CAPSTONE PROJECT
DATA ANALYSIS
OF

IPL

INDIAN PREMIER LEAGUE

PRESENTATION TEMPLATE

SUBMITTED BY:-
DEEPESH RAI
COHORT-4

SQL QUERIES

Which batsman has scored the highest total runs?

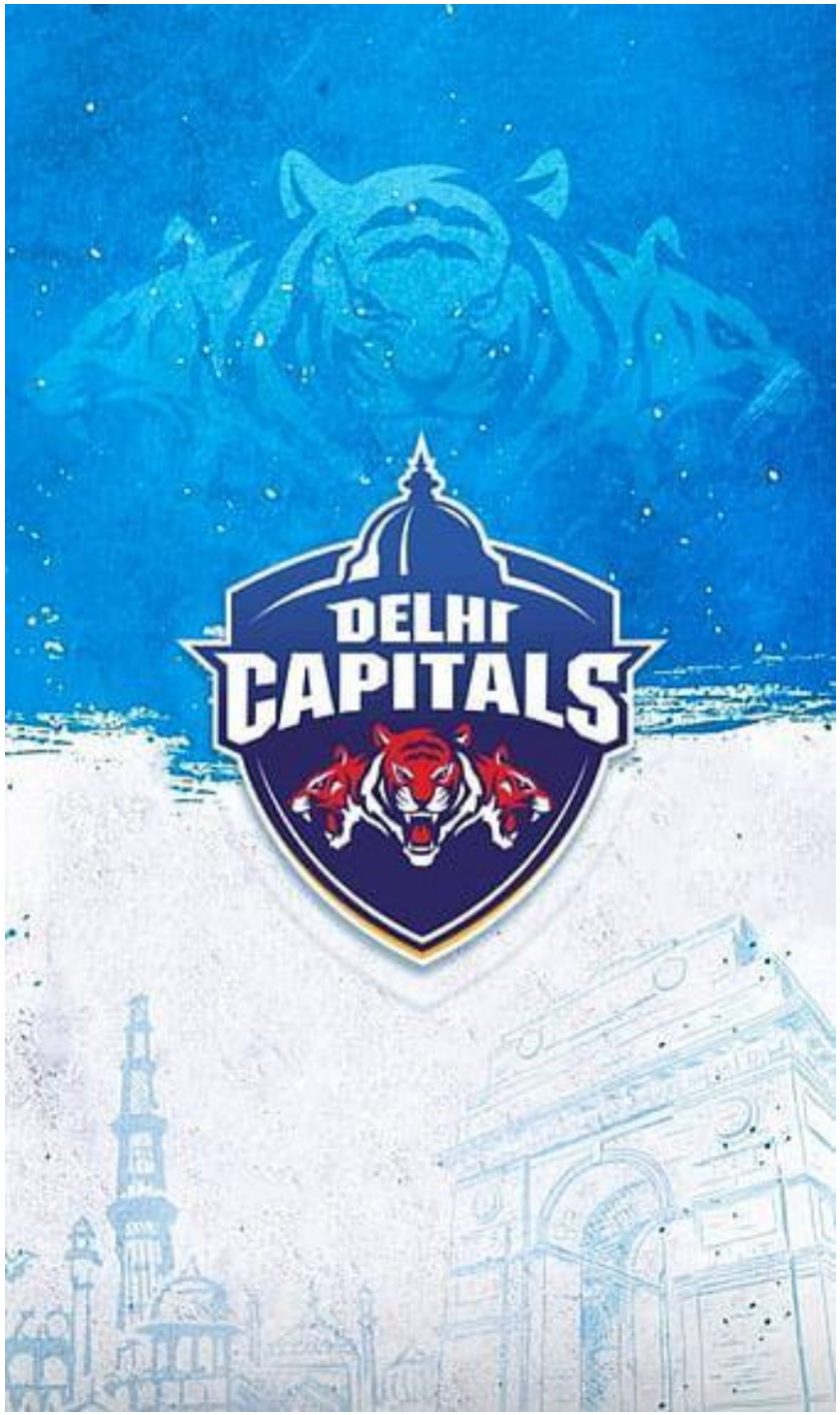
```
SELECT
batter AS batsman,
SUM(batsman_runs) AS
total_runs
GROUP BY
Batter
ORDER BY
total_runs DESC
LIMIT 1;
```

Which team has conceded the least extras in the matches?

```
SELECT
bowling_team,
SUM(extra_runs)AS
 total_extras_conceded
GROUP BY
    bowling_team
ORDER BY
 total_extras_conceded ASC
LIMIT 1;
```

What is the total number of runs scored by each team across all matches?

```
SELECT     batting_team,
    SUM(total_runs) AS
 total_runs_scored
GROUP BY
 batting_team
ORDER BY
 total_runs_scored DESC;
```



Who are the top 5 bowlers based on runs conceded per over (economy rate)?

```
SELECT
   bowler,
    SUM(total_runs) AS runs_conceded,
COUNT(*) AS balls_bowled,
ROUND(SUM(total_runs) * 1.0 /
(COUNT(*) / 6.0), 2) AS economy_rate
GROUP BY
 bowler
HAVING     COUNT(*) >= 60 -- filter:
bowler must have bowled at least 10
overs
ORDER BY
economy_rate ASC
LIMIT 5;
```

Which batsman has faced the most balls?

```
SELECT
  batter AS batsman,    COUNT(*) AS
balls_faced
WHERE
 extra_type IS NULL OR
 extra_type != 'wides‘
GROUP BY     batter
ORDER BY     balls_faced DESC
LIMIT 1;
```

Which bowling team has taken part in the most overs?

```
SELECT
bowling_team,
COUNT(DISTINCT match_id || '-' ||
innings || '-' || over) AS
total_overs_bowled
GROUP BY
bowling_team
ORDER BY
total_overs_bowled DESC
LIMIT 1;
```

What is the average number of runs scored per ball by each batsman?

```
SELECT
    batter AS batsman,
SUM(batsman_runs) AS
 total_runs,
COUNT(*) FILTER (WHERE extra_type
IS NULL OR extra_type != 'wides') AS
balls_faced,
ROUND(SUM(batsman_runs) * 1.0 /
COUNT(*) FILTER (WHERE extra_type
IS NULL OR extra_type != 'wides'), 2)
AS avg_runs_per_ball
GROUP BY
Batter
ORDER BY
avg_runs_per_ball DESC;
```

Which teams have the best win percentage according to the team_performance table?

```
SELECT
   team,    matches_played,
matches_won,
ROUND((matches_won *
100.0) / matches_played, 2)
AS win_percentage
  WHERE
 matches_played > 0
ORDER BY
win_percentage DESC;
```

List all players who are all-rounders (both batting and bowling style available).

```
SELECT
    player_name,
   batting_style,
  bowling_style
 WHERE
batting_style IS NOT NULL
 AND TRIM(batting_style) != ''
AND bowling_style IS NOT
NULL
AND TRIM(bowling_style) != '';
```

What is the average number of runs scored per over by each batting team?

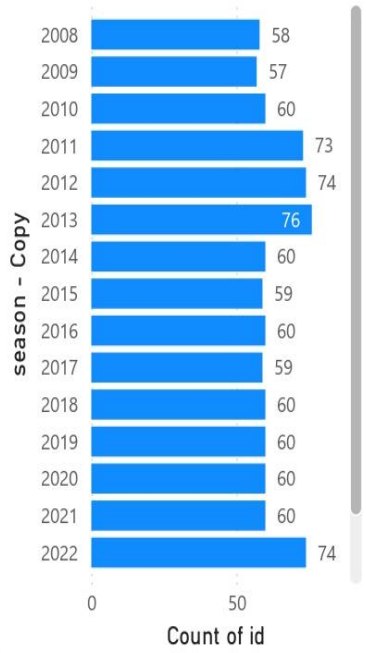
```
SELECT
 batting_team,
SUM(total_runs) AS total_runs,
COUNT(DISTINCT match_id || '-' ||
innings || '-' || over)
AS total_overs,
ROUND(SUM(total_runs) * 1.0 /
COUNT(DISTINCT match_id || '-' ||
innings || '-' || over), 2)
AS avg_runs_per_over
GROUP BY
batting_team
ORDER BY
avg_runs_per_over DESC;
```


IPL 2024
Forever Focused on Exceptional Design
DASHBOARD ANALYSIS
SHREYAS
DHL
33
HARDIK
SAMSON
India Cements
7
DHONI
MARKRAM
KEI
Wires & Cables
13
FAF
7
GILL

DASHBOARD-1

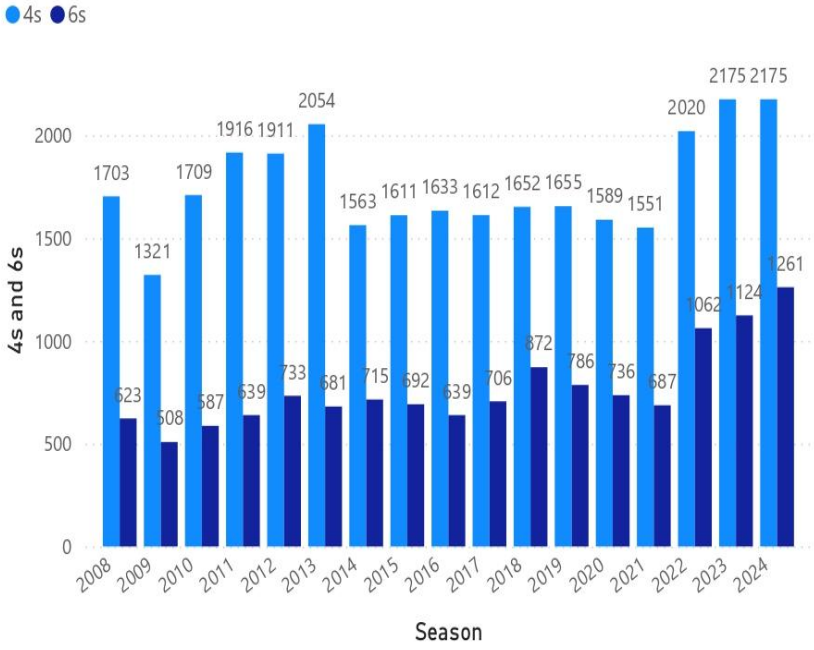
IPL Season Highlights

Matches Played Per Season

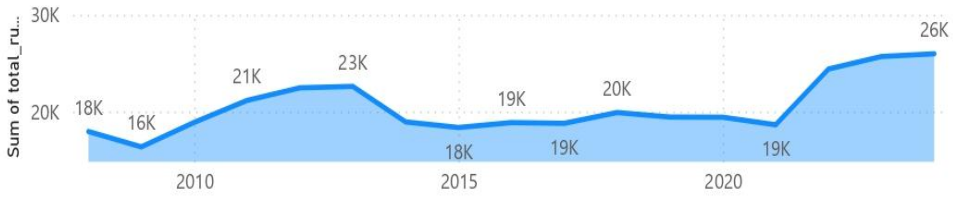


Season	Season Winner	Runner Up
2008	Rajasthan Royals	Chennai Super Kings
2009	Deccan Chargers	Royal Challengers Bengaluru
2010	Chennai Super Kings	Mumbai Indians
2011	Chennai Super Kings	Royal Challengers Bengaluru
2012	Kolkata Knight Riders	Chennai Super Kings
2013	Mumbai Indians	Chennai Super Kings
2014	Kolkata Knight Riders	Kings XI Punjab
2015	Mumbai Indians	Chennai Super Kings
2016	Sunrisers Hyderabad	Royal Challengers Bengaluru
2017	Mumbai Indians	Rising Pune Supergiants
2018	Chennai Super Kings	Sunrisers Hyderabad
2019	Mumbai Indians	Chennai Super Kings
2020	Mumbai Indians	Delhi Capitals
2021	Chennai Super Kings	Kolkata Knight Riders
2022	Gujarat Titans	Rajasthan Royals
2023	Chennai Super Kings	Gujarat Titans
2024	Kolkata Knight Riders	Sunrisers Hyderabad

4s and 6s by Season



Sum of total_runs by season



Win Margin Wickets	Win Margin Runs	Number of Boundaries	Number of Sixes
30	6.19	30K	13K
Average	Average	4s	6s

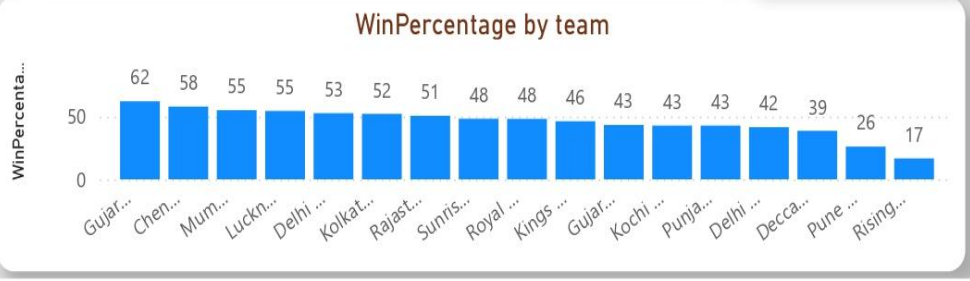
DASHBOARD-2

Team Performance

Insights

Team	WinPercentage	MatchesPlayed	MatchesWon
Chennai Super Kings	57.98	238	138
Deccan Chargers	38.67	75	29
Delhi Capitals	52.75	91	48
Delhi Daredevils	41.61	161	67
Gujarat Lions	43.33	30	13
Gujarat Titans	62.22	45	28
Kings XI Punjab	46.32	190	88
Kochi Tuskers Kerala	42.86	14	6
Kolkata Knight Riders	52.19	251	131
Lucknow Super Giants	54.55	44	24
Mumbai Indians	55.17	261	144
Pune Warriors	26.09	46	12
Punjab Kings	42.86	56	24
Rajasthan Royals	50.68	221	112
Rising Pune Supergiants	16.67	30	5
Royal Challengers Bengaluru	48.24	255	123
Sunrisers Hyderabad	48.35	182	88
Total			

Opponent Team	Chennai Super Kings		Deccan Chargers		Delhi Capitals		Delhi Daredevils		Gujarat Lions		C
	Wins	Losses	Wins	Losses	Wins	Losses	Wins	Losses	Wins	Losses	W
Chennai Super Kings			6	4	7	5	12	6			
Deccan Chargers	4	6					4	7			
Delhi Capitals	5	7									
Delhi Daredevils	6	12	7	4					3	1	
Gujarat Lions							1	3			
Gujarat Titans	4	3			2	3					
Kings XI Punjab	9	14	7	3	2	2	13	9	2	2	
Kochi Tuskers Kerala	1	1		1			1	1			
Kolkata Knight Riders	10	19	7	2	5	7	13	8	1	3	
Lucknow Super Giants	3	1			3	2					
Mumbai Indians	20	17	6	4	8	5	11	11	2	2	
NA							1				
Pune Warriors	2	4	1	3			2	3			
Punjab Kings	5	2			2	5					
Rajasthan Royals	13	16	7	2	4	7	11	7			
Rising Pune Supergiant											
Rising Pune Supergiants							2	2		3	
Royal Challengers Bengaluru	11	21	5	6	5	5	14	6	3	2	



Total Matches Played

1095

Wins by Toss Decision

554

Season

All

Match Type

All

DASHBOARD-3

Player Statistics

Insights

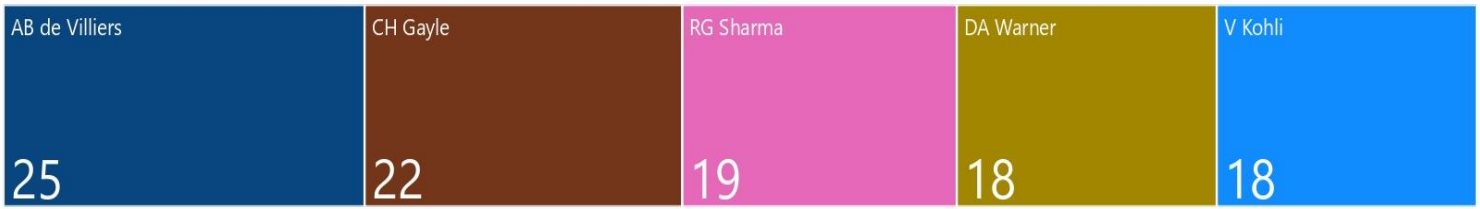
Top Batter with Their Average

Season	TopScorerWithAverage
2008	SE Marsh - Avg 68.4
2009	ML Hayden - Avg 57.2
2010	SR Tendulkar - Avg 47.5
2011	CH Gayle - Avg 67.6
2012	CH Gayle - Avg 61.1
2013	MEK Hussey - Avg 56.4
2014	RV Uthappa - Avg 44.0
2015	DA Warner - Avg 46.8
2016	V Kohli - Avg 81.1
2017	DA Warner - Avg 58.3
2018	KS Williamson - Avg 52.5
2019	DA Warner - Avg 69.2
2020	KL Rahul - Avg 48.3
2021	RD Gaikwad - Avg 45.4
2022	JC Buttler - Avg 57.5
2023	Shubman Gill - Avg 59.3
2024	V Kohli - Avg 61.8

Bowler Stats

Season	Wickets	RunsConceded	OversBowled	EconomyRate
2023	**859**	**25260**	**2,872.83**	**8.79**
2022	**850**	**23982**	**2,868.33**	**8.36**
2013	**831**	**22143**	**2,949.83**	**7.51**
2024				
HV Patel	24	477	49.33	9.67
CV Varun	21	401	50.33	7.97
JJ Bumrah	20	336	52.50	6.40
AD Russell	19	295	29.33	10.06
Arshdeep Singh	19	505	50.33	10.03
Avesh Khan	19	526	55.00	9.56
Harshit Rana	19	383	42.17	9.08
T Natarajan	19	465	51.33	9.06
PJ Cummins	18	566	61.00	9.28
YS Chahal	18	546	58.17	9.39
KK Ahmed	17	479	50.17	9.55
MA Starc	17	444	41.83	10.61
Mukesh Kumar	17	368	36.00	10.22
SP Narine	17	368	55.00	6.69

Top 5 Player of the match winners



Season: All

Match Type: All

Top Six Hitter: CH Gayle

Top Fours Hitter: S Dhawan

Method Type: All

Super Over: All

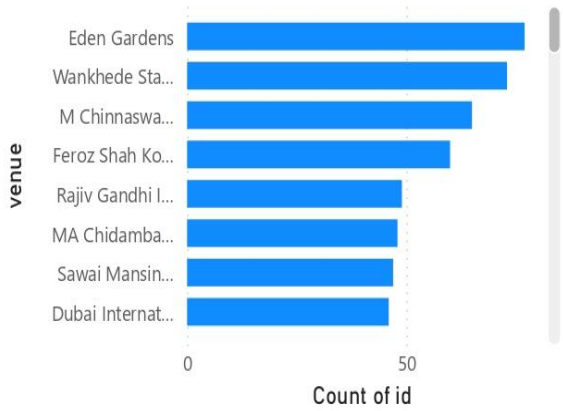
DASHBOARD-4

Season: All

Match Type: All

IPL Venue Analysis

Top venues by matches Played



Venue Wise Average Score

venue	AverageFirstInningsScore	AverageSecondInningsScore
Eden Gardens, Kolkata	195.63	184.69
M Chinnaswamy Stadium, Bengaluru	196.21	183.93
Arun Jaitley Stadium, Delhi	199.06	181.63
Total	**165.55**	**152.19**

Venue Wise Average Score

venue	WinPercent
Arun Jaitley Stadium	**0.00**
Chennai Super Kings	50.00
Delhi Capitals	57.14
Delhi Daredevils	57.14
Kings XI Punjab	50.00
Mumbai Indians	50.00
Royal Challengers Bengaluru	50.00
Sunrisers Hyderabad	100.00
Arun Jaitley Stadium, Delhi	**0.00**
Barabati Stadium	**0.00**
Total	**0.00**

Toss outcome and result pattern per venue

toss_decision	bat		field	
venue	Sum of TotalMatches	Sum of WinsWhenTossWon	Sum of TotalMatches	Sum of WinsWhenTossWon
Arun Jaitley Stadium	6	3	8	4
Arun Jaitley Stadium, Delhi	4	2	12	5
Barabati Stadium	2	2	5	3
Barsapara Cricket Stadium, Guwahati	1		2	
Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow	6	4	8	4
Brabourne Stadium	6	4	4	2
Brabourne Stadium, Mumbai	3	1	14	7
Buffalo Park	3	2		
De Beers Diamond Oval	2	1	1	1

winner	HomeWinPercent	HomeWins	AwayWins	AwayWinPercent
Kolkata Knight Riders	34.35	45	86	65.65
Mumbai Indians	29.17	42	102	70.83
Rajasthan Royals	28.57	32	80	71.43
Sunrisers Hyderabad	11.36	10	78	88.64
Delhi Capitals	8.33	4	44	91.67
Chennai Super Kings			138	100.00
Deccan Chargers			29	100.00
Delhi Daredevils			67	100.00
Gujarat Lions			13	100.00
Gujarat Titans			28	100.00
Kings XI Punjab			88	100.00
Kochi Tuskers Kerala			6	100.00
Lucknow Super Giants			24	100.00
Total	**12.15**	**133**	**962**	**87.85**

DASHBOARD-5

IPL Match Situations

Batting Team: All

Bowling Team: All

578
Won by Chasing

498
Won by Defending

14
Superover Matches

Last 5 Overs Performance

Total_runs	4s	6s	Wicket	Matches	Extra_runs	Balls	Batter Runs
93884	6781	4725	4788	1083	4977	216718	88907

SuperOver Matches Overview

team1	team2	winner	player_of_match
Chennai Super Kings	Kings XI Punjab	Kings XI Punjab	J Theron
Delhi Capitals	Kings XI Punjab	Delhi Capitals	MP Stoinis
Delhi Capitals	Sunrisers Hyderabad	Delhi Capitals	PP Shaw
Gujarat Lions	Mumbai Indians	Mumbai Indians	KH Pandya
Kolkata Knight Riders	Delhi Capitals	Delhi Capitals	PP Shaw
Kolkata Knight Riders	Rajasthan Royals	Rajasthan Royals	JP Faulkner
Kolkata Knight Riders	Rajasthan Royals	Rajasthan Royals	YK Pathan
Kolkata Knight Riders	Sunrisers Hyderabad	Kolkata Knight Riders	LH Ferguson
Mumbai Indians	Kings XI Punjab	Kings XI Punjab	KL Rahul
Mumbai Indians	Sunrisers Hyderabad	Mumbai Indians	JJ Bumrah
Rajasthan Royals	Kings XI Punjab	Kings XI Punjab	SE Marsh
Royal Challengers Bengaluru	Delhi Daredevils	Royal Challengers Bengaluru	V Kohli
Royal Challengers Bengaluru	Mumbai Indians	Royal Challengers Bengaluru	AB de Villiers

Power Play Performance

Total_runs	4s	6s	Wicket	Matches
103217	12532	2960	3163	1095

Middle Over Performance

Total_runs	4s	6s	Wicket	Matches
135816	9508	4985	4555	1094

Death over Performance

Total_runs	4s	6s	Wicket	Matches
75412	5538	3873	4052	1076

Fall of wickets timing

over	Num. of Wicket
19	1308
18	989
17	976
16	779
15	736
14	695
13	633
12	585
4	576
10	568
11	567
3	548
5	540
8	537
2	533

DASHBOARD-6

IPL
Season-wise Trends

Season: All

Player Participation Trend by Season

Season	Bowlers Count	Batters Count	Allrounders
2008	98	154	92
2009	112	150	97
2010	123	164	108
2011	134	180	116
2012	129	170	111
2013	131	172	107
2014	101	133	82
2015	99	129	83

Match totals vs overs comparison

inning	1			2		
match_id	Season	Total Runs	Overs	Season	Total Runs	Overs
335982	2008	222	20	2008	82	16
335983	2008	240	20	2008	207	20
335984	2008	129	20	2008	132	16
335985	2008	165	20	2008	166	20
335986	2008	110	19	2008	112	19
335987	2008	166	20	2008	168	19
335988	2008	142	20	2008	143	14
335989	2008	208	20	2008	202	20

Strike rate evolution

Season	StrikeRate
2024	151.22
2023	142.48
2018	138.31
2022	134.48
2019	134.28
2017	133.72
2015	132.52
2020	132.06
2016	131.90
2008	129.80
2014	129.59
2021	127.64
2010	127.38
2012	123.91
2013	121.83
2011	121.33
2009	117.42
Total	131.25

348K
Sum of total_runs

13K
Sum of is_wicket

17
Count of team

30K
4s

13K
6s

Bowling economy trends

Strike rate per Player

batter	StrikeRate	BallsFaced	Runs
A Ashish Reddy	145.08	193	280
A Badoni	134.32	472	634
A Chopra	74.65	71	53
A Flintoff	116.98	53	62
A Manohar	132.76	174	231
A Mishra	91.37	417	381
A Nehra	68.33	60	41
A Raghuvanshi	155.24	105	163
Total	132.05	246430	325411

SUMMARY

IPL Dashboard 1: Season Highlights*This dashboard summarizes each IPL season from 2008 to 2024, highlighting major statistics and visual trends. The total number of matches played per season ranged from 57 to 76, reflecting an expanding tournament structure. Notably, Chennai Super Kings, Mumbai Indians, and Kolkata Knight Riders dominated with multiple championships, while newer entrants like Gujarat Titans made an impact in 2022.A bar chart displays total runs scored per season, showing a consistent increase over the years, peaking in 2023 and 2024, indicating an evolution toward high-scoring games. The total number of boundaries also increased, with 2175 fours and 1261 sixes recorded in 2024 alone. The win margins averaged 30 runs and 6.19 wickets, suggesting a balance between strong batting and bowling performances.The dashboard reflects IPL's transformation into a power-packed, batting-friendly tournament, with a focus on crowd-pulling big hits and competitive cricket across seasons.

IPL Dashboard 2: Team Performance*This dashboard focuses on team statistics such as matches played, won, and win percentages. Gujarat Titans top the win percentage chart at 62.22%, followed by Chennai Super Kings at 57.98% and Mumbai Indians at 55.17%. Teams like Rising Pune Supergiants and Pune Warriors struggled with win rates below 30%.A comparative matrix offers a deep dive into team-vs-team outcomes, revealing competitive rivalries and dominance patterns. CSK vs MI and RCB vs KKR have been high-voltage contests with fluctuating results.Toss decisions also influence outcomes, with 554 matches won by the team winning the toss out of a total of 1095, suggesting a moderate impact. This dashboard effectively showcases team strengths and weaknesses, critical for strategic planning and fan analysis.---

IPL Dashboard 3: Player Statistics*This dashboard highlights individual brilliance across seasons. Top batters like V Kohli (2024, Avg 61.8), DA Warner, and KL Rahul consistently top the charts with high averages. CH Gayle is the leading six-hitter, while S Dhawan leads in fours.On the bowling side, HV Patel led 2023 with 24 wickets. Other key performers include JJ Bumrah and CV Varun, showcasing their consistency and economy. Bowling data shows an increase in run rates, with 2023 witnessing the highest economy rates.Strike rates have risen dramatically over time, with 2024 reaching 151.22, the highest in IPL history. This dashboard effectively celebrates individual records and contributions to the league's growth.

IPL Dashboard 4: Venue Analysis*This dashboard dissects venue-wise statistics including average scores, win percentages, and match counts. Top venues like M Chinnaswamy Stadium and Eden Gardens support high scores, with average first innings nearing 200.Win percentage varies with location. For instance, Delhi Capitals and Sunrisers Hyderabad have higher away win percentages. Toss decision analysis shows that batting or fielding first influences results differently across venues.The home-away win analysis reveals teams like Mumbai Indians and Kolkata Knight Riders leveraging home advantage, while some teams like CSK dominate irrespective of venue. This dashboard is valuable for match predictions and team strategy.

IPL Dashboard 5: Match Situations*This dashboard provides a breakdown of match phases: Powerplay, Middle Overs, and Death Overs. Powerplays see explosive starts (\~103K runs), while death overs see 75K+ runs and the highest six counts, highlighting their match-deciding nature.Super Over insights cover 14 intense games, showcasing thrilling finishes and clutch performances by players like AB de Villiers and KL Rahul.Fall-of-wicket data indicates maximum dismissals in the 19th over, hinting at pressure situations in death overs. This dashboard is crucial for understanding how match momentum shifts over different phases.

IPL Dashboard 6: Season-wise Trends*This dashboard analyzes long-term trends. Player participation peaked during 2011–2013 with over 400 total players per season. Strike rates have steadily increased, showing a shift toward aggressive batting strategies.Bowling economy has also risen, reflecting challenges faced by bowlers in modern T20 cricket. A season-wise match total vs overs comparison shows increasingly balanced and competitive games.This dashboard offers macro-level insights into how the IPL has evolved into a data-driven, fast-paced format. It's ideal for analysts tracking the league's transformation over time.


VISUAL STORYTELLING USING
PYTHON & LIBRARIES

IPL ANALYSIS USING PYTHON (LIBRARIES)

```
import pandas as pd
```

IPL Analysis Using Python Pandas

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from IPython.display import Image, display

display(Image('/content/canvasnobg.png'))
```

```
# Reading the files as DataFrames

matches = pd.read_csv('/content/matches.csv')
deliveries = pd.read_csv('/content/deliveries.csv')

# Merge deliveries with matches to bring in season, venue, etc.

# Merge on match ID
df = deliveries.merge(matches[['id', 'season', 'venue', 'team1', 'team2']],
                      left_on='match_id',
                      right_on='id',
                      how='left')

# Clean & convert columns
# ✔ Remove unused columns

df.drop(['id'], axis=1, inplace=True)  # id from matches is now redundant

df.head()
```

	match_id	inning	batting_team	bowling_team	over	ball	batter	bowler	non_striker	batsman_runs	...	total_runs	extras_typ
0	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	1	SC Ganguly	P Kumar	BB McCullum	0	...	1	legbye
1	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	2	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na
2	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	3	BB McCullum	P Kumar	SC Ganguly	0	...	1	wide
3	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	4	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na
4	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	5	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na

5 rows × 21 columns

```
# ✔ Convert column names to lowercase (optional)

df.columns = df.columns.str.lower()

df.head()
```

	match_id	inning	batting_team	bowling_team	over	ball	batter	bowler	non_striker	batsman_runs	...	total_runs	extras_typ
0	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	1	SC Ganguly	P Kumar	BB McCullum	0	...	1	legbye
1	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	2	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na
2	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	3	BB McCullum	P Kumar	SC Ganguly	0	...	1	wide
3	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	4	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na
4	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	5	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na

5 rows × 21 columns

```
# Handle nulls and NAs
# Example: Fill or drop common NAs

df['player_dismissed'] = df['player_dismissed'].fillna('Not Out')
df['dismissal_kind'] = df['dismissal_kind'].fillna('NA')
```

```
df.head()
```

	match_id	inning	batting_team	bowling_team	over	ball	batter	bowler	non_striker	batsman_runs	...	total_runs	extras_typ
0	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	1	SC Ganguly	P Kumar	BB McCullum	0	...	1	legbye
1	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	2	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na
2	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	3	BB McCullum	P Kumar	SC Ganguly	0	...	1	wide
3	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	4	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na
4	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	5	BB McCullum	P Kumar	SC Ganguly	0	...	0	Na

5 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260920 entries, 0 to 260919
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   match_id          260920 non-null  int64
 1   inning            260920 non-null  int64
 2   batting_team      260920 non-null  object
 3   bowling_team      260920 non-null  object
 4   over              260920 non-null  int64
 5   ball              260920 non-null  int64
 6   batter            260920 non-null  object
 7   bowler            260920 non-null  object
 8   non_striker       260920 non-null  object
 9   batsman_runs      260920 non-null  int64
 10  extra_runs        260920 non-null  int64
 11  total_runs        260920 non-null  int64
 12  extras_type       14125 non-null   object
 13  is_wicket         260920 non-null  int64
 14  player_dismissed  260920 non-null  object
 15  dismissal_kind    260920 non-null  object
 16  fielder           9354 non-null    object
 17  season            260920 non-null  int64
 18  venue             260920 non-null  object
 19  team1             260920 non-null  object
 20  team2             260920 non-null  object
dtypes: int64(9), object(12)
```

```
memory usage: 41.8+ MB
```

```
# Fix dtypes if needed

# If season, match_id, or over are not integers:

df['season'] = df['season'].astype(int)
df['match_id'] = df['match_id'].astype(int)
df['over'] = df['over'].astype(int)
df['ball'] = df['ball'].astype(int)


# Save combined cleaned data (optional, if needed for further analysis)

df.to_csv('/content/ipl_cleaned.csv', index=False)


# import plotting and visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px


# Merge to get season in deliveries
df = deliveries.merge(matches[['id', 'season']], left_on='match_id', right_on='id', how='left')


df.head()
```

	match_id	inning	batting_team	bowling_team	over	ball	batter	bowler	non_striker	batsman_runs	extra_runs	total_runs	ext
0	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	1	SC Ganguly	P Kumar	BB McCullum	0	1	1	
1	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	2	BB McCullum	P Kumar	SC Ganguly	0	0	0	
2	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	3	BB McCullum	P Kumar	SC Ganguly	0	1	1	
3	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	4	BB McCullum	P Kumar	SC Ganguly	0	0	0	
4	335982	1	Kolkata Knight Riders	Royal Challengers Bengaluru	0	5	BB McCullum	P Kumar	SC Ganguly	0	0	0	

```
# Total Runs per Season
season_runs = df.groupby('season')['total_runs'].sum().reset_index()

# Matches per Season (for avg)

season_matches = matches.groupby('season')['id'].nunique().reset_index(name='match_count')
```

Analyze run trends over the years

```
season_matches
```

	season	match_count
0	2008	58
1	2009	57
2	2010	60
3	2011	73
4	2012	74
5	2013	76
6	2014	60
7	2015	59
8	2016	60
9	2017	59
10	2018	60
11	2019	60
12	2020	60
13	2021	60
14	2022	74
15	2023	74
16	2024	71

Next steps: Generate code with season_matches | View recommended plots | New interactive sheet

season_runs

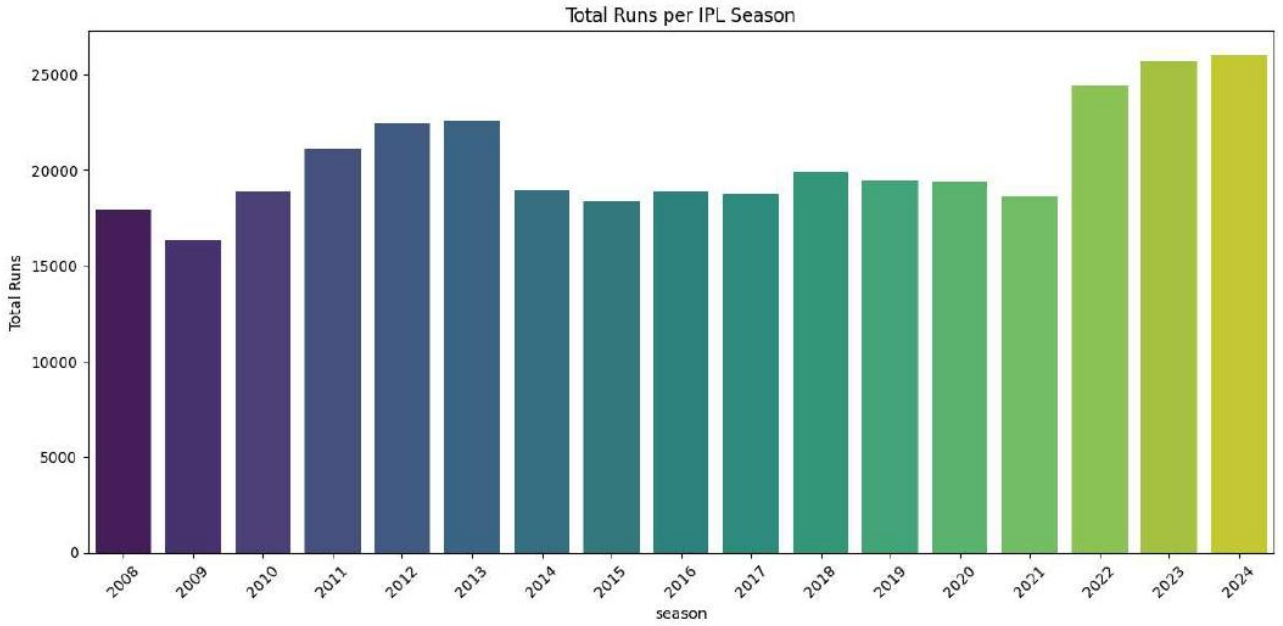
	season	total_runs
0	2008	17937
1	2009	16353
2	2010	18883
3	2011	21154
4	2012	22453
5	2013	22602
6	2014	18931
7	2015	18353
8	2016	18862
9	2017	18786
10	2018	19901
11	2019	19434
12	2020	19416
13	2021	18637
14	2022	24395
15	2023	25688
16	2024	25971

Next steps: Generate code with season_runs | View recommended plots | New interactive sheet

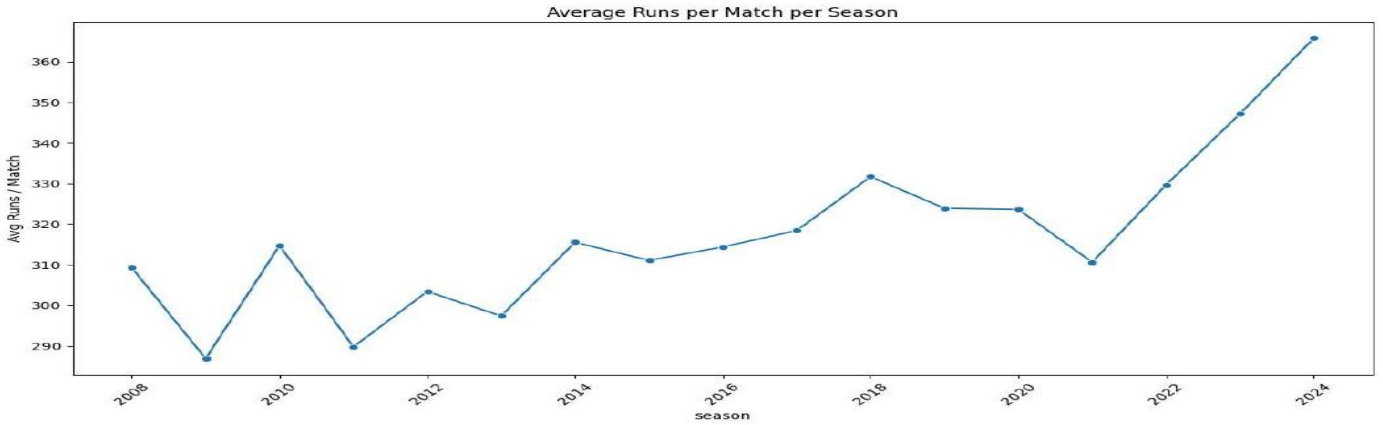
```
# Combine both

season_stats = season_runs.merge(season_matches, on='season')
season_stats['avg_runs_per_match'] = season_stats['total_runs'] / season_stats['match_count']


# Barplot - Total Runs per Season (Matplotlib + Seaborn)

plt.figure(figsize=(12,6))
sns.barplot(x='season', y='total_runs', data=season_stats, palette='viridis')
plt.title('Total Runs per IPL Season')
plt.ylabel('Total Runs')
plt.xticks(rotation=45)
```

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

```
/tmp/ipython-input-48-914421280.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le
```




```
# Line Chart – Average Runs per Match per Season (Seaborn)

plt.figure(figsize=(12,6))
sns.lineplot(x='season', y='avg_runs_per_match', data=season_stats, marker='o')
plt.title('Average Runs per Match per Season')
plt.ylabel('Avg Runs / Match')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

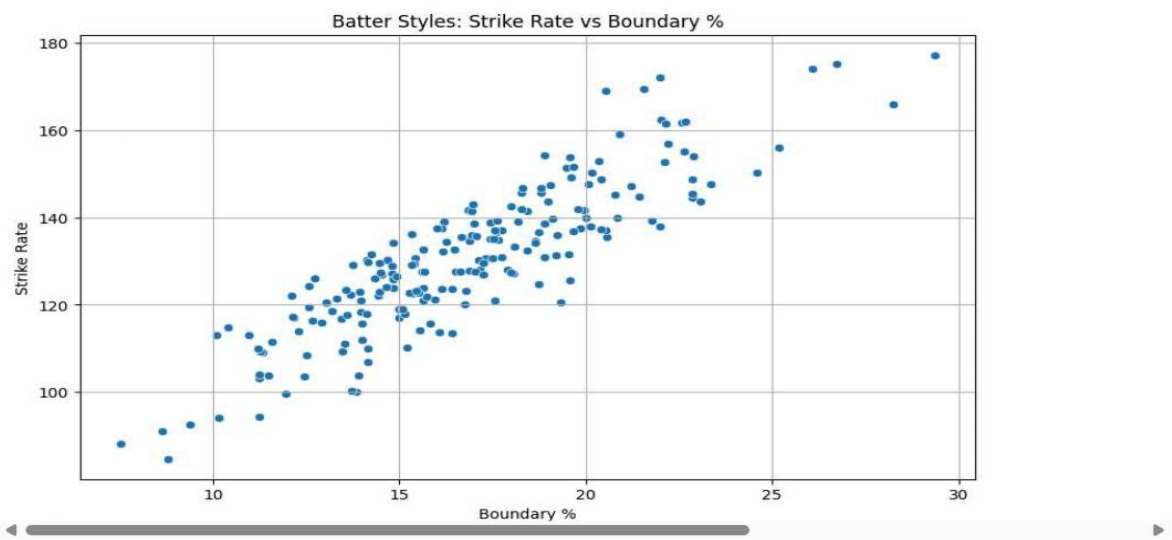



```
# Interactive Plotly Chart

fig = px.line(season_stats, x='season', y='avg_runs_per_match',
              title='Interactive: Avg Runs per Match Over IPL Seasons',
              markers=True)
fig.update_traces(line_color='crimson')
fig.show()
```

Compare batting styles (anchor vs aggressive) via strike rate and boundary %

```
# Calculate per-batter stats

# Filter out extras like wides/noballs
valid_balls = deliveries[
    deliveries['extras_type'].isna() |
    (~deliveries['extras_type'].isin(['wides', 'noballs']))
]
```

```
# Calculate stats
batter_stats = valid_balls.groupby('batter').agg(
    runs=('batsman_runs', 'sum'),
    balls=('ball', 'count'),
    fours=('batsman_runs', lambda x: (x == 4).sum()),
    sixes=('batsman_runs', lambda x: (x == 6).sum())
).reset_index()

batter_stats['strike_rate'] = batter_stats['runs'] / batter_stats['balls'] * 100
batter_stats['boundary_pct'] = (batter_stats['fours'] + batter_stats['sixes']) / batter_stats['balls'] * 100

# Filter to batters who faced enough balls (e.g., 200+)
batter_stats = batter_stats[batter_stats['balls'] >= 200]


plt.figure(figsize=(10, 6))
sns.scatterplot(data=batter_stats, x='boundary_pct', y='strike_rate')
plt.title('Batter Styles: Strike Rate vs Boundary %')
plt.xlabel('Boundary %')
plt.ylabel('Strike Rate')
plt.grid(True)
plt.show()
```

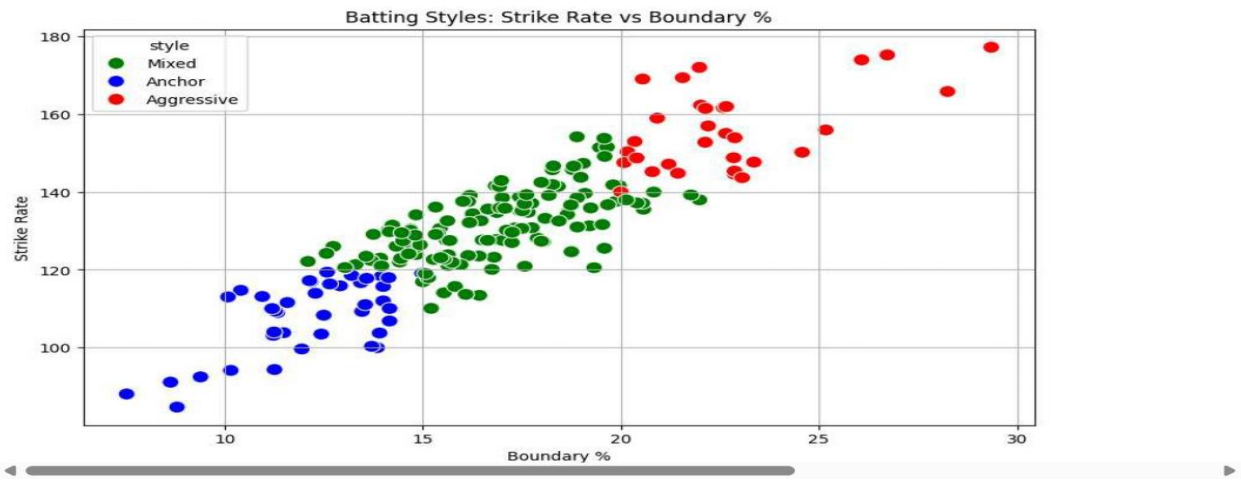



```
# Interactive Plotly Bubble Chart


fig = px.scatter(
    batter_stats,
    x='boundary_pct',
    y='strike_rate',
    size='runs',
    text='batter',
    hover_name='batter',
    title='Batting Styles: Anchor vs Aggressive (IPL)',
    labels={'boundary_pct': 'Boundary %', 'strike_rate': 'Strike Rate'}
)
fig.update_traces(textposition='top center')
fig.show()
```

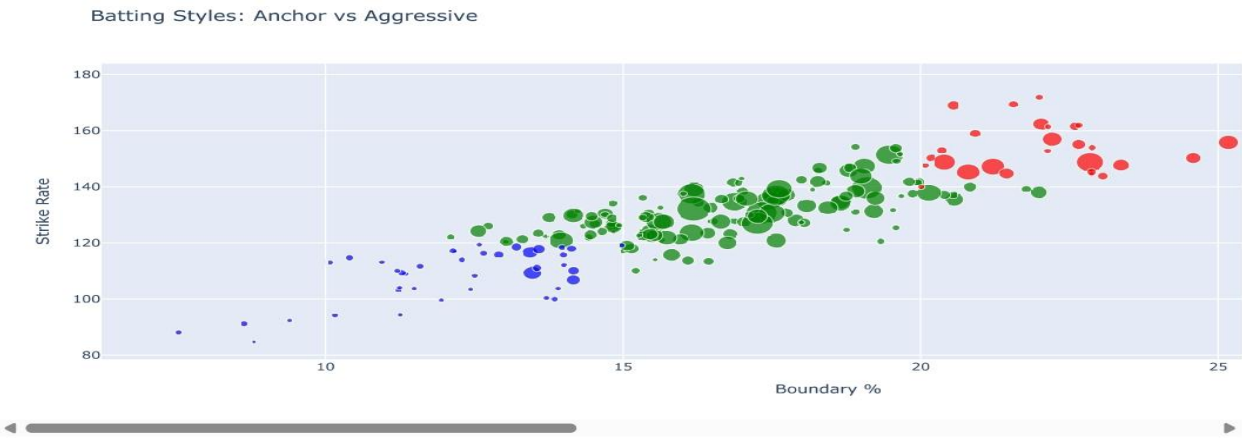

```
# Categorize Player Types

def label_style(row):
    if row['strike_rate'] >= 140 and row['boundary_pct'] >= 20:
        return 'Aggressive'
    elif row['strike_rate'] < 120 and row['boundary_pct'] < 15:
        return 'Anchor'
    else:
        return 'Mixed'

batter_stats['style'] = batter_stats.apply(label_style, axis=1)


# Catgorizing Batters
```

Blue → Anchor
Red → Aggressive
Green → Mixed

```
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=batter_stats,
    x='boundary_pct',
    y='strike_rate',
    hue='style',
    palette={'Aggressive': 'red', 'Anchor': 'blue', 'Mixed': 'green'},
    s=100  # size of points
)
plt.title('Batting Styles: Strike Rate vs Boundary %')
plt.xlabel('Boundary %')
plt.ylabel('Strike Rate')
plt.grid(True)
plt.show()
```




```
import plotly.express as px

fig = px.scatter(
    batter_stats,
    x='boundary_pct',
    y='strike_rate',
    color='style',
    size='runs',
    hover_name='batter',
    title='Batting Styles: Anchor vs Aggressive',
    labels={
        'boundary_pct': 'Boundary %',
        'strike_rate': 'Strike Rate'
    },
    color_discrete_map={
        'Anchor': 'blue',
        'Aggressive': 'red',
        'Mixed': 'green'
    }
)
fig.update_traces(textposition='top center')
fig.show()
```

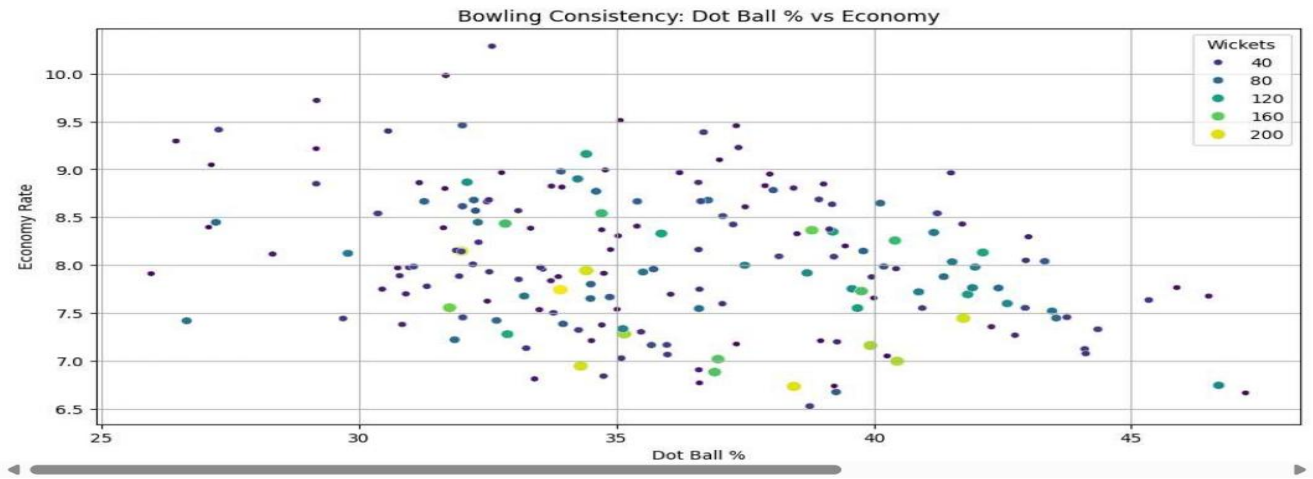


Study bowling consistency (dot balls, economy rate, average)

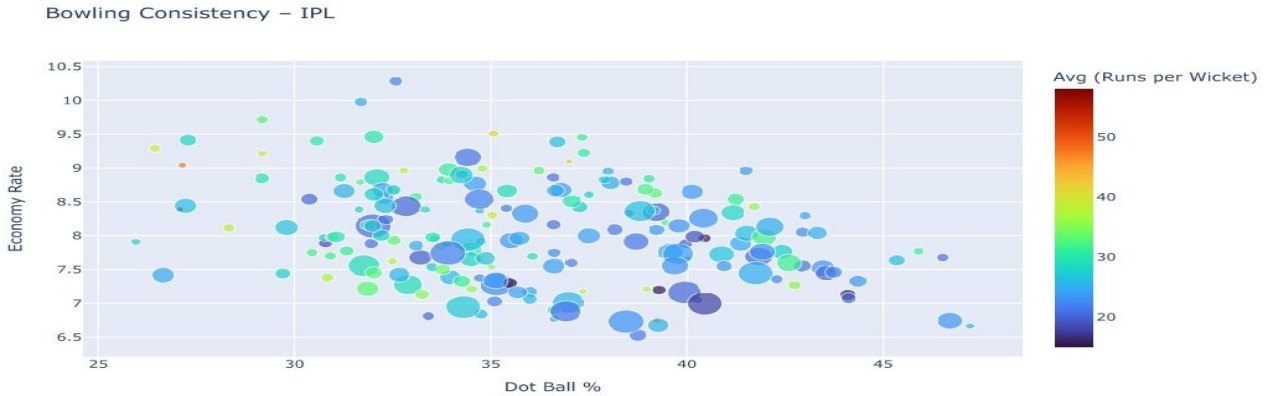
Key Metrics for Bowling Consistency

Metric	Formula
Dot Ball %	Dot Balls / Legal Deliveries * 100
Economy	Runs Conceded / Overs Bowled
Bowling Average	Runs Conceded / Wickets Taken (how costly each wicket was)

```
# Clean and Filter Legal Deliveries

# Filter legal deliveries (exclude wides and no-balls)
legal_deliveries = deliveries[
    deliveries['extras_type'].isna() |
    ~deliveries['extras_type'].isin(['wides', 'noballs'])
]


bowling_stats = legal_deliveries.groupby('bowler').agg(
    Balls_Bowled=('ball', 'count'),
    Runs_Conceded=('total_runs', 'sum'),
    Wickets=('is_wicket', 'sum'),
    Dot_Balls=('total_runs', lambda x: (x == 0).sum())
).reset_index()

# Compute final metrics
bowling_stats['Overs'] = bowling_stats['Balls_Bowled'] / 6
bowling_stats['Economy'] = bowling_stats['Runs_Conceded'] / bowling_stats['Overs']
bowling_stats['Bowling_Average'] = bowling_stats['Runs_Conceded'] / bowling_stats['Wickets']
bowling_stats['DotBall%'] = (bowling_stats['Dot_Balls'] / bowling_stats['Balls_Bowled']) * 100

# Filter: minimum 300 balls
bowling_stats = bowling_stats[bowling_stats['Balls_Bowled'] >= 300]


# Dot Ball % vs Economy (Seaborn)


plt.figure(figsize=(12, 6))
sns.scatterplot(
    data=bowling_stats,
    x='DotBall%',
    y='Economy',
    size='Wickets',
    hue='Wickets',
    palette='viridis',
    legend=True
)

plt.title('Bowling Consistency: Dot Ball % vs Economy')
plt.xlabel('Dot Ball %')
plt.ylabel('Economy Rate')
plt.grid(True)
plt.show()
```




```
fig = px.scatter(
    bowling_stats,
    x='DotBall%',
    y='Economy',
    size='Wickets',
    color='Bowling_Average',
    hover_name='bowler',
    title='Bowling Consistency - IPL',
    labels={
        'DotBall%': 'Dot Ball %',
        'Economy': 'Economy Rate',
        'Bowling_Average': 'Avg (Runs per Wicket)'
    },
    color_continuous_scale='Turbo'
)
fig.show()
```




```
# Bonus: Identify Top Consistent Bowlers

top_bowlers = bowling_stats.sort_values(by=['DotBall%', 'Economy'], ascending=[False, True]).head(10)
print(top_bowlers[['bowler', 'DotBall%', 'Economy', 'Bowling_Average', 'Wickets']])
```

	bowler	DotBall%	Economy	Bowling_Average	Wickets
147	GD McGrath	47.222222	6.666667	25.714286	14
138	DW Steyn	46.700275	6.745188	23.361905	105
84	BW Hilfenhaus	46.505376	7.677419	20.695652	23
170	IC Pandey	45.887446	7.766234	31.473684	19
417	S Sreesanth	45.340909	7.636364	26.046512	43
184	JC Archer	44.350581	7.330517	23.612245	49
132	DP Nannes	44.117647	7.077399	20.052632	38
117	DE Bollinger	44.097222	7.125000	15.906977	43
381	RJ Harris	43.750000	7.456731	22.000000	47
289	MM Patel	43.542435	7.447970	20.512195	82

Visualize performance in different overs (Powerplay, Death)

🎯 Goals:

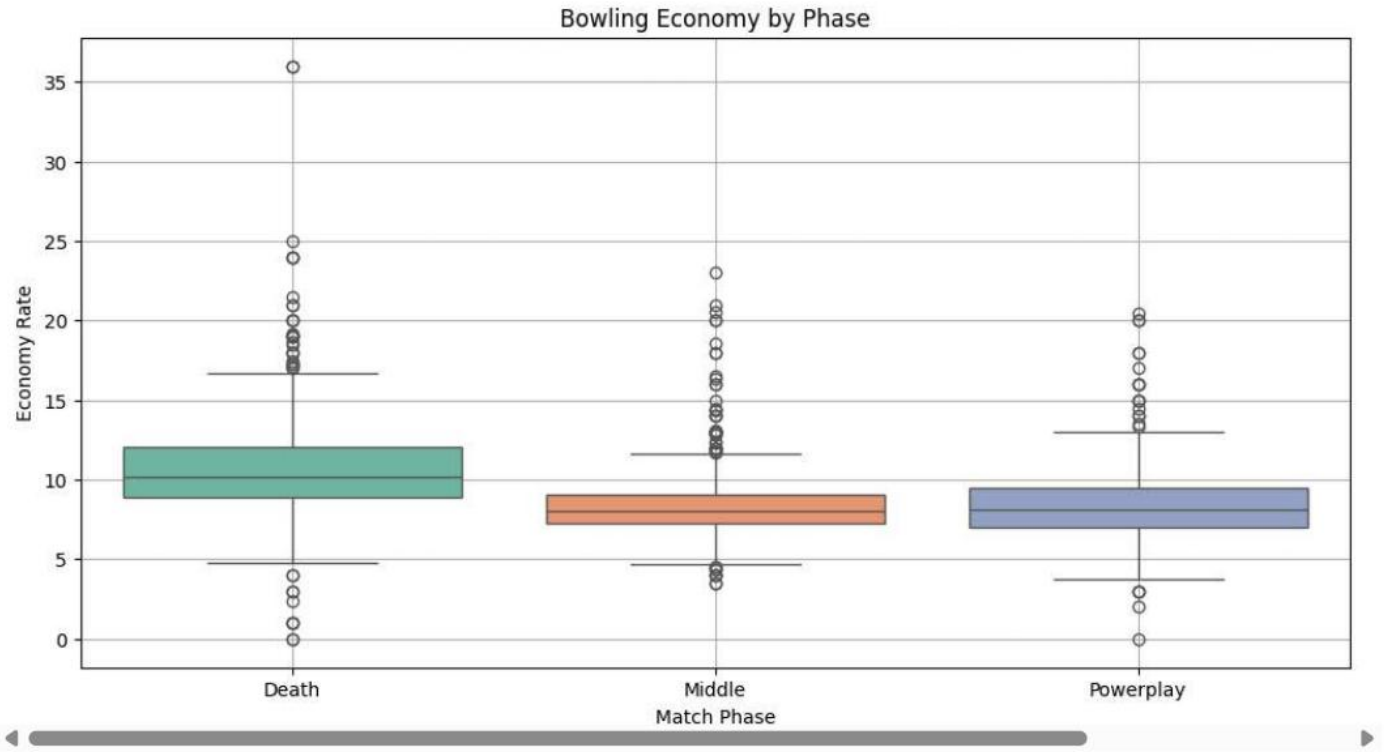
Classify overs into phases: Powerplay, Middle, Death

Calculate performance metrics per phase

Visualize: Heatmaps / Bar Charts / Bubble Charts

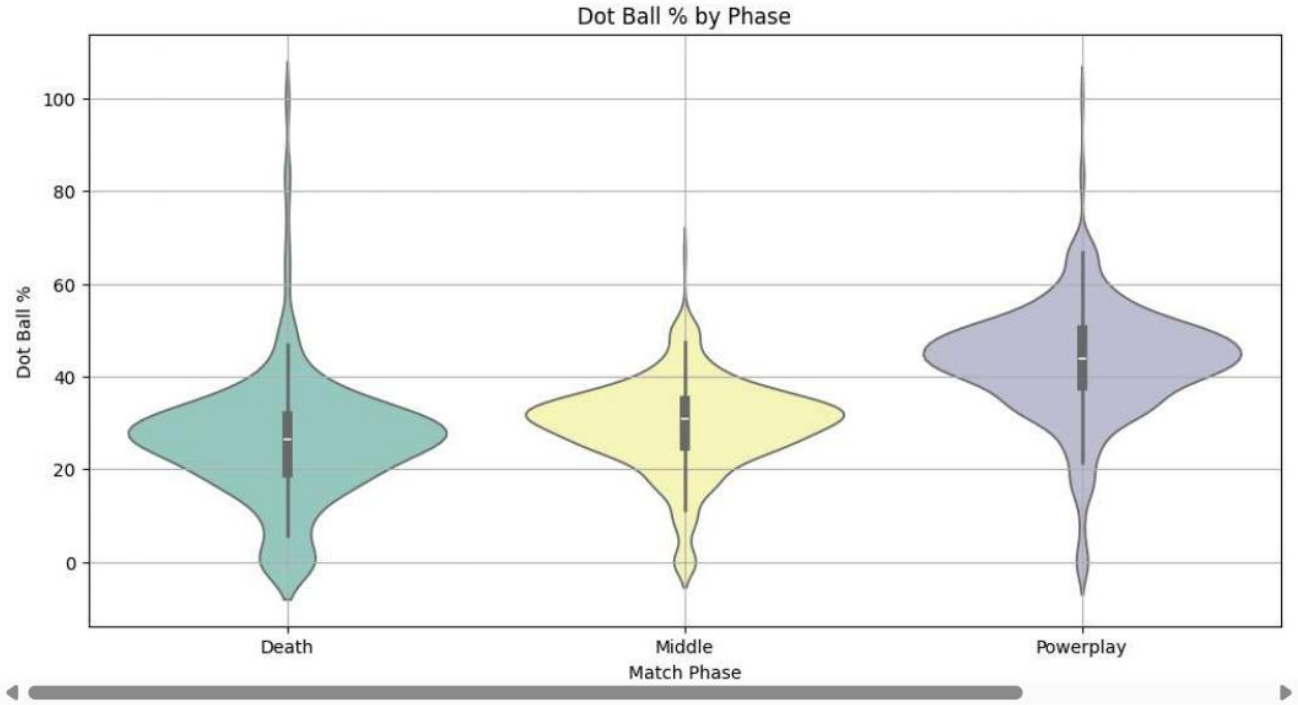
```
# Add Over Phase Column

def over_phase(over):
    if over < 6:
        return 'Powerplay'
    elif over < 16:
        return 'Middle'
    else:
        return 'Death'

deliveries['phase'] = deliveries['over'].apply(over_phase)


# Filter Legal Deliveries

legal_deliveries = deliveries[
    deliveries['extras_type'].isna() |
    ~deliveries['extras_type'].isin(['wides', 'noballs'])
]


 # Calculate Metrics per Phase

phase_stats = legal_deliveries.groupby(['bowler', 'phase']).agg(
    Balls=('ball', 'count'),
    Runs=('total_runs', 'sum'),
    Wickets=('is_wicket', 'sum'),
    DotBalls=('total_runs', lambda x: (x == 0).sum())
).reset_index()

# Add derived metrics
phase_stats['Overs'] = phase_stats['Balls'] / 6
phase_stats['Economy'] = phase_stats['Runs'] / phase_stats['Overs']
phase_stats['DotBall%'] = (phase_stats['DotBalls'] / phase_stats['Balls']) * 100


plt.figure(figsize=(12, 6))
sns.boxplot(data=phase_stats, x='phase', y='Economy', palette='Set2')
plt.title('Bowling Economy by Phase')
plt.ylabel('Economy Rate')
plt.xlabel('Match Phase')
plt.grid(True)
plt.show()
```

```
/tmp/ipython-input-69-1385549497.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le
```

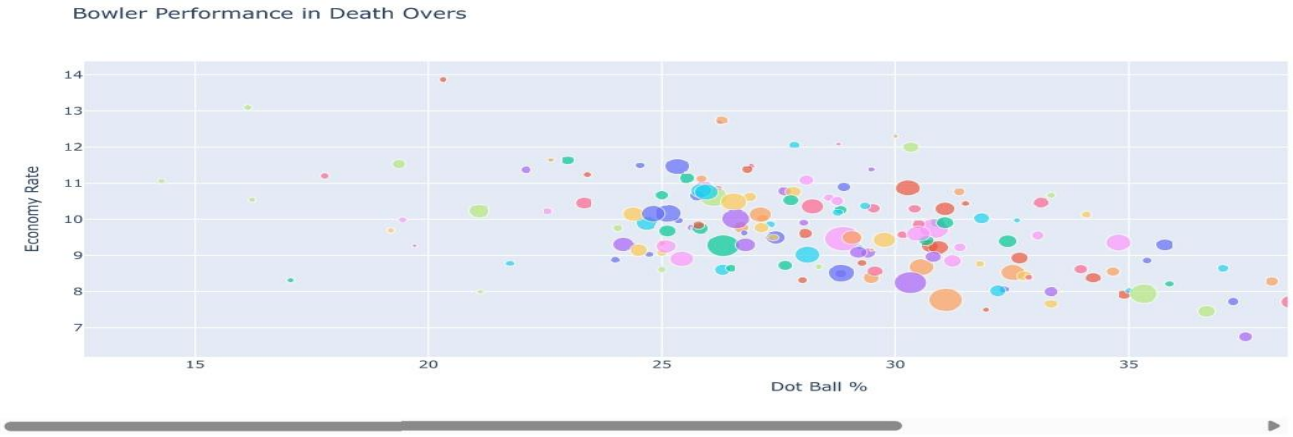



```
# Dot Ball %

plt.figure(figsize=(12, 6))
sns.violinplot(data=phase_stats, x='phase', y='DotBall%', palette='Set3')
plt.title('Dot Ball % by Phase')
plt.ylabel('Dot Ball %')
plt.xlabel('Match Phase')
plt.grid(True)
plt.show()
```

```
/tmp/ipython-input-70-1987724442.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le
```




```
# Interactive


death_stats = phase_stats[(phase_stats['phase'] == 'Death') & (phase_stats['Balls'] >= 60)]

fig = px.scatter(
    death_stats,
    x='DotBall%',
    y='Economy',
    size='Wickets',
    color='bowler',
    hover_name='bowler',
    title='Bowler Performance in Death Overs',
    labels={'DotBall%': 'Dot Ball %', 'Economy': 'Economy Rate'}
)
fig.update_traces(marker=dict(opacity=0.7))
fig.show()
```



Insights to Look For:

High Dot%, Low Econ 🎯 **Elite Death Bowler (e.g., Bumrah)**
Low Dot%, High Econ 🔴 **Leaky / less effective in pressure**
High Dot%, High Econ 🎲 **Defensive but not economical**

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

🎯 **Goal:**

**Analyze which venues consistently produce high scores (batting-friendly)
**

Identify low-scoring venues (bowling-friendly)

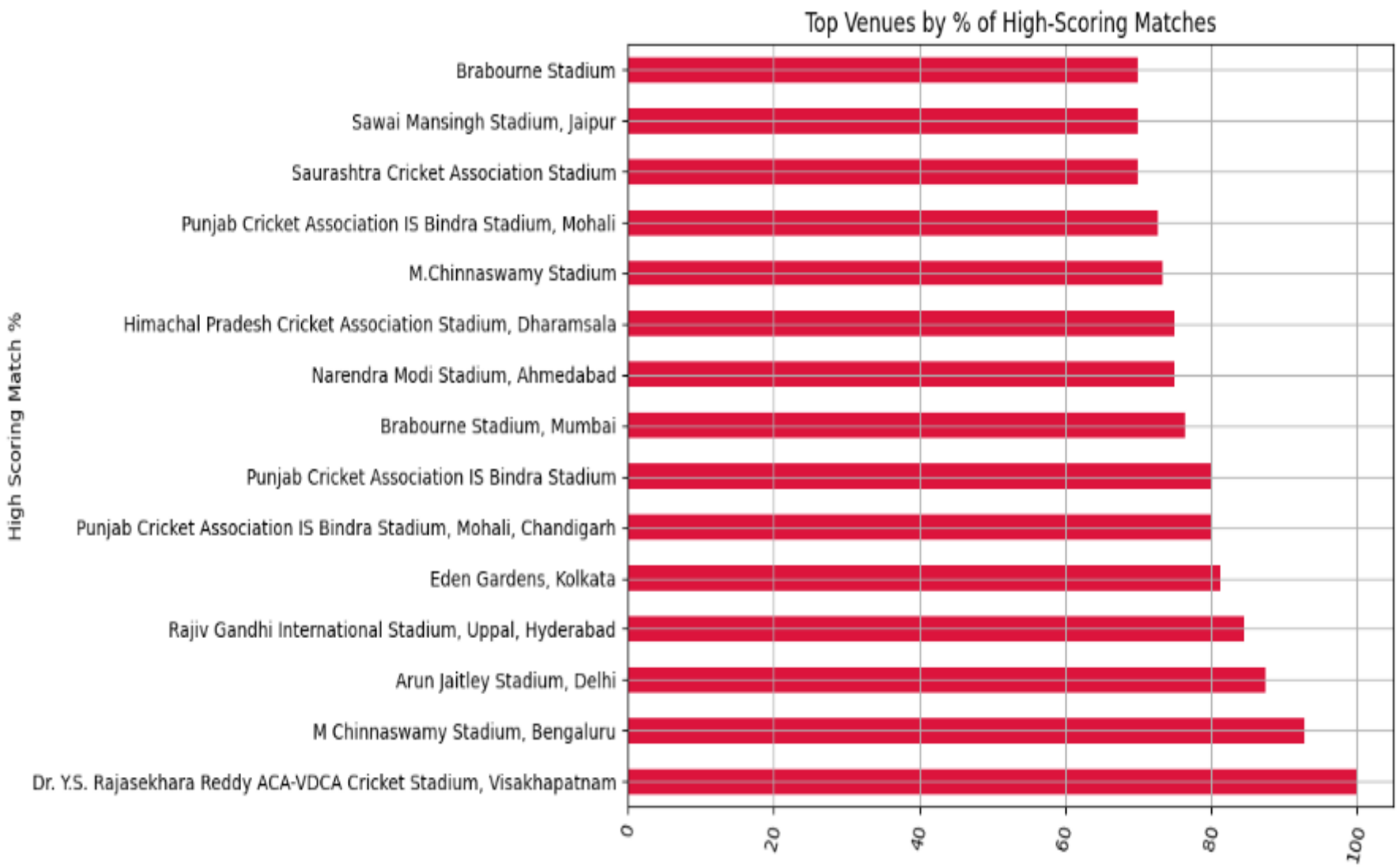
Classify matches into high and low scoring based on total match runs

```
# Merge to get venue & match data
df = deliveries.merge(
    matches[['id', 'venue']],
    left_on='match_id', right_on='id',
    how='left'
)
# Compute Match Total Runs

match_totals = df.groupby(['match_id', 'venue'])['total_runs'].sum().reset_index()
match_totals.rename(columns={'total_runs': 'match_runs'}, inplace=True)


# Classify as High vs Low Scoring

# Use a fixed threshold (e.g., 350) or median:

# Option 1: Based on median
threshold = match_totals['match_runs'].median()

# Option 2: Use fixed value
# threshold = 350

match_totals['score_type'] = match_totals['match_runs'].apply(lambda x: 'High' if x >= threshold else 'Low')


# Venue-wise Summary

venue_summary = match_totals.groupby(['venue', 'score_type']).size().unstack(fill_value=0)
```

```
# Venue-wise Summary

venue_summary = match_totals.groupby(['venue', 'score_type']).size().unstack(fill_value=0)


# Add total matches & percentage of high scoring games
venue_summary['Total_Matches'] = venue_summary.sum(axis=1)
venue_summary['HighScoring%'] = (venue_summary['High'] / venue_summary['Total_Matches']) * 100
```

:

```
# Visualize - Top 15 Venues by High Scoring %
# 📊 Matplotlib Bar Chart

top_venues = venue_summary.head(15)

plt.figure(figsize=(12,6))
top_venues['HighScoring%'].plot(kind='barh', color='crimson')
plt.title('Top Venues by % of High-Scoring Matches')
plt.ylabel('High Scoring Match %')
plt.xticks(rotation=75)
plt.grid(True)
plt.tight_layout()
plt.show()
```

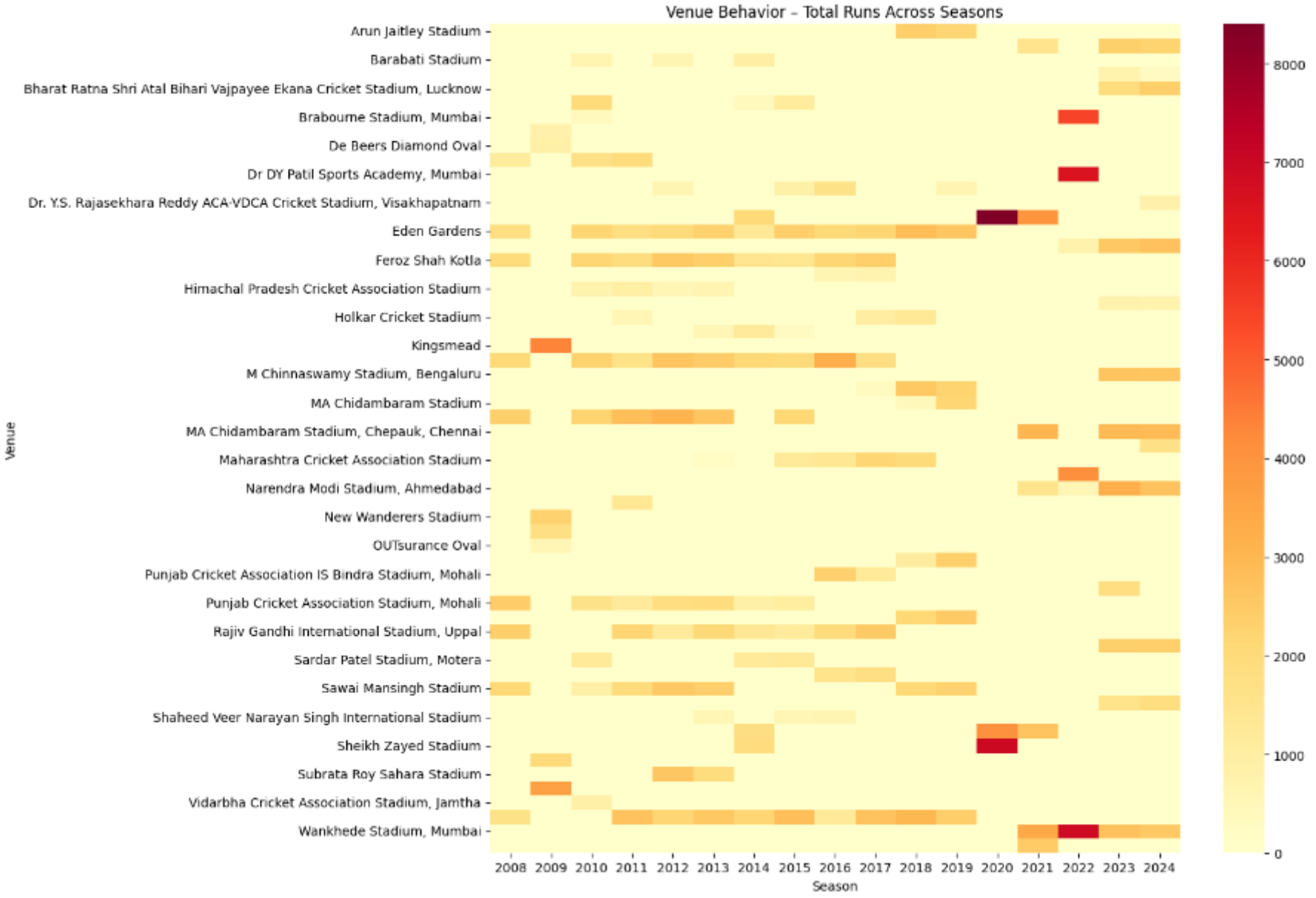

Top Venues by % of High-Scoring Matches
High Scoring Match %
Brabourne Stadium
Sawai Mansingh Stadium, Jaipur
Saurashtra Cricket Association Stadium
Punjab Cricket Association IS Bindra Stadium, Mohali
M.Chinnaswamy Stadium
Himachal Pradesh Cricket Association Stadium, Dharamsala
Narendra Modi Stadium, Ahmedabad
Brabourne Stadium, Mumbai
Punjab Cricket Association IS Bindra Stadium
Punjab Cricket Association IS Bindra Stadium, Mohali, Chandigarh
Eden Gardens, Kolkata
Rajiv Gandhi International Stadium, Uppal, Hyderabad
Arun Jaitley Stadium, Delhi
M Chinnaswamy Stadium, Bengaluru
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam
0
20
40
60
80
100

In [ ]:

```
fig = px.bar(
    venue_summary.reset_index(),
    x='venue',
    y='HighScoring%',
    title='Venue-wise High Scoring Match %',
    labels={'HighScoring%': 'High Scoring %'},
    color='HighScoring%',
    color_continuous_scale='Reds'
)
fig.update_layout(xaxis_tickangle=-45)
fig.show()
```

In [ ]:

```
# Bonus: Heatmap of Venue Behavior Over Years

# First, merge season info
df = df.merge(matches[['id', 'season']], left_on='match_id', right_on='id', how='left')

venue_season = df.groupby(['venue', 'season'])['total_runs'].sum().reset_index()

pivot = venue_season.pivot(index='venue', columns='season', values='total_runs').fillna(0)

# Plot heatmap
import seaborn as sns
plt.figure(figsize=(15,10))
sns.heatmap(pivot, cmap='YlOrRd')
plt.title('Venue Behavior - Total Runs Across Seasons')
plt.xlabel('Season')
plt.ylabel('Venue')
plt.tight_layout()
plt.show()
```


Venue Behavior – Total Runs Across Seasons
Arun Jaitley Stadium
Barabati Stadium
Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow
Brabourne Stadium, Mumbai
De Beers Diamond Oval
Dr DY Patil Sports Academy, Mumbai
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam
Eden Gardens
Feroz Shah Kotla
Himachal Pradesh Cricket Association Stadium
Holkar Cricket Stadium
Kingsmead
M Chinnaswamy Stadium, Bengaluru
MA Chidambaram Stadium
MA Chidambaram Stadium, Chepauk, Chennai
Maharashtra Cricket Association Stadium
Narendra Modi Stadium, Ahmedabad
New Wanderers Stadium
OUTsurance Oval
Punjab Cricket Association IS Bindra Stadium, Mohali
Punjab Cricket Association Stadium, Mohali
Rajiv Gandhi International Stadium, Uppal
Sardar Patel Stadium, Motera
Sawai Mansingh Stadium
Shaheed Veer Narayan Singh International Stadium
Sheikh Zayed Stadium
Subrata Roy Sahara Stadium
Vidarbha Cricket Association Stadium, Jamtha
Wankhede Stadium, Mumbai
Venue
2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024
Season
8000
7000
6000
5000
4000
3000
2000
1000
0

IPL Dataset EDA – Conclusion

1. **Run Trends Over the Years**: The average runs per match have gradually increased over the years, indicating a shift towards more aggressive batting.2018–2020 seasons saw some of the highest scoring games, possibly due to better pitches and deeper batting line-ups.

2. **Average Runs Per Season**: The average team score per inning hovers between 150–170 runs, with occasional peaks above 180 in recent seasons. This shows a trend toward high-scoring games , emphasizing the importance of strike rate and boundary-hitting.

3. **Venue Behavior**: Wankhede Stadium and Chinnaswamy Stadium consistently produce high-scoring matches due to small boundaries and flat pitches. Chepauk Stadium and Abu Dhabi tend to be low-scoring venues, favoring bowlers due to slow pitches. Venue impact is crucial for match strategies—teams often pick line-ups based on pitch behavior.

4. **Team Performance Insights:** Dominant teams like MI and CSK have maintained strong performance records, often adapting well to pitch and match conditions. * Some underperforming teams show inconsistency in both batting and bowling, affecting their overall stats.

5. **Toss and Match Outcome:** Toss winners tend to have a slight advantage, especially when choosing to chase. This trend is more prominent in night matches* where dew plays a role.

6. **High vs Low Scoring Matches:** High scoring matches (>180) are often decided by death over performance and power play acceleration.
Low scoring matches usually highlight
bowler-dominated games, where wickets in clusters change the course.

GIT HUB LINK

https://github.com/DipeshRai002/Python-EDA-/blob/main/Copy_of_IPL_Analysis_Using_Python_Pandas.ipynb


CHENNAI SUPER KINGS
MUMBAI INDIANS
ROYAL CHALLENGERS BANGALORE
#PLAYBOLD
SUNRISERS HYDERABAD
PUNJAB KINGS
KOLKATA Knight Riders
RAJASTHAN ROYALS
DELHI CAPITALS
THANK YOU
GUJARAT TITANS
LUCKNOW SUPER GIANTS