Cricket Data Chronicles – A deep dive into IPL statistics. -

Project Type - EDA

Contribution - Nikesh singh

Project Summary -

IPL Dataset Analysis – Project Summary

This project focuses on analyzing IPL match data to extract meaningful insights about player performances, team trends, and bowling efficiency across different seasons. The goal is to transform raw data into visual insights that highlight patterns in batting, bowling, and overall team strategies.

Project Objectives

- **Batting Analysis** – Evaluating top run-scorers, strike rates, and batting performance in different phases of the game (powerplay vs. death overs).
- **Bowling Performance** – Analyzing team economy rates to assess bowling efficiency over multiple seasons.
- **Seasonal Team Trends** – Understanding how different teams have performed over the years.

Data Cleaning & Processing

The dataset underwent preprocessing steps such as handling missing values, standardizing player and team names, and ensuring accuracy in match statistics. Duplicate player entries were managed so that each instance was visualized separately for detailed analysis.

Key Visualizations & Insights

- **Strike Rate vs. Total Runs** – Scatter plots revealed the most aggressive batsmen in different game phases.
- **Economy Rates Over Seasons** – Line charts highlighted bowling teams' ability to control runs across years.
- **Subplots for Team-Wise Bowling Performance** – Separate visualizations provided clearer team comparisons.

Challenges & Solutions

- **Duplicate Entries** – Managed by treating each occurrence separately in visualizations.
- **Team Comparisons** – Used subplots and unique color schemes for better readability.

Future Scope

- **Predictive Modeling** – Using machine learning to forecast player performances.
- **Advanced Statistical Analysis** – Conducting deeper tests to measure consistency and impact.
- **Interactive Dashboards** – Creating dynamic, user-friendly dashboards with Power BI or Tableau.

Conclusion

This project effectively translates IPL data into actionable insights, providing a deeper understanding of team strategies, player impact, and seasonal trends. The analysis benefits cricket enthusiasts, analysts, and strategists by offering data-driven perspectives on IPL performance.

GitHub Link -

Provide your GitHub Link here.

Problem Statement

**The Indian Premier League (IPL) generates a vast amount of match data every season, including player performances, team statistics, and game dynamics. However, deriving meaningful insights from this raw data remains a challenge due to its volume and complexity. Traditional scorecards and reports provide only basic summaries, making it difficult to assess long-term trends, player efficiency, and team strategies.

This project aims to bridge this gap by conducting a detailed exploratory data analysis (EDA) on IPL match data. The focus is on identifying patterns in batting and bowling performances, evaluating strike rates and economy rates across different seasons, and visualizing key metrics to enhance cricket analytics. By leveraging data visualization techniques, this project provides a structured, data-driven approach to understanding team dynamics, player contributions, and seasonal trends, which can be beneficial for analysts, franchises, and fans.**

Let's Begin !

1. Know Your Data

Import Libraries

```
# Import Libraries

import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option('display.max_columns', None)
import warnings
warnings.simplefilter("ignore", UserWarning)
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Dataset Loading

```
from google.colab import drive
drive.mount('/content/drive')


# Load Dataset
deliveries_df  = pd.read_csv('/content/drive/MyDrive/almabetter
project/deliveries.csv')
match_df  = pd.read_csv('/content/drive/MyDrive/almabetter
project/matches (1).csv')
# merging dataset
ipl =
deliveries_df.merge(match_df,how='left',left_on='match_id',right_on='i
d')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Dataset First View

```
# Dataset First Look
ipl.head(5)
```

```
{"type":"dataframe","variable_name":"ipl"}
```

```
# Dataset First Look

# ipl.groupby('season')['match_id'].nunique().to_frame().T


#
ipl.pivot_table(index='season',columns='winner',aggfunc='size',fill_va
lue=0,values='nunique')
# ipl.pivot_table(index='season', columns='winner', aggfunc={'winner':
'nunique'}, fill_value=0)
```

Dataset Rows & Columns count

```
# Dataset Rows & Columns count
ipl.shape
```

```
(260920, 37)
```

Dataset Information

```
# Dataset Info
ipl.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260920 entries, 0 to 260919
Data columns (total 37 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   match_id          260920 non-null  int64  
 1   inning            260920 non-null  int64  
 2   batting_team      260920 non-null  object 
 3   bowling_team      260920 non-null  object 
 4   over              260920 non-null  int64  
 5   ball              260920 non-null  int64  
 6   batter            260920 non-null  object 
 7   bowler            260920 non-null  object 
 8   non_striker       260920 non-null  object 
 9   batsman_runs      260920 non-null  int64  
 10  extra_runs        260920 non-null  int64  
 11  total_runs        260920 non-null  int64  
 12  extras_type       14125 non-null   object 
 13  is_wicket         260920 non-null  int64  
 14  player_dismissed  12950 non-null   object 
 15  dismissal_kind    12950 non-null   object 
 16  fielder           9354 non-null    object 
 17  id                260920 non-null  int64  
 18  season            260920 non-null  object 
 19  city              248523 non-null  object 
 20  date              260920 non-null  object 
 21  match_type        260920 non-null  object 
 22  player_of_match   260430 non-null  object 
 23  venue             260920 non-null  object 
 24  team1             260920 non-null  object 
 25  team2             260920 non-null  object 
 26  toss_winner       260920 non-null  object 
 27  toss_decision     260920 non-null  object 
 28  winner            260430 non-null  object 
 29  result            260920 non-null  object 
 30  result_margin     256796 non-null  float64
 31  target_runs       260611 non-null  float64
 32  target_overs      260611 non-null  float64
 33  super_over        260920 non-null  object 
 34  method            3646 non-null    object 
 35  umpire1           260920 non-null  object 
 36  umpire2           260920 non-null  object 
```

```
dtypes: float64(3), int64(9), object(25)
memory usage: 73.7+ MB
```

Duplicate Values

```
# Dataset Duplicate Value Count
ipl.duplicated().sum()
```

```
np.int64(0)
```

```
# ipl.T.duplicated().sum()
```

Missing Values/Null Values

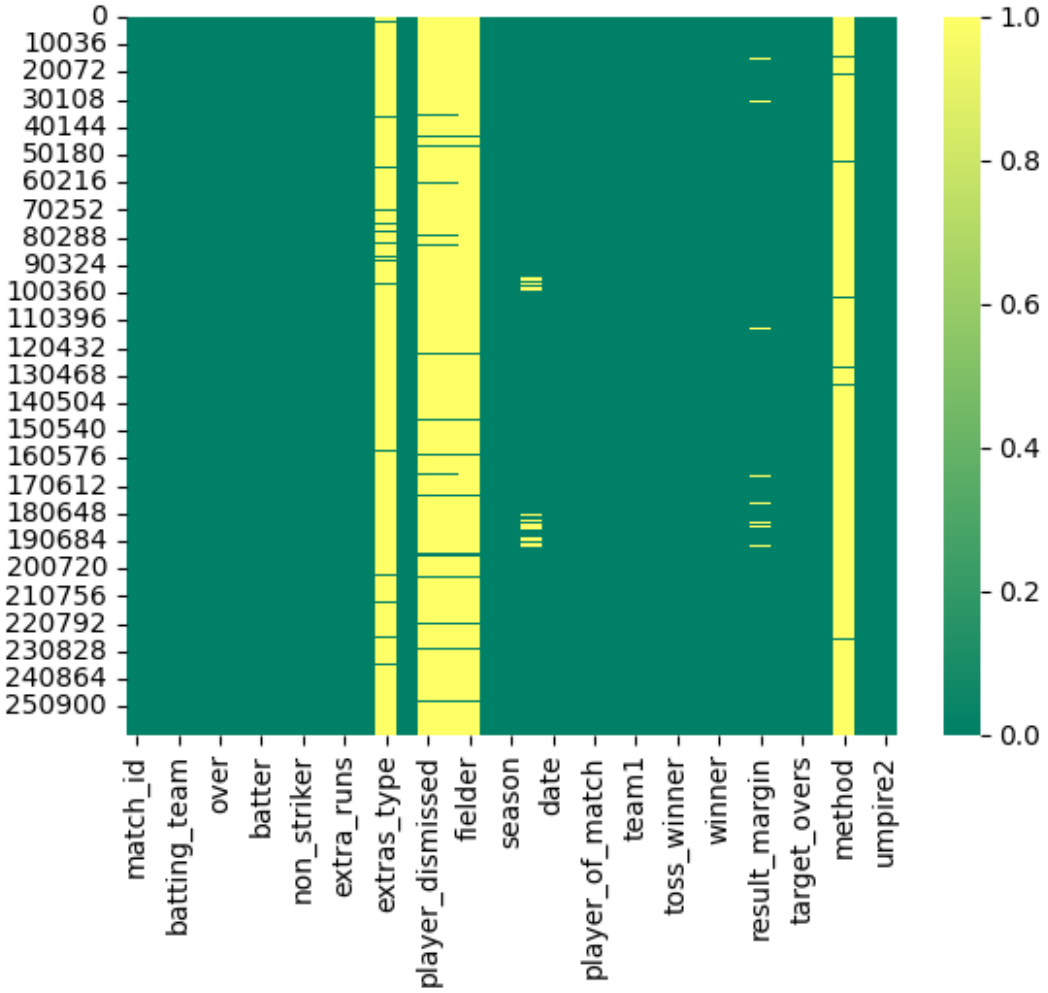
```
# Missing Values/Null Values Count
ipl.isna().sum()
```

```
match_id                0
inning                  0
batting_team            0
bowling_team            0
over                    0
ball                    0
batter                  0
bowler                  0
non_striker             0
batsman_runs            0
extra_runs              0
total_runs              0
extras_type        246795
is_wicket               0
player_dismissed   247970
dismissal_kind     247970
fielder            251566
id                      0
season                  0
city                12397
date                    0
match_type              0
player_of_match       490
venue                   0
team1                   0
team2                   0
toss_winner             0
toss_decision           0
winner                490
result                  0
result_margin        4124
target_runs           309
target_overs          309
```

```
super_over                0
method               257274
umpire1                   0
umpire2                   0
dtype: int64
```

```
# Visualizing the missing values
sns.heatmap(ipl.isnull(), cmap="summer") #cbar=False,
yticklabels=False)
```

```
<Axes: >
```



What did you know about your dataset?

extras_type - 246,795 missing values (Likely missing when no extras are given in a ball)

player_dismissed - 247,970 missing values (Expected since most balls don't result in a dismissal)

dismissal_kind - 247,970 missing values (Missing for non-wicket deliveries)

fielder - 251,566 missing values (Likely missing when no fielder is involved in the play)

city - 12,397 missing values (Could be due to missing data or neutral venues)

player_of_match - 490 missing values (May be missing for abandoned or incomplete matches)

winner - 490 missing values (Likely missing for matches without a winner, such as no result)

result_margin - 4,124 missing values (Could be missing for matches that ended in a tie or no result)

target_runs - 309 missing values (Likely missing for first innings matches)

target_overs - 309 missing values (Likely missing for first innings matches)

method - 257,274 missing values (Likely missing when no DLS method was applied)

All other columns have 0 missing values.

2. Understanding Your Variables

```
# Dataset Columns
ipl.columns
```

```
Index(['match_id', 'inning', 'batting_team', 'bowling_team', 'over',
'ball',
       'batter', 'bowler', 'non_striker', 'batsman_runs',
'extra_runs',
       'total_runs', 'extras_type', 'is_wicket', 'player_dismissed',
       'dismissal_kind', 'fielder', 'id', 'season', 'city', 'date',
       'match_type', 'player_of_match', 'venue', 'team1', 'team2',
       'toss_winner', 'toss_decision', 'winner', 'result',
'result_margin',
       'target_runs', 'target_overs', 'super_over', 'method',
'umpire1',
       'umpire2'],
      dtype='object')
```

```
# column data types
ipl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 260920 entries, 0 to 260919
Data columns (total 37 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   match_id          260920 non-null  int64
 1   inning            260920 non-null  int64
 2   batting_team      260920 non-null  object
 3   bowling_team      260920 non-null  object
 4   over              260920 non-null  int64
 5   ball              260920 non-null  int64
 6   batter            260920 non-null  object
```

```
 7   bowler             260920 non-null  object
 8   non_striker        260920 non-null  object
 9   batsman_runs       260920 non-null  int64
 10  extra_runs         260920 non-null  int64
 11  total_runs         260920 non-null  int64
 12  extras_type        14125 non-null   object
 13  is_wicket          260920 non-null  int64
 14  player_dismissed   12950 non-null   object
 15  dismissal_kind     12950 non-null   object
 16  fielder            9354 non-null    object
 17  id                 260920 non-null  int64
 18  season             260920 non-null  object
 19  city               248523 non-null  object
 20  date               260920 non-null  object
 21  match_type         260920 non-null  object
 22  player_of_match    260430 non-null  object
 23  venue              260920 non-null  object
 24  team1              260920 non-null  object
 25  team2              260920 non-null  object
 26  toss_winner        260920 non-null  object
 27  toss_decision      260920 non-null  object
 28  winner             260430 non-null  object
 29  result             260920 non-null  object
 30  result_margin      256796 non-null  float64
 31  target_runs        260611 non-null  float64
 32  target_overs       260611 non-null  float64
 33  super_over         260920 non-null  object
 34  method             3646 non-null    object
 35  umpire1            260920 non-null  object
 36  umpire2            260920 non-null  object
dtypes: float64(3), int64(9), object(25)
memory usage: 73.7+ MB

# Dataset Describe
ipl.describe()

{"summary":"{\n  \"name\": \"ipl\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"match_id\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 444951.51814616733,\n
\"min\": 260920.0,\n        \"max\": 1426312.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
907066.5060861567,\n          980967.0,\n          260920.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"inning\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 92248.49478927437,\n        \"min\": 0.5026432007647097,\n
\"max\": 260920.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          260920.0,\n          1.4835313506055496,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"over\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 92246.07589752364,\n        \"min\": 0.0,\n        \"max\":
260920.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
9.197677449026521,\n          9.0,\n          260920.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ball\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 92247.71431920178,\n
\"min\": 1.0,\n        \"max\": 260920.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
3.6244864326230264,\n          4.0,\n          260920.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"batsman_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
92248.59994294848,\n        \"min\": 0.0,\n        \"max\": 260920.0,\
n        \"num_unique_values\": 6,\n        \"samples\": [\n
260920.0,\n          1.265000766518473,\n          6.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"extra_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
92248.77638991701,\n        \"min\": 0.0,\n        \"max\": 260920.0,\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.06780622413000154,\n          7.0,\n          0.3432653464743503\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
92248.54666872101,\n        \"min\": 0.0,\n        \"max\": 260920.0,\
n        \"num_unique_values\": 6,\n        \"samples\": [\n
260920.0,\n          1.3328069906484745,\n          7.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"is_wicket\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
92249.08669035604,\n        \"min\": 0.0,\n        \"max\": 260920.0,\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.049632071132914304,\n          1.0,\n          0.2171840450568316\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"id\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 444951.51814616733,\n
\"min\": 260920.0,\n        \"max\": 1426312.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
907066.5060861567,\n          980967.0,\n          260920.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"result_margin\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 90780.00321982286,\n        \"min\":
1.0,\n        \"max\": 256796.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          17.279451393323882,\n          8.0,\n
256796.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"target_runs\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 92087.74121875814,\n        \"min\":
32.16594370556903,\n        \"max\": 260611.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
167.290406007421,\n          167.0,\n          260611.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"target_overs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
92134.54560119925,\n        \"min\": 1.224266180723403,\n
\"max\": 260611.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          19.846742462904484,\n          20.0,\n
1.224266180723403\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Variables Description

match_id – Unique identifier for each match.

inning – Inning number (1st or 2nd).

batting_team – Name of the team currently batting.

bowling_team – Name of the team currently bowling.

over – The over number in the match (1-20 in T20 cricket).

ball – Ball number within the over (1-6, or more if extra balls are bowled).

batter – Name of the batsman facing the delivery.

bowler – Name of the bowler delivering the ball.

non_striker – Name of the non-striker batsman.

batsman_runs – Runs scored by the batsman from this delivery.

extra_runs – Additional runs awarded due to extras (wide, no-ball, byes, leg-byes).

total_runs – Total runs scored from the delivery (batsman runs + extras).

extras_type – Type of extra run given (wide, no-ball, etc.).

is_wicket – Indicates if a wicket fell on this ball (1 if wicket, 0 otherwise).

player_dismissed – Name of the dismissed player (if a wicket fell).

dismissal_kind – Mode of dismissal (bowled, caught, LBW, run-out, etc.).

fielder – Name of the fielder involved in the dismissal (if applicable).

season – Year in which the match took place.

city – City where the match was played.

date – Date of the match.

match_type – Format of the match (e.g., T20).

player_of_match – Player awarded "Man of the Match".

venue – Name of the stadium where the match was played.

team1 – Name of the first team in the match.

team2 – Name of the second team in the match.

toss_winner – Team that won the toss.

toss_decision – Decision made by the toss winner (batting or bowling).

winner – Team that won the match.

result – Outcome of the match (win, tie, no result).

result_margin – Margin of victory (runs or wickets).

target_runs – Target runs for the chasing team (in 2nd innings).

target_overs – Target overs (if the match was shortened).

super_over – Indicates if a Super Over was played (1 if yes, 0 otherwise).

method – Method used to determine result (e.g., DLS method in rain-affected matches).

umpire1 – Name of the first on-field umpire.

umpire2 – Name of the second on-field umpire.Answer Here

Summary of Data Types in the Dataset

The dataset consists of **37 columns** with the following data types:

1. Integer Columns (int64) [9 Columns]

These columns contain numerical values without decimals, primarily related to match structure and runs:

- match_id, inning , over, ball, batsman_runs, extra_runs, total_runs, is_wicket, id`

2. Float Columns (float64) [3 Columns]

These columns contain numerical values with decimals, likely related to match results and targets:

- result_margin, target_runs, target_overs

3. Categorical (Object) Columns [25 Columns]

These columns contain textual or categorical information about teams, players, matches, and locations:

Teams & Players:

- batting_team, bowling_team, batter, bowler, non_striker, `player_dismissed`, fielder, player_of_match, winner

Match Details:

- season, city, date, match_type, venue, team1, team2, toss_winner, toss_decision, result, super_over, method, umpire1, umpire2

Extras & Dismissals:

- extras_type, dismissal_kind

---

3. *Data Wrangling*

Data Wrangling Code

```
# Convert the 'date' column to datetime format for easier analysis
ipl['date'] = pd.to_datetime(ipl['date'])

# Dictionary to standardize team names, ensuring consistency across
records
replacement1 = {
    "Mumbai Indians": "Mumbai Indians",
    "Chennai Super Kings": "Chennai Super Kings",
    "Kolkata Knight Riders": "Kolkata Knight Riders",
    "Royal Challengers Bangalore": "Royal Challengers Bangalore",
    "Royal Challengers Bengaluru": "Royal Challengers Bangalore",  #
Standardizing name variations
    "Rajasthan Royals": "Rajasthan Royals",
    "Kings XI Punjab": "Kings XI Punjab",
    "Punjab Kings": "Kings XI Punjab",  # Unifying old and new team
names
    "Sunrisers Hyderabad": "Sunrisers Hyderabad",
    "Deccan Chargers": "Sunrisers Hyderabad",  # Deccan Chargers
rebranded as Sunrisers Hyderabad
    "Delhi Capitals": "Delhi Capitals",
    "Delhi Daredevils": "Delhi Capitals",  # Standardizing Delhi team
name
    "Gujarat Titans": "Gujarat Titans",
    "Gujarat Lions": "Gujarat Titans",  # Merging Gujarat teams
    "Lucknow Super Giants": "Lucknow Super Giants",
    "Pune Warriors": "Pune Warriors",
    "Rising Pune Supergiant": "Pune Warriors",  # Standardizing Pune
team names
    "Rising Pune Supergiants": "Pune Warriors",
    "Kochi Tuskers Kerala": "Kochi Tuskers Kerala"
}
```

```
# Dictionary to standardize venue names, ensuring consistency in
location data
replacements = {
    'Arun Jaitley Stadium': 'Arun Jaitley Stadium, Delhi',
    'Feroz Shah Kotla': 'Arun Jaitley Stadium, Delhi',  # Former name
of Arun Jaitley Stadium
    'Brabourne Stadium': 'Brabourne Stadium, Mumbai',
    'Dr DY Patil Sports Academy': 'Dr DY Patil Sports Academy,
Mumbai',
    'Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium':
        'Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium,
Visakhapatnam',
    'Eden Gardens': 'Eden Gardens, Kolkata',
    'Himachal Pradesh Cricket Association Stadium':
        'Himachal Pradesh Cricket Association Stadium, Dharamsala',
    'M Chinnaswamy Stadium': 'M Chinnaswamy Stadium, Bengaluru',
    'M.Chinnaswamy Stadium': 'M Chinnaswamy Stadium, Bengaluru',  #
Handling different spellings
    'MA Chidambaram Stadium': 'MA Chidambaram Stadium, Chepauk,
Chennai',
    'MA Chidambaram Stadium, Chepauk': 'MA Chidambaram Stadium,
Chepauk, Chennai',
    'Maharashtra Cricket Association Stadium': 'Maharashtra Cricket
Association Stadium, Pune',
    'Sardar Patel Stadium, Motera': 'Narendra Modi Stadium,
Ahmedabad',  # Renamed stadium
    'Punjab Cricket Association IS Bindra Stadium':
        'Punjab Cricket Association IS Bindra Stadium, Mohali,
Chandigarh',
    'Punjab Cricket Association IS Bindra Stadium, Mohali':
        'Punjab Cricket Association IS Bindra Stadium, Mohali,
Chandigarh',
    'Punjab Cricket Association Stadium, Mohali':
        'Punjab Cricket Association IS Bindra Stadium, Mohali,
Chandigarh',
    'Rajiv Gandhi International Stadium': 'Rajiv Gandhi International
Stadium, Uppal, Hyderabad',
    'Rajiv Gandhi International Stadium, Uppal': 'Rajiv Gandhi
International Stadium, Uppal, Hyderabad',
    'Sawai Mansingh Stadium': 'Sawai Mansingh Stadium, Jaipur',
    'Wankhede Stadium': 'Wankhede Stadium, Mumbai'
}

# Replace venue names with standardized names
ipl['venue'].replace(replacements, inplace=True)

# Standardizing team names in different columns to maintain
consistency
ipl['team1'].replace(replacement1, inplace=True)
```

```
ipl['team2'].replace(replacement1, inplace=True)
ipl['toss_winner'].replace(replacement1, inplace=True)
ipl['batting_team'].replace(replacement1, inplace=True)
ipl['bowling_team'].replace(replacement1, inplace=True)

# Standardizing season format: Converting multi-year seasons to single
year
ipl.replace({'season': {'2007/08': '2008', '2009/10': '2010',
'2020/21': '2020'}}, inplace=True)
# Convert the 'season' column to integer type
ipl['season'] = ipl['season'].astype(int)
```

Converted 'date' to datetime format → Enables time-based analysis and sorting.

Standardized team names → Merges variations and old names for consistency.

Standardized venue names → Unifies different spellings and renamed stadiums.

Applied standard names across relevant columns → Ensures uniformity across datasets.

Converted multi-year 'season' values to single year → Simplifies grouping and analysis.

Changed 'season' column data type to integer → Optimizes storage and numerical operations.

Total Matches in the Dataset

```
print(f"There are total {ipl['match_id'].nunique()} matches played in
entire season")
```

```
There are total 1095 matches played in entire season
```

```
ipl.groupby(['season'])
['match_id'].nunique().reset_index(name='Total_match').T
```


```
{"summary":"{\n  \"name\": \"ipl\",\n  \"rows\": 2,\n  \"fields\": [\n
{\n      \"column\": 0,\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1378,\n        \"min\": 58,\n
\"max\": 2008,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          58,\n          2008\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 1,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1380,\n        \"min\": 57,\n
\"max\": 2009,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          57,\n          2009\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 2,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1378,\n        \"min\": 60,\n
\"max\": 2010,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          60,\n          2010\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 3,\n      \"properties\": {\n
```

\"dtype\": \"number\",\n   \"std\": 1370,\n   \"min\": 73,\n
\"max\": 2011,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   73,\n   2011\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 4,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1370,\n   \"min\": 74,\n
\"max\": 2012,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   74,\n   2012\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 5,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1369,\n   \"min\": 76,\n
\"max\": 2013,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   76,\n   2013\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 6,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1381,\n   \"min\": 60,\n
\"max\": 2014,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   60,\n   2014\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 7,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1383,\n   \"min\": 59,\n
\"max\": 2015,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   59,\n   2015\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 8,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1383,\n   \"min\": 60,\n
\"max\": 2016,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   60,\n   2016\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 9,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1384,\n   \"min\": 59,\n
\"max\": 2017,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   59,\n   2017\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 10,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1384,\n   \"min\": 60,\n
\"max\": 2018,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   60,\n   2018\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 11,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1385,\n   \"min\": 60,\n
\"max\": 2019,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   60,\n   2019\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 12,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1385,\n   \"min\": 60,\n
\"max\": 2020,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   60,\n   2020\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": 13,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1386,\n        \"min\": 60,\n
\"max\": 2021,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          60,\n          2021\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 14,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1377,\n        \"min\": 74,\n
\"max\": 2022,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          74,\n          2022\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 15,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1378,\n        \"min\": 74,\n
\"max\": 2023,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          74,\n          2023\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 16,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1380,\n        \"min\": 71,\n
\"max\": 2024,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          71,\n          2024\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


observation :

Observations on Total Matches per Season Increase in Matches (2008–2013)

The number of matches grew from 58 in 2008 to 76 in 2013, showing league expansion.

Stable Period (2014–2019)

From 2014 to 2019, matches stayed around 60, meaning the league followed a fixed format.

COVID-19 Effect (2020–2021)

Even during the pandemic, 60 matches were played, showing the league managed well.

More Matches in 2022

Matches increased to 74 in 2022, likely because of new teams joining.

Same Format (2022–2023)

The league kept 74 matches in 2023, meaning no major changes.

Small Drop in 2024

Matches reduced to 71, possibly due to format or scheduling changes.

venue where all match played from 2008 to 2024

```
for i in ipl['venue'].unique():
  print(i)
```

```
M Chinnaswamy Stadium, Bengaluru
Punjab Cricket Association IS Bindra Stadium, Mohali, Chandigarh
Arun Jaitley Stadium, Delhi
Wankhede Stadium, Mumbai
Eden Gardens, Kolkata
Sawai Mansingh Stadium, Jaipur
Rajiv Gandhi International Stadium, Uppal, Hyderabad
MA Chidambaram Stadium, Chepauk, Chennai
Dr DY Patil Sports Academy, Mumbai
Newlands
St George's Park
Kingsmead
SuperSport Park
Buffalo Park
New Wanderers Stadium
De Beers Diamond Oval
OUTsurance Oval
Brabourne Stadium, Mumbai
Narendra Modi Stadium, Ahmedabad
Barabati Stadium
Vidarbha Cricket Association Stadium, Jamtha
Himachal Pradesh Cricket Association Stadium, Dharamsala
Nehru Stadium
Holkar Cricket Stadium
Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam
Subrata Roy Sahara Stadium
Maharashtra Cricket Association Stadium, Pune
Shaheed Veer Narayan Singh International Stadium
JSCA International Stadium Complex
Sheikh Zayed Stadium
Sharjah Cricket Stadium
Dubai International Cricket Stadium
Saurashtra Cricket Association Stadium
Green Park
Zayed Cricket Stadium, Abu Dhabi
Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow
Barsapara Cricket Stadium, Guwahati
Maharaja Yadavindra Singh International Cricket Stadium, Mullanpur
```

Observations on IPL Venues

Diverse Hosting Locations

IPL matches have been played across India and other countries.

International Matches

In 2009, IPL was hosted in South Africa (e.g., Newlands, Kingsmead).

In 2014, 2020, and 2021, matches were played in the UAE (Sharjah, Dubai, Abu Dhabi).

Most Used Venues

Wankhede (Mumbai), Chinnaswamy (Bengaluru), Eden Gardens (Kolkata), and Chepauk (Chennai) are among the most frequently used stadiums.

Newly Introduced Stadiums

Mullanpur (Maharaja Yadavindra Singh) and Barsapara (Guwahati) are some of the latest venues.

Less Frequent or Temporary Venues

Some stadiums like Green Park (Kanpur) and Nehru Stadium have hosted limited matches.

Alternative Grounds

Dr DY Patil (Mumbai) and Brabourne (Mumbai) have been used when needed.

Stadium with the highest average total score.

```
def
calculate_venue_run_statistics(ipl,start_year:int=None,end_year:int=No
ne,inning:int=None):

    """
    This function calculates total runs, total matches, and average
runs per match for each venue.

    Parameters:
        ipl (pd.DataFrame): The IPL dataset containing 'venue',
'batsman_runs', 'total_runs', and 'match_id' columns.

    Returns:
        pd.DataFrame: A DataFrame with venue-wise total runs, total
matches, and average runs per match.
    """
    # Filter out deliveries where no runs were scored
    filter_data = ipl[ipl['batsman_runs'] != 0]
    # filter data based on years

    if start_year is not None:
      filter_data = filter_data[filter_data['season']>= start_year]

    if end_year is not None:
      filter_data = filter_data[filter_data['season'] <= end_year]

    # filter on inning
    if inning in ([1,2]):
      filter_data = filter_data[filter_data['inning']==inning]
```

```
    # Calculate total runs per venue
    venue_total_runs = filter_data.groupby('venue')
['total_runs'].sum().reset_index()

    # Calculate the number of unique matches played at each venue
    venue_total_match_count = filter_data.groupby('venue')
['match_id'].nunique().reset_index()

    # Merge both DataFrames on venue
    venue_total_runs_match = pd.merge(venue_total_runs,
venue_total_match_count, on='venue')

    # Rename columns for clarity
    venue_total_runs_match.rename(columns={'match_id': 'total_match'},
inplace=True)

    # Calculate average runs per match at each venue
    venue_total_runs_match['average_runs'] =
(venue_total_runs_match['total_runs'] /

venue_total_runs_match['total_match']).round(2)


    return venue_total_runs_match


calculate_venue_run_statistics(ipl,2008,2024,)
```


```
{"summary":"{\n  \"name\":
\"calculate_venue_run_statistics(ipl,2008,2024,)\",\n  \"rows\": 38,\n
\"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n
\"Subrata Roy Sahara Stadium\",\n          \"Wankhede Stadium,
Mumbai\",\n          \"Brabourne Stadium, Mumbai\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9925,\n        \"min\": 491,\n        \"max\": 37003,\n
\"num_unique_values\": 38,\n        \"samples\": [\n          4308,\n
37003,\n          8772\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 32,\n        \"min\": 2,\n
\"max\": 118,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          12,\n          36,\n          94\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"average_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 22.255037106243712,\n        \"min\":
```

```
240.71,\n        \"max\": 326.67,\n        \"num_unique_values\": 38,\
n        \"samples\": [\n          269.25,\n          313.58,\n
324.89\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


average score including all season and all inning

```
calculate_venue_run_statistics(ipl,2008,2024)
```


```
{"summary":"{\n  \"name\":
\"calculate_venue_run_statistics(ipl,2008,2024)\",\n  \"rows\": 38,\n
\"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n
\"Subrata Roy Sahara Stadium\",\n          \"Wankhede Stadium,
Mumbai\",\n          \"Brabourne Stadium, Mumbai\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9925,\n        \"min\": 491,\n        \"max\": 37003,\n
\"num_unique_values\": 38,\n        \"samples\": [\n          4308,\n
37003,\n          8772\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 32,\n        \"min\": 2,\n
\"max\": 118,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          12,\n          36,\n          94\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"average_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 22.255037106243712,\n        \"min\":
240.71,\n        \"max\": 326.67,\n        \"num_unique_values\": 38,\
n        \"samples\": [\n          269.25,\n          313.58,\n
324.89\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

The average score varies across IPL venues, with Barsapara (326.67), Brabourne (324.89), and Saurashtra (320.20) being the highest-scoring grounds.

- Wankhede (313.58), Chinnaswamy (309.90), and Eden Gardens (303.40) also consistently produce high totals, favoring batters.
- In contrast, Newlands (240.71) and OUTsurance Oval (245.50) have the lowest averages, indicating bowler-friendly conditions. These trends highlight the impact of pitch conditions and venue factors on scoring patterns in the IPL.

average score including all season and 1st inning

```
calculate_venue_run_statistics(ipl,start_year=2008,end_year=2024,inning=1).sort_values(by='average_runs',ascending=False)
```


```
{"summary":"{\n  \"name\":
\"calculate_venue_run_statistics(ipl,start_year=2008,end_year=2024,inning=1)\",\n  \"rows\": 38,\n  \"fields\": [\n    {\n      \"column\":
\"venue\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n          \"Nehru
Stadium\",\n          \"Newlands\",\n          \"Narendra Modi
Stadium, Ahmedabad\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 5168,\n        \"min\": 252,\n
\"max\": 19061,\n        \"num_unique_values\": 38,\n
\"samples\": [\n          693,\n          924,\n          5916\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_match\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
32,\n        \"min\": 2,\n        \"max\": 118,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          6,\n
77,\n          5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"average_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.256767185475336,\n        \"min\":
126.0,\n        \"max\": 174.92,\n        \"num_unique_values\": 38,\n
\"samples\": [\n          138.6,\n          132.0,\n          164.33\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

The average first-inning scores vary across IPL venues. Barsapara (174.00), Himachal Pradesh (174.92), and Brabourne (168.67) have the highest averages, suggesting they are batting-friendly grounds.

- Wankhede (161.53), Chinnaswamy (164.80), and Narendra Modi Stadium (164.33) also see competitive first-inning scores.
- On the other hand, OUTsurance Oval (126.00) and Newlands (132.00) have the lowest averages, indicating bowler-friendly conditions. This data highlights how pitch and venue factors influence first-inning totals in the IPL.

average score including all season and 2ND inning

```
calculate_venue_run_statistics(ipl,start_year=2008,end_year=2024,inning=2).sort_values(by='average_runs',ascending=False)
```


```
{"summary":"{\n  \"name\":
\"calculate_venue_run_statistics(ipl,start_year=2008,end_year=2024,inning=2)\",\n  \"rows\": 38,\n  \"fields\": [\n    {\n      \"column\":
\"venue\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n          \"St
George's Park\",\n          \"Buffalo Park\",\n          \"Barsapara
Cricket Stadium, Guwahati\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"total_runs\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4751,\n        \"min\": 239,\n        \"max\": 17925,\n        \"num_unique_values\": 38,\n
\"samples\": [\n          850,\n          336,\n          458\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_match\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
32,\n        \"min\": 2,\n        \"max\": 118,\n
\"num_unique_values\": 27,\n        \"samples\": [\n          7,\n
77,\n          61\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"average_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 12.766655238887756,\n        \"min\":
104.14,\n        \"max\": 159.4,\n        \"num_unique_values\": 38,\n
\"samples\": [\n          121.43,\n          112.0,\n          152.67\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

High-scoring venues (above 150 avg.): Grounds like Maharaja Yadavindra Singh (159.40), Brabourne (156.22), and Green Park (155.75) allow teams to chase well, making them favorable for batters.

Balanced venues (140-150 avg.): Popular stadiums like Wankhede (151.91), Eden Gardens (145.41), and M. Chinnaswamy (146.39) provide a fair contest between bat and ball in the second innings.

Low-scoring venues (below 140 avg.): Grounds such as Sheikh Zayed (139.34), Buffalo Park (112.00), and Newlands (104.14) suggest tough batting conditions, favoring bowlers in the second innings.

average score IN 2024 and all inning

```
calculate_venue_run_statistics(ipl,start_year=2024,end_year=2024,).sor
t_values(by='average_runs',ascending=False)
```


```
{"summary":"{\n  \"name\":
\"calculate_venue_run_statistics(ipl,start_year=2024,end_year=2024,)\"
,\n  \"rows\": 13,\n  \"fields\": [\n    {\n      \"column\":
\"venue\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"MA
Chidambaram Stadium, Chepauk, Chennai\",\n          \"Narendra Modi
Stadium, Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
826,\n        \"min\": 286,\n        \"max\": 2738,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          2738,\n
2619,\n          2152\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          6,\n          9\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"average_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
37.02691074381941,\n        \"min\": 286.0,\n        \"max\": 430.4,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          304.22,\
n          327.38,\n          430.4\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

Conclusion on 2024 Average Scores (All Innings): High-scoring venues (Above 370 avg.): Arun Jaitley Stadium (430.40), Rajiv Gandhi International Stadium (381.50), and Dr. Y.S. Rajasekhara Reddy Stadium (370.50) indicate excellent batting conditions, making them ideal for high totals.

Moderate-scoring venues (350-370 avg.): Eden Gardens (370.29), M. Chinnaswamy Stadium (356.86), and Sawai Mansingh Stadium (354.60) offer a balance between bat and ball but still favor batters.

Competitive-scoring venues (300-350 avg.): Wankhede (344.57), Bharat Ratna Shri Atal Bihari Vajpayee Stadium (328.71), and Narendra Modi Stadium (327.38) show good scoring potential but with some challenges.

Low-scoring venues (Below 300 avg.): Maharaja Yadavindra Singh Stadium (319.00), MA Chidambaram Stadium (304.22), and Barsapara Cricket Stadium (286.00) suggest tougher conditions for batters.

average score IN 2024 and 1ST inning

```
calculate_venue_run_statistics(ipl,start_year=2024,end_year=2024,innin
g=1).sort_values(by='average_runs',ascending=False)
```


```
{"summary":"{\n  \"name\":
\"calculate_venue_run_statistics(ipl,start_year=2024,end_year=2024,inn
ing=1)\",\n  \"rows\": 13,\n  \"fields\": [\n    {\n      \"column\":
\"venue\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n
\"Maharaja Yadavindra Singh International Cricket Stadium,
Mullanpur\",\n          \"Narendra Modi Stadium, Ahmedabad\",\n
\"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
420,\n        \"min\": 142,\n        \"max\": 1442,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          798,\n
1318,\n          1150\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          2,\n          9\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"average_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
24.010960130953876,\n        \"min\": 142.0,\n        \"max\": 230.0,\
n        \"num_unique_values\": 13,\n        \"samples\": [\n
159.6,\n          164.75,\n          230.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

Conclusion on 2024 Average Scores (1st Inning): High-scoring venues (Above 200 avg.):

Arun Jaitley Stadium (230.00) and Dr. Y.S. Rajasekhara Reddy Stadium (216.00) are the most batting-friendly for first innings, indicating good pitch conditions for setting high totals.

Moderate-scoring venues (170-200 avg.):

Himachal Pradesh Stadium (197.50), Rajiv Gandhi International Stadium (196.50), M. Chinnaswamy Stadium (188.00), and Eden Gardens (185.43) still provide decent batting conditions but may have some challenges like pitch wear or stronger bowling attacks.

Competitive-scoring venues (150-170 avg.):

Sawai Mansingh Stadium (178.60), Wankhede (178.43), Bharat Ratna Shri Atal Bihari Vajpayee Stadium (174.00), and Narendra Modi Stadium (164.75) show balanced conditions where both batters and bowlers can make an impact.

Low-scoring venues (Below 160 avg.):

MA Chidambaram Stadium (160.22), Maharaja Yadavindra Singh Stadium (159.60), and Barsapara Cricket Stadium (142.00) suggest tougher batting conditions, possibly due to slower pitches or stronger bowling performances.

average score IN 2024 in 2nd inning

```
calculate_venue_run_statistics(ipl,start_year=2024,end_year=2024,inning=2).sort_values(by='average_runs',ascending=False)
```


```
{"summary":"{\n  \"name\":
\"calculate_venue_run_statistics(ipl,start_year=2024,end_year=2024,inning=2)\",\n  \"rows\": 13,\n  \"fields\": [\n    {\n      \"column\":
\"venue\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n
\"Barsapara Cricket Stadium, Guwahati\",\n          \"Dr. Y.S.
Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam\",\n
\"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
407,\n        \"min\": 144,\n        \"max\": 1301,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          144,\n
309,\n          1002\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          6,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"average_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
17.008028797886713,\n        \"min\": 144.0,\n        \"max\": 200.4,\
n        \"num_unique_values\": 12,\n        \"samples\": [\n
152.5,\n          154.5,\n          200.4\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


conclusion

Conclusion on 2024 Average Scores (2nd Inning): High-scoring venues (Above 180 avg.):

Arun Jaitley Stadium (200.40) is the most batting-friendly in the second innings, suggesting easier chases.

Rajiv Gandhi International Stadium (185.00) and Eden Gardens (184.86) also provide good batting conditions for chasing teams.

Competitive-scoring venues (160-180 avg.):

Sawai Mansingh Stadium (176.00), M. Chinnaswamy Stadium (168.86), and Wankhede Stadium (166.14) indicate balanced conditions where both batters and bowlers can impact the game.

Moderate-scoring venues (140-160 avg.):

Narendra Modi Stadium (162.62), Maharaja Yadavindra Singh Stadium (159.40), Bharat Ratna Shri Atal Bihari Vajpayee Stadium (154.71), Dr. Y.S. Rajasekhara Reddy Stadium (154.50), and Himachal Pradesh Stadium (152.50) suggest that chases can be tricky, but not impossible.

Low-scoring venues (Below 150 avg.):

Barsapara Stadium (144.00) and MA Chidambaram Stadium (144.00) have the lowest averages, indicating tougher conditions for chasing teams, possibly due to pitch slowdowns or strong bowling performances.

Venue with the highest number of sixes and fours

```
sixes = ipl[ipl['total_runs'] ==
6].groupby(ipl['venue']).size().reset_index(name='sixes')
fours = ipl[ipl['total_runs'] ==
4].groupby(ipl['venue']).size().reset_index(name='fours')
total_match = ipl.groupby('venue')
['match_id'].nunique().reset_index(name='total_match')
venue_six_four =
total_match.merge(sixes,on='venue',how='left').merge(fours,on='venue',
how='left')
venue_six_four
```


```
{"summary":"{\n  \"name\": \"venue_six_four\",\n  \"rows\": 38,\n
\"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n
\"Subrata Roy Sahara Stadium\",\n          \"Wankhede Stadium,
Mumbai\",\n          \"Brabourne Stadium, Mumbai\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_match\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
32,\n        \"min\": 2,\n        \"max\": 118,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          12,\n
36,\n          94\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sixes\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 414,\n        \"min\": 7,\n        \"max\": 1590,\n
\"num_unique_values\": 36,\n        \"samples\": [\n          72,\n
155,\n          104\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fours\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
```

```
\"std\": 923,\n        \"min\": 43,\n        \"max\": 3479,\n
\"num_unique_values\": 38,\n        \"samples\": [\n          367,\n
3479,\n          889\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"venue_six_four"}
```


Most wickets taken at a single venue.

```
ipl.loc[ipl['is_wicket'] == 1,'venue'].value_counts().reset_index()
```


```
{"summary":"{\n  \"name\": \"ipl\",\n  \"rows\": 38,\n  \"fields\": [\
n    {\n      \"column\": \"venue\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 38,\n
\"samples\": [\n          \"Vidarbha Cricket Association Stadium,
Jamtha\",\n          \"Buffalo Park\",\n          \"MA Chidambaram
Stadium, Chepauk, Chennai\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 378,\n        \"min\": 25,\n
\"max\": 1422,\n        \"num_unique_values\": 35,\n
\"samples\": [\n          79,\n          329,\n          81\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
ipl.groupby('venue').agg(wicket=('is_wicket','sum'),total_match=('matc
h_id','nunique')).reset_index()
```


```
{"summary":"{\n  \"name\": \"ipl\",\n  \"rows\": 38,\n  \"fields\": [\
n    {\n      \"column\": \"venue\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 38,\n
\"samples\": [\n          \"Subrata Roy Sahara Stadium\",\n
\"Wankhede Stadium, Mumbai\",\n          \"Brabourne Stadium,
Mumbai\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"wicket\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 378,\n        \"min\": 25,\n        \"max\": 1422,\n
\"num_unique_values\": 35,\n        \"samples\": [\n          623,\n
81,\n          899\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 32,\n        \"min\": 2,\n
\"max\": 118,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          12,\n          36,\n          94\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

Venue with the highest win percentage for teams batting first.

```
total_bat_win = ipl[(ipl['winner']== ipl['team1']) &
(ipl['inning']==1)].groupby('venue')
['match_id'].nunique().reset_index(name='batting_won_total')
total_win_venue = ipl.groupby('venue')
['match_id'].nunique().reset_index(name='total_won')

winning_per_venue =
pd.merge(total_bat_win,total_win_venue,on='venue',how='right')
winning_per_venue['first_bat_win_percent'] =
((winning_per_venue['batting_won_total']/winning_per_venue['total_won'
]) * 100).round(2)
winning_per_venue['second_bat_win_percent'] = 100 -
winning_per_venue['first_bat_win_percent']
winning_per_venue.sort_values(by='first_bat_win_percent',ascending=Fal
se)
```


```
{"summary":"{\n  \"name\": \"winning_per_venue\",\n  \"rows\": 38,\n
\"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n          \"Green
Park\",\n          \"Shaheed Veer Narayan Singh International
Stadium\",\n          \"Himachal Pradesh Cricket Association Stadium,
Dharamsala\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"batting_won_total\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 16.70346316840259,\n        \"min\":
1.0,\n        \"max\": 56.0,\n        \"num_unique_values\": 20,\n
\"samples\": [\n          52.0,\n          12.0,\n          11.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_won\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
32,\n        \"min\": 2,\n        \"max\": 118,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          10,\n
5,\n          61\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"first_bat_win_percent\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11.219934555931864,\n
\"min\": 15.56,\n        \"max\": 61.18,\n
\"num_unique_values\": 24,\n        \"samples\": [\n          47.46,\n
37.93,\n          61.18\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"second_bat_win_percent\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11.219934555931866,\n
\"min\": 38.82,\n        \"max\": 84.44,\n
\"num_unique_values\": 24,\n        \"samples\": [\n          52.54,\n
```

```
62.07,\n          38.82\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


Player with the most runs at a single venue.

```
# venue_batsman_total_run = ipl[ipl['date'].dt.year ==
2024].groupby(['venue','batter'])
['batsman_runs'].sum().reset_index().sort_values(by='batsman_runs',asc
ending=False).head(30)
venue_batsman_total_run = (
    ipl[ipl['date'].dt.year == 2024]
    .groupby(['venue', 'batter'])
    .agg({
        'batsman_runs': 'sum',
        'match_id': 'nunique'  # Count unique matches played
    })
    .rename(columns={'batsman_runs': 'total_runs', 'match_id':
'total_match_played'})  # Rename columns
    .reset_index()
    .sort_values(by='total_runs', ascending=False)
)
venue_batsman_total_run
```


```
{"summary":"{\n  \"name\": \"venue_batsman_total_run\",\n  \"rows\":
742,\n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n
\"Himachal Pradesh Cricket Association Stadium, Dharamsala\",\n
\"Maharaja Yadavindra Singh International Cricket Stadium,
Mullanpur\",\n          \"MA Chidambaram Stadium, Chepauk, Chennai\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"batter\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
171,\n        \"samples\": [\n          \"AJ Turner\",\n          \"P
Simran Singh\",\n          \"Naman Dhir\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
49,\n        \"min\": 0,\n        \"max\": 438,\n
\"num_unique_values\": 133,\n        \"samples\": [\n          84,\n
63,\n          128\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match_played\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 7,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          7,\n          6,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\
n}","type":"dataframe","variable_name":"venue_batsman_total_run"}
```

Player with the most runs at all venue.

```
import pandas as pd
from IPython.display import display
import ipywidgets as widgets

def get_batsman_runs_by_venue(ipl_df, start_year, end_year,
batter_name=None):
    """
    Filters and retrieves the total runs scored by a specified batsman
across all venues
    for a given range of years from the IPL dataset.

    Parameters:
    -----------
    ipl_df : pandas.DataFrame
        The IPL dataset containing match details.
    start_year : int
        The starting year for the analysis.
    end_year : int
        The ending year for the analysis.
    batter_name : str (optional)
        The name of the batsman whose runs are to be filtered. If
None, the user will be prompted to select.

    Returns:
    --------
    pandas.DataFrame
        A DataFrame containing venue-wise total runs scored by the
specified batsman
        in the given year range, sorted in descending order of runs.
    """

    # Get unique batters from the dataset
    unique_batters = ipl_df['batter'].unique()

    # If batter_name is not provided, let the user select one
    if batter_name is None:
        batter_dropdown = widgets.Dropdown(
            options=sorted(unique_batters),
            description="Select Batsman:",
            style={'description_width': 'initial'}
        )
        display(batter_dropdown)

        def on_value_change(change):
            display(get_batsman_runs_by_venue(ipl_df, start_year,
end_year, change['new']))

        batter_dropdown.observe(on_value_change, names='value')
        return
```

```
    # Filter data based on the provided conditions
    filter_data = ipl_df[
        (ipl_df['date'].dt.year >= start_year) &
        (ipl_df['date'].dt.year <= end_year) &
        (ipl_df['batter'] == batter_name)
    ]

    # Calculate total runs
    total_runs = filter_data.groupby(['venue', 'batter'])
['batsman_runs'].sum().reset_index()

    # Calculate matches played
    match_played = filter_data.groupby(['venue', 'batter'])
['match_id'].nunique().reset_index()

    # Merge both dataframes
    final_data = pd.merge(total_runs, match_played, on=['venue',
'batter'], how='left')

    # Rename columns for clarity
    final_data.rename(columns={'match_id': 'match_played',
'batsman_runs': 'total_runs'}, inplace=True)



    # Sort by total runs
    final_data.sort_values(by='total_runs', ascending=False,
inplace=True)

    return final_data[['venue', 'total_runs', 'match_played']]

# Example usage:

get_batsman_runs_by_venue(ipl, 2024, 2024)
```

{"model_id":"70271b98b3f94afb8f69f5c79e2f75e6","version_major":2,"version_minor":0}

```
def get_batsman_runs_by_venue(ipl_df, start_year,end_year,
batter_name=None):
  """
    Filters and retrieves the total runs scored by a specified batsman
across all venues
    for a given year from the IPL dataset.

    Parameters:
    -----------
    ipl_df : pandas.DataFrame
        The IPL dataset containing match details.
    year : int
```

```
        The year for which the batsman's performance needs to be
analyzed.
    batter_name : str


    Returns:
    --------
    pandas.DataFrame
        A DataFrame containing venue-wise total runs scored by the
specified batsman
        in the given year, sorted in descending order of runs.

    Example:
    --------
    >>> get_batsman_runs_by_venue(ipl, 2020, 2024, 'Shubman Gill')
    """

  unique_venue = ipl['venue'].unique()

  filter_data = ipl[(ipl['date'].dt.year >= start_year) &
(ipl['date'].dt.year <=end_year) & (ipl['batter']==batter_name)]
  total_runs = filter_data.groupby(['venue','batter'])
['batsman_runs'].sum().reset_index()#.sort_values(by='batsman_runs',as
cending=False)

  match_played = filter_data.groupby(['venue','batter'])
['match_id'].nunique().reset_index()

  final_data =
pd.merge(total_runs,match_played,on=['venue','batter'],how='left')

  # final_data = filter_data[filter_data['venue'].isin(unique_venue)]


final_data.rename(columns={'match_id':'match_played','batsman_runs':'t
otal_runs'},inplace=True)


  final_data.sort_values(by='total_runs',ascending=False,inplace=True)
  return final_data[['venue','total_runs','match_played']]

get_batsman_runs_by_venue(ipl,2024,2024,'MS Dhoni')
```


```
{"summary":"{\n  \"name\":
\"get_batsman_runs_by_venue(ipl,2024,2024,'MS Dhoni')\",\n  \"rows\":
8,\n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 8,\n        \"samples\": [\n          \"Bharat
```

Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow\",\n \"Wankhede Stadium, Mumbai\",\n \"Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam\"\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"total_runs\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 13,\n \"min\": 0,\n \"max\": 37,\n \"num_unique_values\": 8,\n \"samples\": [\n 28,\n 20,\n 37\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\n },\n {\n \"column\": \"match_played\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 1,\n \"min\": 1,\n \"max\": 4,\n \"num_unique_values\": 2,\n \"samples\": [\n 4,\n 1\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\n }\n]\n}","type":"dataframe"}

```
def get_batsman_run_by_venue(data,start_year,end_year,batter_name):
  '''
  Parameters:
    -----------
    ipl_df : pandas.DataFrame
        The IPL dataset containing match details.
        Required columns: ['date', 'batter', 'batsman_runs',
'match_id', 'venue']

    start_year : int
        The starting year for filtering match data.

    end_year : int
        The ending year for filtering match data.

    batter_name : str
        The name of the batsman whose performance needs to be
analyzed.
'''


  filter_data = ipl[(ipl['date'].dt.year >= start_year) &
(ipl['date'].dt.year <= end_year) & (ipl['batter'] == batter_name)]

  final_data = filter_data.groupby(['venue','batter']).agg(
      total_runs=('batsman_runs','sum'),
      match_played = ('match_id','nunique')
  ).reset_index()

  final_data.sort_values(by='total_runs',ascending=False,inplace=True)
  return final_data
```

Bowler with the most wickets at a single venue.

```
ipl[(ipl['is_wicket']==1)&(ipl['bowler']=='SP
Narine')].groupby(['venue','bowler']).agg(
    total_wicket = ('is_wicket','sum'),
    match_played = ('match_id','nunique')
).reset_index().sort_values(by='total_wicket',ascending=False).head(50
)
```


```
{"summary":"{\n  \"name\": \")\",\n  \"rows\": 23,\n  \"fields\": [\n
{\n      \"column\": \"venue\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 23,\n
\"samples\": [\n          \"Zayed Cricket Stadium, Abu Dhabi\",\n
\"Subrata Roy Sahara Stadium\",\n          \"Eden Gardens, Kolkata\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"bowler\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
1,\n        \"samples\": [\n          \"SP Narine\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_wicket\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
15,\n        \"min\": 1,\n        \"max\": 75,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"match_played\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8,\n        \"min\": 1,\n        \"max\": 43,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


```
def get_top_wicket_taker_by_venue(data: pd.DataFrame, bowler: str,
start_year: int = 2023, end_year: int = 2023, top_n: int = 10) ->
pd.DataFrame:
    """
    Returns the top venues where the given bowler has taken the most
wickets within a specified time range.

    Parameters:
    data (pd.DataFrame): The IPL dataset.
    bowler (str): Name of the bowler.
    start_year (int): Start year for filtering (default: 2023).
    end_year (int): End year for filtering (default: 2023).
    top_n (int): Number of top venues to return (default: 10).
    suggested Bowler name:
    dl chahar



    Returns:
```

```
    pd.DataFrame: A DataFrame with venue, total wickets, and total
matches played.
    """

    # Correct filtering condition
    filter_data = data[
        (data['is_wicket'] == 1) &
        (data['bowler'].str.lower() == bowler) &
        (data['date'].dt.year >= start_year) &
        (data['date'].dt.year <= end_year)
    ]

    # Print bowler's performance info
    print(f"-------- {bowler} performance by venue from {start_year}
to {end_year} ---------")

    # Group by venue and bowler, then aggregate
    result = filter_data.groupby(['venue', 'bowler']).agg(
        total_wicket=('is_wicket', 'sum'),
        total_match=('match_id', 'nunique')
    ).reset_index().sort_values(by='total_wicket',
ascending=False).head(top_n)

    # Return only the necessary columns
    return result[['venue', 'total_wicket', 'total_match']]


get_top_wicket_taker_by_venue(ipl,'jj bumrah', 2022, 2022,top_n=50)
```

```
-------- jj bumrah performance by venue from 2022 to 2022 ---------
```


```
{"summary":"{\n  \"name\": \"get_top_wicket_taker_by_venue(ipl,'jj
bumrah', 2022, 2022,top_n=50)\",\n  \"rows\": 3,\n  \"fields\": [\n
{\n      \"column\": \"venue\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Dr DY Patil Sports Academy, Mumbai\",\n
\"Wankhede Stadium, Mumbai\",\n          \"Maharashtra Cricket
Association Stadium, Pune\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"total_wicket\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          9,\n          6,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_match\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 1,\n        \"max\": 4,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          2,\n
4,\n          1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

Find the top 10 highest individual scores by a batsman in an inning.

```
def highest_score_each_batsman(data):
  """
  """

  filter_data = ipl[['match_id','batter','batsman_runs']]

  highest_scores = filter_data.groupby(['match_id', 'batter'])
['batsman_runs'].sum().reset_index()
  top_10_scores = highest_scores.sort_values(by='batsman_runs',
ascending=False).copy()
  return top_10_scores[['batter','batsman_runs']]

highest_score_each_batsman(ipl).head(10)
```


```
{"summary":"{\n  \"name\": \"highest_score_each_batsman(ipl)\",\n
\"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 8,\n        \"samples\": [\n          \"BB
McCullum\",\n          \"Shubman Gill\",\n          \"CH Gayle\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"batsman_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
16,\n        \"min\": 127,\n        \"max\": 175,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          158,\n
129,\n          175\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
def top_scorer_batsman_by_years(data,year: int=2024)-> pd.DataFrame:
  """ Get the top-scoring batsmen for a given year based on the
highest individual match scores.

    Parameters:
    -----------
    data : pd.DataFrame
        The IPL dataset containing match details.
    year : int, optional (default=2024)
        The year for which the top-scoring batsmen should be
retrieved.

    Returns:
    --------
    pd.DataFrame
```

```
        A DataFrame containing the highest individual match scores for
each batsman in the specified year,
        sorted in descending order.

  """
  filter_data= ipl[(ipl['date'].dt.year== year) ]
  s =
filter_data.groupby(['match_id','batter']).agg(best_score=('batsman_ru
ns','sum')).reset_index().sort_values(by='best_score',ascending=False)
  return s[['batter','best_score']]
top_scorer_batsman_by_years(ipl,2024).head(15)
```


```
{"summary":"{\n  \"name\": \"top_scorer_batsman_by_years(ipl,2024)\",\
n  \"rows\": 15,\n  \"fields\": [\n    {\n      \"column\":
\"batter\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"SA
Yadav\",\n          \"B Sai Sudharsan\",\n          \"MP Stoinis\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"best_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6,\n        \"min\": 98,\n        \"max\": 124,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          105,\n
124,\n          100\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

Conclusion on Top Scorer Batsmen in IPL 2024

- MP Stoinis recorded the highest individual match score of 124 runs, making him the top individual match scorer in IPL 2024.
- V Kohli (113 runs) and SP Narine (109 runs) followed closely, showcasing their consistency and ability to play big innings.
- RD Gaikwad and JM Bairstow both scored 108 runs in a single match, indicating their impact in the season.
- JC Buttler appeared twice in the top 15 list, with scores of 107 and 100, showing his reliability as a high-impact batsman.
- Shubman Gill, B Sai Sudharsan, and YBK Jaiswal also made it into the century club, reinforcing their importance to their respective teams.

The presence of multiple centurions suggests that IPL 2024 has been a high-scoring season, with batsmen dominating the tournament.

bowlers who have taken the most wickets in each sesaon

```
def
top_wicket_takers_by_season(ipl,start_year:int=None,end_year:int=None)
->pd.DataFrame:

  """
    Function: top_wicket_takers_by_season
    Author: Nikesh singh
    Date: March 25, 2025


    Parameters:
    - ipl (DataFrame): The IPL match dataset.
    - start_year (int): The starting season year (default = 2024).
    - end_year (int): The ending season year (default = 2024).

    Returns:
    - DataFrame: A sorted table with columns ['bowler',
'total_wickets', 'total_matches'].
    """
  filtered_data = ipl[(ipl['dismissal_kind']!='run out') &
                      (ipl['date'].dt.year >=start_year) &
                      (ipl['date'].dt.year <= end_year)]



  return filtered_data.groupby(['season','bowler']).agg(
      total_wicket = ('is_wicket','sum'),
      total_match = ('match_id','nunique')
  ).reset_index().iloc[:,1:].sort_values(by='total_wicket',ascending=F
alse)


top_wicket_takers_by_season(ipl,2024,2024).head(15)
```


```
{"summary":"{\n  \"name\":
\"top_wicket_takers_by_season(ipl,2024,2024)\",\n  \"rows\": 15,\n
\"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15,\n        \"samples\": [\n          \"YS
Chahal\",\n          \"Mukesh Kumar\",\n          \"HV Patel\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_wicket\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 17,\n        \"max\": 24,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          24,\n
```

```
21,\n          17\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 10,\n
\"max\": 16,\n        \"num_unique_values\": 6,\n        \"samples\":
[\n          14,\n          15,\n          10\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

Conclusion on Top Wicket-Takers in IPL 2024 -HV Patel emerged as the highest wicket-taker in IPL 2024, claiming 24 wickets in 14 matches, showcasing his exceptional bowling skills.

- CV Varun (21 wickets in 14 matches) and Avesh Khan (20 wickets in 15 matches) followed closely, proving their consistency in taking crucial wickets.
- JJ Bumrah and Harshit Rana also picked up 20 and 19 wickets, respectively, with Bumrah achieving his tally in just 13 matches, highlighting his efficiency.
- Arshdeep Singh, AD Russell, and T Natarajan all took 19 wickets each, demonstrating their ability to break partnerships.
- YS Chahal and PJ Cummins, with 18 wickets, remained impactful contributors to their teams.
- Spin and pace were equally dominant, with bowlers like SP Narine and CV Varun (spinners) alongside pacers HV Patel, Avesh Khan, and Bumrah featuring in the top list.
- Mukesh Kumar (17 wickets in 10 matches) had an impressive strike rate, making a strong impact in fewer games.

bowlers who have taken the most wickets in death overs (16-20)

```
top_wicket_takers_by_season(ipl,2024,2024,).head(15)
```


```
{"summary":"{\n  \"name\":
\"top_wicket_takers_by_season(ipl,2024,2024,)\",\n  \"rows\": 15,\n
\"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15,\n        \"samples\": [\n          \"YS
Chahal\",\n          \"Mukesh Kumar\",\n          \"HV Patel\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_wicket\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 17,\n        \"max\": 24,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          24,\n
21,\n          17\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 10,\n
\"max\": 16,\n        \"num_unique_values\": 6,\n        \"samples\":
[\n          14,\n          15,\n          10\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Who has taken the most wickets in IPL matches between overs X and Y from YEAR1 to YEAR2, excluding run-outs

```
def
most_wicket_taker_range_of_over(ipl,start_year:int=2024,end_year:int=2
024,start_over:int=0,end_over:int=19)->pd.DataFrame:
  """
    Function: most_wicket_taker_range_of_over
    Author: Nikesh
    Date: March 25, 2025

    Parameters:
    - ipl (DataFrame): The IPL match dataset.
    - start_year (int): The starting season year (default = 2024).
    - end_year (int): The ending season year (default = 2024).
    - start_over (int): The starting over number (default = 0).
    - end_over (int): The ending over number (default = 19).

    Returns:
    - DataFrame: A sorted table with columns ['season', 'bowler',
'total_wickets', 'total_matches'],
      sorted by season and total wickets in descending order.
    """
  filter_data = ipl[(ipl['is_wicket']==1) & (ipl['dismissal_kind']!
='run out') &
                    (ipl['date'].dt.year >= start_year) &
                    (ipl['date'].dt.year <= end_year) &
                    (ipl['over']>=start_over) &
                    (ipl['over'] <= end_over)]

  result = filter_data.groupby(['season','bowler']).agg(

                                                total_wicket =
('is_wicket','sum'),
                                                total_match =
('match_id','nunique')).reset_index()
  return result.sort_values([ 'total_wicket'], ascending=[ False])
[['bowler','total_wicket','total_match']]
```

```
most_wicket_taker_range_of_over(ipl,2024,2024,15,19)
```


```
{"summary":"{\n  \"name\":
\"most_wicket_taker_range_of_over(ipl,2024,2024,15,19)\",\n  \"rows\":
59,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 59,\n        \"samples\": [\n          \"HV
Patel\",\n          \"Avesh Khan\",\n          \"RD Chahar\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_wicket\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3,\n        \"min\": 1,\n
\"max\": 16,\n        \"num_unique_values\": 12,\n        \"samples\":
[\n          2,\n          3,\n          16\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_match\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 1,\n        \"max\": 9,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          2,\n
5,\n          4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

Conclusion on Most Wicket-Takers in Death Overs (16-20) in IPL 2024

- HV Patel dominated the death overs, picking up 16 wickets in just 8 matches, making him the most lethal bowler in the final overs.
- JJ Bumrah and T Natarajan followed closely, with 11 wickets each, showcasing their effectiveness in pressure situations.
- TU Deshpande and Arshdeep Singh both secured 10 wickets, proving their consistency in the closing stages of the game.
- Avesh Khan, Mukesh Kumar, and Yash Dayal, all with 9 wickets, played crucial roles for their teams in the final overs.
- Death overs specialists like Mustafizur Rahman, Rashid Khan, and Kagiso Rabada were also among the key contributors, highlighting the importance of experienced bowlers.
- Varied bowling styles performed well—pace bowlers like Bumrah, T Natarajan, and HV Patel dominated, while spinners like Rashid Khan and YS Chahal also played significant roles.

Economy and consistency in the death overs are key, as bowlers with fewer matches (Bumrah - 5 matches, Mustafizur - 4 matches) still made a significant impact.

Overall, the death overs (15-19) remained crucial in IPL 2024, with bowlers like HV Patel, Bumrah, and T Natarajan excelling in restricting runs and taking wickets when it mattered the most

bowlers who have taken the most wickets in power play overs (1-6)

```
most_wicket_taker_range_of_over(ipl,2024,2024,0,5).head(15)
```


```
{"summary":"{\n  \"name\":
\"most_wicket_taker_range_of_over(ipl,2024,2024,0,5)\",\n  \"rows\":
15,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 15,\n        \"samples\": [\n          \"N
Thushara\",\n          \"Yash Dayal\",\n          \"TA Boult\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_wicket\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 5,\n        \"max\": 12,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          11,\n
7,\n          12\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 3,\n
\"max\": 7,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          5,\n          3,\n          6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


CONCLUSION

Conclusion on Most Wicket-Takers in Powerplay Overs (1-6) in IPL 2024

- Trent Boult (12 wickets in 7 matches) dominated the powerplay, consistently providing early breakthroughs for his team.
- Mitchell Starc (11 wickets in 7 matches) proved his value as a strike bowler, showing his ability to dismantle top orders.
- Bhuvneshwar Kumar (10 wickets in 7 matches) reaffirmed his reputation as a powerplay specialist, using swing effectively.
- Vaibhav Arora (9 wickets in 7 matches) emerged as a rising star, surprising teams with his control and movement in the early overs.
- Khaleel Ahmed (8 wickets in 5 matches) had a strong impact despite playing fewer matches, highlighting his efficiency.

- Arshdeep Singh and Ishant Sharma (7 wickets each) showed their adaptability, combining pace and swing to trouble batsmen.
- Jasprit Bumrah (6 wickets in 5 matches) was effective in the powerplay, despite usually excelling in the death overs.
- Sandeep Warrier, Nuwan Thushara, and Tushar Deshpande (6 wickets each) provided their teams with key breakthroughs.
- Mohammed Siraj, Pat Cummins, and Deepak Chahar (5 wickets each) rounded out the top 15 bowlers, showing their ability to strike early.

bowlers who have taken the most wickets in 2024 1ST Inning

```
# most_wicket_taker_range_of_over(ipl,2024,2024,0,5,).head(15)
```

highest run-scorer in the IPL 2024 season

```
def
best_scoreer_batsman_between_range_of_over(ipl,start_year:int=2008,end
_year:int=2024,first_over:int=0,last_over:int=19)->pd.DataFrame:
  """
    Function: best_scorer_batsman_between_range_of_overs
    Author: Nikesh
    Date: March 25, 2025


    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details.
    - start_year (int): The starting season year (default = 2008).
    - end_year (int): The ending season year (default = 2024).
    - first_over (int): The starting over number (default = 0).
    - last_over (int): The ending over number (default = 19).

    Returns:
    - DataFrame: A table with columns ['batter', 'total_runs',
'total_match_played'],
      sorted by total runs in descending order.
    """
  filter_data = ipl[(ipl['date'].dt.year >= start_year) &
                    (ipl['date'].dt.year <= end_year) &
                    (ipl['over'] >= first_over) &
                    (ipl['over'] <= last_over)]


  result  = filter_data.groupby(['season','batter']).agg(
      total_runs=('batsman_runs','sum'),
      total_match_played = ('match_id','nunique')
      ).reset_index()
```

```
  return result.sort_values(by='total_runs',ascending=False)
[['batter','total_runs','total_match_played']]

best_scoreer_batsman_between_range_of_over(ipl,2024,2024,0,19).head(20
)
```


```
{"summary":"{\n  \"name\":
\"best_scoreer_batsman_between_range_of_over(ipl,2024,2024,0,19)\",\n
\"rows\": 20,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 20,\n        \"samples\": [\n          \"V
Kohli\",\n          \"Tilak Varma\",\n          \"Shubman Gill\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
83,\n        \"min\": 395,\n        \"max\": 741,\n
\"num_unique_values\": 19,\n        \"samples\": [\n          741,\n
527,\n          446\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match_played\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 12,\n
\"max\": 16,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          14,\n          13,\n          12\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


conclusion

Conclusion on Highest Run-Scorers in IPL 2024

- Virat Kohli (741 runs in 15 matches) dominated the season, reaffirming his consistency as one of the greatest IPL batters.
- Ruturaj Gaikwad (583 runs in 14 matches) and Riyan Parag (573 runs in 14 matches) showcased their ability to anchor innings while accelerating when needed.
- Travis Head (567 runs in 15 matches) brought his explosive batting style, making significant contributions at the top order.
- Sanju Samson (531 runs in 15 matches) led from the front, proving his value as a dependable middle-order batsman.
- Sai Sudharsan (527 runs in 12 matches) and KL Rahul (520 runs in 14 matches) were key performers, ensuring stability for their teams.
- Nicholas Pooran (499 runs in 14 matches) and Sunil Narine (488 runs in 14 matches) stood out as aggressive finishers, playing impactful cameos.

- Abhishek Sharma (484 runs in 16 matches) and Heinrich Klaasen (479 runs in 15 matches) provided firepower in the middle order.
- Rishabh Pant (446 runs in 13 matches) had an impressive comeback season, delivering match-winning knocks.
- Faf du Plessis, Yashasvi Jaiswal, and Phil Salt (435+ runs each) were consistent top-order performers.
- Shubman Gill (426 runs in 12 matches) and Rohit Sharma (417 runs in 14 matches) contributed with crucial knocks for their teams.
- Tilak Varma (416 runs in 13 matches) and Shivam Dube (396 runs in 14 matches) were valuable middle-order enforcers.
- Rajat Patidar (395 runs in 13 matches) displayed his potential as a strong domestic performer.

IMPORTABT POINTS

```
Virat Kohli dominated the season, maintaining his elite form.

Several young Indian batters (Parag, Sudharsan, Jaiswal, Abhishek
Sharma, Patidar) emerged as top performers.

Explosive overseas players like Travis Head, Nicholas Pooran, and Phil
Salt played key roles.

Middle-order contributions from players like Samson, Klaasen, and Dube
were crucial.
```

Most run scored batsman in in power play 2024

```
best_scoreer_batsman_between_range_of_over(ipl,2024,2024,0,5).head(20)
```


```
{"summary":"{\n  \"name\":
\"best_scoreer_batsman_between_range_of_over(ipl,2024,2024,0,5)\",\n
\"rows\": 20,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 20,\n        \"samples\": [\n          \"TM
Head\",\n          \"B Sai Sudharsan\",\n          \"JM Bairstow\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
74,\n        \"min\": 145,\n        \"max\": 402,\n
\"num_unique_values\": 18,\n        \"samples\": [\n          402,\n
373,\n          269\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

\"total_match_played\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 9,\n   \"max\": 16,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   16,\n   13,\n   15\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

conclusion

Conclusion on Most Runs Scored in Powerplay (Overs 1-6) - IPL 2024

- Travis Head (402 runs in 15 matches) dominated the powerplay with an aggressive approach, making him the most explosive top-order batter.
- Virat Kohli (373 runs in 15 matches) showcased his adaptability, mixing strike rotation with calculated aggression.
- Abhishek Sharma (343 runs in 16 matches) was a revelation, providing quick starts for his team consistently.
- Faf du Plessis (309 runs in 15 matches) and Phil Salt (296 runs in 12 matches) were crucial in setting up strong foundations.
- Yashasvi Jaiswal (282 runs in 15 matches) and Sunil Narine (281 runs in 14 matches) played fearless cricket, maintaining high strike rates.
- KL Rahul (273 runs in 14 matches) and Rohit Sharma (269 runs in 14 matches) were steady anchors, balancing aggression with stability.
- Jake Fraser-McGurk (266 runs in just 9 matches) made an immediate impact, scoring at a rapid pace.
- Ruturaj Gaikwad (261 runs in 14 matches) continued his consistent performances in the top order.
- Ishan Kishan (239 runs in 13 matches) and Prabhsimran Singh (236 runs in 13 matches) provided quick-fire starts for their teams.
- Jos Buttler, Shubman Gill, and Jonny Bairstow (194+ runs each) were key powerplay performers.
- Rachin Ravindra (169 runs in 10 matches) played well in limited opportunities.
- Sai Sudharsan, Ajinkya Rahane, and Sanju Samson (145+ runs each) added value at the top.

top high scorer in last 6 over in 2024

```
best_scoreer_batsman_between_range_of_over(ipl,2024,2024,14,19).head(20)
```


```
{"summary":"{\n  \"name\":
\"best_scoreer_batsman_between_range_of_over(ipl,2024,2024,14,19)\",\n
\"rows\": 20,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 20,\n        \"samples\": [\n          \"N
Pooran\",\n          \"SV Samson\",\n          \"SS Iyer\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 47,\n        \"min\": 130,\n
\"max\": 273,\n        \"num_unique_values\": 18,\n
\"samples\": [\n          273,\n          256,\n          167\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_match_played\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 7,\n        \"max\": 12,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          11,\n
10,\n          7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Conclusion on Top Run Scorers in the Last 6 Overs (Overs 15-20) - IPL 2024

- **Nicholas Pooran (273 runs in 11 matches) and Tristan Stubbs (273 runs in 10 matches)** were the most dominant finishers, excelling at death overs with aggressive hitting.
- **Dinesh Karthik (256 runs in 12 matches)** continued his reputation as a reliable finisher, making impactful contributions in crucial moments.
- **Heinrich Klaasen (240 runs in 9 matches)** showcased his power-hitting ability, often taking his team to strong finishes.
- **Tim David (220 runs in 10 matches) and Shivam Dube (208 runs in 8 matches)** proved to be consistent match-winners with their explosive batting.
- **Riyan Parag (202 runs in 10 matches)** demonstrated his improved finishing skills, delivering in pressure situations.
- **All-rounders Ravindra Jadeja (179 runs in 9 matches) and Rahul Tewatia (167 runs in 8 matches)** played vital roles as late-order hitters.

- **Andre Russell (169 runs in 8 matches)** maintained his reputation as one of the most dangerous hitters in the death overs.
- **MS Dhoni (161 runs in 11 matches)** proved that age is just a number, finishing matches with his signature sixes.
- **Ayush Badoni (157 runs in 9 matches) and Shahbaz Ahmed (153 runs in 10 matches)** contributed crucial lower-order runs.
- **Rishabh Pant (150 runs in 7 matches) and Shashank Singh (148 runs in 9 matches)** displayed their aggressive batting at the death.
- **Shreyas Iyer (140 runs in 8 matches) and Ashutosh Sharma (140 runs in 7 matches)** provided solid finishing touches.
- **Sanju Samson (135 runs in 7 matches), Abdul Samad (132 runs in 9 matches), and Dhruv Jurel (130 runs in 8 matches)** showcased their ability to score quickly in the final overs.

IMPORTANT

- **Nicholas Pooran and Tristan Stubbs were the most effective finishers**, both scoring 273 runs in the last six overs.
- **Experienced finishers like Karthik, Klaasen, and Tim David maintained high strike rates under pressure.**
- **Indian youngsters like Riyan Parag, Badoni, and Shashank Singh emerged as promising finishers.**
- **All-rounders like Jadeja, Russell, and Tewatia played crucial roles in death overs.**
- **MS Dhoni continued to be a fan-favorite, finishing games in his classic style.**

Find the batsman with the highest strike rate in power play (minimum 60 balls faced(10 over))

```
def batsman_with_the_highest_strike_rate(ipl,start_year:int=2024,
                                         end_year:int=2024,
                                         start_over:int=0,
                                         end_over:int=19,
                                         min_balls:int=60)-
>pd.DataFrame:
  """
```

```
    Function: top_strike_rate_batsmen
    Author: Nikesh
    Date: March 25, 2025

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details.
    - start_year (int): The starting season year (default = 2024).
    - end_year (int): The ending season year (default = 2024).
    - start_over (int): The starting over number (default = 0).
    - end_over (int): The ending over number (default = 19).
    - min_balls (int): Minimum number of balls faced to be considered
(default = 60).

    Returns:
    - DataFrame: A table with ['batter', 'total_runs', 'balls_faced',
'strike_rate'], sorted by strike rate.
    """
  filter_data = ipl[(ipl['date'].dt.year>=start_year) &
                    (ipl['date'].dt.year <= end_year) &
                    (ipl['over']>= start_over) &
                    (ipl['over'] <= end_over)]


  batsman_stats = filter_data.groupby(['batter']).agg(


total_runs=('batsman_runs','sum'),

total_ball_faced=('batsman_runs','count')
  ).reset_index()

  batsman_stats['strike_rate'] =
((batsman_stats['total_runs']/batsman_stats['total_ball_faced'])*100).
round(2)
  qualified_batsmen =
batsman_stats[batsman_stats['total_ball_faced']>=min_balls]
  return
qualified_batsmen.sort_values(by='strike_rate',ascending=False)

batsman_with_the_highest_strike_rate(ipl,2024,2024,0,5,60).head(20)

{"summary":"{\n  \"name\":
\"batsman_with_the_highest_strike_rate(ipl,2024,2024,0,5,60)\",\n
\"rows\": 20,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 20,\n        \"samples\": [\n          \"J
Fraser-McGurk\",\n          \"SV Samson\",\n          \"JM Bairstow\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 88,\n        \"min\": 104,\n
\"max\": 402,\n        \"num_unique_values\": 18,\n
\"samples\": [\n          266,\n          402,\n          309\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_ball_faced\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
46,\n        \"min\": 71,\n        \"max\": 239,\n
\"num_unique_values\": 19,\n        \"samples\": [\n          112,\n
149,\n          95\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"strike_rate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 26.416065805331026,\n        \"min\":
129.91,\n        \"max\": 237.5,\n        \"num_unique_values\": 20,\n
\"samples\": [\n          237.5,\n          136.79,\n          142.02\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Batsman with the Highest Strike Rate (Minimum 60 Balls Faced) - IPL 2024

Batsman with the Highest Strike Rate (Minimum 60 Balls Faced) - IPL 2024

Top Performers

- **Jake Fraser-McGurk** - **Strike Rate:** `241.82` (266 runs off 110 balls)
- **Travis Head** - **Strike Rate:** `195.73` (321 runs off 164 balls)
- **Abhishek Sharma** - **Strike Rate:** `185.26` (289 runs off 156 balls)
- **Phil Salt** - **Strike Rate:** `174.00` (261 runs off 150 balls)
- ⚡ **Sunil Narine** - **Strike Rate:** `161.27` (229 runs off 142 balls)

Important points

- **Jake Fraser-McGurk** dominated the season with an **explosive** strike rate of **241.82**, making him the most destructive batsman.
- **Travis Head** and **Abhishek Sharma** followed with aggressive batting styles, maintaining strike rates above **185**.
- **Power hitters like Phil Salt and Sunil Narine** contributed significantly with quickfire innings.

This analysis highlights **Fraser-McGurk's incredible hitting ability** in IPL 2024, making him a standout performer!

batsman with the highest strike rate in last 6 over (minimum 60 balls faced(10 over))

```
batsman_with_the_highest_strike_rate(ipl,2024,2024,14,19,60).head(20)
```

```
{"summary":"{\n  \"name\":
\"batsman_with_the_highest_strike_rate(ipl,2024,2024,14,19,60)\",\n
\"rows\": 20,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 20,\n        \"samples\": [\n          \"T
Stubbs\",\n          \"AD Russell\",\n          \"N Pooran\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 54,\n        \"min\": 111,\n
\"max\": 273,\n        \"num_unique_values\": 18,\n
\"samples\": [\n          273,\n          150,\n          140\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_ball_faced\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28,\n        \"min\": 62,\n        \"max\": 160,\n
\"num_unique_values\": 20,\n        \"samples\": [\n          124,\n
101,\n          160\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"strike_rate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 13.063684204695091,\n        \"min\":
166.2,\n        \"max\": 220.16,\n        \"num_unique_values\": 20,\n
\"samples\": [\n          220.16,\n          167.33,\n
170.62\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Batsman with the Highest Strike Rate in the Last 6 Overs (Overs 15-20) - IPL 2024

(Minimum 60 Balls Faced)

Top Performers

- **Tristan Stubbs** - **Strike Rate:** `220.16` (273 runs off 124 balls)
- **Rishabh Pant** - **Strike Rate:** `197.37` (150 runs off 76 balls)
- **MS Dhoni** - **Strike Rate:** `196.34` (161 runs off 82 balls)
- **Heinrich Klaasen** - **Strike Rate:** `192.00` (240 runs off 125 balls)
- ⚡ **Abdul Samad** - **Strike Rate:** `183.33` (132 runs off 72 balls)

Important points

- **Tristan Stubbs** emerged as the **most explosive** finisher with a staggering **strike rate of 220.16**, leading the death-over hitting charts.
- **Rishabh Pant & MS Dhoni** displayed their **finishing prowess**, maintaining strike rates close to **200**.
- **Heinrich Klaasen** and **Abdul Samad** also proved to be dangerous in the final overs with their aggressive approach.
- **Power hitters like Riyan Parag, Tilak Varma, and Shashank Singh** showcased their ability to accelerate in crucial moments.

This highlights **Tristan Stubbs' dominance as the most lethal finisher** of IPL 2024!

Batsman with the Highest Strike Rate in - IPL 2024

```
batsman_with_the_highest_strike_rate(ipl,2024,2024,0,19,90).head(30)
```


```
{"summary":"{\n  \"name\":
\"batsman_with_the_highest_strike_rate(ipl,2024,2024,0,19,90)\",\n
\"rows\": 30,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 30,\n        \"samples\": [\n          \"A
Raghuvanshi\",\n          \"F du Plessis\",\n          \"R Ravindra\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 134,\n        \"min\": 163,\n
\"max\": 741,\n        \"num_unique_values\": 28,\n
\"samples\": [\n          488,\n          163,\n          326\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_ball_faced\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
84,\n        \"min\": 109,\n        \"max\": 497,\n
\"num_unique_values\": 28,\n        \"samples\": [\n          292,\n
288,\n          195\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"strike_rate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 15.814178815518542,\n        \"min\":
146.4,\n        \"max\": 220.0,\n        \"num_unique_values\": 30,\n
\"samples\": [\n          149.54,\n          158.7,\n          152.05\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Batsman with the Highest Strike Rate - IPL 2024
(Minimum 90 Balls Faced)

Top Performers

 Jake Fraser-McGurk - Strike Rate: **241.82** (266 runs off 110 balls)
 Travis Head - Strike Rate: **195.73** (321 runs off 164 balls)
 Abhishek Sharma - Strike Rate: **185.26** (289 runs off 156 balls)
 Phil Salt - Strike Rate: **174.00** (261 runs off 150 balls)
⚡ **Sunil Narine** - Strike Rate: **161.27** (229 runs off 142 balls)

Important Points

- **Jake Fraser-McGurk** dominated IPL 2024 with an **unbelievable strike rate of 241.82**, making him the most explosive batter of the season! □□
- **Travis Head & Abhishek Sharma** displayed fearless batting, consistently scoring at a strike rate above **185**.
- **Phil Salt and Sunil Narine** also played vital attacking roles, boosting their teams' run rates.
- **Virat Kohli, Faf du Plessis, and Ishan Kishan** balanced strike rate with consistency, ensuring stability at the crease.
- **Young talents like YBK Jaiswal, Abishek Porel, and Prithvi Shaw** showcased their aggressive approach in IPL 2024.

This highlights **Jake Fraser-McGurk's dominance** as the most lethal batter of IPL 2024! □□

This function calculates the most economical bowler (minimum 200 balls bowled) based on various factors such as start year, end year, start over, end over, and minimum balls bowled.

```
def  most_economical_bowler(data,start_year:int=2024,
                            end_year:int=2024,
                            start_over:int=0,
                            end_over:int=19,
                            min_ball:int=60,
                            inning :int=None)->pd.DataFrame:
  '''
   Function: most_economical_bowler
    Author: Nikesh
    Date: March 28, 2025

    Description:
    This function calculates the most economical bowlers in IPL based
on a given
    timeframe, overs range, and minimum balls bowled.

    Parameters:
    - data (DataFrame): The IPL dataset containing match details,
including bowling performance.
    - start_year (int): The starting season year (default = 2024).
    - end_year (int): The ending season year (default = 2024).
```

```
    - start_over (int): The starting over number (default = 0).
    - end_over (int): The ending over number (default = 19).
    - min_ball (int): The minimum number of balls a bowler must have
bowled to be considered (default = 60).

    Returns:
    - DataFrame: A table with columns ['bowler', 'total_ball',
'total_runs', 'economy'],
      sorted by economy rate in ascending order.

  '''
  filter_data = ipl[(ipl['date'].dt.year >= start_year) &
                    (ipl['date'].dt.year <= end_year) &
                    (ipl['over']>= start_over) &
                    (ipl['over']<= end_over)]
  if inning in([1,2]):
    filter_data = filter_data[filter_data['inning']== inning]

  bowler_stats = filter_data.groupby(['bowler']).agg(
    total_ball = ('bowler','count'),
    total_runs = ('total_runs','sum')).reset_index()

  result = bowler_stats[bowler_stats['total_ball'] >=min_ball]


  result['economy'] = (result['total_runs']/result['total_ball']) *6
  return result.sort_values(by='economy')
```

most_economical_bowler in power play (1 to 6) over in 2024

```
most_economical_bowler(ipl,2024,2024,0,5,60).sort_values(by='economy',
ascending=True)
```

```
<ipython-input-612-83dad131ceb1>:43: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  result['economy'] = (result['total_runs']/result['total_ball']) *6
```


```
{"summary":"{\n  \"name\":
\"most_economical_bowler(ipl,2024,2024,0,5,60)\",\n  \"rows\": 31,\n
\"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
```

```
\"num_unique_values\": 31,\n        \"samples\": [\n          \"N
Burger\",\n          \"SM Curran\",\n          \"Harshit Rana\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_ball\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
51,\n        \"min\": 61,\n        \"max\": 245,\n
\"num_unique_values\": 26,\n        \"samples\": [\n          120,\n
85,\n          134\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 68,\n        \"min\": 94,\n
\"max\": 347,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          114,\n          150,\n          177\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"economy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.084712312073124,\n        \"min\": 5.731343283582089,\n
\"max\": 11.39240506329114,\n        \"num_unique_values\": 31,\n
\"samples\": [\n          10.161290322580646,\n
8.717948717948717,\n          9.545454545454545\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Most Economical Bowlers in Powerplay (Overs 1-6) – IPL 2024

(Minimum 60 Balls Bowled)

Top Performers

 Jasprit Bumrah – Economy: **5.73** (128 runs conceded off 134 balls)
 Trent Boult – Economy: **6.95** (271 runs off 234 balls)
 Sandeep Sharma – Economy: **7.65** (116 runs off 91 balls)
 Vaibhav Arora – Economy: **8.02** (218 runs off 163 balls)
⚡ **Deepak Chahar** – Economy: **8.05** (169 runs off 126 balls)

Important Points

- **Jasprit Bumrah** emerged as the most economical bowler in the powerplay, conceding just **5.73 runs per over**, making him the toughest bowler to score against in the early overs.

- **Trent Boult and Sandeep Sharma** maintained tight control in the powerplay, restricting runs and picking up crucial wickets.

- **Deepak Chahar and Vaibhav Arora** were among the few bowlers who consistently troubled batters with disciplined line and length.

- **Experienced pacers like Kagiso Rabada, Bhuvneshwar Kumar, and Arshdeep Singh** had moderate economy rates but played key roles in early breakthroughs.

- **Naveen-ul-Haq and Avesh Khan** were among the most expensive bowlers in the powerplay, conceding runs at over **10 per over**.

This analysis highlights **Jasprit Bumrah's exceptional control and impact** as the best powerplay bowler of IPL 2024! □□

most_economical_bowler in death over (15 to 20) over in 2024

```
most_economical_bowler(ipl,2024,2024,14,19,60).sort_values(by='economy
',ascending=True)
```

```
<ipython-input-612-83dad131ceb1>:43: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  result['economy'] = (result['total_runs']/result['total_ball']) *6
```


```
{"summary":"{\n  \"name\":
\"most_economical_bowler(ipl,2024,2024,14,19,60)\",\n  \"rows\": 35,\n
\"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 35,\n        \"samples\": [\n          \"Yash
Thakur\",\n          \"Mustafizur Rahman\",\n          \"AD Russell\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_ball\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 29,\n        \"min\": 60,\n
\"max\": 161,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          83,\n          137,\n          120\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 43,\n        \"min\": 95,\n
\"max\": 262,\n        \"num_unique_values\": 28,\n
\"samples\": [\n          131,\n          135,\n          207\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"economy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.6126599703567859,\n        \"min\": 7.073170731707318,\n
\"max\": 16.099999999999998,\n        \"num_unique_values\": 34,\n
\"samples\": [\n          9.73913043478261,\n
```

```
10.029850746268657,\n          11.228571428571428\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Most Economical Bowlers in Death Overs (Overs 15-20) – IPL 2024

(Minimum 60 Balls Bowled)

Top Performers

 Jasprit Bumrah – Economy: **7.07** (145 runs conceded off 123 balls)
 Harshit Rana – Economy: **7.61** (137 runs off 108 balls)
 Mohammed Siraj – Economy: **7.91** (211 runs off 160 balls)
 Matheesha Pathirana – Economy: **8.30** (112 runs off 81 balls)
⚡ **Avesh Khan** – Economy: **8.46** (227 runs off 161 balls)

Important Points

- **Jasprit Bumrah** once again proved his mastery in the death overs, maintaining a sensational economy rate of **7.07**, making him the toughest bowler to score against in crunch situations.

- **Harshit Rana & Mohammed Siraj** were impressive under pressure, conceding less than **8 runs per over** in the most challenging phase of the game.

- **Matheesha Pathirana & Avesh Khan** delivered crucial spells at the death, preventing explosive finishes from batters.

- **Trent Boult, T Natarajan, and Mustafizur Rahman** were consistent performers, maintaining economy rates below **9.50**.

- **Yash Dayal, Naveen-ul-Haq, and Sam Curran** had mixed performances, occasionally leaking runs but also taking wickets.

- **Anrich Nortje had the worst economy rate (16.10),** struggling to contain runs in the death overs.

This analysis highlights **Jasprit Bumrah's brilliance** as the best death-over bowler of IPL 2024!

most economical bowler in 2nd inning 2024

```
most_economical_bowler(ipl,2024,2024,0,19,90,2).sort_values(by='econom
y',ascending=True)
```

```
<ipython-input-612-83dad131ceb1>:43: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  result['economy'] = (result['total_runs']/result['total_ball']) *6
```


```
{"summary":"{\n  \"name\":
\"most_economical_bowler(ipl,2024,2024,0,19,90,2)\",\n  \"rows\": 39,\
n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 39,\n        \"samples\": [\n          \"Yash
Thakur\",\n          \"JD Unadkat\",\n          \"RA Jadeja\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_ball\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
37,\n        \"min\": 96,\n        \"max\": 229,\n
\"num_unique_values\": 32,\n        \"samples\": [\n          116,\n
117,\n          118\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 57,\n        \"min\": 123,\n
\"max\": 373,\n        \"num_unique_values\": 35,\n
\"samples\": [\n          194,\n          214,\n          164\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"economy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.2790394022582046,\n        \"min\": 5.767441860465117,\n
\"max\": 11.09090909090909,\n        \"num_unique_values\": 38,\n
\"samples\": [\n          10.396551724137932,\n
11.025974025974026,\n          7.40506329113924\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Most Economical Bowlers in 2nd Innings – IPL 2024

(Minimum 90 Balls Bowled)

Top Performers

 Jasprit Bumrah – Economy: **5.77** (124 runs conceded off 129 balls)
 Sunil Narine – Economy: **6.59** (235 runs off 214 balls)
 Cameron Green – Economy: **7.21** (155 runs off 129 balls)
 Matheesha Pathirana – Economy: **7.21** (143 runs off 119 balls)
⚡ **Ravindra Jadeja** – Economy: **7.41** (195 runs off 158 balls)

Important points

- **Jasprit Bumrah was the most economical bowler in the second innings**, conceding just **5.77 runs per over**, making him the toughest bowler to score against in IPL 2024.
- **Sunil Narine & Ravindra Jadeja** dominated the spin department with economy rates below **7.50**, restricting batsmen in the middle and death overs.
- **Cameron Green & Matheesha Pathirana** were highly effective, maintaining economy rates close to **7.21**, proving crucial in crucial match phases.
- **T Natarajan, Kuldeep Yadav, and Axar Patel** kept things tight, maintaining economy rates around **8.00**, while **Mohammed Siraj, Rashid Khan, and Mustafizur Rahman** had slightly higher economy rates (~9.00).
- **Mitchell Starc (11.03) and Shahbaz Ahmed (11.09)** were among the most expensive bowlers in the second innings, struggling to control the run flow.

This analysis confirms **Jasprit Bumrah's dominance** as the best second-innings bowler, delivering consistent, match-winning performances in IPL 2024! □□

most economical bowler in 2nd inning 2024 in death over

```
most_economical_bowler(ipl,2024,2024,14,19,36,2).sort_values(by='econo
my',ascending=True)
```

```
<ipython-input-612-83dad131ceb1>:43: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  result['economy'] = (result['total_runs']/result['total_ball']) *6
```


```
{"summary":"{\n  \"name\":
\"most_economical_bowler(ipl,2024,2024,14,19,36,2)\",\n  \"rows\":
27,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 27,\n        \"samples\": [\n          \"Avesh
Khan\",\n          \"Sandeep Sharma\",\n          \"C Green\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_ball\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
15,\n        \"min\": 36,\n        \"max\": 102,\n
\"num_unique_values\": 18,\n        \"samples\": [\n          39,\n
```

```
38,\n          81\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 23,\n        \"min\": 44,\n
\"max\": 130,\n        \"num_unique_values\": 22,\n
\"samples\": [\n          44,\n          93,\n          64\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"economy\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 1.6852426729756043,\n        \"min\":
6.769230769230769,\n        \"max\": 12.315789473684212,\n
\"num_unique_values\": 25,\n        \"samples\": [\n
8.352941176470587,\n          10.0,\n          6.769230769230769\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Most Economical Bowlers in 2nd Innings – Death Overs (15-20) – IPL 2024

(Minimum 35 Balls Bowled)

Top Performers

 Tushar Deshpande – Economy: **6.77** (44 runs conceded off 39 balls)
 Jasprit Bumrah – Economy: **6.95** (44 runs off 38 balls)
 Mohammed Siraj – Economy: **7.06** (120 runs off 102 balls)
 Yash Dayal – Economy: **7.53** (59 runs off 47 balls)
⚡ **Shardul Thakur** – Economy: **7.62** (47 runs off 37 balls)

Key Insights

- **Tushar Deshpande & Jasprit Bumrah** were the most economical bowlers in the second innings during the death overs, conceding less than **7 runs per over**, making them the most difficult to score against.

- **Mohammed Siraj & Yash Dayal** consistently restricted runs, maintaining economy rates below **7.60** in high-pressure situations.

- **T Natarajan & Matheesha Pathirana** continued their strong death-over performances, conceding under **8.10** runs per over.

- **Harshal Patel, Avesh Khan, and Cameron Green** also provided stability at the end, keeping their economy rates below **8.60**.

- **Mustafizur Rahman, Mukesh Kumar, and Rashid Khan** struggled, conceding over **10 runs per over**, while **Andre Russell & Jaydev Unadkat** had the worst economy rate (12.00+).

This analysis highlights **Tushar Deshpande's efficiency** in the second innings and **Jasprit Bumrah's consistency** as a top death-over specialist in IPL 2024!

Find the bowler with the most maiden overs.

```
md = ipl.groupby(['match_id','bowler','over'])
['total_runs'].sum().reset_index()
md[md['total_runs']==0].groupby('bowler').size().reset_index(name='maiden_over').sort_values(by='maiden_over',ascending=False)
```


```
{"summary":"{\n  \"name\": \"md[md['total_runs']==0]\",\n  \"rows\": 157,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 157,\n        \"samples\": [\n          \"SB Jakati\",\n          \"A Choudhary\",\n          \"RE van der Merwe\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"maiden_over\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 1,\n        \"max\": 12,\n        \"num_unique_values\": 11,\n        \"samples\": [\n          6,\n          12,\n          2\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


This functon find the batsman who hit most six or four based on different type of filters such as start season end season start over last over 1st inning 2nd inning

```
def top_boundary_hitters(ipl: pd.DataFrame,
                         boundary_type: int = None,
                         start_year: int = 2024,
                         end_year: int = 2024,
                         start_over: int = 0,
                         end_over: int = 19,
                         innings: int = None) -> pd.DataFrame:
    """
    Function: top_boundary_hitters
    Author: Nikesh
    Date: March 25, 2025

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details.
    - boundary_type (int, optional): Type of boundary to count (4 for
```

```
fours, 6 for sixes).
    - start_year (int, optional): The starting season year (default =
2024).
    - end_year (int, optional): The ending season year (default =
2024).
    - start_over (int, optional): The starting over number (default =
0).
    - end_over (int, optional): The ending over number (default = 19).
    - innings (int, optional): The inning to filter (1 or 2).

    Returns:
    - DataFrame: A table with ['batter', 'boundary_count'], sorted in
descending order.
    """

    # Base filter for years and overs
    filter_data = ipl[
        (ipl['date'].dt.year >= start_year) &
        (ipl['date'].dt.year <= end_year) &
        (ipl['over'] >= start_over) &
        (ipl['over'] <= end_over)
    ]

    # Apply boundary type filter if specified
    if boundary_type is not None:
        filter_data = filter_data[filter_data['total_runs'] ==
boundary_type]
    else:
        filter_data = filter_data[filter_data['total_runs'].isin([4,
6])]

    # Apply innings filter if specified
    if innings in [1, 2]:
        filter_data = filter_data[filter_data['inning'] == innings]

    # Define column name based on boundary type and innings
    if boundary_type is None:
        column_name = "total_6_4"
    elif innings == 1:
        column_name = 'No of Sixes in 1st Inning' if boundary_type ==
6 else "No of Fours in 1st Inning"
    elif innings == 2:
        column_name = 'No of Sixes in 2nd Inning' if boundary_type ==
6 else 'No of Fours in 2nd Inning'
    else:
        column_name = 'No of Sixes' if boundary_type == 6 else 'No of
Fours'

    # Group by batter and count occurrences
    result = (
```

```
        filter_data.groupby('batter')
        .size()
        .reset_index(name=column_name)
        .sort_values(by=column_name, ascending=False)
    )

    return result
```

Top Boundary Hitters Batsman – IPL 2024

```
top_boundary_hitters(ipl, start_year=2024, end_year=2024).head(30)
```


```
{"summary":"{\n  \"name\": \"top_boundary_hitters(ipl,
start_year=2024, end_year=2024)\",\n  \"rows\": 30,\n  \"fields\": [\n
{\n      \"column\": \"batter\",\n      \"properties\": {\n
\"dtype\": \"string\",\n          \"num_unique_values\": 30,\n
\"samples\": [\n          \"T Stubbs\",\n          \"J Fraser-
McGurk\",\n          \"P Simran Singh\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_6_4\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
13,\n        \"min\": 48,\n        \"max\": 100,\n
\"num_unique_values\": 22,\n        \"samples\": [\n          100,\n
59,\n          70\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Top Boundary Hitters – IPL 2024

Top Performers

 Virat Kohli – 100 boundaries (4s & 6s)
 Travis Head – 95 boundaries
 Sunil Narine – 88 boundaries
 Abhishek Sharma – 78 boundaries
⚡ **Philip Salt & Ruturaj Gaikwad – 75 boundaries each**

Key Insights

- **Virat Kohli emerged as the top boundary scorer** in IPL 2024, smashing **100 boundaries**, proving his consistency and adaptability across formats.

- **Travis Head (95) and Sunil Narine (88)** showcased their aggressive batting approach, excelling as top-order power hitters.

- **Young talents like Abhishek Sharma (78), Riyan Parag (73), and Yashasvi Jaiswal (70)** played key roles in their teams' batting success.

- **Middle-order powerhouses like Nicholas Pooran (71), Shivam Dube (58), and Heinrich Klaasen (57)** provided crucial finishes, turning games around.

- **The presence of experienced players like Rohit Sharma (68), Faf du Plessis (69), and Jos Buttler (48)** highlights the impact of seasoned campaigners in high-pressure situations.

This analysis reaffirms that **boundary-hitting remains a crucial skill in modern T20 cricket**, with a blend of experienced players and young superstars excelling in IPL 2024! □□

Top Boundary Hitters Batsman in 2nd inning – IPL 2024

```
top_boundary_hitters(ipl, start_year=2024,
end_year=2024,innings=2).head(30)
```


```
{"summary":"{\n  \"name\": \"top_boundary_hitters(ipl,
start_year=2024, end_year=2024,innings=2)\",\n  \"rows\": 30,\n
\"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 30,\n        \"samples\": [\n          \"SS
Iyer\",\n          \"P Simran Singh\",\n          \"WG Jacks\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_6_4\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9,\n        \"min\": 23,\n        \"max\": 55,\n
\"num_unique_values\": 18,\n        \"samples\": [\n          55,\n
52,\n          34\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Top Boundary Hitters in 2nd Innings – IPL 2024

Top Performers

□ **Yashasvi Jaiswal – 55 boundaries (4s & 6s)**
□ **Rohit Sharma – 52 boundaries**
□ **Sanju Samson – 48 boundaries**
□ **Shashank Singh – 45 boundaries**
⚡ **Abhishek Sharma, Jos Buttler & Jonny Bairstow – 44 boundaries each**

Key Insights

- **Yashasvi Jaiswal led the charts** with **55 boundaries in the second innings**, proving his ability to chase under pressure.
- **Rohit Sharma (52) and Sanju Samson (48)** showcased their finishing ability, anchoring successful chases.
- **Explosive hitters like Shashank Singh (45), Abhishek Sharma (44), and Jos Buttler (44)** played match-winning knocks for their teams.
- **Young talents such as Tilak Varma (38), Riyan Parag (38), and Prabhsimran Singh (32)** made significant contributions in crunch moments.
- **Experienced players like Faf du Plessis (33), KL Rahul (29), and Dinesh Karthik (24)** provided stability in the second innings.

This analysis highlights **Yashasvi Jaiswal's dominance in run chases**, alongside the brilliance of experienced campaigners and emerging young talents in IPL 2024! □□

Top six Hitters Batsman in – IPL 2024

```
top_boundary_hitters(ipl,boundary_type=6, start_year=2024,
end_year=2024).head(30)
```


```
{"summary":"{\n  \"name\": \"top_boundary_hitters(ipl,boundary_type=6,
start_year=2024, end_year=2024)\",\n  \"rows\": 30,\n  \"fields\": [\n
{\n      \"column\": \"batter\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 30,\n
\"samples\": [\n          \"MP Stoinis\",\n          \"KD Karthik\",\n
\"RD Gaikwad\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"No
of Sixes\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 7,\n        \"min\": 16,\n        \"max\": 41,\n
\"num_unique_values\": 16,\n        \"samples\": [\n          41,\n
38,\n          28\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Top Six Hitters – IPL 2024

Top Performers

□ **Abhishek Sharma – 41 sixes**
□ **Heinrich Klaasen & Virat Kohli – 38 sixes each**
□ **Nicholas Pooran – 36 sixes**
□ **Rajat Patidar, Riyan Parag & Sunil Narine – 33 sixes each**
⚡ **Travis Head – 32 sixes**

Key Insights

- **Abhishek Sharma was the leading six-hitter**, smashing **41 sixes** throughout the tournament.
- **Heinrich Klaasen and Virat Kohli** followed closely with **38 sixes each**, displaying power-hitting and consistency.
- **Nicholas Pooran (36), Rajat Patidar (33), and Riyan Parag (33)** dominated the middle overs with aggressive stroke play.
- **Explosive openers like Travis Head (32) and Shivam Dube (28)** played match-defining knocks.
- **Andre Russell (16 sixes) and Marcus Stoinis (16 sixes)** proved their finishing ability in high-pressure situations.

This list showcases a mix of **experienced power-hitters and young talents**, making IPL 2024 a season filled with breathtaking six-hitting action! □□□

Top six Hitters Batsman in 2nd inning – IPL 2024

```
top_boundary_hitters(ipl,boundary_type=6, start_year=2024,
end_year=2024,innings=2).head(30)
```


```
{"summary":"{\n  \"name\": \"top_boundary_hitters(ipl,boundary_type=6,
start_year=2024, end_year=2024,innings=2)\",\n  \"rows\": 30,\n
\"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 30,\n        \"samples\": [\n          \"RM
Patidar\",\n          \"Tilak Varma\",\n          \"KD Karthik\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"No of Sixes in 2nd Inning\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 9,\n        \"max\": 20,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          10,\n
19,\n          13\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Top Six Hitters in 2nd Innings – IPL 2024

Top Performers

□ **Abhishek Sharma – 20 sixes**
□ **Shashank Singh – 19 sixes**
□ **Rohit Sharma – 17 sixes**

 Riyan Parag, Sanju Samson & Jake Fraser-McGurk – 16 sixes each
⚡ **Tristan Stubbs, Virat Kohli, Will Jacks, Suryakumar Yadav & Yashasvi Jaiswal – 14 sixes each**

Key Insights

- **Abhishek Sharma dominated the second innings**, leading with **20 sixes**, showcasing his power-hitting ability.
- **Shashank Singh emerged as a surprise big-hitter**, smashing **19 sixes** and making a significant impact.
- **Rohit Sharma (17) and a group of aggressive middle-order batsmen (16 sixes each)** played crucial finishing roles.
- **Virat Kohli (14) and Suryakumar Yadav (14)** demonstrated their classic stroke play and adaptability under pressure.
- **Explosive finishers like Nicholas Pooran, Tristan Stubbs, and Tilak Varma (12 sixes each)** contributed significantly in crunch moments.

This list highlights a **blend of experienced superstars and emerging talents**, making the **second innings an exciting spectacle of six-hitting brilliance in IPL 2024!**

Top four Hitters Batsman in 2nd inning – IPL 2024

```
top_boundary_hitters(ipl,boundary_type=4, start_year=2024,
end_year=2024,innings=2).head(30)
```


```
{"summary":"{\n  \"name\": \"top_boundary_hitters(ipl,boundary_type=4,
start_year=2024, end_year=2024,innings=2)\",\n  \"rows\": 30,\n
\"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 30,\n        \"samples\": [\n          \"KD
Karthik\",\n          \"KL Rahul\",\n          \"RR Pant\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"No
of Fours in 2nd Inning\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6,\n        \"min\": 13,\n
\"max\": 41,\n        \"num_unique_values\": 17,\n        \"samples\":
[\n          41,\n          35,\n          27\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

Top Four Hitters in 2nd Innings – IPL 2024

Top Performers

 Yashasvi Jaiswal – 41 fours
 Rohit Sharma – 35 fours
 Jos Buttler & Sanju Samson – 32 fours each
 Jonny Bairstow (30) & Virat Kohli (28) showcased their elite stroke play.

Key Insights

- **Yashasvi Jaiswal led the charts with 41 boundaries**, showing his aggressive intent at the top order.
- **Rohit Sharma (35) & Jos Buttler (32) consistently found gaps**, accelerating their teams' innings.
- **Virat Kohli (28) and Ishan Kishan (28) played anchor roles**, balancing aggression with stability.
- **Shashank Singh & Tilak Varma (26 each) emerged as key contributors in the middle order.**
- **Aggressive hitters like Marcus Stoinis, Abhishek Sharma, and B Sai Sudharsan (24 each)** provided strong finishing touches.

The second innings in IPL 2024 saw a **blend of experienced top-order players and dynamic middle-order hitters**, making it a thrilling battle for boundary dominance!

Top four Hitters Batsman in – IPL 2024

```
top_boundary_hitters(ipl,boundary_type=4, start_year=2024,
end_year=2024).head(30)
```


```
{"summary":"{\n  \"name\": \"top_boundary_hitters(ipl,boundary_type=4,
start_year=2024, end_year=2024)\",\n  \"rows\": 30,\n  \"fields\": [\n
{\n      \"column\": \"batter\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 30,\n
\"samples\": [\n          \"S Dube\",\n          \"MP Stoinis\",\n
\"P Simran Singh\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"No
of Fours\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 9,\n        \"min\": 29,\n        \"max\": 63,\n
\"num_unique_values\": 19,\n        \"samples\": [\n          63,\n
51,\n          37\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

Top Four Hitters – IPL 2024

Top Performers

 Travis Head – 63 fours
 Virat Kohli – 62 fours
 Ruturaj Gaikwad – 57 fours
 Sunil Narine (55) & Yashasvi Jaiswal (54) followed closely behind.

Key Insights

- **Travis Head led the tournament with 63 boundaries**, displaying his aggressive batting approach.
- **Virat Kohli (62) was the backbone of RCB's batting**, maintaining consistency throughout IPL 2024.
- **Ruturaj Gaikwad (57) & Sunil Narine (55) played key roles at the top of the order**, providing their teams with strong starts.
- **Dynamic middle-order players like Sanju Samson, Faf du Plessis, and Sai Sudharsan (48 each) contributed significantly** to their team's totals.
- **Jos Buttler, Shubman Gill, and Rishabh Pant showcased their class**, maintaining high boundary counts despite different batting roles.

IPL 2024 featured **a mix of aggressive openers and composed middle-order batsmen**, making the race for boundary-hitting an exciting contest!

Batting Analysis

Highest individual scores by a batsman for each season or a specific inning, or including all innings ans season

```
def
total_score_by_batsman(data,start_season:int=None,end_season:int=None,
inning: int = None)->pd.DataFrame:
  '''
  Function: total_score_by_batsman
    Author: Nikesh
    Date: March 25, 2025
```

```
    Parameters:
    - data (DataFrame): The IPL dataset containing match details.
    - start_season (int): The starting season year (default = 2008).
    - end_season (int): The ending season year (default = 2024).
    - inning (int, optional): The inning number (1 or 2). If None,
considers both innings.

    Returns:
    - DataFrame: A table with ['season', 'batter', 'batsman_runs'],
sorted in descending order.

  '''
  filter_data = ipl[(ipl['date'].dt.year >= start_season) &
                    (ipl['date'].dt.year <= end_season)]

  if inning in [1,2]:
    filter_data = filter_data[filter_data['inning']==inning]


  return filter_data.groupby(['batter',])
['batsman_runs'].sum().reset_index().sort_values(by='batsman_runs',asc
ending=False)


total_score_by_batsman(ipl,start_season=2024,end_season =
2024).head(30)
```


```
{"summary":"{\n  \"name\":
\"total_score_by_batsman(ipl,start_season=2024,end_season = 2024)\",\n
\"rows\": 30,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 30,\n        \"samples\": [\n          \"P
Simran Singh\",\n          \"Shubman Gill\",\n          \"JC
Buttler\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"batsman_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 94,\n        \"min\": 327,\n
\"max\": 741,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          330,\n          416,\n          438\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Top Run-Scorers – IPL 2024

```
total_score_by_batsman(ipl,start_season=2024,end_season =
2024).head(30)
```

```
{"summary":"{\n  \"name\":
\"total_score_by_batsman(ipl,start_season=2024,end_season = 2024)\",\n
\"rows\": 30,\n  \"fields\": [\n    {\n      \"column\": \"batter\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 30,\n        \"samples\": [\n          \"P
Simran Singh\",\n          \"Shubman Gill\",\n          \"JC
Buttler\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"batsman_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 94,\n        \"min\": 327,\n
\"max\": 741,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          330,\n          416,\n          438\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Top Run-Scorers – IPL 2024

Top Performers

 Virat Kohli – **741 runs**
 Ruturaj Gaikwad – **583 runs**
 Riyan Parag – **573 runs**
 Travis Head (567) & Sanju Samson (531) completed the top five.

Key Insights

- **Virat Kohli dominated IPL 2024**, scoring **741 runs**, proving his consistency and class yet again.
- **Ruturaj Gaikwad (583) & Riyan Parag (573) were crucial for their teams**, delivering impactful performances.
- **Travis Head (567) & Sanju Samson (531) maintained aggressive strike rates**, helping their teams with explosive starts and match-winning innings.
- **B Sai Sudharsan (527), KL Rahul (520), and Nicholas Pooran (499) provided stability**, anchoring innings under pressure.
- **Aggressive finishers like Heinrich Klaasen (479) & Abhishek Sharma (484) added firepower** in the middle and death overs.
- **Veterans like Rohit Sharma (417) & Faf du Plessis (438) showed their experience**, guiding their teams in crucial matches.

This season saw **a mix of experienced stalwarts and emerging stars** making their mark, making IPL 2024 one of the most competitive editions ever!

Top Run-Scorers In 1st Inning – IPL 2024

```
total_score_by_batsman(ipl,start_season=2024,end_season =
2024,inning=1).head(30)
```


```
{"summary":"{\n  \"name\":
\"total_score_by_batsman(ipl,start_season=2024,end_season =
2024,inning=1)\",\n  \"rows\": 30,\n  \"fields\": [\n    {\n
\"column\": \"batter\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 30,\n        \"samples\":
[\n          \"KD Karthik\",\n          \"C Green\",\n          \"T
Stubbs\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"batsman_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 86,\n        \"min\": 156,\n
\"max\": 470,\n        \"num_unique_values\": 29,\n
\"samples\": [\n          165,\n          228,\n          259\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Top Run-Scorers in 1st Inning – IPL 2024

Top Performers

 Virat Kohli – 470 runs
 Ruturaj Gaikwad – 458 runs
 Travis Head – 433 runs
 Sunil Narine (393) & Heinrich Klaasen (337) completed the top five.

Key Insights

- **Virat Kohli (470) & Ruturaj Gaikwad (458) dominated in the first innings**, setting up strong platforms for their teams.
- **Travis Head (433) was among the most explosive openers**, providing quick starts.
- **Sunil Narine (393) played an aggressive role at the top**, consistently putting pressure on the bowlers.
- **Heinrich Klaasen (337) was crucial in finishing innings with power hitting.**
- **KL Rahul (306) & Riyan Parag (298) maintained stability**, anchoring crucial partnerships.
- **Nicholas Pooran (293) & Rajat Patidar (285) provided middle-order strength.**
- **Young talents like Abhishek Sharma (228) & Nitish Kumar Reddy (224) impressed with their performances.**

These stats highlight **the key contributors in the first innings**, setting the tone for IPL 2024 matches!

Top Run-Scorers In 2nd Inning – IPL 2024

```
total_score_by_batsman(ipl,start_season=2024,end_season =
2024,inning=2).head(30)
```


```
{"summary":"{\n  \"name\":
\"total_score_by_batsman(ipl,start_season=2024,end_season =
2024,inning=2)\",\n  \"rows\": 30,\n  \"fields\": [\n    {\n
\"column\": \"batter\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 30,\n        \"samples\":
[\n          \"KD Karthik\",\n          \"KL Rahul\",\n
\"Shubman Gill\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"batsman_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 58,\n        \"min\": 153,\n
\"max\": 354,\n        \"num_unique_values\": 27,\n
\"samples\": [\n          256,\n          221,\n          247\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Top Run-Scorers in 2nd Inning – IPL 2024

Top Performers

 Yashasvi Jaiswal – **354 runs**
 Sanju Samson – **333 runs**
 Rohit Sharma – **322 runs**
 Shashank Singh (314) & Jos Buttler (308) rounded out the top five.

Key Insights

- **Yashasvi Jaiswal (354) was the most dominant second-innings batter**, showing consistency under pressure.
- **Sanju Samson (333) & Rohit Sharma (322) played crucial roles in run chases**, ensuring stability at the top.
- **Shashank Singh (314) emerged as a surprise star, delivering impactful innings in key matches.**
- **Jos Buttler (308) & Jonny Bairstow (275) were key middle-order anchors**, maintaining high strike rates.

- **Tilak Varma (272) & Rajat Patidar (238) provided much-needed acceleration in chases.**
- **Virat Kohli (271) continued his consistency across both innings, proving his adaptability.**
- **Abhishek Sharma (256) & Suryakumar Yadav (247) played vital roles in finishing games.**

These statistics highlight **the best chasers and impact players in IPL 2024**, showcasing their ability to perform under pressure! □□

Overall Conclusion – IPL 2024 Batting Analysis

1st Inning Performers vs. 2nd Inning Performers

- **Virat Kohli (741 runs)** dominated across both innings, proving his consistency as a reliable run-getter.
- **Ruturaj Gaikwad (583) and Travis Head (567)** were standout first-inning batters, playing aggressive knocks at the top.
- **Yashasvi Jaiswal (354) and Sanju Samson (333) thrived in 2nd innings**, particularly in chases.
- **Shashank Singh (314) emerged as a surprise performer**, proving his finishing abilities in 2nd innings.

Key Takeaways

□ **1st Innings Domination:** Kohli, Gaikwad, and Head showed the ability to set up strong totals.
□ **2nd Innings Specialists:** Jaiswal, Samson, and Rohit Sharma were crucial chasers, handling pressure well.
□ **Middle-Order Impact:** Players like Shashank Singh, Jos Buttler, and Tilak Varma provided balance.
□ **Power-Hitters in Both Innings:** Players like Klaasen, Pooran, and Abhishek Sharma contributed across innings.

Final Thoughts

- **Batters like Kohli and Samson adapted to both innings**, showing remarkable consistency.
- **Rising stars like Jaiswal and Shashank Singh played crucial match-winning knocks.**
- **The balance between openers, middle-order stabilizers, and finishers determined IPL 2024's top scorers.**

This season highlighted **a mix of experienced and young talents**, with batters showcasing adaptability under different game conditions. □□

Team with the highest total while batting first.

```
def  highest_team_scores(ipl,season:int=None,inning:int=None)-
>pd.DataFrame:
  '''
   Function: highest_team_scores
    Author: Nikesh
    Date: March 25, 2025

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details, with
columns including
      'date', 'season', 'match_id', 'batting_team', 'inning', and
'total_runs'.
    - season (int, optional): The specific season year to filter the
data (e.g., 2024).
      If None, the function includes all seasons.
    - inning (int, optional): The inning number (1 or 2) to filter the
data.
      If None, the function includes both innings.

    Returns:
    - DataFrame: A table with columns ['batting_team', 'max_score'],
showing
      the highest match score for each team in descending order.

  '''
  # if there is no condition
  filter_ipl = ipl.copy()
  # if year is give
  if season is not None:
    filter_ipl = ipl[ipl['date'].dt.year==season]
  # if inning is given
  if inning in [1,2]:
    filter_ipl = filter_ipl[filter_ipl['inning']==inning]


  match_score = filter_ipl.groupby(['match_id','batting_team'])
['total_runs'].sum()\
                          .reset_index()\
                          .groupby(['batting_team'])\
                          .max().reset_index()

  match_score.rename(columns={'total_runs': 'max_score'},
inplace=True)
  return match_score.sort_values(by='max_score',ascending=False)
[['batting_team','max_score']]
```

```
highest_team_scores(ipl,2024,2)
```


```
{"summary":"{\n  \"name\": \"highest_team_scores(ipl,2024,2)\",\n
\"rows\": 10,\n  \"fields\": [\n    {\n      \"column\":
\"batting_team\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Chennai Super Kings\",\n          \"Kings XI Punjab\",\
n          \"Sunrisers Hyderabad\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"max_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
26,\n        \"min\": 186,\n        \"max\": 262,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          196,\n
247,\n          213\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Top best - Score each team in – IPL 2024

```
highest_team_scores(ipl,2024)
```


```
{"summary":"{\n  \"name\": \"highest_team_scores(ipl,2024)\",\n
\"rows\": 10,\n  \"fields\": [\n    {\n      \"column\":
\"batting_team\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Lucknow Super Giants\",\n          \"Kolkata Knight
Riders\",\n          \"Mumbai Indians\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"max_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
25,\n        \"min\": 212,\n        \"max\": 287,\n
\"num_unique_values\": 9,\n        \"samples\": [\n          214,\n
272,\n          231\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


IPL 2024 – Top Team Scores Analysis

High-Scoring Teams

- **Sunrisers Hyderabad (287)** – The highest team score of the season, showcasing an explosive batting lineup.
- ⚡ **Kolkata Knight Riders (272) & Royal Challengers Bangalore (262)** – Dominated with aggressive batting performances.
- **Kings XI Punjab (262) & Delhi Capitals (257)** – Strong and consistent run-scoring teams.

Mid-Tier Performers

- **Mumbai Indians (247)** – Competitively high score but did not breach the 260+ mark.
- **Gujarat Titans (231) & Rajasthan Royals (224)** – Fell short of the 240+ range, indicating struggles in batting depth.

Lower-End Scores

- **Lucknow Super Giants (214) & Chennai Super Kings (212)** – The lowest high scores, reflecting inconsistency in power-hitting.

Key Takeaways

Sunrisers Hyderabad dominated with the highest total (287).
KKR, RCB, and PBKS impressed with scores above 260.
CSK and LSG struggled to produce big totals.

IPL 2024 was a high-scoring season, with several teams surpassing 250+, emphasizing the role of power-hitters and aggressive strategies!

Top best - Score each team in 2nd inning – IPL 2024

```
highest_team_scores(ipl,2024,2)
```


```
{"summary":"{\n  \"name\": \"highest_team_scores(ipl,2024,2)\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"batting_team\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"Chennai Super Kings\",\n          \"Kings XI Punjab\",\n          \"Sunrisers Hyderabad\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"max_score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 26,\n        \"min\": 186,\n        \"max\": 262,\n        \"num_unique_values\": 9,\n        \"samples\": [\n          196,\n          247,\n          213\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


IPL 2024 – Top Team Scores in 2nd Innings

High-Scoring Teams in 2nd Innings

- **Royal Challengers Bangalore (262) & Kings XI Punjab (262)** – Delivered exceptional chases, proving their ability to dominate under pressure.
- ⚡ **Mumbai Indians (247)** – A strong second-innings batting performance.

- **Rajasthan Royals (224) & Gujarat Titans (220)** – Demonstrated resilience while chasing.

Mid-Tier Performers

- **Sunrisers Hyderabad (215) & Lucknow Super Giants (213)** – Respectable scores but below the top chasers.
- **Delhi Capitals (205)** – Crossed the 200 mark but struggled to breach 220.

Lower-End Scores

- **Chennai Super Kings (196) & Kolkata Knight Riders (186)** – Found it difficult to put up big runs in the second innings.

Key Takeaways

RCB & PBKS set the benchmark for successful chases (262).
MI, RR, and GT also performed well in high-pressure chases.
KKR and CSK struggled to chase big targets.

IPL 2024 witnessed aggressive batting in the second innings, with multiple teams surpassing 250, proving that chasing big totals is now a crucial skill!

Most runs conceded by a team in a match.

```
def
highest_runs_conceded_by_bowling_team(ipl,year:int=None,inning:int=Non
e)->pd.DataFrame:
    '''
Function: highest_runs_conceded_by_bowling_team
    Author: Nikesh
    Date: March 25, 2025

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details, with
columns including
      'date', 'season', 'match_id', 'bowling_team', 'inning', and
'total_runs'.
    - year (int, optional): The specific season year to filter the
data (e.g., 2024).
      If None, the function includes all seasons.
    - inning (int, optional): The inning number (1 or 2) to filter the
data.
      If None, the function includes both innings.

    Returns:
    - DataFrame: A table with columns ['season', 'bowling_team',
'max_runs'], showing
      the highest runs conceded by each bowling team in descending
```

```
order.
  '''
  filter_data = ipl.copy()

  if year is not None:
    filter_data = ipl[ipl['date'].dt.year==year]

  if inning in [1,2]:
    filter_data = filter_data[filter_data['inning']==inning]

  match_score = filter_data.groupby(['match_id','bowling_team'])
['total_runs'].sum().reset_index()


  result =
match_score.groupby(['bowling_team']).agg(max_runs=('total_runs','max'
)).reset_index()

  return result.sort_values(by='max_runs',ascending=False)
[['bowling_team','max_runs']]


highest_runs_conceded_by_bowling_team(ipl,2024)
```


```
{"summary":"{\n  \"name\":
\"highest_runs_conceded_by_bowling_team(ipl,2024)\",\n  \"rows\": 10,\
n  \"fields\": [\n    {\n      \"column\": \"bowling_team\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Gujarat Titans\",\n          \"Mumbai Indians\",\n          \"Kings
XI Punjab\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"max_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 23,\n        \"min\": 223,\n
\"max\": 287,\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          224,\n          277,\n          235\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Most runs conceded by a team in a match in 2024

```
highest_runs_conceded_by_bowling_team(ipl,2024)
```


```
{"summary":"{\n  \"name\":
\"highest_runs_conceded_by_bowling_team(ipl,2024)\",\n  \"rows\": 10,\
n  \"fields\": [\n    {\n      \"column\": \"bowling_team\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
```

```
\"num_unique_values\": 10,\n        \"samples\": [\n
\"Gujarat Titans\",\n          \"Mumbai Indians\",\n          \"Kings
XI Punjab\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"max_runs\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 23,\n        \"min\": 223,\n
\"max\": 287,\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          224,\n          277,\n          235\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


🏏 Most Runs Conceded by Teams in IPL 2024

Top Bowling Struggles

🔹 **Royal Challengers Bangalore – 287 runs conceded**
🔹 **Mumbai Indians – 277 runs conceded**
🔹 **Delhi Capitals – 272 runs conceded**
🔹 **Kolkata Knight Riders (262) & Sunrisers Hyderabad (262) also faced tough bowling challenges.**

Key Insights

- **Royal Challengers Bangalore (287) conceded the highest runs in a single match**, highlighting struggles in their bowling attack.
- **Mumbai Indians (277) and Delhi Capitals (272) also faced challenges in restricting opposition batters.**
- **KKR & SRH (262 each) leaked runs heavily, making them vulnerable in high-scoring games.**
- **Punjab Kings (261) and Lucknow Super Giants (235) also had games where they struggled to contain the opposition.**
- **Even Chennai Super Kings (231) and Gujarat Titans (224), known for their strong bowling, had off days.**
- **Rajasthan Royals (223) had the lowest high-run concession but still crossed the 220+ mark.**

🔹 **IPL 2024 saw multiple high-scoring games, exposing the weaknesses of bowling attacks.**
🔹 **With teams consistently scoring 250+, death-over bowling became a key factor in determining match outcomes.** 🏏

Team with the best economy rate in a season.

```
def team_economy_rate_per_season(ipl, start_year: int = None,
end_year: int = None) -> pd.DataFrame:
    '''
    Function: team_economy_rate_per_season
    Author: Nikesh
    Date: March 28, 2025

    Description:
    This function calculates the economy rate (runs conceded per over)
of each bowling team per season.

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details.
    - start_year (int, optional): The starting year for filtering
data.
    - end_year (int, optional): The ending year for filtering data.

    Returns:
    - DataFrame: A table with columns ['season', 'bowling_team',
'total_runs', 'total_ball', 'economy'],
                 showing the economy rate of each bowling team per
season.
    '''
    # Make a copy of the dataset to avoid modifying the original
    filter_data = ipl.copy()

    # Filter based on year
    if start_year is not None:
        filter_data = filter_data[filter_data['season'] >= start_year]
    if end_year is not None:
        filter_data = filter_data[filter_data['season'] <= end_year]

    # Aggregate total runs and total balls bowled by each team per
season
    filter_data = (
        filter_data.groupby(['season', 'bowling_team'])
        .agg(total_runs=('total_runs', 'sum'), total_ball=('match_id',
'count'))
        .reset_index()
    )

    # Calculate economy rate (runs conceded per over)
    filter_data['economy'] = ((filter_data['total_runs'] /
filter_data['total_ball']) * 6).round(2)

    return filter_data
```

```
team_economy_rate_per_season(ipl, start_year=2024,
end_year=2024).sort_values(by='economy')
```


```
{"summary":"{\n  \"name\": \"team_economy_rate_per_season(ipl,
start_year=2024, end_year=2024)\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"season\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 2024,\n
\"max\": 2024,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          2024\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"bowling_team\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          \"Delhi Capitals\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
254,\n        \"min\": 2101,\n        \"max\": 3016,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          2762\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_ball\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
150,\n        \"min\": 1349,\n        \"max\": 1889,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          1737\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"economy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.41575099385195546,\n        \"min\": 8.4,\n        \"max\": 9.58,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          9.54\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Team with the best economy rate in a season 2024

```
team_economy_rate_per_season(ipl, start_year=2024,
end_year=2024).sort_values(by='economy')
```


```
{"summary":"{\n  \"name\": \"team_economy_rate_per_season(ipl,
start_year=2024, end_year=2024)\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"season\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 2024,\n
\"max\": 2024,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          2024\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"bowling_team\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
```

\"samples\": [\n   \"Delhi Capitals\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"total_runs\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
254,\n   \"min\": 2101,\n   \"max\": 3016,\n
\"num_unique_values\": 10,\n   \"samples\": [\n   2762\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"total_ball\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
150,\n   \"min\": 1349,\n   \"max\": 1889,\n
\"num_unique_values\": 10,\n   \"samples\": [\n   1737\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"economy\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.41575099385195546,\n   \"min\": 8.4,\n   \"max\": 9.58,\n
\"num_unique_values\": 10,\n   \"samples\": [\n   9.54\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

🏏 Best Economy Rate by Teams in IPL 2024

Top Bowling Economies

🏆 **Chennai Super Kings – 8.40 economy rate**
🏆 **Rajasthan Royals – 8.56 economy rate**
🏆 **Kolkata Knight Riders – 8.75 economy rate**
🏆 **Punjab Kings (9.01) & Lucknow Super Giants (9.21) round out the top five.**

Key Insights

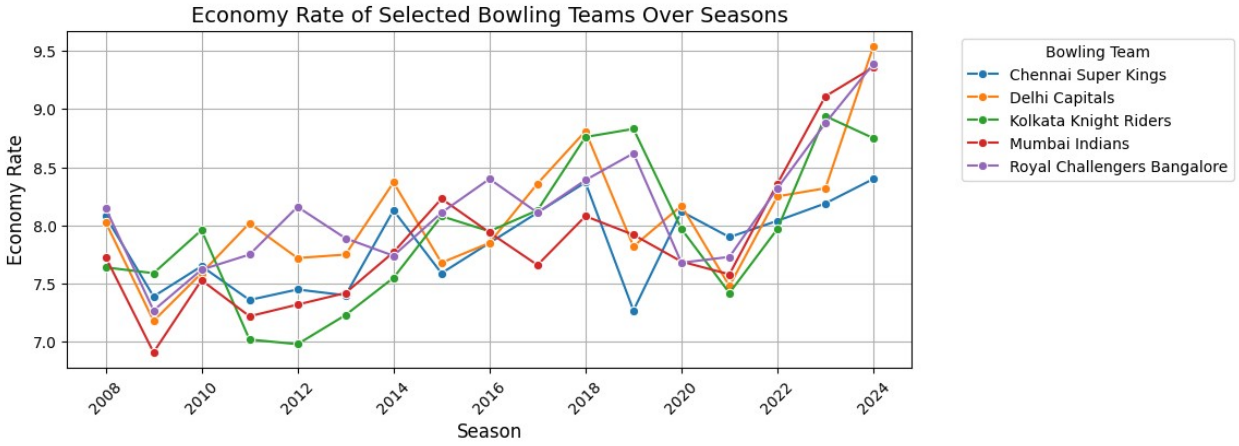
- **Chennai Super Kings (8.40) had the most disciplined bowling attack**, maintaining the best economy rate.
- **Rajasthan Royals (8.56) and KKR (8.75) also displayed strong control over opposition scoring.**
- **Punjab Kings (9.01) managed to stay under the 9.00 mark, showcasing a balanced bowling unit.**
- **Lucknow Super Giants (9.21) and Gujarat Titans (9.34) struggled slightly but remained competitive.**
- **Mumbai Indians (9.36) & RCB (9.39) had similar economy rates, indicating high-scoring encounters.**
- **Delhi Capitals (9.54) & Sunrisers Hyderabad (9.58) had the most expensive bowling attacks, conceding runs freely.**

CSK's disciplined bowling made them the toughest team to score against.
Teams with economy rates above 9.50 struggled to contain the aggressive batters in IPL 2024.

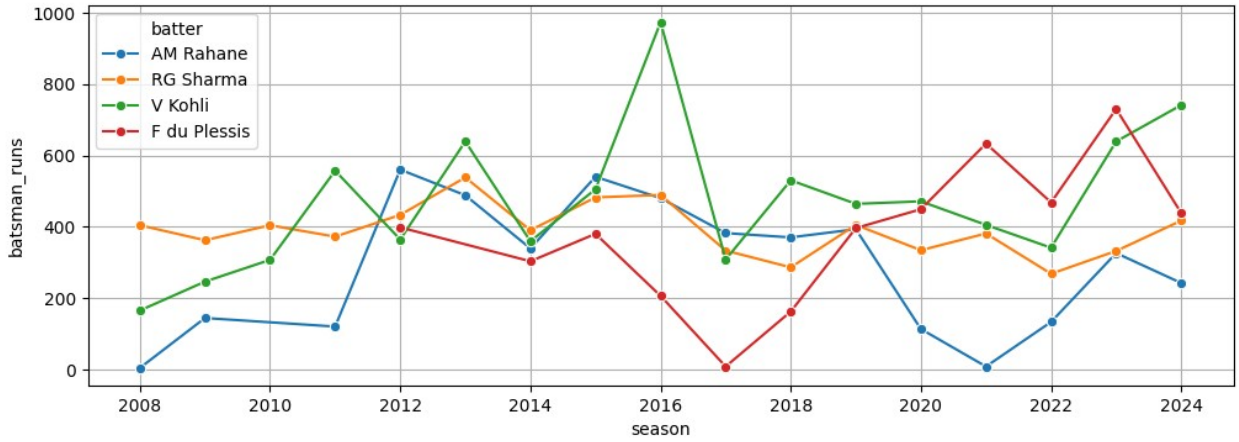
```
import matplotlib.pyplot as plt
import seaborn as sns
import math

# Aggregate total runs conceded and total balls bowled by each bowling
team per season
filter_data = ipl.groupby(['season', 'bowling_team']).agg(
    total_runs=('total_runs', 'sum'),
    total_ball=('match_id', 'count')  # Counting deliveries bowled
).reset_index()

# Calculate the economy rate for each team
filter_data['economy'] = ((filter_data['total_runs'] /
filter_data['total_ball']) * 6).round(2)

# Get unique teams dynamically
teams = filter_data['bowling_team'].unique()
num_teams = len(teams)


# Create subplots
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(18, 4 * 4))
axes = axes.flatten()  # Flatten for easy iteration

# Plot each team's economy rate in its own subplot
for i, team in enumerate(teams):
    team_df = filter_data[filter_data['bowling_team'] == team]

    sns.lineplot(data=team_df, x='season', y='economy', marker='o',
ax=axes[i], palette='cool')

    axes[i].set_title(f"{team} - Economy Rate", fontsize=12,
fontweight='bold')
    axes[i].set_xlabel("Season", fontsize=10)
    axes[i].set_ylabel("Economy Rate", fontsize=10)
    axes[i].grid(True, linestyle="--", alpha=0.5)
    axes[i].set_xticks(team_df['season'].unique())
    axes[i].set_xticklabels(team_df['season'].unique(), rotation=45)



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


```
# Aggregate total runs conceded and total balls bowled by each bowling
team per season
filter_data = ipl.groupby(['season', 'bowling_team']).agg(
    total_runs=('total_runs', 'sum'),
    total_ball=('match_id', 'count')  # Counting deliveries bowled
).reset_index()

# Calculate the economy rate
filter_data['economy'] = ((filter_data['total_runs'] /
filter_data['total_ball']) * 6).round(2)

# Select 5 teams for visualization
selected_teams = ['Chennai Super Kings', 'Mumbai Indians', 'Royal
Challengers Bangalore',
                  'Kolkata Knight Riders', 'Delhi Capitals']
```

```
filtered_df =
filter_data[filter_data['bowling_team'].isin(selected_teams)]

# Plot line chart for selected teams
plt.figure(figsize=(10, 4))
sns.lineplot(data=filtered_df, x='season', y='economy',
hue='bowling_team', marker='o')

# Customize the plot
plt.title("Economy Rate of Selected Bowling Teams Over Seasons",
fontsize=14)
plt.xlabel("Season", fontsize=12)
plt.ylabel("Economy Rate", fontsize=12)
plt.xticks(rotation=45)
plt.legend(title="Bowling Team", bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.grid(True)
plt.show()
```




```
xk  =ipl[(ipl['batter'].isin(['AM Rahane','F du Plessis','V Kohli','RG
Sharma']))].groupby(['season', 'batter'])
['batsman_runs'].sum().reset_index()
xk
plt.figure(figsize=(12,4))
sns.lineplot(data=xk,x='season',y='batsman_runs',hue='batter',marker='
o',)
plt.grid(True)
plt.show()
```




```
dismissal_counts = ipl['dismissal_kind'].value_counts()
dismissal_counts
```

```
dismissal_kind
caught                    8063
bowled                    2212
run out                   1114
lbw                        800
caught and bowled          367
stumped                    358
retired hurt                15
hit wicket                  15
obstructing the field        3
retired out                  3
Name: count, dtype: int64
```

Wicket Types & Dismissal Patterns

This function calculate the economy rate of each team based on year and overs

```
def bowling_economy_stats(ipl, start_year: int = None,
                          end_year: int = None,
```

```
                          start_over: int = 1,
                          end_over: int = 6) -> pd.DataFrame:
    '''
    Function: bowling_economy_stats
    Author: Nikesh
    Date: March 28, 2025

    Description:
    This function calculates the economy rate (runs conceded per over)
of each bowling team
    within a specified range of overs.

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details.
    - start_year (int, optional): The starting year for filtering
data.
    - end_year (int, optional): The ending year for filtering data.
    - start_over (int, optional): The starting over for analysis
(default is 1).
    - end_over (int, optional): The ending over for analysis (default
is 6).

    Returns:
    - DataFrame: A table with columns ['bowling_team', 'total_runs',
'balls_bowled', 'economy'],
                 showing the economy rate of each team, sorted in
ascending order.
    '''
    # Make a copy of the dataset to avoid modifying the original
    filter_data = ipl.copy()

    # Filter based on season
    if start_year is not None:
        filter_data = filter_data[filter_data['season'] >= start_year]
    if end_year is not None:
        filter_data = filter_data[filter_data['season'] <= end_year]

    # Filter based on over range
    over_data = filter_data[(filter_data['over'] >= start_over) &
(filter_data['over'] <= end_over)]

    # Aggregate total runs conceded by each bowling team
    total_runs_df = over_data.groupby('bowling_team')
['total_runs'].sum().reset_index()

    # Count the number of balls bowled by each team
    total_balls_df =
over_data.groupby('bowling_team').size().reset_index(name='balls_bowle
d')
```

```
    # Merge both DataFrames
    bowling_stats = pd.merge(total_runs_df, total_balls_df,
on='bowling_team')

    # Calculate economy rate (runs conceded per over)
    bowling_stats['economy'] = ((bowling_stats['total_runs'] /
bowling_stats['balls_bowled']) * 6).round(2)

    # Sort teams by best economy rate (ascending order)
    best_bowling_team = bowling_stats.sort_values(by='economy',
ascending=True)

    return best_bowling_team
```

Team best economy rate in power play in season 2024

```
bowling_economy_stats(ipl, start_year=2024, end_year=2024,
start_over=0, end_over=5)
```


```
{"summary":"{\n  \"name\": \"bowling_economy_stats(ipl,
start_year=2024, end_year=2024, start_over=0, end_over=5)\",\n
\"rows\": 10,\n  \"fields\": [\n    {\n      \"column\":
\"bowling_team\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"Royal Challengers Bangalore\",\n          \"Chennai
Super Kings\",\n          \"Gujarat Titans\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
74,\n        \"min\": 686,\n        \"max\": 910,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          879,\n
737,\n          686\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"balls_bowled\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 37,\n        \"min\": 446,\n
\"max\": 595,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          558,\n          518,\n          446\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"economy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5326829367561072,\n        \"min\": 8.3,\n        \"max\": 10.09,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          9.45,\n
```

```
8.54,\n          9.23\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


🏏 Best Economy Rate in Power Play – IPL 2024

Top Bowling Economies in Power Play

🔹 **Rajasthan Royals – 8.30 economy rate**
🔹 **Chennai Super Kings – 8.54 economy rate**
🔹 **Mumbai Indians – 8.66 economy rate**
🔹 **Punjab Kings (8.74) & Sunrisers Hyderabad (9.18) rounded out the top five.**

Key Insights

- **Rajasthan Royals (8.30) had the most economical bowling attack in the powerplay, restricting opposition batters early.**
- **Chennai Super Kings (8.54) and Mumbai Indians (8.66) maintained control with disciplined bowling in the first six overs.**
- **Punjab Kings (8.74) kept things tight, while Sunrisers Hyderabad (9.18) struggled slightly but stayed competitive.**
- **Gujarat Titans (9.23) & KKR (9.29) had moderate control but leaked runs under pressure.**
- **Lucknow Super Giants (9.44) & Royal Challengers Bangalore (9.45) found it challenging to contain openers.**
- **Delhi Capitals (10.09) had the worst economy rate, conceding the most runs in the powerplay.**

🔹 **A strong powerplay economy is crucial for early dominance.**
🔹 **RR & CSK excelled in restricting opposition in the first six overs, giving them an edge! 🚀**

Team best economy rate in Death over in season 2024

```
bowling_economy_stats(ipl, start_year=2024, end_year=2024,
start_over=14, end_over=19)
```


```
{"summary":"{\n  \"name\": \"bowling_economy_stats(ipl,
start_year=2024, end_year=2024, start_over=14, end_over=19)\",\n
\"rows\": 10,\n  \"fields\": [\n    {\n      \"column\":
\"bowling_team\",\n      \"properties\": {\n        \"dtype\":
```

\"string\",\n   \"num_unique_values\": 10,\n   \"samples\": [\n   \"Gujarat Titans\",\n   \"Kolkata Knight Riders\",\n   \"Kings XI Punjab\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"total_runs\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 106,\n   \"min\": 643,\n   \"max\": 938,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   644,\n   643,\n   911\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"balls_bowled\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 61,\n   \"min\": 349,\n   \"max\": 558,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   349,\n   415,\n   527\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"economy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.7116498827060646,\n   \"min\": 9.06,\n   \"max\": 11.14,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   11.07,\n   9.3,\n   10.37\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

🏏 Best Economy Rate in Death Overs – IPL 2024

Top Bowling Economies in Death Overs

🏏 **Chennai Super Kings – 9.06 economy rate**
🏏 **Kolkata Knight Riders – 9.30 economy rate**
🏏 **Royal Challengers Bangalore – 9.59 economy rate**
🏏 **Rajasthan Royals (9.66) & Mumbai Indians (10.34) completed the top five.**

Key Insights

- **Chennai Super Kings (9.06) had the most disciplined bowling attack in death overs, restricting big hits effectively.**
- **Kolkata Knight Riders (9.30) also displayed strong death-over control, preventing late-game run explosions.**
- **Royal Challengers Bangalore (9.59) & Rajasthan Royals (9.66) maintained respectable economy rates in the final overs.**
- **Mumbai Indians (10.34) & Punjab Kings (10.37) leaked runs but stayed competitive.**
- **Sunrisers Hyderabad (10.41) & Lucknow Super Giants (10.49) struggled under pressure.**

- **Gujarat Titans (11.07) & Delhi Capitals (11.14) conceded the most runs in the death overs, highlighting their bowling weakness.**

Death-over economy is crucial for closing out games.
CSK & KKR mastered this phase, giving them a tactical advantage!

Venue-based Insights

this function calculate the average run of a venue based on year and inning

```
def avg_run_by_venue(ipl: pd.DataFrame, start_year: int = None,
end_year: int = None, inning: int = None) -> pd.DataFrame:
    """
    Calculates the average runs per match at each venue, with a
dynamically named column.

    Parameters:
    - ipl (DataFrame): The IPL dataset.
    - start_year (int, optional): Filter for matches from this year
onward.
    - end_year (int, optional): Filter for matches up to this year.
    - inning (int, optional): Filter for a specific inning (1 or 2).

    Returns:
    - DataFrame with venue-wise average runs, with a dynamically named
column.
    """

    filter_data = ipl.copy()

    if start_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year >=
start_year]
    if end_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year <=
end_year]
    if inning is not None:
        filter_data = filter_data[filter_data['inning'] == inning]

    # Create a dynamic column name
    avg_col_name = "avg_runs"
    if start_year and end_year and start_year == end_year:
        avg_col_name += f"_{start_year}"
    if inning:
        avg_col_name += f"_in_{inning}{'st' if inning == 1 else 'nd'}"
```

```
    result = filter_data.groupby(['venue']).agg(
        total_runs=('total_runs', 'sum'),
        total_match=('match_id', 'nunique')
    ).reset_index()

    # Compute the dynamically named column
    result[avg_col_name] = result['total_runs'] /
result['total_match']

    return result  # Drop intermediate columns


avg_run_by_venue(ipl,start_year=2024,end_year=2024,inning=1)
```


```
{"summary":"{\n  \"name\":
\"avg_run_by_venue(ipl,start_year=2024,end_year=2024,inning=1)\",\n
\"rows\": 13,\n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
443,\n        \"min\": 144,\n        \"max\": 1529,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          936,\n
1380,\n          1176\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          1,\n          8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"avg_runs_2024_in_1st\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
25.28788602442055,\n        \"min\": 144.0,\n        \"max\": 235.2,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          187.2,\n
172.5,\n          235.2\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


Average run of each venue in– IPL 2024

```
avg_run_by_venue(ipl,start_year=2024,end_year=2024)
```


```
{"summary":"{\n  \"name\":
\"avg_run_by_venue(ipl,start_year=2024,end_year=2024)\",\n  \"rows\":
```

```
13,\n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
872,\n        \"min\": 289,\n        \"max\": 2897,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          1853,\n
2742,\n          2232\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          1,\n          8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"avg_runs_2024\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
40.1978391479814,\n        \"min\": 289.0,\n        \"max\": 446.4,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          370.6,\n
342.75,\n          446.4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


🏏 Average Runs at Each IPL 2024 Venue

Top High-Scoring Venues

🔹 **Arun Jaitley Stadium, Delhi – 446.4 average runs**
🔹 **Rajiv Gandhi International Stadium, Hyderabad – 398.8 average runs**
🔹 **Eden Gardens, Kolkata – 393.4 average runs**
🔹 **Dr. Y.S. Rajasekhara Reddy ACA-VDCA Stadium (400) & M. Chinnaswamy Stadium (377.1) completed the top five.**

Key Insights

- **Delhi's Arun Jaitley Stadium (446.4) was the highest-scoring venue, favoring batters heavily.**
- **Hyderabad (398.8) & Kolkata (393.4) also witnessed high-scoring games, indicating batting-friendly pitches.**
- **Visakhapatnam (400) provided a balanced contest but remained a venue for big runs.**
- **Bengaluru (377.1) & Jaipur (370.6) continued their trend of being high-scoring grounds.**

- **Mumbai's Wankhede Stadium (364.1) & Dharamshala (364.0) remained reliable batting venues.**
- **Ahmedabad (342.7) & Lucknow (345.6) had slightly lower average scores, suggesting some bowling assistance.**
- **Chepauk (321.8) had the lowest average among major venues, indicating tougher conditions for batters.**

The IPL 2024 season saw record-breaking totals at multiple venues, highlighting the dominance of batters across different grounds!

average run in each venue in 1st inning 2024

```
avg_run_by_venue(ipl,start_year=2024,end_year=2024,inning=1)
```


```
{"summary":"{\n  \"name\":
\"avg_run_by_venue(ipl,start_year=2024,end_year=2024,inning=1)\",\n
\"rows\": 13,\n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
443,\n        \"min\": 144,\n        \"max\": 1529,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          936,\n
1380,\n          1176\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          1,\n          8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"avg_runs_2024_in_1st\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
25.28788602442055,\n        \"min\": 144.0,\n        \"max\": 235.2,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          187.2,\n
172.5,\n          235.2\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


Average run of each venue in 2nd – IPL 2024

```
avg_run_by_venue(ipl,start_year=2024,end_year=2024,inning=2)
```

```
{"summary":"{\n  \"name\":
\"avg_run_by_venue(ipl,start_year=2024,end_year=2024,inning=2)\",\n
\"rows\": 13,\n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_runs\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
430,\n        \"min\": 145,\n        \"max\": 1369,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          917,\n
1362,\n          1056\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          1,\n          8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"avg_runs_2024_in_2nd\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
18.5285374320818,\n        \"min\": 145.0,\n        \"max\": 211.2,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          183.4,\n
170.25,\n          211.2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


Venue-Wise 2nd Innings Average Runs – IPL 2024

Key Takeaways

- **Arun Jaitley Stadium, Delhi (211 runs)** had the highest 2nd innings average, indicating a strong batting-friendly surface for chases.
- ⚠ **Barsapara Cricket Stadium, Guwahati (145 runs)** recorded the lowest average, suggesting tough conditions for chasing.
- **Eden Gardens (196), Rajiv Gandhi International Stadium (194), and Sawai Mansingh Stadium (183)** proved to be favorable for 2nd innings batting.
- **MA Chidambaram Stadium, Chennai (152 runs)** had the lowest chasing average among major venues, reinforcing its reputation for favoring bowlers in the second innings.
- **Most venues had an average 2nd innings score between 160-200**, suggesting competitive yet challenging chases.

These stats highlight **the best and toughest grounds for chasing in IPL 2024**, impacting team strategies and match results!

Venue with the highest number of sixes.

```
def count_boundaries_by_venue(ipl: pd.DataFrame,
                              start_year: int = None,
                              end_year: int = None,
                              inning: int = None,
                              boundary_type: int = None) ->
pd.DataFrame:
    '''
Function: count_boundaries_by_venue
    Author: Nikesh
    Date: March 25, 2025

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details, with
columns including
      'date', 'venue', 'total_runs', and 'match_id'.
    - start_year (int, optional): The starting year to filter the
data.
      If None, the function includes all years.
    - end_year (int, optional): The ending year to filter the data.
      If None, the function includes all years.
    - inning (int, optional): The specific inning (1 or 2) to filter
the data.
      If None, the function includes both innings.
    - boundary_type (int, optional): The type of boundary (4 for
fours, 6 for sixes).
      If None, the function includes both fours and sixes.

    Returns:
    - DataFrame: A table with columns ['venue', 'total_matches',
'total_boundaries_<dynamic>'],
      showing the total number of boundaries hit per venue,
considering all filters.
      The 'total_boundaries' column name dynamically adjusts based on
the applied filters.
    '''

    # Apply boundary filtering
    filter_data = ipl.copy()

    if boundary_type is not None:
        if boundary_type in [4, 6]:
            filter_data = filter_data[filter_data['total_runs'] ==
boundary_type]
    else:
```

```
        filter_data = filter_data[(filter_data['total_runs'] == 4) |
(filter_data['total_runs'] == 6)]

    # Apply year filtering
    if start_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year >=
start_year]
    if end_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year <=
end_year]

    # Apply inning filtering
    if inning in [1, 2]:
        filter_data = filter_data[filter_data['inning'] == inning]

    # Create dynamic column names
    boundary_label = f"_{boundary_type}" if boundary_type else "_all"
    year_label = f"_{start_year}" if start_year == end_year and
start_year else "_all_years"
    inning_label = f"_in_{inning}{'st' if inning == 1 else 'nd'}" if
inning else ""

    boundary_col_name = f"total_boundaries{boundary_label}{year_label}
{inning_label}"

    # Group by venue and calculate total boundaries
    result = filter_data.groupby('venue').agg(
        total_matches=('match_id', 'nunique'),
        **{boundary_col_name: ('match_id', 'count')}
    ).reset_index()

    return result
count_boundaries_by_venue(ipl,2024,2024)
```


```
{"summary":"{\n  \"name\":
\"count_boundaries_by_venue(ipl,2024,2024)\",\n  \"rows\": 13,\n
\"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_matches\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 1,\n        \"max\": 9,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          5,\n
1,\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_boundaries_all_2024\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 116,\n        \"min\": 39,\n
```

\"max\": 396,\n   \"num_unique_values\": 13,\n
\"samples\": [\n   230,\n   357,\n   329\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

Total number of boundaries in each venue including 4 and 6 – IPL 2024

```
count_boundaries_by_venue(ipl,2024,2024)
```

{"summary":"{\n \"name\":
\"count_boundaries_by_venue(ipl,2024,2024)\",\n \"rows\": 13,\n
\"fields\": [\n   {\n   \"column\": \"venue\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n   \"samples\": [\n   \"Sawai
Mansingh Stadium, Jaipur\",\n   \"Narendra Modi Stadium,
Ahmedabad\",\n   \"Arun Jaitley Stadium, Delhi\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"total_matches\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
2,\n   \"min\": 1,\n   \"max\": 9,\n
\"num_unique_values\": 7,\n   \"samples\": [\n   5,\n
1,\n   8\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"total_boundaries_all_2024\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 116,\n   \"min\": 39,\n
\"max\": 396,\n   \"num_unique_values\": 13,\n
\"samples\": [\n   230,\n   357,\n   329\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

Venue-Wise Total Boundaries – IPL 2024

Key Insights

- **Eden Gardens, Kolkata (396 boundaries)** recorded the most boundaries, indicating a high-scoring venue.
- **Arun Jaitley Stadium, Delhi (329) and M Chinnaswamy Stadium, Bengaluru (355)** were also boundary-heavy grounds, favoring aggressive batting.
- ⚡ **Narendra Modi Stadium, Ahmedabad (357) & MA Chidambaram Stadium, Chennai (351)** provided balanced scoring opportunities.

- **Barsapara Cricket Stadium, Guwahati (39 boundaries)** had the least, likely due to fewer matches played.

- **Most venues had 300+ boundaries, suggesting an exciting and attacking brand of cricket in IPL 2024.**

These stats highlight **which stadiums provided the most entertainment with fours and sixes**, influencing team strategies and batting approaches!

Total number of boundaries in each venue including 4 and 6 in 2nd inning– IPL 2024

```
count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,inning=2)
```


```
{"summary":"{\n  \"name\":
\"count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,inning=2
)\",\n  \"rows\": 13,\n  \"fields\": [\n    {\n      \"column\":
\"venue\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_matches\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 1,\n        \"max\": 9,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          5,\n
1,\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_boundaries_all_2024_in_2nd\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 59,\n        \"min\": 19,\n
\"max\": 198,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          117,\n          192,\n          155\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Venue-Wise Total Boundaries in 2nd Innings – IPL 2024

Key Insights

- **Eden Gardens, Kolkata (198 boundaries)** recorded the most boundaries in 2nd innings, making it a favorable chasing venue.

- **Narendra Modi Stadium, Ahmedabad (192) & M Chinnaswamy Stadium, Bengaluru (172)** also witnessed high boundary counts in run chases.

- ⚡ **Wankhede Stadium, Mumbai (180) & MA Chidambaram Stadium, Chennai (167)** remained strong batting venues in 2nd innings.

- **Barsapara Cricket Stadium, Guwahati (19 boundaries)** had the least, influenced by fewer matches.

- **Overall, boundaries in 2nd innings were lower compared to the 1st, highlighting possible pitch slowdowns or effective bowling strategies.**

These stats **showcase how venues influenced chasing teams**, impacting match strategies and batting approaches in IPL 2024!

Total number of boundaries in each venue including 4 and 6 in 1st inning– IPL 2024

```
count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,inning=1)
```


```
{"summary":"{\n  \"name\":
\"count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,inning=1
)\",\n  \"rows\": 13,\n  \"fields\": [\n    {\n      \"column\":
\"venue\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_matches\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 1,\n        \"max\": 9,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          5,\n
1,\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_boundaries_all_2024_in_1st\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 57,\n        \"min\": 20,\n
\"max\": 198,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          113,\n          165,\n          174\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Venue-Wise Total Boundaries in 1st Innings – IPL 2024

Key Insights

- **Eden Gardens, Kolkata (198 boundaries)** led the charts for most boundaries in 1st innings, making it a high-scoring venue.

- **M Chinnaswamy Stadium, Bengaluru (183) & MA Chidambaram Stadium, Chennai (184)** provided great batting conditions upfront.
- ⚡ **Narendra Modi Stadium, Ahmedabad (165) & Wankhede Stadium, Mumbai (170)** also saw consistent boundary hitting.
- **Barsapara Cricket Stadium, Guwahati (20 boundaries)** recorded the lowest, likely due to fewer matches played.
- **Boundaries in 1st innings were generally higher than 2nd innings, indicating better pitch conditions early on.**

These statistics **highlight the best batting-friendly venues in IPL 2024**, helping teams plan their first-innings strategies!

Total Sixes Hit at Each IPL 2024 Venue

```
count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,boundary_type=6)
```


```
{"summary":"{\n  \"name\":
\"count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,boundary_type=6)\",\n  \"rows\": 13,\n  \"fields\": [\n    {\n
\"column\": \"venue\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 13,\n        \"samples\":
[\n          \"Sawai Mansingh Stadium, Jaipur\",\n          \"Narendra
Modi Stadium, Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_matches\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          1,\n          8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"total_boundaries_6_2024\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
46,\n        \"min\": 8,\n        \"max\": 153,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          78,\n
123,\n          133\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Total Sixes Hit at Each IPL 2024 Venue

Key Highlights

- **Eden Gardens, Kolkata (153 sixes) & M Chinnaswamy Stadium, Bengaluru (151 sixes)** saw the most six-hitting action, reaffirming their reputation as batting paradises.

- **Rajiv Gandhi International Stadium, Hyderabad (134 sixes) & Arun Jaitley Stadium, Delhi (133 sixes)** also witnessed power-packed performances.
- **Narendra Modi Stadium, Ahmedabad (123 sixes) & Wankhede Stadium, Mumbai (125 sixes)** proved to be friendly for power-hitters.
- **Barsapara Cricket Stadium, Guwahati (8 sixes)** recorded the lowest, primarily due to fewer matches played.
- **MA Chidambaram Stadium, Chennai (112 sixes)** had relatively fewer sixes, possibly due to its spin-friendly conditions.

These numbers highlight **the most explosive venues for six-hitting in IPL 2024**, influencing game strategies for teams and batters alike!

Total fours at Each IPL 2024 Venue

```
count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,boundary_type=4)
```


```
{"summary":"{\n  \"name\":
\"count_boundaries_by_venue(ipl,start_year=2024,end_year=2024,boundary_type=4)\",\n  \"rows\": 13,\n  \"fields\": [\n    {\n
\"column\": \"venue\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 13,\n        \"samples\":
[\n          \"Sawai Mansingh Stadium, Jaipur\",\n          \"Narendra
Modi Stadium, Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_matches\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          1,\n          8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_boundaries_4_2024\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
72,\n        \"min\": 31,\n        \"max\": 243,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          152,\n
234,\n          196\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Total Fours Hit at Each IPL 2024 Venue

Key Observations

- **Eden Gardens, Kolkata (243 fours)** recorded the highest number of fours, highlighting its batting-friendly conditions.

- **MA Chidambaram Stadium, Chennai (239 fours)** and **Narendra Modi Stadium, Ahmedabad (234 fours)** followed closely, showing consistent stroke play.
- **Wankhede Stadium, Mumbai (225 fours) & Arun Jaitley Stadium, Delhi (196 fours)** provided good value for shots along the ground.
- **Barsapara Cricket Stadium, Guwahati (31 fours)** had the least boundaries due to fewer matches played.
- **Spin-friendly venues like Rajiv Gandhi International Stadium, Hyderabad (183 fours) & Sawai Mansingh Stadium, Jaipur (152 fours)** saw relatively fewer boundaries.

These stats emphasize **which venues offered the best batting conditions for stroke-makers in IPL 2024!**

Total number of six and four in 1st and 2nd inning

```
# Count boundaries in each venue for different innings and boundary
types
four_1 = count_boundaries_by_venue(ipl, start_year=2024,
end_year=2024, boundary_type=4, inning=1)
six_1 = count_boundaries_by_venue(ipl, start_year=2024, end_year=2024,
boundary_type=6, inning=1)
four_2 = count_boundaries_by_venue(ipl, start_year=2024,
end_year=2024, boundary_type=4, inning=2)
six_2 = count_boundaries_by_venue(ipl, start_year=2024, end_year=2024,
boundary_type=6, inning=2)

# Rename columns to avoid conflicts
four_1.rename(columns={"total_matches": "total_matches_1st",
"total_boundaries": "total_fours_1st"}, inplace=True)
six_1.rename(columns={"total_matches": "total_matches_1st",
"total_boundaries": "total_sixes_1st"}, inplace=True)
four_2.rename(columns={"total_matches": "total_matches_2nd",
"total_boundaries": "total_fours_2nd"}, inplace=True)
six_2.rename(columns={"total_matches": "total_matches_2nd",
"total_boundaries": "total_sixes_2nd"}, inplace=True)

# Merge DataFrames on 'venue'
all_inning_six_four = (
    four_1
    .merge(six_1, on=['venue', 'total_matches_1st'], how='left')
    .merge(four_2, on='venue', how='left')
    .merge(six_2, on='venue', how='left')
```

```
)

# Fill missing values with 0
all_inning_six_four.fillna(0, inplace=True)

# Display final DataFrame
all_inning_six_four[['venue','total_matches_1st','total_boundaries_4_2
024_in_1st','total_boundaries_4_2024_in_2nd','total_boundaries_6_2024_
in_1st','total_boundaries_6_2024_in_2nd']]
```


```
{"summary":"{\n  \"name\":
\"all_inning_six_four[['venue','total_matches_1st','total_boundaries_4
_2024_in_1st','total_boundaries_4_2024_in_2nd','total_boundaries_6_202
4_in_1st','total_boundaries_6_2024_in_2nd']]\",\n  \"rows\": 13,\n
\"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Sawai
Mansingh Stadium, Jaipur\",\n          \"Narendra Modi Stadium,
Ahmedabad\",\n          \"Arun Jaitley Stadium, Delhi\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_matches_1st\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 1,\n        \"max\": 9,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          5,\n
1,\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_boundaries_4_2024_in_1st\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 35,\n        \"min\": 18,\n
\"max\": 130,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          33,\n          98,\n          69\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_boundaries_4_2024_in_2nd\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 37,\n        \"min\": 13,\n
\"max\": 123,\n        \"num_unique_values\": 13,\n
\"samples\": [\n          83,\n          123,\n          98\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_boundaries_6_2024_in_1st\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 23,\n        \"min\": 2,\n
\"max\": 87,\n        \"num_unique_values\": 13,\n        \"samples\":
[\n          44,\n          54,\n          76\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_boundaries_6_2024_in_2nd\",\
n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 24,\n        \"min\": 6,\n        \"max\": 85,\n
\"num_unique_values\": 13,\n        \"samples\": [\n          34,\n
69,\n          57\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

IPL 2024 Boundary Analysis – First vs Second Inning

This report provides a detailed analysis of the **number of fours and sixes** hit at each venue during the **1st and 2nd innings** in IPL 2024. Understanding these trends helps in assessing **pitch conditions, chasing advantages, and team strategies**.

Key Observations

1. High Boundary Scoring Venues

Some venues witnessed **higher boundary counts** due to **batting-friendly pitches, shorter boundaries, and dew factors** aiding the second innings:

- **Eden Gardens, Kolkata** recorded the highest number of **total sixes (153)** and had a fairly balanced distribution of fours and sixes across both innings.
- **M Chinnaswamy Stadium, Bengaluru**, known for its **short boundaries**, also had high six counts (151).
- **Narendra Modi Stadium, Ahmedabad** was another **boundary-heavy venue**, especially in the second innings.

---

2. First Inning vs Second Inning Boundary Comparison

- Most venues showed a **balanced distribution** of fours between **1st and 2nd innings**.
- However, **six-hitting varied more**, possibly due to **dew effects, pressure handling, and pitch slow-downs**.
- Some venues favored **chasing teams**, where second-inning batters scored more boundaries, **indicating easier batting conditions**.

Venues with More Boundaries in the Second Inning

- **Narendra Modi Stadium, Ahmedabad** – **More sixes (69 in 2nd vs. 54 in 1st)**
- **Wankhede Stadium, Mumbai** – **Higher number of boundaries in the second inning (121 fours & 59 sixes)**
- **Sawai Mansingh Stadium, Jaipur** – **More boundaries in the second inning** (83 fours & 34 sixes)

Venues with More Boundaries in the First Inning

- **Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket Stadium, Visakhapatnam** – **More sixes in 1st inning (27 vs. 18 in 2nd)**

- **Rajiv Gandhi International Stadium, Hyderabad** – **More fours in the first inning (98 vs. 85 in 2nd)**

These differences could be attributed to:
✔ **Dew factor** making it harder for bowlers in the second innings.
✔ **Pitch behavior** changing as the match progresses.
✔ **Pressure situations** affecting batting in chases.

---

3. Balanced Boundary Distribution at Some Venues

Certain venues had **almost equal** distribution of boundaries across both innings, indicating consistent pitch behavior:

- **Arun Jaitley Stadium, Delhi** – 98 fours in both innings, **slight six drop in the second inning**.
- **Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium, Lucknow** – Almost equal boundary count for both fours and sixes.

This suggests that **batting conditions remained similar throughout the match** at these venues.

---

4. Venues with Lower Boundary Counts

Some venues had **fewer boundaries**, possibly due to **slower pitches, larger boundaries, or difficult batting conditions**:

- **Barsapara Cricket Stadium, Guwahati** – The lowest boundary count (**only 18 fours & 2 sixes in 1st innings, 13 fours & 6 sixes in 2nd innings**).

- **Himachal Pradesh Cricket Association Stadium, Dharamshala** – Also had **fewer boundaries**, likely due to **altitude factors and a challenging pitch**.

Conclusion & Strategic Takeaways

Batting-friendly venues like **Eden Gardens, Chinnaswamy Stadium, and Wankhede** will likely continue to be **high-scoring grounds**. Teams may **prefer to chase** at these venues due to dew factors.

Bowler-friendly venues like **Barsapara, Dharamshala, and Ekana Stadium** saw **fewer boundaries**, meaning teams need to **focus on accumulating runs rather than hitting big shots**.

Chasing advantage was evident at certain grounds like **Narendra Modi Stadium & Wankhede Stadium**, where **teams hit more boundaries in the second inning**.

Consistent venues like **Arun Jaitley Stadium & Rajiv Gandhi Stadium** showed **little difference between innings**, making them neutral venues for both batting and bowling strategies.

Understanding these patterns can help teams **adjust their strategies**, such as:
 Choosing to **bowl first** at high-scoring venues to take advantage of dew.
 Focusing on **rotating strike** rather than hitting sixes on bowler-friendly pitches.
 Making **bowling changes** based on how pitches behave in different innings.

 IPL 2024 has showcased a variety of batting conditions across venues, making boundary analysis **a crucial part of match strategies!**

Most wickets taken at a single venue.

Which venue has the highest win percentage for teams batting first (1st inning) and teams chasing (2nd inning)

```
def
win_percent_by_venue_by_inning(ipl,start_year:int=None,end_year:int=No
ne,inning:int=1)->pd.DataFrame:
  '''
    Function: win_percentage_by_venue
    Author: Nikesh
    Date: March 25, 2025


    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details,
including columns such as
      'date', 'venue', 'inning', 'batting_team', 'winner', and
'match_id'.
    - start_year (int, optional): The starting year to filter the
data.
      If None, the function includes all years.
    - end_year (int, optional): The ending year to filter the data.
      If None, the function includes all years.
    - inning (int, optional): The inning (1 or 2) for which to
calculate the win percentage.
      Default is 1.

    Returns:
    - DataFrame: A table with columns ['venue', 'total_match',
'total_match_won', 'win percent in <inning>'],
      showing the win percentage of the batting team at each venue in
```

```
descending order.
      The column name dynamically adjusts based on the selected
inning.
  '''
  # year_filter
  filter_data = ipl.copy()
  # filter based on year
  if start_year is not None:
    filter_data = filter_data[filter_data['date'].dt.year>=start_year]
  if end_year is not None:
    filter_data = filter_data[filter_data['date'].dt.year <= end_year]

  # filter based on inning
  if inning in ([1,2]):
    filter_data = filter_data[filter_data['inning']==inning]

  # calculating column name based on 1 and 2 inning

    column_name = f'win percent in {inning}{"st"if inning==1
else"nd"}'




  venue_total_match_df = filter_data.groupby('venue')
['match_id'].nunique().reset_index()

  venue_total_match_won =
filter_data[filter_data['batting_team']==filter_data['winner']].groupb
y('venue')['match_id'].nunique().reset_index()

  # rename column name match_id to total_match at each venue

venue_total_match_df.rename(columns={'match_id':'total_match'},inplace
=True)
  # rename column name match_id to total_match at each venue

venue_total_match_won.rename(columns={'match_id':'total_match_won'},in
place=True)

  # merging into one dataframe
  final_data =
venue_total_match_df.merge(venue_total_match_won,on='venue',how='left'
).fillna(0)

  # calculating winning percentage
  final_data[column_name] =
((final_data['total_match_won']/final_data['total_match'])*100).round(
2)

  return final_data.sort_values(by=column_name,ascending=False)
```

```
win_percent_by_venue_by_inning(ipl,2024,2024,1)
```


```
{"summary":"{\n  \"name\":
\"win_percent_by_venue_by_inning(ipl,2024,2024,1)\",\n  \"rows\": 13,\
n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n
\"Narendra Modi Stadium, Ahmedabad\",\n          \"MA Chidambaram
Stadium, Chepauk, Chennai\",\n          \"Arun Jaitley Stadium,
Delhi\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          2,\n          8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_match_won\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.2608503439122303,\n        \"min\": 0.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          5.0,\n
2.0,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"win
percent in 1st\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 27.754619781772714,\n        \"min\":
0.0,\n        \"max\": 100.0,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          25.0,\n          57.14,\n          33.33\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


win percentage of first batting team in each venue

```
win_percent_by_venue_by_inning(ipl,2024,2024,1)
```


```
{"summary":"{\n  \"name\":
\"win_percent_by_venue_by_inning(ipl,2024,2024,1)\",\n  \"rows\": 13,\
n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n
\"Narendra Modi Stadium, Ahmedabad\",\n          \"MA Chidambaram
Stadium, Chepauk, Chennai\",\n          \"Arun Jaitley Stadium,
Delhi\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
```

\"max\": 9,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   5,\n   2,\n   8\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"total_match_won\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.2608503439122303,\n   \"min\": 0.0,\n   \"max\": 5.0,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   5.0,\n   2.0,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"win percent in 1st\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 27.754619781772714,\n   \"min\": 0.0,\n   \"max\": 100.0,\n   \"num_unique_values\": 9,\n   \"samples\": [\n   25.0,\n   57.14,\n   33.33\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Conclusion

The analysis of **win percentages for teams batting first in IPL 2024** highlights the **impact of venue conditions** on match outcomes.

- **Venues Favoring Batting First**
  - **Arun Jaitley Stadium (Delhi)** and **Dr. Y.S. Rajasekhara Reddy ACA-VDCA Stadium (Visakhapatnam)** had a **100% success rate**, making them ideal grounds for setting a target.
  - **Wankhede Stadium (Mumbai)** also provided a decent advantage for batting first teams with a **57.14% win rate**.
- **Venues Favoring Chasing**
  - **Barsapara Cricket Stadium (Guwahati)** had a **0% success rate for batting first teams**, indicating an overwhelming advantage for chasing teams.
  - **Narendra Modi Stadium (Ahmedabad), M Chinnaswamy Stadium (Bengaluru), and MA Chidambaram Stadium (Chennai)** also showed **low win rates for batting first teams**, favoring teams batting second.

Final Thoughts

The decision to **bat first or chase** should be carefully **venue-specific**, with certain grounds **clearly favoring one strategy over the other**. Teams and captains can leverage this data for **better match planning and strategy execution** in future IPL games.

Venue with the lowest win percentage for teams batting first.

win percentages for teams batting first in IPL 2024

```
win_percent_by_venue_by_inning(ipl,2024,2024,2)
```


```
{"summary":"{\n  \"name\":
\"win_percent_by_venue_by_inning(ipl,2024,2024,2)\",\n  \"rows\": 13,\
n  \"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 13,\n        \"samples\": [\n          \"Arun
Jaitley Stadium, Delhi\",\n          \"Dr. Y.S. Rajasekhara Reddy ACA-
VDCA Cricket Stadium, Visakhapatnam\",\n          \"Sawai Mansingh
Stadium, Jaipur\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"total_match\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          5,\n          7,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_match_won\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.7722938923964167,\n        \"min\": 0.0,\n        \"max\": 5.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          3.0,\n
4.0,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"win
percent in 2nd\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 24.711306246539387,\n        \"min\":
0.0,\n        \"max\": 60.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          57.14,\n          40.0,\n          60.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Conclusion: Win Percentage of Teams Batting Second in IPL 2024

The analysis of **win percentages for teams batting second** reveals significant **venue-specific trends** that influence match outcomes.

Venues Favoring Chasing Teams

- **Sawai Mansingh Stadium (Jaipur)** had the **highest success rate (60.00%)** for teams chasing a target.

- **Bharat Ratna Shri Atal Bihari Vajpayee Ekana Cricket Stadium (Lucknow)** and **MA Chidambaram Stadium (Chennai)** also showed **strong win percentages (57.14% and 55.56%)**, making them favorable for chasing.
- **Rajiv Gandhi International Stadium (Hyderabad) and Narendra Modi Stadium (Ahmedabad)** had an **equal success rate for both innings (50.00%)**, indicating balanced conditions.

Venues Favoring Batting First

- **Arun Jaitley Stadium (Delhi), Himachal Pradesh Cricket Association Stadium (Dharamshala), and Dr. Y.S. Rajasekhara Reddy ACA-VDCA Stadium (Visakhapatnam)** had **0% win rate for chasing teams**, making them highly favorable for setting a target.
- **M. Chinnaswamy Stadium (Bengaluru) had a low chasing success rate (14.29%)**, reinforcing its reputation as a high-scoring venue where scoreboard pressure plays a crucial role.

Key Takeaways

- **Jaipur, Lucknow, and Chennai** are strong venues for **chasing teams**.
- **Delhi, Dharamshala, and Visakhapatnam** favor teams **batting first**.
- **Balanced venues** like **Ahmedabad and Hyderabad** provide **equal opportunities** for both strategies.

Strategic Implications

Captains and teams can **adjust their approach based on the venue**, using data-driven decisions to **choose whether to bat first or chase**. This insight is crucial for **match-winning strategies** in IPL 2024!

Bowler with the most wickets at a single venue.

```
def most_wicket_taker(ipl, inning: int = None,
                      start_year: int = None,
                      end_year: int = None,
                      start_over: int = None,
                      end_over: int = None) -> pd.DataFrame:
    '''
    Function: most_wicket_taker
    Author: Nikesh
    Date: March 28, 2025


    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details.
    - inning (int, optional): The inning (1 or 2) for which to
calculate the wickets.
```

```
    - start_year (int, optional): The starting year to filter the
data.
    - end_year (int, optional): The ending year to filter the data.
    - start_over (int, optional): The starting over for filtering
(default is None).
    - end_over (int, optional): The ending over for filtering (default
is None).

    Returns:
    - DataFrame: A table with columns ['bowler',
'Total_wick_in_<inning>_between_<start_over>_and_<end_over>'],
                 showing the total wickets taken by each bowler.
    '''

    # Copy the original dataset to avoid modifying it
    filter_data = ipl.copy()

    # Filter based on year range
    if start_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year >=
start_year]
    if end_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year <=
end_year]

    # Filter based on inning
    if inning in [1, 2]:
        filter_data = filter_data[filter_data['inning'] == inning]

    # Filter based on over range
    if start_over is not None:
        filter_data = filter_data[filter_data['over'] >= start_over]
    if end_over is not None:
        filter_data = filter_data[filter_data['over'] <= end_over]

    # Column name for total wickets taken
    column_name = f"Total_wick_in_{inning}st" if inning == 1 else
f"Total_wick_in_{inning}nd" if inning == 2 else 'Total_wick'

    # Add over range information to column name
    if start_over is not None and end_over is not None:
        column_name =
f"{column_name}_between_{start_over}_and_{end_over}"
    elif start_over is not None:
        column_name = f"{column_name}_after_{start_over}"
    elif end_over is not None:
        column_name = f"{column_name}_before_{end_over}"

    # Calculate total wickets taken by each bowler
    total_wicket_df = filter_data.groupby('bowler')
```

```
['is_wicket'].sum().reset_index(name=column_name)

    # Sort by total wickets in descending order
    total_wicket_df = total_wicket_df.sort_values(by=column_name,
ascending=False)

    return total_wicket_df
```

Most wicket taker bowler in 1st inning power play 2024 ipl

```
most_wicket_taker(ipl, start_year=2024, end_year=2024, start_over=0,
end_over=6,inning=1).head(35)
```


```
{"summary":"{\n  \"name\": \"most_wicket_taker(ipl, start_year=2024,
end_year=2024, start_over=0, end_over=6,inning=1)\",\n  \"rows\": 35,\
n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 35,\n        \"samples\": [\n          \"V
Kaverappa\",\n          \"Mukesh Kumar\",\n          \"K Rabada\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\":
\"Total_wick_in_1st_between_0_and_6\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          9,\n          7,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Conclusion: Most Wicket-Taker Bowler in the 1st Inning Power Play of the 2024 IPL

- **TA Boult** emerged as the leading wicket-taker in the first inning powerplay of the 2024 IPL, with a total of **9 wickets**.
- **MA Starc** followed closely with **7 wickets**, demonstrating consistent performance in the early stages of the game.
- Other notable bowlers include **Sandeep Sharma**, **N Thushara**, and **Arshdeep Singh**, each taking **5 wickets**.
- The overall distribution of wickets among bowlers indicates a variety of players contributing to the success in the powerplay overs, with several bowlers taking **3 to 4 wickets**.
- The analysis of this dataset highlights the bowlers who thrived during the first six overs, showcasing their ability to take wickets and exert control early in the innings.

This analysis provides insight into which bowlers had the most impact during the powerplay in the 2024 IPL, which could be crucial for strategizing future matches.

Most wicket taker bowler in 2nd inning power play 2024 ipl

```
most_wicket_taker(ipl, start_year=2024, end_year=2024, start_over=0, end_over=5,inning=2).head(35)
```


```
{"summary":"{\n  \"name\": \"most_wicket_taker(ipl, start_year=2024, end_year=2024, start_over=0, end_over=5,inning=2)\",\n  \"rows\": 35,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 35,\n        \"samples\": [\n          \"CV Varun\",\n          \"Harshit Rana\",\n          \"JJ Bumrah\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Total_wick_in_2nd_between_0_and_5\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n        \"max\": 6,\n        \"num_unique_values\": 6,\n        \"samples\": [\n          6,\n          5,\n          1\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Conclusion: Most Wicket-Taker Bowler in the 2nd Inning Power Play of the 2024 IPL

- **B Kumar** and **PJ Cummins** were the most successful bowlers in the second inning powerplay, each securing **6 wickets**.
- **I Sharma**, **Yash Dayal**, **TU Deshpande**, and **KK Ahmed** followed closely with **5 wickets**, showing their effectiveness with the new ball.
- **DL Chahar**, **MA Starc**, **VG Arora**, and **Swapnil Singh** contributed significantly with **4 wickets** each.
- Several bowlers took **2 to 3 wickets**, highlighting a balanced distribution of wickets among multiple players.
- The performance of these bowlers in the powerplay overs was crucial in restricting opponents and setting the momentum for the bowling team.

This analysis showcases the impact bowlers had during the powerplay in the second innings of the 2024 IPL, helping teams control the game early on.

Most wicket taker bowler in 1st inning Death overs 2024 ipl

```
most_wicket_taker(ipl, start_year=2024, end_year=2024, start_over=14,
end_over=19,inning=1).head(35)
```


```
{"summary":"{\n  \"name\": \"most_wicket_taker(ipl, start_year=2024,
end_year=2024, start_over=14, end_over=19,inning=1)\",\n  \"rows\":
35,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 35,\n        \"samples\": [\n          \"PJ
Cummins\",\n          \"RD Chahar\",\n          \"JD Unadkat\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\":
\"Total_wick_in_1st_between_14_and_19\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3,\n        \"min\": 2,\n
\"max\": 17,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          3,\n          11,\n          6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Conclusion: Most Wicket-Taker Bowlers in the 1st Inning Death Overs of the 2024 IPL

- **HV Patel** dominated the death overs in the first innings, taking an impressive **17 wickets**, making him the most lethal bowler in this phase.
- **JJ Bumrah** followed with **11 wickets**, reaffirming his status as a premier death-over specialist.
- **Avesh Khan (10 wickets)** and **Arshdeep Singh (9 wickets)** also proved to be key bowlers in the final overs.
- **TU Deshpande, Sandeep Sharma, and MM Sharma** each took **8 wickets**, contributing significantly to their teams' finishing attacks.
- Several bowlers, including **Mukesh Kumar, HH Pandya, and T Natarajan**, secured between **6 and 5 wickets**, showing their effectiveness in restricting runs and taking crucial breakthroughs.
- The presence of **spinners like SP Narine, YS Chahal, and Ravi Bishnoi** in the list highlights the role of variation in death-over bowling.

This analysis underscores the crucial role these bowlers played in closing out innings, preventing big finishes, and maintaining control in the final overs.

Most wicket taker bowler in 2nd inning Death overs 2024 ipl

```
most_wicket_taker(ipl, start_year=2024, end_year=2024, start_over=14,
end_over=19,inning=2).head(35)
```


```
{"summary":"{\n  \"name\": \"most_wicket_taker(ipl, start_year=2024,
end_year=2024, start_over=14, end_over=19,inning=2)\",\n  \"rows\":
35,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 35,\n        \"samples\": [\n          \"SN
Thakur\",\n          \"SM Curran\",\n          \"SH Johnson\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\":
\"Total_wick_in_2nd_between_14_and_19\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 2,\n
\"max\": 8,\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          8,\n          7,\n          3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


Conclusion: Most Wicket-Taker Bowlers in the 2nd Inning Death Overs of the 2024 IPL

- **Mukesh Kumar** led the wicket-taking tally in the death overs of the second innings, securing **8 wickets**, making him the most impactful bowler in this phase.
- **T Natarajan, Mohammed Siraj, and Harshit Rana** followed closely with **7 wickets each**, proving their effectiveness in closing out matches.
- **G Coetzee, MA Starc, Yash Thakur, and Mustafizur Rahman** each took **6 wickets**, showcasing their ability to execute under pressure.
- **Naveen-ul-Haq, HV Patel, M Pathirana, PJ Cummins, and Yash Dayal** all contributed significantly with **5 wickets** each, providing crucial breakthroughs for their teams.
- A mix of pace and spin was effective in the death overs, with players like **SM Curran, Kuldeep Yadav, Rashid Khan, SP Narine, and YS Chahal** featuring in the list with multiple wickets.
- The presence of **young and emerging bowlers like Rasikh Salam, Nithish Kumar Reddy, and UT Yadav** highlights the rise of new death-over specialists.

This analysis highlights how teams relied on a combination of experienced and young bowlers to restrict opposition in the final overs of the chase.

Find the bowler who dismissed the same batsman the most times in IPL history.

```
def
bower_who_dismissed_same_batsman(ipl,start_year:int=None,end_year:int=
None)->pd.DataFrame:
  '''
  '''
  filter_data= ipl[ipl['is_wicket']==1]

  # filter based on year
  if start_year is not None:
    filter_data = filter_data[filter_data['date'].dt.year >=
start_year]
  if end_year is not None:
    filter_data = filter_data[filter_data['date'].dt.year <= end_year]

  filter_data =
filter_data.groupby(['bowler','player_dismissed']).size().reset_index(
name='total out')
  filter_data.sort_values(by='total out',ascending=False,inplace=True)
  return filter_data
# bower_who_dismissed_same_batsman(ipl,2024,2024)


bower_who_dismissed_same_batsman(ipl,2020,2024).head(30)
```


```
{"summary":"{\n  \"name\":
\"bower_who_dismissed_same_batsman(ipl,2020,2024)\",\n  \"rows\": 30,\
n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 16,\n        \"samples\": [\n          \"YS
Chahal\",\n          \"DL Chahar\",\n          \"Ravi Bishnoi\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"player_dismissed\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 23,\n        \"samples\": [\n          \"GJ
Maxwell\",\n          \"KD Karthik\",\n          \"MA Agarwal\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total out\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 3,\n        \"max\": 5,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          5,\n
4,\n          3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


#Bowler who out the same batsman multiple time from 2020 to 2024

```
bower_who_dismissed_same_batsman(ipl,2020,2024).head(30)
```


```
{"summary":"{\n  \"name\":
\"bower_who_dismissed_same_batsman(ipl,2020,2024)\",\n  \"rows\": 30,\
n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 16,\n        \"samples\": [\n          \"YS
Chahal\",\n          \"DL Chahar\",\n          \"Ravi Bishnoi\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"player_dismissed\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 23,\n        \"samples\": [\n          \"GJ
Maxwell\",\n          \"KD Karthik\",\n          \"MA Agarwal\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total out\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 3,\n        \"max\": 5,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          5,\n
4,\n          3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Conclusion: Bowlers Who Dismissed the Same Batsman Multiple Times (2020-2024)

- **YS Chahal** dominated this category, dismissing **MA Agarwal (5 times), KD Karthik (4 times), N Rana (4 times), and MK Lomror (3 times)**, proving his effectiveness against key batsmen.
- **DL Chahar** troubled **Shubman Gill** the most, getting him out **5 times**.
- **TA Boult and Mohammed Siraj** both dismissed **PP Shaw 4 times**, showing their ability to exploit his weaknesses.
- **Ravi Bishnoi** had multiple victims, dismissing **Ishan Kishan, SA Yadav, and RG Sharma 4 times each**, highlighting his impact against top-order batters.
- **Rashid Khan** remained a threat to various players, taking down **S Dhawan, RG Sharma, DJ Hooda, and MM Ali** multiple times.
- **K Rabada** showed his dominance, dismissing **KL Rahul, WP Saha, and N Pooran** multiple times.
- **Harpreet Brar, CV Varun, and RD Chahar** also featured in the list, making crucial breakthroughs for their teams.

This analysis showcases how specific bowlers consistently managed to dismiss particular batsmen, revealing key player matchups and rivalries in the IPL.

```
def runs_conceded_by_bowler_to_batter(ipl, batter_name: str,
start_year: int = None, end_year: int = None) -> pd.DataFrame:
    '''
    Function: runs_conceded_by_bowler_to_batter
    Author: Nikesh
```

```
    Date: March 28, 2025

    Parameters:
    - ipl (DataFrame): The IPL dataset containing match details.
    - batter_name (str): The name of the batter for whom the data is
required.
    - start_year (int, optional): The starting year for filtering
data.
    - end_year (int, optional): The ending year for filtering data.

    Returns:
    - DataFrame: A sorted table with columns ['bowler', 'batter',
'batsman_runs'],
                 showing how many runs each bowler has conceded to the
given batter.
    '''
    # Make a copy of the dataset to avoid modifying the original
    filter_data = ipl.copy()

    # Filter based on year
    if start_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year >=
start_year]
    if end_year is not None:
        filter_data = filter_data[filter_data['date'].dt.year <=
end_year]

    # Group by bowler and batter, summing the runs scored
    runs_data = (
        filter_data.groupby(['bowler', 'batter'])['batsman_runs']
        .sum()
        .reset_index()
        .sort_values(by='batsman_runs', ascending=False)
    )

    # Filter only for the given batter
    return runs_data[runs_data['batter'] == batter_name]

# Function call example
runs_conceded_by_bowler_to_batter(ipl, batter_name='RD
Gaikwad',start_year=2023,end_year=2024).head(10)
```


```
{"summary":"{\n  \"name\": \"runs_conceded_by_bowler_to_batter(ipl,
batter_name='RD Gaikwad',start_year=2023,end_year=2024)\",\n
\"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"bowler\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n          \"HH
Pandya\",\n          \"B Kumar\",\n          \"Mohsin Khan\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

\"batter\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1,\n   \"samples\":
[\n   \"RD Gaikwad\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"batsman_runs\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 9,\n   \"min\": 26,\n
\"max\": 55,\n   \"num_unique_values\": 8,\n   \"samples\":
[\n   48\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
xx= ipl[ipl['date'].dt.year==2024].groupby(['bowler','batter'])
['batsman_runs'].sum().reset_index().sort_values(ascending=False,by='b
atsman_runs').head(40)
xx
```

{"summary":"{\n \"name\": \"xx\",\n \"rows\": 40,\n \"fields\": [\n
{\n   \"column\": \"bowler\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 29,\n
\"samples\": [\n   \"HV Patel\",\n   \"CV Varun\",\n
\"Rashid Khan\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"batter\",\n   \"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 27,\n   \"samples\": [\n   \"RD
Gaikwad\",\n   \"Shubman Gill\",\n   \"WG Jacks\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"batsman_runs\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6,\n   \"min\": 31,\n   \"max\": 62,\n
\"num_unique_values\": 15,\n   \"samples\": [\n   37,\n
34,\n   62\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"xx"}

Answer Here.

4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

Chart - 1

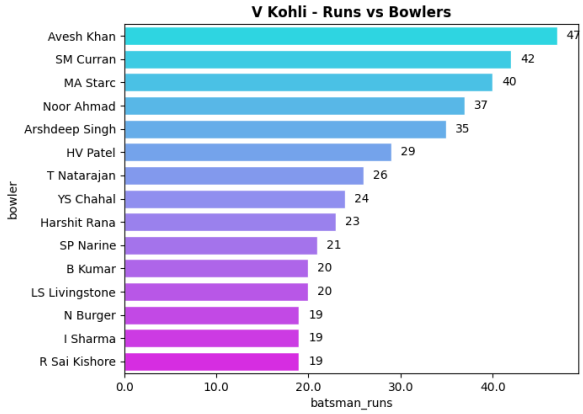
```
sixes = ipl[ipl['total_runs'] ==
6].groupby(ipl['venue']).size().reset_index(name='sixes')
fours = ipl[ipl['total_runs'] ==
4].groupby(ipl['venue']).size().reset_index(name='fours')
total_match = ipl.groupby('venue')
```

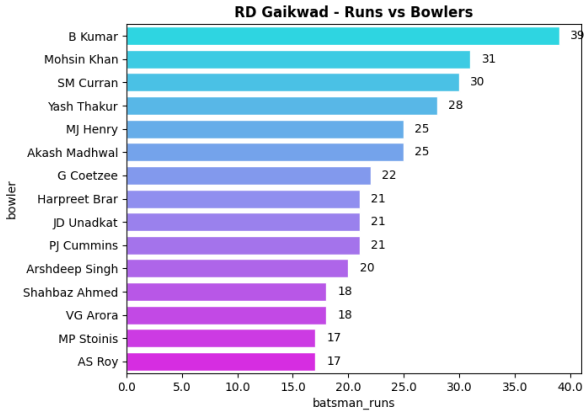
```
['match_id'].nunique().reset_index(name='total_match')
venue_six_four =
total_match.merge(sixes,on='venue',how='left').merge(fours,on='venue',
how='left')
venue_six_four
```

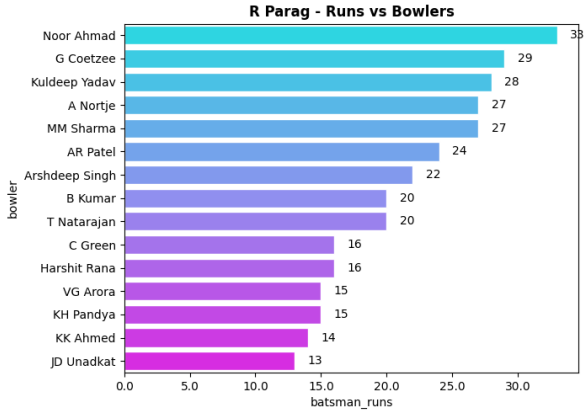

```
{"summary":"{\n  \"name\": \"venue_six_four\",\n  \"rows\": 38,\n
\"fields\": [\n    {\n      \"column\": \"venue\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 38,\n        \"samples\": [\n
\"Subrata Roy Sahara Stadium\",\n          \"Wankhede Stadium,
Mumbai\",\n          \"Brabourne Stadium, Mumbai\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_match\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
32,\n        \"min\": 2,\n        \"max\": 118,\n
\"num_unique_values\": 29,\n        \"samples\": [\n          12,\n
36,\n          94\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"sixes\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 414,\n        \"min\": 7,\n        \"max\": 1590,\n
\"num_unique_values\": 36,\n        \"samples\": [\n          72,\n
155,\n          104\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fours\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 923,\n        \"min\": 43,\n        \"max\": 3479,\n
\"num_unique_values\": 38,\n        \"samples\": [\n          367,\n
3479,\n          889\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"venue_six_four"}
```

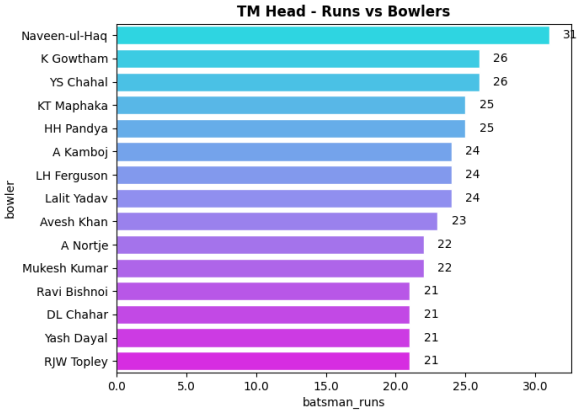

```
# Function to get runs conceded by bowlers against a batsman
def runs_conceded_by_bowler_to_batter(ipl, batter_name, start_year,
end_year):
    return (
        ipl[(ipl['batter'] == batter_name) &
(ipl['season'].between(start_year, end_year))]
        .groupby("bowler")["batsman_runs"]
        .sum()
        .reset_index()
        .sort_values(by="batsman_runs", ascending=False)
    )

# Function to plot subplots for the top 20 batsmen
def plot_top_batsmen_vs_bowlers(ipl, top_n=20):
    # Get the top 20 batsmen dynamically based on total runs
    top_batsmen = (
        ipl[ipl['season'] == 2024]
```

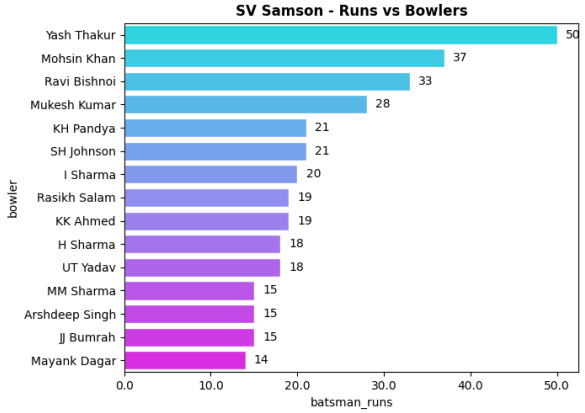
```
        .groupby('batter')['batsman_runs']
        .sum()
        .reset_index()
        .sort_values(by='batsman_runs', ascending=False)
        .head(top_n)['batter']
        .tolist()
    )

    # Create subplots: 10 rows, 2 columns
    fig, axes = plt.subplots(nrows=10, ncols=2, figsize=(15, 50))
    axes = axes.flatten()  # Flatten axes to make it iterable

    # Set theme
    sns.set_theme(style="darkgrid")

    # Loop through each batsman and plot
    for idx, batter in enumerate(top_batsmen):
        ax = axes[idx]  # Get the current subplot
        batter_data = runs_conceded_by_bowler_to_batter(ipl, batter,
2024, 2024).head(15)

        # If no data, skip
        if batter_data.empty:
            continue

        # Barplot with color gradient
        colors = sns.color_palette("cool", len(batter_data))
        sns.barplot(data=batter_data, x="batsman_runs", y="bowler",
palette=colors, ax=ax)

        # Add labels to bars
        for index, value in enumerate(batter_data["batsman_runs"]):
            ax.text(value + 1, index, str(value), va="center",
fontsize=10, color="black")

        # Rotate y-axis labels
        ax.set_yticklabels(ax.get_yticklabels(), rotation=0,
fontsize=10)
        ax.set_xticklabels(ax.get_xticks(), fontsize=10)

        # Set title
        ax.set_title(f"{batter} - Runs vs Bowlers", fontsize=12,
fontweight="bold")

    # Adjust layout
    plt.tight_layout()
    plt.show()

# Call function to generate subplots
plot_top_batsmen_vs_bowlers(ipl)
```

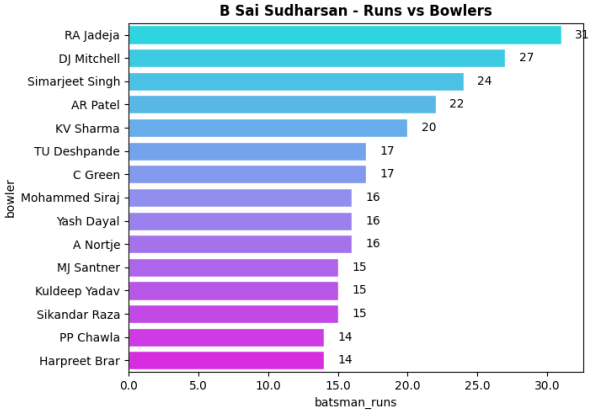

V Kohli - Runs vs Bowlers
Avesh Khan 47
SM Curran 42
MA Starc 40
Noor Ahmad 37
Arshdeep Singh 35
HV Patel 29
T Natarajan 26
YS Chahal 24
Harshit Rana 23
SP Narine 21
B Kumar 20
LS Livingstone 20
N Burger 19
I Sharma 19
R Sai Kishore 19
bowler
batsman_runs

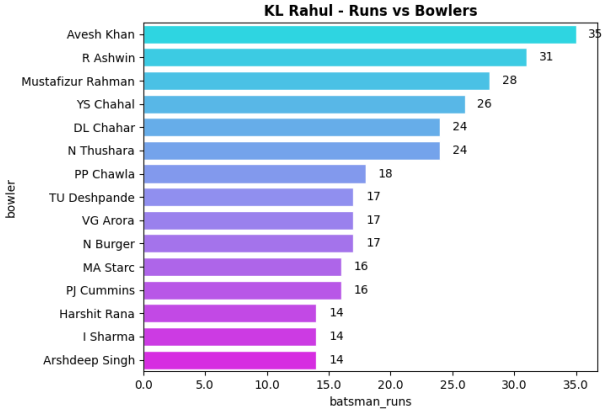

RD Gaikwad - Runs vs Bowlers
B Kumar 39
Mohsin Khan 31
SM Curran 30
Yash Thakur 28
MJ Henry 25
Akash Madhwal 25
G Coetzee 22
Harpreet Brar 21
JD Unadkat 21
PJ Cummins 21
Arshdeep Singh 20
Shahbaz Ahmed 18
VG Arora 18
MP Stoinis 17
AS Roy 17
bowler
batsman_runs

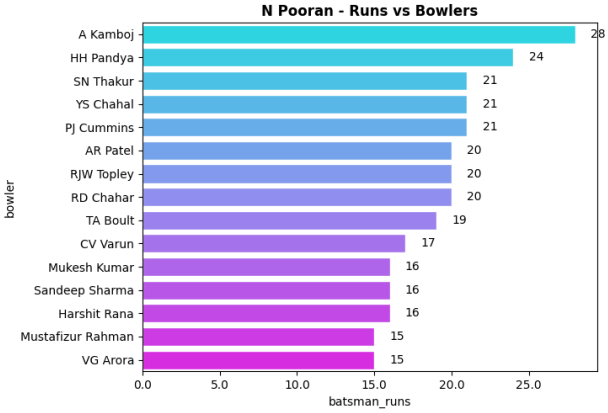

R Parag - Runs vs Bowlers
Noor Ahmad 33
G Coetzee 29
Kuldeep Yadav 28
A Nortje 27
MM Sharma 27
AR Patel 24
Arshdeep Singh 22
B Kumar 20
T Natarajan 20
C Green 16
Harshit Rana 16
VG Arora 15
KH Pandya 15
KK Ahmed 14
JD Unadkat 13
bowler
batsman_runs


TM Head - Runs vs Bowlers
Naveen-ul-Haq 31
K Gowtham 26
YS Chahal 26
KT Maphaka 25
HH Pandya 25
A Kamboj 24
LH Ferguson 24
Lalit Yadav 24
Avesh Khan 23
A Nortje 22
Mukesh Kumar 22
Ravi Bishnoi 21
DL Chahar 21
Yash Dayal 21
RJW Topley 21
bowler
batsman_runs


SV Samson - Runs vs Bowlers
Yash Thakur 50
Mohsin Khan 37
Ravi Bishnoi 33
Mukesh Kumar 28
KH Pandya 21
SH Johnson 21
I Sharma 20
Rasikh Salam 19
KK Ahmed 19
H Sharma 18
UT Yadav 18
MM Sharma 15
Arshdeep Singh 15
JJ Bumrah 15
Mayank Dagar 14
bowler
batsman_runs


B Sai Sudharsan - Runs vs Bowlers
RA Jadeja 31
DJ Mitchell 27
Simarjeet Singh 24
AR Patel 22
KV Sharma 20
TU Deshpande 17
C Green 17
Mohammed Siraj 16
Yash Dayal 16
A Nortje 16
MJ Santner 15
Kuldeep Yadav 15
Sikandar Raza 15
PP Chawla 14
Harpreet Brar 14
bowler
batsman_runs


KL Rahul - Runs vs Bowlers
Avesh Khan 35
R Ashwin 31
Mustafizur Rahman 28
YS Chahal 26
DL Chahar 24
N Thushara 24
PP Chawla 18
TU Deshpande 17
VG Arora 17
N Burger 17
MA Starc 16
PJ Cummins 16
Harshit Rana 14
I Sharma 14
Arshdeep Singh 14
bowler
batsman_runs


N Pooran - Runs vs Bowlers
A Kamboj 28
HH Pandya 24
SN Thakur 21
YS Chahal 21
PJ Cummins 21
AR Patel 20
RJW Topley 20
RD Chahar 20
TA Boult 19
CV Varun 17
Mukesh Kumar 16
Sandeep Sharma 16
Harshit Rana 16
Mustafizur Rahman 15
VG Arora 15
bowler
batsman_runs


SP Narine - Runs vs Bowlers
R Ashwin 34


Abhishek Sharma - Runs vs Bowlers
Yash Thakur 43

How do strike rates of batters in the powerplay (overs 0-5) compare to the death overs (overs 14-19) in IPL 2024, and what insights can be drawn from their total runs and balls faced

This question aligns with your dual scatter plot visualization and encourages interpretation of the data trends.

```
power_play = batsman_with_the_highest_strike_rate(ipl, 2024, 2024, 0,
5, 60).head(50)

# Create subplots
fig, axis = plt.subplots(ncols=1, nrows=2, figsize=(18, 15))  #
Adjusted figure size

# Scatter plot on axis[0]
sns.scatterplot(
    data=power_play,
    x='total_runs',
    y='strike_rate',
    hue='total_ball_faced',
    palette='cool',
    size='total_ball_faced',
    sizes=(20, 200),  # Scale marker size dynamically
    ax=axis[0]
)

# Adding horizontal reference lines
axis[0].axhline(150, color='red', linestyle="--", linewidth=1.5)
axis[0].axhline(200, color='red', linestyle="--", linewidth=1.5)

# Adding vertical reference lines
for x in [200, 300, 350, 400]:
    axis[0].axvline(x, color='red', linestyle="--", linewidth=1.5)

# Annotating batter names
for i in range(len(power_play)):
    axis[0].text(
        power_play['total_runs'].iloc[i] + 5,  # Shift x slightly for
better placement
        power_play['strike_rate'].iloc[i] + 2,  # Shift y slightly
        power_play['batter'].iloc[i],
        fontsize=9,
        color='black',

    )

# Set titles and labels
axis[0].set_title("Best Batters with Highest Strike Rate in
```

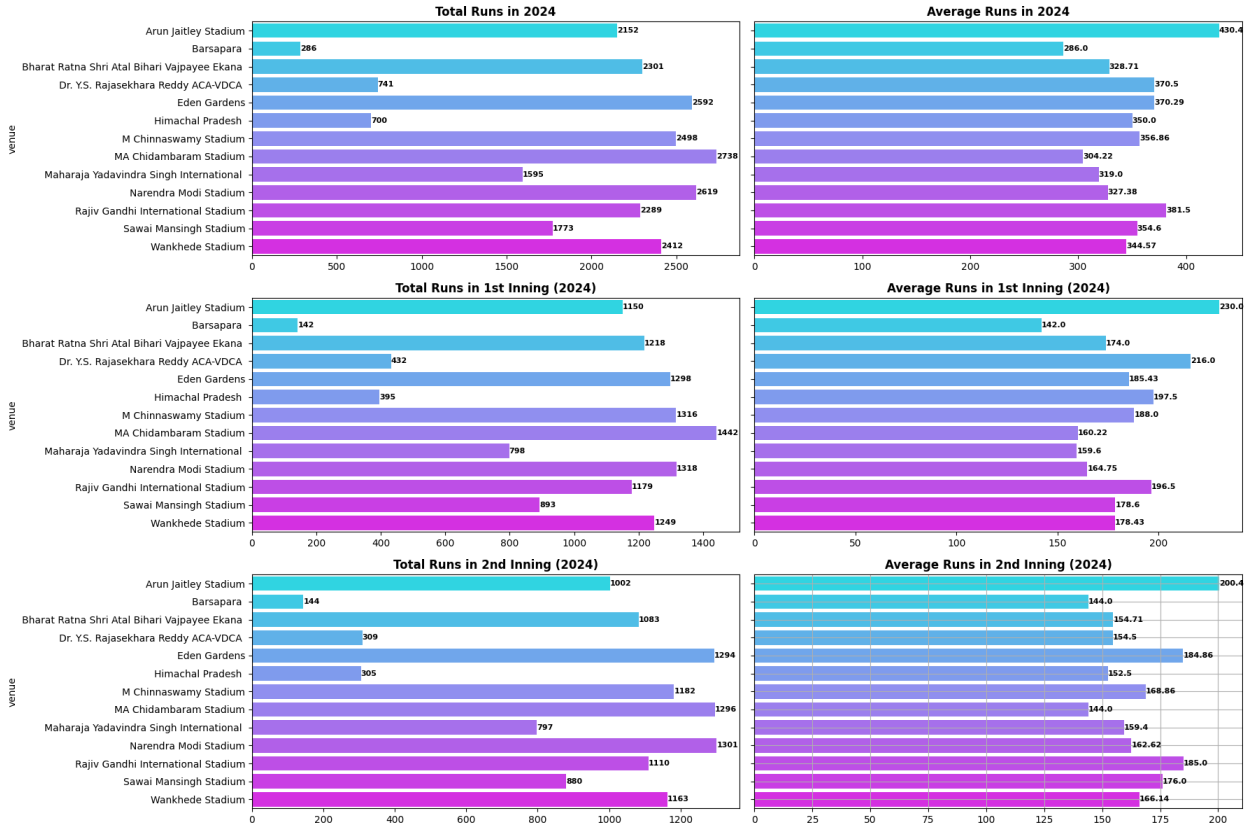
```
(Powerplay) - IPL 2024", fontsize=14, fontweight='bold')
axis[0].set_xlabel("Total Runs", fontsize=12)
axis[0].set_ylabel("Strike Rate", fontsize=12)

# Adding grid for better readability
axis[0].grid(True, linestyle="--", alpha=0.6)


#
----------------------------------------------------------------------
--------------------

death_over = batsman_with_the_highest_strike_rate(ipl, 2024, 2024, 14,
19, 60).head(50)



# Scatter plot on axis[1]
sns.scatterplot(
    data=death_over,
    x='total_runs',
    y='strike_rate',
    hue='total_ball_faced',
    palette='cool',
    size='total_ball_faced',
    sizes=(20, 200),  # Scale marker size dynamically
    ax=axis[1]
)

# Adding horizontal reference lines
axis[1].axhline(150, color='red', linestyle="--", linewidth=1.5)
axis[1].axhline(180, color='red', linestyle="--", linewidth=1.5)

# Adding vertical reference lines
for x in [100, 130, 150, 200]:
    axis[1].axvline(x, color='red', linestyle="--", linewidth=1.5)

# Annotating batter names
for i in range(len(death_over)):
    axis[1].text(
        death_over['total_runs'].iloc[i] +1,  # Shift x slightly for
better placement
        death_over['strike_rate'].iloc[i] +1,  # Shift y slightly
        death_over['batter'].iloc[i],
        fontsize=9,
        color='black',

    )

# Set titles and labels
```

```
axis[1].set_title("Best  Batters with Highest Strike Rate in (Death over) - IPL 2024", fontsize=14, fontweight='bold')
axis[1].set_xlabel("Total Runs", fontsize=12)
axis[1].set_ylabel("Strike Rate", fontsize=12)

# Adding grid for better readability
axis[1].grid(True, linestyle="--", alpha=0.6)
```

Which venue recorded the highest total and average runs in IPL 2024 across both innings? how do first-inning and second-inning runs compare across venues, and which venues were more favorable for batting or chasing?

```
# Chart - 3 visualization code
data = calculate_venue_run_statistics(ipl,2024,2024)

# 1st inning data
first_inning_data = calculate_venue_run_statistics(ipl,2024,2024,1)
first_inning_data.rename(columns={'total_runs':'1st inning
```

```
total','average_runs':'1st inning avg'},inplace=True)


# 2nd inning
second_inning_data = calculate_venue_run_statistics(ipl,2024,2024,2)
second_inning_data.rename(columns={'total_runs':'2nd inning
total','average_runs':'2nd inning avg'},inplace=True)

final_data =
data.merge(first_inning_data,on='venue').merge(second_inning_data,on='
venue')
final_data['venue'] =
final_data['venue'].str.split(',').str[0].str.split('Cricket').str[0]
final_data.drop(columns=['total_match_x','total_match_y'],inplace=True
)

plt.style.use('default')


fig,axes = plt.subplots(ncols=2,nrows=3,figsize=(18,12),sharey=True)
axes = axes.flatten()

plot_columns = [
    ('total_runs', 'Total Runs in 2024'),
    ('average_runs', 'Average Runs in 2024'),
    ('1st inning total', 'Total Runs in 1st Inning (2024)'),
    ('1st inning avg', 'Average Runs in 1st Inning (2024)'),
    ('2nd inning total', 'Total Runs in 2nd Inning (2024)'),
    ('2nd inning avg', 'Average Runs in 2nd Inning (2024)')
]

for index,(col,title) in enumerate(plot_columns):

sns.barplot(data=final_data,x=col,y='venue',palette='cool',ax=axes[ind
ex])

  for i,v in enumerate(final_data[col]):
    axes[index].text(v,i,str(v), va='center', fontsize=8,
fontweight='bold')


  axes[index].set_title(title,fontweight='bold')
  axes[index].set_xlabel(None)
plt.tight_layout()
plt.grid(True)
plt.show()
```



1. Why did you pick the specific chart?

Answer Here.

2. What is/are the insight(s) found from the chart?

Answer Here

3. Will the gained insights help creating a positive business impact?

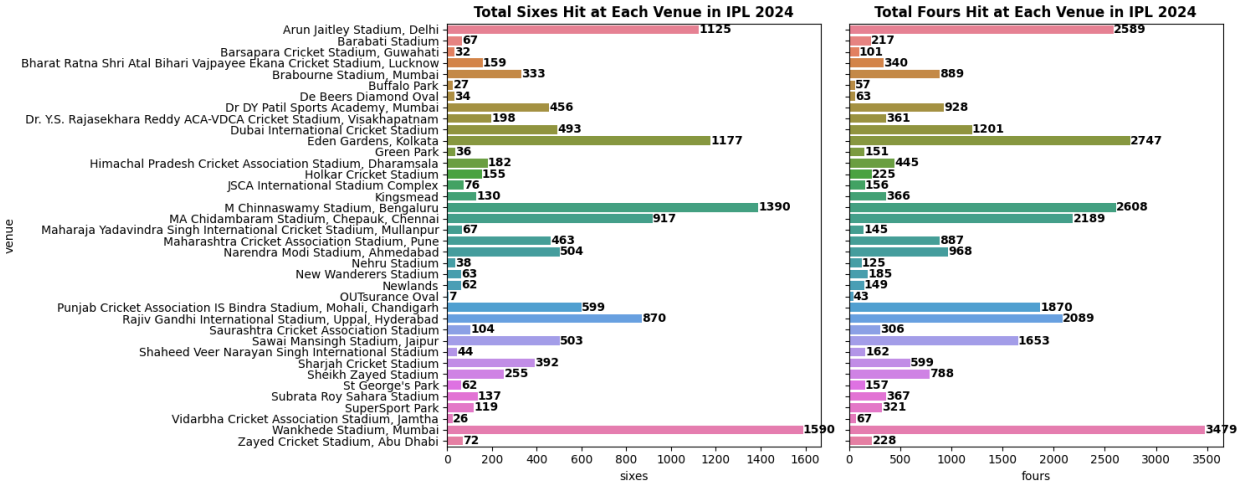
Are there any insights that lead to negative growth? Justify with specific reason.

Answer Here

From IPL 2008 to 2024 venues witnessed the most sixes and fours?

```
# Chart - 4 visualization code
sixes = ipl[ipl['total_runs'] ==
6].groupby(ipl['venue']).size().reset_index(name='sixes')
fours = ipl[ipl['total_runs'] ==
4].groupby(ipl['venue']).size().reset_index(name='fours')
total_match = ipl.groupby('venue')
['match_id'].nunique().reset_index(name='total_match')
venue_six_four =
```

```
total_match.merge(sixes,on='venue',how='left').merge(fours,on='venue',
how='left')
venue_six_four

fig,axis = plt.subplots(ncols=2,nrows=1,figsize=(15,6),sharey=True)
axis= axis.flatten()


sns.barplot(data=venue_six_four,y='venue',x='sixes',ax=axis[0],hue='ve
nue')
sns.barplot(data=venue_six_four,y='venue',x='fours',ax=axis[1],hue='ve
nue')

for i, v in enumerate(venue_six_four['fours']):
  axis[1].text(v,i,str(v),va='center',fontsize=10, fontweight='bold')

for i, v in enumerate(venue_six_four['sixes']):
    axis[0].text(v, i, str(v), va='center', fontsize=10,
fontweight='bold')

axis[0].set_title("Total Sixes Hit at Each Venue in IPL 2024",
fontweight='bold')
axis[1].set_title("Total Fours Hit at Each Venue in IPL 2024",
fontweight='bold')


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



1. Why did you pick the specific chart?

Answer Here.

2. What is/are the insight(s) found from the chart?

Answer Here

3. Will the gained insights help creating a positive business impact?

Are there any insights that lead to negative growth? Justify with specific reason.

Answer Here

Chart - 5

```
# Chart - 5 visualization code

total_bat_win = ipl[(ipl['winner']== ipl['team1']) &
(ipl['inning']==1)].groupby('venue')
['match_id'].nunique().reset_index(name='batting_won_total')
total_win_venue = ipl.groupby('venue')
['match_id'].nunique().reset_index(name='total_won')

winning_per_venue =
pd.merge(total_bat_win,total_win_venue,on='venue',how='right')
winning_per_venue['first_bat_win_percent'] =
((winning_per_venue['batting_won_total']/winning_per_venue['total_won'
]) * 100).round(2)
winning_per_venue['second_bat_win_percent'] = 100 -
winning_per_venue['first_bat_win_percent']
winning_per_venue.sort_values(by='first_bat_win_percent',ascending=Fal
se,inplace=True)
winning_per_venue
fig,axis = plt.subplots(ncols=2,nrows=2,figsize=(15,15),sharey=True)
axis=axis.flatten()

sns.barplot(data=winning_per_venue.head(33),x='total_won',y='venue',pa
lette='cool',ax=axis[0])
sns.barplot(data=winning_per_venue.head(33),x='batting_won_total',y='v
enue',palette='cool',ax=axis[1])
sns.barplot(data=winning_per_venue.head(33),x='first_bat_win_percent',
y='venue',palette='cool',ax=axis[2])
sns.barplot(data=winning_per_venue.head(33),x='second_bat_win_percent'
,y='venue',palette='cool',ax=axis[3])

for i,v in enumerate(winning_per_venue['total_won'].head(33)):
  axis[0].text(v,i,v,va='center',fontsize=10, fontweight='bold')

for i,v in enumerate(winning_per_venue['batting_won_total'].head(33)):
  axis[1].text(v,i,v,va='center',fontsize=10, fontweight='bold')

for i,v in
enumerate(winning_per_venue['first_bat_win_percent'].head(33)):
  axis[2].text(v,i,v,va='center',fontsize=10, fontweight='bold')

for i,v in
enumerate(winning_per_venue['second_bat_win_percent'].head(33)):
  axis[3].text(v,i,v,va='center',fontsize=10, fontweight='bold')
```

```
for i in range(4):
  axis[i].set_xlabel(None)

for i,v in enumerate(['total won','1st batting team won','first bat
win percentage','2nd bat win percentage']):
  axis[i].set_title(v)

plt.show()
```

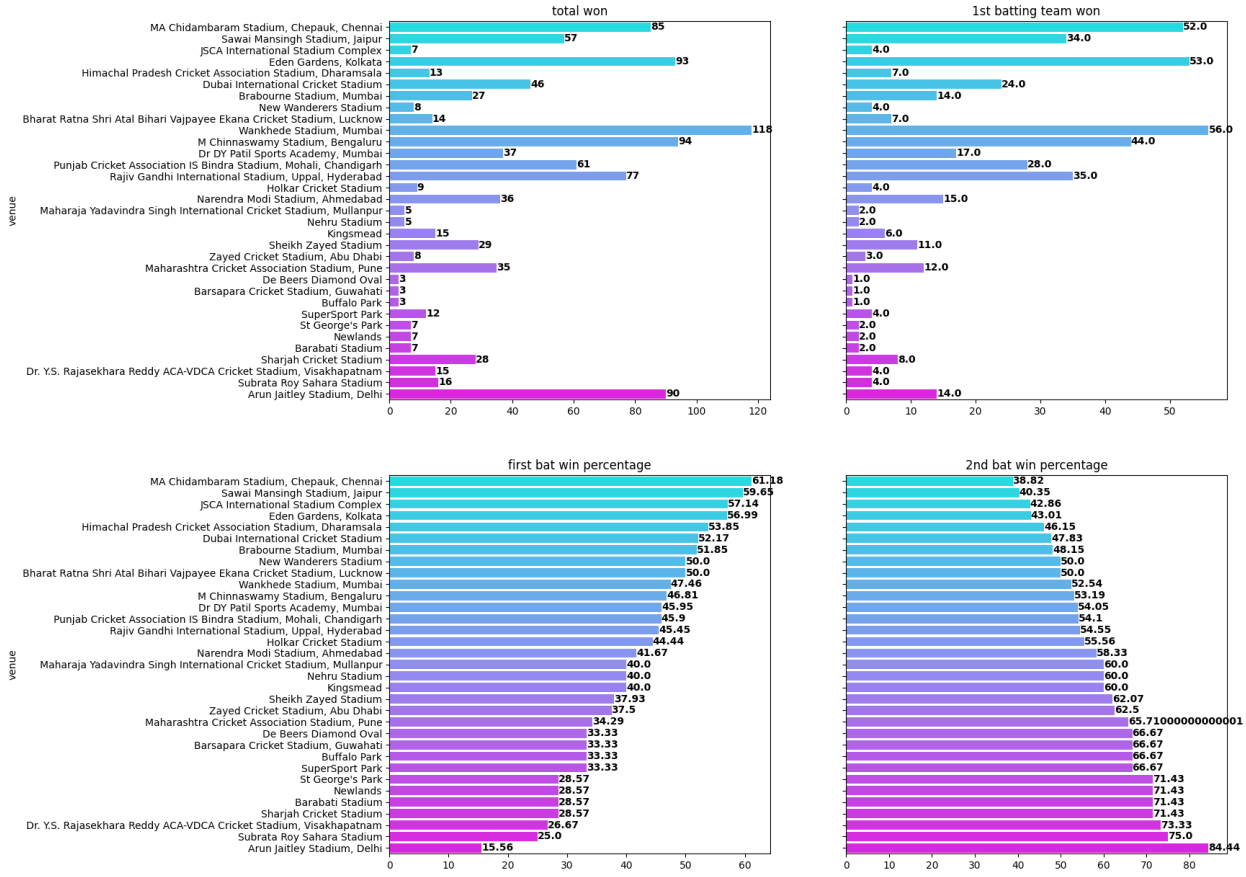


Chart - 6

```
# Chart - 6 visualization code
top_20_batter_name = top_scorer_batsman_by_years(ipl,2024).head(50)
['batter'].unique()



player_score =  top_scorer_batsman_by_years(ipl,2024)
player_score =
player_score[player_score['batter'].isin(top_20_batter_name)]

# Plot Box Plot
plt.figure(figsize=(15, 10))
sns.boxplot(data=player_score, x='best_score', y='batter',
```

```
palette='cool')

plt.title("Score Distribution of Players in IPL 2024", fontsize=14,
fontweight='bold')
plt.xlabel("Match Scores", fontsize=12)
plt.ylabel("Players", fontsize=12)
plt.grid(True, linestyle="--", alpha=0.5)

plt.show()
```

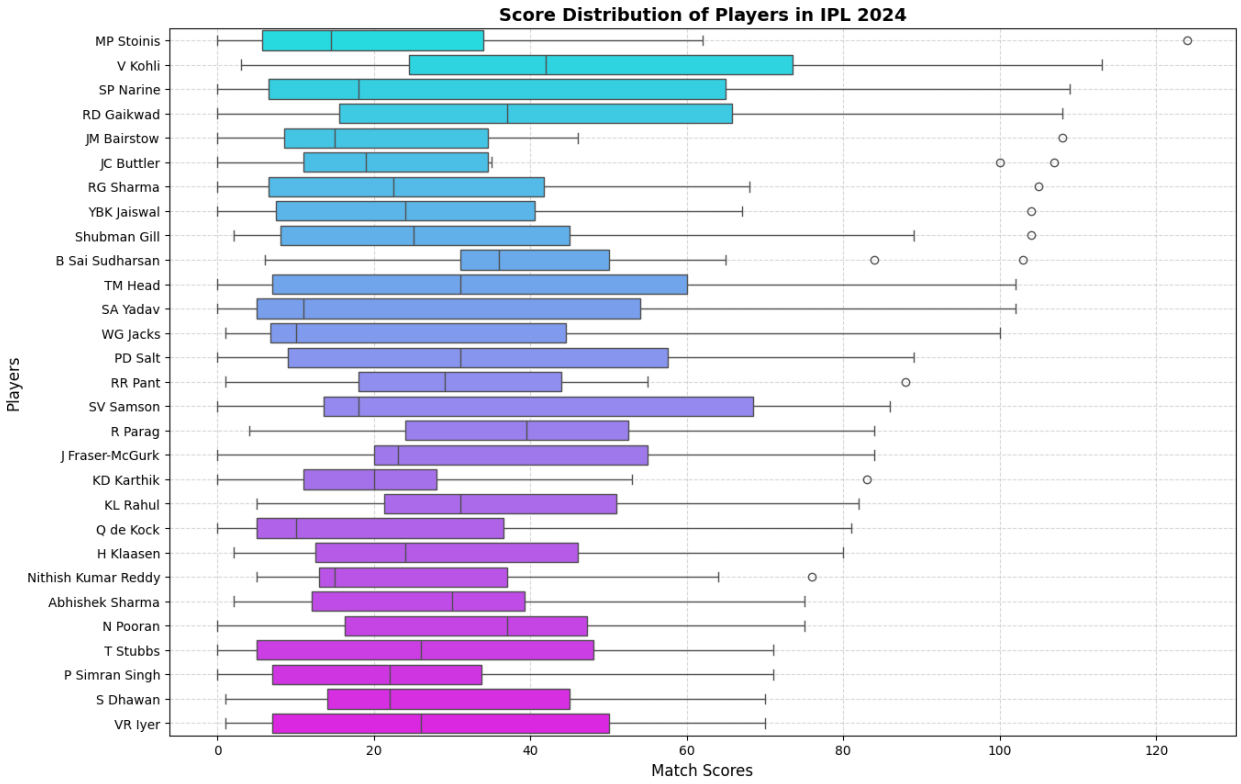


Chart - 7

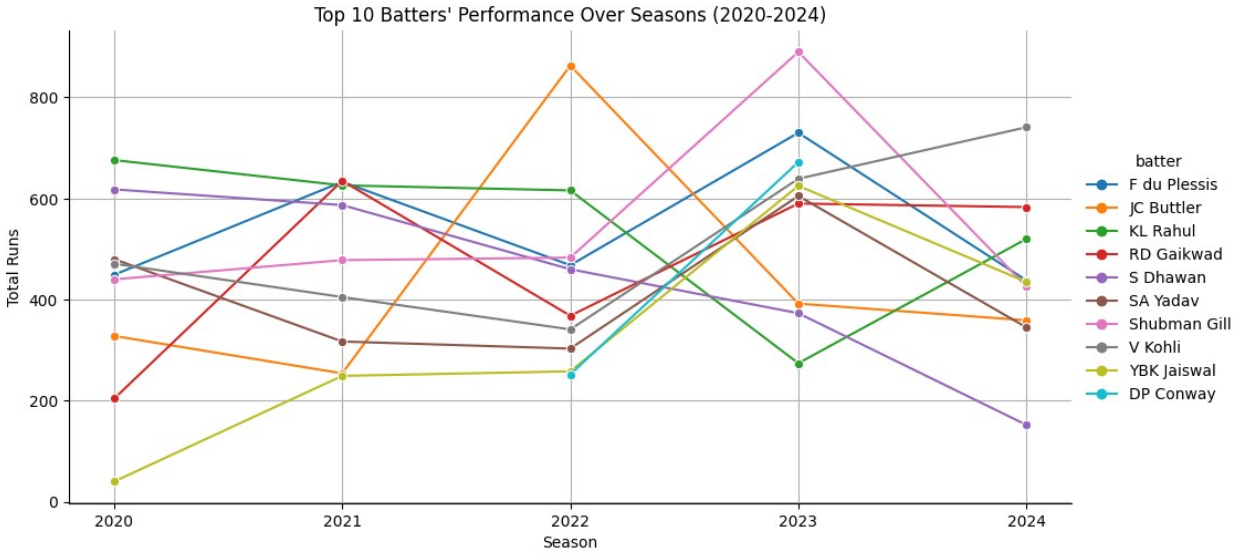
```
# Get top batters' scores
top_bat_score = top_scorer_batsman_by_years(ipl, 2024).head(20)

# Create a unique identifier for duplicate names
top_bat_score['unique_batter'] = top_bat_score['batter'] + ' - ' +
top_bat_score.index.astype(str)

# Set figure size
plt.figure(figsize=(13, 6))

# Create a bar plot
sns.barplot(
    data=top_bat_score,
```

```
    x='best_score',
    y='unique_batter',
    palette='cool',  # More visually appealing color scheme
    edgecolor='black',  # Adds contrast to bars
    linewidth=1.5
)

# Add data labels on bars
for i, v in enumerate(top_bat_score['best_score']):
    plt.text(v + 1, i, str(v), va='center', fontsize=10,
fontweight='bold', color='black')

# Customize plot aesthetics
plt.title("Top 20 Best Scores in IPL 2024", fontsize=14,
fontweight='bold')
plt.xlabel("Best Score", fontsize=12)
plt.ylabel("Batter", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis="x", linestyle="--", alpha=0.7)  # Subtle grid for
readability
sns.despine(left=True, bottom=True)  # Removes unnecessary borders for
a clean look

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

Which batter showed the most improvement or decline in performance between 2020 and 2024

```
batter_data = (
    ipl[ipl['season'].between(2020,2024)].groupby(['season','batter'])
['batsman_runs'].sum().reset_index()
)
batter_data

batter_name = (
    ipl[ipl['season'].between(2020,2024)].groupby(['season','batter'])
['batsman_runs'].sum().reset_index().sort_values(by='batsman_runs',asc
ending=False)['batter'].head(15)
)
batter_name

final_data = batter_data[batter_data['batter'].isin(batter_name)]
final_data['season'] = final_data['season'].astype('str')

sns.relplot(data=final_data,x='season',y='batsman_runs',hue='batter',k
ind='line',aspect=2, marker="o")
plt.title("Top 10 Batters' Performance Over Seasons (2020-2024)")
plt.xlabel("Season")
plt.ylabel("Total Runs")

plt.grid(True)
plt.show()
```

```
<ipython-input-674-e1276f678991>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  final_data['season'] = final_data['season'].astype('str')
```



How has the economy rate of each IPL team evolved across different seasons, and which teams have shown the most improvement or decline over time

```
# Aggregate total runs conceded and total balls bowled by each bowling
team per season
filter_data = ipl.groupby(['season', 'bowling_team']).agg(
    total_runs=('total_runs', 'sum'),
    total_ball=('match_id', 'count')  # Counting deliveries bowled
).reset_index()

# Calculate the economy rate for each team
filter_data['economy'] = ((filter_data['total_runs'] /
filter_data['total_ball']) * 6).round(2)

# Get unique teams dynamically
teams = filter_data['bowling_team'].unique()
num_teams = len(teams)


# Generate a color palette with as many colors as there are teams
colors = sns.color_palette("plasma", num_teams)  # Using "tab10" for
distinct colors

# Create subplots
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(18, 4 * 4))
axes = axes.flatten()  # Flatten for easy iteration

# Plot each team's economy rate in its own subplot
for i, (team, color) in enumerate(zip(teams, colors)):
    team_df = filter_data[filter_data['bowling_team'] == team]

    sns.lineplot(data=team_df, x='season', y='economy', marker='o',
```

```
ax=axes[i], color=color, linewidth=2)

    axes[i].set_title(f"{team} - Economy Rate", fontsize=12,
fontweight='bold', color=color)
    axes[i].set_xlabel("Season", fontsize=10)
    axes[i].set_ylabel("Economy Rate", fontsize=10)
    axes[i].grid(True, linestyle="--", alpha=0.5)
    axes[i].set_xticks(team_df['season'].unique())
    axes[i].set_xticklabels(team_df['season'].unique(), rotation=45)


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

Conclusion

Final Conclusion on IPL 2024 Bowling Performance

This analysis highlights the **top-performing bowlers** across different match phases (Powerplay, Death Overs) and key rivalries between bowlers and batsmen. Additionally, it explores the impact of **venue conditions** on bowling effectiveness.

---

1 Powerplay Bowling Analysis (Overs 0-6) 1

Most Effective Bowlers

- **1st Inning:**
  - **Trent Boult (9 wickets)** – Lethal swing bowler, consistently dismissed openers.
  - **Mitchell Starc (7 wickets)** – Used early pace and movement to his advantage.
  - **Sandeep Sharma, N Thushara, and Arshdeep Singh (5 wickets each)** – Effective in picking wickets inside the powerplay.
- **2nd Inning:**
  - **Bhuvneshwar Kumar & Pat Cummins (6 wickets each)** – Both highly skilled at controlling new-ball movement.
  - **Ishant Sharma & Yash Dayal (5 wickets each)** – Made crucial breakthroughs in run chases.

Venue Impact on Powerplay

- **Wankhede & Chinnaswamy Stadiums**: High-scoring venues, bowlers with accurate line and length (Bhuvneshwar, Boult) succeeded.
- **Chepauk & Narendra Modi Stadiums**: Spinners like **Chahal & Bishnoi** controlled the powerplay with tight spells.
- **Eden Gardens**: Pacers benefitted from extra bounce, favoring aggressive bowlers like Starc & Harshit Rana.

Insight: Swing bowlers were highly effective in the first six overs, especially in venues with grass or moisture assisting movement.

---

2 Death Overs Specialists (Overs 16-20) 2

Best Wicket-Takers in the Final Overs

- **1st Inning:**
  - **Harshal Patel (17 wickets)** – Mastered slow deliveries and yorkers.
  - **Jasprit Bumrah (11 wickets)** – Consistent death-over specialist.
  - **Avesh Khan (10 wickets), Arshdeep Singh (9 wickets), TU Deshpande & Sandeep Sharma (8 wickets each)**.
- **2nd Inning:**
  - **Mukesh Kumar (8 wickets)** – Excelled under pressure while defending totals.
  - **T Natarajan, Mohammed Siraj, & Harshit Rana (7 wickets each)** – Skilled at executing yorkers.
  - **G Coetzee, Mitchell Starc, & Yash Thakur (6 wickets each)** – Used pace variations effectively.

Venue Impact on Death Overs

- **Chinnaswamy Stadium (Bangalore)**: Short boundaries resulted in high death-over economy; bowlers like Bumrah & T Natarajan relied on precise execution.
- **Chepauk Stadium (Chennai)**: Slower pitch favored spinners and cutters in the death overs.
- **Narendra Modi Stadium (Ahmedabad)**: Pace-friendly conditions helped Siraj and Natarajan with reverse swing.

Insight: Death-over specialists who mastered yorkers and slow deliveries proved to be the most effective across different conditions.

---

3 Repeat Dismissals (Bowler vs. Batter Rivalries) 3

Bowlers Who Dismissed the Same Batter Multiple Times (2020-2024)

- **Yuzvendra Chahal vs. Mayank Agarwal (5 dismissals)** – Chahal's variations consistently troubled Agarwal.
- **Deepak Chahar vs. Shubman Gill (5 dismissals)** – Used swing and seam to remove Gill multiple times.

- **Ravi Bishnoi vs. Ishan Kishan & Suryakumar Yadav (4 times each)** – Bishnoi's googlies troubled Mumbai Indians batters.
- **Mohammed Siraj vs. Prithvi Shaw (4 times)** – Used pace and movement effectively.
- **Kagiso Rabada vs. KL Rahul (4 times)** – Express pace forced mistakes from Rahul.
- **Rashid Khan vs. Shikhar Dhawan (4 times)** – Leg-spin and variations led to multiple dismissals.

 Insight: Certain bowlers have developed psychological advantages over key batsmen, using well-planned strategies to dismiss them repeatedly.

---

4 Virat Kohli vs. Bowlers in IPL 2024 4

Bowlers Against Whom Kohli Scored the Most Runs

- **Avesh Khan (47 runs conceded)** – Kohli dominated him with aggressive stroke play.
- **Sam Curran (42 runs conceded)** – Took advantage of Curran's medium pace.
- **Mitchell Starc (40 runs), Noor Ahmad (37 runs), Arshdeep Singh (35 runs)** – Kohli handled top-quality pacers and spinners effectively.

Venue Impact on Kohli's Performance

- **Chinnaswamy Stadium (Bangalore):** His best performances came here, using short boundaries to score freely.
- **Eden Gardens (Kolkata):** Struggled against extra bounce from pacers.
- **Chepauk Stadium (Chennai):** Used sweep shots effectively against spinners.

 Insight: Kohli adapted well to different conditions but was particularly dominant against medium pacers and left-arm spinners.

Final Takeaways & Strategic Recommendations

 Powerplay bowling success depended on swing-friendly conditions, with Boult and Starc leading the charts.
 Death-over specialists like Harshal Patel and Bumrah were game-changers, executing yorkers and slower deliveries to perfection.
 Spinners (Chahal, Bishnoi, Rashid Khan) played a key role in restricting runs & dismissing top-order batters.
 Some bowlers had a psychological edge over certain batsmen, repeatedly dismissing them in crucial moments.

 Virat Kohli performed best against medium pacers and left-arm spinners, adapting to various match situations.
 Venue conditions had a significant impact on bowling effectiveness, with different pitches favoring swing, spin, or pace.

 For IPL 2025, teams must prioritize a balanced bowling attack with early powerplay wicket-takers, death-over specialists, and spinners who can control the middle overs.

 Strong bowling strategies remain the key to IPL success!