IPL_Analysis (1)

March 24, 2025

To view the full plots which are missing, go to this Google Colab link

1 This Notebook explores the Datasets `Deliveries.csv` and `Matches.csv` provided by BrainDead Organising Comittee

Importing necessary Libraries for Data Abalysis and Exploration

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly
import plotly.express as px
import plotly.graph_objects as go
import os
```

Importing the datasets to work with

[2]:
```
deliveries_df = pd.read_csv('Brain Dead IPL Dataset\deliveries.csv')
matches_df = pd.read_csv("Brain Dead IPL Dataset\matches.csv")
```

```
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:2: SyntaxWarning: invalid escape sequence '\m'
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:2: SyntaxWarning: invalid escape sequence '\m'
C:\Users\SOHAM\AppData\Local\Temp\ipykernel_1684\2917070527.py:1: SyntaxWarning:
invalid escape sequence '\d'
  deliveries_df = pd.read_csv('Brain Dead IPL Dataset\deliveries.csv')
C:\Users\SOHAM\AppData\Local\Temp\ipykernel_1684\2917070527.py:2: SyntaxWarning:
invalid escape sequence '\m'
  matches_df = pd.read_csv("Brain Dead IPL Dataset\matches.csv")
```

[3]:
```
deliveries_df
```

[3]:

	match_id	inning	batting_team	bowling_team	\
0	335982	1	Kolkata Knight Riders	Royal Challengers Bangalore	
1	335982	1	Kolkata Knight Riders	Royal Challengers Bangalore	

```
2         335982       1  Kolkata Knight Riders  Royal Challengers Bangalore
3         335982       1  Kolkata Knight Riders  Royal Challengers Bangalore
4         335982       1  Kolkata Knight Riders  Royal Challengers Bangalore
...          ...     ...                    ...                          ...
260915   1426312       2  Kolkata Knight Riders          Sunrisers Hyderabad
260916   1426312       2  Kolkata Knight Riders          Sunrisers Hyderabad
260917   1426312       2  Kolkata Knight Riders          Sunrisers Hyderabad
260918   1426312       2  Kolkata Knight Riders          Sunrisers Hyderabad
260919   1426312       2  Kolkata Knight Riders          Sunrisers Hyderabad

        over  ball       batter         bowler  non_striker  batsman_runs  \
0          0     1   SC Ganguly        P Kumar  BB McCullum             0
1          0     2  BB McCullum        P Kumar   SC Ganguly             0
2          0     3  BB McCullum        P Kumar   SC Ganguly             0
3          0     4  BB McCullum        P Kumar   SC Ganguly             0
4          0     5  BB McCullum        P Kumar   SC Ganguly             0
...      ...   ...          ...            ...          ...           ...
260915     9     5      SS Iyer     AK Markram      VR Iyer             1
260916     9     6      VR Iyer     AK Markram      SS Iyer             1
260917    10     1      VR Iyer  Shahbaz Ahmed      SS Iyer             1
260918    10     2      SS Iyer  Shahbaz Ahmed      VR Iyer             1
260919    10     3      VR Iyer  Shahbaz Ahmed      SS Iyer             1

        extra_runs  total_runs extras_type  is_wicket player_dismissed  \
0                1           1     legbyes          0              NaN
1                0           0         NaN          0              NaN
2                1           1       wides          0              NaN
3                0           0         NaN          0              NaN
4                0           0         NaN          0              NaN
...            ...         ...         ...        ...              ...
260915           0           1         NaN          0              NaN
260916           0           1         NaN          0              NaN
260917           0           1         NaN          0              NaN
260918           0           1         NaN          0              NaN
260919           0           1         NaN          0              NaN

       dismissal_kind fielder
0                 NaN     NaN
1                 NaN     NaN
2                 NaN     NaN
3                 NaN     NaN
4                 NaN     NaN
...               ...     ...
260915            NaN     NaN
260916            NaN     NaN
260917            NaN     NaN
260918            NaN     NaN
```

```
260919            NaN     NaN

[260920 rows x 17 columns]
```

[4]:
```
matches_df
```

[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
...       ...      ...         ...         ...          ...              ...
1090  1426307     2024   Hyderabad  2024-05-19       League  Abhishek Sharma
1091  1426309     2024   Ahmedabad  2024-05-21  Qualifier 1         MA Starc
1092  1426310     2024   Ahmedabad  2024-05-22   Eliminator         R Ashwin
1093  1426311     2024     Chennai  2024-05-24  Qualifier 2    Shahbaz Ahmed
1094  1426312     2024     Chennai  2024-05-26        Final         MA Starc

                                              venue  \
0                             M Chinnaswamy Stadium
1          Punjab Cricket Association Stadium, Mohali
2                                  Feroz Shah Kotla
3                                  Wankhede Stadium
4                                      Eden Gardens
...                                             ...
1090  Rajiv Gandhi International Stadium, Uppal, Hyd...
1091                  Narendra Modi Stadium, Ahmedabad
1092                  Narendra Modi Stadium, Ahmedabad
1093          MA Chidambaram Stadium, Chepauk, Chennai
1094          MA Chidambaram Stadium, Chepauk, Chennai

                         team1                         team2  \
0     Royal Challengers Bangalore        Kolkata Knight Riders
1                 Kings XI Punjab          Chennai Super Kings
2                Delhi Daredevils             Rajasthan Royals
3                  Mumbai Indians  Royal Challengers Bangalore
4           Kolkata Knight Riders              Deccan Chargers
...                           ...                           ...
1090                Punjab Kings          Sunrisers Hyderabad
1091          Sunrisers Hyderabad        Kolkata Knight Riders
1092  Royal Challengers Bengaluru             Rajasthan Royals
1093          Sunrisers Hyderabad             Rajasthan Royals
1094          Sunrisers Hyderabad        Kolkata Knight Riders

                     toss_winner toss_decision                        winner  \
0     Royal Challengers Bangalore         field        Kolkata Knight Riders
```

```
1              Chennai Super Kings           bat          Chennai Super Kings
2                 Rajasthan Royals           bat             Delhi Daredevils
3                   Mumbai Indians           bat  Royal Challengers Bangalore
4                   Deccan Chargers           bat        Kolkata Knight Riders
...                             ...           ...                          ...
1090                  Punjab Kings           bat          Sunrisers Hyderabad
1091            Sunrisers Hyderabad           bat        Kolkata Knight Riders
1092               Rajasthan Royals         field             Rajasthan Royals
1093               Rajasthan Royals         field          Sunrisers Hyderabad
1094            Sunrisers Hyderabad           bat        Kolkata Knight Riders

       result  result_margin  target_runs  target_overs super_over method  \
0        runs          140.0        223.0          20.0          N    NaN
1        runs           33.0        241.0          20.0          N    NaN
2     wickets            9.0        130.0          20.0          N    NaN
3     wickets            5.0        166.0          20.0          N    NaN
4     wickets            5.0        111.0          20.0          N    NaN
...       ...            ...          ...           ...        ...    ...
1090  wickets            4.0        215.0          20.0          N    NaN
1091  wickets            8.0        160.0          20.0          N    NaN
1092  wickets            4.0        173.0          20.0          N    NaN
1093     runs           36.0        176.0          20.0          N    NaN
1094  wickets            8.0        114.0          20.0          N    NaN

                    umpire1              umpire2
0                 Asad Rauf           RE Koertzen
1                 MR Benson            SL Shastri
2                 Aleem Dar        GA Pratapkumar
3                  SJ Davis             DJ Harper
4                 BF Bowden           K Hariharan
...                     ...                   ...
1090            Nitin Menon             VK Sharma
1091          AK Chaudhary              R Pandit
1092  KN Ananthapadmanabhan  MV Saidharshan Kumar
1093            Nitin Menon             VK Sharma
1094          J Madanagopal           Nitin Menon

[1095 rows x 20 columns]
```

Cleaning the Dataset

[5]:
```
matches_df['team1'] = matches_df['team1'].replace('Rising Pune Supergiants',␣
↪'CSK')
matches_df['team1'] = matches_df['team1'].replace('Rising Pune Supergiant',␣
↪'CSK')
```

```
matches_df['team2'] = matches_df['team2'].replace('Rising Pune Supergiants',␣
↪'CSK')
matches_df['team2'] = matches_df['team2'].replace('Rising Pune Supergiant',␣
↪'CSK')

matches_df['team1'] = matches_df['team1'].replace('Royal Challengers Bengaluru',␣
↪'RCB')
matches_df['team1'] = matches_df['team1'].replace('Royal Challengers Bangalore',␣
↪'RCB')

matches_df['team2'] = matches_df['team2'].replace('Royal Challengers Bengaluru',␣
↪'RCB')
matches_df['team2'] = matches_df['team2'].replace('Royal Challengers Bangalore',␣
↪'RCB')

matches_df['team1'] = matches_df['team1'].replace('Gujarat Lions', 'RR')
matches_df['team2'] = matches_df['team2'].replace('Gujarat Lions', 'RR')

matches_df['team1'] = matches_df['team1'].replace('Rajasthan Royals', 'RR')
matches_df['team2'] = matches_df['team2'].replace('Rajasthan Royals', 'RR')

matches_df['team1'] = matches_df['team1'].replace('Chennai Super Kings', 'CSK')
matches_df['team2'] = matches_df['team2'].replace('Chennai Super Kings', 'CSK')

matches_df['team1'] = matches_df['team1'].replace('Sunrisers Hyderabad', 'SRH')
matches_df['team2'] = matches_df['team2'].replace('Sunrisers Hyderabad', 'SRH')

matches_df['team1'] = matches_df['team1'].replace('Deccan Chargers', 'SRH')
matches_df['team2'] = matches_df['team2'].replace('Deccan Chargers', 'SRH')

matches_df['team1'] = matches_df['team1'].replace('Delhi Daredevils', 'DD')
matches_df['team2'] = matches_df['team2'].replace('Delhi Daredevils', 'DD')

matches_df['team1'] = matches_df['team1'].replace('Delhi Capitals', 'DD')
matches_df['team2'] = matches_df['team2'].replace('Delhi Capitals', 'DD')

matches_df['team1'] = matches_df['team1'].replace('Kings XI Punjab', 'PBKS')
matches_df['team2'] = matches_df['team2'].replace('Kings XI Punjab', 'PBKS')

matches_df['team1'] = matches_df['team1'].replace('Pune Warriors', 'LSG')
matches_df['team2'] = matches_df['team2'].replace('Pune Warriors', 'LSG')

matches_df['team1'] = matches_df['team1'].replace('Punjab Kings', 'PBKS')
matches_df['team2'] = matches_df['team2'].replace('Punjab Kings', 'PBKS')

matches_df['team1'] = matches_df['team1'].replace('Kochi Tuskers Kerala', 'GT')
```

```
matches_df['team2'] = matches_df['team2'].replace('Kochi Tuskers Kerala', 'GT')

matches_df['team1'] = matches_df['team1'].replace('Gujarat Titans', 'GT')
matches_df['team2'] = matches_df['team2'].replace('Gujarat Titans', 'GT')

matches_df['team1'] = matches_df['team1'].replace('Lucknow Supergiants', 'LSG')
matches_df['team2'] = matches_df['team2'].replace('Lucknow Supergiants', 'LSG')

matches_df['team1'] = matches_df['team1'].replace('Lucknow Super Giants', 'LSG')
matches_df['team2'] = matches_df['team2'].replace('Lucknow Super Giants', 'LSG')

matches_df['team1'] = matches_df['team1'].replace('Kolkata Knight Riders', 'KKR')
matches_df['team2'] = matches_df['team2'].replace('Kolkata Knight Riders', 'KKR')

matches_df['team1'] = matches_df['team1'].replace('Mumbai Indians', 'MI')
matches_df['team2'] = matches_df['team2'].replace('Mumbai Indians', 'MI')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Rising Pune␣
 ↪Supergiants', 'CSK')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Rising Pune␣
 ↪Supergiant', 'CSK')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Royal Challengers␣
 ↪Bengaluru', 'RCB')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Royal Challengers␣
 ↪Bangalore', 'RCB')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Gujarat Lions',␣
 ↪'RR')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Rajasthan␣
 ↪Royals', 'RR')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Chennai Super␣
 ↪Kings', 'CSK')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Sunrisers␣
 ↪Hyderabad', 'SRH')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Deccan Chargers',␣
 ↪'SRH')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Delhi␣
 ↪Daredevils', 'DD')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Delhi Capitals',␣
 ↪'DD')
```

```
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Kings XI Punjab',␣
↪'PBKS')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Pune Warriors',␣
↪'LSG')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Punjab Kings',␣
↪'PBKS')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Kochi Tuskers␣
↪Kerala', 'GT')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Gujarat Titans',␣
↪'GT')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Lucknow Super␣
↪Giants', 'LSG')
matches_df['toss_winner'] = matches_df['toss_winner'].replace('Kolkata Knight␣
↪Riders', 'KKR')

matches_df['toss_winner'] = matches_df['toss_winner'].replace('Mumbai Indians',␣
↪'MI')

matches_df['winner'] = matches_df['winner'].replace('Rising Pune Supergiants',␣
↪'CSK')
matches_df['winner'] = matches_df['winner'].replace('Rising Pune Supergiant',␣
↪'CSK')

matches_df['winner'] = matches_df['winner'].replace('Royal Challengers␣
↪Bengaluru', 'RCB')
matches_df['winner'] = matches_df['winner'].replace('Royal Challengers␣
↪Bangalore', 'RCB')

matches_df['winner'] = matches_df['winner'].replace('Gujarat Lions', 'RR')
matches_df['winner'] = matches_df['winner'].replace('Rajasthan Royals', 'RR')

matches_df['winner'] = matches_df['winner'].replace('Chennai Super Kings', 'CSK')

matches_df['winner'] = matches_df['winner'].replace('Sunrisers Hyderabad', 'SRH')
matches_df['winner'] = matches_df['winner'].replace('Deccan Chargers', 'SRH')

matches_df['winner'] = matches_df['winner'].replace('Delhi Daredevils', 'DD')
matches_df['winner'] = matches_df['winner'].replace('Delhi Capitals', 'DD')

matches_df['winner'] = matches_df['winner'].replace('Kings XI Punjab', 'PBKS')
matches_df['winner'] = matches_df['winner'].replace('Pune Warriors', 'LSG')

matches_df['winner'] = matches_df['winner'].replace('Punjab Kings', 'PBKS')
```

```
matches_df['winner'] = matches_df['winner'].replace('Kochi Tuskers Kerala', 'GT')
matches_df['winner'] = matches_df['winner'].replace('Gujarat Titans', 'GT')

matches_df['winner'] = matches_df['winner'].replace('Lucknow Super Giants',␣
↪'LSG')
matches_df['winner'] = matches_df['winner'].replace('Kolkata Knight Riders',␣
↪'KKR')

matches_df['winner'] = matches_df['winner'].replace('Mumbai Indians', 'MI')

matches_df
```

[5]:

	id	season	city	date	match_type	player_of_match	venue	team1	team2	toss_winner	toss_decision	winner	result	result_margin	target_runs
0	335982	2007/08	Bangalore	2008-04-18	League	BB McCullum	M Chinnaswamy Stadium	RCB	KKR	RCB	field	KKR	runs	140.0	223.0
1	335983	2007/08	Chandigarh	2008-04-19	League	MEK Hussey	Punjab Cricket Association Stadium, Mohali	PBKS	CSK	CSK	bat	CSK	runs	33.0	241.0
2	335984	2007/08	Delhi	2008-04-19	League	MF Maharoof	Feroz Shah Kotla	DD	RR	RR	bat	DD	wickets	9.0	130.0
3	335985	2007/08	Mumbai	2008-04-20	League	MV Boucher	Wankhede Stadium	MI	RCB	MI	bat	RCB	wickets	5.0	166.0
4	335986	2007/08	Kolkata	2008-04-20	League	DJ Hussey	Eden Gardens	KKR	SRH	SRH	bat	KKR	wickets	5.0	111.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1090	1426307	2024	Hyderabad	2024-05-19	League	Abhishek Sharma	Rajiv Gandhi International Stadium, Uppal, Hyd...	PBKS	SRH	PBKS	bat	SRH	wickets	4.0	215.0
1091	1426309	2024	Ahmedabad	2024-05-21	Qualifier 1	MA Starc	Narendra Modi Stadium, Ahmedabad	SRH	KKR	KKR	field	KKR	runs	8.0	160.0
1092	1426310	2024	Ahmedabad	2024-05-22	Eliminator	R Ashwin	Narendra Modi Stadium, Ahmedabad	RCB	RR	RCB	bat	RCB	runs	36.0	173.0
1093	1426311	2024	Chennai	2024-05-24	Qualifier 2	Shahbaz Ahmed	MA Chidambaram Stadium, Chepauk, Chennai	SRH	RR	RR	field	RR	runs	24.0	176.0
1094	1426312	2024	Chennai	2024-05-26	Final	MA Starc	MA Chidambaram Stadium, Chepauk, Chennai	SRH	KKR	KKR	field	KKR	runs	8.0	114.0

```
1091         SRH           bat    KKR  wickets            8.0        160.0
1092          RR         field     RR  wickets            4.0        173.0
1093          RR         field    SRH     runs           36.0        176.0
1094         SRH           bat    KKR  wickets            8.0        114.0

      target_overs super_over method                umpire1  \
0             20.0          N    NaN              Asad Rauf
1             20.0          N    NaN             MR Benson
2             20.0          N    NaN             Aleem Dar
3             20.0          N    NaN              SJ Davis
4             20.0          N    NaN             BF Bowden
...            ...        ...    ...                   ...
1090          20.0          N    NaN           Nitin Menon
1091          20.0          N    NaN          AK Chaudhary
1092          20.0          N    NaN  KN Ananthapadmanabhan
1093          20.0          N    NaN           Nitin Menon
1094          20.0          N    NaN         J Madanagopal

                  umpire2
0              RE Koertzen
1               SL Shastri
2           GA Pratapkumar
3                DJ Harper
4              K Hariharan
...                    ...
1090             VK Sharma
1091              R Pandit
1092  MV Saidharshan Kumar
1093             VK Sharma
1094           Nitin Menon

[1095 rows x 20 columns]
```

[6]:
```python
# Replace team names in batting_team and bowling_team columns
deliveries_df['batting_team'] = deliveries_df['batting_team'].replace({
    'Rising Pune Supergiants': 'CSK',
    'Rising Pune Supergiant': 'CSK',
    'Royal Challengers Bengaluru': 'RCB',
    'Royal Challengers Bangalore': 'RCB',
    'Gujarat Lions': 'RR',
    'Rajasthan Royals': 'RR',
    'Chennai Super Kings': 'CSK',
    'Sunrisers Hyderabad': 'SRH',
    'Deccan Chargers': 'SRH',
    'Delhi Daredevils': 'DD',
    'Delhi Capitals': 'DD',
    'Kings XI Punjab': 'PBKS',
```

```
    'Pune Warriors': 'LSG',
    'Punjab Kings': 'PBKS',
    'Kochi Tuskers Kerala': 'GT',
    'Gujarat Titans': 'GT',
    'Lucknow Supergiants': 'LSG',
    'Kolkata Knight Riders': 'KKR',
    'Mumbai Indians': 'MI',
    'Lucknow Super Giants': 'LSG'
})

deliveries_df['bowling_team'] = deliveries_df['bowling_team'].replace({
    'Rising Pune Supergiants': 'CSK',
    'Rising Pune Supergiant': 'CSK',
    'Royal Challengers Bengaluru': 'RCB',
    'Royal Challengers Bangalore': 'RCB',
    'Gujarat Lions': 'RR',
    'Rajasthan Royals': 'RR',
    'Chennai Super Kings': 'CSK',
    'Sunrisers Hyderabad': 'SRH',
    'Deccan Chargers': 'SRH',
    'Delhi Daredevils': 'DD',
    'Delhi Capitals': 'DD',
    'Kings XI Punjab': 'PBKS',
    'Pune Warriors': 'LSG',
    'Punjab Kings': 'PBKS',
    'Kochi Tuskers Kerala': 'GT',
    'Gujarat Titans': 'GT',
    'Lucknow Supergiants': 'LSG',
    'Lucknow Super Giants': 'LSG',
    'Kolkata Knight Riders': 'KKR',
    'Mumbai Indians': 'MI',
    'Lucknow Super Giants': 'LSG'
})
```

Function to reduce the string variables

[7]:
```
def abbreviate_strings(data):
    """
    Abbreviate strings in categorical columns by taking the first letter of each␣
 ↪word
    if the string has more than one word. Otherwise, keep the string as it is.

    Parameters:
    data (pd.DataFrame or pd.Series): Input data containing strings to abbreviate

    Returns:
    pd.DataFrame or pd.Series: Data with abbreviated strings
```

```
    """
    def abbreviate(value):
        if isinstance(value, str):
            words = value.split()
            if len(words) > 1:
                return ''.join(word[0] for word in words)
            else:
                return value
        return value

    if isinstance(data, pd.Series):
        # If input is a Series, apply the abbreviation logic
        print("Data passed is a Series")
        return data.apply(abbreviate)
    elif isinstance(data, pd.DataFrame):
        # If input is a DataFrame, apply to all columns
        print("Data passed is a DataFrame")
        return data.apply(lambda col: col.apply(abbreviate))
    else:
        return data
```

[8]:
```
deliveries_df.columns
```

[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')
```

Abbreviating the string values of Categorical Columns of Deliveries

[9]:
```
# Select categorical columns of Deliveries DataFrame
categorical_columns = deliveries_df.select_dtypes(include=['object']).columns
categorical_columns = categorical_columns.drop(labels=['batting_team',␣
↪'bowling_team', 'batter', 'bowler','player_dismissed']) # Done so that KKR ->␣
↪K is not changed

# Apply the abbreviate_strings function to the selected columns
deliveries_df[categorical_columns] =␣
↪abbreviate_strings(deliveries_df[categorical_columns])

# Verify the changes
deliveries_df
```

```
Data passed is a DataFrame
```

[9]:

	match_id	inning	batting_team	bowling_team	over	ball	batter	\
0	335982	1	KKR	RCB	0	1	SC Ganguly	

```
1          335982       1          KKR          RCB     0     2  BB McCullum
2          335982       1          KKR          RCB     0     3  BB McCullum
3          335982       1          KKR          RCB     0     4  BB McCullum
4          335982       1          KKR          RCB     0     5  BB McCullum
...           ...     ...          ...          ...   ...   ...          ...
260915    1426312       2          KKR          SRH     9     5      SS Iyer
260916    1426312       2          KKR          SRH     9     6      VR Iyer
260917    1426312       2          KKR          SRH    10     1      VR Iyer
260918    1426312       2          KKR          SRH    10     2      SS Iyer
260919    1426312       2          KKR          SRH    10     3      VR Iyer

                bowler non_striker  batsman_runs  extra_runs  total_runs  \
0             P Kumar          BM             0           1           1
1             P Kumar          SG             0           0           0
2             P Kumar          SG             0           1           1
3             P Kumar          SG             0           0           0
4             P Kumar          SG             0           0           0
...               ...         ...           ...         ...         ...
260915     AK Markram          VI             1           0           1
260916     AK Markram          SI             1           0           1
260917  Shahbaz Ahmed          SI             1           0           1
260918  Shahbaz Ahmed          VI             1           0           1
260919  Shahbaz Ahmed          SI             1           0           1

       extras_type  is_wicket player_dismissed dismissal_kind fielder
0          legbyes          0              NaN            NaN     NaN
1              NaN          0              NaN            NaN     NaN
2            wides          0              NaN            NaN     NaN
3              NaN          0              NaN            NaN     NaN
4              NaN          0              NaN            NaN     NaN
...            ...        ...              ...            ...     ...
260915         NaN          0              NaN            NaN     NaN
260916         NaN          0              NaN            NaN     NaN
260917         NaN          0              NaN            NaN     NaN
260918         NaN          0              NaN            NaN     NaN
260919         NaN          0              NaN            NaN     NaN

[260920 rows x 17 columns]
```

Abbreviating the string values of Categorical Columns of Matches

[10]:
```
# Select categorical columns
categorical_columns = matches_df.select_dtypes(include=['object']).columns
categorical_columns = categorical_columns.drop(labels=['team1', 'team2',␣
↪'toss_winner', 'winner','player_of_match'])

# Apply the abbreviate_strings function to the selected columns
```

```
matches_df[categorical_columns] = ␣
 ↪abbreviate_strings(matches_df[categorical_columns])

# Verify the changes
matches_df
```

```
Data passed is a DataFrame
```

[10]:

	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
0	335982	2007/08	Bangalore	2008-04-18	League	BB McCullum	MCS	RCB	KKR	RCB	field	KKR	runs	140.0	223.0	20.0	N	NaN	AR
1	335983	2007/08	Chandigarh	2008-04-19	League	MEK Hussey	PCASM	PBKS	CSK	CSK	bat	CSK	runs	33.0	241.0	20.0	N	NaN	MB
2	335984	2007/08	Delhi	2008-04-19	League	MF Maharoof	FSK	DD	RR	RR	bat	DD	wickets	9.0	130.0	20.0	N	NaN	AD
3	335985	2007/08	Mumbai	2008-04-20	League	MV Boucher	WS	MI	RCB	MI	bat	RCB	wickets	5.0	166.0	20.0	N	NaN	SD
4	335986	2007/08	Kolkata	2008-04-20	League	DJ Hussey	EG	KKR	SRH	SRH	bat	KKR	wickets	5.0	111.0	20.0	N	NaN	BB
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1090	1426307	2024	Hyderabad	2024-05-19	League	Abhishek Sharma	RGISUH	PBKS	SRH	PBKS	bat	SRH	wickets	4.0	215.0	20.0	N	NaN	NM
1091	1426309	2024	Ahmedabad	2024-05-21	Q1	MA Starc	NMSA	SRH	KKR	SRH	bat	KKR	wickets	8.0	160.0	20.0	N	NaN	AC
1092	1426310	2024	Ahmedabad	2024-05-22	Eliminator	R Ashwin	NMSA	RCB	RR	RR	field	RR	wickets	4.0	173.0	20.0	N	NaN	KA
1093	1426311	2024	Chennai	2024-05-24	Q2	Shahbaz Ahmed	MCSCC	SRH	RR	RR	field	SRH	runs	36.0	176.0	20.0	N	NaN	NM
1094	1426312	2024	Chennai	2024-05-26	Final	MA Starc	MCSCC	SRH	KKR	SRH	bat	KKR	wickets	8.0	114.0	20.0	N	NaN	JM

```
     umpire2
0         RK
1         SS
2         GP
3         DH
4         KH
...      ...
1090      VS
1091      RP
1092     MSK
1093      VS
1094      NM

[1095 rows x 20 columns]
```

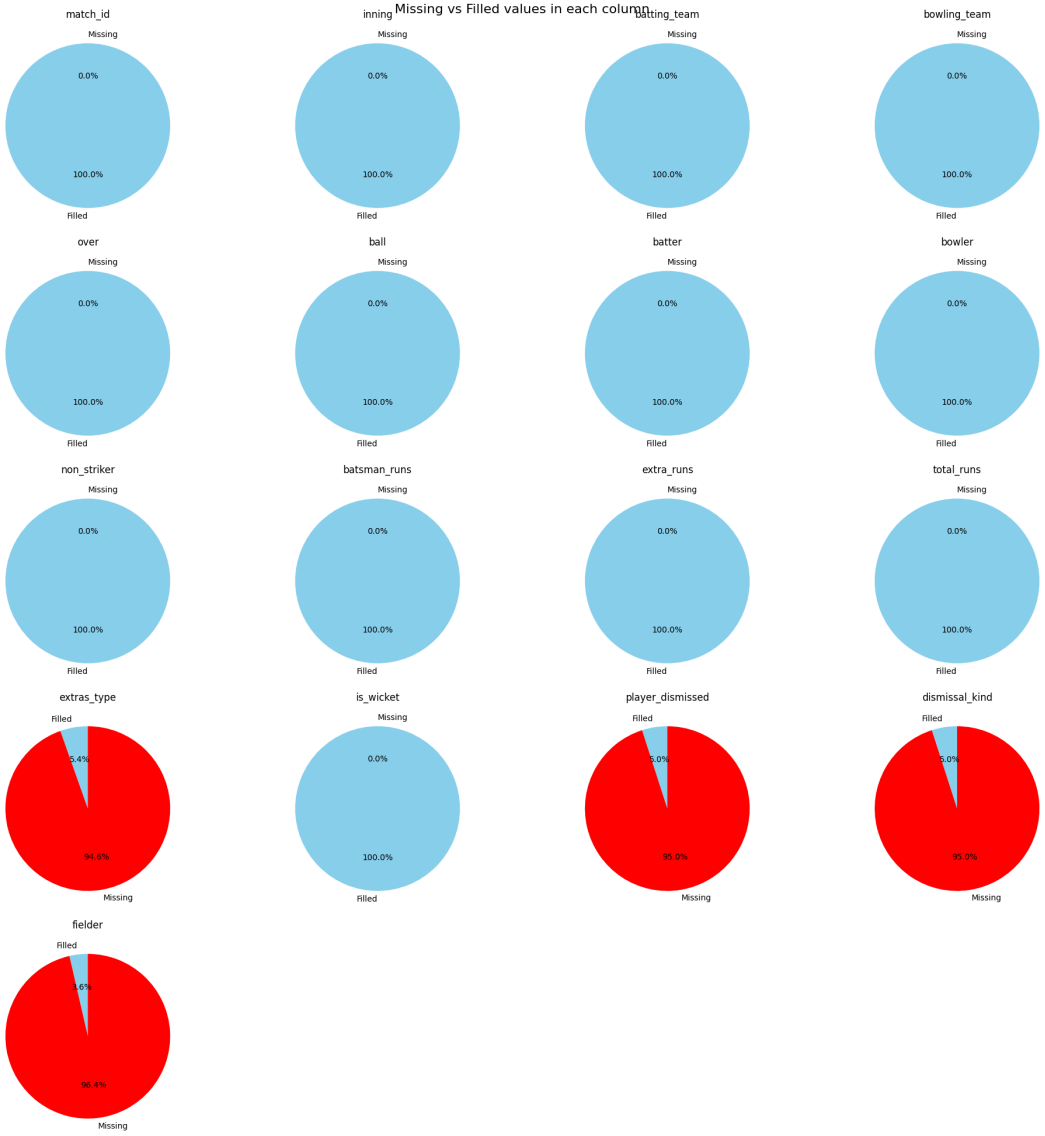
2 *Feature Extraction*

Function to See the amount of Sparcity in the Dataset

[11]:
```
def plot_missing_vs_filled(dataset, nrows=5, ncols=4, figsize=(20, 20),␣
↪title="Missing vs Filled values in each column\n"):
    # Create subplots for each column
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=figsize)
    axes = axes.flatten()

    # Iterate through each column and plot pie charts
    for i, col in enumerate(dataset.columns):
        total = dataset[col].size
        missing = dataset[col].isna().sum()
        filled = total - missing
        slices = [filled, missing]
        labels = ['Filled', 'Missing']
        colors = ['skyblue', 'red']  # Replace 'cobalt' with 'blue' or any valid␣
↪color name

        axes[i].pie(slices, labels=labels, colors=colors, autopct='%1.1f%%',␣
↪startangle=90)
        axes[i].set_title(col)

    # Hide any empty subplots if they exist
    for j in range(i + 1, len(axes)):
        axes[j].axis('off')

    plt.suptitle(title, fontsize=16)
    plt.tight_layout()
    plt.show()
```

Plotting Deliveries Dataset Sparcity : This plot is to visualize the amount of missing information in each Column of Deliveries dataset

[12]:
```
plot_missing_vs_filled(deliveries_df, nrows=5, ncols=4, figsize=(20, 20),␣
↪title="Missing vs Filled values in each column")
```



Plotting Matches Dataset Sparcity : This plot is to visualize the amount of missing information in each Column of Matches dataset

[13]:
```
plot_missing_vs_filled(matches_df, nrows=5, ncols=4, figsize=(20, 20),␣
↪title="Missing vs Filled values in each column")
```

Pearson and Spearman Correlation Functions: To Calculate the degree of Correlation between the Features

[14]:
```
def calculate_correlations(dataset):
    """
    Calculate Spearman Correlation for categorical values and Pearson␣
↪Correlation for numerical values.

    Parameters:
    dataset (pd.DataFrame): The dataset to calculate correlations for.
```

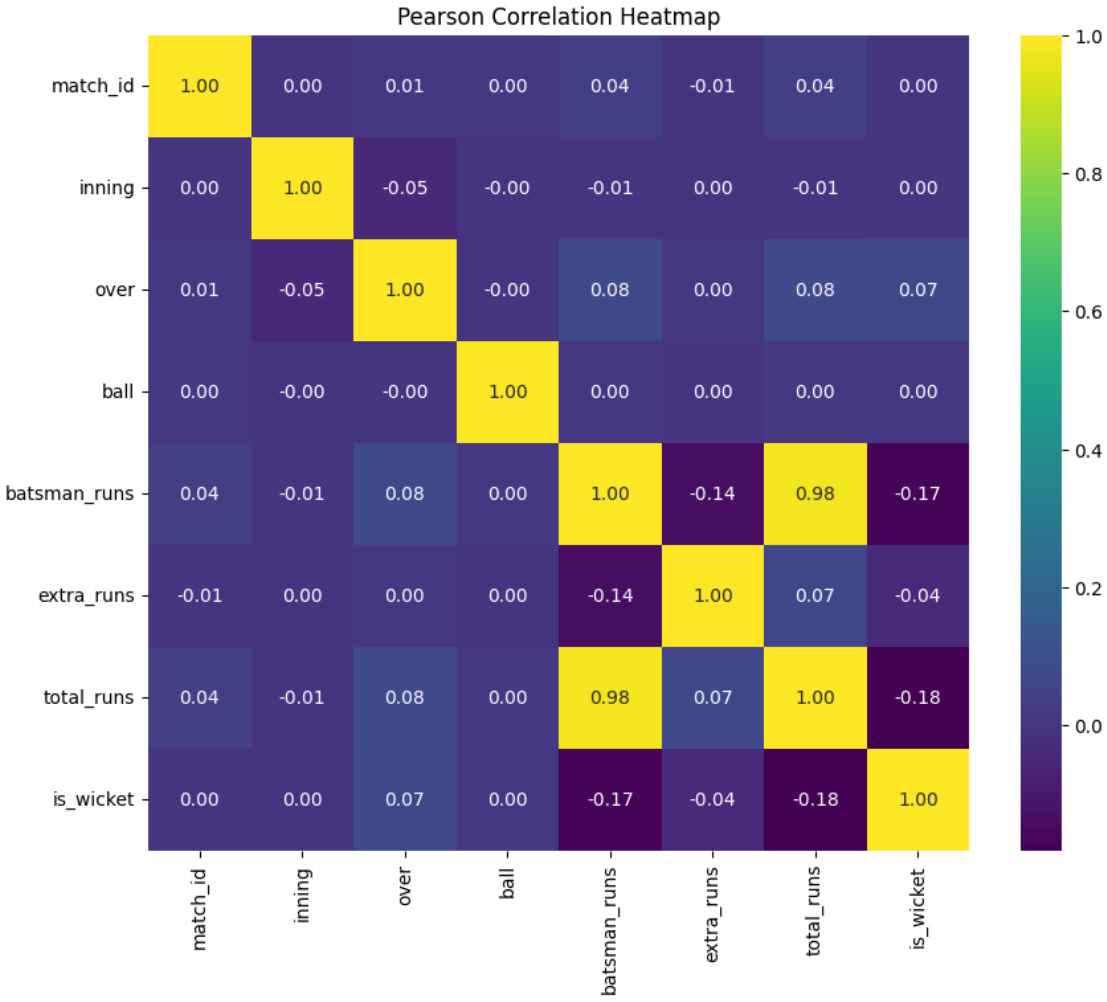
```
    Returns:
    dict: A dictionary containing Spearman and Pearson correlation matrices.
    """
    # Separate numerical and categorical columns
    numerical_cols = dataset.select_dtypes(include=['float64', 'int64']).columns
    categorical_cols = dataset.select_dtypes(include=['object']).columns

    # Calculate Pearson Correlation for numerical columns
    pearson_corr = dataset[numerical_cols].corr(method='pearson') if not␣
↪numerical_cols.empty else None
    print(f"The numerical columns are: {numerical_cols}")

    # Encode categorical columns into numeric values for Spearman Correlation

    if not categorical_cols.empty:
        # Convert categorical columns to ranks using their category codes
        encoded_dataset = dataset[categorical_cols].apply(lambda col: col.
↪astype('category').cat.codes)
        ranked_dataset = encoded_dataset.rank()  # Rank the encoded values
        spearman_corr = ranked_dataset.corr(method='spearman')  # Calculate␣
↪Spearman correlation on ranks
    else:
        spearman_corr = None

    print(f"The categorical columns are: {categorical_cols}")

    return {
        'Pearson Correlation': pearson_corr,
        'Spearman Correlation': spearman_corr
    }
```

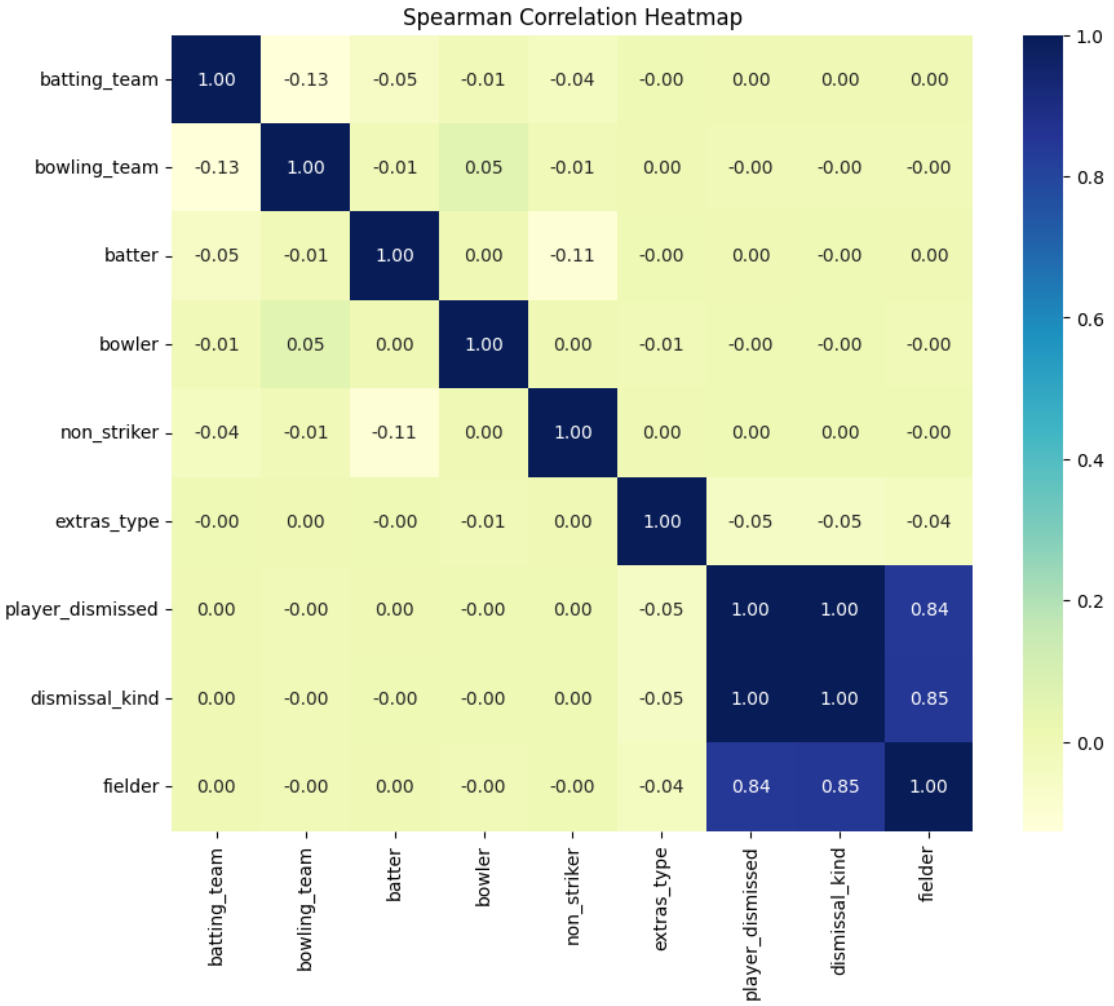
Deliveries Dataset Correlation Heatmaps

[15]:
```
# Calculate correlations
correlations = calculate_correlations(deliveries_df)

# Plot heatmaps for correlations
if correlations['Pearson Correlation'] is not None:
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlations['Pearson Correlation'], annot=True, cmap='viridis',␣
↪fmt='.2f')
    plt.title('Pearson Correlation Heatmap')
    plt.show()

if correlations['Spearman Correlation'] is not None:
    plt.figure(figsize=(10, 8))
```

```
    sns.heatmap(correlations['Spearman Correlation'], annot=True, cmap='YlGnBu',␣
↪fmt='.2f')
    plt.title('Spearman Correlation Heatmap')
    plt.show()
```

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



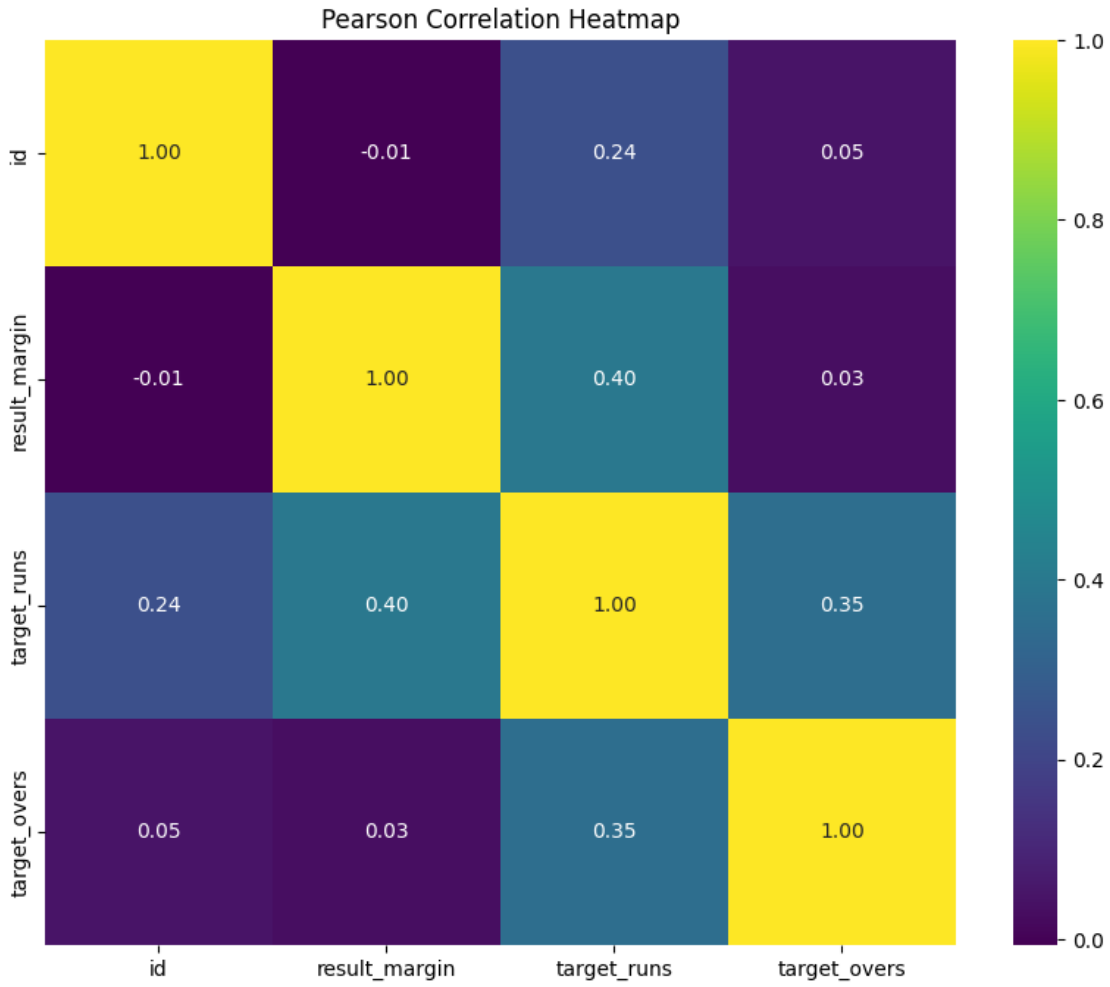


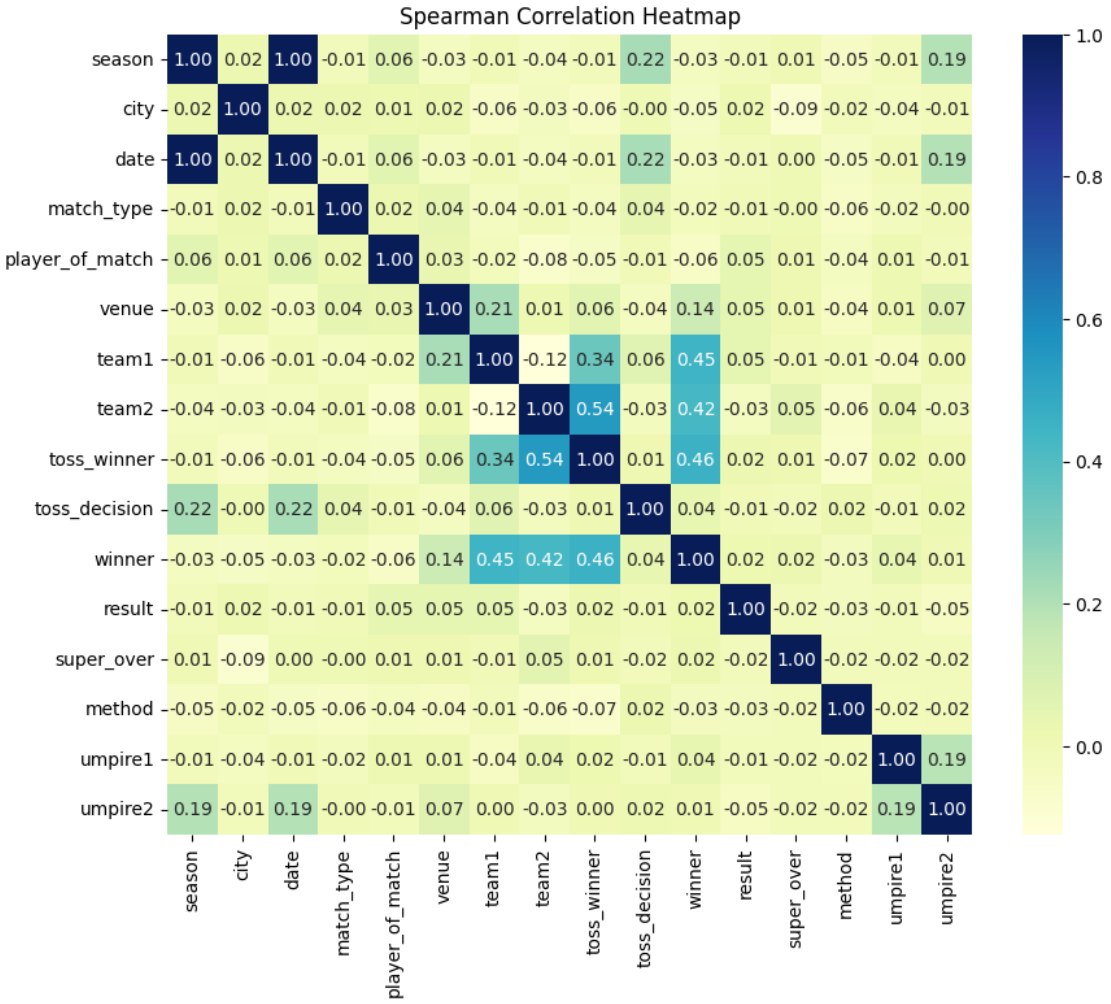
Matches Dataset Correlation HeatMaps

[16]:
```
# Calculate correlations
correlations = calculate_correlations(matches_df)

# Plot heatmaps for correlations
if correlations['Pearson Correlation'] is not None:
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlations['Pearson Correlation'], annot=True, cmap='viridis',␣
 ↪fmt='.2f')
    plt.title('Pearson Correlation Heatmap')
    plt.show()

if correlations['Spearman Correlation'] is not None:
    plt.figure(figsize=(10, 8))
```

```
    sns.heatmap(correlations['Spearman Correlation'], annot=True, cmap='YlGnBu',␣
↪fmt='.2f')
    plt.title('Spearman Correlation Heatmap')
    plt.show()
```

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





Further cleaning the Matches dataset

[17]:
```
matches_df['super_over'] = matches_df['super_over'].map({'Y': 1, 'N': 0})

# # Identify columns with more than 95% similar values
# threshold = 0.95
# columns_to_drop = [col for col in matches_df.columns if matches_df[col].
 ↪value_counts(normalize=True).iloc[0] > threshold]

# print(f"Columns with more than 95% similar values: {columns_to_drop}")

# # Drop the identified columns
# matches_df = matches_df.drop(columns=columns_to_drop)
```

3 *Filling the Numerical Values in empty slots*

Checking the number of missing values per Column

[18]:
```
for col in deliveries_df.columns:
    nan_count = deliveries_df[col].isna().sum()
    print(f"Column '{col}' has {nan_count} NaN or missing values")
```

```
Column 'match_id' has 0 NaN or missing values
Column 'inning' has 0 NaN or missing values
Column 'batting_team' has 0 NaN or missing values
Column 'bowling_team' has 0 NaN or missing values
Column 'over' has 0 NaN or missing values
Column 'ball' has 0 NaN or missing values
Column 'batter' has 0 NaN or missing values
Column 'bowler' has 0 NaN or missing values
Column 'non_striker' has 0 NaN or missing values
Column 'batsman_runs' has 0 NaN or missing values
Column 'extra_runs' has 0 NaN or missing values
Column 'total_runs' has 0 NaN or missing values
Column 'extras_type' has 246795 NaN or missing values
Column 'is_wicket' has 0 NaN or missing values
Column 'player_dismissed' has 247970 NaN or missing values
Column 'dismissal_kind' has 247970 NaN or missing values
Column 'fielder' has 251566 NaN or missing values
```

[19]:
```
# Fill missing values with the mean of the respective columns
for cols in deliveries_df.columns:
    if type(cols) == np.float64:
        for entry in deliveries_df[cols]:
            if entry.isnull():
                deliveries_df[entry][cols] = np.mean(deliveries_df[cols])
```

[20]:
```
columns_to_process = ['extras_type', 'player_dismissed', 'fielder',␣
 ↪'dismissal_kind']

for col in columns_to_process:
    if col == 'extras_type':
        deliveries_df[col] = deliveries_df[col].fillna('No Extra')
    elif col in ['player_dismissed', 'dismissal_kind']:
        deliveries_df[col] = deliveries_df[col].fillna('No Dismissal')
    elif col == 'fielder':
        deliveries_df[col] = deliveries_df[col].fillna('No Fielder')
```

[21]:
```
for col in deliveries_df.columns:
    if deliveries_df[col].isnull().any():
        print(f"The `{col}` column still has missing values")
else:
```

```
    print("All ok")
```

```
All ok
```

4 *Dataset Statistics*

[22]:
```
print("The Stats of the Matches Dataframe are: ")
matches_df.describe()
```

```
The Stats of the Matches Dataframe are:
```

[22]:

	id	result_margin	target_runs	target_overs	super_over
count	1.095000e+03	1076.000000	1092.000000	1092.000000	1095.000000
mean	9.048283e+05	17.259294	165.684066	19.759341	0.012785
std	3.677402e+05	21.787444	33.427048	1.581108	0.112399
min	3.359820e+05	1.000000	43.000000	5.000000	0.000000
25%	5.483315e+05	6.000000	146.000000	20.000000	0.000000
50%	9.809610e+05	8.000000	166.000000	20.000000	0.000000
75%	1.254062e+06	20.000000	187.000000	20.000000	0.000000
max	1.426312e+06	146.000000	288.000000	20.000000	1.000000

[23]:
```
print(" The Stats of the Deliveries Dataframe are: ")
deliveries_df.describe()
```

```
 The Stats of the Deliveries Dataframe are:
```

[23]:

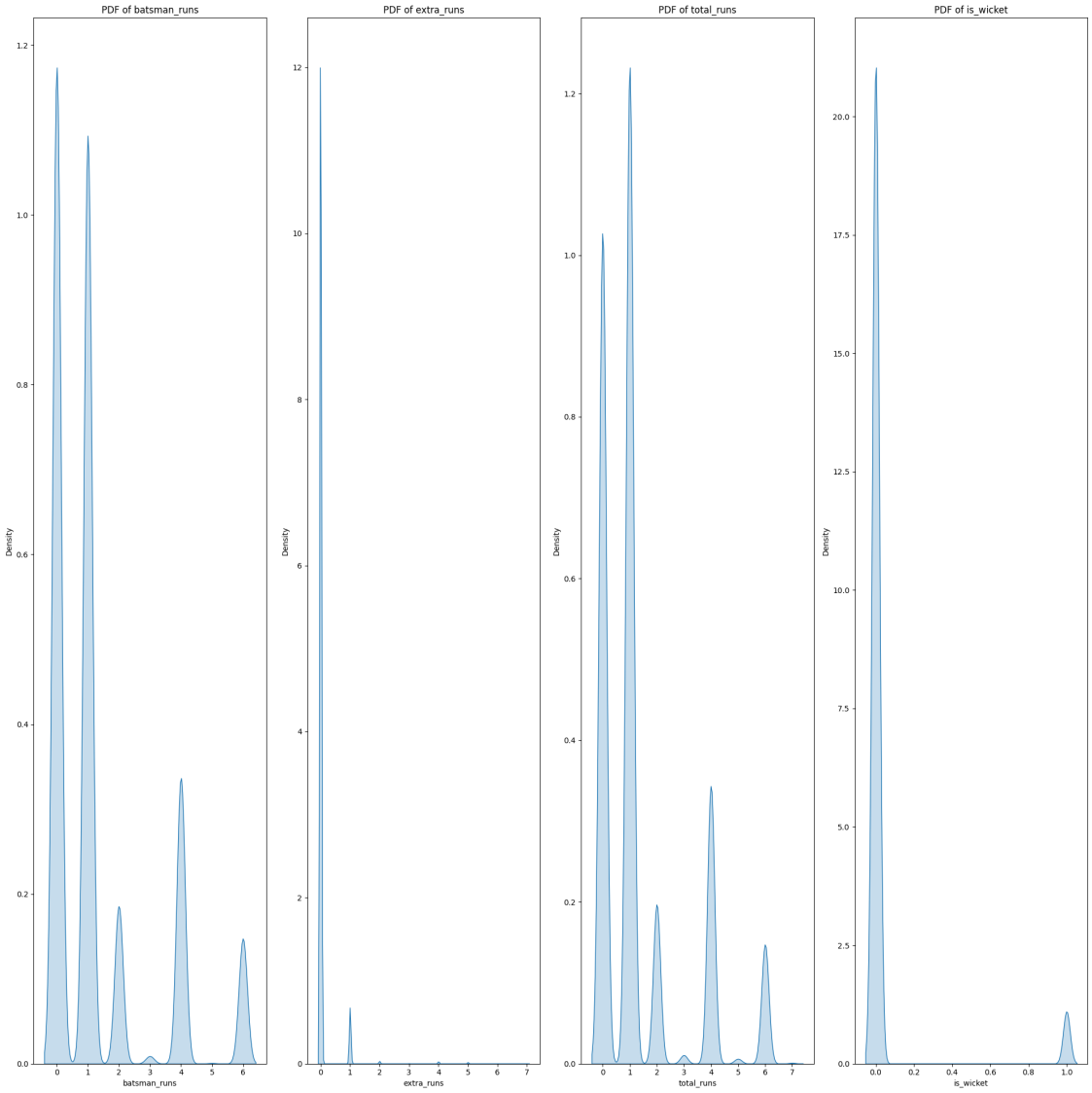
	match_id	inning	over	ball	\
count	2.609200e+05	260920.000000	260920.000000	260920.000000	
mean	9.070665e+05	1.483531	9.197677	3.624486	
std	3.679913e+05	0.502643	5.683484	1.814920	
min	3.359820e+05	1.000000	0.000000	1.000000	
25%	5.483340e+05	1.000000	4.000000	2.000000	
50%	9.809670e+05	1.000000	9.000000	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

Function to plot the Probability Density function of the Dataset

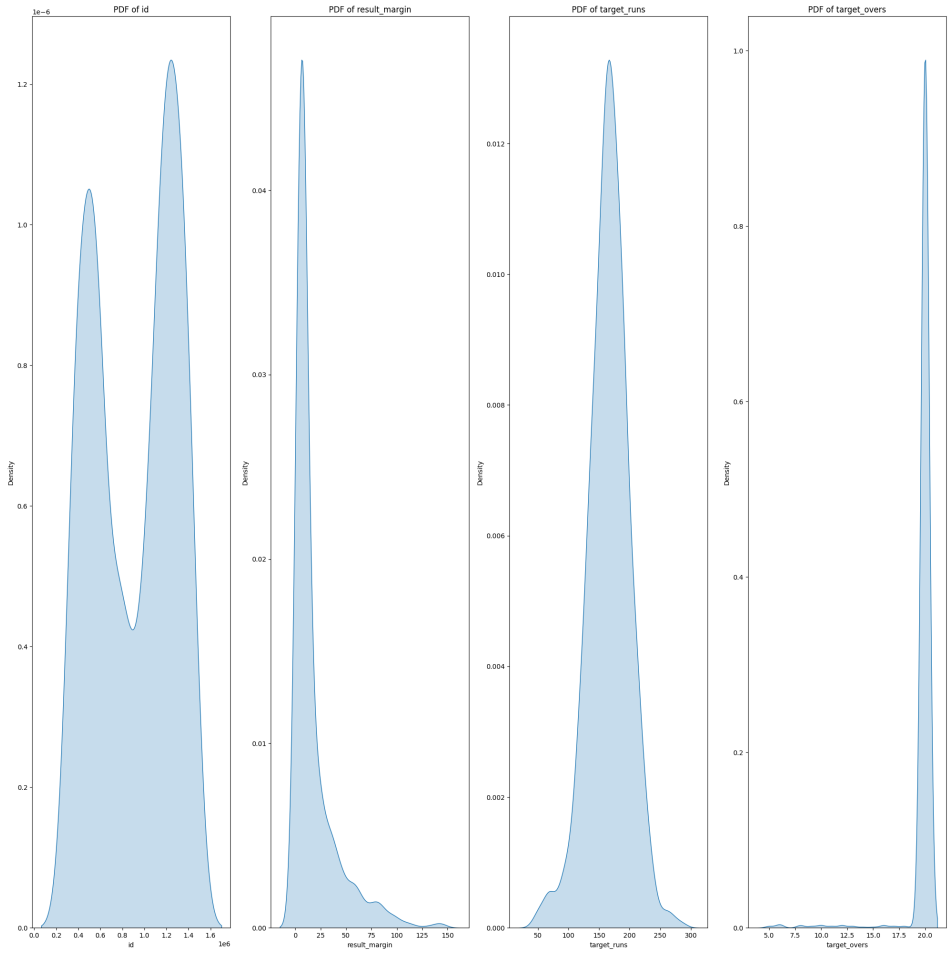
[24]:
```
def plot_pdf_of_dataset(dataset, figsize=(20, 10), cols_per_row=3):
    """
    Plots Probability Density Functions (PDFs) for all numerical columns in the␣
↪dataset.

    Parameters:
    dataset (pd.DataFrame): The dataset containing numerical columns.
    figsize (tuple): The size of the figure (width, height).
    cols_per_row (int): Number of plots per row.

    Returns:
    None
    """
    # Select numerical columns for plotting PDFs
    numerical_cols = dataset.select_dtypes(include=['float64', 'int64']).columns

    # Calculate the number of rows and columns for subplots
    n_cols = len(numerical_cols)
    n_rows = (n_cols + cols_per_row - 1) // cols_per_row  # Arrange plots in␣
↪specified columns per row

    # Set up the figure
    fig, axes = plt.subplots(n_rows, cols_per_row, figsize=(figsize[0],␣
↪figsize[1] * n_rows))
    axes = axes.flatten()

    # Plot PDF for each numerical column
    for idx, col in enumerate(numerical_cols):
        sns.kdeplot(data=dataset[col].dropna(), ax=axes[idx], fill=True)
        axes[idx].set_title(f'PDF of {col}')
        axes[idx].set_xlabel(col)
        axes[idx].set_ylabel('Density')

    # Remove empty subplots if any
    for idx in range(len(numerical_cols), len(axes)):
        fig.delaxes(axes[idx])

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

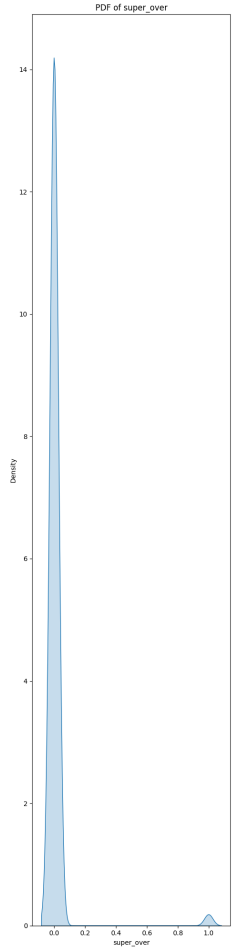
[25]:
```
plot_pdf_of_dataset(deliveries_df.drop(labels=['match_id', 'inning', 'over',␣
↪'ball'], axis=1), figsize=(20, 20), cols_per_row=4)
```



Stats of Matches Dataset

[26]:
```
plot_pdf_of_dataset(matches_df, figsize=(20, 20), cols_per_row=4)
```


PDF of id
1e−6
Density
id
1e6
PDF of result_margin
Density
result_margin
PDF of target_runs
Density
target_runs
PDF of target_overs
Density
target_overs


PDF of super_over
Density
super_over

[27]:
```
batting_teams = deliveries_df['batting_team'].unique()
team_data = {}
total_entries = 0

for team in batting_teams:
    team_data[f"{team}"] = deliveries_df[deliveries_df['batting_team'] == team]
    print(f"Data for {team}:")
    print(f"Number of entries: {len(team_data[f'{team}'])}")
    total_entries += len(team_data[f'{team}'])
    print("="*20)

print(f"Total entries in the dataset: {total_entries}")
```

```
Data for KKR:
Number of entries: 29514
====================
Data for RCB:
Number of entries: 30023
====================
Data for CSK:
Number of entries: 32131
====================
Data for PBKS:
Number of entries: 29479
====================
Data for RR:
Number of entries: 29808
====================
Data for DD:
Number of entries: 29732
====================
Data for MI:
Number of entries: 31437
====================
Data for SRH:
Number of entries: 30877
====================
Data for GT:
Number of entries: 7076
====================
Data for LSG:
Number of entries: 10843
====================
Total entries in the dataset: 260920
```

5 *Korbo Lorbo Jeetbo Re. . . .*

[28]:
```
team_data['KKR']
```

[28]:
```
        match_id  inning batting_team bowling_team  over  ball       batter  \
0         335982       1          KKR          RCB     0     1   SC Ganguly
1         335982       1          KKR          RCB     0     2  BB McCullum
2         335982       1          KKR          RCB     0     3  BB McCullum
3         335982       1          KKR          RCB     0     4  BB McCullum
4         335982       1          KKR          RCB     0     5  BB McCullum
...          ...     ...          ...          ...   ...   ...          ...
260915   1426312       2          KKR          SRH     9     5      SS Iyer
260916   1426312       2          KKR          SRH     9     6      VR Iyer
260917   1426312       2          KKR          SRH    10     1      VR Iyer
260918   1426312       2          KKR          SRH    10     2      SS Iyer
260919   1426312       2          KKR          SRH    10     3      VR Iyer

                bowler non_striker  batsman_runs  extra_runs  total_runs  \
0             P Kumar          BM             0           1           1
1             P Kumar          SG             0           0           0
2             P Kumar          SG             0           1           1
3             P Kumar          SG             0           0           0
4             P Kumar          SG             0           0           0
...               ...         ...           ...         ...         ...
260915     AK Markram          VI             1           0           1
260916     AK Markram          SI             1           0           1
260917  Shahbaz Ahmed          SI             1           0           1
260918  Shahbaz Ahmed          VI             1           0           1
260919  Shahbaz Ahmed          SI             1           0           1

       extras_type  is_wicket player_dismissed dismissal_kind     fielder
0          legbyes          0     No Dismissal   No Dismissal  No Fielder
1         No Extra          0     No Dismissal   No Dismissal  No Fielder
2            wides          0     No Dismissal   No Dismissal  No Fielder
3         No Extra          0     No Dismissal   No Dismissal  No Fielder
4         No Extra          0     No Dismissal   No Dismissal  No Fielder
...            ...        ...              ...            ...         ...
260915    No Extra          0     No Dismissal   No Dismissal  No Fielder
260916    No Extra          0     No Dismissal   No Dismissal  No Fielder
260917    No Extra          0     No Dismissal   No Dismissal  No Fielder
260918    No Extra          0     No Dismissal   No Dismissal  No Fielder
260919    No Extra          0     No Dismissal   No Dismissal  No Fielder

[29514 rows x 17 columns]
```

[29]:
```
# Select categorical columns
categorical_columns = matches_df.select_dtypes(include=['object']).columns
```

```
categorical_columns = categorical_columns.drop(labels=['team1', 'team2',␣
↪'toss_winner', 'winner'])

# Apply the abbreviate_strings function to the selected columns
matches_df[categorical_columns] =␣
↪abbreviate_strings(matches_df[categorical_columns])

# Verify the changes
matches_df
```

```
Data passed is a DataFrame
```

[29]:

	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
0	335982	2007/08	Bangalore	2008-04-18	League	BM	MCS	RCB	KKR	RCB	field	KKR	runs	140.0	223.0	20.0	0	NaN	AR
1	335983	2007/08	Chandigarh	2008-04-19	League	MH	PCASM	PBKS	CSK	CSK	bat	CSK	runs	33.0	241.0	20.0	0	NaN	MB
2	335984	2007/08	Delhi	2008-04-19	League	MM	FSK	DD	RR	RR	bat	DD	wickets	9.0	130.0	20.0	0	NaN	AD
3	335985	2007/08	Mumbai	2008-04-20	League	MB	WS	MI	RCB	MI	bat	RCB	wickets	5.0	166.0	20.0	0	NaN	SD
4	335986	2007/08	Kolkata	2008-04-20	League	DH	EG	KKR	SRH	SRH	bat	KKR	wickets	5.0	111.0	20.0	0	NaN	BB
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1090	1426307	2024	Hyderabad	2024-05-19	League	AS	RGISUH	PBKS	SRH	PBKS	bat	SRH	wickets	4.0	215.0	20.0	0	NaN	NM
1091	1426309	2024	Ahmedabad	2024-05-21	Q1	MS	NMSA	SRH	KKR	SRH	bat	KKR	wickets	8.0	160.0	20.0	0	NaN	AC
1092	1426310	2024	Ahmedabad	2024-05-22	Eliminator	RA	NMSA	RCB	RR	RR	field	RR	wickets						
1093	1426311	2024	Chennai	2024-05-24	Q2	SA	MCSCC	SRH	RR	RR	field	SRH	runs						
1094	1426312	2024	Chennai	2024-05-26	Final	MS	MCSCC	SRH	KKR	SRH	bat	KKR	wickets						

```
1092            4.0        173.0          20.0           0    NaN      KA
1093           36.0        176.0          20.0           0    NaN      NM
1094            8.0        114.0          20.0           0    NaN      JM

     umpire2
0         RK
1         SS
2         GP
3         DH
4         KH
...      ...
1090      VS
1091      RP
1092     MSK
1093      VS
1094      NM

[1095 rows x 20 columns]
```

6 *Team Performance Analysis*

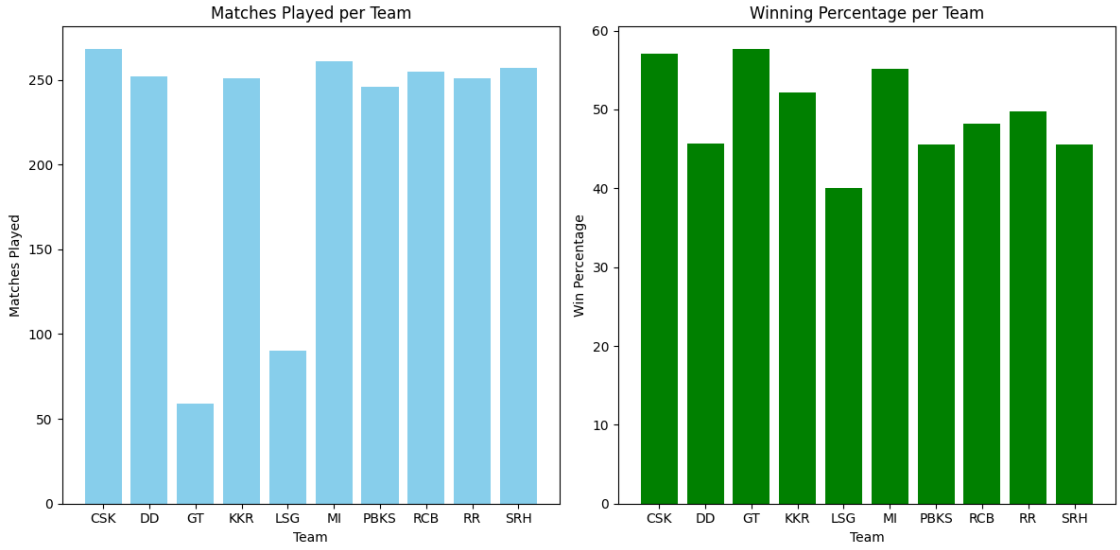
7 Matches Played and Winning Percentages

[30]:
```
matches_df.head()
```

[30]:
```
        id   season        city        date match_type player_of_match  venue  \
0  335982  2007/08   Bangalore  2008-04-18     League              BM    MCS
1  335983  2007/08  Chandigarh  2008-04-19     League              MH  PCASM
2  335984  2007/08       Delhi  2008-04-19     League              MM    FSK
3  335985  2007/08      Mumbai  2008-04-20     League              MB     WS
4  335986  2007/08     Kolkata  2008-04-20     League              DH     EG

  team1 team2 toss_winner toss_decision winner   result  result_margin  \
0   RCB   KKR         RCB         field    KKR     runs          140.0
1  PBKS   CSK         CSK           bat    CSK     runs           33.0
2    DD    RR          RR           bat     DD  wickets            9.0
3    MI   RCB          MI           bat    RCB  wickets            5.0
4   KKR   SRH         SRH           bat    KKR  wickets            5.0

   target_runs  target_overs  super_over method umpire1 umpire2
0        223.0          20.0           0    NaN      AR      RK
1        241.0          20.0           0    NaN      MB      SS
2        130.0          20.0           0    NaN      AD      GP
3        166.0          20.0           0    NaN      SD      DH
4        111.0          20.0           0    NaN      BB      KH
```

[31]:
```
# Get unique teams from matches_df
teams = np.union1d(matches_df['team1'], matches_df['team2'])
# Initialize dictionaries
matches_played = {}
wins = {}

# Calculate matches played (appearing as team1 or team2) and wins (winner column)
for team in teams:
    played = matches_df[(matches_df['team1'] == team) | (matches_df['team2'] ==␣
 ↪team)].shape[0]

    win = matches_df[matches_df['winner'] == team].shape[0]
    print(team,played,win)
    matches_played[team] = played
    wins[team] = win

# Calculate win percentage for each team
win_percentage = {team: (wins[team] / matches_played[team]) * 100 for team in␣
 ↪teams}

# Plot the results
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.bar(matches_played.keys(), matches_played.values(), color='skyblue')
plt.xlabel('Team')
plt.ylabel('Matches Played')
plt.title('Matches Played per Team')

plt.subplot(1, 2, 2)
plt.bar(win_percentage.keys(), win_percentage.values(), color='green')
plt.xlabel('Team')
plt.ylabel('Win Percentage')
plt.title('Winning Percentage per Team')

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

```
CSK 268 153
DD 252 115
GT 59 34
KKR 251 131
LSG 90 36
MI 261 144
PBKS 246 112
RCB 255 123
RR 251 125
SRH 257 117
```



Here we must mention the nature of plots of matches played by each team is uniform because there are teams that were created later than others

8 *Economy and Run Rates*

[32]:
```
deliveries_df
```

[32]:

	match_id	inning	batting_team	bowling_team	over	ball	batter	bowler	non_striker	batsman_runs	extra_runs	total_runs
0	335982	1	KKR	RCB	0	1	SC Ganguly	P Kumar	BM	0	1	1
1	335982	1	KKR	RCB	0	2	BB McCullum	P Kumar	SG	0	0	0
2	335982	1	KKR	RCB	0	3	BB McCullum	P Kumar	SG	0	1	1
3	335982	1	KKR	RCB	0	4	BB McCullum	P Kumar	SG	0	0	0
4	335982	1	KKR	RCB	0	5	BB McCullum	P Kumar	SG	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
260915	1426312	2	KKR	SRH	9	5	SS Iyer	AK Markram	VI	1	0	1
260916	1426312	2	KKR	SRH	9	6	VR Iyer					
260917	1426312	2	KKR	SRH	10	1	VR Iyer					
260918	1426312	2	KKR	SRH	10	2	SS Iyer					
260919	1426312	2	KKR	SRH	10	3	VR Iyer					

```
260916     AK Markram          SI             1           0           1
260917  Shahbaz Ahmed          SI             1           0           1
260918  Shahbaz Ahmed          VI             1           0           1
260919  Shahbaz Ahmed          SI             1           0           1

       extras_type  is_wicket player_dismissed dismissal_kind     fielder
0          legbyes          0     No Dismissal   No Dismissal  No Fielder
1         No Extra          0     No Dismissal   No Dismissal  No Fielder
2            wides          0     No Dismissal   No Dismissal  No Fielder
3         No Extra          0     No Dismissal   No Dismissal  No Fielder
4         No Extra          0     No Dismissal   No Dismissal  No Fielder
...            ...        ...              ...            ...         ...
260915    No Extra          0     No Dismissal   No Dismissal  No Fielder
260916    No Extra          0     No Dismissal   No Dismissal  No Fielder
260917    No Extra          0     No Dismissal   No Dismissal  No Fielder
260918    No Extra          0     No Dismissal   No Dismissal  No Fielder
260919    No Extra          0     No Dismissal   No Dismissal  No Fielder

[260920 rows x 17 columns]
```

[33]:
```python
import plotly.express as px
import plotly.graph_objects as go

def calculate_and_plot_interactive_rates(deliveries_df, matches_df):
    """
    Calculate run rate and economy rate from deliveries DataFrame and plot the␣
 ↪results interactively using Plotly.

    Parameters:
    deliveries_df (pd.DataFrame): DataFrame containing ball-by-ball delivery␣
 ↪details.
    matches_df (pd.DataFrame): DataFrame containing match details.
    """

    # Sort by match_id to process deliveries correctly
    deliveries_df = deliveries_df.sort_values(by=['match_id', 'over', 'ball'])

    # Unique bowlers and teams
    teams = deliveries_df['bowling_team'].unique()

    # Data structures to hold results
    run_rate_overwise = {}
    bowler_economy_rate = {}

    total_overs = 20  # Set to 20 overs per match

    for team in teams:
```

```
        games_played = deliveries_df[deliveries_df['bowling_team'] == team]

        for current_over in range(1, total_overs + 1):
            runs_in_over = games_played[games_played['over'] ==␣
↪current_over]['total_runs'].sum()

            if team not in run_rate_overwise:
                run_rate_overwise[team] = {}
            run_rate_overwise[team][current_over] = runs_in_over  # Runs per␣
↪inning

        # Calculate economy rate per bowler
        team_bowlers = games_played['bowler'].unique()
        for bowler in team_bowlers:
            bowler_deliveries = games_played[games_played['bowler'] == bowler]
            total_runs_conceded = bowler_deliveries['total_runs'].sum()

            # Get total valid balls (ignore wides/no-balls that are not rebowled)
            valid_balls = bowler_deliveries.shape[0]
            total_overs_bowled = valid_balls / 6  # Convert balls to overs

            if total_overs_bowled > 0:
                bowler_economy_rate[bowler] = total_runs_conceded /␣
↪total_overs_bowled

    # Plot the over-wise run rate for each team
    fig = go.Figure()
    for team, overs in run_rate_overwise.items():
        fig.add_trace(go.Scatter(x=list(overs.keys()), y=list(overs.values()),␣
↪mode='lines', name=team))

    fig.update_layout(title='Over-wise Team Run Rate',
                      xaxis_title='Over',
                      yaxis_title='Runs')
    fig.show()

    # Plot the economy rates for bowlers
    bowler_names = list(bowler_economy_rate.keys())
    economy_rates = list(bowler_economy_rate.values())

    fig = px.bar(x=bowler_names, y=economy_rates, labels={'x': 'Bowler', 'y':␣
↪'Economy Rate'}, title="Bowler's Economy Rate in the Tournament")
    fig.update_xaxes(tickangle=45)  # Tilt labels at 45 degrees for visibility
    fig.show()

# Call the function
calculate_and_plot_interactive_rates(deliveries_df, matches_df)
```

9 *Highest and Lowest Scores of Each Team*

[34]:
```
matches_df.loc[matches_df['result'] == 'no result', 'result_margin'] = 'no␣
↪result'

# For ties: if result_margin is missing (NaN), then set it to 0
matches_df.loc[(matches_df['result'] == 'tie') & (matches_df['result_margin'].
↪isna()), 'result_margin'] = 0

# Check and print the number of missing values in result_margin
missing_count = matches_df['result_margin'].isna().sum()
print("Number of missing result_margin entries:", missing_count)
```

```
C:\Users\SOHAM\AppData\Local\Temp\ipykernel_1684\2546502323.py:1: FutureWarning:

Setting an item of incompatible dtype is deprecated and will raise an error in a
future version of pandas. Value 'no result' has dtype incompatible with float64,
please explicitly cast to a compatible dtype first.


Number of missing result_margin entries: 5
```

[35]:
```
import pandas as pd
import plotly.graph_objects as go

def analyze_innings_scores(deliveries_df, matches_df):
    """
    For each team, create a dictionary mapping batting teams to their innings␣
↪scores (only innings 1 & 2).
    Plot the maximum and minimum scores for each team.
    Excludes matches with 'No Result' and innings 3-4.
    """
    # Filter out matches with 'No Result'
    valid_matches = matches_df[(matches_df['result'] != 'nr') &␣
↪(matches_df['method'] != 'D/L')]['id']
    deliveries_df = deliveries_df[deliveries_df['match_id'].isin(valid_matches)]

    # Filter out innings 3 and 4
    deliveries_df = deliveries_df[~deliveries_df['inning'].isin([3, 4, 5, 6])]

    # Group by match_id and inning
    innings_scores = deliveries_df.groupby(['match_id', 'inning']).agg(
        total_score=('total_runs', 'sum'),
        batting_team=('batting_team', 'first')
    ).reset_index()

    # Build team scores dictionary
    team_scores = {}
```

```
    for _, row in innings_scores.iterrows():
        team = row['batting_team']
        score = row['total_score']
        team_scores.setdefault(team, []).append(score)

    # Sort scores in descending order
    for team in team_scores:
        team_scores[team].sort(reverse=True)

    # Visualization
    teams = list(team_scores.keys())
    max_scores = [max(scores) for scores in team_scores.values()]
    min_scores = [min(scores) for scores in team_scores.values()]

    fig = go.Figure()
    fig.add_trace(go.Bar(
        x=teams,
        y=max_scores,
        name='Maximum Score',
        marker_color='indianred'  # Valid CSS color name
    ))
    fig.add_trace(go.Bar(
        x=teams,
        y=min_scores,
        name='Minimum Score',
        marker_color='lightsalmon'  # Valid CSS color name
    ))

    fig.update_layout(
        title='Maximum and Minimum Innings Scores by Team (Valid Matches Only)',
        xaxis=dict(title='Team', tickangle=-45),
        yaxis=dict(title='Runs'),
        barmode='group'
    )
    fig.show()

    return team_scores
team_scores = analyze_innings_scores(deliveries_df, matches_df)
```

10 *Total 4s and 6s*

[36]:
```
import pandas as pd
import plotly.graph_objects as go

def plot_total_boundaries(deliveries_df):
    """
```

```
    Calculate and plot total boundaries (4s & 6s) by team using vectorized␣
↪operations

    Parameters:
    deliveries_df (pd.DataFrame): Ball-by-ball delivery data with columns:
                                  ['batting_team', 'batsman_runs']
    """
    # Pre-filter boundary data
    boundaries = deliveries_df[deliveries_df['batsman_runs'].isin([4, 6])]

    # Group and count using pandas vectorized operations
    boundary_counts = (
        boundaries.groupby(['batting_team', 'batsman_runs'])
        .size()
        .unstack(fill_value=0)
        .rename(columns={4: 'fours', 6: 'sixes'})
        .reset_index()
    )

    # Create visualization
    fig = go.Figure()
    fig.add_trace(go.Bar(
        x=boundary_counts['batting_team'],
        y=boundary_counts['fours'],
        name='Fours',
        marker_color='#1f77b4',
        text=boundary_counts['fours'],
        textposition='auto'
    ))

    fig.add_trace(go.Bar(
        x=boundary_counts['batting_team'],
        y=boundary_counts['sixes'],
        name='Sixes',
        marker_color='#ff7f0e',
        text=boundary_counts['sixes'],
        textposition='auto'
    ))

    fig.update_layout(
        title='Total Boundaries Hit by Each Team (IPL History)',
        xaxis=dict(title='Team', tickangle=-45),
        yaxis=dict(title='Number of Boundaries'),
        barmode='group',
        hovermode='x unified',
        template='plotly_white'
    )
```

```
    fig.show()
    return boundary_counts

# Usage
boundary_data = plot_total_boundaries(deliveries_df)
```

11 *Power-Play Analysis*

[37]:
```
import pandas as pd
import plotly.graph_objects as go
def plot_phase_averages(deliveries_df, matches_df):
    # Filter valid matches (exclude No Result and D/L)
    valid_matches = matches_df[
        (matches_df['result'] != 'nr') &
        (matches_df['result'] != 'D/L')
    ]['id']

    # Filter deliveries data with proper phase handling
    filtered = deliveries_df[
        (deliveries_df['match_id'].isin(valid_matches)) &
        (deliveries_df['inning'].isin([1, 2])) &  # Only first two innings
        (deliveries_df['over'].between(1, 20))    # Ensure valid overs
    ].copy()

    # Create phase indicators with explicit bins
    filtered['phase'] = pd.cut(
        filtered['over'],
        bins=[1, 7, 16, 21],  # 1-6, 7-15, 16-20
        labels=['powerplay', 'middle', 'death'],
        right=False,
        include_lowest=True
    )

    # Calculate phase totals with fill_value=0
    phase_totals = filtered.groupby(
        ['batting_team', 'match_id', 'phase']
    )['total_runs'].sum().unstack(fill_value=0).reset_index()

    # Get total matches played per team
    matches_played = filtered.groupby('batting_team')['match_id'].nunique().
↪reset_index()

    # Merge with phase totals
    phase_avg = phase_totals.merge(
        matches_played,
```

```
        on='batting_team',
        how='left'
    ).fillna(0)

    # Calculate averages safely
    phase_avg['powerplay_avg'] = phase_avg['powerplay'] / phase_avg['match_id_y']
    phase_avg['death_avg'] = phase_avg['death'] / phase_avg['match_id_y']

    # Visualization (same as before)
    fig = go.Figure()
    fig = go.Figure()
    print(phase_avg)
    # Powerplay Plot
    fig.add_trace(go.Scatter(
        x=phase_avg['match_id_y'],
        y=phase_avg['powerplay_avg'].round(1),
        mode='markers+text',
        text=phase_avg['batting_team'],
        marker=dict(size=12, color='blue'),
        name='Powerplay',
        textposition='top center'
    ))

    # Death Overs Plot
    fig.add_trace(go.Scatter(
        x=phase_avg['match_id_y'],
        y=phase_avg['death_avg'].round(1),
        mode='markers+text',
        text=phase_avg['batting_team'],
        marker=dict(size=12, color='red'),
        name='Death Overs',
        textposition='bottom center'
    ))

    fig.update_layout(
        title='Batting Performance vs Matches Played',
        xaxis_title='Total Matches Played',
        yaxis_title='Average Runs',
        hovermode='closest',
        template='plotly_white'
    )

    fig.show()

    return phase_avg

# Usage
```

```
phase_data = plot_phase_averages(deliveries_df, matches_df)
```

```
C:\Users\SOHAM\AppData\Local\Temp\ipykernel_1684\1345819120.py:27:
FutureWarning:

The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
```

	batting_team	match_id_x	powerplay	middle	death	match_id_y	\
0	CSK	335982	0	0	0	267	
1	CSK	335983	57	107	71	267	
2	CSK	335984	0	0	0	267	
3	CSK	335985	0	0	0	267	
4	CSK	335986	0	0	0	267	
...	...	...	...	...	...	...	
10895	SRH	1426307	79	101	26	257	
10896	SRH	1426309	42	76	33	257	
10897	SRH	1426310	0	0	0	257	
10898	SRH	1426311	68	55	39	257	
10899	SRH	1426312	44	51	15	257	

	powerplay_avg	death_avg
0	0.000000	0.000000
1	0.213483	0.265918
2	0.000000	0.000000
3	0.000000	0.000000
4	0.000000	0.000000
...	...	...
10895	0.307393	0.101167
10896	0.163424	0.128405
10897	0.000000	0.000000
10898	0.264591	0.151751
10899	0.171206	0.058366

```
[10900 rows x 8 columns]
```

12 *Powerplay Analysis*

In most domestic leagues and international Twenty20 cricket, the first six overs of an innings will be a mandatory powerplay, with only two fielders allowed outside the 30-yard circle

[38]:
```
import math
# Define overs_mapping for powerplay
overs_mapping = {
    'power_play': [1, 2, 3, 4, 5, 6],
    'middle_overs': [7, 8, 9, 10, 11, 12, 13, 14, 15],
```

```
    'death_overs': [16, 17, 18, 19, 20]
}

# Initialize lists to store data for each over
power_play_runs_list = []
power_play_wickets_list = []
power_play_run_rate_list = []

unique_matches = deliveries_df['match_id'].nunique()  # Get the number of unique␣
↪matches
# Iterate through each over in the powerplay
for over in overs_mapping['power_play']:
    over_data = deliveries_df[deliveries_df['over'] == over]

    # Calculate runs, wickets, and run rate for the current over
    power_play_runs = math.floor(over_data['total_runs'].sum() / unique_matches)
    power_play_wickets = math.floor(over_data['is_wicket'].sum() /␣
↪unique_matches)
    power_play_run_rate = math.floor((power_play_runs / over_data['ball'].
↪nunique()) / unique_matches)

    # Append the calculated values to the respective lists
    power_play_runs_list.append(power_play_runs)
    power_play_wickets_list.append(power_play_wickets)
    power_play_run_rate_list.append(power_play_run_rate)

# Plot power_play_runs
fig_runs = go.Figure()
fig_runs.add_trace(go.Scatter(
    x=overs_mapping['power_play'],
    y=power_play_runs_list,
    mode='lines+markers',
    name='Power Play Runs',
    line=dict(color='blue'),
    marker=dict(size=8)
))
fig_runs.update_layout(
    title='Power Play Runs Over-wise',
    xaxis_title='Over',
    yaxis_title='Runs',
    template='plotly_white'
)
fig_runs.show()

# Plot power_play_wickets
fig_wickets = go.Figure()
fig_wickets.add_trace(go.Scatter(
```

```
    x=overs_mapping['power_play'],
    y=power_play_wickets_list,
    mode='lines+markers',
    name='Power Play Wickets',
    line=dict(color='red'),
    marker=dict(size=8)
))
fig_wickets.update_layout(
    title='Power Play Wickets Over-wise',
    xaxis_title='Over',
    yaxis_title='Wickets',
    template='plotly_white'
)
fig_wickets.show()
```

13 *Average Powerplay and Death Overs Score*

[39]:
```
import math

def plot_powerplay_vs_death_analysis(deliveries_df):
    """
    Analyze and plot comparative analysis of Power Play and Death Overs runs.

    Parameters:
    deliveries_df (pd.DataFrame): DataFrame containing ball-by-ball delivery␣
 ↪details.

    Returns:
    None
    """
    # Define overs_mapping for powerplay and death overs
    overs_mapping = {
        'power_play': [1, 2, 3, 4, 5, 6],
        'death_overs': [16, 17, 18, 19, 20]
    }

    # Initialize lists to store data for each phase
    power_play_runs_list = []
    death_overs_runs_list = []

    unique_matches = deliveries_df['match_id'].nunique()  # Get the number of␣
 ↪unique matches

    # Analyze Power Play runs
    for over in overs_mapping['power_play']:
        over_data = deliveries_df[deliveries_df['over'] == over]
```

```
        power_play_runs = math.floor(over_data['total_runs'].sum() /␣
↪unique_matches)
        power_play_runs_list.append(power_play_runs)

    # Analyze Death Overs runs
    for over in overs_mapping['death_overs']:
        over_data = deliveries_df[deliveries_df['over'] == over]
        death_overs_runs = math.floor(over_data['total_runs'].sum() /␣
↪unique_matches)
        death_overs_runs_list.append(death_overs_runs)

    # Plot comparative analysis
    fig = go.Figure()

    # Power Play Runs
    fig.add_trace(go.Scatter(
        x=overs_mapping['power_play'],
        y=power_play_runs_list,
        mode='lines+markers',
        name='Power Play Runs',
        line=dict(color='blue'),
        marker=dict(size=8)
    ))

    # Death Overs Runs
    fig.add_trace(go.Scatter(
        x=overs_mapping['death_overs'],
        y=death_overs_runs_list,
        mode='lines+markers',
        name='Death Overs Runs',
        line=dict(color='red'),
        marker=dict(size=8)
    ))

    fig.update_layout(
        title='Comparative Analysis: Power Play vs Death Overs Runs',
        xaxis_title='Over',
        yaxis_title='Average Runs',
        template='plotly_white'
    )
    fig.show()

# Usage
plot_powerplay_vs_death_analysis(deliveries_df)
```

14 *Player Performance Analysis*

15 Get the top 20 run-scorers

[40]:
```
# A measure of how many runs a batter scores per 100 balls faced
def plot_top_20_run_scorers(deliveries_df):
    """
    Top 20 Run Scoring Bastmen in IPL

    Parameters:
    deliveries_df (pd.DataFrame): DataFrame containing ball-by-ball delivery␣
↪details.
    """

    # Calculate total runs for each batsman
    batsmen = deliveries_df['batter'].unique()
    matches = deliveries_df['match_id'].unique()
    batsman_runs = {}

    for batsman in batsmen:
        batsman_data = deliveries_df[deliveries_df['batter'] == batsman]
        total_runs = batsman_data['batsman_runs'].sum()
        batsman_runs[batsman] = total_runs

    # Sort the batsmen by total runs scored in descending order and get the top␣
↪20
    top_20_scorers = sorted(batsman_runs.items(), key=lambda x: x[1],␣
↪reverse=True)[:20]

    # Extract names and scores for plotting
    names, runs = zip(*top_20_scorers)

    # Plot using Plotly
    fig = px.bar(
        x=names,
        y=runs,
        labels={'x': 'Batsman', 'y': 'Total Runs'},
        title='Top 20 Run-Scorers',
        color_discrete_sequence=['skyblue']
    )

    fig.update_xaxes(tickangle=45)
    fig.show()

plot_top_20_run_scorers(deliveries_df)
```

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

[41]:
```
import pandas as pd
import numpy as np
import plotly.graph_objects as go

def plot_batting_metrics(deliveries_df):
    """
    Analyze and visualize batting average vs strike rate for top 20 run-scorers
    """
    # Create dismissal indicator
    deliveries_df['is_dismissal'] = deliveries_df['player_dismissed'] ==␣
↪deliveries_df['batter']

    # Filter out extra deliveries (wides, etc.) for balls faced calculation
    valid_deliveries = deliveries_df[deliveries_df['extra_runs'] == 0]

    # Calculate batting statistics
    batter_stats = deliveries_df.groupby('batter').agg(
        total_runs=('batsman_runs', 'sum'),
        balls_faced=('ball', 'count'),
        dismissals=('is_dismissal', 'sum')
    ).reset_index()

    # Calculate metrics
    batter_stats['strike_rate'] = (batter_stats['total_runs'] /␣
↪batter_stats['balls_faced']) * 100
    batter_stats['batting_avg'] = batter_stats['total_runs'] /␣
↪batter_stats['dismissals'].replace(0, np.inf)

    # Filter top 20 run-scorers
    top_20 = batter_stats.nlargest(20, 'total_runs').sort_values('total_runs',␣
↪ascending=False)

    # Dynamically adjust axis ranges
    x_min, x_max = top_20['strike_rate'].min() - 10, top_20['strike_rate'].max()␣
↪+ 10
    y_min, y_max = top_20['batting_avg'].min() - 5, top_20['batting_avg'].max()␣
↪+ 10

    # Create interactive visualization
    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=top_20['strike_rate'],
        y=top_20['batting_avg'],
```

```
        mode='markers+text',
        text=top_20['batter'],
        textposition='top center',
        marker=dict(
            size=15,  # Increased size for better visibility
            color=top_20['total_runs'],
            colorscale='Viridis',
            showscale=True,
            colorbar=dict(title='Total Runs')
        ),
        hovertemplate=(
            "<b>%{text}</b><br>"
            "Strike Rate: %{x:.1f}<br>"
            "Batting Avg: %{y:.1f}<br>"
            "Total Runs: %{marker.color:,}"
        )
    ))

    fig.update_layout(
        title='Batting Average vs Strike Rate (Top 20 IPL Run-Scorers)',
        xaxis=dict(title='Strike Rate (runs per 100 balls)', range=[x_min,␣
↪x_max]),
        yaxis=dict(title='Batting Average (runs per dismissal)', range=[y_min,␣
↪y_max]),
        template='plotly_white',
        height=700,
        hoverlabel=dict(bgcolor="white")
    )

    fig.show()
    return top_20

top_batters = plot_batting_metrics(deliveries_df)
```

17 *Highest Average and Strike Rate for players with >50 matches*

[42]:
```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

def plot_highest_average_strike_rate(deliveries_df):
    """
    Calculate and plot the highest average and strike rate for players with more␣
↪than 50 matches using Plotly.

    Parameters:
```

```
    deliveries_df (pd.DataFrame): DataFrame containing ball-by-ball delivery␣
↪details.
    """

    # Data preparation
    batsmen = deliveries_df['batter'].unique()
    batsman_stats = {}

    for batsman in batsmen:
        batsman_data = deliveries_df[deliveries_df['batter'] == batsman]
        matches_played = len(batsman_data['match_id'].unique())

        if matches_played > 50:
            total_runs = batsman_data['batsman_runs'].sum()
            total_balls = batsman_data.shape[0]
            average_runs = total_runs / matches_played
            strike_rate = (total_runs / total_balls) * 100

            batsman_stats[batsman] = {
                'Average': average_runs,
                'Strike Rate': strike_rate
            }

    # Convert to DataFrame for Plotly
    stats_df = pd.DataFrame.from_dict(batsman_stats, orient='index').
↪reset_index()
    stats_df.columns = ['Batsman', 'Average', 'Strike Rate']

    # Plot highest averages and strike rates
    fig = go.Figure()

    fig.add_trace(go.Bar(
        x=stats_df['Batsman'],
        y=stats_df['Average'],
        name='Average',
        marker_color='indianred'
    ))

    fig.add_trace(go.Bar(
        x=stats_df['Batsman'],
        y=stats_df['Strike Rate'],
        name='Strike Rate',
        marker_color='lightseagreen'
    ))

    fig.update_layout(
        title='Highest Average and Strike Rate for Players with >50 Matches',
```

```
        xaxis=dict(title='Player'),
        yaxis=dict(title='Value'),
        barmode='group',
        xaxis_tickangle=-45
    )

    fig.show()

plot_highest_average_strike_rate(deliveries_df)
```

18 *Top wicket-takers*

[43]:
```
import plotly.express as px

def plot_top_wicket_takers(deliveries_df):
    """
    Calculate and plot the top wicket-takers from deliveries DataFrame using␣
↪Plotly.

    Parameters:
    deliveries_df (pd.DataFrame): DataFrame containing ball-by-ball delivery␣
↪details.
    """

    # Calculate total wickets for each bowler
    bowlers = deliveries_df['bowler'].unique()
    bowler_wickets = {}

    for bowler in bowlers:
        bowler_data = deliveries_df[deliveries_df['bowler'] == bowler]
        total_wickets = bowler_data['is_wicket'].sum()
        # total_wickets += bowler_data['player_dismissed' != 'No Dismissal'].
↪sum()
        bowler_wickets[bowler] = total_wickets

    # Sort the bowlers by total wickets taken in descending order and get the␣
↪top 10
    top_wicket_takers = sorted(bowler_wickets.items(), key=lambda x: x[1],␣
↪reverse=True)[:10]

    # Extract names and wickets for plotting
    names, wickets = zip(*top_wicket_takers)

    # Plot using Plotly
    fig = px.bar(
        x=names,
```

```
        y=wickets,
        labels={'x': 'Bowler', 'y': 'Total Wickets'},
        title='Top Wicket-Takers',
        color_discrete_sequence=['purple']
    )

    fig.update_xaxes(tickangle=45)
    fig.show()

plot_top_wicket_takers(deliveries_df)
```

19 *Top highest individual scores*

[44]:
```
import plotly.express as px

def plot_top_wicket_takers(deliveries_df):
    """
    Calculate and plot the top run-makers from deliveries DataFrame using Plotly.

    Parameters:
    deliveries_df (pd.DataFrame): DataFrame containing ball-by-ball delivery␣
 ↪details.
    """

    # Calculate total wickets for each bowler
    batsmen = deliveries_df['batter'].unique()
    batsman_run = {}

    for batsman in batsmen:
        batsman_data = deliveries_df[deliveries_df['batter'] == batsman]
        total_runs = batsman_data['batsman_runs'].sum()
        batsman_run[batsman] = total_runs

    # Sort the batsmen by total wickets taken in descending order and get the␣
 ↪top 10
    top_wicket_takers = sorted(batsman_run.items(), key=lambda x: x[1],␣
 ↪reverse=True)[:10]

    # Extract names and wickets for plotting
    names, wickets = zip(*top_wicket_takers)

    # Plot using Plotly
    fig = px.bar(
        x=names,
        y=wickets,
        labels={'x': 'Batsman', 'y': 'Total Runs'},
```

```
        title='Top Run Scores',
        color_discrete_sequence=['darkorange']
    )

    fig.update_xaxes(tickangle=45)
    fig.show()

plot_top_wicket_takers(deliveries_df)
```

20 *Man of the Match Count Analysis*

[45]:
```
import plotly.express as px

def plot_player_of_match_histogram(matches_df):
    """
    Calculate and visualize the frequency of 'Player of the Match' awards using␣
 ↪a Plotly histogram.

    Parameters:
    matches_df (pd.DataFrame): DataFrame containing match information.
    """

    # Calculate the count of 'Player of the Match' awards for each player
    player_of_match_counts = matches_df['player_of_match'].value_counts()

    # Filter out players with a count less than 1 (though it's logically␣
 ↪redundant)
    player_of_match_counts = player_of_match_counts[player_of_match_counts >= 1]

    # Plot histogram using Plotly
    fig = px.histogram(
        x=player_of_match_counts.index,
        y=player_of_match_counts.values,
        labels={'x': 'Player of the Match', 'y': 'Count'},
        title='Player of the Match Distribution',
        color_discrete_sequence=['mediumturquoise']
    )

    fig.update_xaxes(tickangle=45)
    fig.show()

plot_player_of_match_histogram(matches_df)
```

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

[46]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import plotly.express as px

# Calculate Batting Averages and Bowling Economy Rates
players = list(set(deliveries_df['batter']).union(set(deliveries_df['bowler'])))
player_stats = {}

for player in players:
    player_data_bat = deliveries_df[deliveries_df['batter'] == player]
    total_runs = player_data_bat['batsman_runs'].sum()
    matches_bat = player_data_bat['match_id'].nunique()
    bat_average = total_runs / matches_bat if matches_bat > 0 else 0

    player_data_bowl = deliveries_df[deliveries_df['bowler'] == player]
    total_wickets = player_data_bowl['is_wicket'].sum()
    total_runs_conceded = player_data_bowl['total_runs'].sum()
    total_overs_bowled = len(player_data_bowl) / 6
    bowl_economy = total_runs_conceded / total_overs_bowled if␣
 ↪total_overs_bowled > 0 else 0

    player_stats[player] = {
        'Batting Average': bat_average,
        'Bowling Economy': bowl_economy
    }

# Convert the stats to a DataFrame for KMeans
player_stats_df = pd.DataFrame.from_dict(player_stats, orient='index').fillna(0)

# K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
player_stats_df['Cluster'] = kmeans.fit_predict(player_stats_df[['Batting␣
 ↪Average', 'Bowling Economy']])

# Rename clusters for better understanding
cluster_labels = {
    0: 'Batsman',        # High Batting Average, High Bowling Economy
    1: 'Bowler',         # Low Bowling Economy, Low Batting Average
    2: 'All-Rounder'     # Medium Bowling Economy, Medium Batting Average
}
player_stats_df['Cluster'] = player_stats_df['Cluster'].map(cluster_labels)
```

```
# Plotting the results using Plotly
fig = px.scatter(
    player_stats_df,
    x='Batting Average',
    y='Bowling Economy',
    color='Cluster',
    hover_name=player_stats_df.index,
    title='K-Means Clustering: Batting Average vs Bowling Economy',
    labels={'Cluster': 'Player Type'},
    color_discrete_sequence=['skyblue', 'lightgreen', 'orange']
)

fig.show()
```

22 *Top 6's, Top 4's, Top 2's, Top 1's*

[47]:
```
# Plot for Top 6's Scorer
# Calculate top 10 players for 6's, 4's, 2's, and 1's
top_6s = deliveries_df[deliveries_df['batsman_runs'] == 6]['batter'].
 ↪value_counts().head(10)
top_4s = deliveries_df[deliveries_df['batsman_runs'] == 4]['batter'].
 ↪value_counts().head(10)
top_2s = deliveries_df[deliveries_df['batsman_runs'] == 2]['batter'].
 ↪value_counts().head(10)
top_1s = deliveries_df[deliveries_df['batsman_runs'] == 1]['batter'].
 ↪value_counts().head(10)

# Plot for Top 6's Scorer
fig_6s = px.bar(
    top_6s,
    x=top_6s.index,
    y=top_6s.values,
    labels={'x': "Batsman", 'y': "Number of 6's"},
    title="Top 10 6's Scorers",
    color_discrete_sequence=['#06E575']
)
fig_6s.update_traces(marker=dict(line=dict(color='black', width=1)))
fig_6s.update_xaxes(tickangle=45)
fig_6s.show()

# Plot for Top 4's Scorer
fig_4s = px.bar(
    top_4s,
    x=top_4s.index,
    y=top_4s.values,
```

```
    labels={'x': "Batsman", 'y': "Number of 4's"},
    title="Top 10 4's Scorers",
    color_discrete_sequence=['#AEEF38']
)
fig_4s.update_traces(marker=dict(line=dict(color='black', width=1)))
fig_4s.update_xaxes(tickangle=45)
fig_4s.show()

# Plot for Top 2's Scorer
fig_2s = px.bar(
    top_2s,
    x=top_2s.index,
    y=top_2s.values,
    labels={'x': "Batsman", 'y': "Number of 2's"},
    title="Top 10 2's Scorers",
    color_discrete_sequence=['#D8FE00']
)
fig_2s.update_traces(marker=dict(line=dict(color='black', width=1)))
fig_2s.update_xaxes(tickangle=45)
fig_2s.show()

# Plot for Top 1's Scorer
fig_1s = px.bar(
    top_1s,
    x=top_1s.index,
    y=top_1s.values,
    labels={'x': "Batsman", 'y': "Number of 1's"},
    title="Top 10 1's Scorers",
    color_discrete_sequence=['#FAE818']
)
fig_1s.update_traces(marker=dict(line=dict(color='black', width=1)))
fig_1s.update_xaxes(tickangle=45)
fig_1s.show()
```

23 *Seasonal Analysis*

24 *Average runs per Match per Season*

[48]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Merge matches and deliveries dataframes on match_id
merged_df = pd.merge(deliveries_df, matches_df[['id', 'season']],␣
 ↪left_on='match_id', right_on='id')

# Calculate total runs per player per season
```

```
player_runs = merged_df.groupby(['batter', 'season'])['batsman_runs'].sum().
↪reset_index()

player_runs
```

[48]:

	batter	season	batsman_runs
0	A Ashish Reddy	2012	35
1	A Ashish Reddy	2013	125
2	A Ashish Reddy	2015	73
3	A Ashish Reddy	2016	47
4	A Badoni	2022	161
...	...	...	...
2612	Z Khan	2011	21
2613	Z Khan	2012	12
2614	Z Khan	2014	8
2615	Z Khan	2016	6
2616	Z Khan	2017	4

[2617 rows x 3 columns]

[49]:
```
max_run_for_player = 0
for index, row in player_runs.iterrows():
    if row['batsman_runs'] > max_run_for_player:
        max_run_for_player = row['batsman_runs']
        player_name = row['batter']
        season = row['season']

print(f"Player with the highest runs in a season: {player_name} with␣
↪{max_run_for_player} runs in the {season} season")

merged_df
```

```
Player with the highest runs in a season: V Kohli with 973 runs in the 2016
season
```

[49]:

	match_id	inning	batting_team	bowling_team	over	ball	batter	\
0	335982	1	KKR	RCB	0	1	SC Ganguly	
1	335982	1	KKR	RCB	0	2	BB McCullum	
2	335982	1	KKR	RCB	0	3	BB McCullum	
3	335982	1	KKR	RCB	0	4	BB McCullum	
4	335982	1	KKR	RCB	0	5	BB McCullum	
...	...	...	...	...	...	...	...	
260915	1426312	2	KKR	SRH	9	5	SS Iyer	
260916	1426312	2	KKR	SRH	9	6	VR Iyer	
260917	1426312	2	KKR	SRH	10	1	VR Iyer	
260918	1426312	2	KKR	SRH	10	2	SS Iyer	
260919	1426312	2	KKR	SRH	10	3	VR Iyer	

```
              bowler non_striker  batsman_runs  extra_runs  total_runs  \
0            P Kumar          BM             0           1           1
1            P Kumar          SG             0           0           0
2            P Kumar          SG             0           1           1
3            P Kumar          SG             0           0           0
4            P Kumar          SG             0           0           0
...              ...         ...           ...         ...         ...
260915     AK Markram          VI             1           0           1
260916     AK Markram          SI             1           0           1
260917  Shahbaz Ahmed          SI             1           0           1
260918  Shahbaz Ahmed          VI             1           0           1
260919  Shahbaz Ahmed          SI             1           0           1

       extras_type  is_wicket player_dismissed dismissal_kind     fielder  \
0          legbyes          0     No Dismissal   No Dismissal  No Fielder
1         No Extra          0     No Dismissal   No Dismissal  No Fielder
2            wides          0     No Dismissal   No Dismissal  No Fielder
3         No Extra          0     No Dismissal   No Dismissal  No Fielder
4         No Extra          0     No Dismissal   No Dismissal  No Fielder
...            ...        ...              ...            ...         ...
260915    No Extra          0     No Dismissal   No Dismissal  No Fielder
260916    No Extra          0     No Dismissal   No Dismissal  No Fielder
260917    No Extra          0     No Dismissal   No Dismissal  No Fielder
260918    No Extra          0     No Dismissal   No Dismissal  No Fielder
260919    No Extra          0     No Dismissal   No Dismissal  No Fielder

        is_dismissal       id   season
0              False   335982  2007/08
1              False   335982  2007/08
2              False   335982  2007/08
3              False   335982  2007/08
4              False   335982  2007/08
...              ...      ...      ...
260915         False  1426312     2024
260916         False  1426312     2024
260917         False  1426312     2024
260918         False  1426312     2024
260919         False  1426312     2024

[260920 rows x 20 columns]
```

[50]:
```python
# Merge player_runs with merged_df on 'batter' and 'season'
combined_df = pd.merge(player_runs, merged_df[['match_id', 'season', 'batter']],␣
↪on=['batter', 'season'])

# Display the combined DataFrame
combined_df
```

[50]:
```
                batter season  batsman_runs  match_id
0       A Ashish Reddy   2012            35    548346
1       A Ashish Reddy   2012            35    548346
2       A Ashish Reddy   2012            35    548346
3       A Ashish Reddy   2012            35    548346
4       A Ashish Reddy   2012            35    548346
...                ...    ...           ...       ...
260915          Z Khan   2017             4   1082635
260916          Z Khan   2017             4   1082635
260917          Z Khan   2017             4   1082635
260918          Z Khan   2017             4   1082635
260919          Z Khan   2017             4   1082646

[260920 rows x 4 columns]
```

25 *Average Runs per Player Across the Seasons*

[51]:
```
import pandas as pd
import plotly.express as px

# Assuming deliveries_df and matches_df are available from previous context

# Merge deliveries and matches dataframes on match_id
merged_df = pd.merge(deliveries_df, matches_df[['id', 'season']],␣
 ↪left_on='match_id', right_on='id')

# Calculate total runs per player per season
player_runs = merged_df.groupby(['batter', 'season'])['batsman_runs'].sum().
 ↪reset_index()

# Calculate total matches per player per season
player_matches = merged_df.groupby(['batter', 'season'])['match_id'].nunique().
 ↪reset_index()

# Merge to get average runs per player per season
player_avg_runs = pd.merge(player_runs, player_matches, on=['batter', 'season'])
player_avg_runs['average_runs'] = player_avg_runs['batsman_runs'] /␣
 ↪player_avg_runs['match_id']

# Get unique seasons
seasons = player_avg_runs['season'].unique()

# Plotting the results for each season
# Define a color scheme for seasons
color_scheme = {
    season: px.colors.qualitative.Plotly[i % len(px.colors.qualitative.Plotly)]
```

```
    for i, season in enumerate(seasons)
}

for season in seasons:
    season_data = player_avg_runs[player_avg_runs['season'] == season]

    # Filter for notable players (e.g., top 30 by average runs)
    notable_players = season_data.nlargest(30, 'average_runs')

    fig = px.bar(
        notable_players,
        x='batter',
        y='average_runs',
        title=f'Notable Player Performance in Season {season}',
        hover_data=['batsman_runs', 'match_id'],
        color_discrete_sequence=[color_scheme[season]]
    )
    fig.update_layout(xaxis_title='Player', yaxis_title='Average Runs')
    fig.show()
```

26 *Identify targets of 200+ runs per Season*

[52]:
```
import pandas as pd
import plotly.express as px

# Sample data for demonstration
seasons = matches_df['season'].unique()
results = []

for season in seasons:
    season_data = matches_df[matches_df['season'] == season]
    match_ids = season_data['id'].unique()

    for match_id in match_ids:
        match_data = matches_df[matches_df['id'] == match_id]

        # Identify 200+ targets in each match per season
        num_200_targets_per_season = 0
        team_targets = {}
        for index, row in match_data.iterrows():
            if row['target_runs'] >= 200:
                num_200_targets_per_season += 1
                team_targets[(row['team1'], season)] = row['target_runs']

        # Store results
        for team, target in team_targets.items():
```

```
            results.append({
                'season': season,
                'team': team[0],
                'target_runs': target
            })

# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Plotting the results
fig = px.bar(results_df, x='season', y='target_runs', color='team', title='200+␣
 ↪Run Targets Per Season')
fig.update_layout(xaxis_title='Season', yaxis_title='Cumuative Target Runs in␣
 ↪the entire Tournament')
fig.show()
```

27 *Find the average score of each team per season*

[53]:
```
def plot_team_scores_per_season(matches_df):
    """
    Compute and visualize total team scores per season and overall average.
    """
    seasons = matches_df['season'].unique()
    results = []
    total_scores = {}

    for season in seasons:
        season_data = matches_df[matches_df['season'] == season]
        team_scores = season_data.groupby('team1')['target_runs'].sum().
 ↪reset_index()
        team_scores['season'] = season

        for _, row in team_scores.iterrows():
            results.append({
                'season': row['season'],
                'team': row['team1'],
                'total_runs': row['target_runs']
            })

            # Accumulate total scores for overall calculation
            if row['team1'] in total_scores:
                total_scores[row['team1']].append(row['target_runs'])
            else:
                total_scores[row['team1']] = [row['target_runs']]

    results_df = pd.DataFrame(results)
```

```
    # Calculate overall total and average scores for each team
    avg_scores = {team: np.mean(scores) for team, scores in total_scores.items()}
    total_scores_all_time = {team: np.sum(scores) for team, scores in␣
↪total_scores.items()}

    avg_scores_df = pd.DataFrame(list(avg_scores.items()), columns=['team',␣
↪'average_runs_per_season'])
    total_scores_df = pd.DataFrame(list(total_scores_all_time.items()),␣
↪columns=['team', 'total_runs_all_time'])

    # Plotting the results
    fig = px.bar(results_df, x='season', y='total_runs', color='team',␣
↪title='Total Runs Scored by Teams Per Season')
    fig.update_layout(xaxis_title='Season', yaxis_title='Total Runs in the␣
↪Tournament')
    fig.show()

    # Display overall average scores
    print("Overall Average Scores Per Season:")
    print(avg_scores_df)
    print("\nTotal Runs Scored by Teams Throughout IPL:")
    print(total_scores_df)

    return results_df, avg_scores_df, total_scores_df

team_scores, avg_team_scores, total_team_scores =␣
↪plot_team_scores_per_season(matches_df)
```

```
Overall Average Scores Per Season:
   team  average_runs_per_season
0   CSK              1416.235294
1    DD              1207.117647
2   KKR              1152.882353
3    MI              1210.588235
4  PBKS              1206.882353
5   RCB              1409.000000
6    RR              1206.000000
7   SRH              1191.117647
8    GT              1187.750000
9   LSG              1230.833333

Total Runs Scored by Teams Throughout IPL:
   team  total_runs_all_time
0   CSK              24076.0
1    DD              20521.0
2   KKR              19599.0
```

```
3    MI              20580.0
4  PBKS              20517.0
5   RCB              23953.0
6    RR              19296.0
7   SRH              20249.0
8    GT               4751.0
9   LSG               7385.0
```

28 *Find top 10 bowlers per season*

We have taken here the liberty of defining our own criteria for deciding who the better bowler is based on common Quality Testing Metrics

[54]:
```python
# Define weightage for each metric
score_for_each_metric = {'total_runs': -1, 'is_wicket': 5, 'extra_runs': -2}

# Get unique bowlers
bowlers = merged_df['bowler'].unique()

# Initialize dictionary to store bowler stats
bowler_stats = {}

# Calculate scores for each bowler per season
for season in merged_df['season'].unique():
    season_data = merged_df[merged_df['season'] == season]
    for bowler in bowlers:
        bowler_data = season_data[season_data['bowler'] == bowler]

        # Calculate metrics
        wickets = bowler_data['is_wicket'].sum()
        total_runs = bowler_data['total_runs'].sum()
        extra_runs = bowler_data['extra_runs'].sum()

        # Calculate score
        score = (wickets * score_for_each_metric['is_wicket'] +
                 total_runs * score_for_each_metric['total_runs'] +
                 extra_runs * score_for_each_metric['extra_runs'])

        if season not in bowler_stats:
            bowler_stats[season] = {}
        bowler_stats[season][bowler] = score

# Find top 10 bowlers per season
top_10_bowlers_per_season = {}
for season, scores in bowler_stats.items():
    # Normalize the scores for top 10 bowlers
    max_score = max(scores.values())
```

```
    min_score = min(scores.values())
    sorted_bowlers = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    normalized_scores = [(bowler, (score - min_score) / (max_score - min_score))␣
↪for bowler, score in sorted_bowlers]
    top_10_bowlers_per_season[season] = normalized_scores[:10]


# Display top 10 bowlers per season
top_10_bowlers_per_season
```

[54]:
```
{'2007/08': [('TM Dilshan', 1.0),
  ('T Thushara', 0.9954954954954955),
  ('A Flintoff', 0.9954954954954955),
  ('SL Malinga', 0.9954954954954955),
  ('Kamran Khan', 0.9954954954954955),
  ('T Henderson', 0.9954954954954955),
  ('JD Ryder', 0.9954954954954955),
  ('DP Nannes', 0.9954954954954955),
  ('AM Salvi', 0.9954954954954955),
  ('YA Abdulla', 0.9954954954954955)],
 '2009': [('AA Noffke', 1.0),
  ('SB Joshi', 1.0),
  ('CL White', 1.0),
  ('JR Hopes', 1.0),
  ('P Amarnath', 1.0),
  ('GD McGrath', 1.0),
  ('B Geeves', 1.0),
  ('SR Watson', 1.0),
  ('D Salunkhe', 1.0),
  ('SM Pollock', 1.0)],
 '2009/10': [('AA Noffke', 1.0),
  ('SB Joshi', 1.0),
  ('JR Hopes', 1.0),
  ('WA Mota', 1.0),
  ('JDP Oram', 1.0),
  ('P Amarnath', 1.0),
  ('GD McGrath', 1.0),
  ('B Geeves', 1.0),
  ('D Salunkhe', 1.0),
  ('SM Pollock', 1.0)],
 '2011': [('AA Noffke', 1.0),
  ('SB Joshi', 1.0),
  ('CL White', 1.0),
  ('SC Ganguly', 1.0),
  ('LR Shukla', 1.0),
  ('K Goel', 1.0),
  ('WA Mota', 1.0),
```

```
 ('P Amarnath', 1.0),
 ('GD McGrath', 1.0),
 ('B Geeves', 1.0)],
'2012': [('AA Noffke', 1.0),
 ('SB Joshi', 1.0),
 ('I Sharma', 1.0),
 ('S Sreesanth', 1.0),
 ('JR Hopes', 1.0),
 ('K Goel', 1.0),
 ('WA Mota', 1.0),
 ('JDP Oram', 1.0),
 ('P Amarnath', 1.0),
 ('Joginder Sharma', 1.0)],
'2013': [('AC Gilchrist', 1.0),
 ('AA Noffke', 0.9899598393574297),
 ('SB Joshi', 0.9899598393574297),
 ('SC Ganguly', 0.9899598393574297),
 ('JR Hopes', 0.9899598393574297),
 ('K Goel', 0.9899598393574297),
 ('WA Mota', 0.9899598393574297),
 ('P Amarnath', 0.9899598393574297),
 ('Joginder Sharma', 0.9899598393574297),
 ('GD McGrath', 0.9899598393574297)],
'2014': [('AA Noffke', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('B Lee', 1.0),
 ('S Sreesanth', 1.0),
 ('JR Hopes', 1.0),
 ('K Goel', 1.0),
 ('WA Mota', 1.0)],
'2015': [('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('LR Shukla', 1.0),
 ('B Lee', 1.0),
 ('S Sreesanth', 1.0),
 ('JR Hopes', 1.0)],
'2016': [('Sachin Baby', 1.0),
 ('AA Noffke', 0.9849137931034483),
 ('JH Kallis', 0.9849137931034483),
 ('SB Joshi', 0.9849137931034483),
```

```
 ('CL White', 0.9849137931034483),
 ('AB Agarkar', 0.9849137931034483),
 ('SC Ganguly', 0.9849137931034483),
 ('LR Shukla', 0.9849137931034483),
 ('B Lee', 0.9849137931034483),
 ('S Sreesanth', 0.9849137931034483)],
'2017': [('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('LR Shukla', 1.0),
 ('B Lee', 1.0),
 ('S Sreesanth', 1.0),
 ('JR Hopes', 1.0)],
'2018': [('P Kumar', 1.0),
 ('Z Khan', 1.0),
 ('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Dinda', 1.0),
 ('I Sharma', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0)],
'2019': [('P Kumar', 1.0),
 ('Z Khan', 1.0),
 ('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Dinda', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('LR Shukla', 1.0)],
'2020/21': [('P Kumar', 1.0),
 ('Z Khan', 1.0),
 ('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Dinda', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('LR Shukla', 1.0)],
'2021': [('P Kumar', 1.0),
```

```
 ('Z Khan', 1.0),
 ('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Dinda', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('LR Shukla', 1.0)],
'2022': [('P Kumar', 1.0),
 ('Z Khan', 1.0),
 ('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Dinda', 1.0),
 ('I Sharma', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0)],
'2023': [('P Kumar', 1.0),
 ('Z Khan', 1.0),
 ('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Dinda', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('LR Shukla', 1.0)],
'2024': [('P Kumar', 1.0),
 ('Z Khan', 1.0),
 ('AA Noffke', 1.0),
 ('JH Kallis', 1.0),
 ('SB Joshi', 1.0),
 ('CL White', 1.0),
 ('AB Dinda', 1.0),
 ('AB Agarkar', 1.0),
 ('SC Ganguly', 1.0),
 ('LR Shukla', 1.0)]}
```

29 *Winning Percentages of Teams*

[55]:
```
import plotly.express as px

# Data for the Pie-Plot
labels = list(win_percentage.keys())
```

```
values = list(win_percentage.values())

# Create the Pie-Plot
fig = px.pie(
    names=labels,
    values=values,
    title="Win Percentage of Teams",
    color_discrete_sequence=px.colors.qualitative.Bold  # Use a more vibrant␣
 ↪color palette
)

# Update layout for better visualization
fig.update_traces(textinfo='percent+label', pull=[0.1 if value == max(values)␣
 ↪else 0 for value in values])
fig.update_layout(
    showlegend=True,
    title_font_size=20,  # Increase title font size
    height=800,  # Increase figure height
    width=800   # Increase figure width
)

# Show the plot
fig.show()
```

30 *Top 10 Bowlers Per Season*

[56]:
```
import pandas as pd
import plotly.express as px


# Define weightage for each metric
score_for_each_metric = {'total_runs': 3, 'is_wicket': 5, 'extra_runs': -2}

# Get unique bowlers
bowlers = merged_df['bowler'].unique()

# Initialize dictionary to store bowler stats
bowler_stats = {}

# Calculate scores for each bowler per season
for season in merged_df['season'].unique():
    season_data = merged_df[merged_df['season'] == season]
    for bowler in bowlers:
        bowler_data = season_data[season_data['bowler'] == bowler]

        # Calculate metrics
```

```
        wickets = bowler_data['is_wicket'].sum()
        total_runs = bowler_data['total_runs'].sum()
        extra_runs = bowler_data['extra_runs'].sum()

        # Calculate score
        score = (wickets * score_for_each_metric['is_wicket'] +
                 total_runs * score_for_each_metric['total_runs'] +
                 extra_runs * score_for_each_metric['extra_runs'])

        if season not in bowler_stats:
            bowler_stats[season] = {}
        bowler_stats[season][bowler] = score

# Find top 10 bowlers per season
top_10_bowlers_per_season = {}
for season, scores in bowler_stats.items():
    # Normalize the scores for top 10 bowlers
    max_score = max(scores.values())
    min_score = min(scores.values())
    sorted_bowlers = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    normalized_scores = [(bowler, (score - min_score) / (max_score - min_score))␣
 ↪for bowler, score in sorted_bowlers]
    top_10_bowlers_per_season[season] = normalized_scores[:10]


# Display top 10 bowlers per season
top_10_bowlers_per_season

# Prepare data for plotting
plot_data = []

for season, bowlers in top_10_bowlers_per_season.items():
    for bowler, normalized_score in bowlers:
        # Retrieve the original metrics for the bowler
        bowler_data = merged_df[(merged_df['bowler'] == bowler) &␣
 ↪(merged_df['season'] == season)]
        total_runs = bowler_data['total_runs'].sum()
        is_wicket = bowler_data['is_wicket'].sum()
        extra_runs = bowler_data['extra_runs'].sum()

        # Append to the plot data
        plot_data.append({
            'Season': season,
            'Bowler': bowler,
            'Normalized Score': normalized_score,
            'Total Runs': total_runs,
            'Wickets': is_wicket,
```

```
            'Extra Runs': extra_runs
        })

# Convert to DataFrame
plot_df = pd.DataFrame(plot_data)

# Plot using Plotly Scatterplot
fig = px.scatter(
    plot_df,
    x='Bowler',
    y='Normalized Score',
    color='Season',
    size='Wickets',  # Use Wickets as the size of the markers
    hover_data=['Total Runs', 'Extra Runs', 'Wickets'],
    title='Top 10 Bowlers Per Season with Metrics',
    labels={'Normalized Score': 'Normalized Score', 'Bowler': 'Bowler'}
)

fig.update_layout(
    xaxis_title='Bowler',
    yaxis_title='Normalized Score',
    xaxis_tickangle=45,
    height=600,  # Adjust height for better visibility
    width=1000   # Adjust width for better visibility
)
fig.show()
```

[57]:
```
import pandas as pd
import plotly.express as px

# Define weightage for each metric
score_for_each_metric = {'total_runs': 3, 'is_wicket': 5, 'extra_runs': -2}

# Get unique bowlers
bowlers = merged_df['bowler'].unique()

# Initialize dictionary to store bowler stats
bowler_stats = {}

# Calculate scores for each bowler per season
for season in merged_df['season'].unique():
    season_data = merged_df[merged_df['season'] == season]
    for bowler in bowlers:
        bowler_data = season_data[season_data['bowler'] == bowler]

        # Calculate metrics
        wickets = bowler_data['is_wicket'].sum()
```

```
        total_runs = bowler_data['total_runs'].sum()
        extra_runs = bowler_data['extra_runs'].sum()

        # Calculate score
        score = (wickets * score_for_each_metric['is_wicket'] +
                 total_runs * score_for_each_metric['total_runs'] +
                 extra_runs * score_for_each_metric['extra_runs'])

        if season not in bowler_stats:
            bowler_stats[season] = {}
        bowler_stats[season][bowler] = score

# Find top 10 bowlers per season and prepare plot data
plot_data = []
for season, scores in bowler_stats.items():
    # Normalize the scores for top 10 bowlers
    max_score = max(scores.values())
    min_score = min(scores.values())
    sorted_bowlers = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    normalized_scores = [(bowler, (score - min_score) / (max_score - min_score))␣
 ↪for bowler, score in sorted_bowlers]
    top_10_bowlers = normalized_scores[:10]

    # Append to plot data
    for bowler, normalized_score in top_10_bowlers:
        plot_data.append({
            'Season': season,
            'Bowler': bowler,
            'Normalized Score': normalized_score
        })

# Convert to DataFrame
plot_df = pd.DataFrame(plot_data)

# Define a color scheme for seasons
# Define a color scheme for seasons
color_scheme = {
    season: px.colors.sequential.Plasma
    for i, season in enumerate(plot_df['Season'].unique())
}

# Create separate sunburst charts for each season
seasons = plot_df['Season'].unique()

for season in seasons:
    season_data = plot_df[plot_df['Season'] == season]
```

```
    # Create a sunburst chart for the current season
    fig = px.sunburst(
        season_data,
        path=['Season', 'Bowler'],  # Hierarchical levels: Season -> Bowler
        values='Normalized Score',  # Use normalized scores as the size
        color='Normalized Score',  # Color by normalized score
        color_continuous_scale='Plasma',
        title=f'Top 10 Bowlers for Season {season} (Sunburst Chart)',
        labels={'Normalized Score': 'Normalized Score'}
    )

    # Update layout for better visualization
    fig.update_layout(
        template='plotly_white',
        title_font_size=20,
        margin=dict(t=50, l=25, r=25, b=25)
    )

    # Show the plot
    fig.show()
```

31 *Analyze runs of Orange Cap Holders per season*

[58]:
```
def calculate_orange_cap_holders(deliveries_df):
    """
    Compute Orange Cap holders (highest run-scorers) per season and plot results.
    """
    orange_cap_holders = deliveries_df.groupby(['season',␣
↪'batter'])['batsman_runs'].sum().reset_index()
    orange_players = orange_cap_holders.loc[orange_cap_holders.
↪groupby('season')['batsman_runs'].idxmax()]

    # Plot Orange Cap holders per season
    fig = px.bar(orange_players, x='season', y='batsman_runs', color='batter',␣
↪title='Orange Cap Holders Per Season')
    fig.update_layout(xaxis_title='Season', yaxis_title='Total Runs')
    fig.show()

    return orange_players
orange_cap_holders = calculate_orange_cap_holders(merged_df)
```

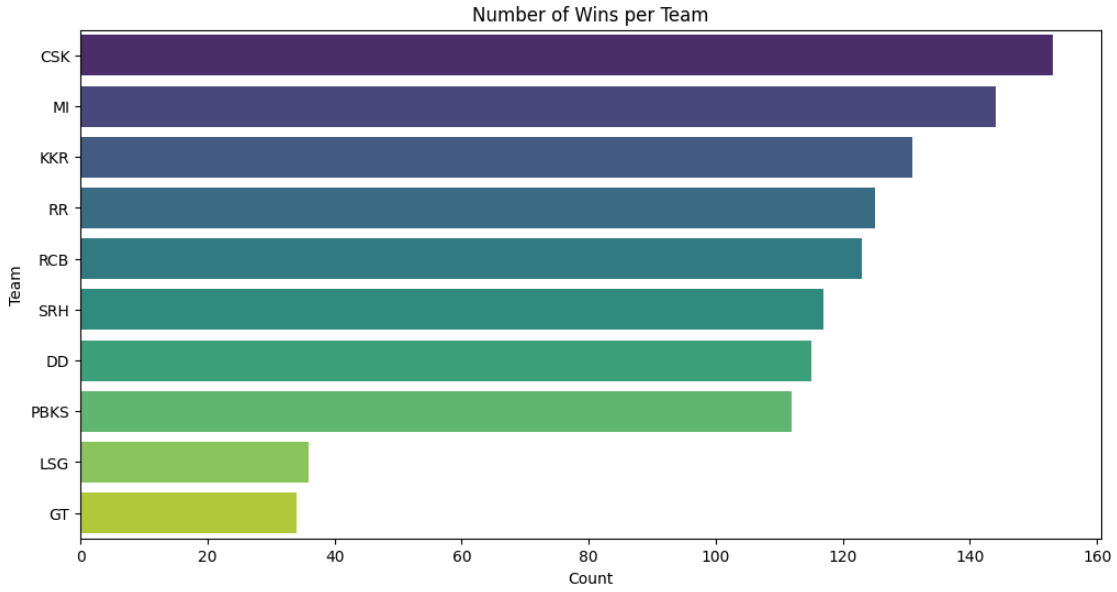
32 *Track wickets of Purple Cap Holders per season*

[59]:
```
def calculate_purple_cap_holders(merged_df):
    """
    Compute Purple Cap holders per season based on a weighted score and plot␣
 ↪results.
    """
    score_for_each_metric = {'total_runs': 3, 'is_wicket': 1, 'extra_runs': -2}
    bowlers = merged_df['bowler'].unique()
    bowler_stats = {}

    for season in merged_df['season'].unique():
        season_data = merged_df[merged_df['season'] == season]
        for bowler in bowlers:
            bowler_data = season_data[season_data['bowler'] == bowler]
            wickets = bowler_data['is_wicket'].sum()

            score = wickets

            if season not in bowler_stats:
                bowler_stats[season] = {}
            bowler_stats[season][bowler] = score

    purple_cap_holders = {season: max(scores.items(), key=lambda x: x[1]) for␣
 ↪season, scores in bowler_stats.items()}

    # Convert results to DataFrame for plotting
    purple_cap_df = pd.DataFrame(purple_cap_holders.items(), columns=['season',␣
 ↪'bowler_score'])
    purple_cap_df[['bowler', 'score']] = purple_cap_df['bowler_score'].apply(pd.
 ↪Series)

    # Plot Purple Cap holders per season
    fig = px.bar(purple_cap_df, x='season', y='score', color='bowler',␣
 ↪title='Purple Cap Holders Per Season')
    fig.update_layout(xaxis_title='Season', yaxis_title='Wickets Taken')
    fig.show()

    return purple_cap_holders
purple_cap_holders = calculate_purple_cap_holders(merged_df)
```

33 *Deeper Analysis of the Datasets*

[60]:
```
def extract_match_features(matches_df):
    """
    Extract key features from matches dataset.
    """
    match_features = matches_df[['id', 'season', 'team1', 'team2', 'winner',␣
 ↪'toss_winner', 'toss_decision', 'result']]
    return match_features

def extract_delivery_insights(deliveries_df):
    """
    Extract crucial insights from deliveries dataset.
    """
    delivery_insights = deliveries_df.groupby(['match_id', 'batter']).agg(
        total_runs=('batsman_runs', 'sum'),
        balls_faced=('ball', 'count'),
        strike_rate=('batsman_runs', lambda x: (x.sum() / len(x)) * 100),
        dismissals=('is_wicket', 'sum')
    ).reset_index()

    return delivery_insights

def plot_match_wins(match_features):
    """
    Plot the number of wins per team.
    """
    plt.figure(figsize=(12, 6))
    sns.countplot(y=match_features['winner'], order=match_features['winner'].
 ↪value_counts().index, palette='viridis')
    plt.title("Number of Wins per Team")
    plt.xlabel("Count")
    plt.ylabel("Team")
    plt.show()
match_features = extract_match_features(matches_df)
plot_match_wins(match_features)
delivery_insights = extract_delivery_insights(deliveries_df)
```

```
C:\Users\SOHAM\AppData\Local\Temp\ipykernel_1684\2942986324.py:26:
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.
```



Pictorial Representation of Total number of wins for each Team over the years

[61]:
```
merged_df = pd.merge(deliveries_df, matches_df, left_on='match_id',␣
↪right_on='id')
merged_df.head()
```

[61]:

	match_id	inning	batting_team	bowling_team	over	ball	batter	\
0	335982	1	KKR	RCB	0	1	SC Ganguly	
1	335982	1	KKR	RCB	0	2	BB McCullum	
2	335982	1	KKR	RCB	0	3	BB McCullum	
3	335982	1	KKR	RCB	0	4	BB McCullum	
4	335982	1	KKR	RCB	0	5	BB McCullum	

	bowler	non_striker	batsman_runs	...	toss_decision	winner	result	\
0	P Kumar	BM	0	...	field	KKR	runs	
1	P Kumar	SG	0	...	field	KKR	runs	
2	P Kumar	SG	0	...	field	KKR	runs	
3	P Kumar	SG	0	...	field	KKR	runs	
4	P Kumar	SG	0	...	field	KKR	runs	

	result_margin	target_runs	target_overs	super_over	method	umpire1	umpire2
0	140.0	223.0	20.0	0	NaN	AR	RK
1	140.0	223.0	20.0	0	NaN	AR	RK
2	140.0	223.0	20.0	0	NaN	AR	RK
3	140.0	223.0	20.0	0	NaN	AR	RK
4	140.0	223.0	20.0	0	NaN	AR	RK

[5 rows x 38 columns]

[62]:
```
import plotly.express as px
import plotly.graph_objects as go

def analyze_victory_correlation_plotly(merged_df):
    """
    Analyze correlation between runs scored and margin of victory using Plotly.
    """
    # Prepare the data
    team_runs = merged_df.groupby(['match_id', 'winner', 'batting_team']).
↪agg(total_runs=('total_runs', 'sum')).reset_index()
    victory_df = merged_df[['match_id', 'result_margin', 'result']].
↪drop_duplicates()
    victory_df = victory_df[~victory_df['result'].isin(['wickets'])]
    final_df = team_runs.merge(victory_df, on='match_id')

    # Create the scatter plot
    fig = px.scatter(
        final_df,
        x='total_runs',
        y='result_margin',
        color='winner',
        hover_data=['batting_team', 'result_margin'],
        title="Correlation Between Runs Scored and Victory Margin",
        labels={'total_runs': 'Total Runs Scored', 'result_margin': 'Win By␣
↪Runs'}
    )

    # Add dropdown menu for team selection
    fig.update_layout(
        updatemenus=[
            {
                "buttons": [
                    {
                        "label": "All Teams",
                        "method": "update",
                        "args": [{"visible": [True] * len(final_df['winner'].
↪unique())}]
                    }
                ] + [
                    {
                        "label": team,
                        "method": "update",
                        "args": [
                            {"visible": [winner == team for winner in␣
↪final_df['winner']]},
```

```
                            {"title": f"Correlation Between Runs Scored and␣
↪Victory Margin - {team}"}
                        ]
                    }
                    for team in final_df['winner'].unique()
                ],
                "direction": "down",
                "showactive": True
            }
        ]
    )

    fig.show()

# Call the function
analyze_victory_correlation_plotly(merged_df)
```

An Interactive plot for viewer to analyse in depth the Victory correlation with the Runs Scored for different teams

[63]:
```
import pandas as pd

# Define IPL 2025 teams and their home cities
ipl_teams_home_city = {
    "CSK": "Chennai",
    "DD": "Delhi",
    "GT": "Ahmedabad",
    "KKR": "Kolkata",
    "LSG": "Lucknow",
    "MI": "Mumbai",
    "PBKS": "Mohali",
    "PBKS": "Chandigarh",
    "RR": "Jaipur",
    "RCB": "Bangalore",
    "SRH": "Hyderabad"
}

def calculate_home_advantage(merged_df):
    """
    Calculates home field advantage for each IPL team based on home city␣
↪performance.

    Parameters:
    merged_df (pd.DataFrame): DataFrame containing match and delivery details.

    Returns:
    pd.DataFrame: DataFrame with Home Field Advantage for each team.
```

```
    """

    home_wins = {}
    home_matches = {}

    for index, row in merged_df.iterrows():
        home_team = None

        # Determine if team1 or team2 is playing at home
        if row['team1'] in ipl_teams_home_city and row['city'] ==␣
↪ipl_teams_home_city[row['team1']]:
            home_team = row['team1']
        elif row['team2'] in ipl_teams_home_city and row['city'] ==␣
↪ipl_teams_home_city[row['team2']]:
            home_team = row['team2']

        # Count matches and wins at home
        if home_team:
            home_matches[home_team] = home_matches.get(home_team, 0) + 1
            if row['winner'] == home_team:
                home_wins[home_team] = home_wins.get(home_team, 0) + 1
    print(home_matches)
    # Compute home field advantage
    home_advantage = {}
    for team in ipl_teams_home_city.keys():
        home_win_rate = home_wins.get(team, 0) / home_matches.get(team, 1)  #␣
↪Avoid division by zero
        home_advantage[team] = home_win_rate
    return pd.DataFrame(list(home_advantage.items()), columns=['Team', 'Home␣
↪Field Advantage'])

# Call the function with merged_df
home_advantage_df = calculate_home_advantage(merged_df)
home_advantage_df['Home Field Advantage'] = home_advantage_df['Home Field␣
↪Advantage'] * 100 # Percentage of Victory in the Home Field

# Display results
home_advantage_df
```

```
{'RCB': 14049, 'PBKS': 14476, 'DD': 19319, 'MI': 25078, 'KKR': 20594, 'RR':
13520, 'SRH': 17860, 'CSK': 17331, 'GT': 3804, 'LSG': 3283}
```

[63]:

	Team	Home Field Advantage
0	CSK	70.278691
1	DD	45.856411
2	GT	57.939012
3	KKR	58.813247

```
4   LSG             52.208346
5    MI             61.432331
6  PBKS             49.875656
7    RR             65.170118
8   RCB             46.722187
9   SRH             52.278835
```

[65]:
```
# Prepare data for the sunburst chart
home_advantage_df['region'] = 'India'  # Top-level category
home_advantage_df['team_category'] = home_advantage_df['Home Field Advantage'].
 ↪apply(
    lambda x: 'High Advantage (>50%)' if x > 50 else 'Low Advantage (50%)'
)

# Create the sunburst chart
fig = px.sunburst(
    home_advantage_df,
    path=['region', 'team_category', 'Team'],  # Hierarchy
    values='Home Field Advantage',  # Size of segments
    color='Home Field Advantage',  # Color scale
    color_continuous_scale='Viridis',
    title='IPL Teams: Home Field Advantage (Win Rate %)'
)

# Update layout for better readability
fig.update_layout(
    template='plotly_dark',
    title_font_size=20,
    margin=dict(t=50, l=25, r=25, b=25)
)

# Show the figure
fig.show()
```

34 *Inference*

We can see that there is a home field advantage for all the teams. The teams with the highest home field advantage are CSK and MI with 70% and 60% respectively. The teams with the lowest home field advantage are PBKS and RCB with 50% and 46% respectively.

- Possible reasons for such an Outcome maybe Familiarity with the Field, Pitch, Ground and Climatic conditions of the Teams.
- However, three teams PBKS, RCB and DD find themselves at a low advantage than other teams. Possible due to inabaptibility to their homefields or some other reason
- This is both a reflection of the number of matches the teams have palyed at home and their familiarity with their city and home ground.