Problem Statement

This project aims to analyze historical IPL match data to uncover key insights about team performance, player statistics, and match outcomes. The goal is to identify winning strategies and impactful players using exploratory data analysis and visualizations.

Importing the Libraries

In [1]:
```
import pandas as pd
```

Loading the Dataset

In [2]:
```
ipl_2022 = pd.read_csv("ipl_2022_deliveries.csv")
ipl_2023 = pd.read_csv("ipl_2023_deliveries.csv")
ipl_2024 = pd.read_csv("ipl_2024_deliveries.csv")
ipl_2025 = pd.read_csv("ipl_2025_deliveries.csv")
```

In [3]:
```
ipl_2022.head()
```

Out[3]:

	match_id	season	match_no	date	venue	batting_team	bowling_team	i
0	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
1	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
2	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
3	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
4	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	

In [7]:
```
ipl_2022.shape
```

Out[7]:
```
(17912, 20)
```

In [7]:
```
ipl_2023.head()
```

Out[7]:

	match_id	season	match_no	date	venue	batting_team	bowling_team
0	202301	2023	1	Mar 31, 2023	Narendra Modi Stadium, Ahmedabad	CSK	GT
1	202301	2023	1	Mar 31, 2023	Narendra Modi Stadium, Ahmedabad	CSK	GT
2	202301	2023	1	Mar 31, 2023	Narendra Modi Stadium, Ahmedabad	CSK	GT
3	202301	2023	1	Mar 31, 2023	Narendra Modi Stadium, Ahmedabad	CSK	GT
4	202301	2023	1	Mar 31, 2023	Narendra Modi Stadium, Ahmedabad	CSK	GT

In [10]:
```
ipl_2023.shape
```

Out[10]: (17386, 20)

In [8]:
```
ipl_2024.head()
```

Out[8]:

	match_id	season	match_no	date	venue	batting_team	bowling_team
0	202401	2024	1	Mar 22, 2024	MA Chidambaram Stadium, Chennai	RCB	CSK
1	202401	2024	1	Mar 22, 2024	MA Chidambaram Stadium, Chennai	RCB	CSK
2	202401	2024	1	Mar 22, 2024	MA Chidambaram Stadium, Chennai	RCB	CSK
3	202401	2024	1	Mar 22, 2024	MA Chidambaram Stadium, Chennai	RCB	CSK
4	202401	2024	1	Mar 22, 2024	MA Chidambaram Stadium, Chennai	RCB	CSK

In [13]:
```
ipl_2024.shape
```

Out[13]: (17053, 20)

In [9]:
```
ipl_2025.head()
```

Out[9]:

	match_id	season	phase	match_no	date	venue	batting_team	bowling_te
0	202501	2025	Group Stage	1	Mar 22, 2025	Eden Gardens, Kolkata	KKR	
1	202501	2025	Group Stage	1	Mar 22, 2025	Eden Gardens, Kolkata	KKR	
2	202501	2025	Group Stage	1	Mar 22, 2025	Eden Gardens, Kolkata	KKR	
3	202501	2025	Group Stage	1	Mar 22, 2025	Eden Gardens, Kolkata	KKR	
4	202501	2025	Group Stage	1	Mar 22, 2025	Eden Gardens, Kolkata	KKR	

5 rows × 21 columns

In [14]:
```
ipl_2025.shape
```

Out[14]:
```
(17246, 21)
```

Checking for unique values in Phase column that is present only in IPL 2025

In [12]:
```
ipl_2025.phase.unique()
```

Out[12]:
```
array(['Group Stage', 'Qualifier 1', 'Eliminator', 'Qualifier 2', 'Final'],
      dtype=object)
```

Making a Copy or clon the 2025 file

In [12]:
```
ipl_2025_copy = ipl_2025.copy(deep = True)
```

Droping Phase column as it is not present in different files

In [15]:
```
ipl_2025.drop("phase",axis = 1,inplace = True)
```

Checking Shape of each file from 2022 to 2025

In [16]:
```
print(ipl_2022.shape)
print(ipl_2023.shape)
print(ipl_2024.shape)
print(ipl_2025.shape)
```

```
(17912, 20)
(17386, 20)
(17053, 20)
(17246, 20)
```

Combine all datasets into a single DataFrame

In [23]:
```
ipl_df = pd.concat([ipl_2022,ipl_2023,ipl_2024,ipl_2025],axis = 0,ignore_index
ipl_df.head()
```

Out[23]:

	match_id	season	match_no	date	venue	batting_team	bowling_team	i
0	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
1	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
2	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
3	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	
4	202201	2022	1	Mar 26, 2022	Wankhede Stadium, Mumbai	CSK	KKR	

In [24]:
```
ipl_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69597 entries, 0 to 69596
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   match_id          69597 non-null  int64
 1   season            69597 non-null  int64
 2   match_no          69597 non-null  int64
 3   date              69597 non-null  object
 4   venue             69597 non-null  object
 5   batting_team      69597 non-null  object
 6   bowling_team      69597 non-null  object
 7   innings           69597 non-null  int64
 8   over              69597 non-null  float64
 9   striker           69597 non-null  object
 10  bowler            69597 non-null  object
 11  runs_of_bat       69597 non-null  int64
 12  extras            69597 non-null  int64
 13  wide              69597 non-null  int64
 14  legbyes           69597 non-null  int64
 15  byes              69597 non-null  int64
 16  noballs           69597 non-null  int64
 17  wicket_type       3552 non-null   object
 18  player_dismissed  3546 non-null   object
 19  fielder           2796 non-null   object
dtypes: float64(1), int64(10), object(9)
memory usage: 10.6+ MB
```

🗒️ Columns Overview:

Column	Description
`match_id`	Unique ID for each match
`season`	IPL season/year
`match_no`	Match number of the season
`date`	Date of the match
`venue`	Stadium where match was played
`batting_team`	Team currently batting
`bowling_team`	Team currently bowling
`innings`	1 or 2
`over`	Ball over (e.g., 0.1 means 1st ball of 1st over)
`striker`	Batter on strike
`bowler`	Bowler delivering the ball
`runs_of_bat`	Runs scored off the bat

Column	Description
`extras`	Extra runs (wides, byes, etc.)
`wide` , `noballs`	Extras breakdown
`wicket_type`	Type of dismissal (if any)
`player_dismissed`	Name of dismissed player (if any)
`fielder`	Fielder involved in dismissal
`phase`	Match phase (Powerplay, Middle, Death)

🔍 Step-by-Step IPL Analysis Plan

Step 1: Basic Stats

- Total matches per season
- Total runs scored per season

Step 2: Batting Analysis

- Top 10 highest run-scorers (batters)
- Most sixes and fours by a player

Step 3: Bowling Analysis

- Top 10 wicket-takers
- Most economical bowlers (min. 10 overs)

Step 4: Team Analysis

- Total runs scored by each team
- Total wickets taken by each team

Step 5: Venue Insights

- Most runs scored at each venue
- Venue with highest average runs per match

In [25]:
```
match = ipl_df[['season', 'match_id']].drop_duplic
```

In [30]:
```
# Step 1: Basic Stats

# Total matches per season
matches_per_season = ipl_df[['season', 'match_id']].drop_duplicates().groupby(

# Total runs scored per season
ipl_df['total_runs'] = ipl_df['runs_of_bat'] + ipl_df['extras']
runs_per_season = ipl_df.groupby('season')['total_runs'].sum().reset_index().r

# Merge both stats into one summary
```

```
season_summary = matches_per_season.merge(runs_per_season, on='season')

season_summary
```

Out[30]:

	season	total_matches	total_runs_scored
0	2022	74	24395
1	2023	72	24994
2	2024	71	25921
3	2025	74	26493

📊 Insights:

The number of matches has been fairly consistent across the seasons.
Total runs scored have increased each year, indicating either more aggressive batting or more matches with higher scores.

In [33]:

```
# Step 2: Batting Analysis

# Grouping runs scored by striker (batter)
top_batters = (
    ipl_df.groupby('striker')['runs_of_bat']
    .sum()
    .reset_index()
    .sort_values(by='runs_of_bat', ascending=False)
    .head(10)
    .rename(columns={'runs_of_bat': 'total_runs'})
)

# Count of sixes by batter (run_of_bat == 6)
sixes = (
    ipl_df[ipl_df['runs_of_bat'] == 6]
    .groupby('striker')
    .size()
    .reset_index(name='sixes')
    .sort_values(by='sixes', ascending=False)
    .head(10)
)

# Count of fours by batter (run_of_bat == 4)
fours = (
    ipl_df[ipl_df['runs_of_bat'] == 4]
    .groupby('striker')
    .size()
    .reset_index(name='fours')
    .sort_values(by='fours', ascending=False)
    .head(10)
)
```

```
display(top_batters), display(sixes), display(fours)
```

	striker	total_runs
259	Shubman Gill	2449
114	Kohli	2378
42	Buttler	2152
198	Rahul	1956
270	Suryakumar Yadav	1937
94	Jaiswal	1877
314	du Plessis	1838
226	Sai Sudharsan	1793
184	Pooran	1687
66	Gaikwad	1663

	striker	sixes
116	Pooran	123
167	Shivam Dube	100
176	Suryakumar Yadav	98
28	Buttler	95
3	Abhishek Sharma	89
74	Klaasen	88
153	Samson	87
171	Shubman Gill	83
76	Kohli	81
146	Russell	80

	striker	fours
201	Shubman Gill	235
77	Jaiswal	225
92	Kohli	225
35	Buttler	214
210	Suryakumar Yadav	191
177	Sai Sudharsan	183
239	du Plessis	174
152	Rahul	171
74	Ishan Kishan	164
3	Abhishek Sharma	157

Out[33]: (None, None, None)

In [35]:
```
# Step 3: Bowling Analysis

# Filter only valid wickets (exclude run outs, retired hurt, etc.)
valid_wickets = ipl_df[ipl_df['wicket_type'].notna()]
valid_wickets = valid_wickets[~valid_wickets['wicket_type'].isin(['run out', '

# Top 10 wicket-takers
top_bowlers = (
    valid_wickets.groupby('bowler')
    .size()
    .reset_index(name='wickets')
    .sort_values(by='wickets', ascending=False)
    .head(10)
)

# Calculate economy rate = total runs conceded / total overs bowled
bowler_stats = ipl_df.groupby('bowler').agg({
    'total_runs': 'sum',
    'over': 'count'  # each delivery
}).rename(columns={'over': 'balls_bowled'})

bowler_stats['overs_bowled'] = bowler_stats['balls_bowled'] / 6
bowler_stats = bowler_stats[bowler_stats['overs_bowled'] >= 10]  # filter for

bowler_stats['economy'] = bowler_stats['total_runs'] / bowler_stats['overs_bow

# Top 10 most economical bowlers
top_economical = bowler_stats.sort_values(by='economy').head(10).reset_index()

display(top_bowlers), display(top_economical[['bowler', 'economy', 'overs_bowl
```

	bowler	wickets
28	Chahal	86
51	Harshal Patel	83
14	Arshdeep Singh	72
25	Boult	70
138	Rashid Khan	67
76	Kuldeep Yadav	63
24	Bhuvneshwar	61
17	Avesh Khan	61
163	Siraj	60
72	Khaleel Ahmed	58

	bowler	economy	overs_bowled
0	Bumrah	6.705508	157.333333
1	Willey	6.787500	26.666667
2	Narine	6.933440	207.833333
3	Chetan Sakariya	7.352113	11.833333
4	Santner	7.355236	81.166667
5	J Suchith	7.356522	19.166667
6	Krunal Pandya	7.493671	158.000000
7	Moeen Ali	7.503326	75.166667
8	M Theekshana	7.556604	35.333333
9	Varun Chakaravarthy	7.597403	51.333333

Out[35]: (None, None)

Step 4: Team Analysis

We'll cover two key insights:
Total runs scored by each team (as batting team)
Total wickets taken by each team (as bowling team)

1 Total Runs Scored by Each Team

In [38]:
```
# Sum of runs grouped by batting team
team_runs = (
    ipl_df.groupby('batting_team')['total_runs']
```

```
    .sum()
    .reset_index()
    .sort_values(by='total_runs', ascending=False)
    .rename(columns={'total_runs': 'total_runs_scored'})
)

print("Total Runs Scored by Each Team")
display(team_runs)
```

```
Total Runs Scored by Each Team
```

	batting_team	total_runs_scored
7	RCB	10709
2	GT	10642
8	RR	10613
5	MI	10454
6	PBKS	10440
9	SRH	10136
4	LSG	10129
0	CSK	9967
1	DC	9373
3	KKR	9340

2⃣ Total Wickets Taken by Each Team

In [40]:
```
# Valid wickets only (exclude run out, retired hurt, etc.)
valid_wickets = ipl_df[
    (ipl_df['wicket_type'].notna()) &
    (~ipl_df['wicket_type'].isin(['run out', 'retired hurt', 'obstructing the
]

# Count dismissals by bowling team
team_wickets = (
    valid_wickets.groupby('bowling_team')
    .size()
    .reset_index(name='total_wickets')
    .sort_values(by='total_wickets', ascending=False)
)

print("Total Wickets Taken by Each Team")
display(team_wickets)
```

```
Total Wickets Taken by Each Team
```

	bowling_team	total_wickets
2	GT	385
7	RCB	374
5	MI	371
0	CSK	356
3	KKR	354
8	RR	354
6	PBKS	351
1	DC	336
4	LSG	335
9	SRH	333

Step 5: Venue Insights

We'll analyze two things:
Total runs scored at each venue
Average runs per match at each venue

Total Runs Scored at Each Venue

In [42]:
```
venue_runs = (
    ipl_df.groupby('venue')['total_runs']
    .sum()
    .reset_index()
    .sort_values(by='total_runs', ascending=False)
    .rename(columns={'total_runs': 'total_runs_scored'})
)

print("Total Runs Scored at Each Venue")
display(venue_runs)
```

```
Total Runs Scored at Each Venue
```

	venue	total_runs_scored
17	Wankhede Stadium, Mumbai	14494
13	Narendra Modi Stadium, Ahmedabad	10207
6	Eden Gardens, Kolkata	8553
9	MA Chidambaram Stadium, Chennai	7442
2	Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...	7337
0	Arun Jaitley Stadium, Delhi	7312
8	M.Chinnaswamy Stadium, Bengaluru	7001
15	Rajiv Gandhi International Stadium, Hyderabad	6866
4	Dr DY Patil Sports Academy, Mumbai	6573
16	Sawai Mansingh Stadium, Jaipur	6087
3	Brabourne Stadium, Mumbai	5463
12	Maharashtra Cricket Association Stadium, Pune	4108
10	Maharaja Yadavindra Singh International Cricke...	3183
14	Punjab Cricket Association IS Bindra Stadium, ...	1856
1	Barsapara Cricket Stadium, Guwahati	1679
7	Himachal Pradesh Cricket Association Stadium, ...	1660
5	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...	1547
11	Maharaja Yadavindra Singh International Cricke...	435

Average Runs per Match at Each Venue

In [44]:
```
# First, find total matches at each venue
matches_per_venue = (
    ipl_df[['venue', 'match_id']]
    .drop_duplicates()
    .groupby('venue')
    .size()
    .reset_index(name='total_matches')
)

# Merge with total runs
venue_avg = venue_runs.merge(matches_per_venue, on='venue')
venue_avg['avg_runs_per_match'] = venue_avg['total_runs_scored'] / venue_avg['

# Sort by average runs
venue_avg_sorted = venue_avg.sort_values(by='avg_runs_per_match', ascending=Fa

print("Average Runs per Match at Each Venue")
display(venue_avg_sorted[['venue', 'avg_runs_per_match', 'total_matches']])
```

Average Runs per Match at Each Venue

	venue	avg_runs_per_match	total_matches
17	Maharaja Yadavindra Singh International Cricke...	435.000000	1
16	Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St...	386.750000	4
5	Arun Jaitley Stadium, Delhi	384.842105	19
2	Eden Gardens, Kolkata	371.869565	23
13	Punjab Cricket Association IS Bindra Stadium, ...	371.200000	5
6	M.Chinnaswamy Stadium, Bengaluru	368.473684	19
1	Narendra Modi Stadium, Ahmedabad	364.535714	28
7	Rajiv Gandhi International Stadium, Hyderabad	361.368421	19
9	Sawai Mansingh Stadium, Jaipur	358.058824	17
0	Wankhede Stadium, Mumbai	345.095238	42
10	Brabourne Stadium, Mumbai	341.437500	16
14	Barsapara Cricket Stadium, Guwahati	335.800000	5
4	Bharat Ratna Shri Atal Bihari Vajpayee Ekana C...	333.500000	22
15	Himachal Pradesh Cricket Association Stadium, ...	332.000000	5
8	Dr DY Patil Sports Academy, Mumbai	328.650000	20
3	MA Chidambaram Stadium, Chennai	323.565217	23
12	Maharaja Yadavindra Singh International Cricke...	318.300000	10
11	Maharashtra Cricket Association Stadium, Pune	316.000000	13

Step 6: Player vs Team Performance Breakdown

We'll analyze:
Total Runs a Player Has Scored Against Each Team
(Optional) Dismissals by a Bowler vs a Particular Team
Let’s begin with batsmen performance vs teams.

1 Total Runs Scored by Each Batsman vs Each Bowling Team

In [56]:
```
# Group by batsman and bowling_team to find total runs scored against each tea
```

```
player_vs_team = (
    ipl_df.groupby(['striker', 'bowling_team'])['runs_of_bat']
    .sum()
    .reset_index()
    .rename(columns={'striker': 'Player', 'bowling_team': 'Opponent', 'runs_of
    .sort_values(['Total_Runs'], ascending=[False])
)

# View the top 10 records
display(player_vs_team.head(10))
```

	Player	Opponent	Total_Runs
630	Kohli	PBKS	418
245	Buttler	RCB	357
1436	Shubman Gill	MI	356
626	Kohli	GT	351
357	Gaikwad	GT	350
1498	Suryakumar Yadav	PBKS	345
1439	Shubman Gill	RR	332
1437	Shubman Gill	PBKS	329
239	Buttler	DC	329
408	Hardik Pandya	RR	320

Bowlers vs Batsmen (Who dismissed whom and how many times?)

In [58]:
```
# Filter only rows where a dismissal occurred (excluding NaNs)
dismissals = ipl_df[ipl_df['player_dismissed'].notna()]

# Group by bowler and dismissed player
bowler_vs_batsman = (
    dismissals.groupby(['bowler', 'player_dismissed'])
    .size()
    .reset_index(name='dismissals')
    .sort_values(by='dismissals', ascending=False)
)

print("Top 10 Bowler vs Batsman Dismissals")
display(bowler_vs_batsman.head(10))
```

```
Top 10 Bowler vs Batsman Dismissals
```

	bowler	player_dismissed	dismissals
1739	Mukesh Kumar	Tilak Varma	4
339	Bhuvneshwar	Prabhsimran	4
365	Boult	Ayush Badoni	3
1833	Narine	Tripathi	3
397	Boult	Rutherford	3
2703	T Natarajan	Rinku Singh	3
2403	Sandeep Sharma	Pooran	3
1799	Narine	Ayush Badoni	3
391	Boult	Prabhsimran	3
2922	Vaibhav Arora	Ishan Kishan	3

Breakdown 2: Wicket Types per Team

See how each bowling team got their wickets — caught, bowled, LBW, etc.

In [61]:
```
# Filter out rows where a dismissal occurred
wicket_data = ipl_df[ipl_df['wicket_type'].notna()]

# Count of each dismissal type by bowling team
wicket_type_team = (
    wicket_data.groupby(['bowling_team', 'wicket_type'])
    .size()
    .reset_index(name='count')
    .sort_values(by='count', ascending=False)
)

print("Wicket Types by Bowling Team")
display(wicket_type_team.head(10))
```

```
Wicket Types by Bowling Team
```

	bowling_team	wicket_type	count
12	GT	caught	275
41	RCB	caught	265
35	PBKS	caught	257
17	KKR	caught	256
29	MI	caught	255
1	CSK	caught	250
47	RR	caught	247
22	LSG	caught	245
6	DC	caught	238
53	SRH	caught	236

🏏 Suggested Analysis: Over-by-Over Scoring Patterns

We'll analyze:
How teams score across each over (e.g., powerplay, middle overs, death overs)
Which overs are most productive or challenging
Team-wise performance trends across different overs

Analysis: Average Runs Scored Per Over by Team

This shows how each team performs on average in each over (1 to 20).

In [63]:
```
# Group by team and over, then calculate total and average runs
avg_runs_per_over = (
    ipl_df.groupby(['batting_team', 'over'])
    .agg(total_runs=('total_runs', 'sum'), balls=('over', 'count'))
    .reset_index()
)

# Calculate average runs per over
avg_runs_per_over['avg_runs'] = avg_runs_per_over['total_runs'] / avg_runs_per

# Optional: Pivot table for better readability
avg_runs_pivot = avg_runs_per_over.pivot(index='over', columns='batting_team',

# View top rows
display(avg_runs_pivot.head())
```

batting_team	CSK	DC	GT	KKR	LSG	MI	PBKS
over							
0.1	0.666667	1.135593	0.968750	0.898305	0.827586	0.571429	1.183333
0.2	0.968254	1.508475	0.983871	1.083333	0.813559	1.079365	1.225806
0.3	0.827586	1.214286	0.903226	1.233333	0.807018	1.000000	1.078125
0.4	1.065574	0.983871	1.031746	0.879310	0.816667	0.901639	1.196721
0.5	0.857143	1.157895	1.062500	1.017544	0.724138	0.938462	1.442623

🔍 Insights You Can Extract:

Which team dominates the powerplay (overs 1–6)?
Which team has the best death over (16–20) scoring rate?
Which over tends to produce the most runs league-wide?

1 Most Consistent Batsmen (Based on Average & Strike Rate)

🔍 Insight: Identify top-performing batsmen across seasons — who scores consistently and quickly?

Metrics:
Batting average = Total runs / number of dismissals
Strike rate = (Total runs / Balls faced) * 100
✅ You have required data:
striker, runs_of_bat, player_dismissed
📈 Ideal for leaderboard-style charts and team selection strategy.

In [68]:
```
# Filter only legal deliveries (exclude wides and no-balls for batsman stats)
legal_deliveries = ipl_df[(ipl_df['wide'].isna()) & (ipl_df['noballs'].isna())

# Group by batsman (striker) to get total runs and balls faced
batsman_stats = legal_deliveries.groupby('striker').agg(
    total_runs=('runs_of_bat', 'sum'),
    balls_faced=('striker', 'count'),
    dismissals=('player_dismissed', lambda x: x.notna().sum())
).reset_index()

# Calculate average and strike rate
batsman_stats['average'] = batsman_stats['total_runs'] / batsman_stats['dismis
batsman_stats['strike_rate'] = (batsman_stats['total_runs'] / batsman_stats['b

# Filter players with a minimum number of balls faced to ensure consistency
consistent_batsmen = batsman_stats[batsman_stats['balls_faced'] >= 100].sort_v
    by=['average', 'strike_rate'], ascending=False)
```

```
consistent_batsmen.head(10)
```

Out[68]:

	striker	total_runs	balls_faced	dismissals	average	strike_rate

In [ ]:

In [ ]:

In [ ]: