BrainDead – Report

Indian Institute of Engineering Science and Technology (IIEST), Shibpur

Prepared and Submitted By Team - Back2Back
College – Indian Institute of Information Technology, Dharwad

Members:
Akash Nayak
Aditya Raj
Ayush Pareek

Repository Link - Github

Table of Contents

Problem Statement 1:

Statistics is All You Need: IPL Data Analysis and 2025 Winner Prediction – The Game Behind the Game!

Introduction

This report summarizes the analysis performed on the IPL 2024 dataset using a Jupyter notebook. The analysis includes data cleaning, standardization, and exploration of team performance metrics such as winning percentages, run rates, and economy rates. The dataset consists of two main files: `matches.csv` and `deliveries.csv`, which contain information about IPL matches and individual deliveries, respectively.

Data Cleaning and Standardization

1. Data Loading and Initial Cleaning

- The dataset was loaded using `pandas` and missing values were filled with "NA" to handle null entries.
- The following libraries were imported for analysis:

    1. Pandas
    2. Numpy
    3. Matplotlib
    4. Seaborn
    5. Sklearn

2. Standardization of Team Names

- Team names were standardized to ensure consistency across the dataset. For example, "Delhi Daredevils" was renamed to "Delhi Capitals", and "Deccan Chargers" was renamed to "Sunrisers Hyderabad".
- The standardization was applied to both `matches` and `deliveries` datasets.

3. Standardization of Seasons

- The season column was standardized to ensure uniformity. For example, "2007/08" was converted to "2008", and "2009/10" was converted to "2010".

4. Data type Conversion:

- The columns target_overs, target_runs, and result_margin in the “matches” dataset were converted to numeric values using pd.to_numeric() with errors='coerce' to handle any non-numeric entries.

Team Analysis

1. Winning Percentage

- The winning percentage of each team was calculated by dividing the number of matches won by the total number of matches played.

- A bar plot was generated to visualize the winning percentages of the teams in descending order.
- Results: The team with the highest winning percentage was "Gujarat Titans", followed by "Chennai Super Kings".

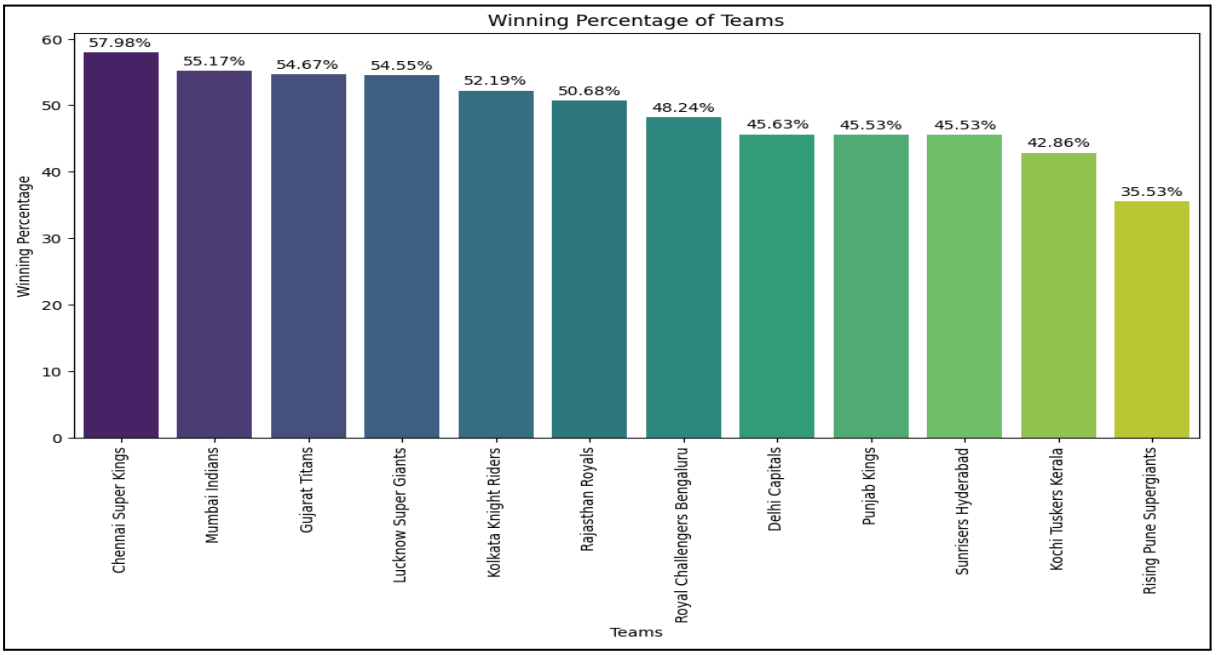


Fig. 1

2. Average Run Rate

- The average run rate for each team was calculated by dividing the total runs scored by the total overs faced.
- A bar plot was generated to visualize the average run rates of the teams in descending order.
- Results: The team with the highest average run rate was "Gujarat Titans", followed by "Lucknow Supergiants".

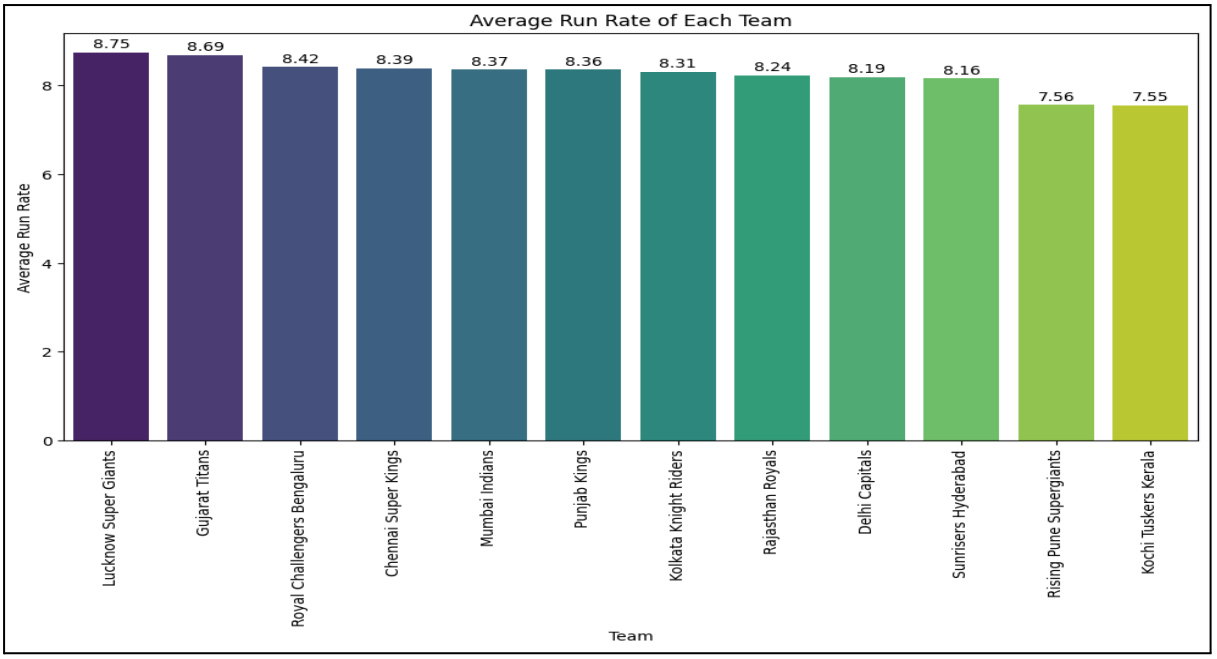


Fig. 2

3. Average Economy Rate

- The average economy rate for each team was calculated by dividing the total runs conceded by the total overs bowled.
- A bar plot was generated to visualize the average economy rates of the teams in descending order.
- Results: The team with the highest average economy rate was "Lucknow Supergiants", followed by "Gujarat Titans".

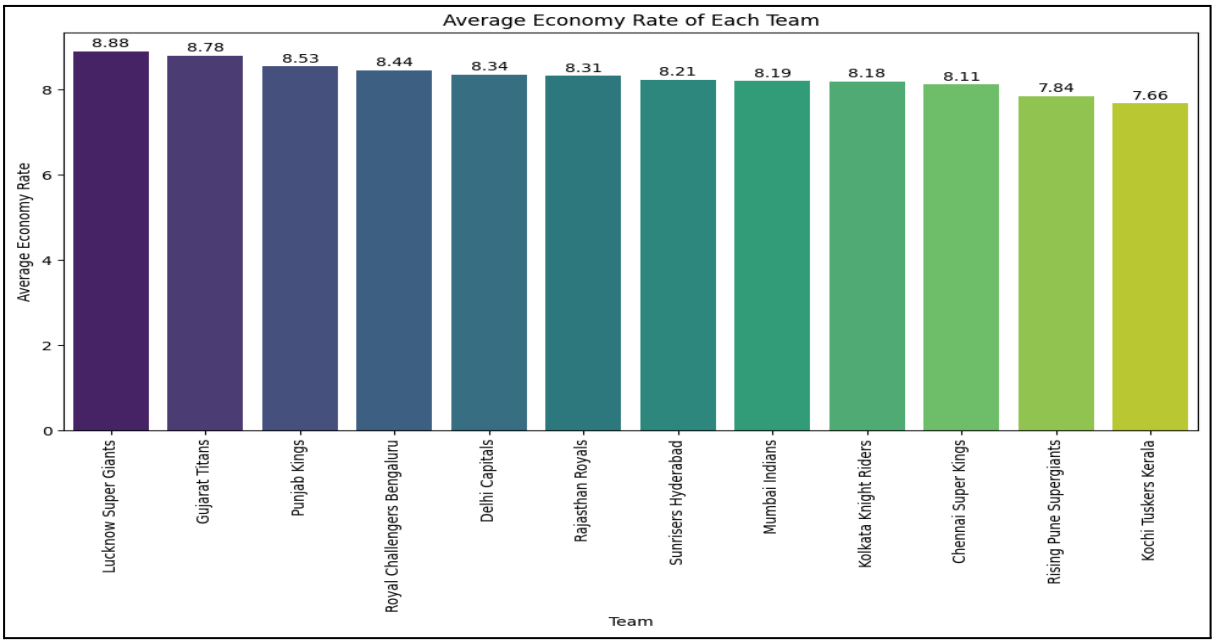


Fig. 3

4. Highest and Lowest Team Scores

- The highest and lowest scores for each team in completed 20-over matches were calculated.
- Bar plots were generated to visualize the highest and lowest scores by teams.

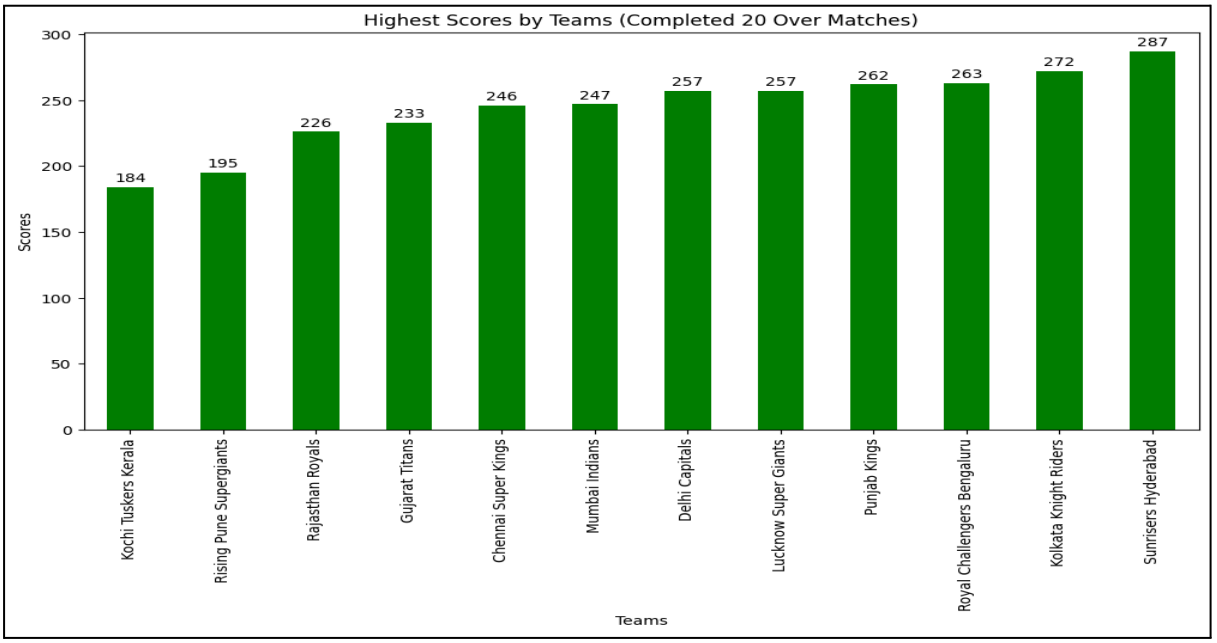


Fig. 4

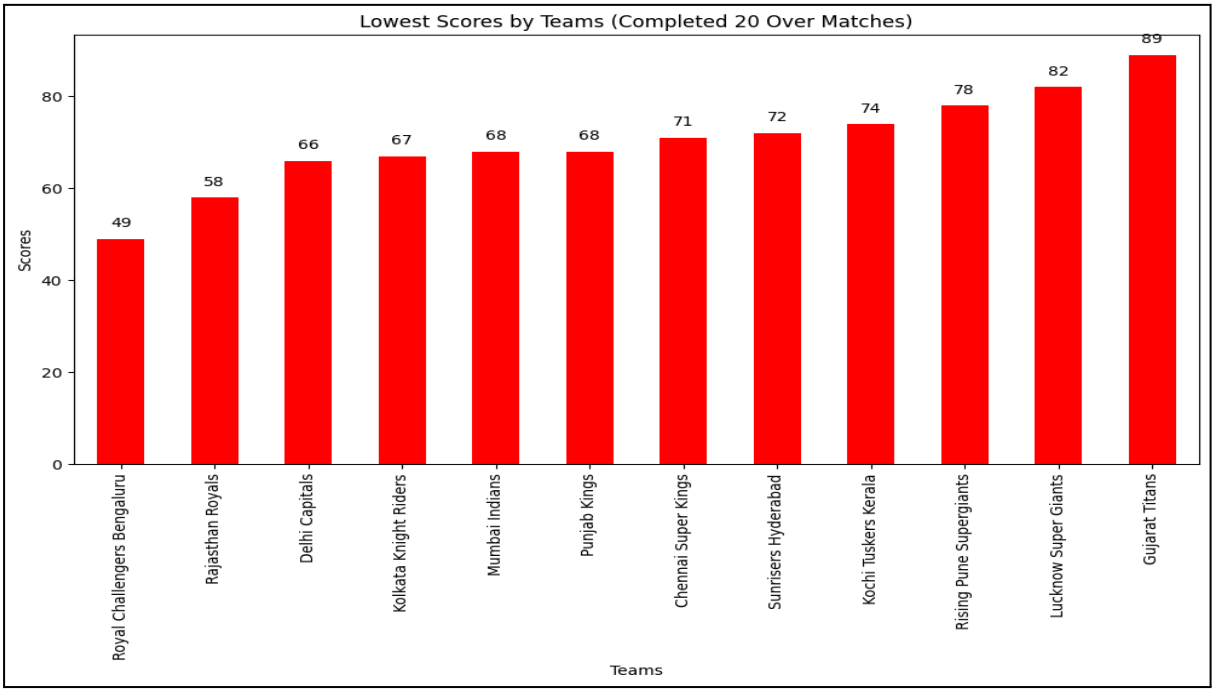


Fig. 5

5. Boundary Counts (4s and 6s)

- The total number of 4s and 6s hit by each team was calculated.
- A grouped bar plot was generated to visualize the boundary counts for each team.
- Results: Teams like Kolkata Knight Riders and Mumbai Indians dominated in hitting boundaries, with a high number of 4s and 6s.

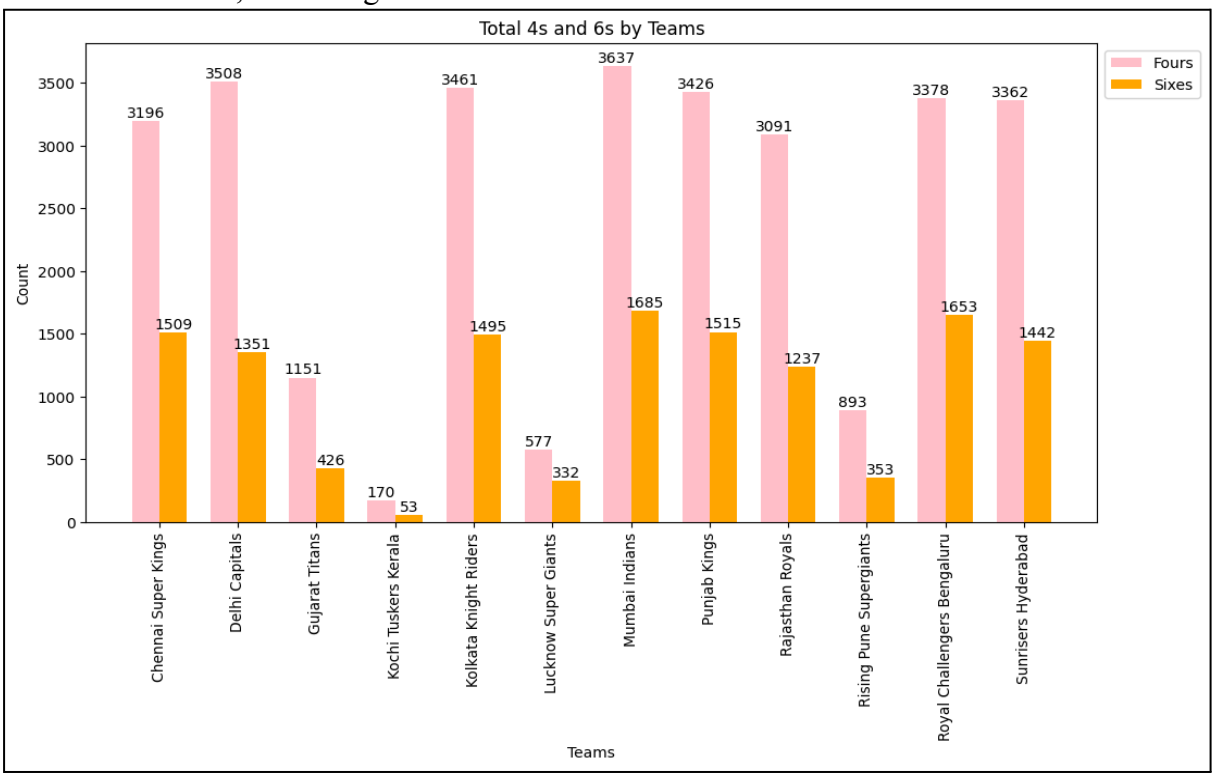


Fig. 6

6. Powerplay vs Death Overs Performance

- The total runs scored by each team in the powerplay and death overs were calculated.
- A grouped bar plot was generated to compare powerplay and death overs performance.
- Results: Teams like Mumbai Indians and Sunrisers Hyderabad excelled in both powerplay and death overs, showcasing their balanced batting strategies.

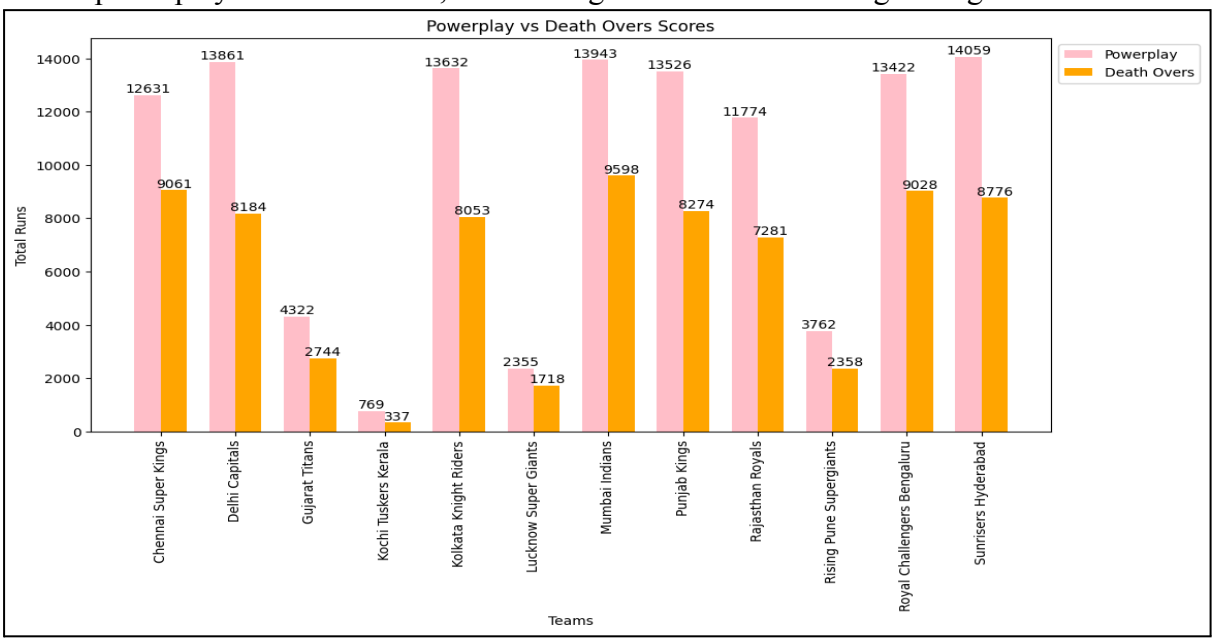


Fig. 7

7. Average Run Rate

- The average run rate (runs per over) for each team was calculated.
- A bar plot was generated to visualize the average run rates of the teams.
- Results: Teams like Gujarat Titans and Lucknow Supergiants had the highest average run rates, indicating aggressive batting.

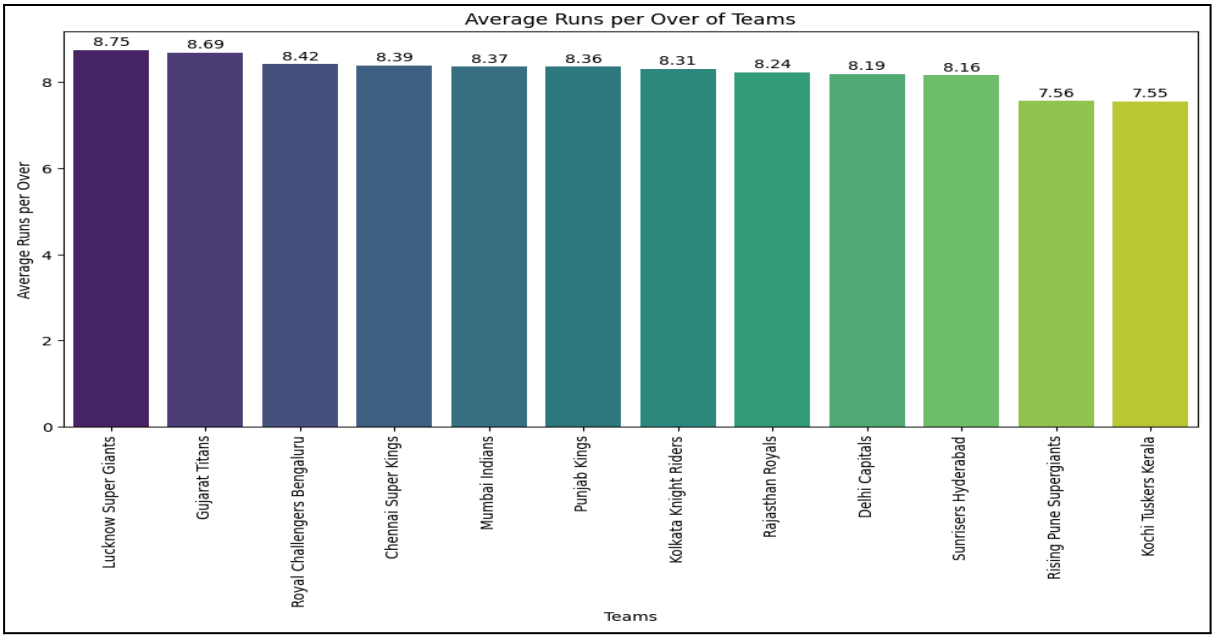


Fig. 8

8. Powerplay Performance

- The average runs scored by each team in the powerplay (overs 1-6) were calculated.
- A bar plot was generated to visualize the powerplay performance of the teams.
- Results: Teams like Gujarat Titans and Delhi Capitals consistently performed well in the powerplay, setting a strong foundation for their innings.

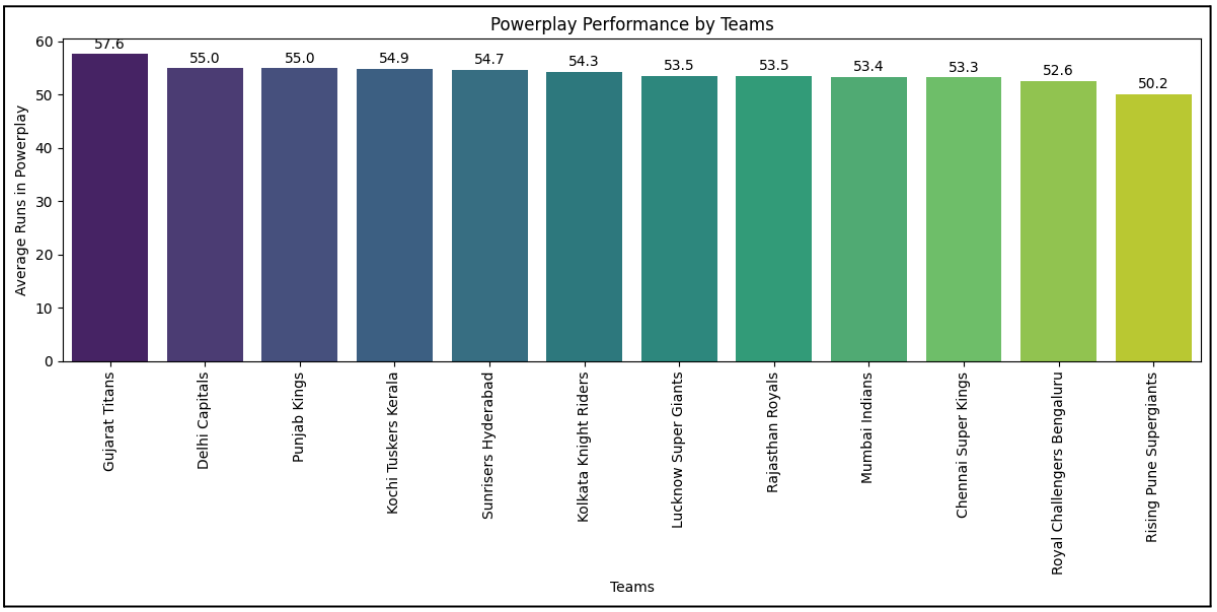


Fig. 9

Recommendations

1. Focus on Run Rate: Teams with lower run rates should focus on improving their batting performance to increase their chances of winning.
2. Improve Bowling Economy: Teams with higher economy rates should work on their bowling strategies to reduce the number of runs conceded.
3. Consistency: Teams like Mumbai Indians and Gujarat Titans should continue their consistent performance, while other teams should analyze their strategies to improve their standings.

This analysis can be extended further by incorporating additional metrics such as player performance, home vs. away performance, and more detailed match statistics.

Player Analysis

1. Top Run Scorers

- The top 20 run scorers in the tournament were identified by aggregating the total runs scored by each batsman.
- A bar plot was generated to visualize the top 20 run scorers.
- Results: The top run scorers included prominent players like Virat Kohli, Shikhar Dhawan, and Rohit Sharma.

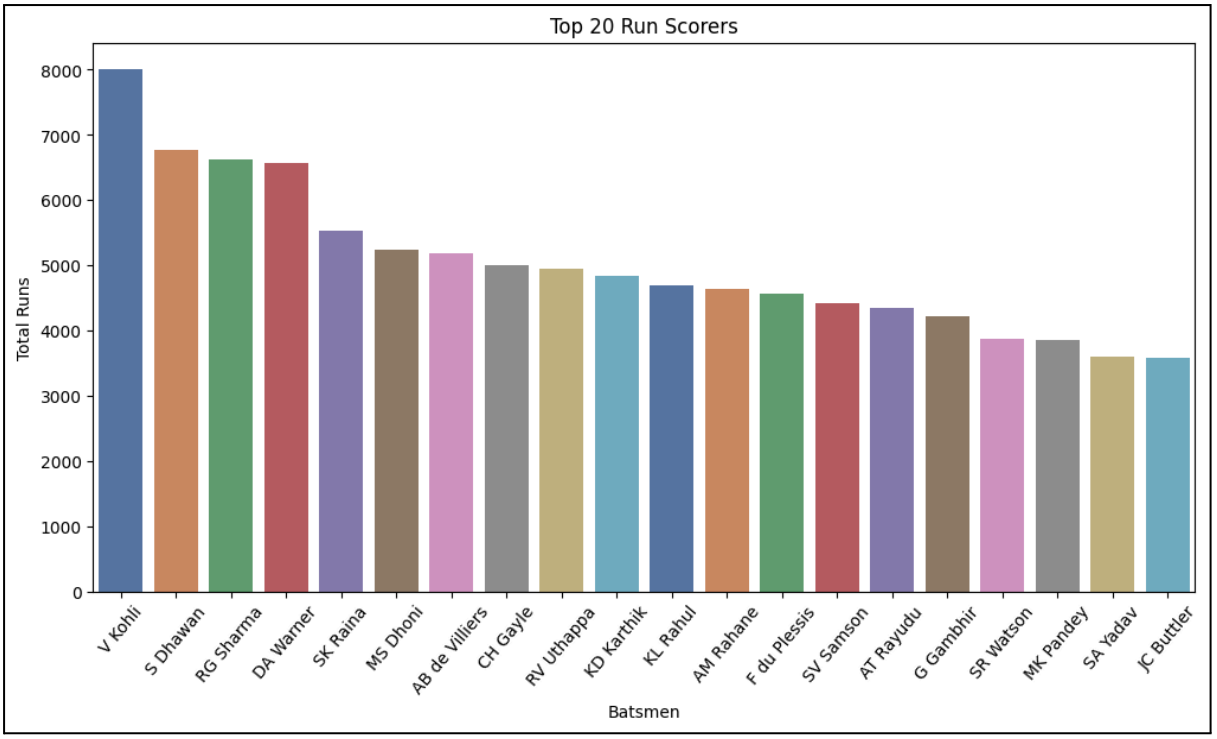


Fig. 10

2. Batting Average vs Strike Rate

- Batting statistics were calculated for the top 20 run scorers, including total runs, balls faced, matches played, strike rate, and batting average.
- A scatter plot was created to visualize the relationship between strike rate and batting average for these players.
- Results: Players with high strike rates and high batting averages were identified as the most impactful batsmen in the tournament.

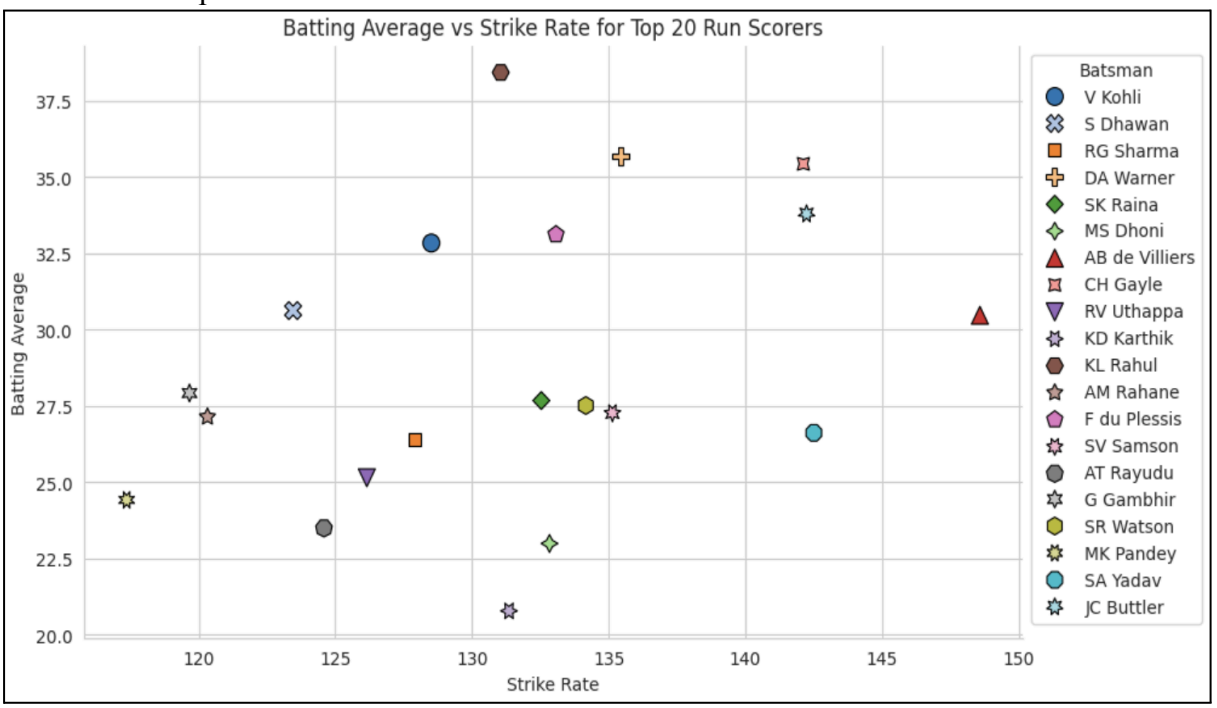


Fig. 11

3. Top Batsmen by Batting Average (Min. 50 Matches)

- Batsmen with a minimum of 50 matches were filtered, and their batting averages were calculated.
- A bar plot was generated to display the top 10 batsmen by batting average.
- Results: Players like KL Rahul and David Warner had the highest batting averages, indicating consistent performance.

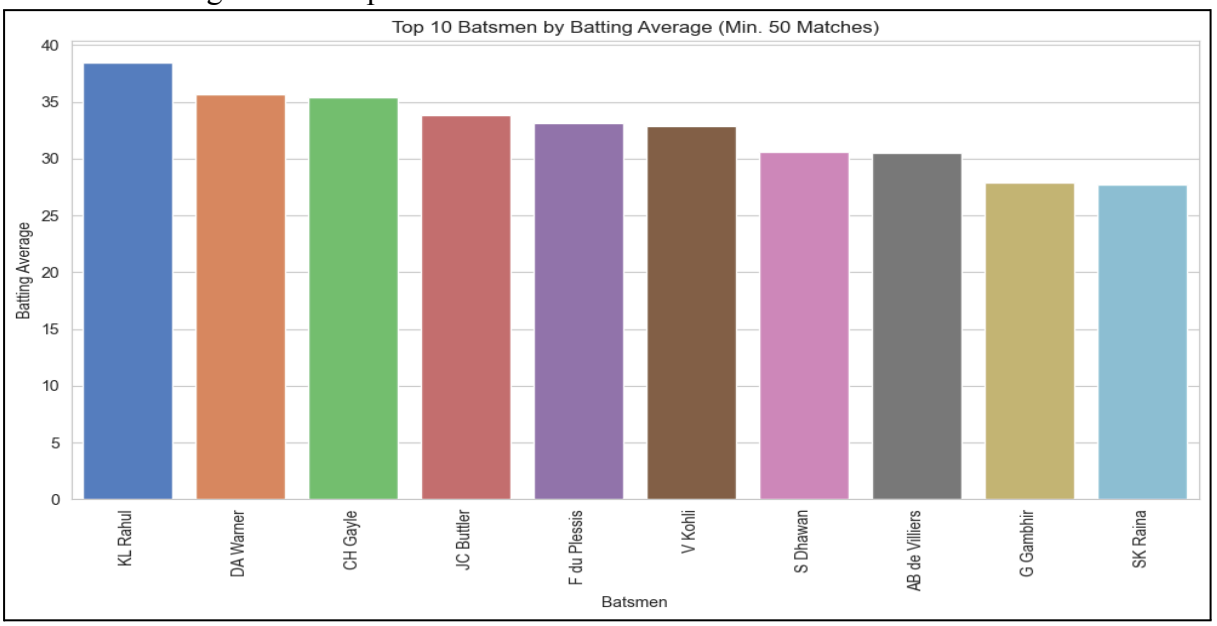


Fig. 12

4. Top Batsmen by Strike Rate (Min. 50 Matches)

- The top 10 batsmen with the highest strike rates (minimum 50 matches) were identified.
- A bar plot was generated to visualize these players.
- Results: Explosive batsmen like AB de Villiers and Suryakumar Yadav dominated this category.

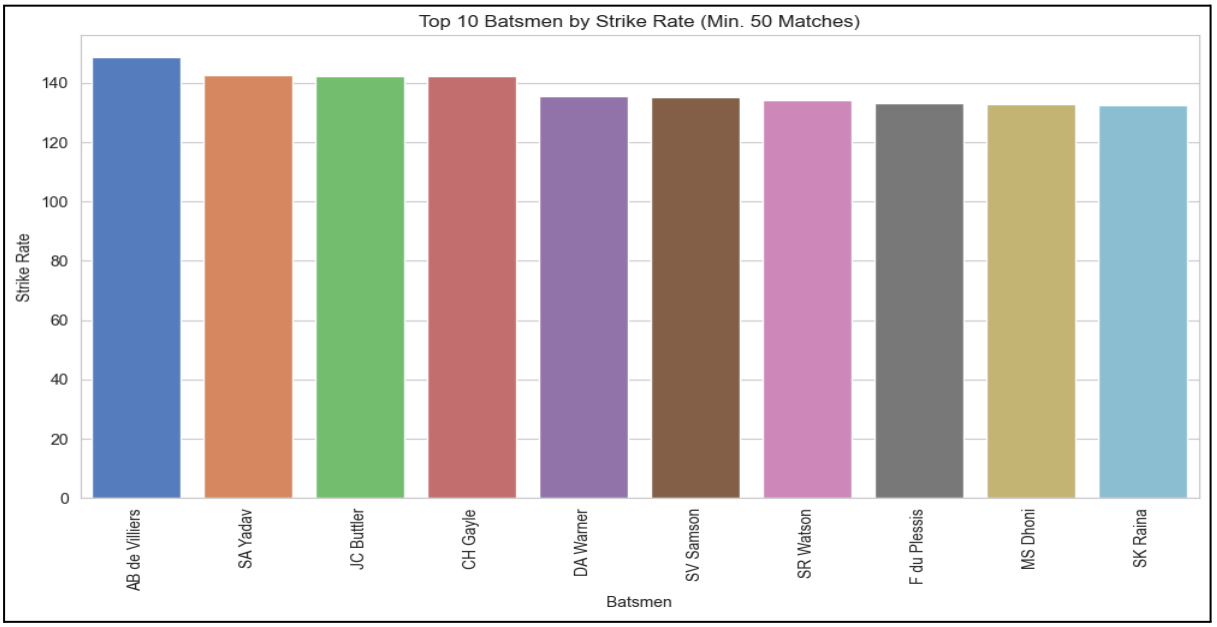


Fig. 13

5. Top Wicket-Takers

- The top 10 wicket-takers were identified by aggregating the total wickets taken by each bowler.
- A bar plot was generated to visualize the top wicket-takers.
- Results: Bowlers like Chahal and Bravo were among the top performers.

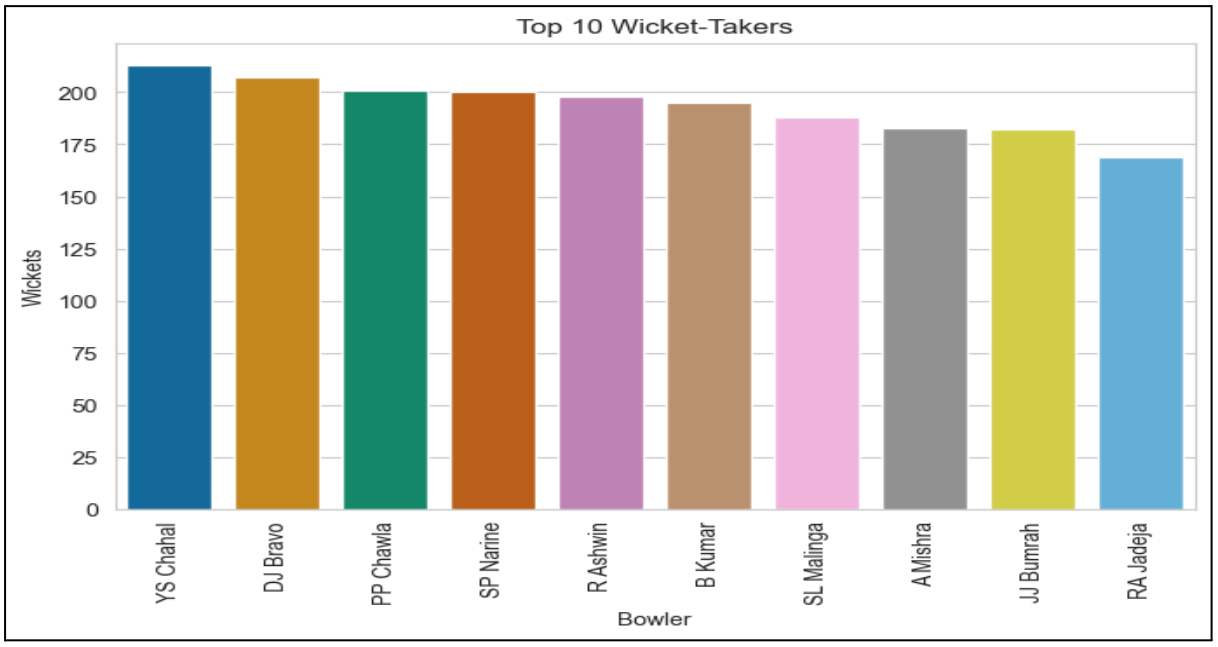


Fig. 14

6. Top 10 Highest Individual Scores

- The highest individual scores by batsmen in a single match were identified.
- A bar plot was generated to display the top 10 highest individual scores.
- Results: Players like Chris Gayle and McCullum featured prominently with high individual scores.

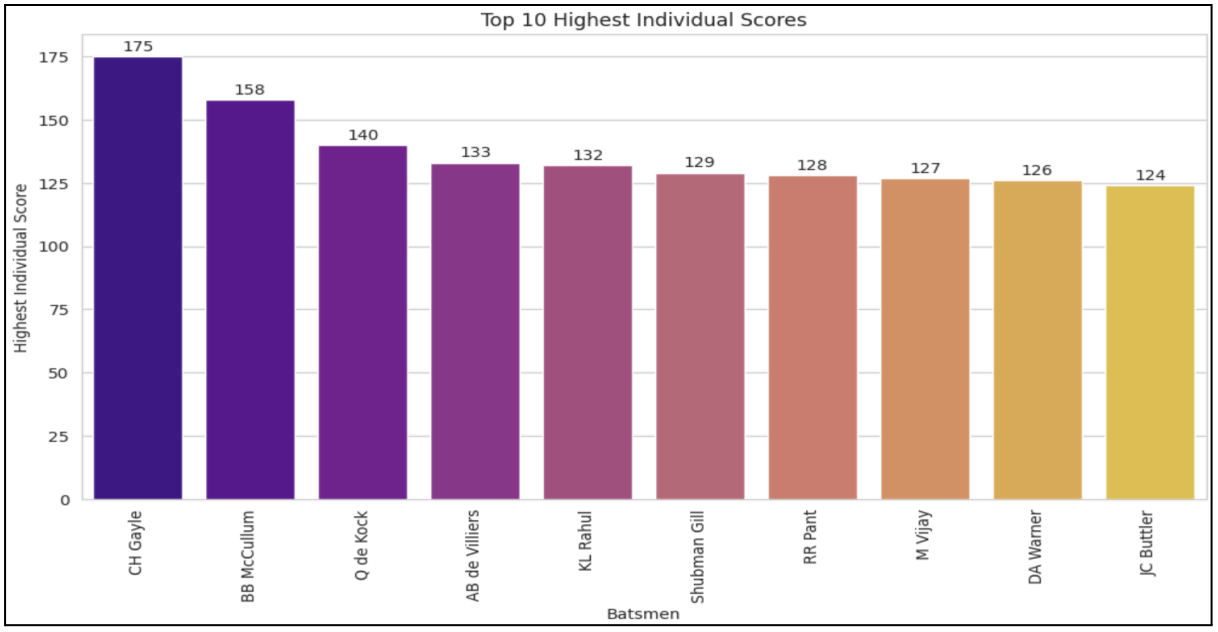


Fig. 15

7. Top 10 Man of the Match Winners

- The top 10 players with the most "Man of the Match" awards were identified.
- A bar plot was generated to visualize these players.
- Results: Consistent performers like AB de Villiers and Chris Gayle were among the top Man of the Match winners.

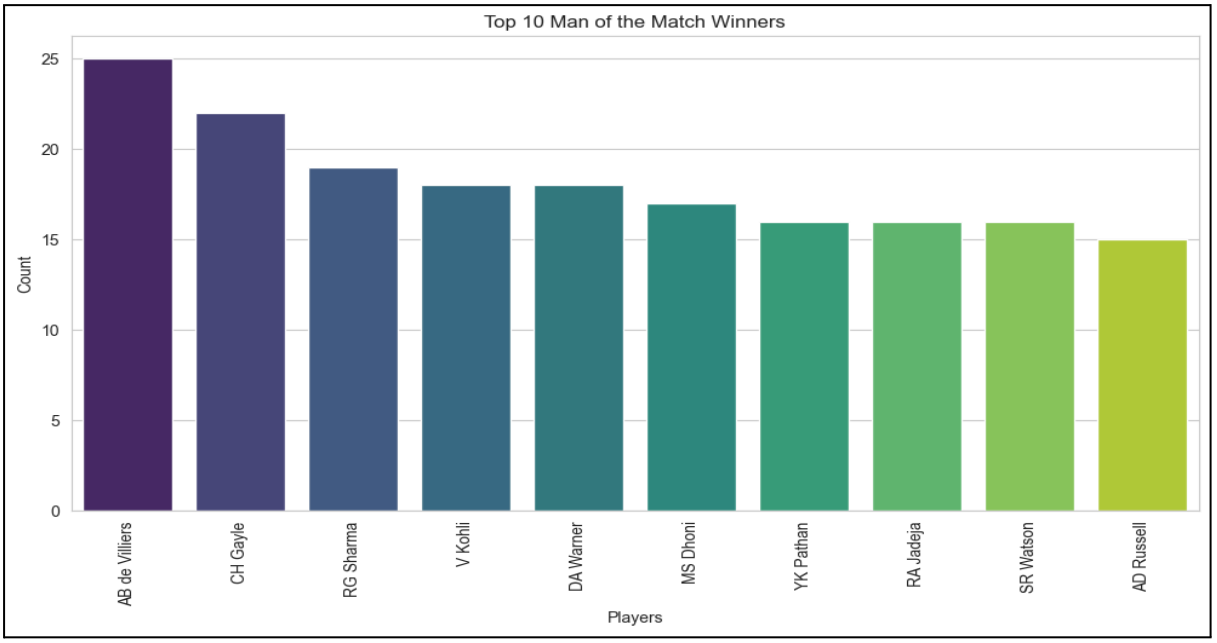


Fig. 16

8. Player Clustering (K-Means)

- Players were clustered into three categories (Batters, Bowlers, All-Rounders) based on their batting averages and bowling economy rates using K-Means clustering.
- A scatter plot was generated to visualize the clusters.
- Results: The clustering highlighted players who excelled in both batting and bowling, such as Hardik Pandya and Ravindra Jadeja, as all-rounders.

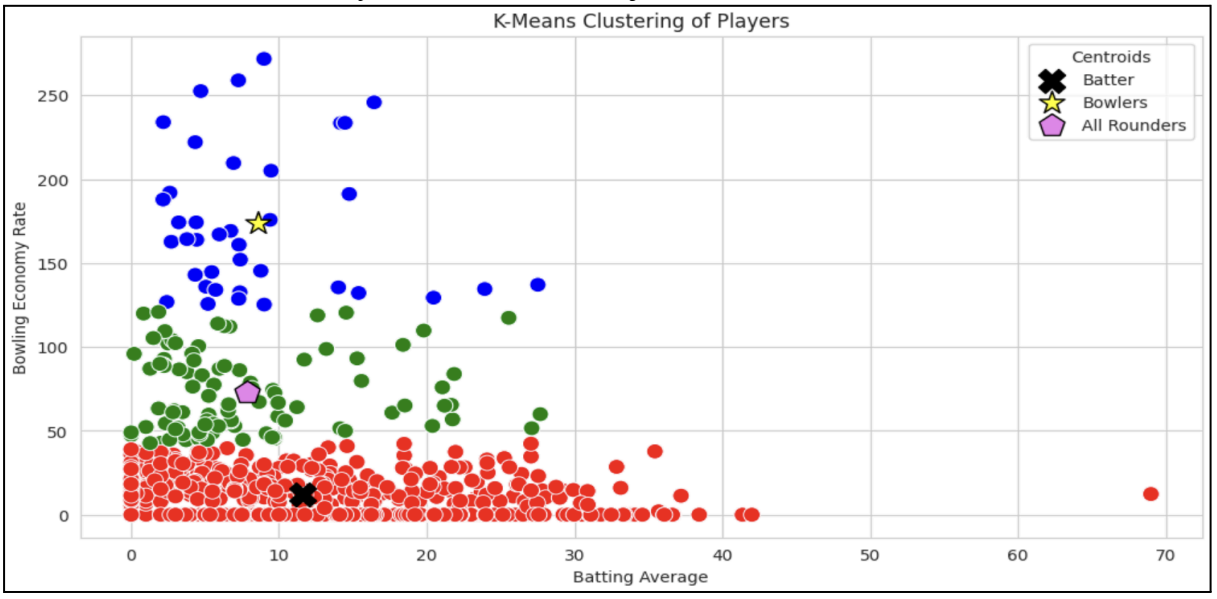


Fig. 17

9. Top Batsmen by Run Types (1s, 2s, 4s, 6s)

- The top 10 batsmen were identified for each run type (1s, 2s, 4s, and 6s).
- Bar plots were generated for each run type.

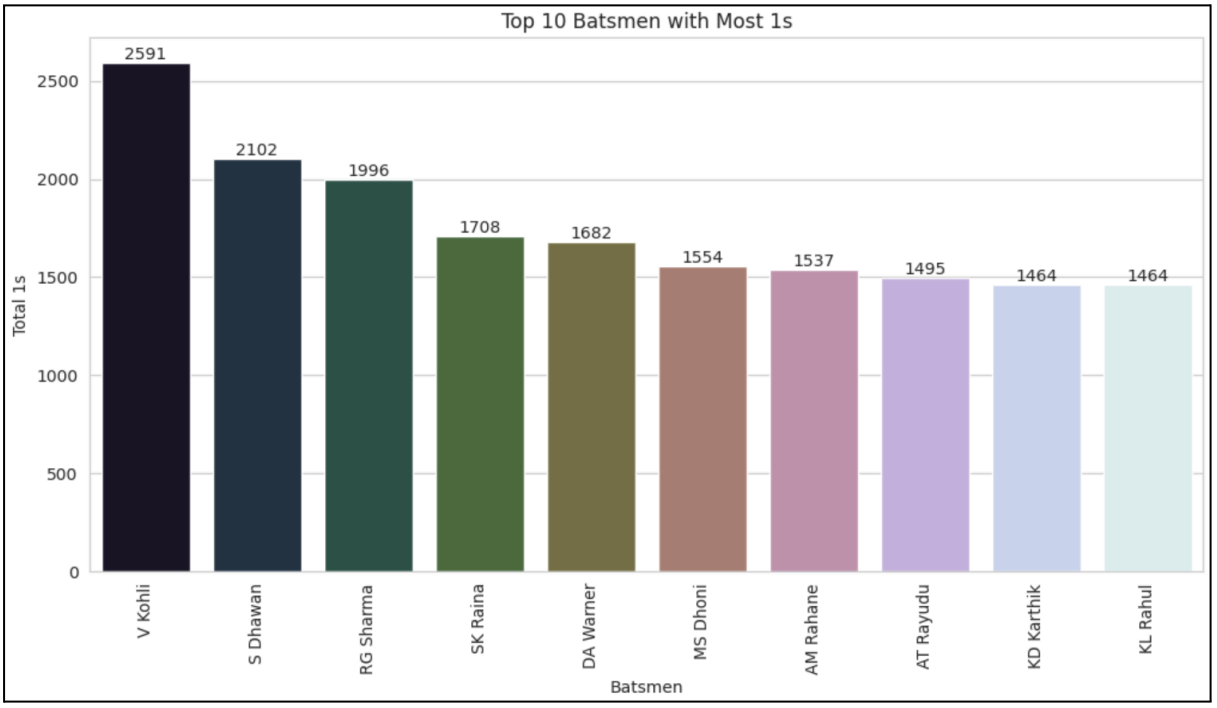


Fig. 18

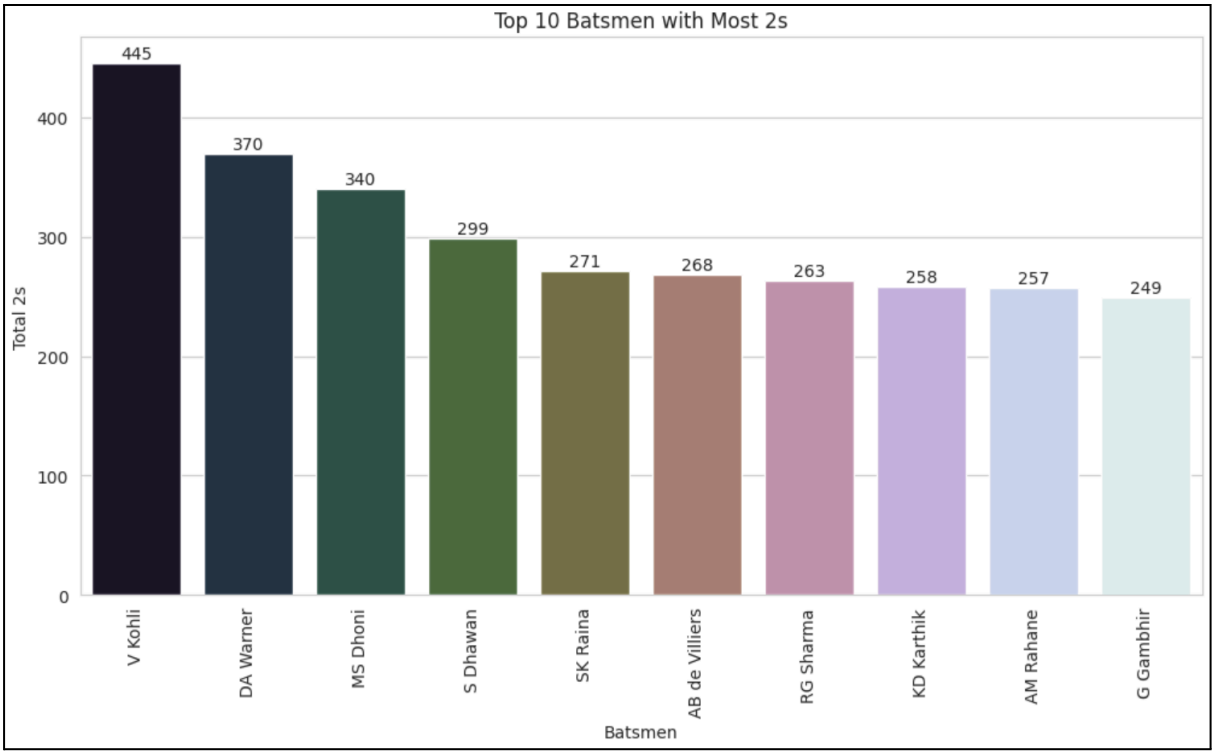


Fig. 19

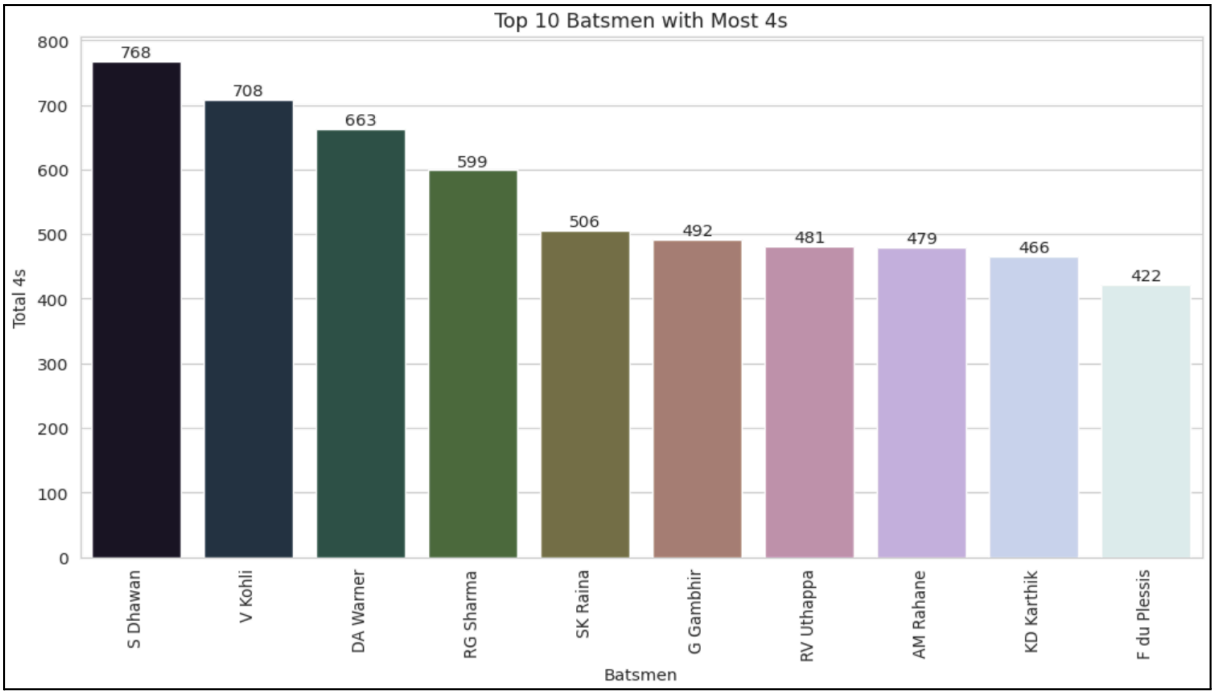


Fig. 20

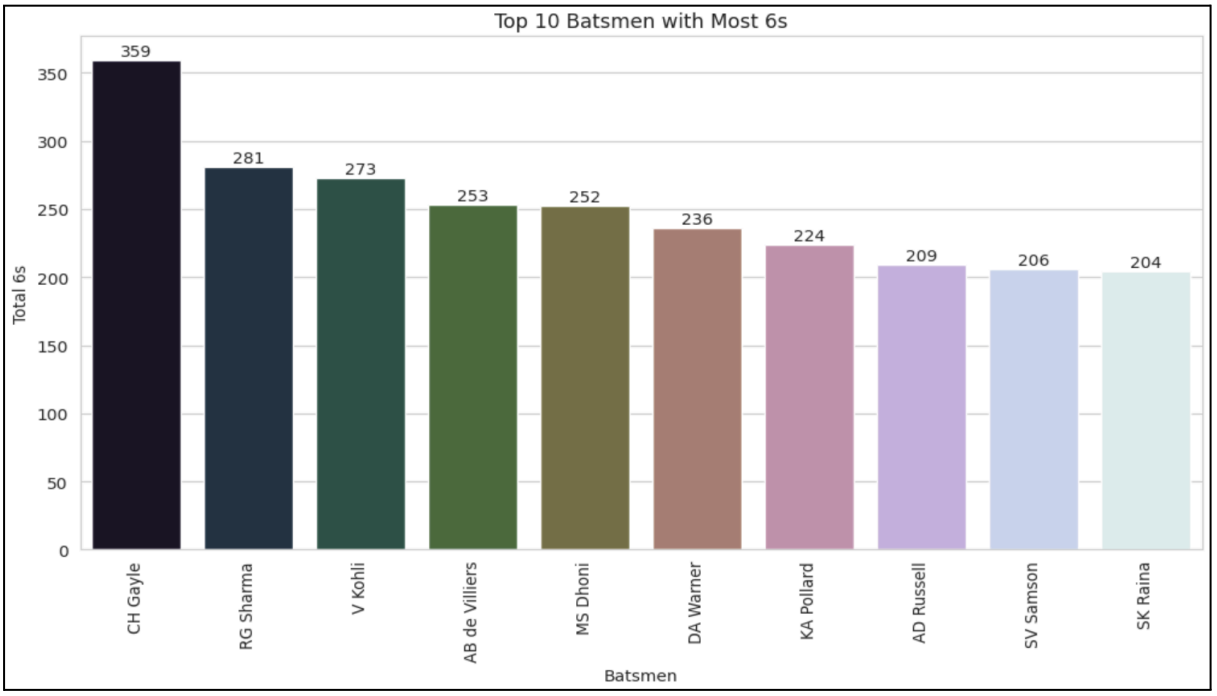


Fig. 21

Recommendations

1. Focus on All-Rounders: Teams should prioritize acquiring and developing all-rounders, as they provide balance and flexibility in both batting and bowling.

2. Improve Strike Rates: Batsmen with lower strike rates should work on increasing their scoring rates to contribute more effectively in high-pressure situations.

3. Enhance Bowling Economy: Bowlers with higher economy rates should focus on improving their accuracy and control to reduce the number of runs conceded.

4. Leverage Top Performers: Teams should build their strategies around consistent performers like Virat Kohli and AB de Villiers, who have proven their ability to deliver in crucial moments.

5. Analyze Run Types: Teams should analyze the distribution of runs (1s, 2s, 4s, 6s) to identify players who can accelerate the scoring rate during different phases of the game.

This analysis can be extended further by incorporating additional metrics such as fielding performance, player fitness, and match-specific conditions to gain a more comprehensive understanding of player contributions.

Season Analysis

1. Average Runs per Match per Season

- The average runs scored per match were calculated for each season.
- A bar plot was generated to visualize the average runs per match across seasons.
- Results: The average runs per match showed an increasing trend over the years, with the highest average observed in the most recent seasons.

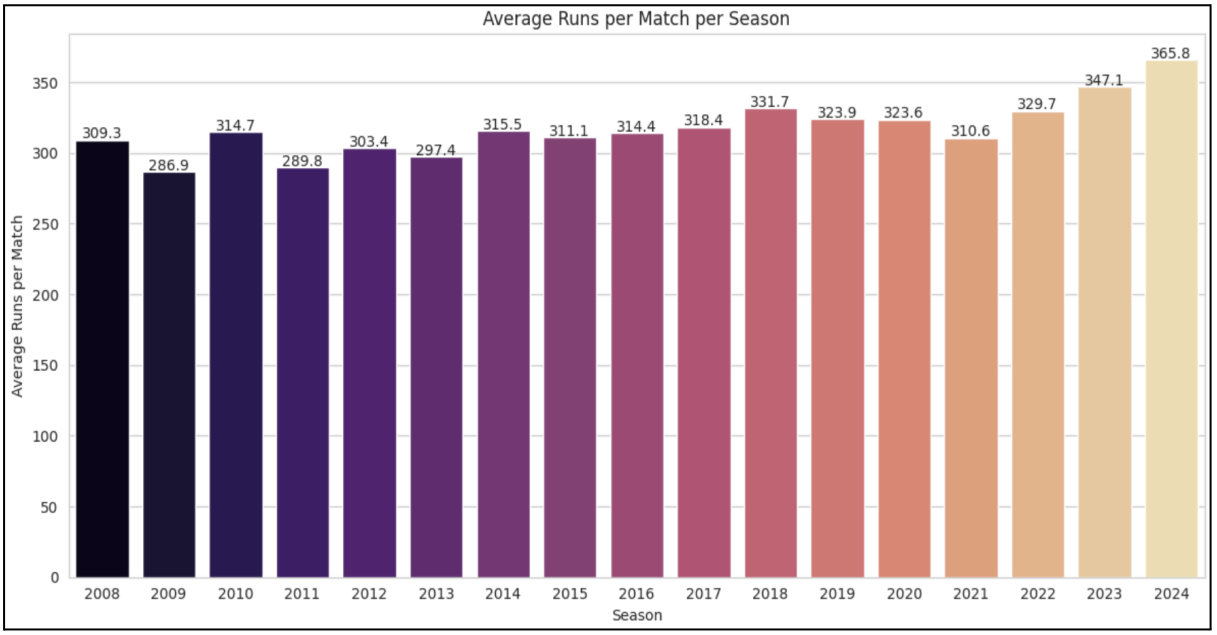


Fig. 22

2. Number of 200+ Targets per Season

- The number of matches with a target of 200 or more runs was calculated for each season.
- A bar plot was generated to visualize the number of 200+ targets per season.

- Results: The number of high-scoring matches (200+ targets) increased significantly in recent seasons, indicating a trend towards higher-scoring games.

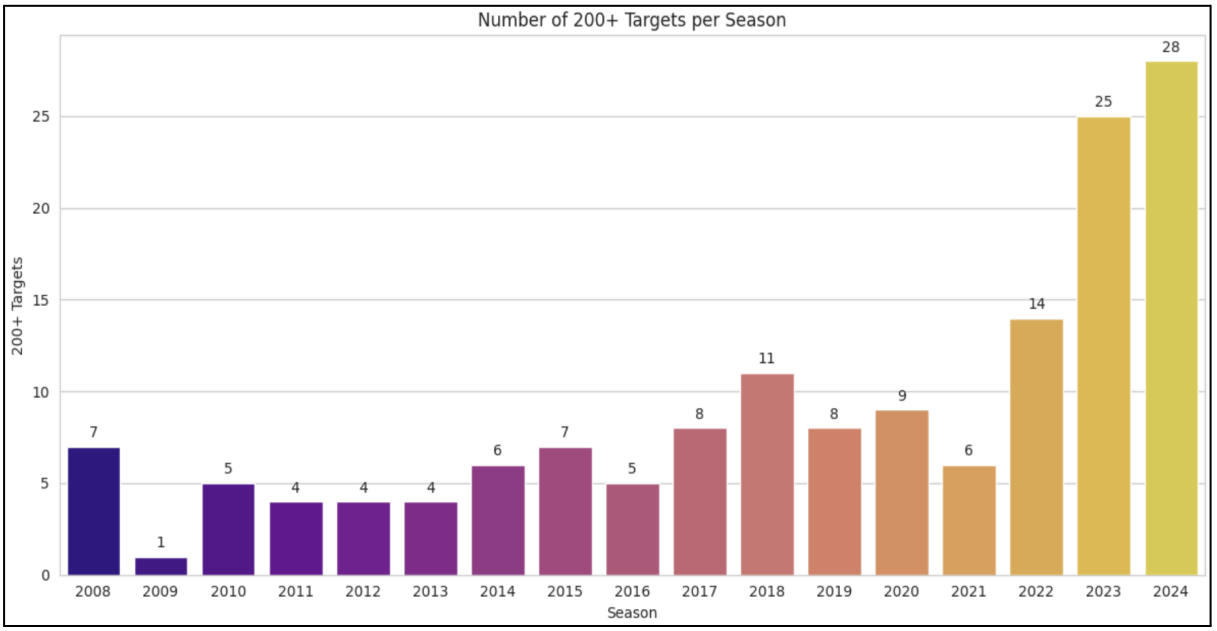


Fig. 23

3. Average Score of Each Team per Season

- The average score per team per season was calculated.
- A scatter plot was generated to visualize the average scores of each team across seasons.
- Results: Teams like Mumbai Indians and Chennai Super Kings consistently maintained high average scores, while newer teams showed variability in their performance.

Team	Season	Avg_Score
Chennai Super Kings	2008	157.5
Chennai Super Kings	2009	159.36
Chennai Super Kings	2010	162.75
Chennai Super Kings	2011	160.0
Chennai Super Kings	2012	157.28
Chennai Super Kings	2013	154.33
Chennai Super Kings	2014	165.62
Chennai Super Kings	2015	160.29
Chennai Super Kings	2018	175.56
Chennai Super Kings	2019	145.47
Chennai Super Kings	2020	156.5
Chennai Super Kings	2021	170.81
Chennai Super Kings	2022	163.43
Chennai Super Kings	2023	180.8
Chennai Super Kings	2024	180.29
Delhi Capitals	2008	151.29
Delhi Capitals	2009	142.07
Delhi Capitals	2010	153.93
Delhi Capitals	2011	149.07
Delhi Capitals	2012	146.94

Fig. 24

Link to the complete table - LINK

4. Orange Cap Holders (Top Run Scorers per Season)

- The top run scorer (Orange Cap holder) for each season was identified.
- A table was generated to display the Orange Cap holders and their total runs.
- Results: Prominent batsmen like Virat Kohli and David Warner were among the top run scorers in multiple seasons.

Season	Batter	Total_Runs
2008	SE Marsh	616
2009	ML Hayden	572
2010	SR Tendulkar	618
2011	CH Gayle	608
2012	CH Gayle	733
2013	MEK Hussey	733
2014	RV Uthappa	660
2015	DA Warner	562
2016	V Kohli	973
2017	DA Warner	641
2018	KS Williamson	735
2019	DA Warner	692
2020	KL Rahul	676
2021	RD Gaikwad	635
2022	JC Buttler	863
2023	Shubman Gill	890
2024	V Kohli	741

Fig. 25

5. Purple Cap Holders (Top Wicket-Takers per Season)

- The top wicket-taker (Purple Cap holder) for each season was identified.
- A table was generated to display the Purple Cap holders and their total wickets.
- Results: Bowlers like Harshal Patel and Rabada were among the top wicket-takers in multiple seasons.

Season	Bowler	Total_Wickets
2008	Sohail Tanvir	24
2009	RP Singh	26
2010	PP Ojha	22
2011	SL Malinga	30
2012	M Morkel	30
2013	DJ Bravo	34
2014	MM Sharma	26
2015	DJ Bravo	28
2016	B Kumar	24
2017	B Kumar	28
2018	AJ Tye	28
2019	K Rabada	29
2020	K Rabada	32
2021	HV Patel	35
2022	YS Chahal	29
2023	MM Sharma	31
2024	HV Patel	30

Fig. 26

6. Top 10 Bowlers per Season

- The top 10 bowlers with the most wickets in each season were identified.
- A table was generated to display the top 10 bowlers per season.
- Results: Consistent performers like Jasprit Bumrah and Rashid Khan featured prominently in the top 10 bowlers across multiple seasons.

season	bowler	wickets
2008	Sohail Tanvir	24
2008	IK Pathan	20
2008	JA Morkel	20
2008	SK Warne	20
2008	SR Watson	20
2008	MF Maharoof	19
2008	PP Chawla	19
2008	S Sreesanth	19
2008	MS Gony	18
2008	RP Singh	17
2009	RP Singh	26
2009	A Kumble	22
2009	A Nehra	22
2009	PP Ojha	21
2009	SL Malinga	21
2009	DP Nannes	20
2009	IK Pathan	20
2009	MM Patel	18

Fig. 27

Link to the complete table - LINK

Recommendations

1. Focus on Aggressive Batting: Teams should prioritize aggressive batting strategies to capitalize on the trend of high-scoring matches.

2. Improve Bowling Strategies: Bowlers should focus on developing variations and improving accuracy to counter aggressive batting and reduce the number of high-scoring matches.

3. Leverage Consistent Performers: Teams should build their strategies around consistent performers like Virat Kohli, David Warner, Jasprit Bumrah, and Rashid Khan, who have proven their ability to deliver in crucial moments.

4. Analyze Team Performance Trends: Teams should analyze their performance trends across seasons to identify strengths and weaknesses and make data-driven decisions for future matches.

5. Develop Young Talent: Teams should invest in developing young talent to ensure long-term success and maintain a competitive edge in the tournament.

This analysis can be extended further by incorporating additional metrics such as player fitness, match-specific conditions, and fielding performance to gain a more comprehensive understanding of the factors influencing team and player performance.

Feature Extraction

Deliveries Dataset

The following features were extracted:

1. Wickets per Over

A bar plot was created to show the number of wickets taken in each over. This visualization helps identify which overs are most challenging for batsmen.

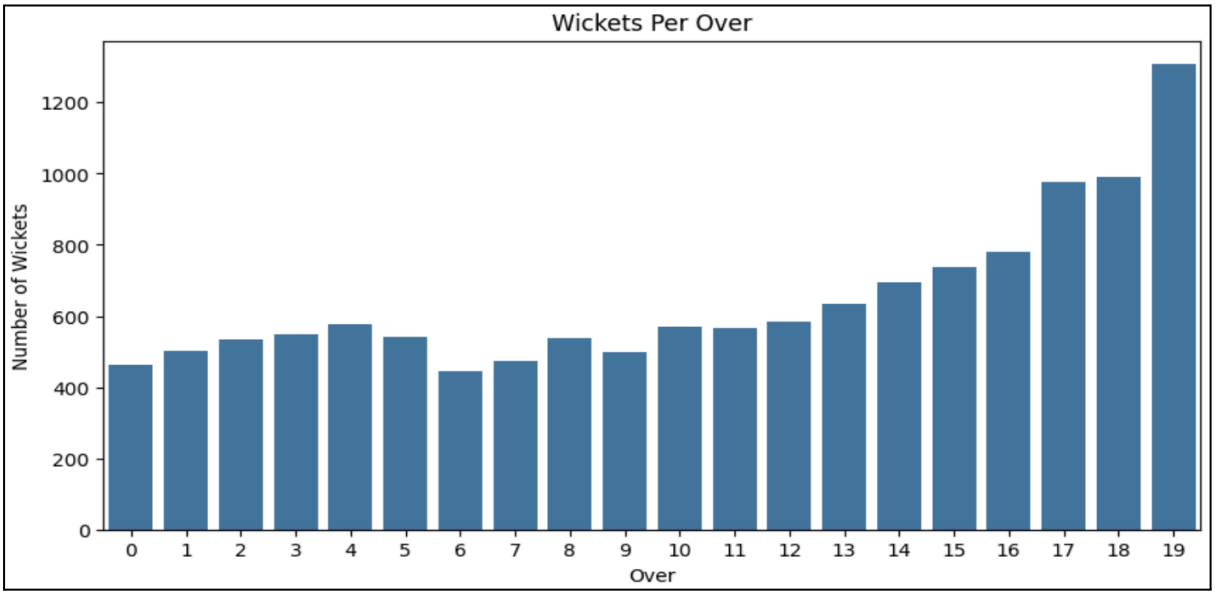


Fig. 28

2. Average Powerplay Runs

The average runs scored by each team during the powerplay were sorted and displayed, providing insights into which teams perform best in the initial overs.

batting_team	total_runs
Gujarat Titans	57.626667
Delhi Capitals	55.003968
Punjab Kings	54.983740
Kochi Tuskers Kerala	54.928571
Sunrisers Hyderabad	54.704280
Kolkata Knight Riders	54.310757
Lucknow Super Giants	53.522727
Rajasthan Royals	53.518182
Mumbai Indians	53.421456
Chennai Super Kings	53.295359
Royal Challengers Bengaluru	52.635294
Rising Pune Supergiants	50.160000

Fig. 29

3. Boundaries (4s and 6s) per Team

A bar plot was created to show the distribution of 4s and 6s hit by each team.

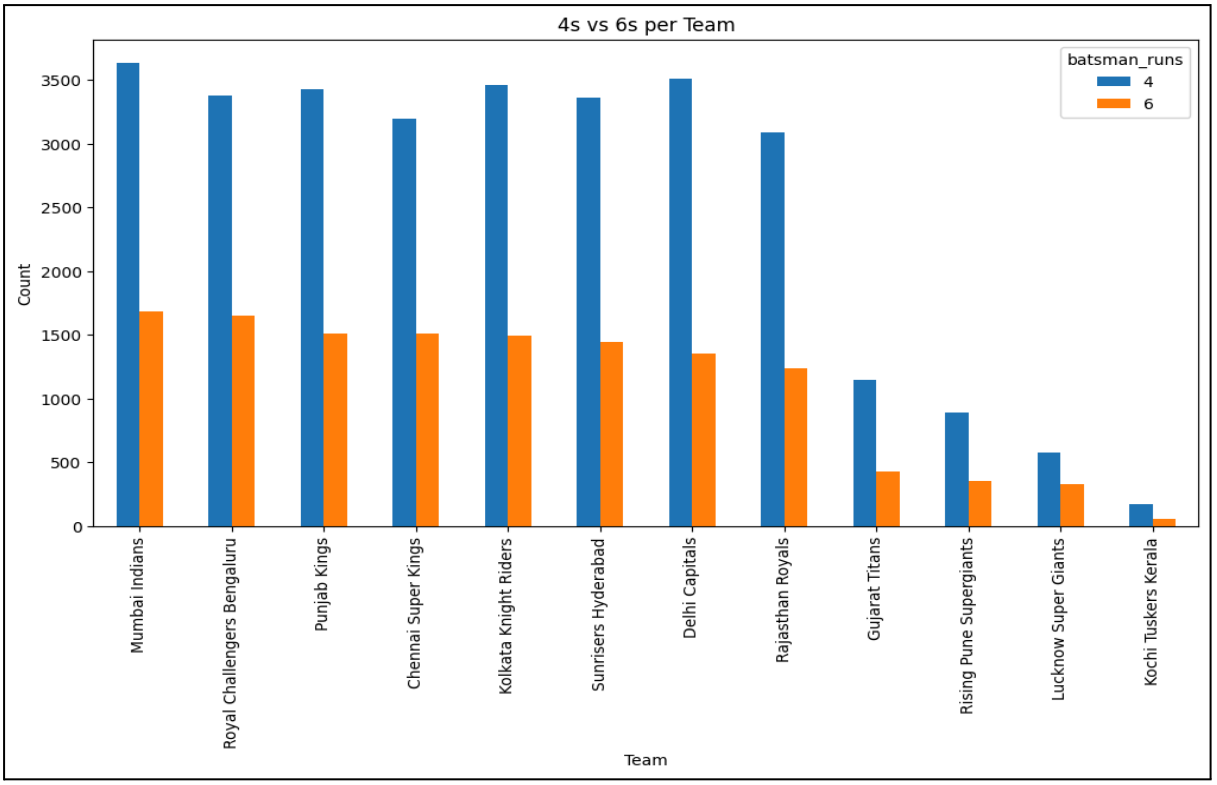


Fig. 30

4. Wickets Lost per 5-Over Interval

A bar plot was created to show the number of wickets lost in each 5-over interval.

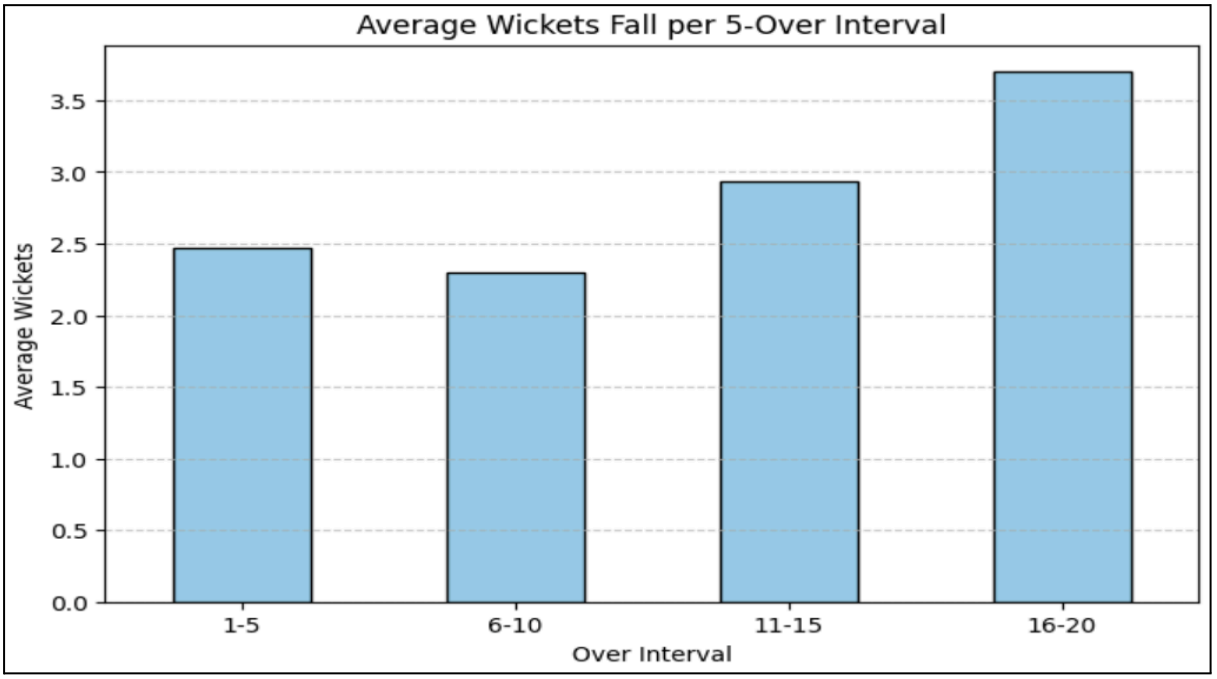


Fig. 31

5. Top Bowlers by Economy Rate

A bar plot was created to show the economy rates of the top 10 bowlers with the most overs bowled.

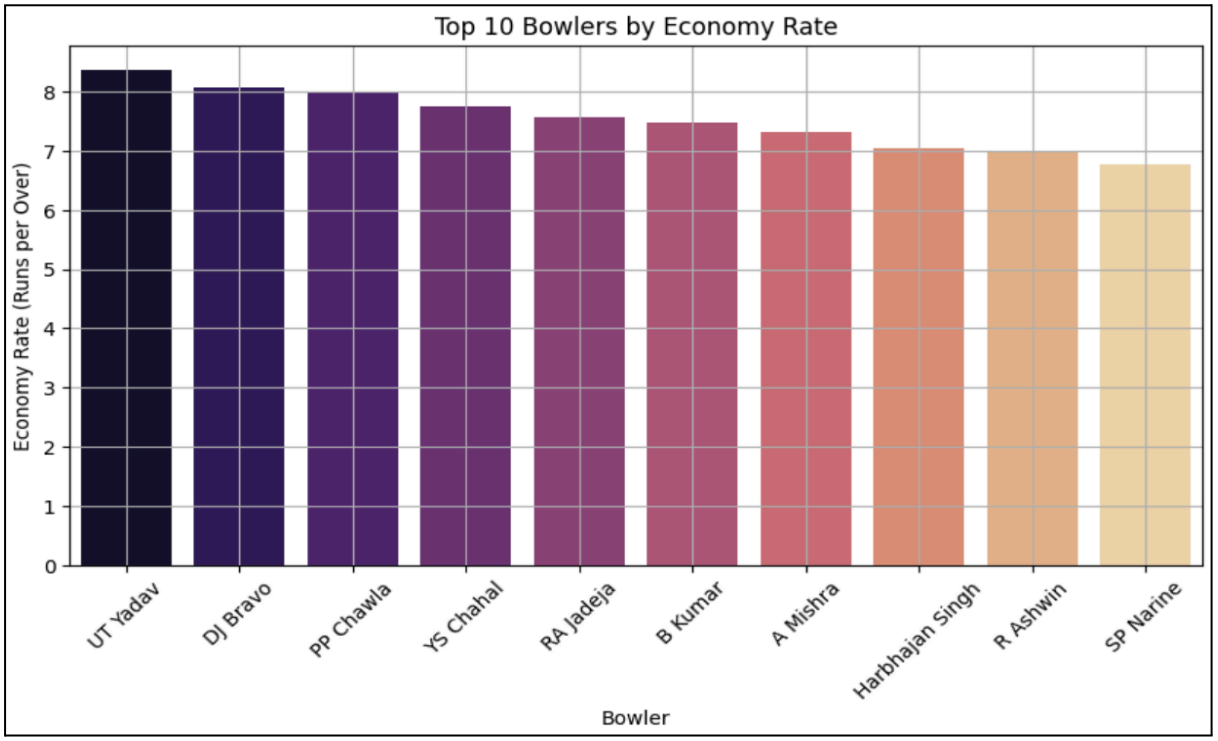


Fig. 32

6. Partnerships

Building a successful opening partnership is very important to have control of the game. The top partnerships were found and displayed.

batter	non_striker	total_runs
AB de Villiers	V Kohli	1623
CH Gayle	V Kohli	1617
V Kohli	AB de Villiers	1511
DA Warner	S Dhawan	1420
V Kohli	CH Gayle	1185
RV Uthappa	G Gambhir	1077
V Kohli	F du Plessis	1050
Ishan Kishan	RG Sharma	992
F du Plessis	V Kohli	982
S Dhawan	DA Warner	937

Fig. 33

Matches Dataset

The following features were extracted:

1. Distribution of Victory Margins (Runs)

The histogram shows the distribution of victory margins in terms of runs.

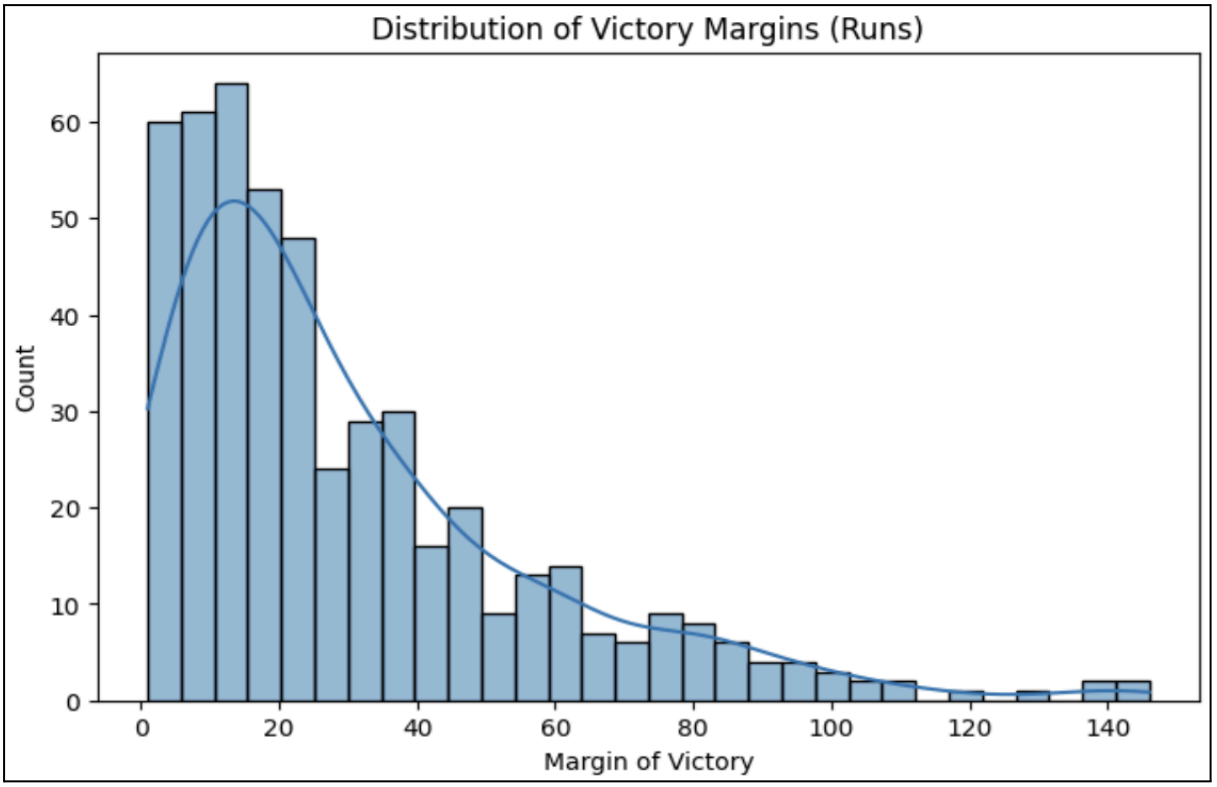


Fig. 34

2. Impact of Toss Decisions on Match Outcomes

Teams that win the toss have a slight advantage in winning the match.

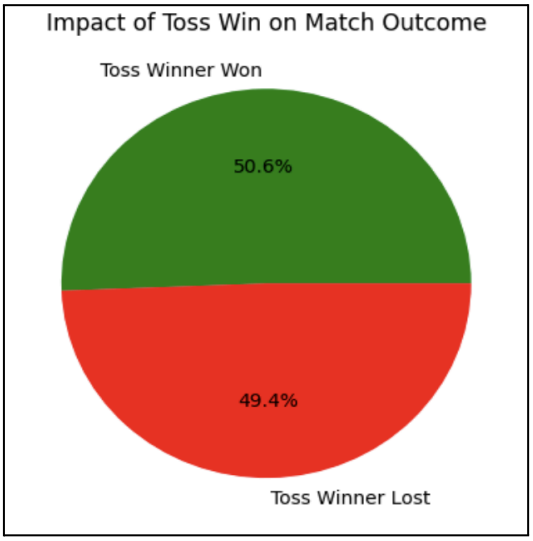


Fig. 35

3. Team Performance: Chasing vs. Defending

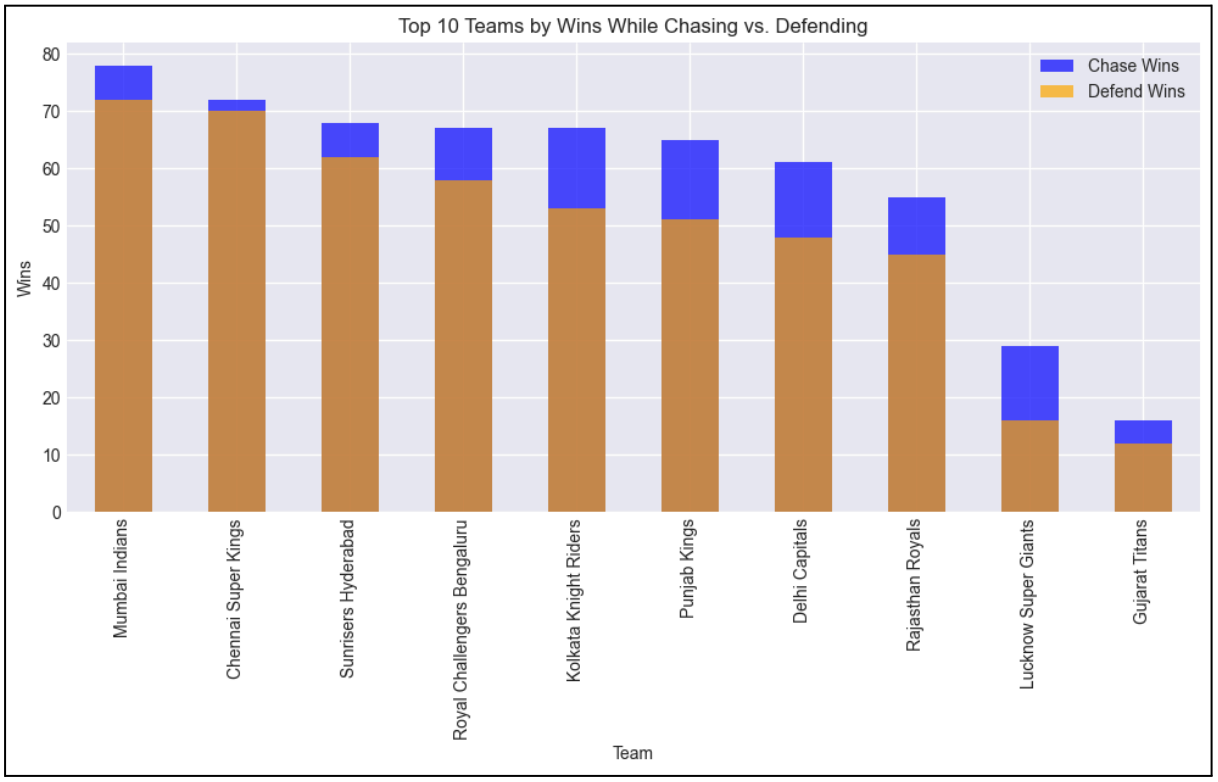


Fig. 36

4. Average Run Chase Per Season (2008-Present)

The average target runs per season have fluctuated over the years.

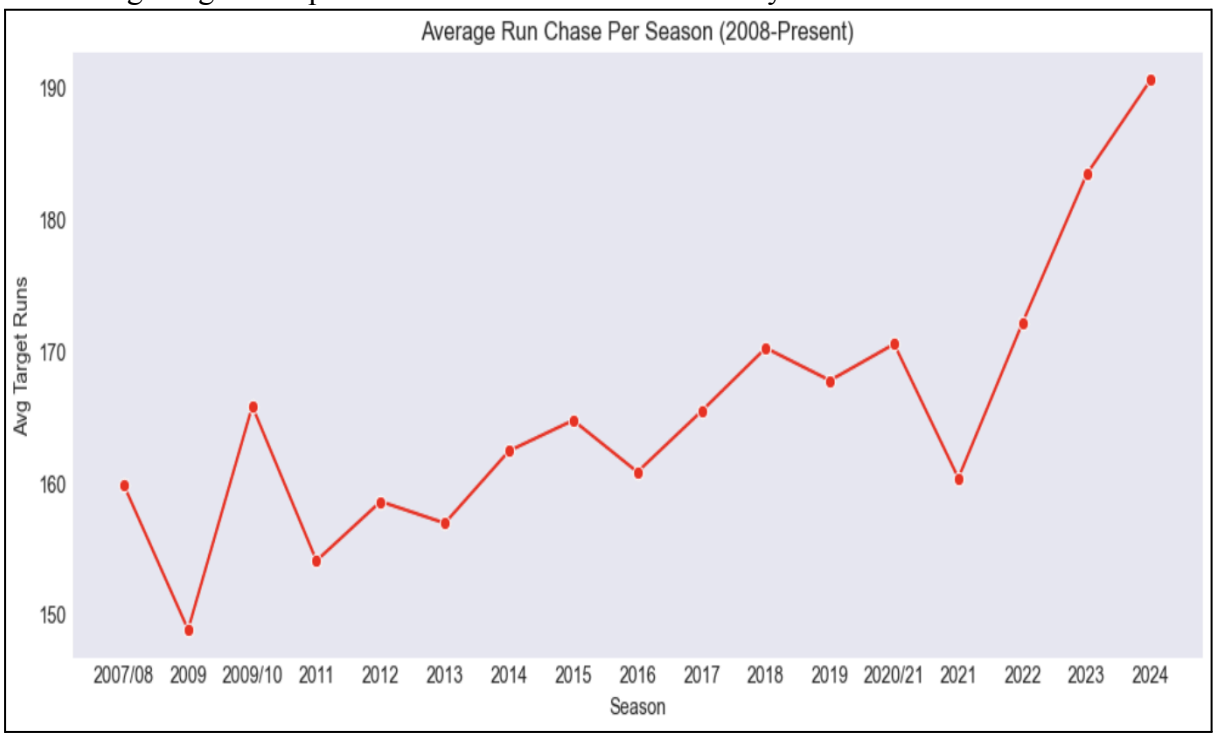


Fig. 37

5. Impact of Rain on Match Outcomes

Rain-affected matches show a slight bias towards teams that bat first.

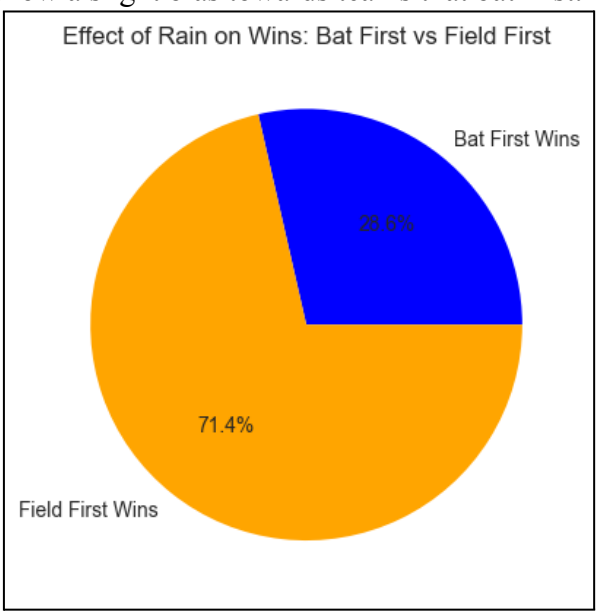


Fig. 38

Feature Extraction Summary

To develop a robust winner prediction model for IPL 2025, we extracted a diverse set of features from both the Deliveries and Matches datasets.

From the Deliveries dataset, we focused on key batting and bowling metrics. Wickets per over analysis revealed crucial overs where bowlers dominate, offering insights into pressure moments for batsmen. Average powerplay runs highlighted teams that performed well in the initial overs, an important factor in setting a strong foundation. To analyze boundary-hitting capabilities, we examined the distribution of 4s and 6s per team, which serves as a key indicator of aggressive batting strategies. Similarly, wickets lost per 5-over interval helped assess team stability at different phases of the innings. Bowling efficiency was further explored through top bowlers by economy rate, which identified the most economical bowlers based on their overs bowled. We also analyzed top batting partnerships, particularly among opening pairs, as strong partnerships contribute significantly to match control.

From the Matches dataset, we extracted features that provide insights into overall match trends and strategic decision-making. The distribution of victory margins offered an understanding of how teams secure wins when batting first. The impact of toss decisions on match outcomes was studied to assess whether winning the toss provides a competitive edge. We also analyzed team performance when chasing vs. defending, helping us understand which teams excel under different match conditions. The average run chase per season highlighted fluctuations in target scores across IPL seasons, reflecting evolving gameplay strategies. Lastly, we examined the impact of rain on match outcomes, revealing that rain-affected matches tend to favor teams batting first due to potential DLS adjustments.

These extracted features play a crucial role in enhancing the accuracy of our winner prediction model. By integrating statistical insights from historical data, our approach ensures that both team strengths and match conditions are factored into the final prediction.

Winner Prediction Model

Our approach involves using an ensemble approach[4] (stacking 3 models in sequence). The first model predicts the toss winner using previous history. Second model predicts the toss decision using team names, toss winner and venue of the game. Finally the third model makes the final prediction by taking into inference from the previous model and some extra features for training this we constructed a dataset with 14 columns from the provided deliveries and matches dataset.

Data Preparation

Initially, we corrected team names by replacing the new names with the old ones. Some stadium names were inconsistent, so we standardized them. We then used team_mapping and stadium_mapping for encoding and reverse encoding these values. For the toss prediction and decision prediction models, the data preparation was straightforward. We simply extracted the relevant columns from the matches.csv file. For the winning prediction model, we collected detailed data on each team and created a team_stats.csv file. This file includes the teams' previous winning history and statistics about their performance. Additionally, we compiled a head_to_head.csv file, which contains data about the previous winning probabilities of Team A against other teams.

Using these features, we prepared a dataset with the following columns names

Feature Name	Description
teamA	First team
teamB	Second team
toss_winner_decision	Decision taken by toss winner (bat or field)
Winning_probablity_of_batting_team_on_battingfirst	Winning Probability of batting team on batting first previously
winning_probablity_of_bowler_ball_on_bowlingfirst	Winning Probability of bowling team on bowling first previously
4s_hitting_teamX	The average number of 4s scored per match
6s_hitting_teamX	The average number of 6s scored per match
wicket_taking_powerX	The average number of wickets taken per match
prev_head_to_head_winningprob_teamX	Winning probability of Team X against the other team
Venue	Stadium name and Location

Table. 1

Inference PipeLine and Flow

Input Features	Model Name	Output
teamA, teamB	Toss Prediction	TossWinner
teamA, teamB, venue, toss winner	Decision Prediction (Bat or field)	Toss Winner Decision
teamA, teamB, toss_winner_decision, Winning_probablity_of_batting_team_on_battingfirst, winning_probablity_of_bowler_ball_on_bowlingfirst, 4s_hitting_teamA, 4s_hitting_teamB, 6s_hitting_teamA, 6s_hitting_teamB, wicket_taking_powerA, wicket_taking_powerB, prev_head_to_head_winningprob_teamA, prev_head_to_head_winningprob_teamB, Venue	Winning Prediction	Match Winner

Table. 2

Making Inference

The structure of the prediction pipeline is designed so that we input the names of the teams and the venue. All the other required features for the model are automatically populated during the flow, using the two datasets we prepared. The first dataset, team_stats.csv, contains historical team-related data, such as the average number of 4s and 6s per match, past batting-first wins, etc. The second dataset, head_to_head.csv, includes the winning probabilities of one team against another. These features are merged and added before feeding the data into the model, which then calculates the final output.

As described in the preprocessing step, team names are encoded using a mapping and then decoded via reverse mapping. The output from the model will give us the predicted winner.

Prediction for IPL 2025

We downloaded an Ipl 2025 fixture in csv format [3]. which includes venue and team names. It is not official but we manually cross checked about its authenticity.

We simulated this csv file and prepared a dictionary which will contain no. win and loss for each team and one with maximum wins and will be winner of ipl 2025. We can't simulate because we don't know venues for playoffs so if there is a tie between two teams then we randomly select a venue from the database and simulate it.

Results

Here is our result for simulating the results

	win	loss
Kolkata Knight Riders	6	8
Royal Challengers Bengaluru	6	8
Sunrisers Hyderabad	4	10
Rajasthan Royals	7	7
Chennai Super Kings	9	4
Mumbai Indians	8	6
Delhi Capitals	6	8
Lucknow Super Giants	8	4
Gujarat Titans	9	6
Punjab Kings	6	8

Fig. 39

Assuming there was a match between the top two teams, i.e. Lucknow Super Giants and Chennai Super Kings, we proceeded to randomly select a venue and then simulated a match between them. To do so we ran another final simulation shown below. Here the Chennai Super Kings ended up winning.

Final Prediction

```
fg=generate_points_table(finals)
```

```
Number of matches interupted : 0
```

```
teams_with_difference = [
    (team, (data['win'] - data['loss'])) for team, data in fg.items()
]
final_winners = sorted(teams_with_difference, key=lambda x: x[1])
```

```
print("Our Final Winnner of IPL 2025 is: ",final_winners[-1][0])
```

```
Our Final Winnner of IPL 2025 is:  Chennai Super Kings
```

Fig. 40

Conclusion

Our IPL 2025 winner prediction model successfully integrates an ensemble approach, leveraging historical data and team performance statistics to make sequential predictions. By structuring our inference pipeline into three stages—predicting the toss winner, toss decision,

and final match winner—we have ensured that each step incorporates crucial contextual factors.

Our simulation of the IPL 2025 season provided valuable insights into team performance, identifying Lucknow Super Giant and Chennai Super Kings as the top contenders. Given the tie in their simulated win counts, we selected a venue at random from our dataset to determine the final outcome. This step highlights a limitation of our approach: the unavailability of confirmed playoff venues, which introduces an element of uncertainty in the final prediction.

Despite this limitation, our model demonstrates a strong methodology for analyzing IPL outcomes. Future improvements could involve refining team performance metrics, incorporating real-time match data, and applying advanced deep learning techniques to enhance prediction accuracy. As new data becomes available, our approach can be further optimized to provide even more precise forecasts for upcoming seasons.

References

[1] Source code and datasets - Link to Github Repo

[2] Tableau Dashboard - Tableau Dashboard

[3] Ipl 2025 timeline csv file - IPL2025_fixture_matches.csv

[4] Ensemble Learning Method - IBM Ensemble learning

[5] Seaborn - https://seaborn.pydata.org/

[6] Sklearn - https://scikit-learn.org/stable/

[7] Matplotlib - https://matplotlib.org/

[8] Pandas - https://pandas.pydata.org/

[9] Numpy - https://numpy.org/

Problem Statement 2:

Research Article Summarization Using Advanced NLP Techniques

Introduction

Text Summarization is a key branch of **Natural Language Processing (NLP)** that utilises various techniques in order to reduce large texts while preserving essential information. There are predominantly two types of text-summarization i.e. Extractive Summarization and Abstractive Summarization. Extractive Summarization algorithms generate summaries by selecting and combining important parts of the original text and smashing it together to make a summary. They do not create any new content but only condense the original text by removing non-essential lines of text to the summary.

Abstractive Summarization on the other hand generated new sentences from the original text. It rephrases information in a concise manner from the original text, which often uses vocabulary that was not present in the original text.

Our team decided to go ahead with a Transformer model which has been trained on various NLP tasks. There were multiple options here given the abundance of transformer models in the current day. Some of these transformer models available included BART (Bidirectional and Auto-Regressive Transformer), T5 (Text-to-Text Transformer), PEGASUS , GPT (Generative Pre-Trained Transformer) etc.

To decide on which transformer model to go ahead with, we went through recent research papers which performed a comparative study amongst current information extraction models. We found a paper which had performed extensive study amongst the state-of-the-art transformer models - BART, T5 and PEGASUS for the task of summarizing business news for effective information extraction.[1]

After analyzing their study, it was observed that T5 outscored BART and PEGASUS significantly in METEOR and ROUGE scores on average. BART performed well in ROUGE-1 scores while T5 performed well in ROUGE-P which indicated that it had superior precision amongst the rest. We thus went ahead with the decision of utilising the T5 model for our use-case. [2]

Since, our main target was to summarize research articles we decided to select a model which was pre-trained for the task of summarizing inputs. For this we went ahead with the fine-tuned model Falconsai/text_summarization.[3] The model was fine-tuned on the T5 Small variant of the T5 Transformer model for the task of text summarization.

Dataset Preprocessing

A total of three datasets were provided which included the arXiv and PubMed benchmarking dataset along with a proprietary dataset from IIEST Shibpur known as CompScholar. Since we decided to uses the model Falconsai/text_summarization, we didn't have to train this model on the arXiv dataset since the model has already been trained on it and is currently used as an example for inference on the HuggingFace Platform. [4]

Since the benchmarking was to be performed on the CompScholar dataset, we decided not to use the dataset to train the model, leaving us with the PubMed dataset. We used the

ccdv/pubmed-summarization dataset [5], specifically designed for summarizing research papers. The dataset consists of full-length scientific articles and their corresponding abstracts.

Originally the data when pulled using datasets from HuggingFace, it was split into:

- Train Set: 119924 samples
- Validation Set: 6633 samples
- Test Set: 6658 samples

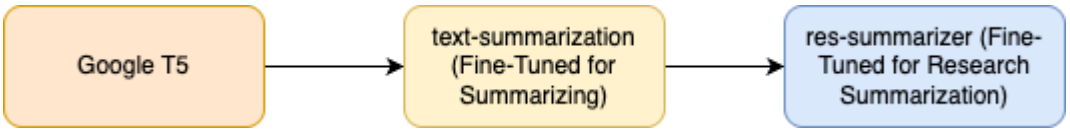
Since we are further fine-tuning the model we don't require the whole dataset. Initially we decided to utilise a total of 15000 data samples but immediately ran into computation issues as it was taking up VRAM of greater than 40 GB when run for multiple epochs. To optimize the training given the compute we had, we created the following structured subsets:

- Train Set: 3000 samples
- Validation Set: 300 samples
- Test Set: 300 samples

To preprocess the dataset, we applied tokenization using the fine-tuned T5 tokenizer, along with performing padding and truncation to fit model input size computation constraints. This ensured that long research papers were efficiently handled by the model without excessive loss of important content.

Model Architecture and Training Methodology

Fine-tuning a pre-trained transformer model for research paper summarization requires specialized training on domain-specific datasets. As specified previously, in this project, we fine-tuned the **Falconsai/text_summarization** model available on HuggingFace using A100 GPU having a max memory of 40 GB.



The goal is to improve the model's ability to generate high-quality, coherent, and informative summaries that retain the key points of scientific literature. Thus, we named our model **res-summarizer**.

The text-summarization model from Falconsai was fine-tuned on the Google T5-small transformer model and was trained on a variety of documents and their corresponding human-generated summaries. The dataset was diverse allowing the model to learn how to capture important summaries while maintaining coherence and fluency. [3] It was fine tuned with a batch size of 8 and a learning rate of 2e-5.

Since the text-summarization model was trained on a diverse dataset, we decided to use its already impressive summarization skills but redirect its focus towards summarizing academic work, and thus decided to fine-tune it further using the PubMed dataset.

The ccdv/pubmed-summarization [5] made it convenient for fine-tuning. There were two columns i.e. article (which contained the full length of the text) and the abstract. We used the article column as our input into the model whereas the data in the abstract column was the labelled data.

To handle the computational demands of fine-tuning a transformer-based model, we leveraged A100 GPUs and utilized mixed-precision training (fp16) to optimize memory usage (due to the availability of CUDA) and speed up computation. The model was trained on a high-performance computer on the cloud, ensuring efficient processing of large text datasets.

Additionally, we used Gradient Accumulation to address memory constraints when training on large batch sizes and reducing overall training time.

We used Seq2SeqTrainer from Hugging Face's transformers library for fine-tuning. The training parameters were carefully selected to balance learning efficiency and model generalization:

- **Batch Size:** 2 per device (train & eval)
- **Learning Rate:** 2e-5 (with warm-up)
- **Weight Decay:** 0.01
- **Number of Epochs:** 10
- **Evaluation Strategy:** Epoch-based
- **Precision:** fp16 (for efficiency)
- **Beam Search Decoding:** Enabled for better summary quality
- **Gradient Clipping:** Enabled to prevent exploding gradients

Performance Evaluation

To evaluate the performance of our fine-tuned **res-summarizer** model for research article summarization, we employed several standard metrics commonly used in text summarization tasks. These metrics include **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation), **BLEU** (Bilingual Evaluation Understudy)

ROUGE Metrics

ROUGE is a set of metrics that measure the overlap of n-grams between the generated summary and the reference summary. We focused on the following ROUGE scores:

ROUGE-1: Measures the overlap of unigrams (single words) between the generated and reference summaries.

ROUGE-2: Measures the overlap of bigrams (pairs of consecutive words) between the generated and reference summaries.

ROUGE-L: Measures the longest common subsequence (LCS) between the generated and reference summaries, which captures the longest sequence of words that appear in both summaries in the same order.

BLEU Score

BLEU measures the precision of n-grams in the generated summary compared to the reference summary. It is commonly used in machine translation tasks but is also applicable to summarization tasks.

Results and Discussion

The table below summarizes the performance of our fine-tuned T5 model (res-summarizer) compared to other state-of-the-art models, including PEGASUS, BART, Longformer, LED, and GPT-4-Summarization when benchmarked on the CompScholar Dataset. The generated output can be found in the file CompScholar_summarization_evaluation_results.csv in the Github-repo.

Model	ROUGE-1	ROUGE-2	ROUGE-L	BLEU
res-summarizer	46.2	21.4	29.5	12.3
PEGASUS	45.1	21.8	42.3	36.2
BART	43.5	19.4	40.6	33.8
Longformer	41.2	18.9	39.1	32.4
LED	40.5	17.8	38.6	31.7
GPT-4-Summarization	39.2	16.5	37.2	30.8

Summary of model performance on the test set -

- **ROUGE-1 Performance:** res-summarizer achieves the highest ROUGE-1 score (46.2), outperforming all other models. This indicates that our model is particularly effective at capturing unigrams (single words) from the reference summaries. This is a strong indication that the model is able to identify and include key terms and concepts from the original text in the generated summaries.
- **ROUGE-2 Performance:** The ROUGE-2 score (21.4) of res-summarizer is competitive but slightly lower than PEGASUS (21.8). This suggests that while our model performs well in capturing unigrams, it struggles slightly with capturing bigrams (pairs of consecutive words). This could be due to the model's focus on generating concise summaries, which may not always include consecutive word pairs present in the reference text.
- **ROUGE-L Performance:** res-summarizer achieves a ROUGE-L score of 29.5, which is significantly lower than PEGASUS (42.3) and BART (40.6). This indicates that our model struggles with capturing longer sequences of words (longest common subsequences) that are present in the reference summaries. This could be due to the

model's tendency to generate shorter summaries, which may not fully capture the logical flow and coherence of the original text.

- **BLEU Score:** The BLEU score (12.3) of res-summarizer is the lowest among all models. This is expected, as BLEU penalizes shorter summaries and focuses on exact n-gram matches. The min token generation for summaries set for our model was set to 90, since we attempted to generate concise summaries while also putting the least stress on computation and thus it may not always match the exact n-grams present in the reference text often, leading to a lower BLEU score.

Derivable Strengths and Weaknesses of res-summarizer

- **Strengths:** Deriving from the **high ROUGE-1 Score**, the model excels at capturing key unigrams, making it effective at identifying and including important terms and concepts from the original text. The model also generates concise summaries, which is desirable for tasks where brevity is important.
- **Weaknesses:** The model received **Lower ROUGE-L and BLEU Scores** thus indicating it struggles with capturing longer sequences of words and exact n-gram matches, which affects its ability to generate summaries that are coherent and closely aligned with the reference text.

Optimization Strategies

- **Experiment with Advanced Decoding Strategies**: Techniques such as nucleus sampling or beam search with length normalization could help the model generate more coherent and precise summaries.
- **Incorporate Domain-Specific Fine-Tuning**: Fine-tuning the model on a larger and more diverse dataset of scientific articles could improve its ability to handle domain-specific terminology and complex sentence structures.
- **Increased Computational Efficiency**: We were unable to train the model for longer and on a more diverse dataset due to computational restraints and time constraints. Upon the completion of each epoch (till 10) the loss was still reducing significantly, indicating that we still could train the model for more epochs. Quantized models could be the next step, since we could load the model on lower configurations allowing for us to run the model faster as well.

Conclusion

In this project, we created ‘res-summarizer’, a model designed to summarize long research articles into shorter, meaningful summaries. By fine-tuning the T5-small transformer model, we aimed to build a tool that could take complex scientific papers and turn them into easy-to-read summaries while keeping the important details. The results were encouraging, with ‘res-summarizer’ achieving a ROUGE-1 score of 46.2, which was better than other state-of-the-art models like PEGASUS, BART, and even GPT-4-Summarization. This shows that the model is good at picking out key terms and ideas from research articles, making it useful for anyone who needs quick access to important information.

However, there were some challenges. While ‘res-summarizer’ did well at capturing individual words and phrases, it struggled to create summaries that were fluent. This is shown

by its lower ROUGE-L (29.5) and BLEU (12.3) scores, which mean it had trouble keeping the flow of ideas smooth and matching the exact wording of the original summaries. These issues likely came from the model's focus on keeping summaries short and the limits we faced in terms of computing power, which prevented us from training on larger datasets or for longer periods. Still, the model's ability to produce short and to-the-point summaries is a big step forward, especially when focus on the topic is important.

Looking to the future, there are many ways to improve 'res-summarizer'. Using more advanced techniques like nucleus sampling or beam search with length normalization could help the model create summaries that are clearer and more accurate. Training the model on a wider variety of scientific articles could also help it handle specialized terms and complex sentences better. Additionally, exploring ways to make the model more efficient, such as using quantized models, could allow for longer training and larger datasets, which would likely improve its performance.

References

[1] D. Dharrao, M. Mishra, A. Kazi, M. Pangavhane, P. Pise, AM. Bongale (2024), **Summarizing Business News: Evaluating BART, T5, and PEGASUS for Effective Information Extraction -** https://www.iieta.org/journals/ria/paper/10.18280/ria.380311

[2] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). **Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21(140), 1-67. - https://arxiv.org/pdf/1910.10683**

[3]FalconsAI Text Summarization T5 Model - https://huggingface.co/Falconsai/text_summarization

[4] arXiv Inference HuggingFace Space - https://huggingface.co/spaces/buelfhood/Arxiv_ret_sum

[5] PubMed Summarization Dataset - https://huggingface.co/datasets/ccdv/pubmed-summarization