 'Mumbai', 'Kolkata', 'Jaipur',
            'Hyderabad', 'Chennai', 'Cape Town', 'Port Elizabeth', 'Durban',
            'Centurion', 'East London', 'Johannesburg', 'Kimberley',
            'Bloemfontein', 'Ahmedabad', 'Cuttack', 'Nagpur', 'Dharamsala',
            'Visakhapatnam', 'Pune', 'Raipur', 'Ranchi', 'Abu Dhabi', nan,
            'Bengaluru', 'Indore', 'Dubai', 'Sharjah', 'Navi Mumbai',
            'Lucknow', 'Guwahati'], dtype=object)
```

Exported the model as required, proceed to streamlit app to witness live prediction.

```
[ ]: import pickle
     pickle.dump(pipe,open('pipe.pkl','wb'))
```

#Thank You!