IPL

INDIAN PREMIER LEAGUE

THE WORLD'S MOST POPULAR T20 CRICKET TOURNAMENTS

MORE THAN JUST A GAME

This is about IPL Historical Analysis , Its Culture, and Global Impact

Varun Kumar

Data Analyst

INDIAN PREMIER LEAGUE®

IPL

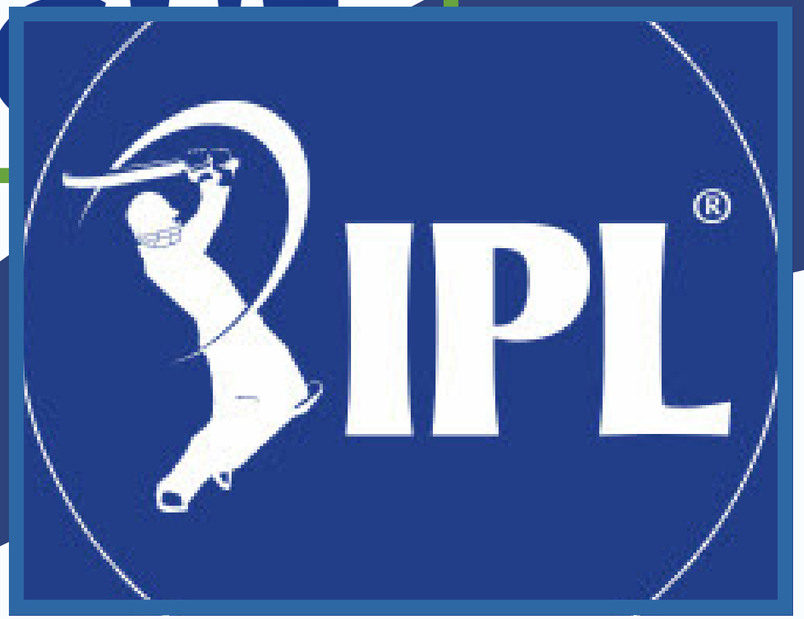
WHAT IS INDIAN PREMIER LEAGUE?

The Indian Premier League (IPL) is a thrilling professional T20 cricket league in India, known for its fast-paced matches, superstar players, and massive global fanbase. It features teams representing major cities, each packed with international and domestic talent. Beyond just sport, IPL blends entertainment, strategy, and fierce competition—making it one of the most watched cricket events in the world

IINDUAN PREMIER LEAGUE

IPL PROJECT ASSIGNMENT

Aims to leverage historical data to uncover actionable insights on player performance, team strategies, match outcomes, and fan engagement trends. The objective is to analyze match-by-match data, player statistics, and season trends to support franchise decision-making in player auctions, match tactics, audience engagement, and performance forecasting.



INDIAN PREMIER LEAGUE

WHAT TOOLS TO DO ANALYSIS ?

MYSQL — IPL Analytical Questions

MICROSOFT POWER BI — Dashboards

PYTHON — Exploratory Data Analysis (EDA)

INDIAN PREMIER LEAGUE

MYSQL: IPL ANALYTICAL QUESTIONS

1. Which batsman has scored the highest total runs?
2. Which team has conceded the least extras in the matches?
3. What is the total number of runs scored by each team across all matches?
4. Who are the top 5 bowlers based on runs conceded per over (economy rate)?
5. Which batsman has faced the most balls?
6. Which bowling team has taken part in the most overs?
7. What is the average number of runs scored per ball by each batsman?
8. Which teams have the best win percentage according to the team_performance table?
9. List all players who are all-rounders (both batting and bowling style available).
10. What is the average number of runs scored per over by each batting team?

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

1. WHICH BATSMAN HAS SCORED THE HIGHEST TOTAL RUNS?

MYSQL QUERY:-

```
SELECT Striker, SUM(runs_scored) AS total_runs
FROM ipl_ballbyball2008_2024_updated
GROUP BY Striker
ORDER BY total_runs DESC
LIMIT 1;
```

IPL

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

2.WHICH TEAM HAS CONCEDED THE LEAST EXTRAS IN THE MATCHES?

MYSQL QUERY:-

```
SELECT Bowling_team, SUM(extras) AS total_extras
FROM ipl_ballbyball2008_2024_updated
GROUP BY Bowling_team
ORDER BY total_extras ASC
LIMIT 1;
```

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

3.WHAT IS THE TOTAL NUMBER OF RUNS SCORED BY EACH TEAM ACROSS ALL MATCHES?

MYSQL QUERY:-

```
SELECT Batting_team, SUM(runs_scored + extras) AS total_runs
FROM ipl_ballbyball2008_2024_updated
GROUP BY Batting_team
ORDER BY total_runs DESC;
```

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

4.WHO ARE THE TOP 5 BOWLERS BASED ON RUNS CONCEDED PER OVER (ECONOMY RATE)?

MYSQL QUERY:-

```
SELECT
    Bowler,
    ROUND(SUM(runs_scored + extras) / (SUM(CASE
                                           WHEN type_of_extras NOT IN ('wide', 'no-ball') OR type_of_extras
                                           THEN 1
                                           ELSE 0
                                       END) / 6.0), 2) AS economy_rate
FROM ipl_ballbyball2008_2024_updated
GROUP BY Bowler
HAVING SUM(CASE
               WHEN type_of_extras NOT IN ('wide', 'no-ball') OR type_of_extras IS NULL
               THEN 1
               ELSE 0
           END) >= 6
ORDER BY economy_rate ASC
LIMIT 5;
```

IPL

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

5.WHICH BATSMAN HAS FACED THE MOST BALLS?

MYSQL QUERY:-

```
SELECT Striker, COUNT(*) AS balls_faced
FROM ipl_ballbyball2008_2024_updated
WHERE type_of_extras IS NULL OR type_of_extras != 'wide'
GROUP BY striker
ORDER BY balls_faced DESC
LIMIT 1;
```

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

6. WHICH BOWLING TEAM HAS TAKEN PART IN THE MOST OVERS?

MYSQL QUERY:-

```
SELECT Bowling_team,
       ROUND(SUM(CASE
                     WHEN type_of_extras NOT IN ('wide', 'no-ball') OR type_of_extras IS NULL
                     THEN 1
                     ELSE 0
                 END) / 6.0, 2) AS total_overs
FROM ipl_ballbyball2008_2024_updated
GROUP BY Bowling_team
ORDER BY total_overs DESC
LIMIT 1;
```

IPL

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

7. WHAT IS THE AVERAGE NUMBER OF RUNS SCORED PER BALL BY EACH BATSMAN?

MYSQL QUERY:-

```
SELECT Striker,
       ROUND(SUM(runs_scored) / COUNT(*), 3) AS avg_runs_per_ball
FROM ipl_ballbyball2008_2024_updated
WHERE type_of_extras IS NULL OR type_of_extras != 'wide'
GROUP BY Striker
ORDER BY avg_runs_per_ball DESC;
```

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

8. WHICH TEAMS HAVE THE BEST WIN PERCENTAGE ACCORDING TO THE TEAM_PERFORMANCE TABLE?

MYSQL QUERY:-

```
SELECT team,
       COUNT(*) AS matches_played,
       SUM(CASE WHEN team = Match_Winner THEN 1 ELSE 0 END) AS matches_won,
       ROUND(SUM(CASE WHEN team = Match_Winner THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2) AS win_percentage
FROM (
    SELECT Match_ID, Match_Winner AS Match_Winner, TeamA AS team FROM (
        SELECT Match_ID, Match_Winner,
               SUBSTRING_INDEX(Teams, ' vs ', 1) AS TeamA,
               SUBSTRING_INDEX(Teams, ' vs ', -1) AS TeamB
        FROM team_performance_dataset_2008to2024
    ) AS t1
    UNION ALL
    SELECT Match_ID, Match_Winner AS Match_Winner, TeamB AS team FROM (
        SELECT Match_ID, Match_Winner,
               SUBSTRING_INDEX(Teams, ' vs ', 1) AS TeamA,
               SUBSTRING_INDEX(Teams, ' vs ', -1) AS TeamB
        FROM team_performance_dataset_2008to2024
    ) AS t2
) AS all_matches
GROUP BY team
ORDER BY win_percentage DESC;
```

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

9. LIST ALL PLAYERS WHO ARE ALL-ROUNDERS (BOTH BATTING AND BOWLING STYLE AVAILABLE).

MYSQL QUERY:-

```
SELECT Team_Name, Player_Name, Player_Role, Batting_Style, Bowling_Style
FROM players_info_2024
WHERE Player_Role = 'Allrounder'
  AND Batting_Style IS NOT NULL AND Batting_Style != ''
  AND Bowling_Style IS NOT NULL AND Bowling_Style != '';
```

IPL

INDIAN PREMIER LEAGUE

MYSQL PROBLEMS STATEMENT

10. WHAT IS THE AVERAGE NUMBER OF RUNS SCORED PER OVER BY EACH BATTING TEAM?

MYSQL QUERY:-

```
SELECT
  Batting_Team,
  ROUND(SUM(runs_Scored + extras) / (SUM(CASE
                                          WHEN type_of_extras NOT IN ('wide', 'no-ball') OR type_of_e
                                          THEN 1
                                          ELSE 0
                                        END) / 6.0), 2) AS avg_runs_per_over
FROM ipl_ballbyball2008_2024_updated
GROUP BY Batting_Team
ORDER BY avg_runs_per_over DESC;
```

INDIAN PREMIER LEAGUE

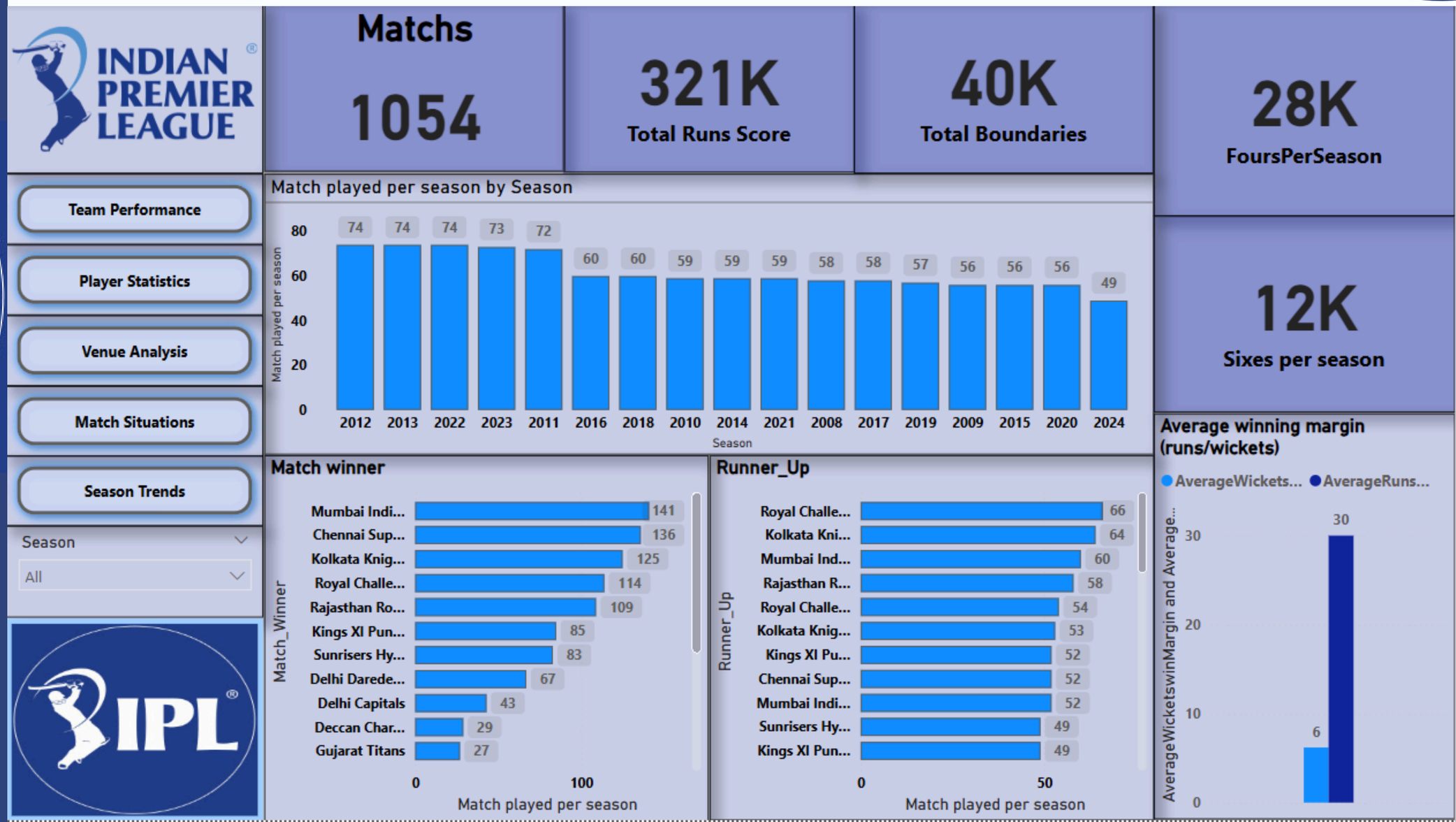
Dashboards (Power BI)

Create 6 interactive dashboards with at least 5 unique insights each. Add slicers/filters for season, team, venue, etc

Dashboard 1: Season Highlights
Matches played per season

- Winner & runner-up teams
- Total runs scored by season
- Number of sixes/fours per season
- Average winning margin (runs/wickets**)**

Dashboard 1: Season Highlights



INDIAN PREMIER LEAGUE

Dashboard 1: Insights

Here’s a crisp summary of the 5 insights from your Season Highlight dashboard:

1. 📉 **IPL 2024 had fewer matches (49)** — a major dip compared to other seasons.
2. ⚔️ **Winning margins are shrinking** — matches are getting tighter and more thrilling.
3. 🏆 **RCB tops as runner-up** — high consistency, but falls short in finals.
4. 🔥 **Peak in boundaries (2018–2020)** — big-hitting era with aggressive playstyles.
5. 🧱 **Stable wicket margins + more dot balls** — bowlers are getting smarter and more economical.

INDIAN PREMIER LEAGUE

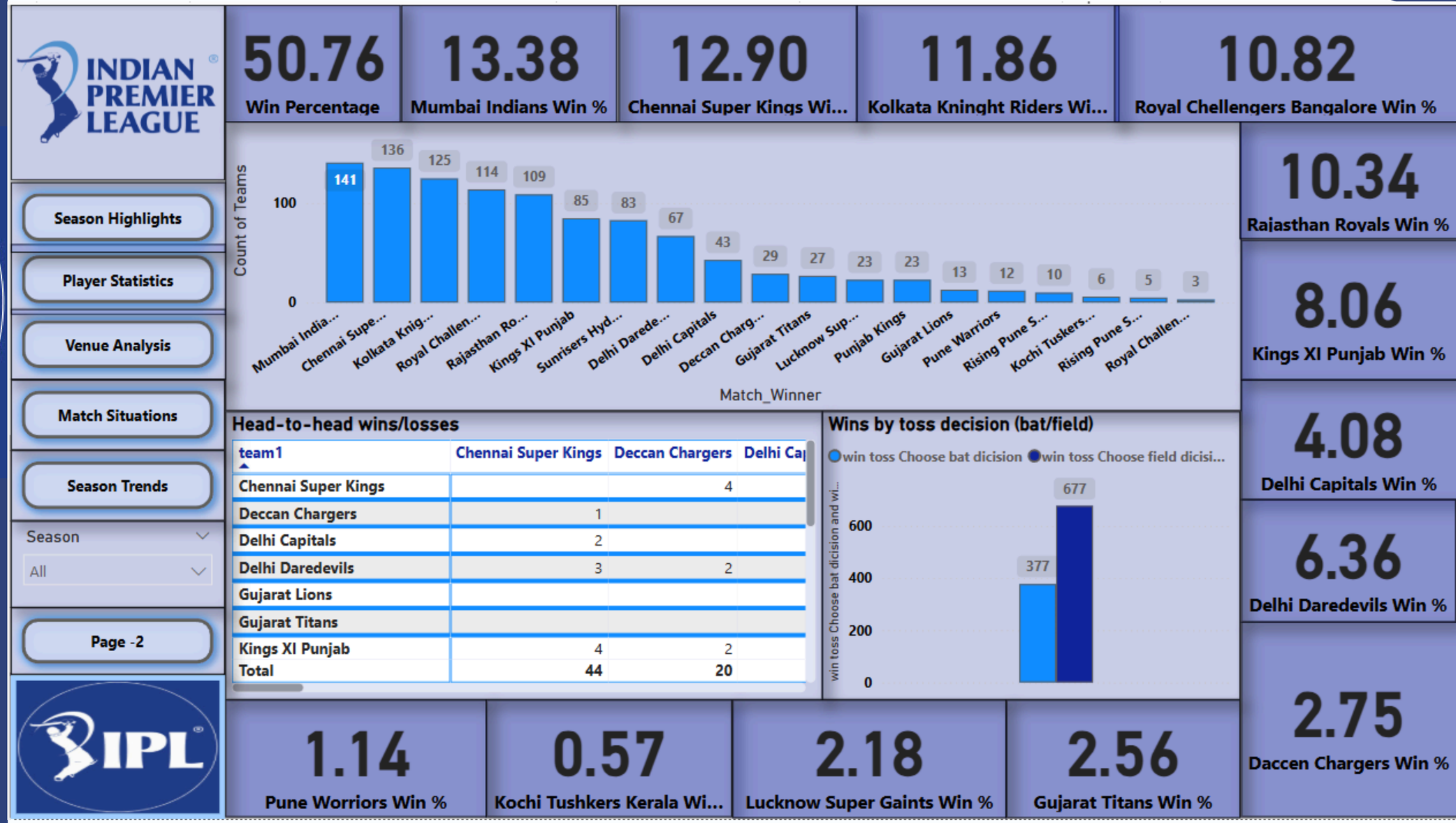
IPL®

Dashboard 2: Team Performance

- Team-wise win percentage
- Wins by toss decision (bat/field)
- Team's match result trends by season
- Head-to-head wins/losses
- Performance in finals

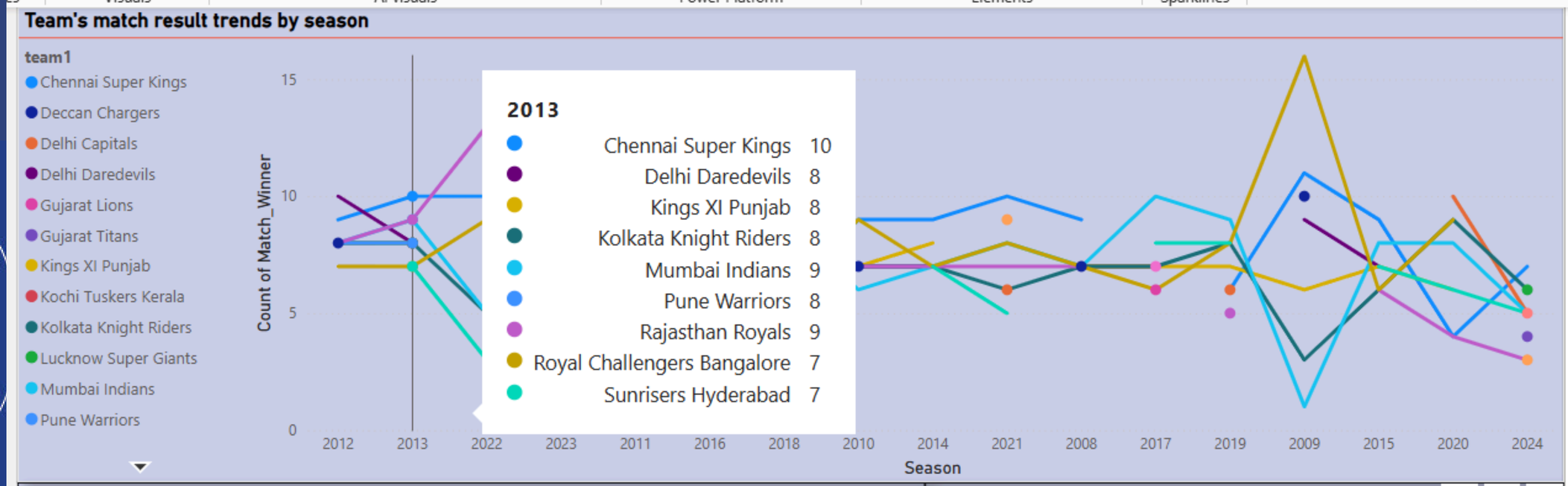
Dashboard 2: Team Performance

Page:-1



Dashboard 2: Team Performance

Page:-2



Performance in finals

Date	Is Final Match	team1	team2	Match_Winner
01 June 2008	Final	Chennai Super Kings	Rajasthan Royals	Rajasthan Royals
24 May 2009	Final	Royal Challengers Bangalore	Deccan Chargers	Deccan Chargers
25 April 2010	Final	Chennai Super Kings	Mumbai Indians	Chennai Super Kings
28 May 2011	Final	Chennai Super Kings	Royal Challengers Bangalore	Chennai Super Kings
27 May 2012	Final	Kolkata Knight Riders	Chennai Super Kings	Kolkata Knight Riders
26 May 2013	Final	Chennai Super Kings	Mumbai Indians	Mumbai Indians
01 June 2014	Final	Kolkata Knight Riders	Kings XI Punjab	Kolkata Knight Riders
24 May 2015	Final	Mumbai Indians	Chennai Super Kings	Mumbai Indians
29 May 2016	Final	Royal Challengers Bangalore	Sunrisers Hyderabad	Sunrisers Hyderabad
21 May 2017	Final	Mumbai Indians	Rising Pune Supergiant	Mumbai Indians
27 May 2018	Final	Sunrisers Hyderabad	Chennai Super Kings	Chennai Super Kings

Dashboard 2: Insight

🏆 Team-wise Win Percentage

1. **Mumbai Indians dominate** with the highest win percentage (50.76%), showcasing consistent top-tier performance.
2. **Chennai Super Kings & KKR follow,** forming a strong second tier in terms of win efficiency.

🌀 Toss Decision Outcomes

1. **Batting after toss yields better results** — 677 wins vs 377 when fielding first, indicating batting-first remains the preferred winning strategy.

📊 Match Result Trends by Season

1. **Match outcomes are more balanced** in recent seasons, with no team overwhelmingly dominant season-to-season.
2. **Old teams show legacy wins,** but newer franchises like Gujarat Titans and LSG are rising fast, shifting seasonal win dynamics.

⚔️ Head-to-Head Finals Battles

1. **CSK vs MI rivalry shines** — faced off thrice in finals with MI winning twice.
2. **KKR's edge over CSK in finals** — KKR clinched 2012 final convincingly.

🏆 Final Match Performance

1. Mumbai Indians most successful

INDIAN PREMIER LEAGUE

IPL®

Dashboard 3: Player Statistics

- Top run scorers and average per season
- Top wicket takers and economy rates
- Most sixes, most boundaries
- Player of the Match awards
- All-rounder performances

Dashboard 3: Player Statistics

INDIAN PREMIER LEAGUE

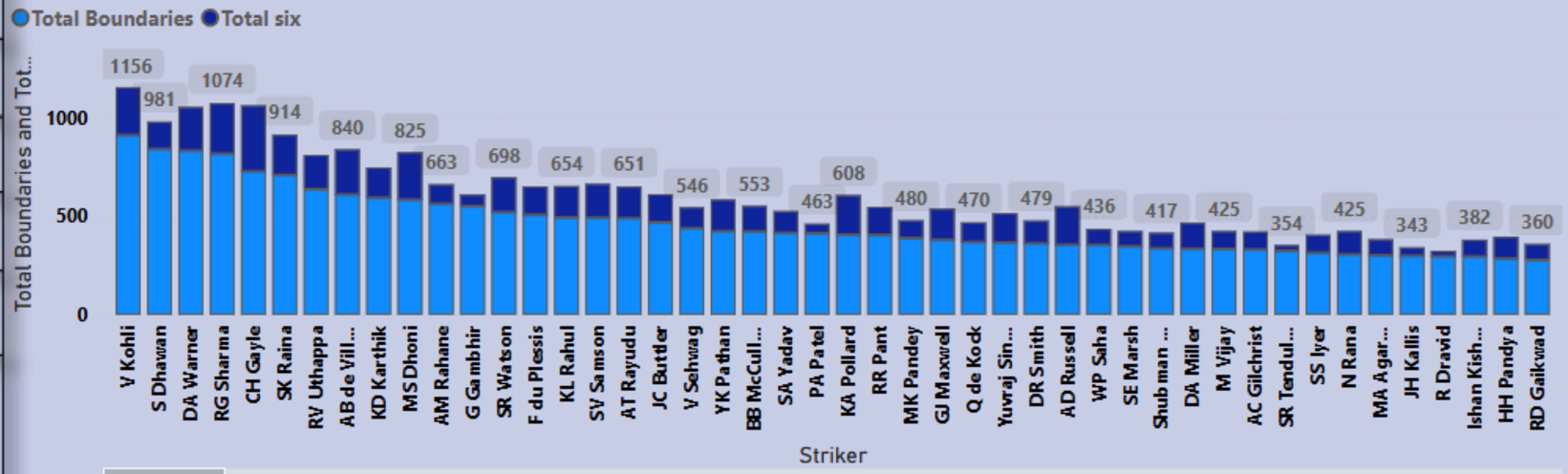
IPL®

INDIAN PREMIER LEAGUE

19.01K	40K	7.57	12K
Average Runs Per Season	Total Boundaries	Economy Rate	Total Wickets

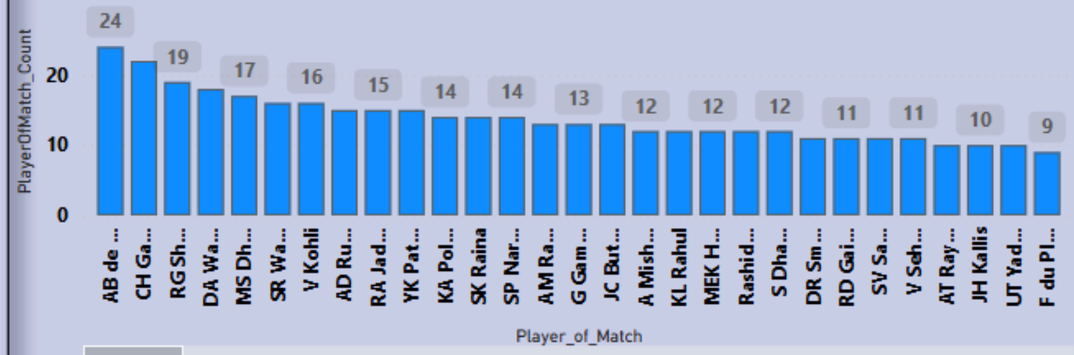
- Season Highlights
- Team Performance
- Venue Analysis
- Match Situations
- Season Trends

Most sixes, most boundaries



Striker	Sum of runs_scored
V Kohli	7302
RG Sharma	6194
S Dhawan	6151
DA Warner	6018
SK Raina	5536
MS Dhoni	4992
RV Uthappa	4758
AB de Villiers	4721
CH Gayle	4702
KD Karthik	4610
AM Rahane	4486
G Gambhir	4217
Total	304170

Player of the Match awards



Bowler	Sum of wicket_confirmation
DJ Bravo	200
SL Malinga	188
SP Narine	187
YS Chahal	186
PP Chawla	185
B Kumar	184
A Mishra	180
R Ashwin	176
Total	11974

Dashboard 3: Insights

Here are 5 crisp insights from the dashboard:

🏏 Batting Stats

1. **Virat Kohli leads in runs (7302)** — unmatched consistency across seasons.
2. **Average runs per season: 19.01K** — reflecting high scoring formats.

🎯 Bowling Stats

1. **DJ Bravo tops wickets (200), Malinga close (188)** — true impact bowlers.
2. **Economy rate sits at 7.57** — tight bowling across tournaments**.**

💥 Power Hitters & Awards

1. **Chris Gayle dominates sixes (1174), Kohli leads boundaries (1156)** — elite strikers.
2. **AB de Villiers wins most Player of the Match awards (24)** — a match-winner through and through.

🔄 All-Rounder Impact

1. Bravo, Pollard, Russell stand out — major contributions with bat and ball.

INDIAN PREMIER LEAGUE

IPL®

Dashboard 4: Venue Analysis

- Top venues by matches played
- Venue-wise average scores
- Team-wise win % by venue
- Toss outcome and result pattern per venue
- Home vs away performance

Dashboard 4: Venue Analysis

Page:-1

INDIAN PREMIER LEAGUE

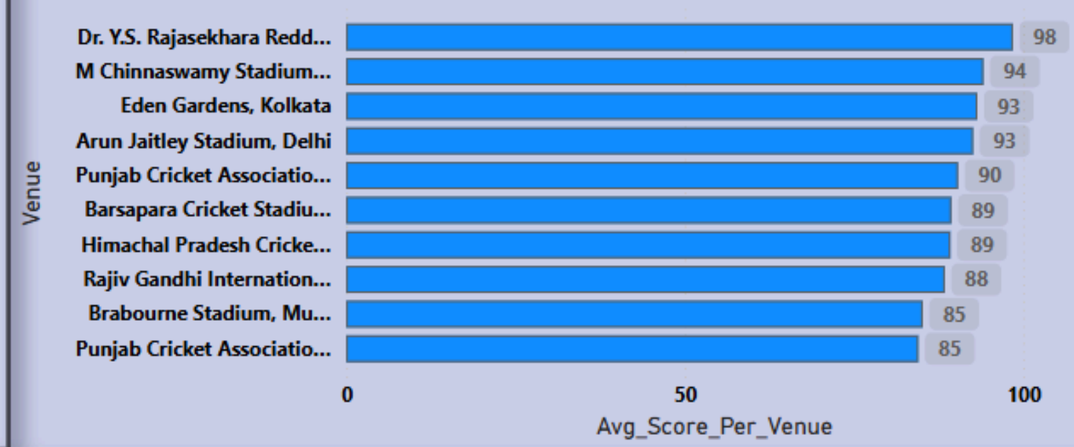
Season Highlights | Team Performance | Match Situations | Season Trends

Player Statistics | Page -2

Top venues by matches played

Venue	Count of Match_ID
Eden Gardens	77
Wankhede Stadium	72
M Chinnaswamy Stadium	62
Feroz Shah Kotla	59
Rajiv Gandhi International Stadium, Uppal	48
MA Chidambaram Stadium, Chepauk	47
Sawai Mansingh Stadium	47
Dubai International Cricket Stadium	43
Wankhede Stadium, Mumbai	42
Total	1054

Venue-wise average scoresby Venue

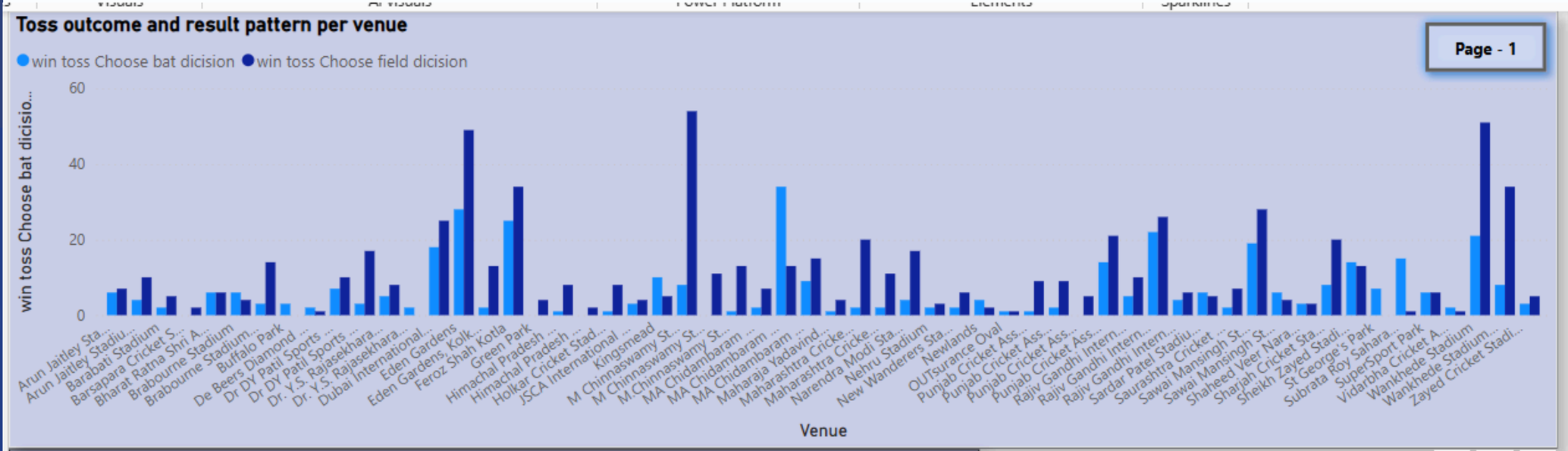


Team-wise win % by venue

Venue	Chennai Super Kings Win %	Mumbai Indians Win %	Delhi Capitals Win %	Kolkata Kninght Riders Win %	Royal Chellenger
Arun Jaitley Stadium	7.69	7.69	23.08		
Arun Jaitley Stadium, Delhi	14.29	21.43	28.57		
Barabati Stadium				42.86	
Barsapara Cricket Stadium, Guwahati					
Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow			8.33		
Brabourne Stadium	10.00	50.00			
Brabourne Stadium, Mumbai		11.76	23.53		
Buffalo Park	33.33	33.33			
De Beers Diamond Oval	33.33				
Total	12.90	13.38	4.08	11.86	

Dashboard 4: Venue Analysis

Page:-2



Home vs away performance

Team	First Result	First Type	First Venue
Rising Pune Supergiant	Loss	Home	Maharashtra Cricket Association Stadium
Royal Challengers Bengaluru	Loss	Home	M Chinnaswamy Stadium, Bengaluru
Kochi Tuskers Kerala	Loss	Home	Holkar Cricket Stadium
Gujarat Lions	Loss	Home	Feroz Shah Kotla
Royal Challengers Bengaluru	Loss	Away	Eden Gardens, Kolkata
Gujarat Lions	Loss	Away	Eden Gardens
Rising Pune Supergiant	Loss	Away	Eden Gardens
Rising Pune Supergiants	Loss	Away	Eden Gardens
Rising Pune Supergiants	Loss	Home	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium
Kochi Tuskers Kerala	Loss	Away	Dr DY Patil Sports Academy
Pune Warriors	Loss	Home	Dr DY Patil Sports Academy
Deccan Chargers	Loss	Away	Brabourne Stadium
Total	**Loss**	**Away**	**Arun Jaitley Stadium**

Dashboard 4: Insights

🏟️ Top Venues by Matches Played

- **Eden Gardens (77 matches)** is the most utilized IPL venue—signaling its historical
- importance and crowd-pulling legacy

📊 Venue-wise Average Scores

- **ACA-VDCA Stadium (avg: 98)** has the highest scoring trend, possibly reflecting batting-friendly conditions or shorter boundaries.
- **M Chinnaswamy Stadium (avg: 94)** also supports high scoring —aligns with its reputation for fast outfields and small boundaries

💪 Team-wise Win % by Venue

- **Kolkata Knight Riders dominate Arun Jaitley Stadium** with a win rate of **42.86%,** making it a tactical stronghold.

🎯 Toss Outcome & Result Pattern

- **Fielding first is preferred at** venues like Eden Gardens and Wankhede—indicating dew impact or chasing advantage.

🎯🏠 Home vs Away Performance

- **All listed teams started with a loss**, both home and away—highlighting that initial venue familiarity doesn't guarantee a win.

INDIAN PREMIER LEAGUE

IPL®

Dashboard 5: Match Situations

- Matches won by chasing vs defending
- Super Over matches overview
- Performance in Powerplay, Middle, Death overs
- Fall of wickets timing
- Pressure performance: last 5 overs

Dashboard 5: Match Situations

INDIAN PREMIER LEAGUE

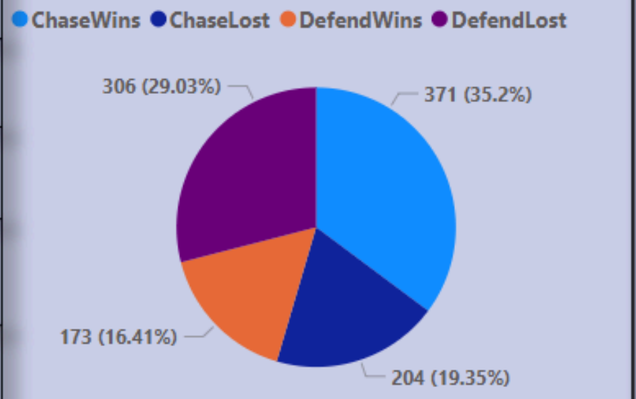
IPL®

INDIAN PREMIER LEAGUE®

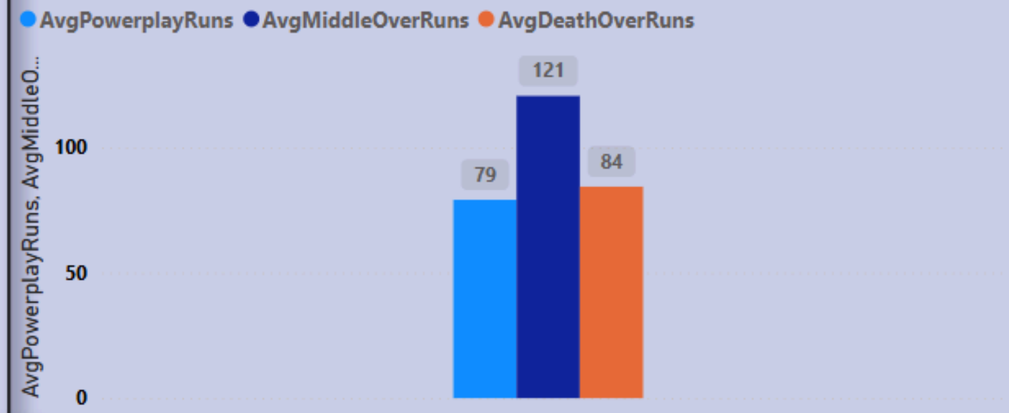
80K	122K	85K	10
PowerplayRuns	MiddleOverRuns	DeathOverRuns	Super_Over_Matches

- Season Highlights
- Team Performance
- Player Statistics
- Venue Analysis
- Season Trends

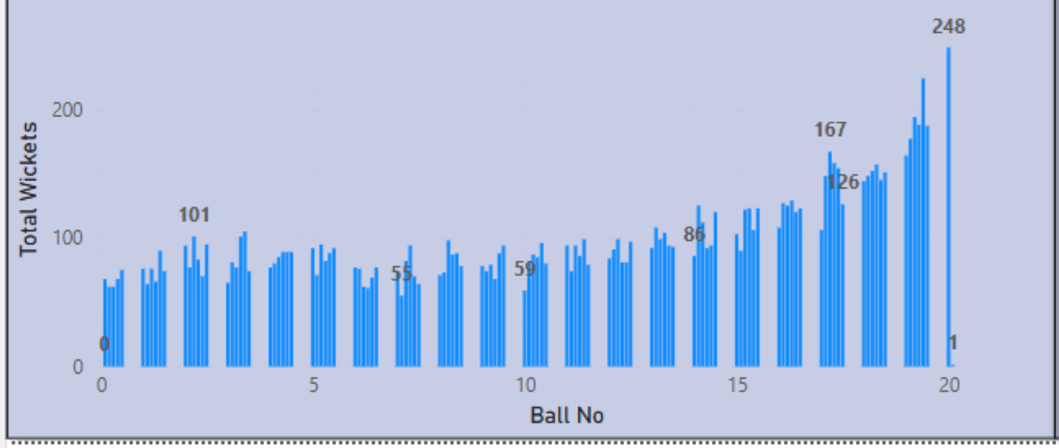
Matches won by chasing vs defending



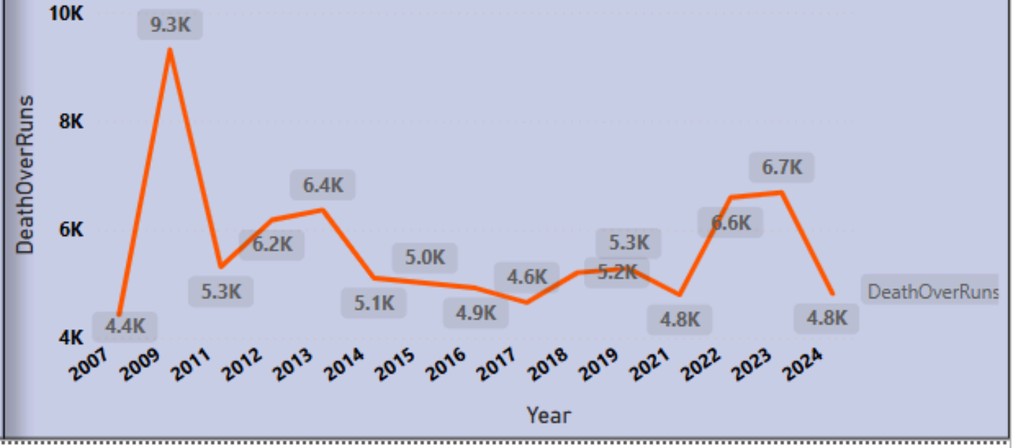
Performance in Powerplay, Middle, Death overs



Fall of wickets timing



Pressure performance: last 5 overs



INDIAN PREMIER LEAGUE

Dashboard 5: Insights

Here are concise, data-driven insights from the dashboard:

🏁 Matches Won: Chasing vs Defending

- **Chasing teams win more often** (371 vs 306)—confirming that **chasing has a strategic edge** in IPL.

🔥 Super Over Matches

- **Only 10 Super Over games** in 17 seasons—highlighting their rarity despite IPL's competitiveness.

📊 Overs Phase Performance

- **Middle overs yield highest avg. runs (121)**—teams capitalize here to set or chase targets effectively.
- Powerplay (79) and Death overs (84) reflect aggressive but volatile scoring.

⏱️ Fall of Wickets Timing

- **Key wickets fall around balls 6, 10, 15 & 20—** suggesting strategic turning points early, mid and at over-ends..

💣 Pressure Performance: Last 5 Overs

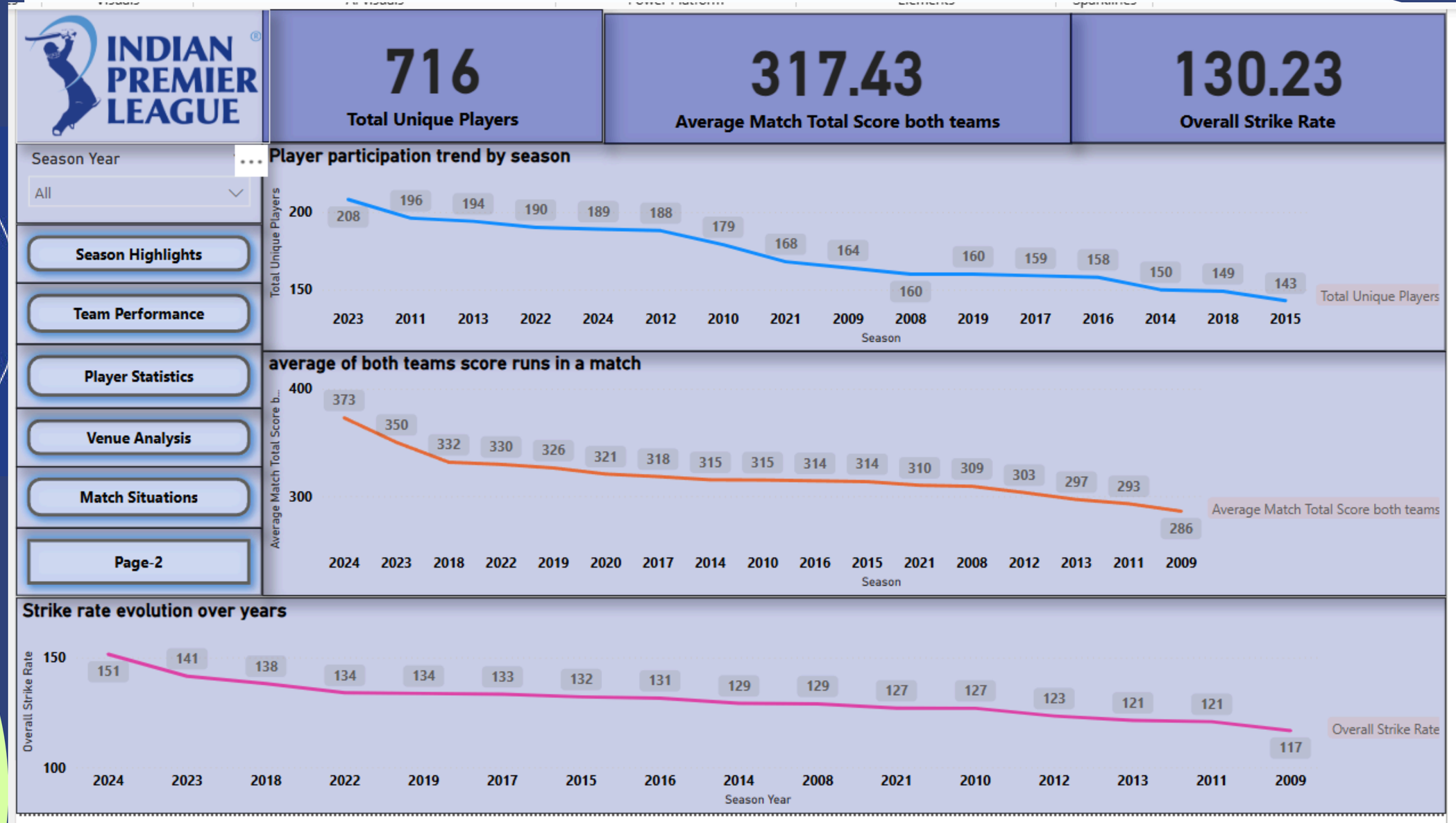
- **Peak finishing years:** 2009 (9.3K runs), 2023 (6.7K)—reflecting evolving death-over dominance across eras.

INDIAN PREMIER LEAGUE

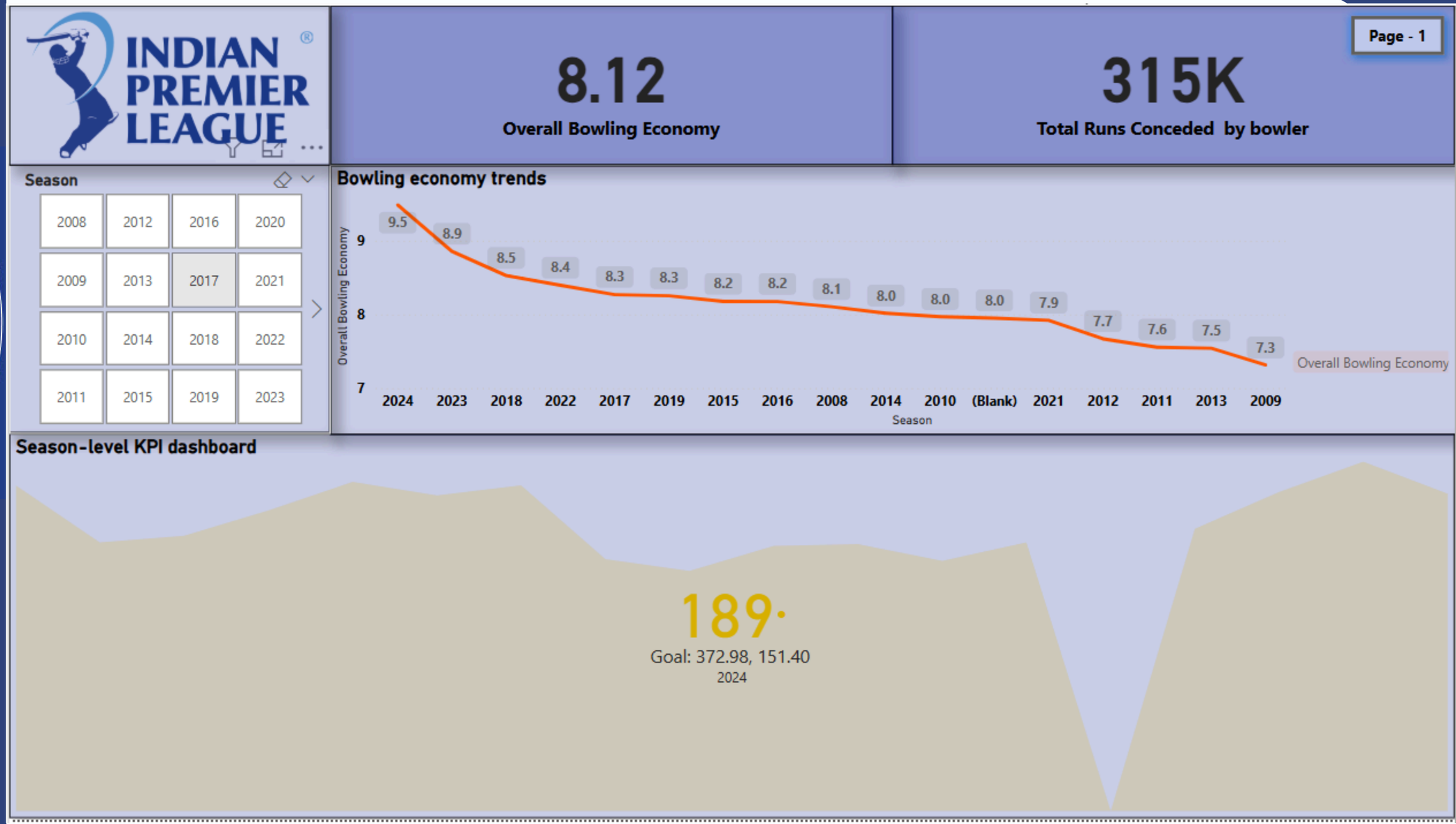
Dashboard 6: Season-wise Trends

- Player participation trend by season
- Match totals vs overs comparison
- Strike rate evolution over years
- Bowling economy trends
- Season-level KPI dashboard

Dashboard 6: Season-wise Trends



Dashboard 6: Season-wise Trends



INDIAN PREMIER LEAGUE

Dashboard 6: Insights

Here’s a snappy set of insights from the dashboard:

👥 Player Participation Trend

- **Gradual decline in unique players** from 208 (2008) to 143 (2015), stabilizing around **160 in recent seasons**—signaling IPL’s focus on consistency and core squad building.

📈 Match Totals vs Overs

- **Average match totals have dropped from 373 (2008) to around 317—** reflecting tighter bowling strategies and evolving pitch dynamics.

🚀 Strike Rate Evolution

- **Strike rates have dipped** from **151 (2008) to 117 (2009)**, then settled around **130** **—**highlighting a shift from ultra-aggression to calculated pacing

🎯 Bowling Economy Trends

- **Significant improvement** from **9.5 (2009) to 8.0 (2024)**— shows bowlers adapting to batter-friendly IPL formats

📌 Season-Level KPI Dashboard

- **2024 KPI value: 189,** well below targets (372.98 / 151.4)— indicates a **performance gap** or possibly stricter evaluation benchmarks.

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Use Python (pandas, seaborn, matplotlib, plotly) to perform IPL insights and visual storytelling:

- **Combine and clean the relevant datasets**
- **Analyze run trends over the years**
- **Compare batting styles (anchor vs aggressive) via strike rate and boundary %**
- **Study bowling consistency (dot balls, economy rate, average)**
- **Visualize performance in different overs (Powerplay, Death)**
- **Compare venue behavior in high-scoring vs low-scoring matches**

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

IMPORT LIBRARYIES LIKE PANDAS ,GLOB, MATPLOYLIV,SEABORN

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

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

ADD FRIST DATASET

```
df1 = pd.read_csv("IPL_BallByBall2008_2024_updated.csv")
df1
```

	Match id	Date	Season	Batting team	Bowling team	Innings No	Ball No	Bowler	Striker	Non Striker	runs_scored	extras	type of extras	score	score/wicket	wicket_confirma
0	335982	2008-04-18	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.1	P Kumar	SC Ganguly	BB McCullum	0	1	legbyes	1	1/0	
1	335982	2008-04-18	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.2	P Kumar	BB McCullum	SC Ganguly	0	0	NaN	1	1/0	
2	335982	2008-04-18	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.2	P Kumar	BB McCullum	SC Ganguly	0	1	wides	2	2/0	
3	335982	2008-04-18	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.3	P Kumar	BB McCullum	SC Ganguly	0	0	NaN	2	2/0	
4	335982	2008-04-18	2007/08	Kolkata Knight Riders	Royal Challengers Bangalore	1	0.4	P Kumar	BB McCullum	SC Ganguly	0	0	NaN	2	2/0	

In this data set some column have NaN values.

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

IN THE DATASET SOME COLUMN HAVE NaN VALUE FILL WITH 0, UN[KNOWN , ‘NO FIELDER INVOLVE , NOT OUT.

```
df1.fillna({'type of extras': 0, 'wicket_type': 'unknown','fielders_involved':'No fielder involve', 'Player Out': 'not out'})
```

type of extras	score	score/wicket	wicket_confirmation	wicket_type	fielders_involved	Player Out
legbyes	1	1/0	0	unknown	No fielder involve	not out
0	1	1/0	0	unknown	No fielder involve	not out
wides	2	2/0	0	unknown	No fielder involve	not out
0	2	2/0	0	unknown	No fielder involve	not out
0	2	2/0	0	unknown	No fielder involve	not out
...	...	...	...	...	...	...
0	156	156/3	0	unknown	No fielder involve	not out
0	160	160/3	0	unknown	No fielder involve	not out

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

Upload Second Dataset

```
df2 = pd.read_csv("team_performance_dataset_2008to2024.csv")
df2
```

	Match_ID	Date	Teams	Venue	Toss_Winner	Toss_Decision	Match_Winner	Win_Type	Win_Margin	First_Innings_Score	Second_Innings_Score	P
0	335982	2008-04-18	Royal Challengers Bangalore vs Kolkata Knight ...	M Chinnaswamy Stadium	Royal Challengers Bangalore	field	Kolkata Knight Riders	runs	140.0	222	82.0	
1	335983	2008-04-19	Kings XI Punjab vs Chennai Super Kings	Punjab Cricket Association Stadium, Mohali	Chennai Super Kings	bat	Chennai Super Kings	runs	33.0	240	207.0	
2	335984	2008-04-19	Delhi Daredevils vs Rajasthan Royals	Feroz Shah Kotla	Rajasthan Royals	bat	Delhi Daredevils	wickets	9.0	129	132.0	
3	335985	2008-04-20	Mumbai Indians vs Royal Challengers Bangalore	Wankhede Stadium	Mumbai Indians	bat	Royal Challengers Bangalore	wickets	5.0	165	166.0	
4	335986	2008-04-20	Kolkata Knight Riders vs Deccan	Eden Gardens	Deccan Chargers	bat	Kolkata Knight Riders	wickets	5.0	110	112.0	

In this dataset no NaN values. t

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

Change the column name to combine both dataset.

```
df2 = df2.rename(columns={"Match_ID": "Match id"})
```

```
print(df2.columns)
```

```
Index(['Match id', 'Date', 'Teams', 'Venue', 'Toss_Winner', 'Toss_Decision',
       'Match_Winner', 'Win_Type', 'Win_Margin', 'First_Innings_Score',
       'Second_Innings_Score', 'Player_of_Match', 'Umpire', 'Umpire1',
       'Umpire2', 'Powerplay_Scores', 'Middle_Overs_Scores',
       'Death_Overs_Scores'],
      dtype='object')
```

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

To join the both dataset based upon ‘Match id’ column is common

```
dfx = pd.merge(df1, df2, on="Match id", how="outer")
print(dfx)
```

```
        Match id      Date_x    Season            Batting team  \
0         335982  2008-04-18   2007/08  Kolkata Knight Riders
1         335982  2008-04-18   2007/08  Kolkata Knight Riders
2         335982  2008-04-18   2007/08  Kolkata Knight Riders
3         335982  2008-04-18   2007/08  Kolkata Knight Riders
4         335982  2008-04-18   2007/08  Kolkata Knight Riders
...          ...         ...       ...                    ...
255754   1426287  2024-05-01      2024           Punjab Kings
255755   1426287  2024-05-01      2024           Punjab Kings
255756   1426287  2024-05-01      2024           Punjab Kings
255757   1426287  2024-05-01      2024           Punjab Kings
255758   1426287  2024-05-01      2024           Punjab Kings

                      Bowling team  Innings No  Ball No      Bowler  \
0       Royal Challengers Bangalore           1      0.1     P Kumar
1       Royal Challengers Bangalore           1      0.2     P Kumar
2       Royal Challengers Bangalore           1      0.2     P Kumar
3       Royal Challengers Bangalore           1      0.3     P Kumar
```

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

Check For Null values.

```
dfx.isnull()
```

	Match id	Date_x	Season	Batting team	Bowling team	Innings No	Ball No	Bowler	Striker	Non Striker	...	Second_Innings_Score	Player_of_Match	Umpire	Umpire1	Umpire2
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
255754	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
255755	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
255756	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
255757	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
255758	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False

Merge Dataset called dfx has no NaN values.

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

Check For Duplicates

```
dfx.duplicated()
```

```
0         False
1         False
2         False
3         False
4         False
          ...
255754    False
255755    False
255756    False
255757    False
255758    False
Length: 255759, dtype: bool
```

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

dfx dataset information

```
dfx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255759 entries, 0 to 255758
Data columns (total 36 columns):
```

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Combine and clean the relevant datasets

Extract Years

```
dfx['Date_y'] = pd.to_datetime(dfx['Date_y'], errors='coerce').dt.year
```

```
print(dfx.Date_y)
```

```
0         2008
1         2008
2         2008
3         2008
4         2008
          ...
255754    2024
255755    2024
255756    2024
255757    2024
255758    2024
Name: Date_y, Length: 255759, dtype: int32
```

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

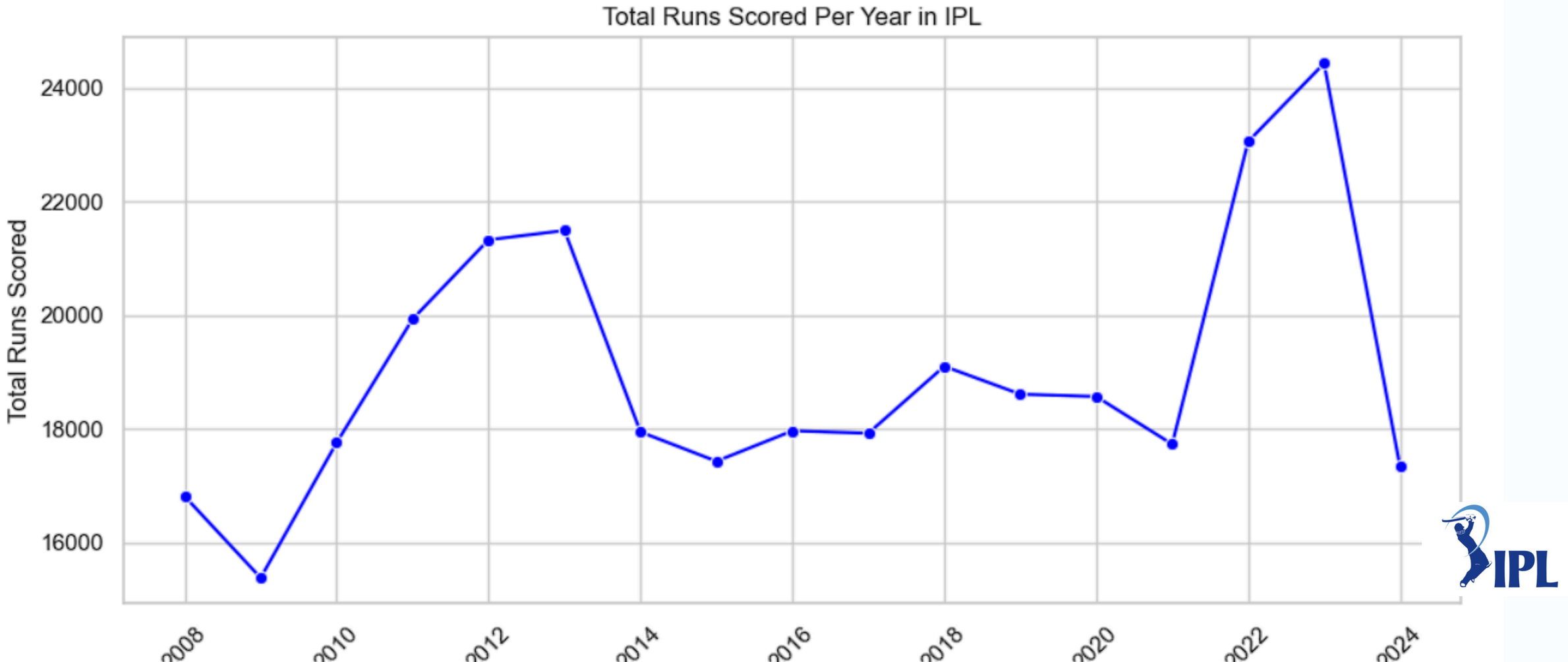
Analyze run trends over the years

```
#Analyze Run Trends Over the Years
runs_per_year = dfx.groupby('Date_y')['runs_scored'].sum().reset_index()
```

```
runs_per_year
```

	Date_y	runs_scored
0	2008	16809
1	2009	15376
2	2010	17754
3	2011	19928
4	2012	21323
5	2013	21487
6	2014	17943
7	2015	17427
8	2016	17962
9	2017	17920
10	2018	19098

```
sns.set_theme(style="whitegrid")
plt.figure(figsize=(10, 5))
sns.lineplot(data=runs_per_year, x='Date_y', y='runs_scored', marker='o', color='blue')
plt.title('Total Runs Scored Per Year in IPL')
plt.xlabel('Year')
plt.ylabel('Total Runs Scored')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Python: Exploratory Data Analysis (EDA)

INDIAN PREMIER LEAGUE

IPL®

Compare batting styles (anchor vs aggressiv) via strike rate an

```
# Convert to numeric and drop missing values
dfx['runs_scored'] = pd.to_numeric(dfx['runs_scored'], errors='coerce')
dfx.dropna(subset=['Striker', 'runs_scored'], inplace=True)
```

```
player_stats = dfx.groupby('Striker').agg(
    total_runs=('runs_scored', 'sum'),
    balls_faced=('Ball No', 'count'),
    fours=('runs_scored', lambda x: (x == 4).sum()),
    sixes=('runs_scored', lambda x: (x == 6).sum())
).reset_index()

# Filter out players with low sample size
player_stats = player_stats[player_stats['balls_faced'] >= 100]

# Calculate strike rate and boundary %
player_stats['strike_rate'] = (player_stats['total_runs'] / player_stats['balls_faced']) * 100
player_stats['boundary_runs'] = player_stats['fours'] * 4 + player_stats['sixes'] * 6
player_stats['boundary_percent'] = (player_stats['boundary_runs'] / player_stats['total_runs']) * 100
```

```
# Define percentiles or thresholds
sr_threshold = player_stats['strike_rate'].quantile(0.75)
bp_threshold = player_stats['boundary_percent'].quantile(0.75)

def classify(row):
    if row['strike_rate'] >= sr_threshold and row['boundary_percent'] >= bp_threshold:
        return 'Aggressive'
    elif row['strike_rate'] < sr_threshold and row['boundary_percent'] < bp_threshold:
        return 'Anchor'
    else:
        return 'Balanced'
```

player_stats

	Striker	total_runs	balls_faced	fours	sixes	strike_rate	boundary_runs	boundary_percent	style
0	A Ashish Reddy	280	196	16	15	142.857143	154	55.000000	Balanced
1	A Badoni	536	441	36	21	121.541950	270	50.373134	Anchor
8	A Manohar	231	181	21	10	127.624309	144	62.337662	Anchor
9	A Mishra	381	440	31	5	86.590909	154	40.419948	Anchor
16	A Symonds	974	781	74	41	124.711908	542	55.646817	Anchor
...	...	...	...	...	...	...	...	...	...
654	YBK Jaiswal	1421	975	175	58	145.743590	1048	73.750880	Aggressive
655	YK Pathan	3222	2334	263	161	138.046272	2018	62.631906	Balanced
658	YV Takawale	192	183	26	3	104.918033	122	63.541667	Anchor
663	Yuvraj Singh	2754	2207	218	149	124.784776	1766	64.124909	Balanced
664	Z Khan	117	141	11	2	82.978723	56	47.863248	Anchor

294 rows × 9 columns

INDIAN PREMIER LEAGUE

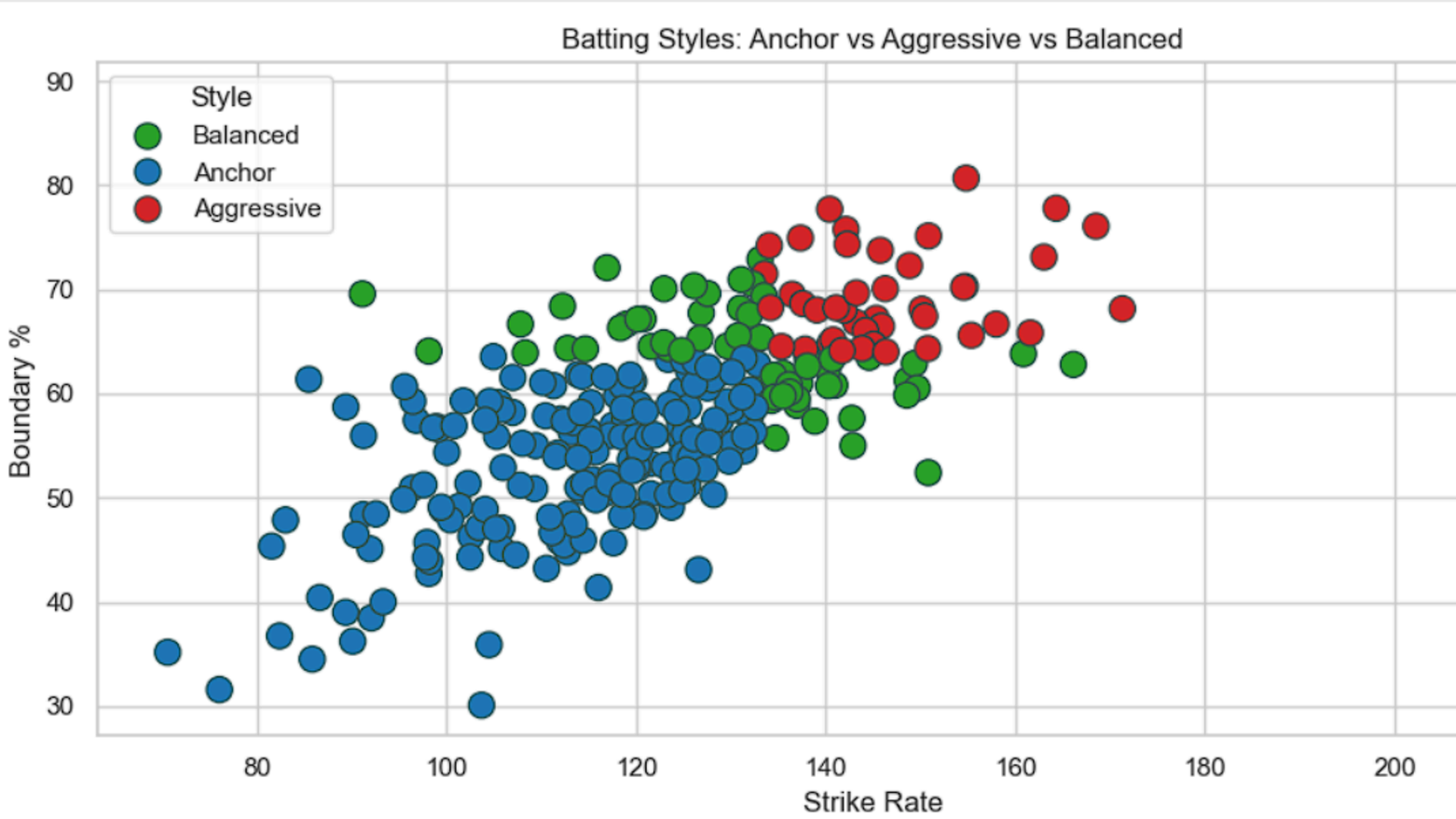
IPL®

graph

```
sns.set(style="whitegrid")

plt.figure(figsize=(10, 5))
sns.scatterplot(data=player_stats,
                x='strike_rate',
                y='boundary_percent',
                hue='style',
                palette={'Anchor': '#1f77b4', 'Aggressive': '#d62728', 'Balanced': '#2ca02c'},
                edgecolor='darkslategray',
                s=120)

plt.title('Batting Styles: Anchor vs Aggressive vs Balanced')
plt.xlabel('Strike Rate')
plt.ylabel('Boundary %')
plt.legend(title='Style')
plt.tight_layout()
plt.show()
```



Python: Exploratory Data Analysis (EDA)

Study bowling consistency (dot balls, economy rate, average)

INDIAN PREMIER LEAGUE

IPL®

```
import numpy as np
```

```
 #Study Bowling Consistency (Dot Balls, Economy Rate, Average)
# Calculate bowling stats
bowler_stats = dfx.groupby('Bowler').agg(
    balls_bowled=('Ball No', 'count'),
    runs_conceded=('runs_scored', 'sum'), # total_runs include extras here, which is standard for econom
    wickets_taken=('wicket_type', lambda x: (x != 'not_out').sum()),
    dot_balls=('runs_scored', lambda x: (x == 0).sum()) # runs_off_bat is 0 for dot ball
).reset_index()
```

```
# Filter for bowlers with enough data
bowler_stats = bowler_stats[bowler_stats['balls_bowled'] >= 60].copy()

# Economy Rate
bowler_stats['overs_bowled'] = bowler_stats['balls_bowled'] / 6
bowler_stats['economy_rate'] = bowler_stats['runs_conceded'] / bowler_stats['overs_bowled']

# Bowling Average
bowler_stats['bowling_average'] = np.where(
    bowler_stats['wickets_taken'] > 0,
    bowler_stats['runs_conceded'] / bowler_stats['wickets_taken'],
    np.inf
)

# Dot Ball Percentage
bowler_stats['dot_ball_percentage'] = (bowler_stats['dot_balls'] / bowler_stats['balls_bowled']) * 100
```

INDIAN PREMIER LEAGUE

Study bowling consistency (dot balls, economy rate, average)

```
bowler_stats
```

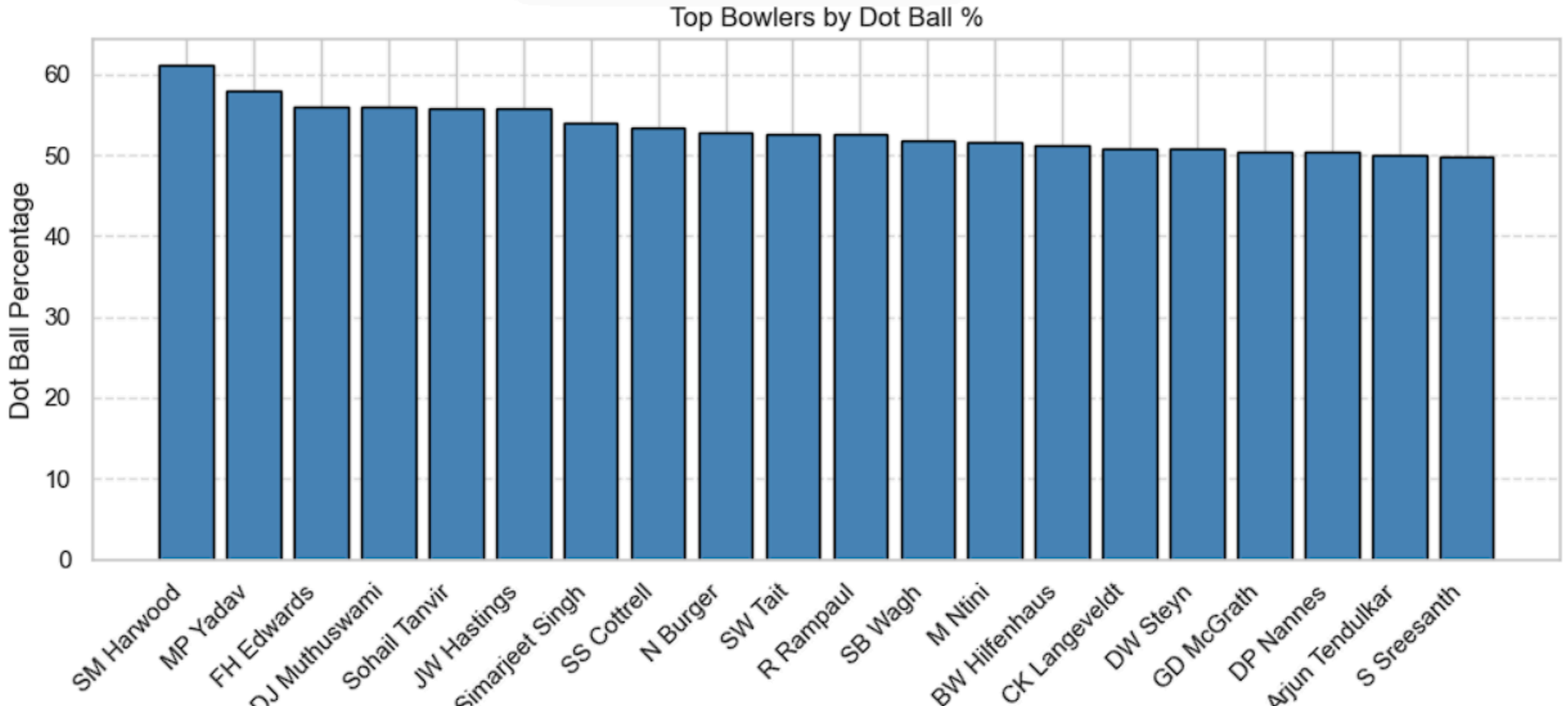
	Bowler	balls_bowled	runs_conceded	wickets_taken	dot_balls	overs_bowled	economy_rate	bowling_average	dot_ball_percentage
0	A Ashish Reddy	270	386	270	89	45.000000	8.577778	1.429630	32.962963
2	A Chandila	234	242	234	105	39.000000	6.205128	1.034188	44.871795
3	A Choudhary	108	137	108	49	18.000000	7.611111	1.268519	45.370370
5	A Flintoff	66	105	66	21	11.000000	9.545455	1.590909	31.818182
6	A Kumble	983	1027	983	410	163.833333	6.268566	1.044761	41.709054
...	...	...	...	...	...	...	...	...	...
517	YS Chahal	3479	4243	3479	1304	579.833333	7.317620	1.219603	37.482035
518	Yash Dayal	516	738	516	213	86.000000	8.581395	1.430233	41.279070
519	Yash Thakur	388	574	388	149	64.666667	8.876289	1.479381	38.402062
521	Yuvraj Singh	882	1064	882	282	147.000000	7.238095	1.206349	31.972789
522	Z Khan	2276	2691	2276	1005	379.333333	7.094025	1.182337	44.156415

372 rows × 9 columns

INDIAN PREMIER LEAGUE

```python
# Top Dot Ball Bowlers
top_dot = bowler_stats.sort_values(by='dot_ball_percentage', ascending=False).head(20)

# Create plot
plt.figure(figsize=(10, 5))
plt.bar(top_dot['Bowler'], top_dot['dot_ball_percentage'], color='steelblue', edgecolor='black')
plt.title('Top Bowlers by Dot Ball %')
plt.xlabel('Bowler')
plt.ylabel('Dot Ball Percentage')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

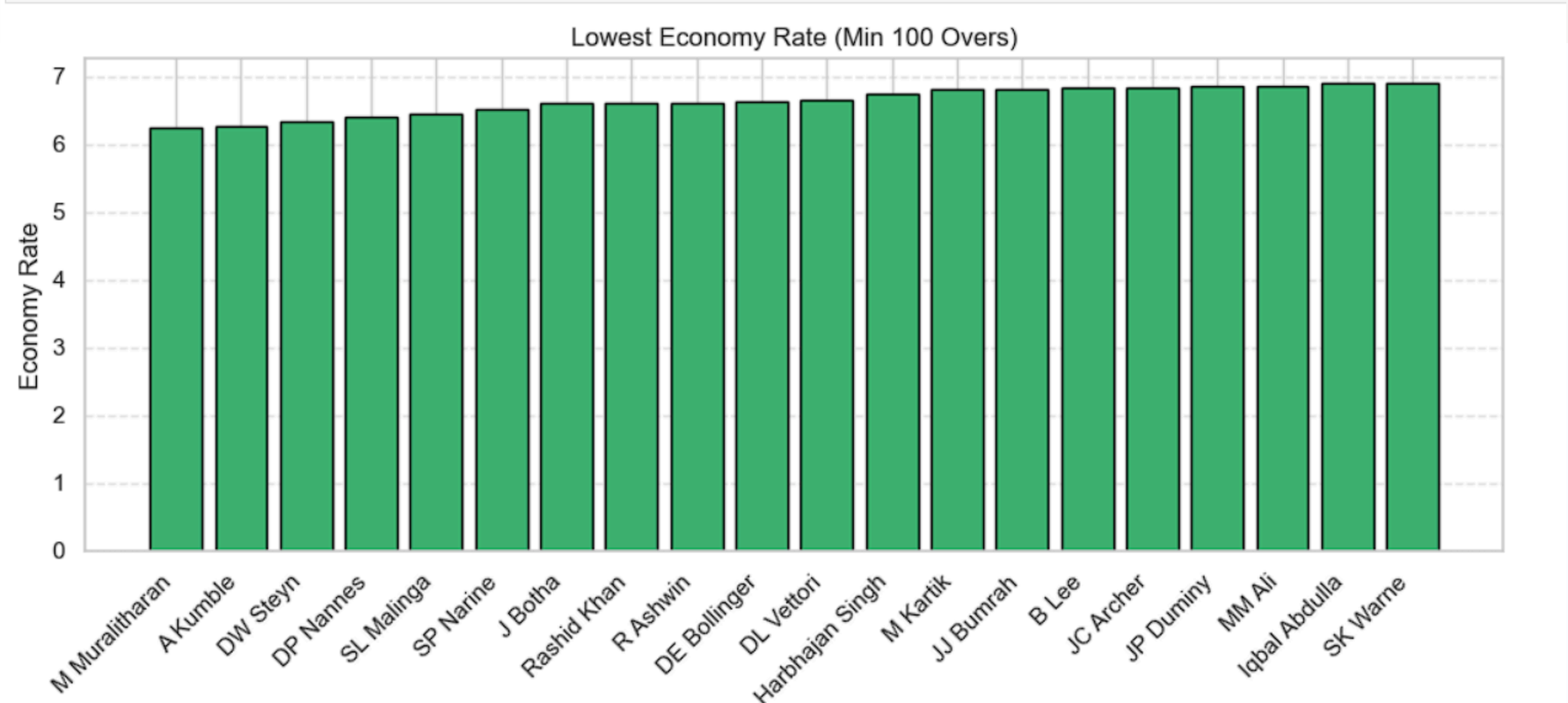


INDIAN PREMIER LEAGUE

IPL®

```python
# Best Economy
economy_filtered = bowler_stats[bowler_stats['overs_bowled'] >= 100]
top_economy = economy_filtered.sort_values(by='economy_rate').head(20)

plt.figure(figsize=(10, 5))
plt.bar(top_economy['Bowler'], top_economy['economy_rate'], color='mediumseagreen', edgecolor='black')
plt.title('Lowest Economy Rate (Min 100 Overs)')
plt.xlabel('Bowler')
plt.ylabel('Economy Rate')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

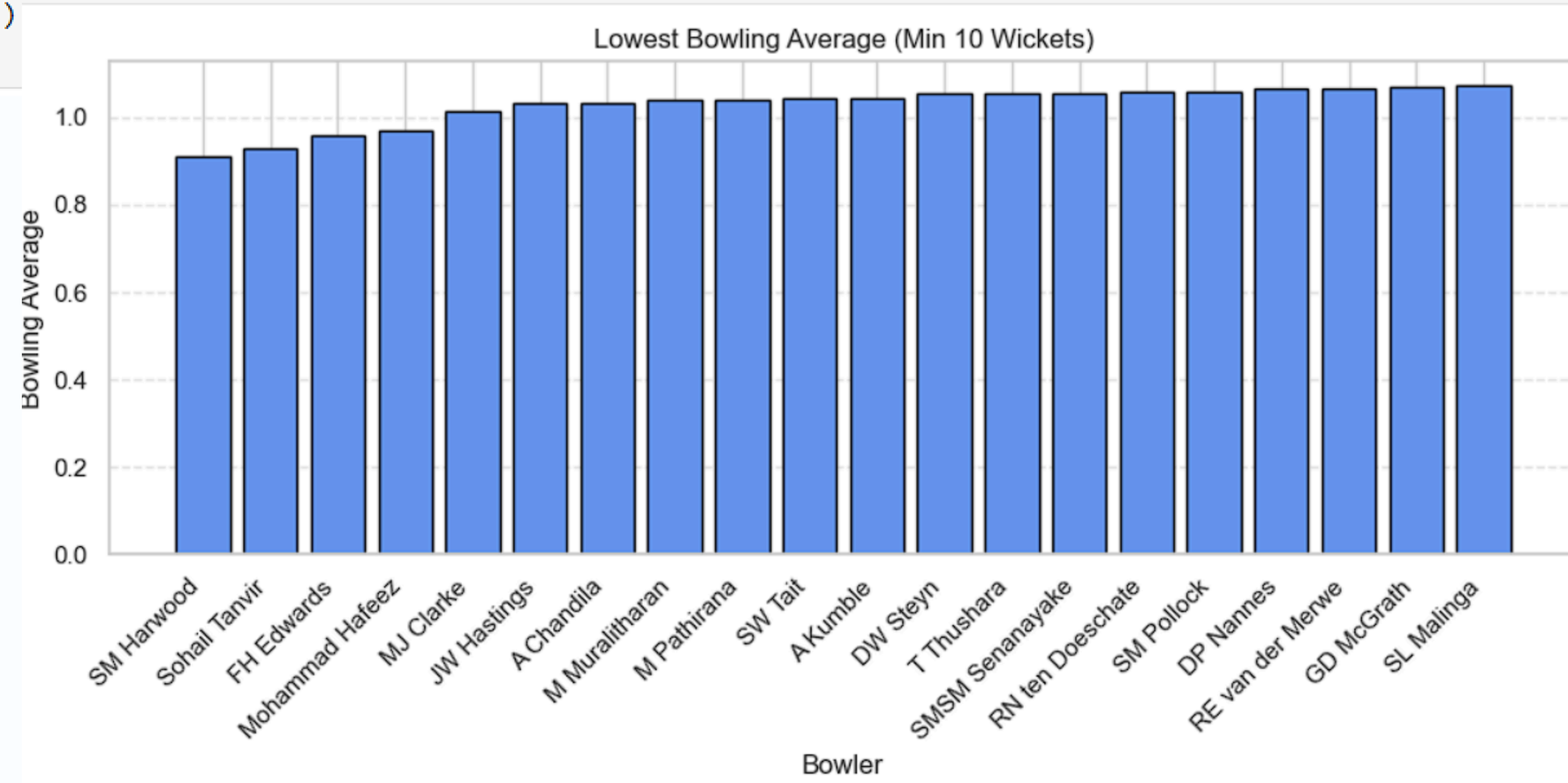


INDIAN PREMIER LEAGUE

IPL®

```python
# Best Bowling Average
avg_filtered = bowler_stats[bowler_stats['wickets_taken'] >= 10]
top_avg = avg_filtered.sort_values(by='bowling_average').head(20)

plt.figure(figsize=(10, 5))
plt.bar(top_avg['Bowler'], top_avg['bowling_average'], color='cornflowerblue', edgecolor='black')
plt.title('Lowest Bowling Average (Min 10 Wickets)')
plt.xlabel('Bowler')
plt.ylabel('Bowling Average')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Python: Exploratory Data Analysis (EDA)

Visualize performance in different overs

```
dfx['overs_bowled'] = dfx['Ball No'] // 6 + (dfx['Ball No'] % 6) / 10
```

```
def classify_phase(overs_bowled):
    if 1 <= overs_bowled <= 6:
        return 'Powerplay'
    elif 7 <= overs_bowled <= 16:
        return 'Middle Overs'
    elif 17 <= overs_bowled <= 20:
        return 'Death Overs'
    else:
        return 'Other'

dfx['over_phase'] = dfx['overs_bowled'].apply(classify_phase)
```

```
# Calculate runs and wickets per over phase per match
runs_wickets_per_phase = dfx.groupby(['Match id', 'over_phase']).agg(
    total_runs_phase=('runs_scored', 'sum'),
    wickets_lost_phase=('wicket_type', lambda x: (x != 'not_out').sum())
).reset_index()
```

runs_wickets_per_phase

	Match id	over_phase	total_runs_phase	wickets_lost_phase
0	335982	Other	70	74
1	335982	Powerplay	198	151
2	335983	Other	105	74
3	335983	Powerplay	325	174
4	335984	Other	91	73
...	...	...	...	...
2140	1426285	Powerplay	161	151
2141	1426286	Other	77	73
2142	1426286	Powerplay	193	175
2143	1426287	Other	96	71
2144	1426287	Powerplay	201	174

2145 rows × 4 columns

Python: Exploratory Data Analysis (EDA)

Visualize performance in different overs

```
# Calculate average runs and wickets per phase across all matches
avg_phase_performance = runs_wickets_per_phase.groupby('over_phase').agg(
    avg_runs=('total_runs_phase', 'mean'),
    avg_wickets=('wickets_lost_phase', 'mean')
).reset_index()
```

```
avg_phase_performance
```

	over_phase	avg_runs	avg_wickets
0	Other	85.764212	72.802423
1	Powerplay	215.215485	165.710821

```
# Ensure phases are in order for plotting
phase_order = ['Powerplay', 'Middle Overs', 'Death Overs']
avg_phase_performance['over_phase'] = pd.Categorical(avg_phase_performance['over_phase'], categories=phase_order, ordered=True)
avg_phase_performance = avg_phase_performance.sort_values('over_phase')
```

```
phase_order
```

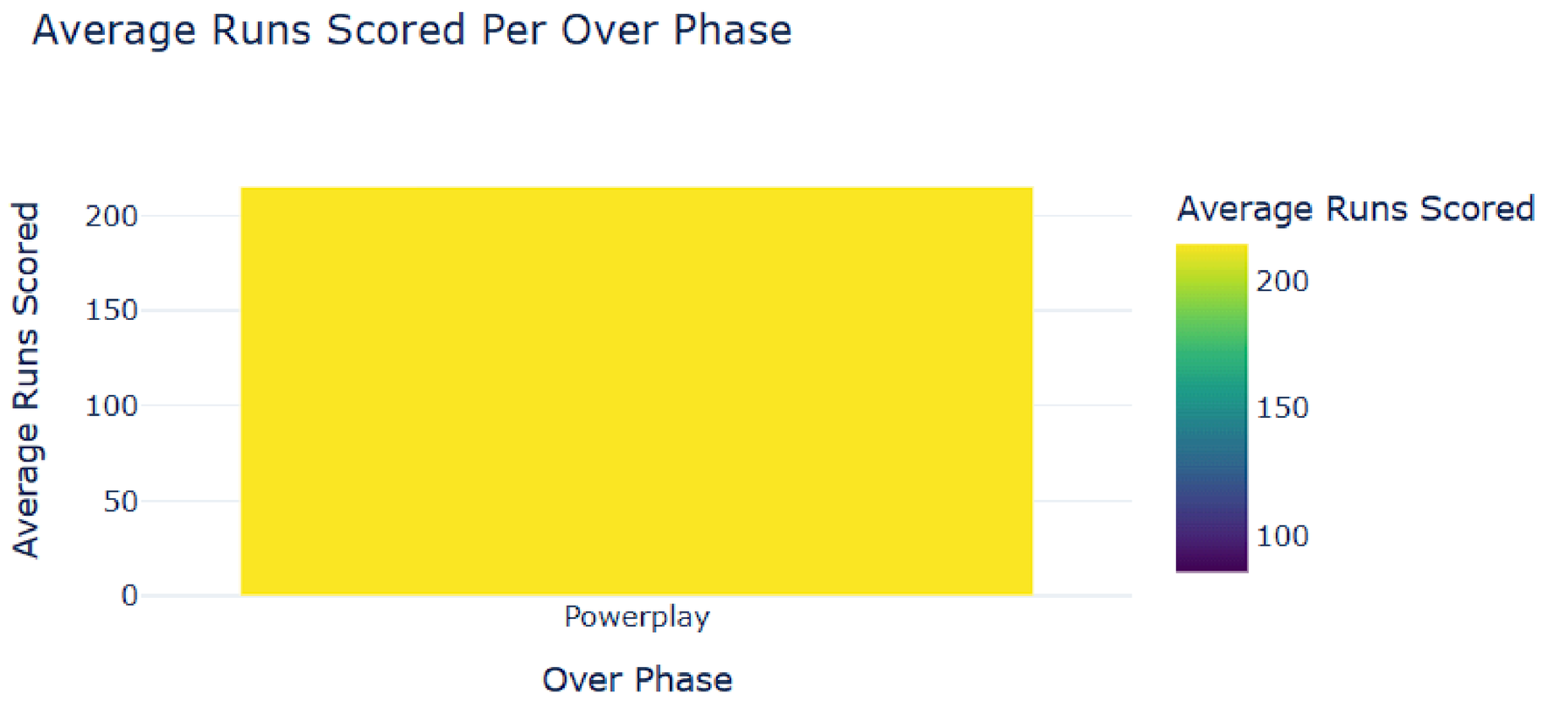
```
['Powerplay', 'Middle Overs', 'Death Overs']
```

INDIAN PREMIER LEAGUE

IPL®

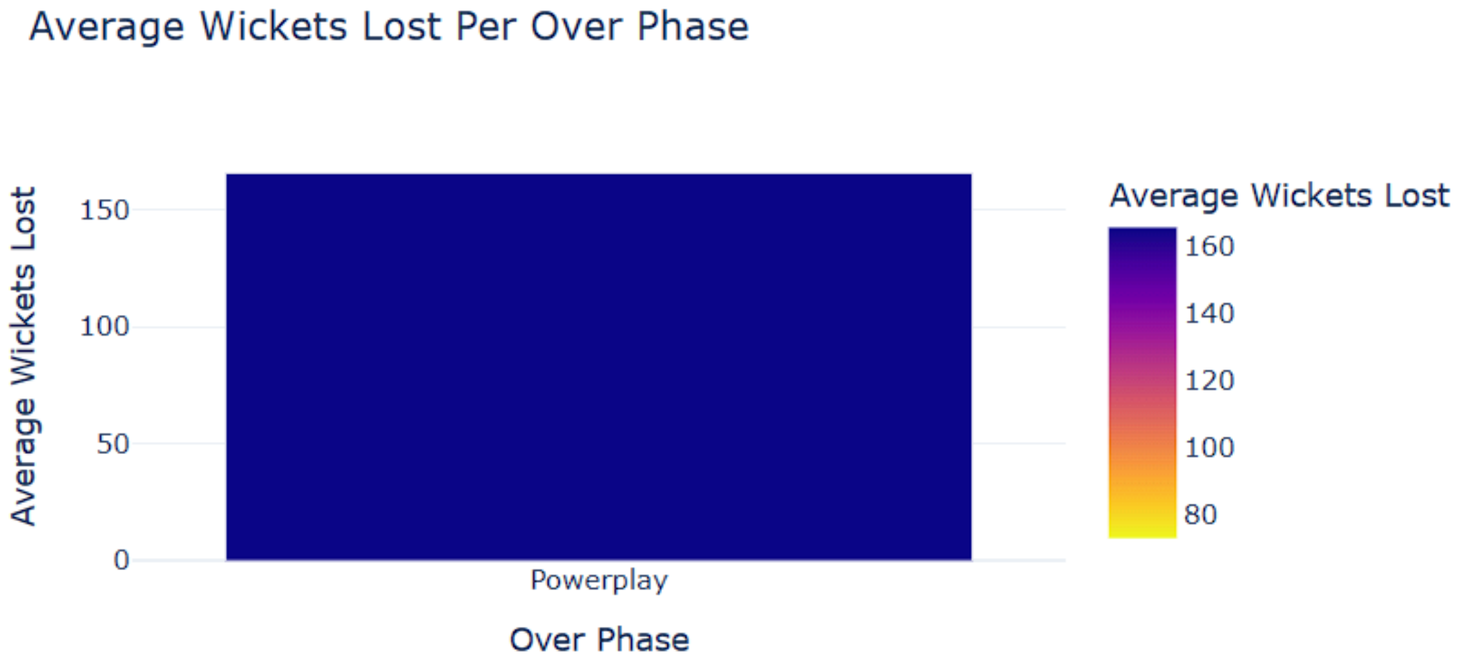
```
import plotly.express as px
```

```
# Plotting Runs per Over Phase
fig_runs_phase = px.bar(avg_phase_performance, x='over_phase', y='avg_runs',
                        title='Average Runs Scored Per Over Phase',
                        labels={'over_phase': 'Over Phase', 'avg_runs': 'Average Runs Scored'},
                        color='avg_runs', color_continuous_scale=px.colors.sequential.Viridis)
fig_runs_phase.update_layout(template="plotly_white")
fig_runs_phase.show()
```



INDIAN PREMIER LEAGUE

```
# Plotting Wickets per Over Phase
fig_wickets_phase = px.bar(avg_phase_performance, x='over_phase', y='avg_wickets',
                           title='Average Wickets Lost Per Over Phase',
                           labels={'over_phase': 'Over Phase', 'avg_wickets': 'Average Wickets Lost'},
                           color='avg_wickets', color_continuous_scale=px.colors.sequential.Plasma_r) # Plasma_r for wickets
fig_wickets_phase.update_layout(template="plotly_white")
fig_wickets_phase.show() #Compare Venue Behavior in High-Scoring vs Low-Scoring Matches
match_total_runs = dfx.groupby('Match_ID').agg(
    total_match_runs=('runs_scored', 'sum'),
    venue=('Venue', 'first')
).reset_index()
```



Compare venue behavior in high-scoring vs low-scoring

```
#Compare Venue Behavior in High-Scoring vs Low-Scoring Matches
match_total_runs = dfx.groupby('Match id').agg(
    total_match_runs=('runs_scored', 'sum'),
    venue=('Venue', 'first')
).reset_index()
```

```
match_total_runs
```

	Match id	total_match_runs	venue
0	335982	268	M Chinnaswamy Stadium
1	335983	430	Punjab Cricket Association Stadium, Mohali
2	335984	244	Feroz Shah Kotla
3	335985	315	Wankhede Stadium
4	335986	184	Eden Gardens
...	...	...	...
1068	1426283	383	Narendra Modi Stadium, Ahmedabad
1069	1426284	334	MA Chidambaram Stadium, Chepauk, Chennai
1070	1426285	293	Eden Gardens, Kolkata
1071	1426286	270	Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...
1072	1426287	297	MA Chidambaram Stadium, Chepauk, Chennai

1073 rows × 3 columns

INDIAN PREMIER LEAGUE

Python: Exploratory Data Analysis (EDA)

Compare venue behavior in high-scoring vs low-scoring matches

```
# Determine a threshold for high vs low scoring matches (e.g., median or 320 for T20s)
total_runs_median = match_total_runs['total_match_runs'].median()
print(f"Median total runs per match: {total_runs_median:.2f}")
```

```
Median total runs per match: 303.00
```

Compare venue behavior in high-scoring vs low-scoring

```
# Categorize matches
match_total_runs['score_category'] = match_total_runs['total_match_runs'].apply(
    lambda x: 'High-Scoring' if x >= total_runs_median else 'Low-Scoring'
)
```

```
match_total_runs
```

	Match id	total_match_runs	venue	score_category
0	335982	268	M Chinnaswamy Stadium	Low-Scoring
1	335983	430	Punjab Cricket Association Stadium, Mohali	High-Scoring
2	335984	244	Feroz Shah Kotla	Low-Scoring
3	335985	315	Wankhede Stadium	High-Scoring
4	335986	184	Eden Gardens	Low-Scoring
...	...	...	...	...
1068	1426283	383	Narendra Modi Stadium, Ahmedabad	High-Scoring
1069	1426284	334	MA Chidambaram Stadium, Chepauk, Chennai	High-Scoring
1070	1426285	293	Eden Gardens, Kolkata	Low-Scoring
1071	1426286	270	Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...	Low-Scoring
1072	1426287	297	MA Chidambaram Stadium, Chepauk, Chennai	Low-Scoring

1073 rows × 4 columns

Compare venue behavior in high-scoring vs low-scoring matches

```
# Count matches by venue and score category
venue_score_distribution = match_total_runs.groupby(['venue', 'score_category']).size().unstack(fill_value=0).reset_index()
```

```
venue_score_distribution
```

score_category	venue	High-Scoring	Low-Scoring
0	Arun Jaitley Stadium	8	6
1	Arun Jaitley Stadium, Delhi	11	3
2	Barabati Stadium	4	3
3	Barsapara Cricket Stadium, Guwahati	2	0
4	Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...	6	7
5	Brabourne Stadium	6	4
6	Brabourne Stadium, Mumbai	13	4
7	Buffalo Park	0	3
8	De Beers Diamond Oval	1	2
9	Dr DY Patil Sports Academy	3	14
10	Dr DY Patil Sports Academy, Mumbai	9	11
11	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...	4	9
12	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...	2	0
13	Dubai International Cricket Stadium	22	24
14	Eden Gardens	42	35
15	Eden Gardens, Kolkata	11	4

Compare venue behavior in high-scoring vs low-scoring matches

```
# Calculate proportions for better comparison
venue_score_distribution['Total Matches'] = venue_score_distribution['High-Scoring'] + venue_score_distribution['Low-Scoring']
venue_score_distribution['High-Scoring %'] = (venue_score_distribution['High-Scoring'] / venue_score_distribution['Total Matches']) * 100
venue_score_distribution['Low-Scoring %'] = (venue_score_distribution['Low-Scoring'] / venue_score_distribution['Total Matches']) * 100
```

```
venue_score_distribution
```

score_category	venue	High-Scoring	Low-Scoring	Total Matches	High-Scoring %	Low-Scoring %
0	Arun Jaitley Stadium	8	6	14	57.142857	42.857143
1	Arun Jaitley Stadium, Delhi	11	3	14	78.571429	21.428571
2	Barabati Stadium	4	3	7	57.142857	42.857143
3	Barsapara Cricket Stadium, Guwahati	2	0	2	100.000000	0.000000
4	Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...	6	7	13	46.153846	53.846154
5	Brabourne Stadium	6	4	10	60.000000	40.000000
6	Brabourne Stadium, Mumbai	13	4	17	76.470588	23.529412
7	Buffalo Park	0	3	3	0.000000	100.000000
8	De Beers Diamond Oval	1	2	3	33.333333	66.666667
9	Dr DY Patil Sports Academy	3	14	17	17.647059	82.352941
10	Dr DY Patil Sports Academy, Mumbai	9	11	20	45.000000	55.000000
11	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...	4	9	13	30.769231	69.230769
12	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...	2	0	2	100.000000	0.000000
13	Dubai International Cricket Stadium	22	24	46	47.826087	52.173913
14	Eden Gardens	42	35	77	54.545455	45.454545

Compare venue behavior in high-scoring vs low-scoring matches

```
# Filter out venues with very few matches to avoid skewed percentages
min_matches_per_venue = 10
venue_score_distribution_filtered = venue_score_distribution[
    venue_score_distribution['Total Matches'] >= min_matches_per_venue
].sort_values(by='High-Scoring %', ascending=False)
```

```
min_matches_per_venue
```

```
10
```

INDIAN PREMIER LEAGUE

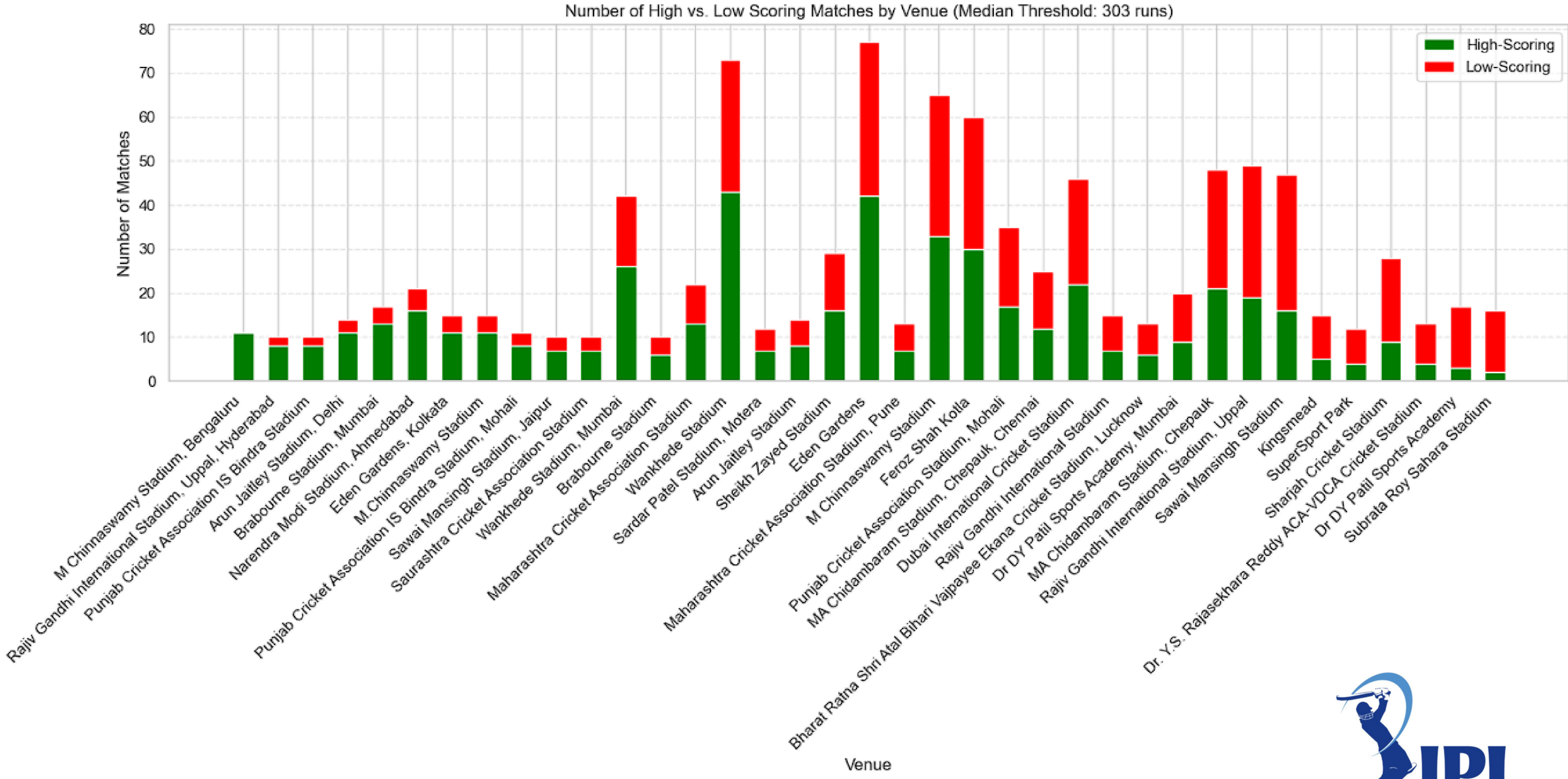
Compare venue behavior in high-scoring vs low-scoring matches

```python
# Data prep
venues = venue_score_distribution_filtered['venue']
high = venue_score_distribution_filtered['High-Scoring']
low = venue_score_distribution_filtered['Low-Scoring']

x = np.arange(len(venues))  # X locations
bar_width = 0.6

# Plot setup
plt.figure(figsize=(16, 8))
plt.bar(x, high, width=bar_width, label='High-Scoring', color='green')
plt.bar(x, low, width=bar_width, bottom=high, label='Low-Scoring', color='red')

plt.title(f'Number of High vs. Low Scoring Matches by Venue (Median Threshold: {total_runs_median:.0f} runs)')
plt.xlabel('Venue')
plt.ylabel('Number of Matches')
plt.xticks(x, venues, rotation=45, ha='right')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```


Number of High vs. Low Scoring Matches by Venue (Median Threshold: 303 runs)
High-Scoring
Low-Scoring
Number of Matches
0
10
20
30
40
50
60
70
80
Venue
M Chinnaswamy Stadium, Bengaluru
Rajiv Gandhi International Stadium, Uppal, Hyderabad
Punjab Cricket Association IS Bindra Stadium
Arun Jaitley Stadium, Delhi
Brabourne Stadium, Mumbai
Narendra Modi Stadium, Ahmedabad
Eden Gardens, Kolkata
M.Chinnaswamy Stadium
Punjab Cricket Association IS Bindra Stadium, Mohali
Sawai Mansingh Stadium, Jaipur
Saurashtra Cricket Association Stadium
Wankhede Stadium, Mumbai
Brabourne Stadium
Maharashtra Cricket Association Stadium
Wankhede Stadium
Sardar Patel Stadium, Motera
Arun Jaitley Stadium
Sheikh Zayed Stadium
Eden Gardens
Maharashtra Cricket Association Stadium, Pune
M Chinnaswamy Stadium
Feroz Shah Kotla
Punjab Cricket Association Stadium, Mohali
MA Chidambaram Stadium, Chepauk, Chennai
Dubai International Cricket Stadium
Rajiv Gandhi International Stadium, Uppal
Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow
Dr DY Patil Sports Academy, Mumbai
MA Chidambaram Stadium, Chepauk
Rajiv Gandhi International Stadium, Uppal
Sawai Mansingh Stadium
Kingsmead
SuperSport Park
Sharjah Cricket Stadium
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium
Dr DY Patil Sports Academy
Subrata Roy Sahara Stadium

INDIAN PREMIER LEAGUE

Compare venue behavior in high-scoring vs low-scoring matches

```
venues = venue_score_distribution_filtered['venue']
high_pct = venue_score_distribution_filtered['High-Scoring %']
low_pct = venue_score_distribution_filtered['Low-Scoring %']

x = np.arange(len(venues))  # X-axis positions
bar_width = 0.6

# Create plot
plt.figure(figsize=(18, 8))
plt.bar(x, high_pct, width=bar_width, label='High-Scoring %', color='green')
plt.bar(x, low_pct, width=bar_width, bottom=high_pct, label='Low-Scoring %', color='red')

# Title and labels
plt.title(f'Percentage of High vs. Low Scoring Matches by Venue (Min {min_matches_per_venue} Matches)')
plt.xlabel('Venue')
plt.ylabel('Percentage of Matches')
plt.xticks(x, venues, rotation=45, ha='right')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

The Originator

VARUN KUMAR

Thank You!

"Turning data into actionable insights".

Find this useful ? Don’t Forget To Like And Share This Post.