Shikshana Prasaraka Mandali's

Sir Parashurambhau College (Autonomous), Pune 30

DEPARTMENT OF STATISTICS

PROJECT TITLE: **IPL Winners' Prediction and their Analyses.**

Carried out by:

Shantaprasad Kamat (23613111)

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who helped us complete this project and complete it very well.

First and foremost, we would like to thank our statistics Prof. Santosh Kamble and Prof. Shantaram Dhum for guiding us throughout the entire process and providing valuable feedback and support.

We would also like to thank our project members who helped each other with data collection, cleaning, pre-processing, and analysis, as well as cooperated throughout working on the project.

Additionally, we would like to acknowledge data collectors of IPL and analysts whose work served as inspiration and provided the foundation for our project.

Finally, we would like to thank our statistics department and HOD Ma'am for allowing us to present our ideas through this project. Without their support and organization, this project would not have been possible. Thank you all for your contributions and support.

INDEX

INTRODUCTION

The Indian Premier League (IPL) is a professional T-20 cricket league in India. It was founded by the Board of Control for Cricket in India (BCCI) in 2007 and is considered the most popular cricket league in the world. The league consists of eight franchise teams representing different cities in India. The teams are owned by prominent business people, actors, and other celebrities, and each team represents a different region or city in India. The IPL season typically takes place over two months, usually between March and May, and features a total of 60 matches.

IPL matches are known for their electrifying atmosphere and fierce competition. The league attracts top international cricket players, making it a truly global event. The league has also become known for its innovative approaches to cricket, such as introducing strategic time-outs, cheerleaders, and player auctions.

Over the years, the IPL has become a significant revenue source for the BCCI and has helped in promoting and developing cricket in India. The league has also created numerous job opportunities for players, coaches, and other support staff, as well as contributing to the overall growth of the sports industry in India.

Why have we selected this topic for our project?

There are various reasons why we have selected to do a project on the Indian Premier League (IPL). Here are some reasons:

Popularity: The IPL is one of the most popular cricket leagues in India and it attracts a massive audience, both in India and abroad. It has a significant impact on the sports industry and the economy of India.

Data availability: With the advent of technology, there is a vast amount of data available on the IPL, including player statistics, team performances, and match data. This data can be used for analysis and research purposes.

Business and marketing: The IPL is not just a cricket tournament but also a platform for businesses to advertise and promote their products. Hence, studying the IPL can provide insights into the marketing strategies and business models of the teams and sponsors.

Social and cultural impact: The IPL has a significant impact on the social and cultural fabric of India, and analyzing the tournament can provide insights into the cultural and societal changes in the country.

Innovation: The IPL has been known for its innovative format and rule changes. Analyzing the IPL can provide insights into the importance of innovation in sports and how it can be used to attract new audiences and create a more engaging fan experience.

Significance of statistics in sports

Statistics plays a crucial role in sports as they provide valuable insights into the performance of individual players and teams. Here are some of the significances of statistics in sports:

Measuring Performance: Statistics help to measure and evaluate the performance of individual players and teams. They can be used to track progress, set benchmarks, and identify areas of improvement.

Scouting and Recruitment: Sports teams use statistics to scout and recruit new players. By analyzing a player's performance statistics, teams can identify potential candidates who may be a good fit for their team.

Strategic Planning: Statistics can be used to develop effective strategies and game plans. Coaches and players can use statistics to identify their opponents' weaknesses and plan accordingly.

In-Game Analysis: Real-time statistics can be used by coaches and players during a game to make quick decisions and adjustments. This can be particularly useful in sports like basketball, where the pace of the game is very fast.

Fan Engagement: Statistics can be used to engage fans and enhance their experience. Fans can use statistics to track their favorite player's progress and compare their performance with others.

Overall, statistics have become an integral part of modern sports. They help to provide valuable insights into player and team performance, support decision-making, and enhance the fan experience.

About Data/Metadata

We have two data sets i. e. ball by ball dataset and 2008 to 2022 dataset. This is a secondary dataset that we have taken from different websites. Here we have mention links of sources of the data sets:

https://www.kaggle.com/datasets/vora1011/ipl-2022-match-dataset

https://www.kaggle.com/datasets/patrickb1912/ipl-complete-dataset-20082020

https://www.kaggle.com/datasets/patrickb1912/indian-premier-league-2021-dataset

1] About Ball-by-ball data

In the ball-by-ball data set we mention details about each and every ball in all matches played in IPL (2008-2022). Through this data, we get the information that the no. of runs at a particular ball and how many illegal deliveries of balls happened.

Till 2022 we have data of 225794 (i. e. no. of rows) balls (excluding super-over). And total numbers of columns are 26 as explained in detail below:

a) match_ no: 1st column represents matches number till 2022, 951 matches were played in total.

b) match_ id: This column shows the match id number. For each match, there is a unique match id number. We can refer to it as a key column.

c) season: The given column shows the edition of the tournament every year.

d) date: In this column only the date of every match is mentioned. For example, suppose 18/4/2008 so in this column only 18 is present.

e) month: In this column only the month number of every match is mentioned. For example, suppose 18/4/2008 so in this column only 4 is present.

f) year: In this column only the year of every match is mentioned. For example, suppose 18/4/2008 so in this column only 2008 is present.

g) venue: This column represents the names of different cricket stadiums where IPL matches were played.

h) innings: An innings in cricket refers to the period of play where one team gets to bat while the other team fields. It is a fundamental unit of the game, and the objective of the batting team is to score runs while the fielding team aims to dismiss the batsmen.
Each team typically has two innings (i. e. 1 and 2) in a match, except in limited-overs cricket formats like in T-20 matches, after 20 overs 2nd innings starts.

i) over: An over is a set of six deliveries (or balls) bowled by a single bowler from one end of the pitch to the batsman at the other end. In IPL in the first innings, there are 20 overs, similarly, there are 20 overs in 2nd innings .so, there are 40 overs in one match.

j) ball: In this column ball number is mentioned like 1 to 6 in 1st over. If there is a wide or no ball in that case the number of balls increases.

k) bowling_ team: In this column, the name of the bowling team is mentioned.

l) striker: The given column shows the batsman on the end facing the bowler

m) non_ striker: The batsman standing at the opposite end of the striker

n) batting _team: The team having the turn to bat.

o) bowler: The column consists of the names of the bowlers in charge of the specific over.

p) runs_ off _bat: The runs gained by hitting the ball with the bat.

q) extras: Extras are the runs gained by an illegal ball i.e. a wide ball or a no ball or leg byes or byes.

r) wide: A wide ball is a ball thrown by a bowler which is outside of the batsmen's range.

s) no balls: A ball is said to be a no ball when the bowler crosses the crease line while throwing the ball, or when the ball goes directly above the waist of the batsman.

t) byes: Byes are credited to the team's total runs, but they are not attributed to any individual batsman. They are considered extras, along with wide and no balls, which are runs given away by the bowling team. Byes can contribute to the team's score without the batsman needing to play a shot.

u) leg byes: Leg byes are only awarded if the ball hits the batsman's body and not the bat. If the ball touches the bat before hitting the body, the runs scored are considered ordinary runs and not leg byes.

v) penalty: Penalty runs can be awarded to the opposing team as a result of misconduct by players or teams. These can include deliberate time-wasting,

ball tampering, or unfair play. The number of penalties runs awarded depends on the severity of the offense and is determined by the umpires.

w) wicket_type: The given column describes the way the batsman gets out.

x) player_dismissed: The player getting out.

y) other_wicket_type: The batsman getting out in a non-traditional manner.

z) other_player_dismissed: The batsman getting out in a non-traditional. manner.

2] About 2008-2022 data

In this data total we have 46 columns that we have explained in brief below:

1. **Match no.:** 1st column represents match number till 2022.

2. **Match ID:** This column shows the match id number. For each match, there is a unique match id number.

3. **Date:** It shows the specific date on which the given match was conducted.

4. **Month:** It shows the specific month in which the given match was conducted.

5. **Year:** It shows the specific year in which the given match was conducted.

6. **Venue:** The given column shows where the match was conducted i. e. the stadium, the state, or even the country.

7. **Team 1:** Team 1 means the team batting in the first innings.

8. **Team 2:** Team 2 means the team batting in the 2nd innings.

9. **Team1 Player1:** The 1st batsman of team 1.

10. **Team1 Player2:** The 2nd batsman of team 1.

11. **Team1 Player3:** The 3rd batsman of team 1.

12. **Team1 Player4:** The 4th batsman of team 1.

13. **Team 1 Player5:** The 5th batsman of team 1.

14. **Team1 Player6:** The 6th batsman of team 1.

15. **Team1 Player7:** The 7th batsman of team 1.

16. **Team1 Player8:** The 8th batsman of team 1.

17. **Team1 Player9:** The 9th batsman of team 1.

18. **Team1 Player10:** The 10th batsman of team 1.

19. **Team1 Player11:** The 11th batsman of team 1.

20. **Team2 Player1:** The 1st batsman of team 2.

21. **Team2 Player2:** The 2nd batsman of team 2.

22. **Team2 Player3:** The 3rd batsman of team 2.

23. **Team2 Player4:** The 4th batsman of team 2.

24. **Team2 Player5:** The 5th batsman of team 2.

25. **Team2 Player6:** The 6th batsman of team 2.

26 **Team2 Player7:** The 7th batsman of team 2.

27. **Team2 Player8:** The 8th batsman of team 2.

28. **Team2 Player9:** The 9th batsman of team 2.

29. **Team2 Player10:** The 10th batsman of team 2.

30. **Team2 Player11:** The 11th batsman of team 2.

31. **Toss winner:** The team who gets to make a decision to bat or ball.

32. **Toss decision:** The decision whether to bat or field.

33. **Stage:** The column shows the group stage or the playoff stage.

34. **First innings score:** The score made by the team 1.

35. **First innings wicket:** The wickets lost at the end of the first innings.

36. **Second innings score:** The score made by team 2 in the second innings.

37. **Second innings wicket:** Wickets lost by Team 2 in the second innings.

38. **Match winner:** The team who wins the match.

39. **Won by:** Column representing if the respective team won the match by runs or wickets.

40. **Margin:** Margin is the column that tells us by how many runs or by how many wickets a team won the match.

41 **Method:** The column indicates if the match was won normally or by Duckworth Lewis (D/L).

42.**SuperOver:** Column indicating if there was any tie that occurred in the given match.

43. **Player of the match:** Column indicating the names of the players of the match.

44. **POTM:** Column indicating the names of the players of the match.

45. **Umpire 1:** Name of the 1st umpire of the respective match.

46. **Umpire 2:** Name of the 2nd umpire of the respective match.

OBJECTIVES

1) To predict the winner of every IPL match based on the data of previous matches
2) To extract these features from the data to achieving the first objective:
   a) Batsman vs Bowler stats of every batsman in Team A vs every bowler in Team B
   b) Batsman and Bowler form of all the players in both the teams
   c) Record of both the teams at the venue of the match in last 5 years
   d) Win ratio of both the teams against each other in the last 5 years
   e) Chase vs Defend win ratio record at the venue of the match in the last 5 years
   f) Momentum of the both the teams
3) To fit various models on the extracted training data and select the model which gives the most accuracy.

DATA CLEANING

Data cleaning is a crucial step in the data preprocessing process. It involves identifying and handling various types of issues and inconsistencies in the dataset to ensure data quality and accuracy.

1] Ball by ball data:

Original data – We had original data in this format.

match_id	season	start_date	venue	innings	ball	batting_te	bowling_t	striker	non_strike	bowler	runs_off_ba	extras	wides	noballs	byes	legbyes	penalty	wicket_ty	player_dis	other_wic	other_player_dismissed
335982	2007/08	4/18/2008	M Chinnas	1	0.1	Kolkata Kr	Royal Cha	SC Gangul	BB McCull	P Kumar	0	1				1					
335982	2007/08	4/18/2008	M Chinnas	1	0.2	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	0.3	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	0	1	1								
335982	2007/08	4/18/2008	M Chinnas	1	0.4	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	0.5	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	0.6	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	0.7	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	0	1				1					
335982	2007/08	4/18/2008	M Chinnas	1	1.1	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	Z Khan	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	1.2	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	Z Khan	4	0									
335982	2007/08	4/18/2008	M Chinnas	1	1.3	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	Z Khan	4	0									
335982	2007/08	4/18/2008	M Chinnas	1	1.4	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	Z Khan	6	0									
335982	2007/08	4/18/2008	M Chinnas	1	1.5	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	Z Khan	4	0									
335982	2007/08	4/18/2008	M Chinnas	1	1.6	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	Z Khan	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	2.1	Kolkata Kr	Royal Cha	SC Gangul	BB McCull	P Kumar	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	2.2	Kolkata Kr	Royal Cha	SC Gangul	BB McCull	P Kumar	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	2.3	Kolkata Kr	Royal Cha	SC Gangul	BB McCull	P Kumar	0	1				1					
335982	2007/08	4/18/2008	M Chinnas	1	2.4	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	4	0									
335982	2007/08	4/18/2008	M Chinnas	1	2.5	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	P Kumar	1	0									
335982	2007/08	4/18/2008	M Chinnas	1	2.6	Kolkata Kr	Royal Cha	SC Gangul	BB McCull	P Kumar	0	0									
335982	2007/08	4/18/2008	M Chinnas	1	3.1	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	AA Noffke	0	5	5								
335982	2007/08	4/18/2008	M Chinnas	1	3.2	Kolkata Kr	Royal Cha	BB McCull	SC Gangul	AA Noffke	6	0									

Cleaned data- After the cleaning procedure we have data in this format.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
match_no	match_id	season	date	month	year	venue	innings	over	ball	bowling_t	striker	non_strik	batting_te	bowler	runs_off_l	extras	wides	noballs	byes	legbyes	penalty	wicket_ty	player_dis	other_wic	other_player_dismissed
1	335982	2007/08	18	4	2008	M. Chinna	1	0	1	Royal Chal	SC Gangul	BB McCull	Kolkata Kr	P Kumar	0	1				1					
1	335982	2007/08	18	4	2008	M. Chinna	1	0	2	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	P Kumar	0	0									
1	335982	2007/08	18	4	2008	M. Chinna	1	0	3	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	P Kumar	0	1	1								
1	335982	2007/08	18	4	2008	M. Chinna	1	0	4	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	P Kumar	0	0									
1	335982	2007/08	18	4	2008	M. Chinna	1	0	5	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	P Kumar	0	0									
1	335982	2007/08	18	4	2008	M. Chinna	1	0	6	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	P Kumar	0	0									
1	335982	2007/08	18	4	2008	M. Chinna	1	0	7	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	P Kumar	0	1				1					
1	335982	2007/08	18	4	2008	M. Chinna	1	1	1	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	Z Khan	0	0									
1	335982	2007/08	18	4	2008	M. Chinna	1	1	2	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	Z Khan	4	0									
1	335982	2007/08	18	4	2008	M. Chinna	1	1	3	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	Z Khan	4	0									
1	335982	2007/08	18	4	2008	M. Chinna	1	1	4	Royal Chal	BB McCull	SC Gangul	Kolkata Kr	Z Khan	6	0									

2] 2008-2022 IPL Matches data:

Original data-

Match ID	Date	Venue	Team1	Team2	Toss Winn	Stage	Toss Decis	First Innin	First Innin	Second Inr	Second Inr	Match Wir	Won by	Margin	Player of the match
1	16-04-201	Sheikh Zay	Kolkata Kn	Mumbai In	Kolkata Kn	Group	Bat	163	5	122	7	Kolkata Kn	Runs	41	Jacques Kallis
2	17-04-201	Sharjah Cri	Delhi Dare	Royal Chal	Royal Chal	Group	Field	145	4	146	2	Royal Chal	Wickets	8	Yuzvendra Chahal
3	18-04-201	Sheikh Zay	Chennai Su	Kings XI Pu	Chennai Su	Group	Bat	205	4	206	4	Kings XI Pu	Wickets	6	Glenn Maxwell
4	18-04-201	Sheikh Zay	Sunrisers H	Rajasthan	Rajasthan	Group	Field	133	6	135	6	Rajasthan	Wickets	4	Ajinkya Rahane
5	19-04-201	Dubai Inte	Mumbai In	Royal Chal	Royal Chal	Group	Field	115	9	116	3	Royal Chal	Wickets	7	Parthiv Patel
6	19-04-201	Dubai Inte	Kolkata Kn	Delhi Dare	Kolkata Kn	Group	Bat	166	5	167	6	Delhi Dare	Wickets	4	JP Duminy
7	20-04-201	Sharjah Cri	Rajasthan	Kings XI Pu	Kings XI Pu	Group	Field	191	5	193	3	Kings XI Pu	Wickets	7	Glenn Maxwell
8	21-04-201	Sheikh Zay	Chennai Su	Delhi Dare	Chennai Su	Group	Bat	177	7	84	10	Chennai Su	Runs	93	Suresh Raina
9	22-04-201	Sharjah Cri	Kings XI Pu	Sunrisers H	Sunrisers H	Group	Field	193	6	121	10	Kings XI Pu	Runs	72	Glenn Maxwell
10	23-04-201	Dubai Inte	Chennai Su	Rajasthan	Rajasthan	Group	Field	140	6	133	10	Chennai Su	Runs	7	Ravindra Jadeja
11	24-04-201	Sharjah Cri	Kolkata Kn	Royal Chal	Royal Chal	Group	Field	150	7	148	5	Kolkata Kn	Runs	2	Chris Lynn
12	25-04-201	Dubai Inte	Sunrisers H	Delhi Dare	Sunrisers H	Group	Bat	184	1	180	4	Sunrisers H	Runs	4	Aaron Finch
13	25-04-201	Dubai Inte	Mumbai In	Chennai Su	Mumbai In	Group	Bat	141	7	142	3	Chennai Su	Wickets	7	Mohit Sharma

This is an example of the 2014 data set available in this format. Similarly, from 2008 to 2022 data is available in this format only separately for each year. We have combined all years' data in one dataset.

Cleaned data-

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
Match no	Match ID	Date	Month	Year	Venue	Team1	Team2	Team1 Play	Team1 Play	Team1 Pla	Team1 Pla	Team1 Pla	Team1 Play	Team1 Play	Team1 Play	Team1 Play	Team1 Play	Team1 Pla	Team2 Pla	Team2 Pla	Team2 Playe
1	1	18	4	2008	M. Chinnasv	Kolkata Knight	Royal Chal	SC Ganguly	BB McCullu	RT Ponting	DJ Hussey	Mohamma	LR Shukla	WP Saha	AB Agarkar	AB Dinda	M Kartik	I Sharma	R Dravid	W Jaffer	V Kohli
2	2	19	4	2008	Punjab Crick	Chennai Super	Punjab Kin	PA Patel	ML Hayden	MEK Huss	MS Dhoni	SK Raina	JDP Oram	S Badrinath	Joginder Sha	P Amarnath	MS Gony	M Muralitl	K Goel	JR Hopes	KC Sangakkar
3	3	19	4	2008	Arun Jaitley	Rajasthan Roy	Delhi Capi	T Kohli	YK Pathan	SR Watsor	M Kaif	DS Lehmar	RA Jadeja	M Rawat	D Salunkhe	SK Warne	SK Trivedi	MM Patel	G Gambhir	V Sehwag	S Dhawan
4	4	20	4	2008	Eden Garder	Sunrisers Hyde	Kolkata Kn	AC Gilchrist	Y Venugopa	VVS Laxma	A Symonds	RG Sharma	SB Styris	AS Yadav	SB Bangar	WPUJC Vaa	RP Singh	PP Ojha	WP Saha	BB McCull	RT Ponting
5	5	20	4	2008	Wankhede S	Mumbai Indian	Royal Chal	L Ronchi	ST Jayasuriy	DJ Thornel	RV Uthapp	PR Shah	AM Nayar	SM Pollock	Harbhajan S	MA Khote	A Nehra	DS Kulkarn	S Chanderp	R Dravid	LRPL Taylor
6	6	21	4	2008	Sawai Mans	Punjab Kings	Rajasthan	K Goel	JR Hopes	KC Sangak	DPMD Jay	Yuvraj Sing	IK Pathan	S Sohal	B Lee	PP Chawla	WA Mota	S Sreesant	M Kaif	Kamran Ak	YK Pathan
7	7	22	4	2008	Rajiv Gandh	Sunrisers Hyde	Delhi Capi	AC Gilchrist	Y Venugopa	VVS Laxma	A Symonds	RG Sharma	Shahid Afrid	SB Bangar	AS Yadav	WPUJC Vaa	RP Singh	PP Ojha	G Gambhir	V Sehwag	S Dhawan
8	8	23	4	2008	M. A. Chidar	Chennai Super	Mumbai In	PA Patel	ML Hayden	MEK Huss	SK Raina	MS Dhoni	JDP Oram	S Badrinath	Joginder Sha	P Amarnath	MS Gony	M Muralitl	L Ronchi	ST Jayasur	RV Uthappa

W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT
Team2 Play	Team2 Pla	Team2 Pla	Team2 Pla	Team2 Pla	Team2 Pla	Team2 Pla	Team2 Pla	Toss winne	Toss decis	Stage	First innin	First innin	Second in	Second in	Match wi	Won by	Margin	Method	SuperOve	Player of t	POTM	Umpire1	Umpire2
JH Kallis	CL White	MV Bouche	B Akhil	AA Noffke	P Kumar	Z Khan	SB Joshi	Royal Chall	Field	Group	222	3	82	10	Kolkata Kn	Runs	140	NA	N	Brendon M	BB McCull	Asad Rauf	RE Koertzen
Yuvraj Sing	SM Katich	IK Pathan	P Dharmar	B Lee	PP Chawla	WA Mota	S Sreesant	Chennai Su	Bat	Group	240	5	207	4	Chennai Si	Runs	33	NA	N	Michael H	MEK Huss	MR Benson	SL Shastri
MK Tiwary	KD Karthik	R Bhatia	M Manhas	DL Vettori	MF Mahar	B Geeves	GD McGra	Rajasthan F	Bat	Group	129	8	132	1	Delhi Capi	Wickets	9	NA	N	Farveez M	MF Mahar	Aleem Dar	GA Pratapkumar
SC Ganguly	DJ Hussey	Mohamma	LR Shukla	AB Agarka	AB Dinda	M Kartik	I Sharma	Sunrisers H	Bat	Group	110	10	112	5	Kolkata Kn	Wickets	5	NA	N	David Hus	DJ Hussey	BF Bowden	K Hariharan
JH Kallis	V Kohli	MV Bouche	B Akhil	P Kumar	Z Khan	SB Joshi	R Vinay Ku	Mumbai Inc	Bat	Group	165	6	166	5	Royal Chal	Wickets	5	NA	N	Mark Bouc	MV Bouch	SJ Davis	DJ Harper
SR Watson	DS Lehmar	RA Jadeja	Pankaj Sin	D Salunkhe	SK Warne	SK Trivedi	MM Patel	Punjab King	Bat	Group	166	8	168	4	Rajasthan	Wickets	6	NA	N	Shane Wat	SR Watsor	Aleem Dar	RB Tiffin
Shoaib Mal	KD Karthik	MK Tiwary	R Bhatia	MF Mahar	Mohamma	VY Mahes	GD McGra	Sunrisers H	Bat	Group	142	8	143	1	Delhi Capi	Wickets	9	NA	N	Virender S	V Sehwag	IL Howell	AM Saheba

We are able to see the changes in the above picture. In the original ball-by-ball dataset there is not mentioned any details of match number.so, we have added a new column of match numbers in the cleaned dataset.

Further, in the original dataset date of matches are in a single column in the format like this 18/4/2008 we split it into three different columns in the cleaned dataset i. e. 18(date) in a separate column,4(month) in a separate column, and 2008(year) in a separate column as per our requirement of the project, and also it is convenience of our further analysis.

In the original dataset, there is the same column for overs with the ball (i.e2.4 means 2overs and 4th ball of the 3rd over) but we split it into two columns: overs and balls.

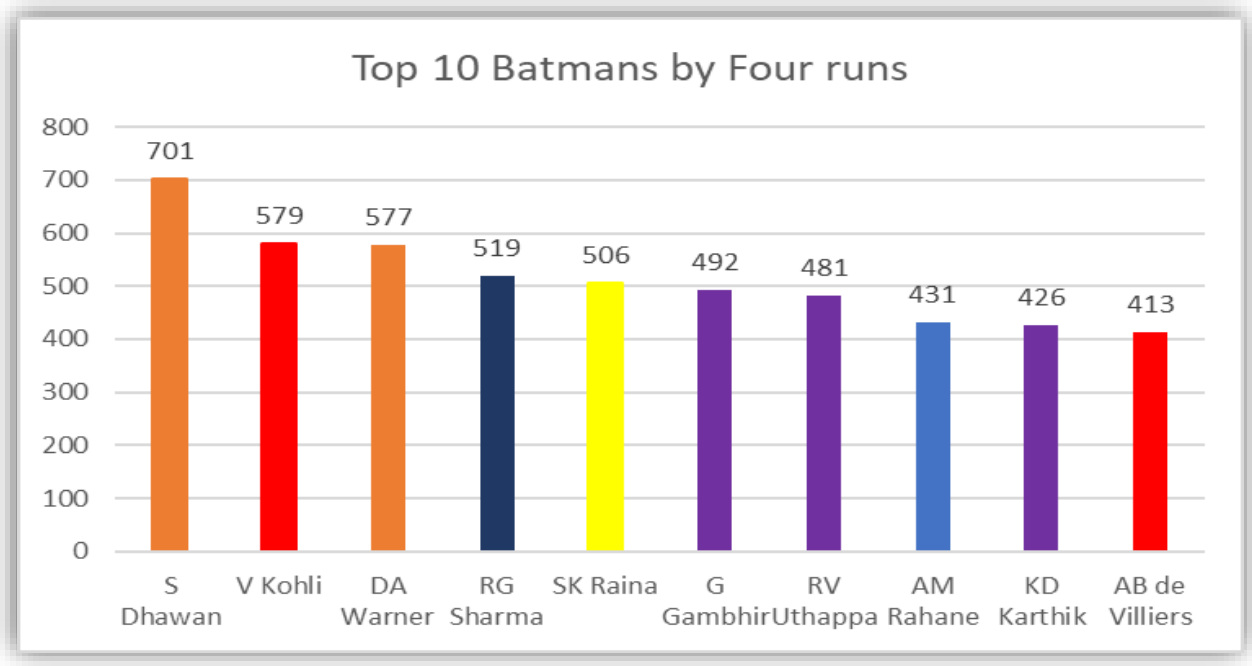
Also, we have removed super over balls detail from our data because in the records of particular player runs gained by Super over is not considered. So, it is not required for our further analysis.

Same modification/cleaning required for 2008-2022 datasets. Originally this dataset was available year wise separately we combined it in one dataset and we were required to change some team names like before February 2021 Punjab Kings is known by Kings Xl Punjab. Similarly for some stadiums names are changed so we had modified that because by changing names data values do not change.

INSIGHTS FROM DATA

Insights from data refer to valuable information and knowledge that is obtained through analyzing and interpreting data. These insights enable us to make informed decisions, solve problems, identify patterns, and gain a deeper understanding of the subject matter at hand.

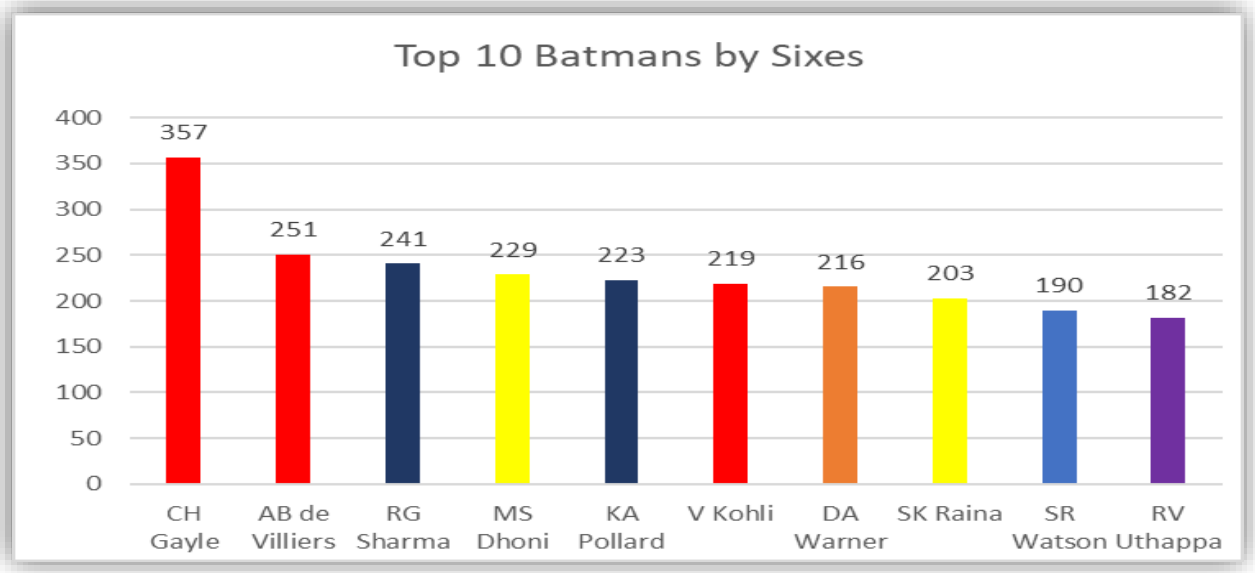
1] Most Fours in IPL:



#Shikhar Dhawan has hit the most number of fours in all IPL seasons.

Followed by him Virat Kohli and David Warner. There is a huge difference between the first two top batsmen.

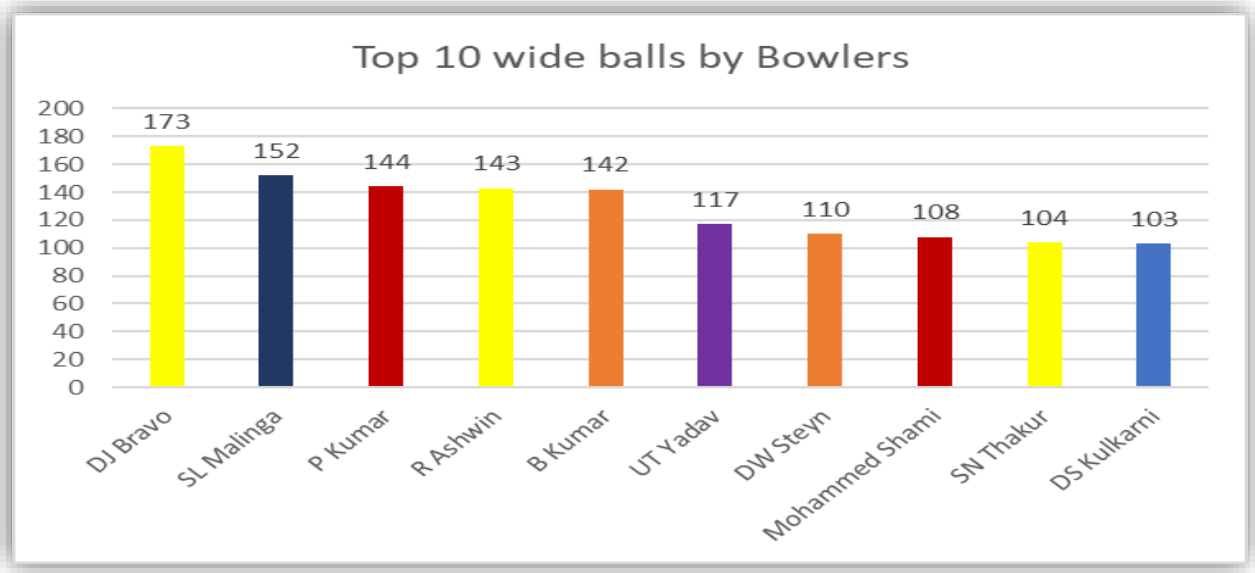
2] Most sixes in IPL:



Chris Gayle has hit the most number of sixes in all IPL seasons.

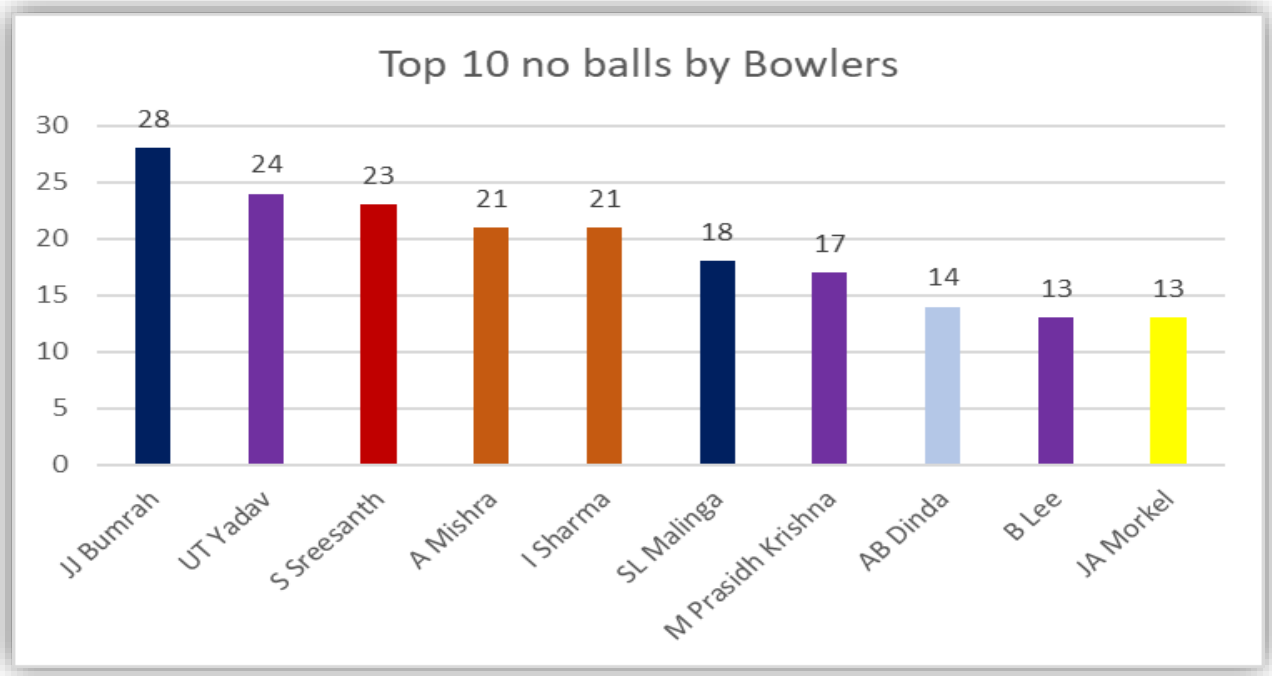
Followed by him AB de Villiers and Rohit Sharma. There is a little huge difference between the first two batman.

3) Most wide balls by bowlers:



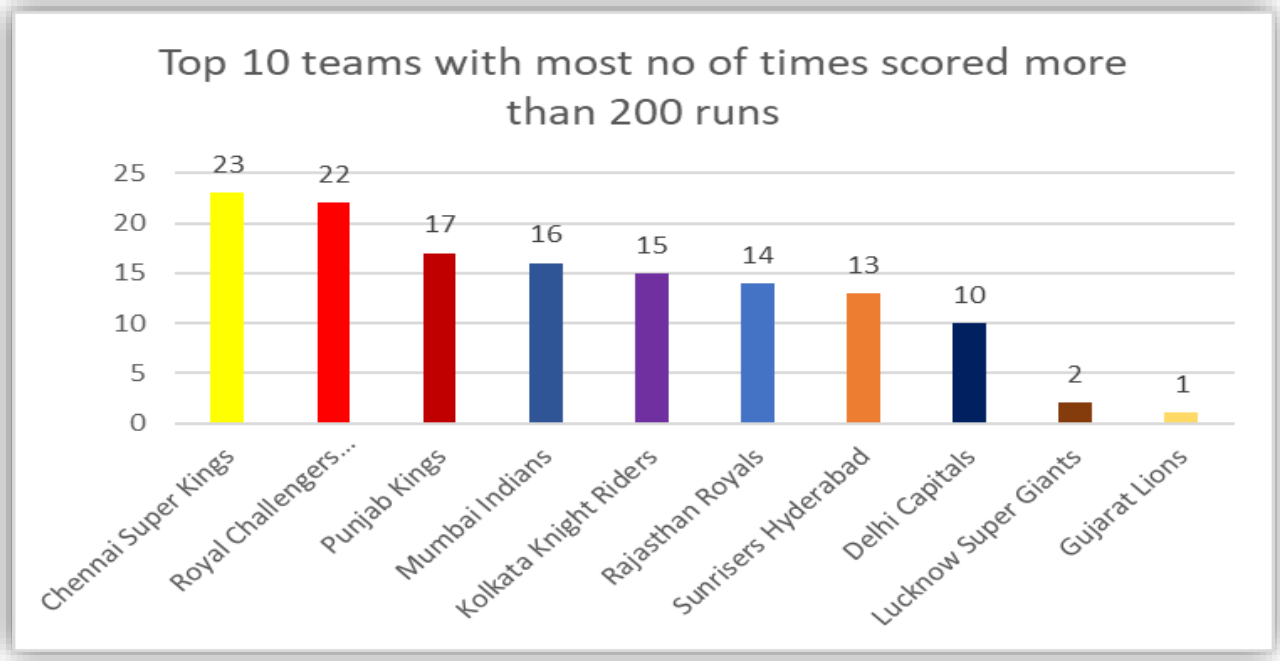
D J Bravo bowled most wide balls throughout IPL seasons.

4] Most no balls by bowlers:



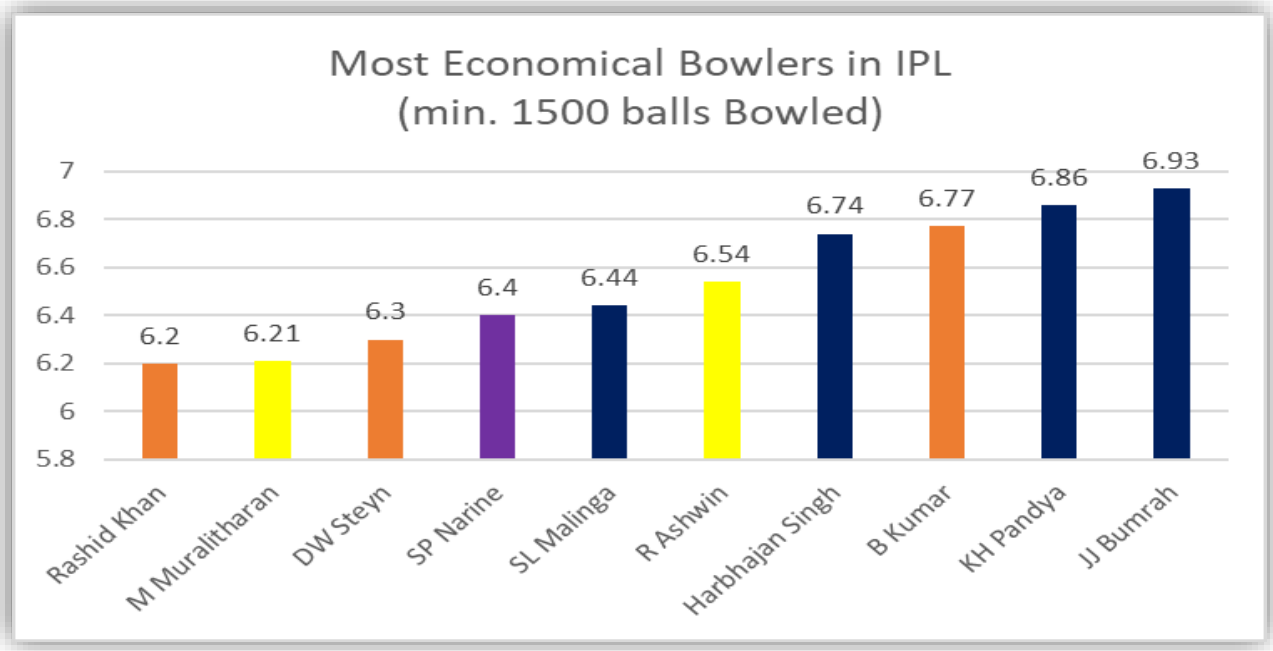
J.Bumrah has bowled the most no. of no balls throughout IPL.

5] Teams with most no. of times 200's:



CSK has scored 200+ in 1st innings the most no. of times.

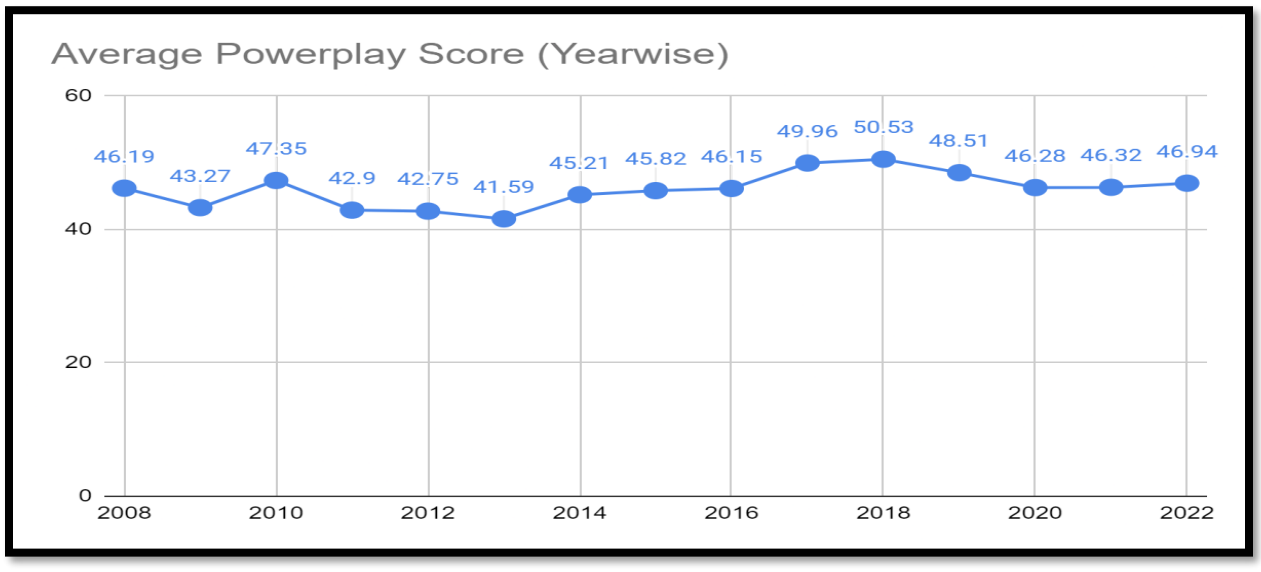
6] Most economical bowler in IPL:



Rashid Khan has been the most economical bowler throughout the IPL.

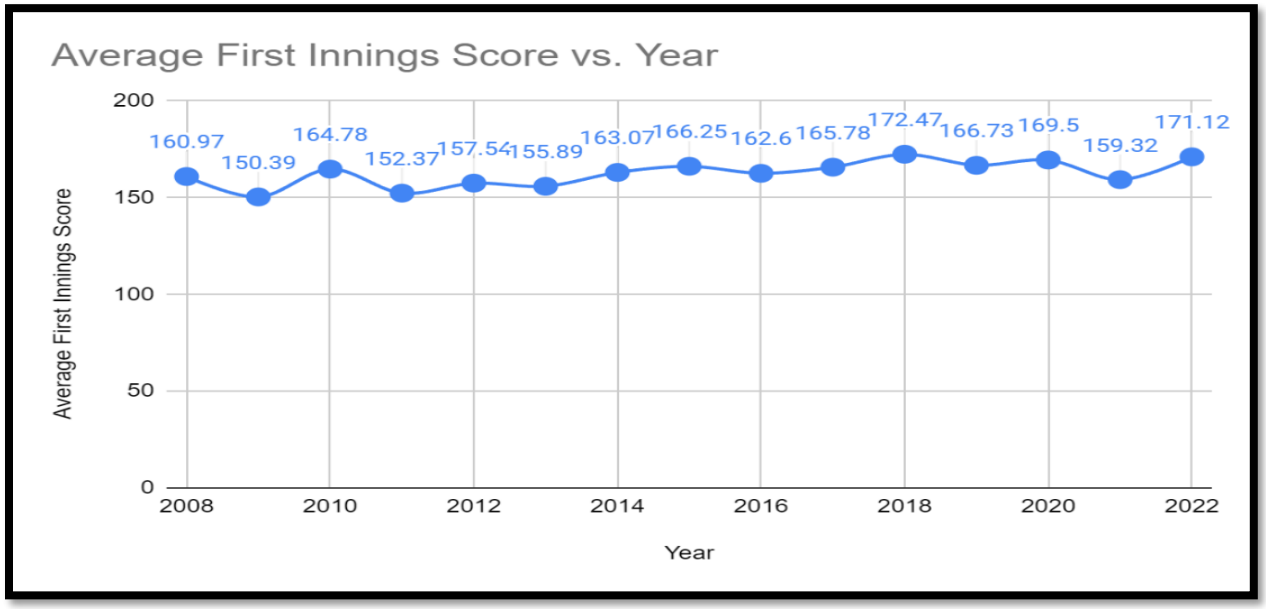
There are 6 spinners and 4 fast bowlers in the above top list.

7] Average score in Powerplay:



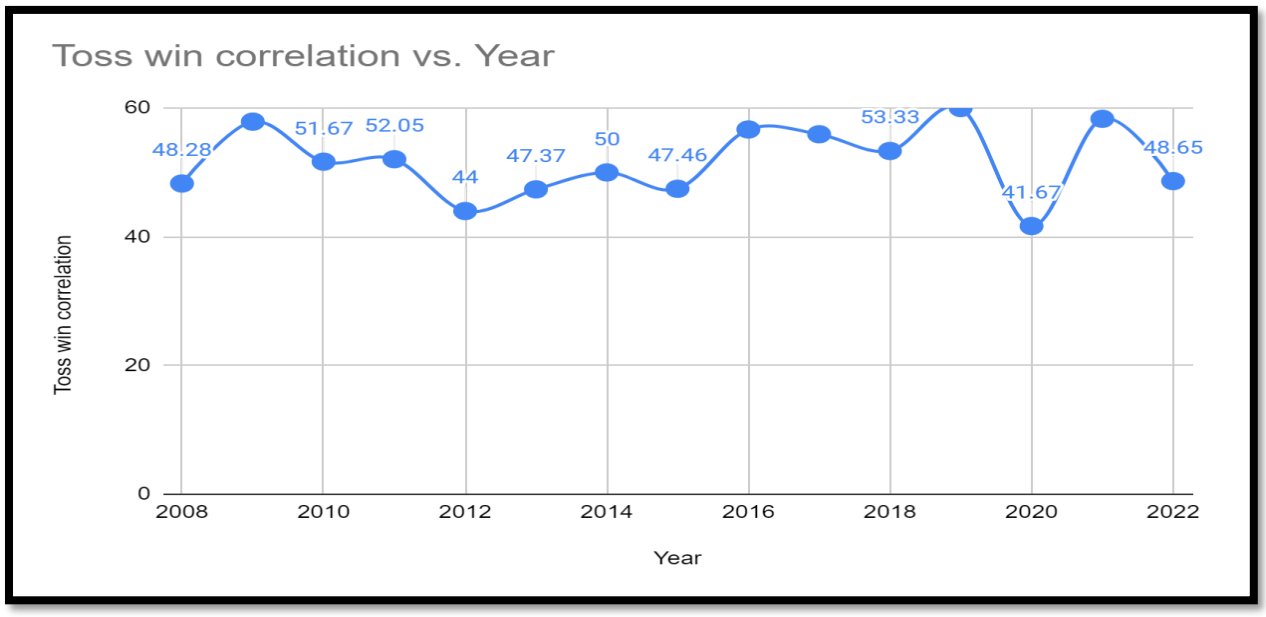
There is not much variation in the Average powerplay score of seasons throughout the IPL.

8] Average first innings score:



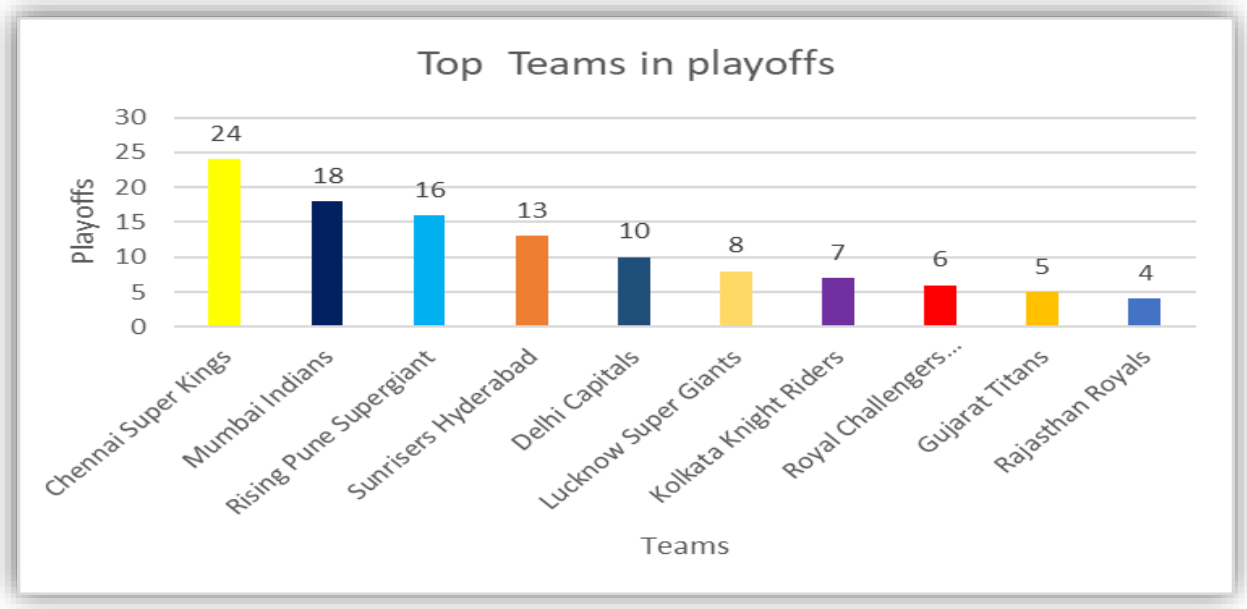
The lowest first-inning score was in 2009 and the highest first-inning score is in 2018. This shows the progress in batting in IPL through the years.

9] The probability of toss winning team winning the match:

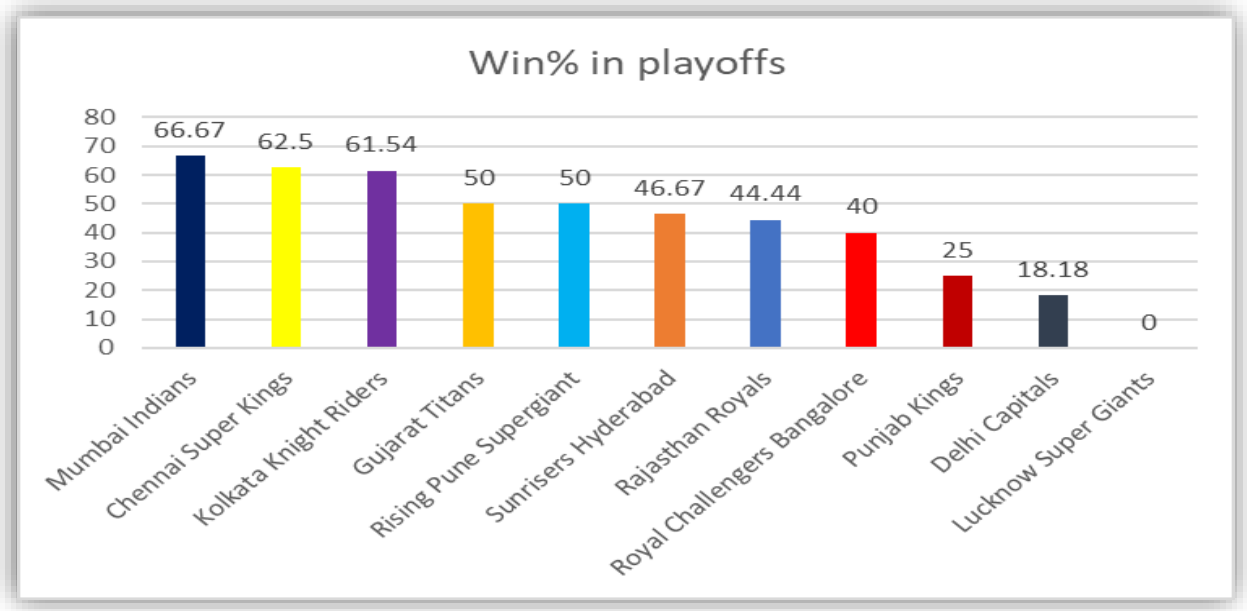


We can see that in 2020 if the team won the toss there is 40 percent chance that team won the match and in 2019 if the team won the toss there is 60 percent chance that team won the match

10] Top teams in playoffs:



This shows that CSK and MI are the teams that are selected the most in the playoffs.

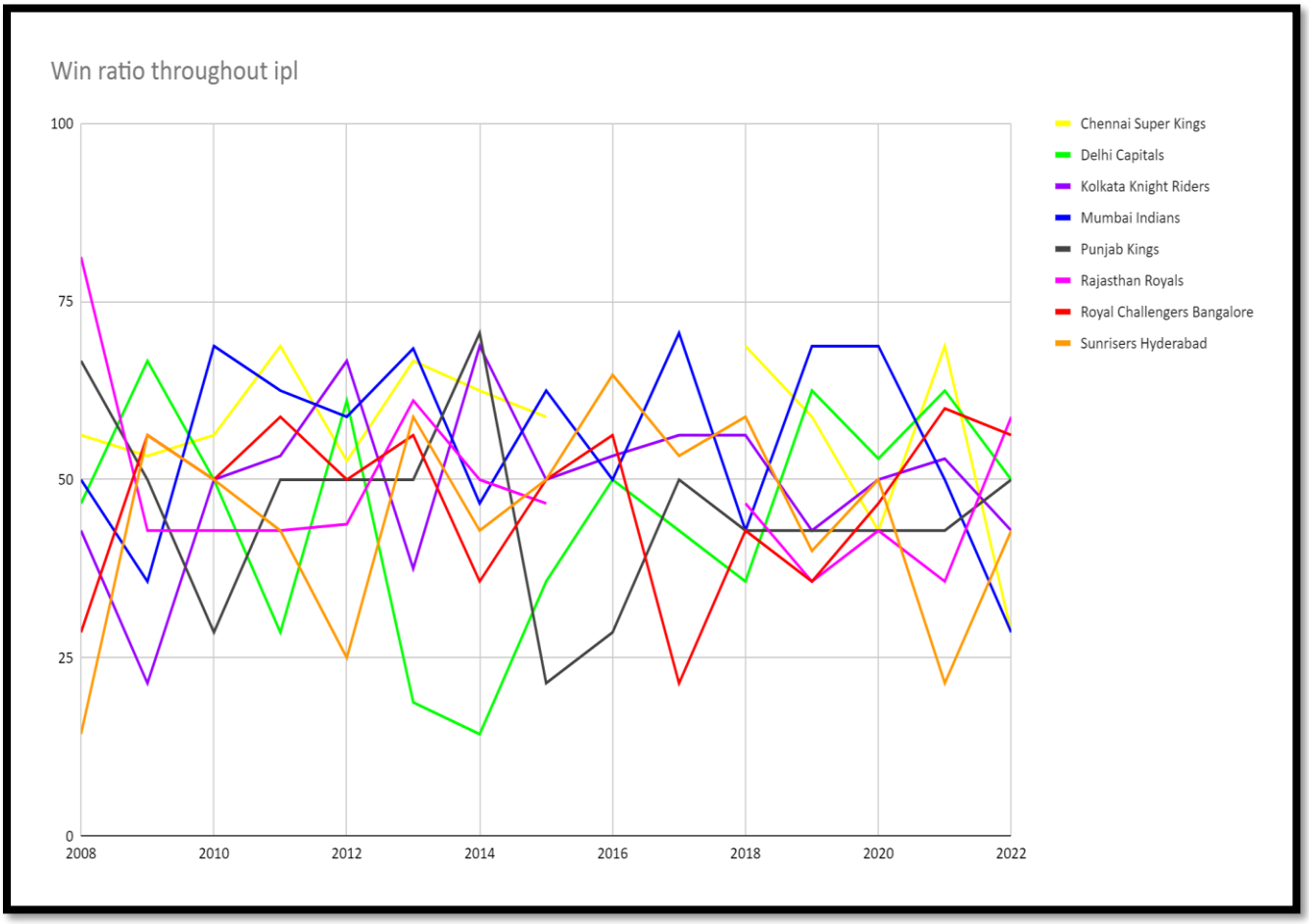


There is a 66.67% and 62.5% probability that CSK and MI have won in the playoff matches. Also, we can see that RCB is in the lower half of the list, and that may be the reason for the fact that they have never won the trophy.

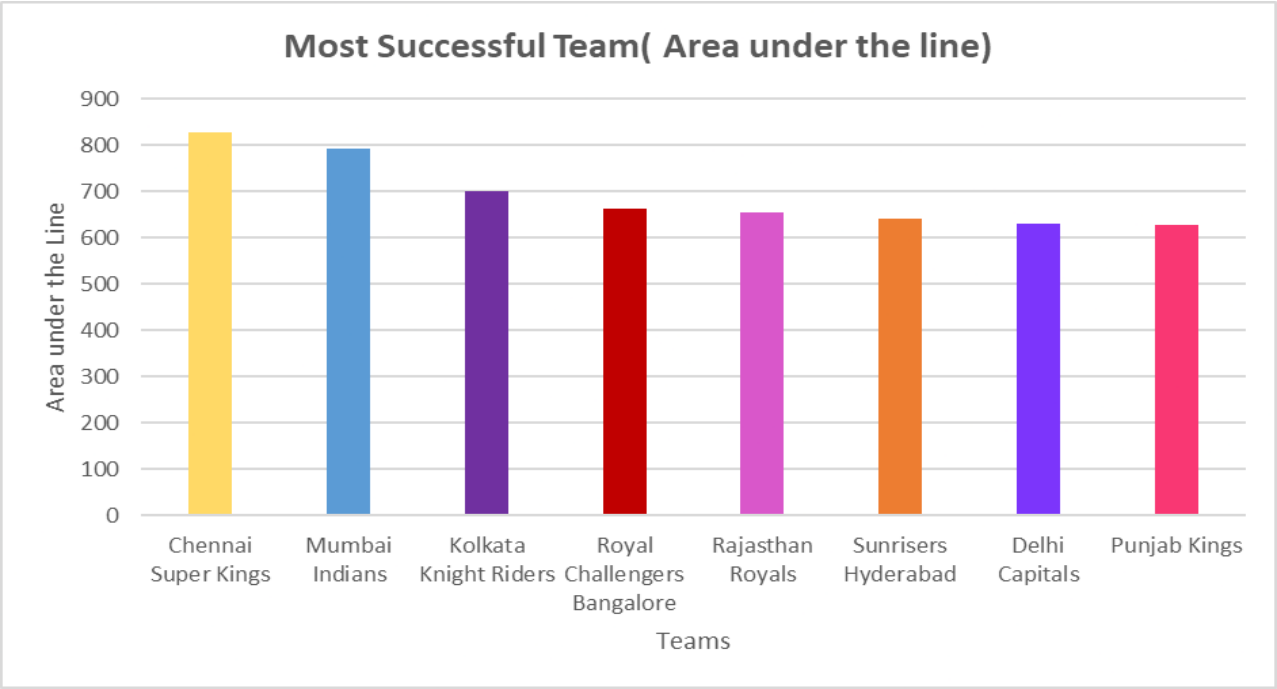
TEAM VS TEAM WIN LOSS RATIO:

	CSK	Deccan Chargers	Delhi Capitals	PBKS	KTK	KKR	MI	PWI	RR	RCB
CSK		6/10	6/10	6/10	1/2	6/10	5/11	3/4	7/11	7/12
Deccan Chargers	4/10		4/11	3/10	1	2/9	4/10	3/4	2/9	6/11
Delhi Capitals	4/10	7/11		5/10	1/2	4/9	5/10	2/3	6/10	5/9
PBKS	4/10	7/10	5/10		1	4/9	6/10	1/4	3/9	5/10
KTK	1/2	0	1/2	0		2/9	1	0	1/2	0
KKR	4/10	7/9	5/9	5/9	0		2/10	1	5/10	5/10
MI	6/11	6/10	5/10	4/10	0	8/10		3/4	5/9	6/11
PWI	1/4	1/4	1/3	3/4	1	0	1/4		0	0
RR	4/11	7/9	4/10	6/9	1/2	5/10	4/9	1		4/9
RCB	5/12	5/11	4/9	5/10	1	5/10	5/11	1	5/9	

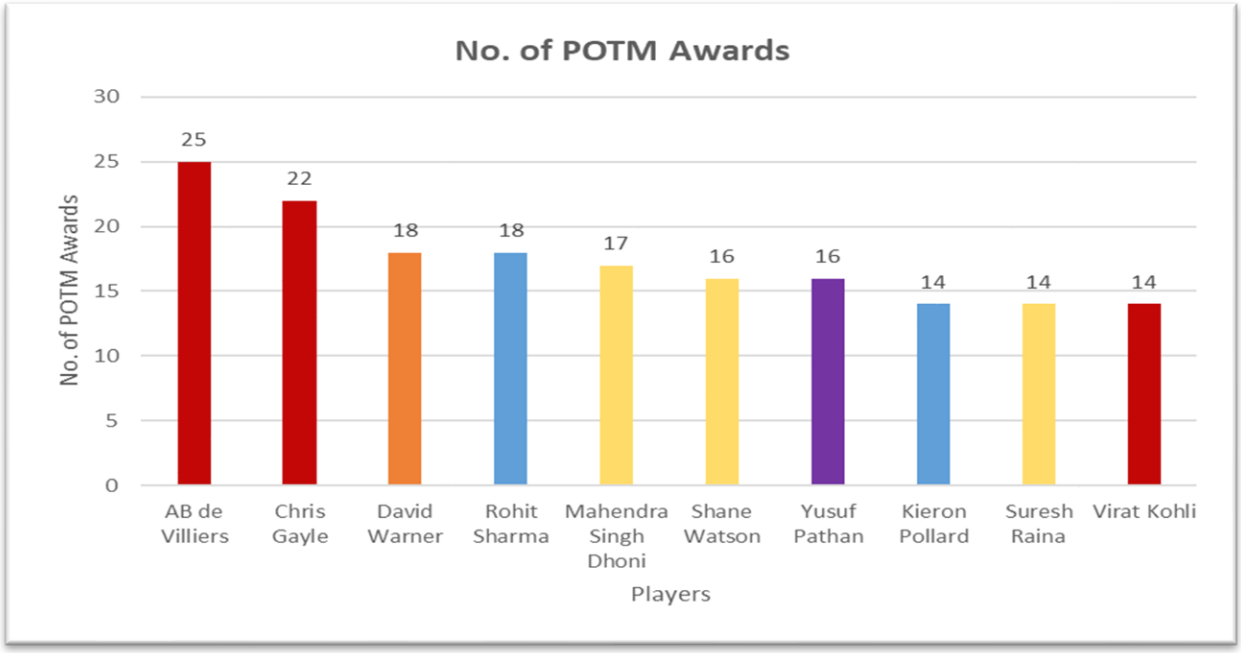
WIN RATIO THROUGH THE IPL



#This graph shows us that CSK has been performing consistently throughout the IPL. Also, RR was very dominant in the year 2008, and hence they won the IPL 2008. But they have never won the IPL since, as it can be seen through this graph.



#This graph was computed by taking the area under the curve of the above graph. As expected, MI and CSK have been the most dominant teams throughout the IPL.



#As we can see, AB De Villiers has won the most Player of The Match awards. Gayle and Kohli are also on this list, but as RCB has never won the IPL, it shows that RCB may be too over-reliant on star players.

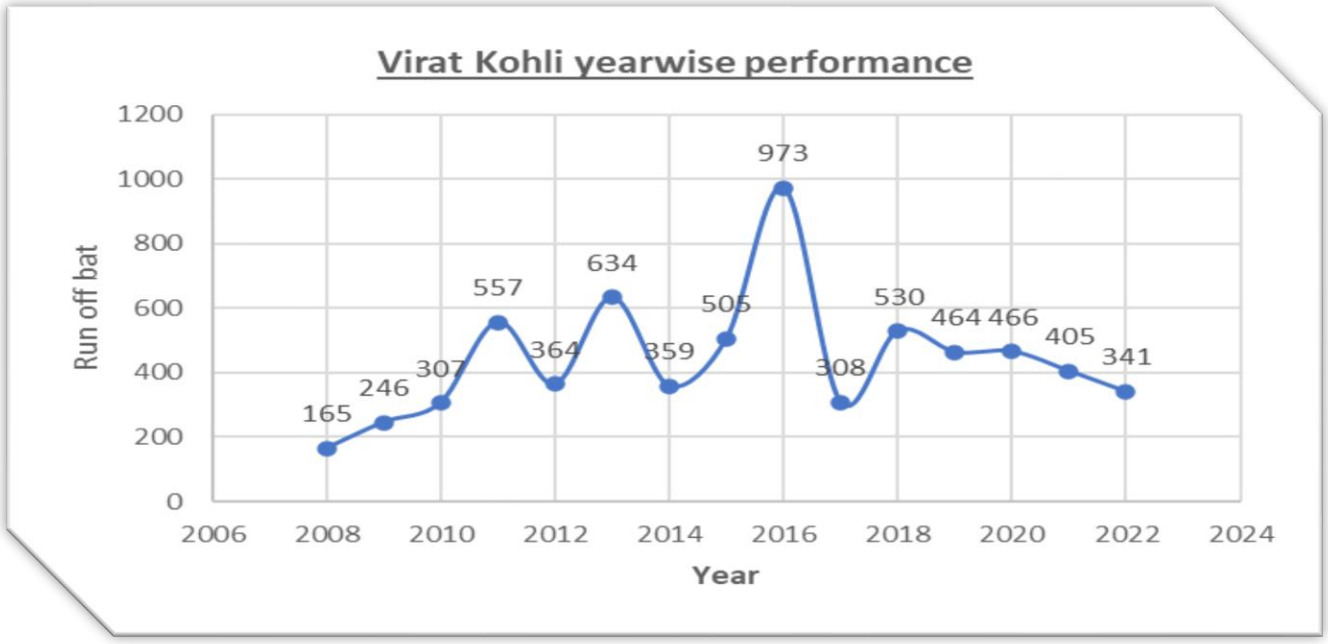
Year Wise Performance of Batsman

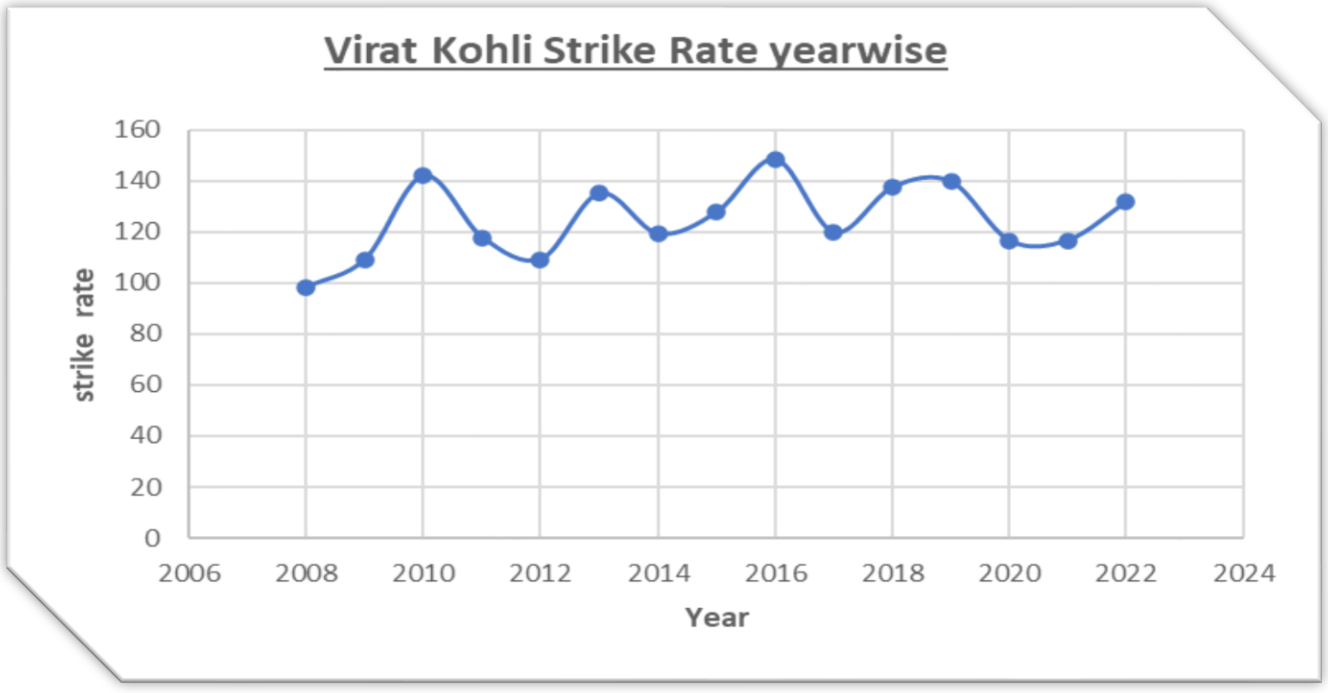
```
import pandas as pd


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

def Total_Runs_of_Batsman_yearwise(x):
  vk_matches = matches.loc[matches['striker'] == x]


  yearly_performance = vk_matches.groupby('year')['runs_off_bat'].sum().reset_index()


  print(yearly_performance)
```

Year Wise Strike Rate of Batsman

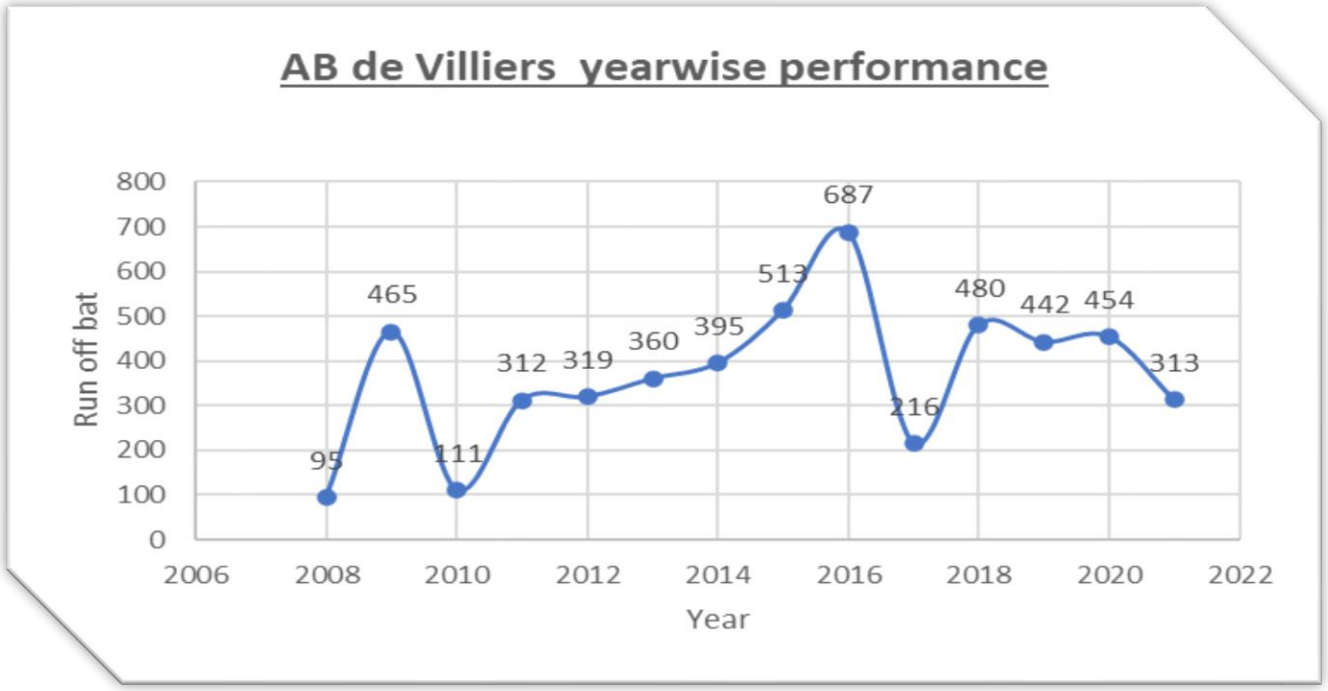
```
import pandas as pd
df = pd.read_csv("/content/ipl_match_ball_by_ball_data_cleaned.csv")



# assuming your DataFrame is named df
# filter to get rows where striker is Virat Kohli
vk_df = df[df['striker'] == 'V Kohli']

# group by year and calculate strike rate for each year
strike_rates = vk_df.groupby('year').apply(lambda x: (x['runs_off_bat'].sum() / len(x)) * 100)

# print the resulting DataFrame
print(strike_rates)
```

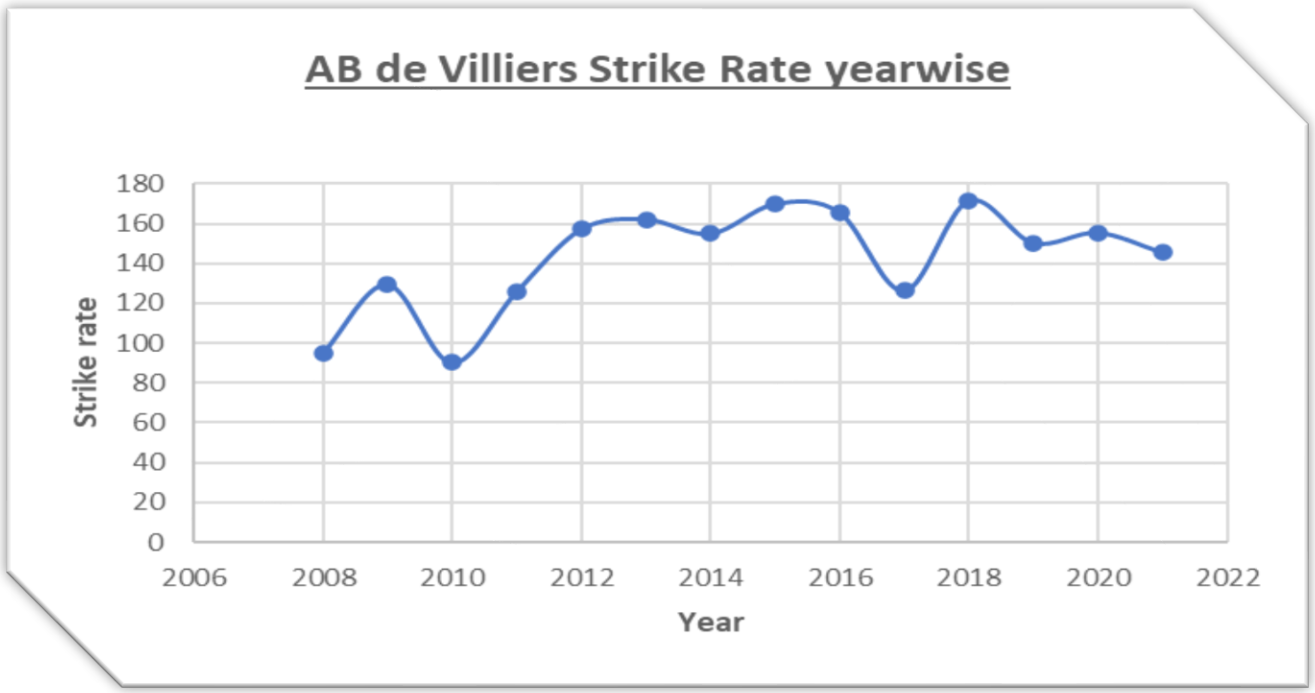
1) Virat Kohli :



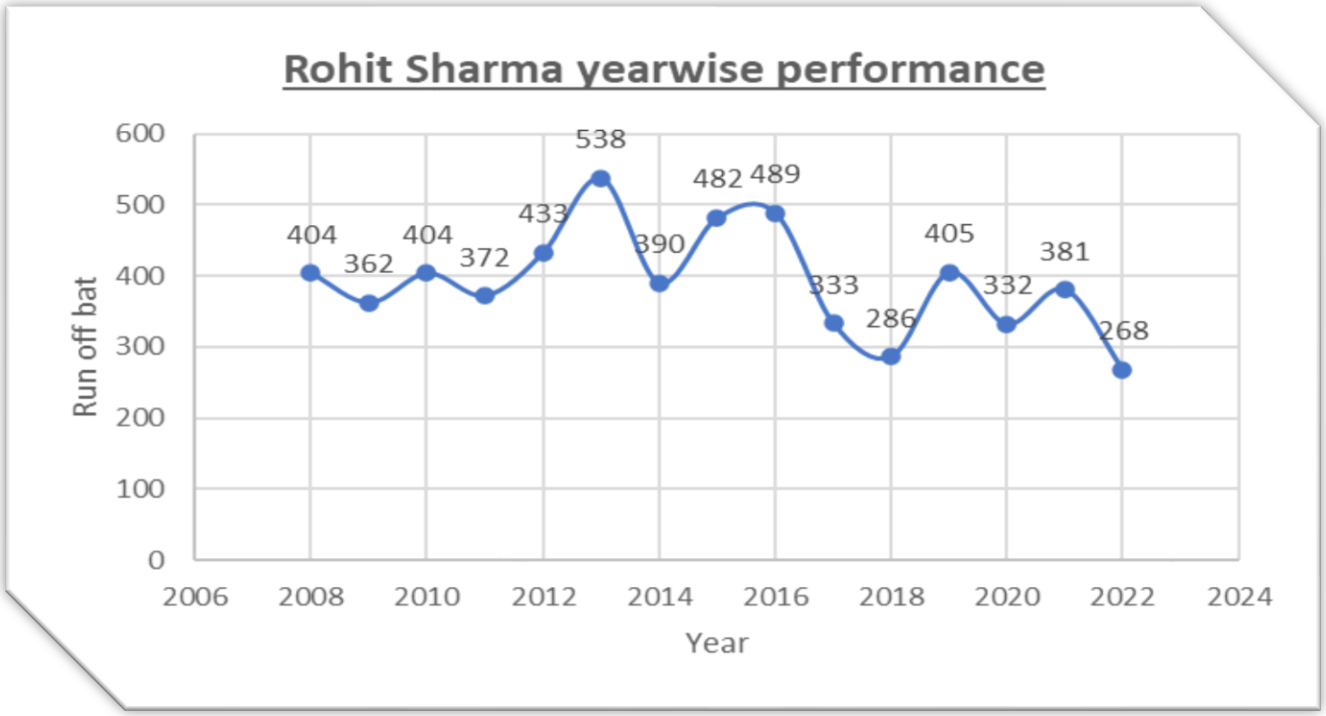


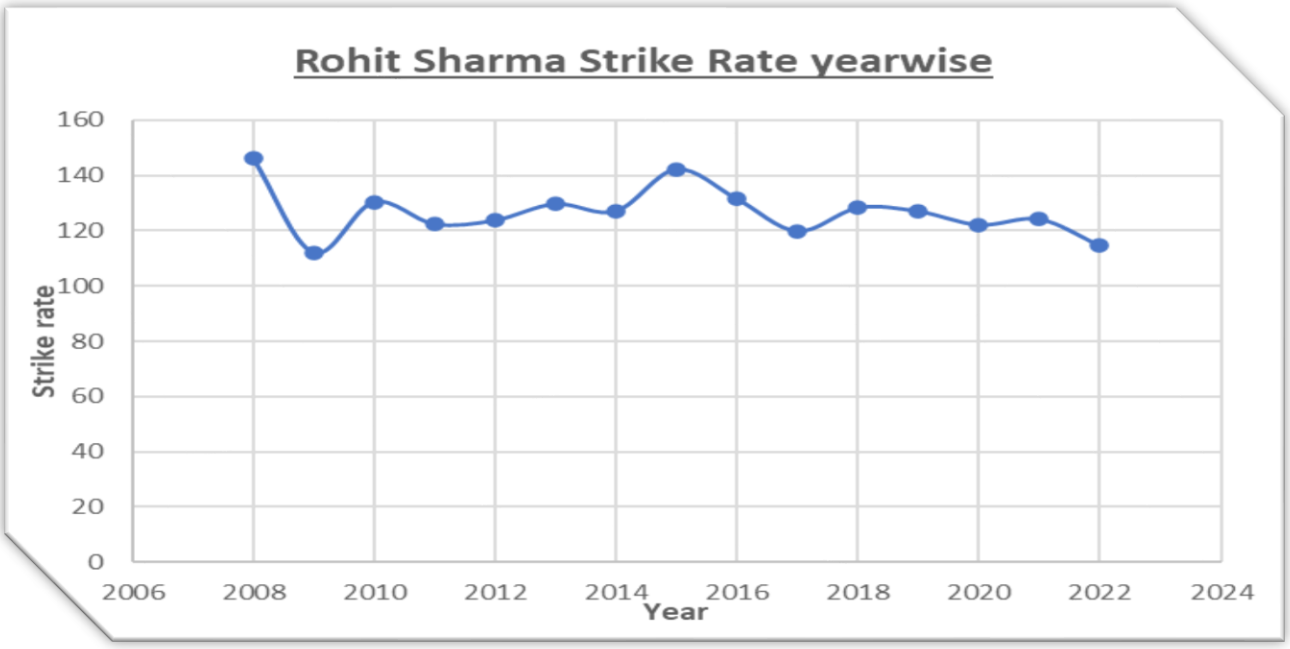
2) AB de Villiers :



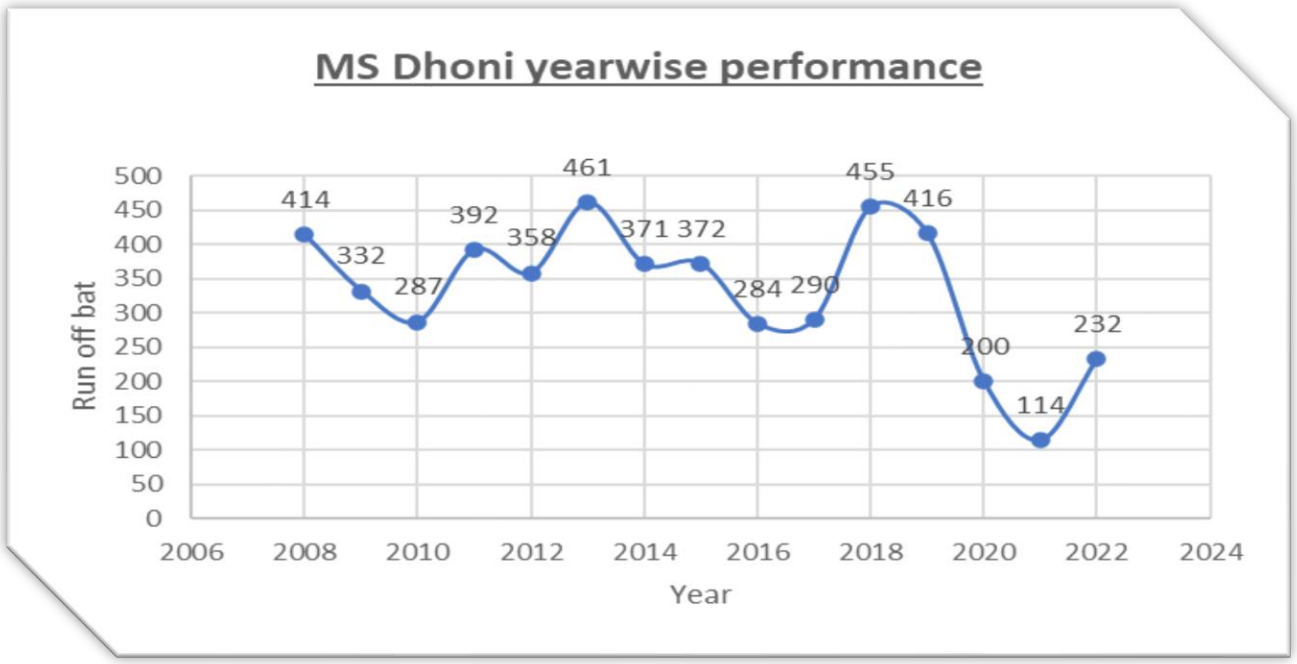


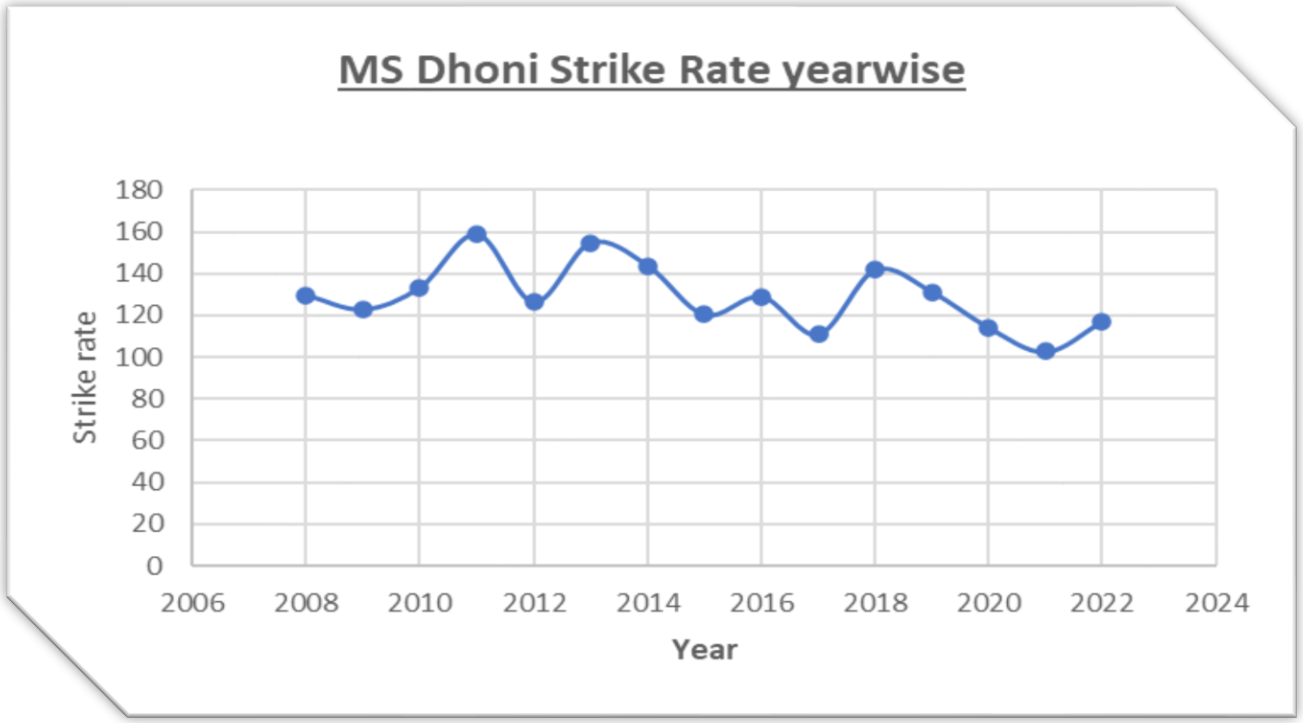
3) Rohit Sharma :



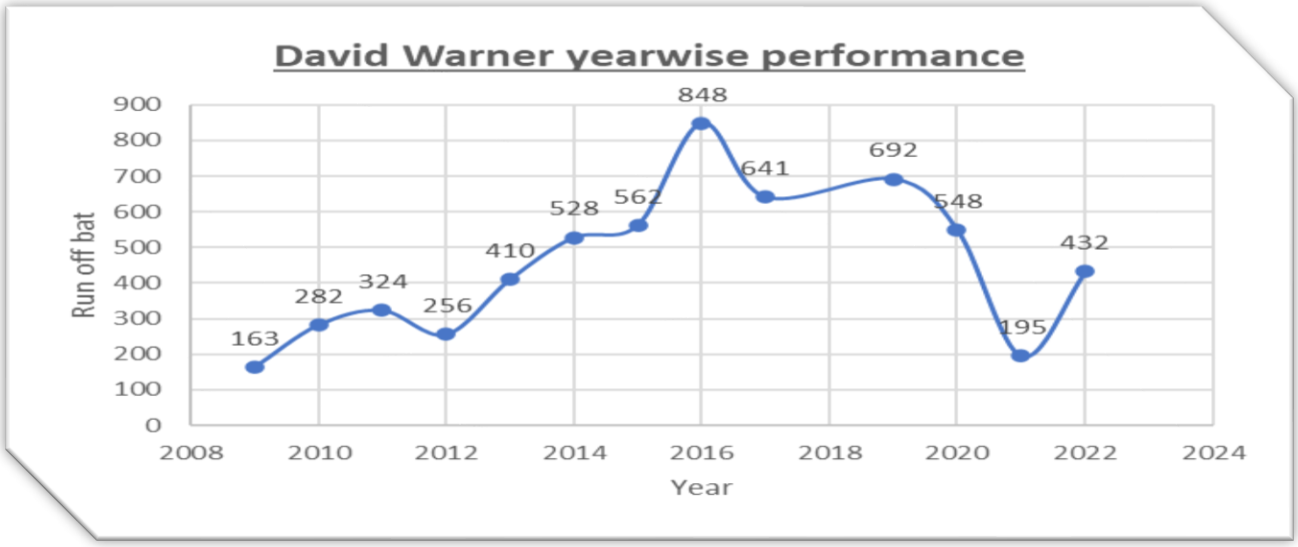


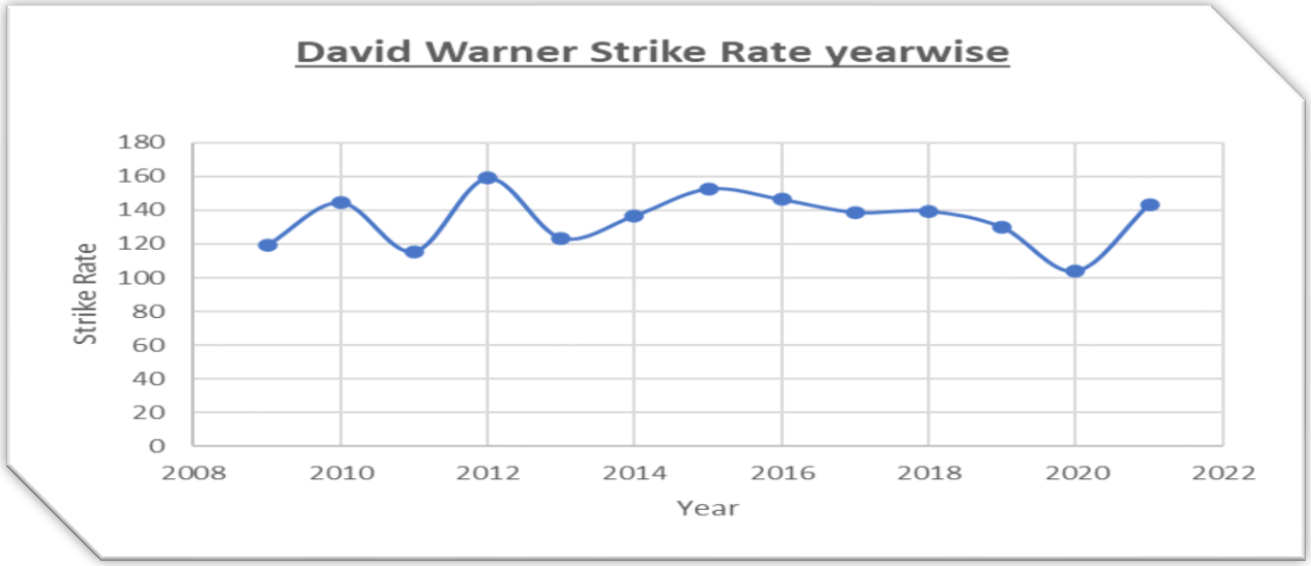
4) MS Dhoni ;


MS Dhoni yearwise performance
Run off bat
Year
500
450
400
350
300
250
200
150
100
50
0
2006
2008
2010
2012
2014
2016
2018
2020
2022
2024
414
332
287
392
358
461
371
372
284
290
455
416
200
114
232


MS Dhoni Strike Rate yearwise
Strike rate
Year
180
160
140
120
100
80
60
40
20
0
2006
2008
2010
2012
2014
2016
2018
2020
2022
2024

5) David Warner :



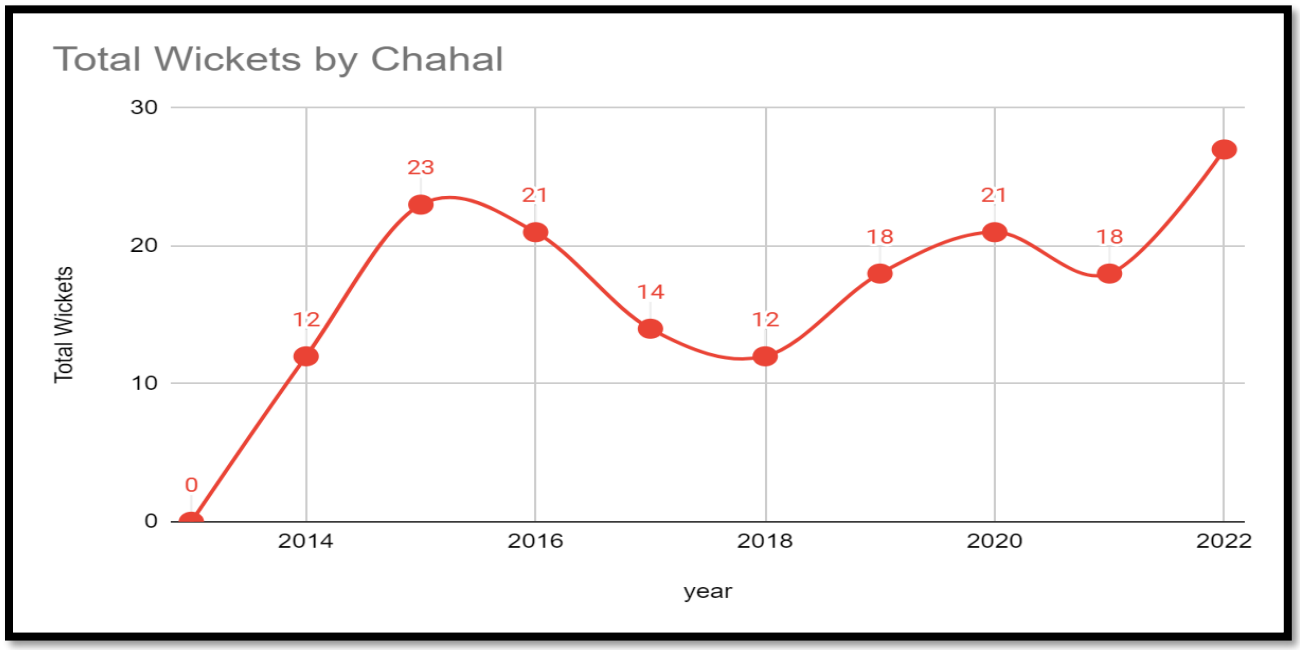


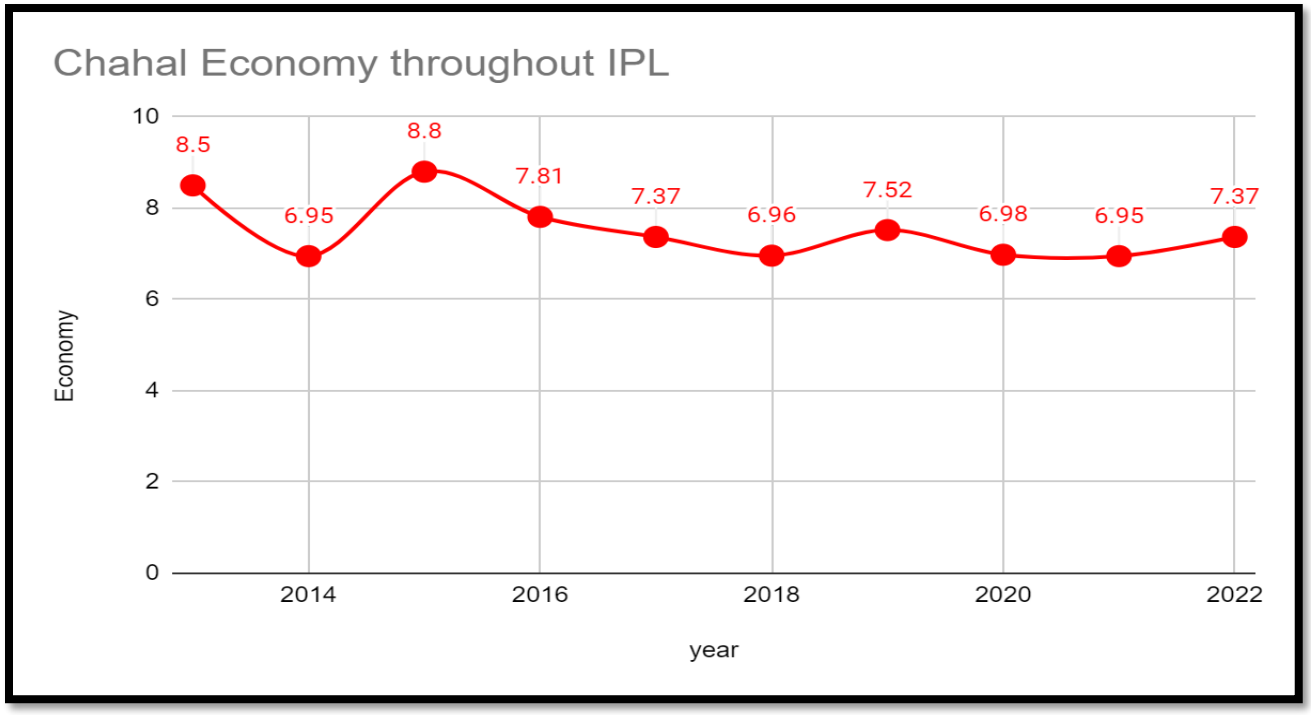
Year Wise Wickets of Bowler

```python
import pandas as pd

# read data into DataFrame
matches = pd.read_csv('/content/ipl_match_ball_by_ball_data_cleaned.csv')



# filter DataFrame to include only YS Chahal's performance
ysc_matches = matches.loc[matches['bowler'] == 'B Kumar']

# filter out wickets that are 'run out'
ysc_wickets = ysc_matches[ysc_matches['wicket_type'] != 'run out']

# group the filtered DataFrame by year and wicket type, and count the number of wickets for each type
yearly_wickets = ysc_wickets.groupby(['year', 'wicket_type'])['wicket_type'].count().reset_index(name='wickets')

# pivot the yearly_wickets DataFrame to create separate columns for each wicket type
yearly_wickets = yearly_wickets.pivot(index='year', columns='wicket_type', values='wickets').reset_index()

# fill NaN values with zero
yearly_wickets = yearly_wickets.fillna(0)

# print the resulting DataFrame
print(yearly_wickets)
```

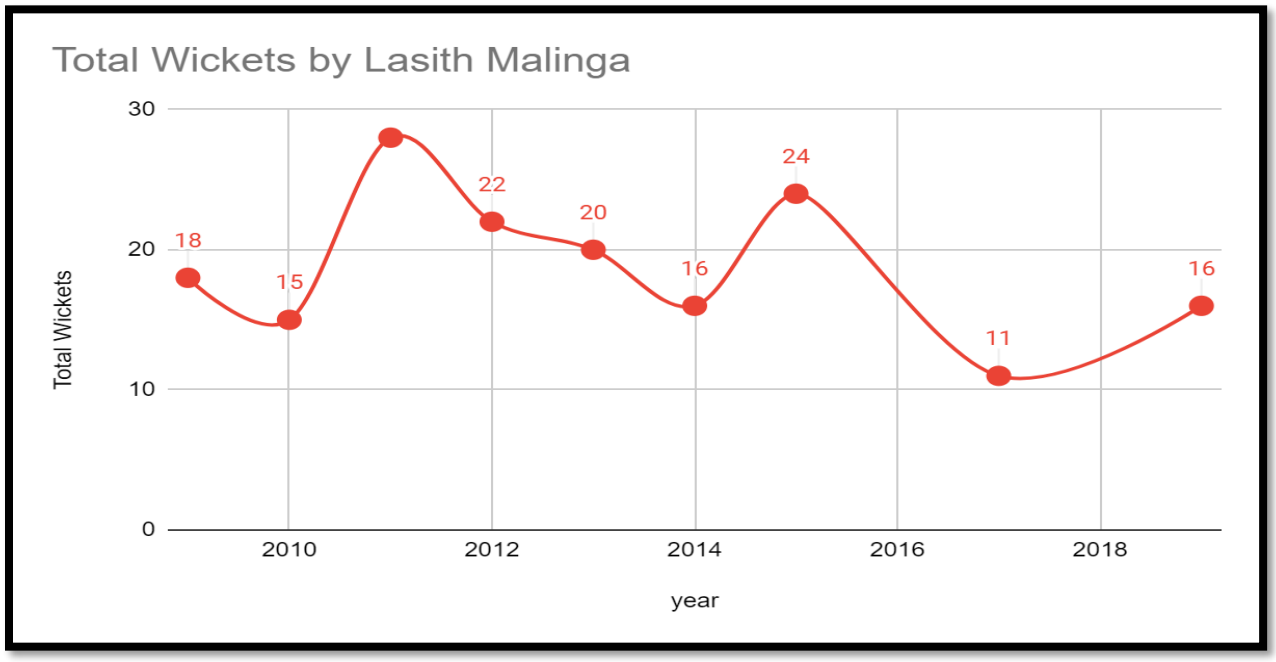
Year Wise Economy of Bowler

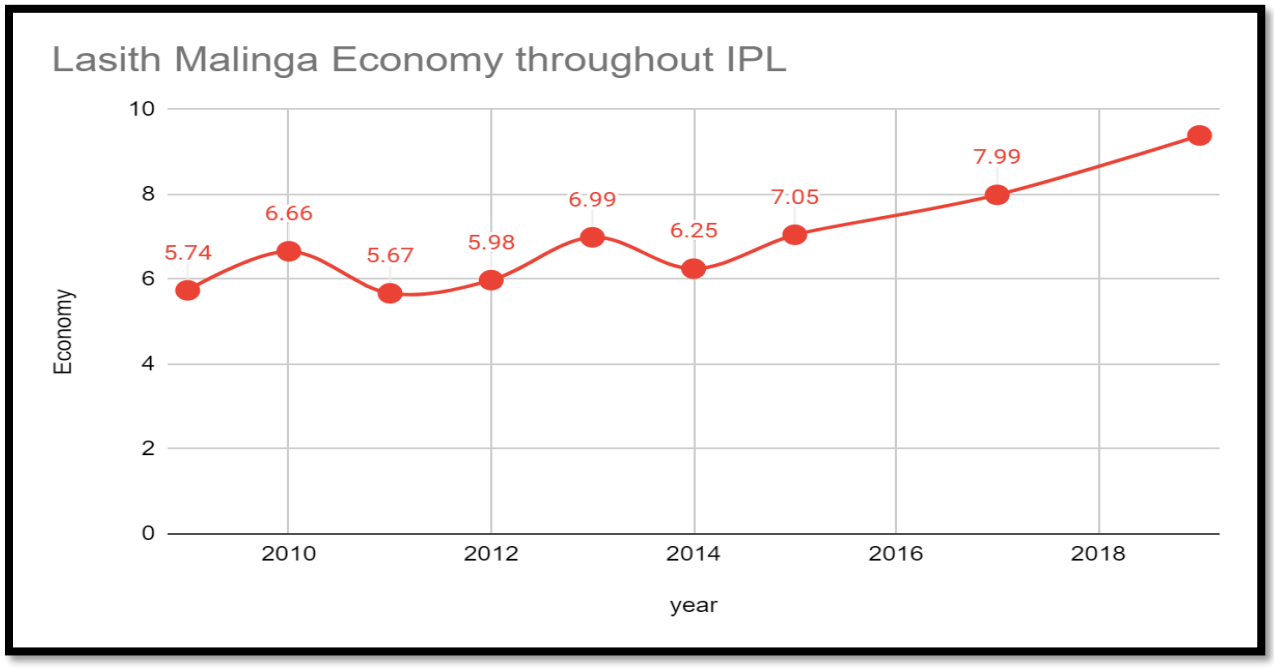
1) Yuzvendra Chahal:



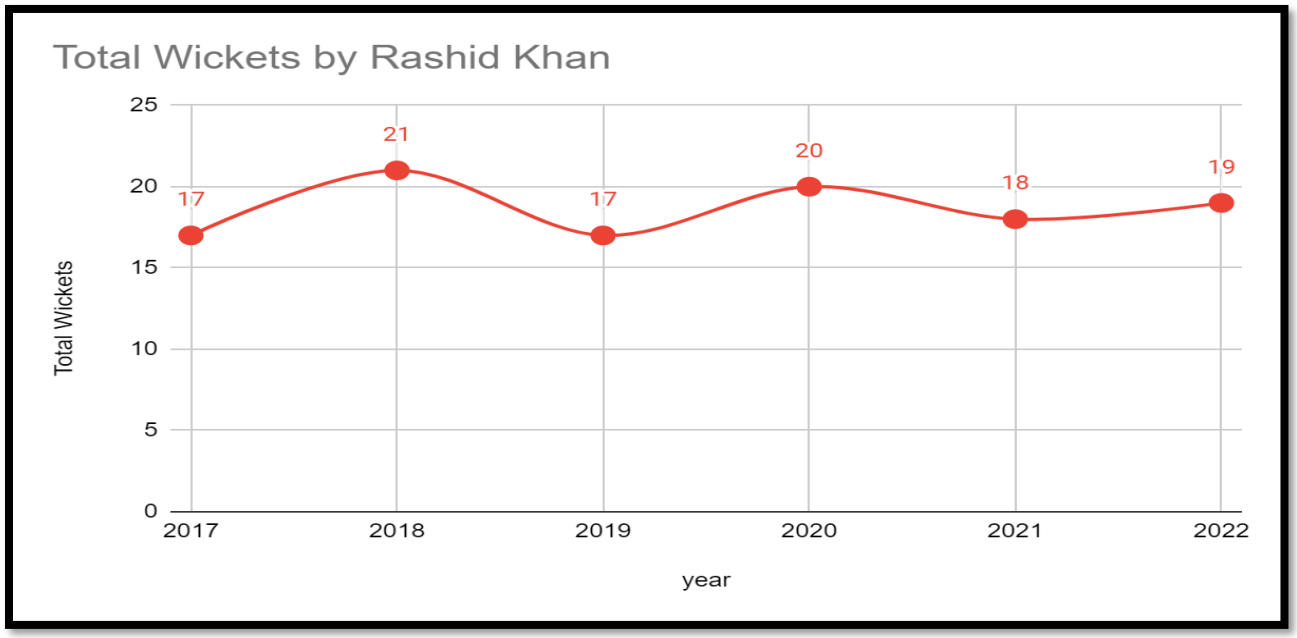


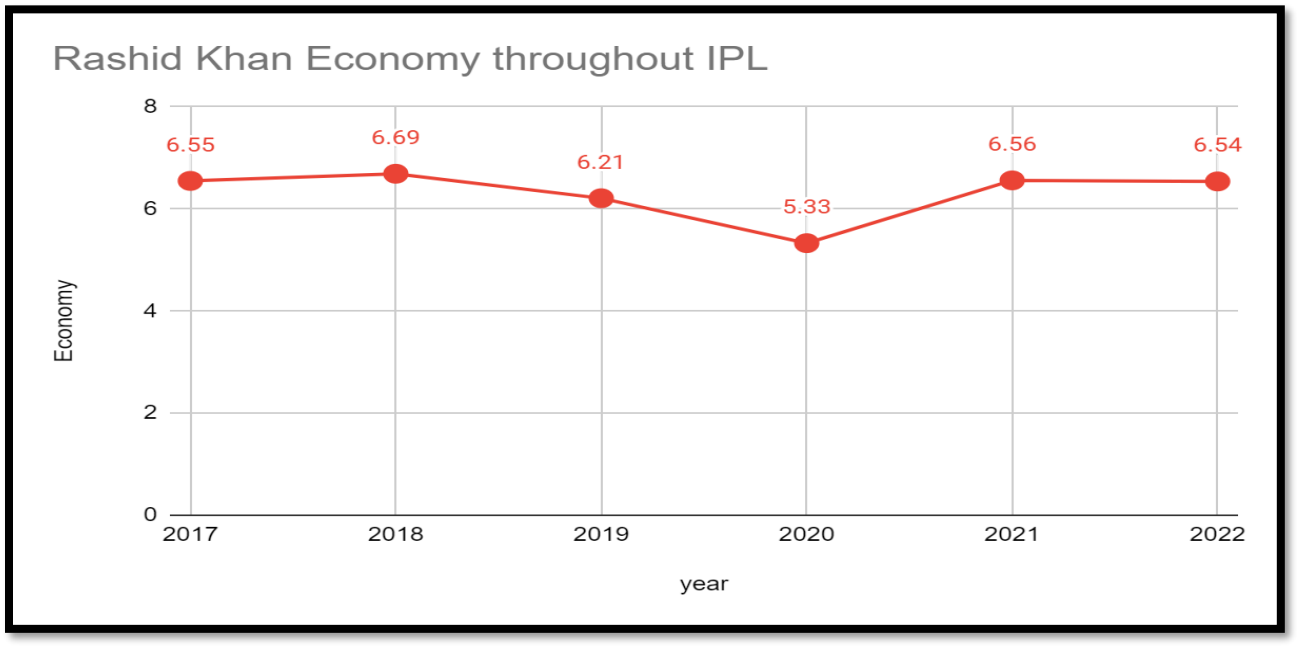
2) Lasith Malinga:



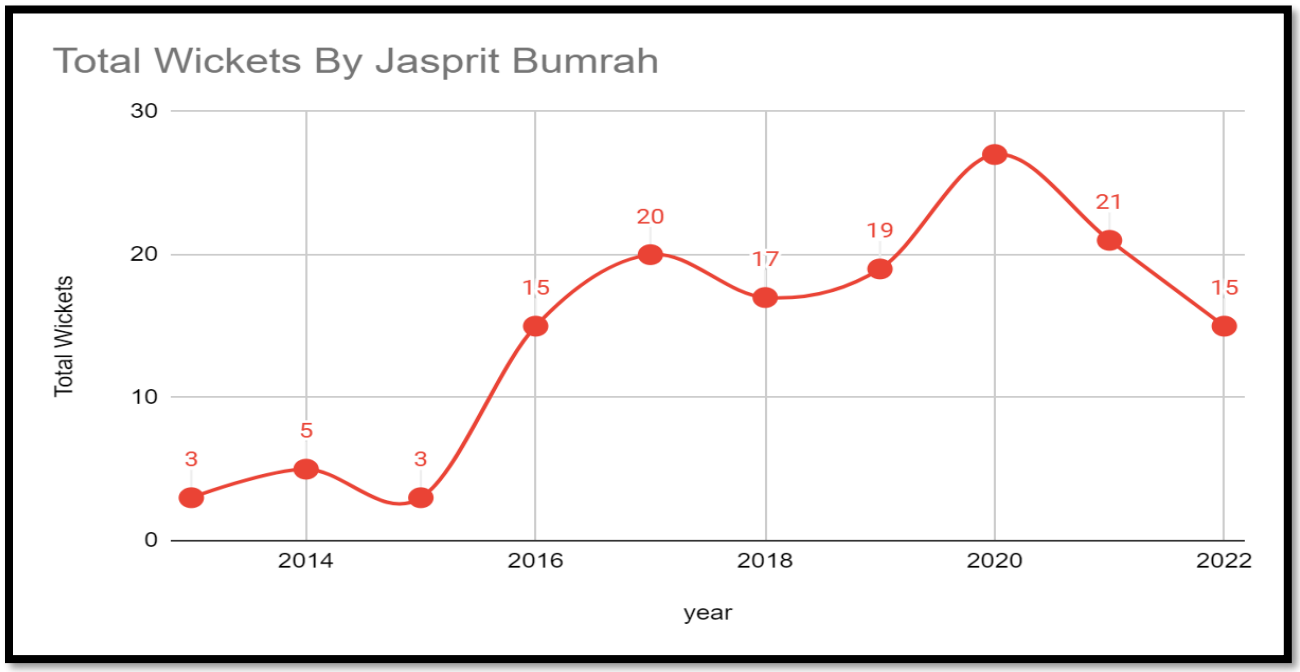


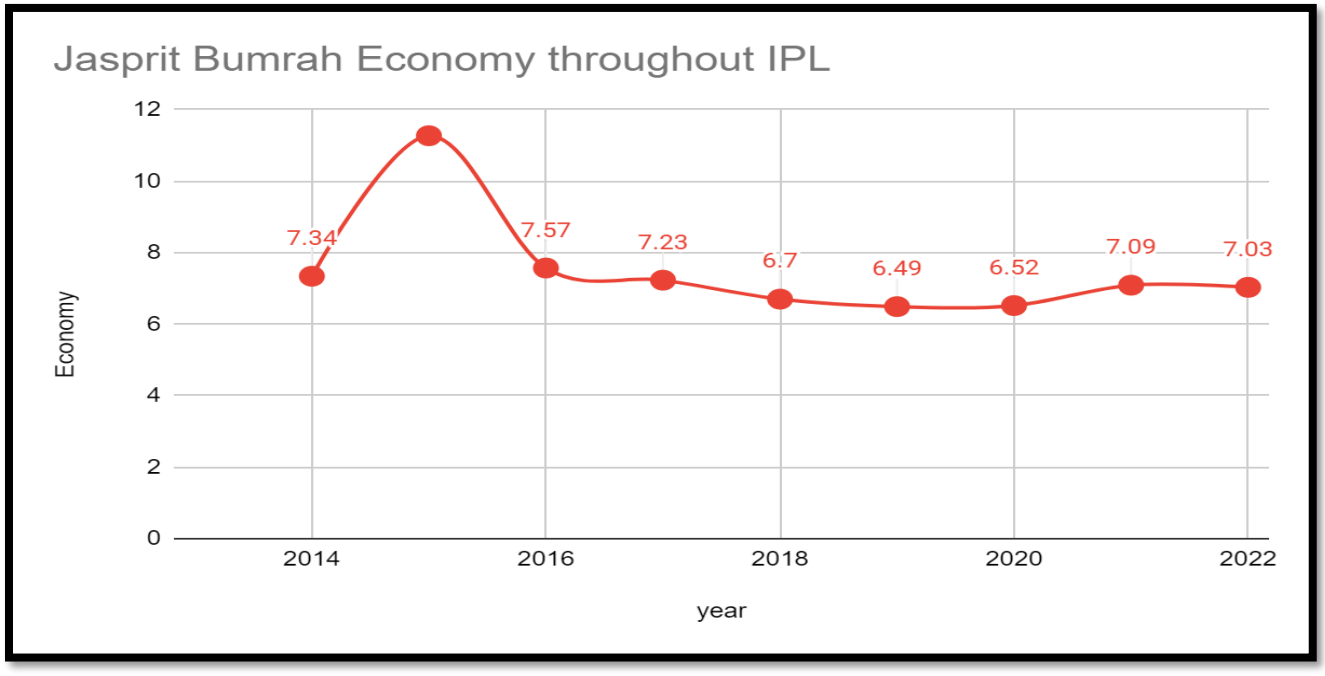
3) Rashid Khan :



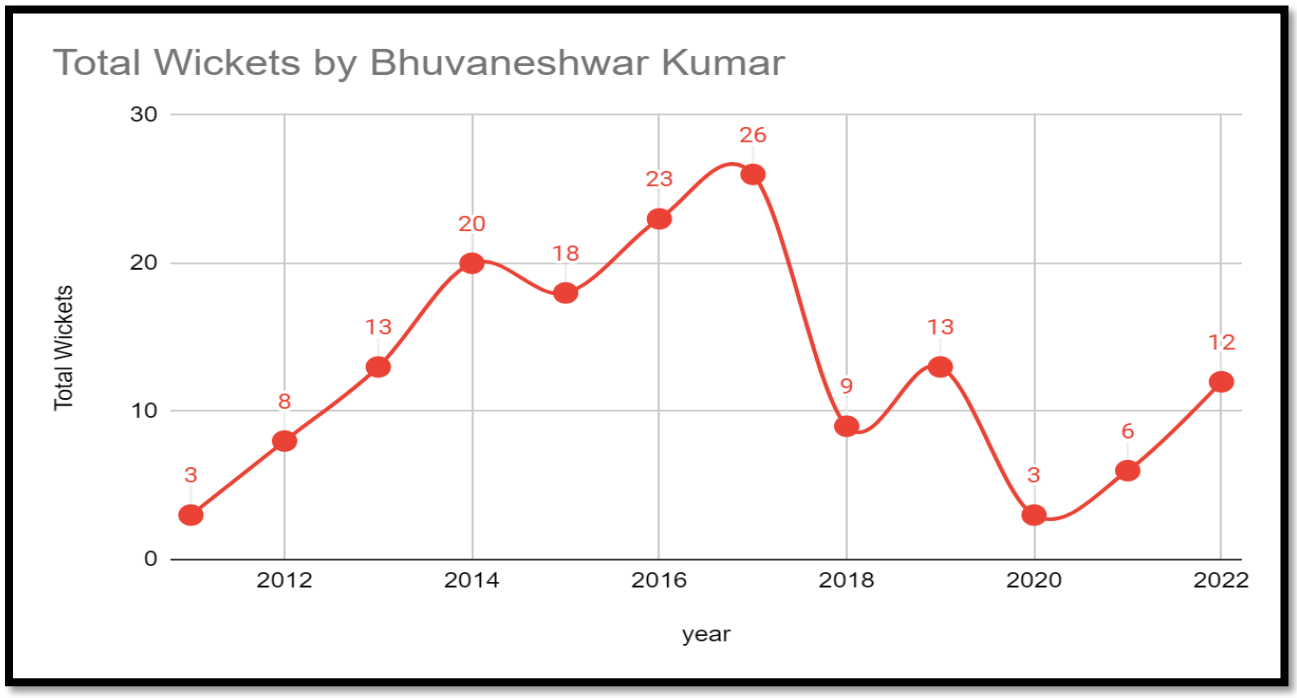


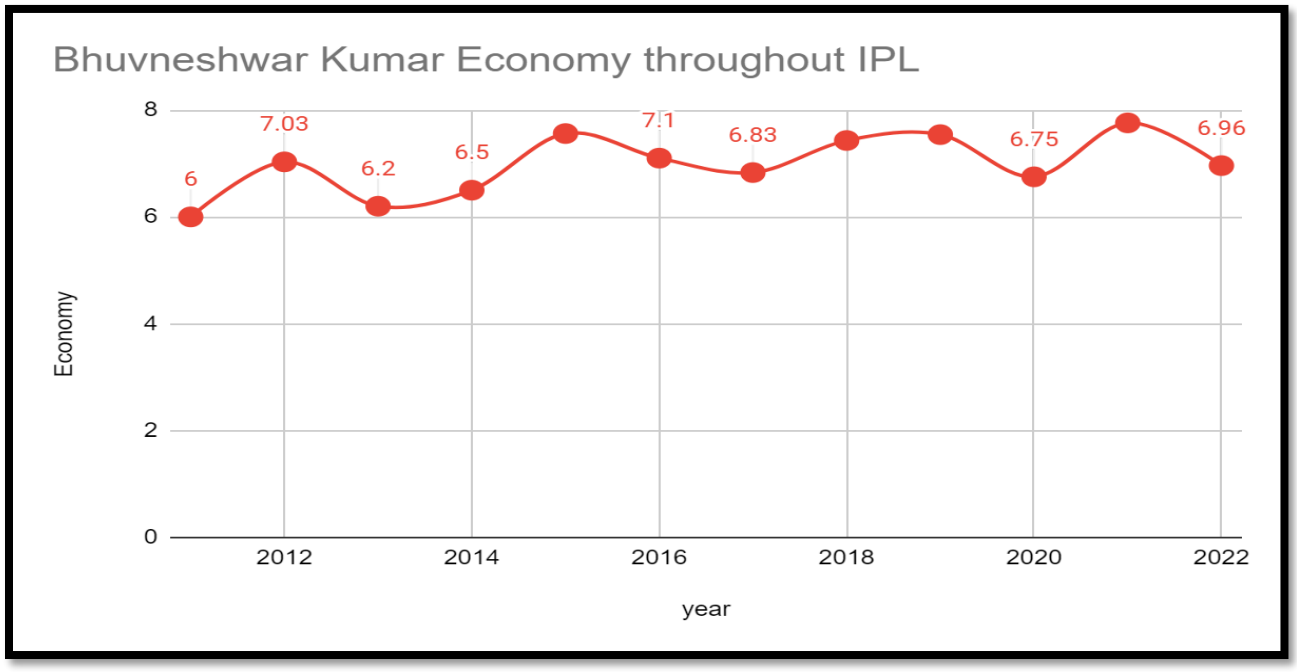
4) Jasprit Bumrah:





5) Bhuvneshwar Kumar:





PREPROCESSING OF DATA

Preprocessing of data refers to the series of steps and techniques applied to raw data to transform and prepare it for further analysis and modeling. It is an essential step in data analysis. Also, it helps us to improve data quality, remove inconsistencies, reduce noise, and extract relevant features. Here we have used Python for this.

```
import pandas as pd
import numpy as np
```

```
alldata = pd.read_csv("/kaggle/input/ball-by-ball-6/ipl_match_ball_by_ball_data_cleaned.csv")
alldata.columns
#Ball by ball data is stored in alldata
```

Ball-by-ball data is read and stored in alldata

```
Index(['match_no', 'match_id', 'season', 'date', 'month', 'year', 'venue',
       'innings', 'over', 'ball', 'bowling_team', 'striker', 'non_striker',
       'batting_team', 'bowler', 'runs_off_bat', 'extras', 'wides', 'noballs',
       'byes', 'legbyes', 'penalty', 'wicket_type', 'player_dismissed',
       'other_wicket_type', 'other_player_dismissed'],
      dtype='object')
```

These are the column names of ball-by-ball data

```
alldata["wicket_type"].value_counts()
#Value counts of method of wicket
```

```
caught                    6817
bowled                    1934
run out                    999
lbw                        683
stumped                    324
caught and bowled          323
hit wicket                  14
retired hurt                13
obstructing the field        2
retired out                  1
Name: wicket_type, dtype: int64
```

These are the value counts of how the batsman has got out in the whole IPL. For e.g. Total 6817 players have been caught out in the whole IPL.

```
allmatchdata = pd.read_csv("/kaggle/input/all-ipl-matches-3/All ipl matches - Cleaned Data.csv")
allmatchdata.columns
#All matches data is stored is allmatchdata
```

```
Index(['Match no.', 'Match ID', 'Date', 'Month', 'Year', 'Venue', 'Team1',
       'Team2', 'Team1 Player1', 'Team1 Player2', 'Team1 Player3',
       'Team1 Player4', 'Team1 Player5', 'Team1 Player6', 'Team1 Player7',
       'Team1 Player8', 'Team1 Player9', 'Team1 Player10', 'Team1 Player11',
       'Team2 Player1', 'Team2 Player2', 'Team2 Player3', 'Team2 Player4',
       'Team2 Player5', 'Team2 Player6', 'Team2 Player7', 'Team2 Player8',
       'Team2 Player9', 'Team2 Player10', 'Team2 Player11', 'Toss winner',
       'Toss decision', 'Stage', 'First innings score', 'First innings wicket',
       'Second innings score', 'Second innings wicket', 'Match winner',
       'Won by', 'Margin', 'Method', 'SuperOver', 'Player of the match',
       'POTM', 'Umpire1', 'Umpire2'],
      dtype='object')
```

All match's data is read and stored in "allmatchdata"

```
alldata.shape
```

```
(225794, 26)
```

```
n = int(input("Data till match no.?"))
#n takes the input till where the match data and ball by ball should be taken
```

```
data = alldata[alldata["match_no"]<=n] #This takes the ball by ball data till nth match
data5= data[data["year"]>=int(data.tail(1)["year"]-5)] #This takes the ball by ball data in the last 5 years of nth match
data
```

Ball-by-ball data has total of 2,25,794 rows.

"n" takes the input till where the match data and ball-by-ball should be taken.

"data" takes the ball-by-ball data till the nth match.

"data5" takes the ball-by-ball data in the last 5 years before nth match

	match_no	match_id	season	date	month	year	venue	innings	over	ball	...	extras	wides	noballs	byes	legbyes	penalty	wicket_type
0	1	335982	2007/08	18	4	2008	M. Chinnaswamy Stadium, Bangalore	1.0	0.0	1.0	...	1.0	NaN	NaN	NaN	1.0	NaN	NaN
1	1	335982	2007/08	18	4	2008	M. Chinnaswamy Stadium, Bangalore	1.0	0.0	2.0	...	0.0	NaN	NaN	NaN	NaN	NaN	NaN
2	1	335982	2007/08	18	4	2008	M. Chinnaswamy Stadium, Bangalore	1.0	0.0	3.0	...	1.0	1.0	NaN	NaN	NaN	NaN	NaN
3	1	335982	2007/08	18	4	2008	M. Chinnaswamy Stadium, Bangalore	1.0	0.0	4.0	...	0.0	NaN	NaN	NaN	NaN	NaN	NaN
4	1	335982	2007/08	18	4	2008	M. Chinnaswamy Stadium, Bangalore	1.0	0.0	5.0	...	0.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
225789	951	1312200	2022	29	5	2022	Narendra Modi Stadium, Ahmedabad	2.0	17.0	3.0	...	0.0	NaN	NaN	NaN	NaN	NaN	NaN
225790	951	1312200	2022	29	5	2022	Narendra Modi Stadium,	2.0	17.0	4.0	...	0.0	NaN	NaN	NaN	NaN	NaN	NaN

Here is how our "data" looks.

```
matchdata = allmatchdata[allmatchdata["Match no."]<=n] #This takes the all match data till nth match
matchdata5= matchdata[matchdata["Year"]>=int(matchdata.tail(1)["Year"]-5)] #This takes the all match data in the last 5 years of
nth match
matchdata
```

"matchdata" takes the match data till the nth match.

"matchdata5" takes the match data in the last 5 years before the nth match.

Match no.	Match ID	Date	Month	Year	Venue	Team1	Team2	Team1 Player1	Team1 Player2	...	Second innings wicket	Match winner	Won by	Margin	Method	SuperOve
1	1	18	4	2008	M. Chinnaswamy Stadium, Bangalore	Kolkata Knight Riders	Royal Challengers Bangalore	SC Ganguly	BB McCullum	...	10.0	Kolkata Knight Riders	Runs	140.0	NaN	N
2	2	19	4	2008	Punjab Cricket Association Stadium, Mohali	Chennai Super Kings	Punjab Kings	PA Patel	ML Hayden	...	4.0	Chennai Super Kings	Runs	33.0	NaN	N
3	3	19	4	2008	Arun Jaitley Stadium, Delhi	Rajasthan Royals	Delhi Capitals	T Kohli	YK Pathan	...	1.0	Delhi Capitals	Wickets	9.0	NaN	N
4	4	20	4	2008	Eden Gardens, Kolkata	Sunrisers Hyderabad	Kolkata Knight Riders	AC Gilchrist	Y Venugopal Rao	...	5.0	Kolkata Knight Riders	Wickets	5.0	NaN	N
5	5	20	4	2008	Wankhede Stadium, Mumbai	Mumbai Indians	Royal Challengers Bangalore	L Ronchi	ST Jayasuriya	...	5.0	Royal Challengers Bangalore	Wickets	5.0	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
947	70	22	5	2022	Wankhede Stadium, Mumbai	Sunrisers Hyderabad	Punjab Kings	PK Garg	Abhishek Sharma	...	5.0	Punjab Kings	Wickets	5.0	NaN	N
948	71	24	5	2022	Eden Gardens, Kolkata	Rajasthan Royals	Gujarat Titans	YBK Jaiswal	JC Buttler	...	3.0	Gujarat Titans	Wickets	7.0	NaN	N
949	72	25	5	2022	Eden Gardens,	Royal Challengers	Lucknow Super	V Kohli	F du Plessis	...	6.0	Royal Challengers	Runs	14.0	NaN	N

Here is how our "matchdata" looks

Now we want to get the scorecard of any given match.

Bowler wickets gives the list of ways bowler can get credited for a batsman being out.

```
bowler_wickets=["bowled","caught","caught and bowled","hit wicket","lbw","stumped"]
#ways a bowler can get the wicket of a batsman
```

```
def scorecard(m): #This function takes the match no. as parameter and gives the scorecard of that match as output
    match=alldata[(alldata["match_no"] == m)] #Finds the mth match's ball by ball data
    matchdata = allmatchdata[(allmatchdata["Match no."] == m)] #Finds the mth match's match data
    print(matchdata.iloc[0]["Team1"],"Vs",matchdata.iloc[0]["Team2"],"\n") #prints Team1 vs Team2
    print(matchdata.iloc[0]["Toss winner"],"won the toss and decided to",matchdata.iloc[0]["Toss decision"],"\n") #prints toss r
esult

    #First, we take the first innings data
    match_1=match[(match["innings"] == 1)] #First innings ball by ball data

    batsmen_1_str=match_1["striker"].unique() #All batsmen who have been at the strikers end
    batsmen_1_nonstr=match_1["non_striker"].unique() #All batsmen who have been at the non striker end
    batsmen_1=np.append(batsmen_1_str,batsmen_1_nonstr) #Adding both batsmen lists
    indexes=np.unique(batsmen_1,return_index=True)[1] #indexes of the list
    batsmen_1=[batsmen_1[index] for index in sorted(indexes)] #sorts the list in a manner where the batsman who batted first come
s first in the list
```

```
    batsmen_1_df=pd.DataFrame(columns = ["Batsman","Wicket","Runs","Balls","Fours","Sixes", "Strike Rate"],index=range(1,len(bat
smen_1)+1))
    #Creating dataframe with column names as above and index as 1,2,3,...

    batsmen_1_out=match_1[["wicket_type","player_dismissed"]].dropna() #All players who have been out and the way they got out
    out_batsmen_1 = batsmen_1_out["player_dismissed"].tolist() #List of all the batsmen who have been out
    for i in range(len(batsmen_1)):
        batsman = batsmen_1[i]
        match_1_batsman=match_1[(match_1["striker"] == batsman)] #getting ball by ball data for specific batsman
        runs=int(match_1_batsman["runs_off_bat"].sum()) #evaluating total runs made by the batsman
        balls=match_1_batsman.shape[0]-match_1_batsman[match_1_batsman["wides"].notnull()].shape[0] #evaluating total balls playe
d by the batsman
        fours=match_1_batsman[match_1_batsman["runs_off_bat"]==4].shape[0] #total fours hit by the batsman
        sixes=match_1_batsman[match_1_batsman["runs_off_bat"]==6].shape[0] #total sixes hit by the batsman
        if batsman in out_batsmen_1: #if batsman has been out
            out=batsmen_1_out[(batsmen_1_out["player_dismissed"] == batsman)]
            wicket_type = out.iloc[0]["wicket_type"] #finding how the batsman got out
        else:
            wicket_type = "not out" #else the batsman is not out
        str_rate=round(runs/balls*100,2) if balls!=0 else "-" #strike rate is runs/balls*100 if he has played some balls otherise
it is blank
```

```
        batsmen_1_df.loc[i+1]=[batsman,wicket_type,runs,balls,fours,sixes,str_rate] #appending batsman data to the dataframe
    print(batsmen_1_df,"\n") #printing batting scorecard for the first inning

    bowlers_1=match_1["bowler"].unique() #All players who have bowled
    bowlers_1_df=pd.DataFrame(columns= ["Bowler","Overs","Over no.","Maidens","Runs","Wickets","Economy","Dots","Fours","Sixe
s","Wides","No balls"],index=range(1,len(bowlers_1)+1))
    #Creatig dataframe with columns names as given and index as 1,2,3,...
    for i in range(len(bowlers_1)):
        bowler = bowlers_1[i]
        match_1_bowler=match_1[(match_1["bowler"] == bowler)] #getting ball by ball data for specific bowler
        runs= int(match_1_bowler["runs_off_bat"].sum()+match_1_bowler["wides"].sum()+match_1_bowler["noballs"].sum()) #total run
s given away by the bowler
        balls= match_1_bowler.shape[0]-match_1_bowler[match_1_bowler["wides"].notnull()].shape[0]-match_1_bowler[match_1_bowler
["noballs"].notnull()].shape[0] #total balls bowled by the bowler
        overs=balls//6+(balls%6)/10 #total overs bowled by the bowler, adds balls as decimal if the over is incomplete
        wickets=match_1_bowler[match_1_bowler["wicket_type"] == "bowled"].shape[0]+match_1_bowler[match_1_bowler["wicket_type"]
== "caught"].shape[0]+match_1_bowler[match_1_bowler["wicket_type"] == "caught and bowled"].shape[0]+match_1_bowler[match_1_bowle
r["wicket_type"] == "hit wicket"].shape[0]+match_1_bowler[match_1_bowler["wicket_type"] == "lbw"].shape[0]+match_1_bowler[match_
1_bowler["wicket_type"] == "stumped"].shape[0]
        #total wickets by the bowler
        eco=round(runs/balls*6,2) #economy of the bowler
        dots=match_1_bowler[(match_1_bowler["runs_off_bat"] == 0) & (match_1_bowler["wides"].isna()) & (match_1_bowler["noball
s"].isna())].shape[0] #total dots by the bowler
```

```
        fours=match_1_bowler[match_1_bowler["runs_off_bat"]==4].shape[0] #total fours hit to the bowler
        sixes=match_1_bowler[match_1_bowler["runs_off_bat"]==6].shape[0] #total sixes hit to the bowler
        wides=match_1_bowler[match_1_bowler["wides"].notnull()].shape[0] #total wides given the bowler
        noballs=match_1_bowler[match_1_bowler["noballs"].notnull()].shape[0] #total noballs given the bowler
        overs_no = match_1_bowler["over"].unique() #overs bowled by the bowler
        over_no = []
        for over in overs_no:
            ov = int(over)+1
            over_no.append(ov)
        maidens=0
        for over in overs_no:
            over_df=match_1_bowler[(match_1_bowler["over"] == over)] #getting ball by ball data for each over bowled by the bowle
r
            if int(over_df["runs_off_bat"].sum()+over_df["wides"].sum()+over_df["noballs"].sum()) == 0:
                maidens+=1

        bowlers_1_df.loc[i+1]=[bowler,overs,over_no,maidens,runs,wickets,eco,dots,fours,sixes,wides,noballs] #appending bowler d
ata to the dataframe
    print(bowlers_1_df,"\n") #print bowling scorecard for the first inning

    print("First innings score by ", matchdata.iloc[0]["Team1"], ": ", int(matchdata.iloc[0]["First innings score"]),"/",int(mat
chdata.iloc[0]["First innings wicket"]),"\n",sep="")
    #print first innings data
```

```
    #Same thing with second innings
    match_2=match[(match["innings"] == 2)]

    batsmen_2_str=match_2["striker"].unique()
    batsmen_2_nonstr=match_2["non_striker"].unique()
    batsmen_2=np.append(batsmen_2_str,batsmen_2_nonstr)
    indexes=np.unique(batsmen_2,return_index=True)[1]
    batsmen_2=[batsmen_2[index] for index in sorted(indexes)]
    batsmen_2_df=pd.DataFrame(columns = ["Batsman","Wicket","Runs","Balls","Fours","Sixes", "Strike Rate"],index=range(1,len(bat
smen_2)+1))
    batsmen_2_out=match_2[["wicket_type","player_dismissed"]].dropna()
    out_batsmen_2 = batsmen_2_out["player_dismissed"].tolist()
    for i in range(len(batsmen_2)):
        batsman = batsmen_2[i]
        match_2_batsman=match_2[(match_2["striker"] == batsman)]
        runs=int(match_2_batsman["runs_off_bat"].sum())
        balls=match_2_batsman.shape[0]-match_2_batsman[match_2_batsman["wides"].notnull()].shape[0]
        fours=match_2_batsman[match_2_batsman["runs_off_bat"]==4].shape[0]
        sixes=match_2_batsman[match_2_batsman["runs_off_bat"]==6].shape[0]
        if batsman in out_batsmen_2:
            out=batsmen_2_out[(batsmen_2_out["player_dismissed"] == batsman)]
            wicket_type = out.iloc[0]["wicket_type"]
```

```
        else:
            wicket_type = "not out"
        str_rate=round(runs/balls*100,2) if balls!=0 else "-"
        batsmen_2_df.loc[i+1]=[batsman,wicket_type,runs,balls,fours,sixes,str_rate]
    print(batsmen_2_df,"\n")

    bowlers_2=match_2["bowler"].unique()
    bowlers_2_df=pd.DataFrame(columns= ["Bowler","Overs","Over no.","Maidens","Runs","Wickets","Economy","Dots","Fours","Sixe
s","Wides","No balls"],index=range(1,len(bowlers_2)+1))
    for i in range(len(bowlers_2)):
        bowler = bowlers_2[i]
        match_2_bowler=match_2[(match_2["bowler"] == bowler)]
        runs= int(match_2_bowler["runs_off_bat"].sum()+match_2_bowler["wides"].sum()+match_2_bowler["noballs"].sum())
        balls= match_2_bowler.shape[0]-match_2_bowler[match_2_bowler["wides"].notnull()].shape[0]-match_2_bowler[match_2_bowler
["noballs"].notnull()].shape[0]
        overs=balls//6+(balls%6)/10
        wickets=match_2_bowler[match_2_bowler["wicket_type"] == "bowled"].shape[0]+match_2_bowler[match_2_bowler["wicket_type"]
== "caught"].shape[0]+match_2_bowler[match_2_bowler["wicket_type"] == "caught and bowled"].shape[0]+match_2_bowler[match_2_bowle
r["wicket_type"] == "hit wicket"].shape[0]+match_2_bowler[match_2_bowler["wicket_type"] == "lbw"].shape[0]+match_2_bowler[match_
2_bowler["wicket_type"] == "stumped"].shape[0]
        eco=round(runs/balls*6,2)
        dots=match_2_bowler[(match_2_bowler["runs_off_bat"] == 0) & (match_2_bowler["wides"].isna()) & (match_2_bowler["noball
s"].isna())].shape[0]
```

```
        fours=match_2_bowler[match_2_bowler["runs_off_bat"]==4].shape[0]
        sixes=match_2_bowler[match_2_bowler["runs_off_bat"]==6].shape[0]
        wides=match_2_bowler[match_2_bowler["wides"].notnull()].shape[0]
        noballs=match_2_bowler[match_2_bowler["noballs"].notnull()].shape[0]
        overs_no = match_2_bowler["over"].unique()
        over_no = []
        for over in overs_no:
            ov = int(over)+1
            over_no.append(ov)
        maidens=0
        for over in overs_no:
            over_df=match_2_bowler[(match_2_bowler["over"] == over)]
            if int(over_df["runs_off_bat"].sum()+over_df["wides"].sum()+over_df["noballs"].sum()) == 0:
                maidens+=1
        bowlers_2_df.loc[i+1]=[bowler,overs,over_no,maidens,runs,wickets,eco,dots,fours,sixes,wides,noballs]
    print(bowlers_2_df,"\n")

    print("Second innings score by ", matchdata.iloc[0]["Team2"], ": ", int(matchdata.iloc[0]["Second innings score"]),"/",int(m
atchdata.iloc[0]["Second innings wicket"]),"\n",sep="")

    #print Match Result
    print("Match Result:", matchdata.iloc[0]["Match winner"],"won by", int(matchdata.iloc[0]["Margin"]), matchdata.iloc[0]["Won
by"])
```

```
scorecard(354)
```

```
Royal Challengers Bangalore Vs Pune Warriors India

Pune Warriors India won the toss and decided to Field

          Batsman   Wicket Runs Balls Fours Sixes Strike Rate
1        CH Gayle  not out  175    66    13    17      265.15
2      TM Dilshan  not out   33    36     5     0       91.67
3         V Kohli  not out   11     9     0     1      122.22
4  AB de Villiers  not out   31     8     3     3       387.5
5       SS Tiwary  not out    2     2     0     0       100.0
6       R Rampaul  not out    0     1     0     0         0.0

       Bowler Overs          Over no. Maidens Runs Wickets Economy Dots Fours  \
1     B Kumar   4.0     [1, 3, 6, 18]       0   23       1    5.75   15     4
2   IC Pandey   2.0           [2, 11]       0   33       0    16.5    2     7
3    AB Dinda   4.0    [4, 9, 17, 20]       0   48       3    12.0    9     3
4    MR Marsh   3.0       [5, 13, 19]       0   56       1   18.67    4     2
5  AG Murtaza   2.0           [7, 15]       0   45       2    22.5    2     3
6    AJ Finch   1.0               [8]       0   29       0    29.0    0     1
7   LJ Wright   4.0  [10, 12, 14, 16]       0   26       1     6.5    8     1

      Sixes Wides No balls
1      0     1         0
2      0     0         1
3      4     1         1
4      7     1         0
5      5     1         0
6      4     0         0
7      1     2         0

First innings score by Royal Challengers Bangalore: 263/5

          Batsman   Wicket Runs Balls Fours Sixes Strike Rate
1      RV Uthappa  not out    0     2     0     0         0.0
2        AJ Finch  not out   18    15     4     0       120.0
3    Yuvraj Singh  not out   16    14     2     0      114.29
4       LJ Wright  not out    7     3     0     1      233.33
5       SPD Smith  not out   41    31     6     0      132.26
6        MR Marsh  not out   24    23     0     2      104.35
7        M Manhas  not out   11    13     1     0       84.62
8         B Kumar  not out    6    12     1     0        50.0
9      AG Murtaza  not out    5     4     1     0       125.0
10      IC Pandey  not out    0     2     0     0         0.0
11       AB Dinda  not out    1     1     0     0       100.0
```

```
          Bowler Overs         Over no. Maidens Runs Wickets Economy Dots  \
1       M Kartik   3.0       [1, 9, 11]       0   25       1    8.33    7
2       RP Singh   4.0   [2, 4, 16, 19]       0   20       1     5.0   14
3      R Rampaul   4.0   [3, 5, 14, 18]       0   21       0    5.25   14
4     JD Unadkat   4.0   [6, 8, 13, 15]       0   37       1    9.25    7
5  R Vinay Kumar   4.0  [7, 10, 12, 17]       0   24       0     6.0   10
6       CH Gayle   1.0             [20]       0    5       0     5.0    4

  Fours Sixes Wides No balls
1     4     0     2        0
2     3     0     0        0
3     1     1     1        0
4     3     2     0        0
5     3     0     0        0
6     1     0     0        0

Second innings score by Pune Warriors India: 133/9

Match Result: Royal Challengers Bangalore won by 130 Runs
```

First it tells us which teams are playing against each other.

Then it tells us the toss result and decision the toss winning team took.

After that, it shows us all the batsman and bowler stats and the aggregate score the batting team scored in the first innings.

Same with second innings stats.

Finally, it shows us the result and the margin of the match

```
#Batsman vs Bowler
```

```
def isbatsman(batsman): #function takes the player name as parameter and gives the output whether he can be classified as a batsma
n
    matches = matchdata[(matchdata["Team1 Player1"] == batsman) | (matchdata["Team1 Player2"] == batsman) | (matchdata["Team1 Pl
ayer3"] == batsman) | (matchdata["Team1 Player4"] == batsman) | (matchdata["Team1 Player5"] == batsman) | (matchdata["Team1 Play
er6"] == batsman) | (matchdata["Team1 Player7"] == batsman) | (matchdata["Team1 Player8"] == batsman) | (matchdata["Team1 Player
9"] == batsman) | (matchdata["Team1 Player10"] == batsman) | (matchdata["Team1 Player11"] == batsman) |
             (matchdata["Team2 Player1"] == batsman) | (matchdata["Team2 Player2"] == batsman) | (matchdata["Team2 Player3"] ==
batsman) | (matchdata["Team2 Player4"] == batsman) | (matchdata["Team2 Player5"] == batsman) | (matchdata["Team2 Player6"] == ba
tsman) | (matchdata["Team2 Player7"] == batsman) | (matchdata["Team2 Player8"] == batsman) | (matchdata["Team2 Player9"] == bats
man) | (matchdata["Team2 Player10"] == batsman) | (matchdata["Team2 Player11"] == batsman)].shape[0]
    innings = data[data["striker"] == batsman]["match_no"].unique().shape[0]
    if matches != 0 and innings/matches >= 0.7:
        return True
    else:
        return False
#The function returns true if the ratio of number of innings to the number of matches played by the batsman is greater than or equa
l to 70%,
#else returns false
```

The function "isbatsman" takes a player name as a parameter and returns true if the ratio of number of innings to the number of matches played by the batsman is greater than or equal to 70%. Otherwise, the function returns false. This is used to check if a player can be classified as a batsman or not.

```
def isbowler(bowler): #function takes the player name as parameter and gives the output whether he can be classified as a bowler
    matches = matchdata[(matchdata["Team1 Player1"] == bowler) | (matchdata["Team1 Player2"] == bowler) | (matchdata["Team1 Play
er3"] == bowler) | (matchdata["Team1 Player4"] == bowler) | (matchdata["Team1 Player5"] == bowler) | (matchdata["Team1 Player6"]
== bowler) | (matchdata["Team1 Player7"] == bowler) | (matchdata["Team1 Player8"] == bowler) | (matchdata["Team1 Player9"] == bo
wler) | (matchdata["Team1 Player10"] == bowler) | (matchdata["Team1 Player11"] == bowler) |
             (matchdata["Team2 Player1"] == bowler) | (matchdata["Team2 Player2"] == bowler) | (matchdata["Team2 Player3"] == b
owler) | (matchdata["Team2 Player4"] == bowler) | (matchdata["Team2 Player5"] == bowler) | (matchdata["Team2 Player6"] == bowle
r) | (matchdata["Team2 Player7"] == bowler) | (matchdata["Team2 Player8"] == bowler) | (matchdata["Team2 Player9"] == bowler) |
(matchdata["Team2 Player10"] == bowler) | (matchdata["Team2 Player11"] == bowler)].shape[0]
    overs = data[data["bowler"] == bowler][["match_no","over"]].drop_duplicates().shape[0]
    if matches != 0 and overs/matches >= 2:
        return True
    else:
        return False
#The function returns true if the average number of overs bowled by the over per match is greater than 2, else returns false
```

The function "isbowler" takes a player name as a parameter and returns true if the average number of overs bowled by the baller per match is

greater than 2. Otherwise, the function returns false. This is used to check whether a player can be classified as a bowler or not.

```
def batvsball(batsman, bowler): #returns record for specific batsman vs specific bowler
    if isbatsman(batsman) and isbowler(bowler):
        batball = data[(data["striker"] == batsman) & (data["bowler"] == bowler)]
        if batball.shape[0] != 0:
            runs = int(batball["runs_off_bat"].sum()) #total runs hit by the batsman to the bowler
            balls_played = batball.shape[0]-batball[batball["wides"].notnull()].shape[0] #total balls played by the batsman from
the bowler
            wickets = batball[(batball["striker"] == batball["player_dismissed"]) & batball["wicket_type"]==].shape[0] #total nu
mber of times batsman has got out to the bowler
            return [runs, balls_played, wickets] #returns list of runs, balls played, wickets
        else:
            return None
    else:
        return None
```

```
#Batsman and bowler form
```

This function "batvsball" returns the record for a specific batsman Vs a specific bowler. It returns the list of runs, balls played, and the number of times the batsman has got out to the bowler.

```
def inseason(L): #returns if all the games were played in the same season
    if all(1<=i<=58 for i in L):
        return True
    elif all(59<=i<=115 for i in L):
        return True
    elif all(116<=i<=175 for i in L):
        return True
    elif all(176<=i<=248 for i in L):
        return True
    elif all(249<=i<=322 for i in L):
        return True
    elif all(323<=i<=398 for i in L):
        return True
    elif all(399<=i<=458 for i in L):
        return True
    elif all(459<=i<=517 for i in L):
        return True
    elif all(518<=i<=577 for i in L):
        return True
    elif all(578<=i<=636 for i in L):
        return True
```

The function "inseason" returns if all the games in a list were played in the same season.

```
    elif all(637<=i<=696 for i in L):
        return True
    elif all(697<=i<=756 for i in L):
        return True
    elif all(757<=i<=816 for i in L):
        return True
    elif all(817<=i<=876 for i in L):
        return True
    elif all(877<=i<=950 for i in L):
        return True
    else:
        return False
```

```
def whichseason(L): #returns which season the games where played
    if all(1<=i<=58 for i in L):
        return 2008
    elif all(59<=i<=115 for i in L):
        return 2009
    elif all(116<=i<=175 for i in L):
        return 2010
    elif all(176<=i<=248 for i in L):
        return 2011
```

```
    elif all(249<=i<=322 for i in L):
        return 2012
    elif all(323<=i<=398 for i in L):
        return 2013
    elif all(399<=i<=458 for i in L):
        return 2014
    elif all(459<=i<=517 for i in L):
        return 2015
    elif all(518<=i<=577 for i in L):
        return 2016
    elif all(578<=i<=636 for i in L):
        return 2017
    elif all(637<=i<=696 for i in L):
        return 2018
    elif all(697<=i<=756 for i in L):
        return 2019
    elif all(757<=i<=816 for i in L):
        return 2020
    elif all(817<=i<=876 for i in L):
        return 2021
    elif all(877<=i<=950 for i in L):
        return 2022
    else:
        return False
```

The function "whichseason" returns if all the games in a list were played in the same season.

```
def last3_bat(batsman):
    if isbatsman(batsman) and data[data["striker"] == batsman]["match_no"].unique().shape[0] >= 3:
        last3_matches = data[data["striker"] == batsman]["match_no"].unique()[-3:]
        if inseason(last3_matches) and whichseason(last3_matches) == int(allmatchdata.iloc[n]["Year"]):
            batdata = data[(data["match_no"] == last3_matches[0]) & (data["striker"] == batsman)]
            for i in range(1,len(last3_matches)):
                batdata = batdata.append(data[(data["match_no"] == last3_matches[i]) & (data["striker"] == batsman)])
            runs = int(batdata["runs_off_bat"].sum()) #total runs score by the batsman in last 3 matches
            balls = batdata.shape[0] - batdata[batdata["wides"].notnull()].shape[0] #total balls played by the batsman in the las
t 3 matches
            wickets = batdata[batdata["player_dismissed"] == batsman].shape[0] #total times the batsman got out in last 3 matches
            return [runs, balls, wickets] #returns list of runs, balls played, wickets
        else:
            return None
    else:
        return None
```

The function last3_bat takes a batsman name as a parameter and returns the number of runs the batsman has scored, the number of balls he has played and the number of times the batsman has got out in the last 3 games.

```
def last3_bowl(bowler):
    if isbowler(bowler) and data[data["bowler"] == bowler]["match_no"].unique().shape[0] >= 3:
        last3_matches = data[data["bowler"] == bowler]["match_no"].unique()[-3:]
        if inseason(last3_matches) and whichseason(last3_matches) == int(allmatchdata.iloc[n]["Year"]):
            bowldata = data[(data["match_no"] == last3_matches[0]) & (data["bowler"] == bowler)]
            for i in range(1,len(last3_matches)):
                bowldata = bowldata.append(data[(data["match_no"] == last3_matches[i]) & (data["bowler"] == bowler)])
            runs = int(bowldata["runs_off_bat"].sum() + bowldata["extras"].sum()) #runs given by the bowler in last 3 matches
            balls = bowldata.shape[0] - bowldata[bowldata["wides"].notnull()].shape[0] - bowldata[bowldata["noballs"].notnull
()].shape[0] #balls bowled by the
            wickets = bowldata[bowldata["player_dismissed"] == bowler].shape[0]
            return [runs, balls, wickets] #returns list of runs, balls played, wickets
        else:
            return None
    else:
        return None
```

The function last3_bowl takes a bowler name as a parameter and returns the number of runs the bowler has given, the number of balls he has bowled and the number of wickets he has taken in the last 3 games.

```
#Team wins and losses at that stadium in last 5 years
```

```
def winlossst(team, stadium): #returns record of specific team at a specific stadium
    mst = matchdata5[((matchdata5["Team1"] == team) | (matchdata5["Team2"] == team)) & (matchdata5["Venue"] == stadium)]
    #match data of the team at the stadium
    if mst.shape[0] != 0:
        wins = mst[mst["Match winner"] == team].shape[0] #total wins of the team at the stadium
        losses = mst[mst["Match winner"] != team].shape[0] #total losses of the team at the stadium
        return [wins, losses] #returns list of wins and losses
    else:
        return None
```

The function "winlossst" takes the team name and stadium name as parameters and returns the number of matches the team has won and lost at that stadium in the last 5 years.

```
#Win ratio of Team A over Team B in last 5 years
```

```
def win_ratio(A,B): #returns win ratio in percentage of team A vs team B
  selected_rows1 = matchdata5.loc[(matchdata5['Team1'] == A) & (matchdata5['Team2'] == B), ['Team1', 'Team2',"Match winner"]]
  selected_rows2 = matchdata5.loc[(matchdata5['Team2'] == A) & (matchdata5['Team1'] == B), ['Team1', 'Team2',"Match winner"]]
  if selected_rows1.shape[0] == 0:
    if selected_rows2.shape[0] == 0:
      return None
    else:
      if len(selected_rows2["Match winner"].dropna().unique()) == 1:
        if selected_rows2["Match winner"].dropna().unique()[0] == A:
          return 100
        else:
          return 0
      else:
        return round(selected_rows2["Match winner"].value_counts()[A]*100/selected_rows2.shape[0],2)
  elif selected_rows2.shape[0] == 0:
    if len(selected_rows1["Match winner"].dropna().unique()) == 1:
        if selected_rows1["Match winner"].dropna().unique()[0] == A:
          return 100
        else:
          return 0
```

The function "win_ratio" takes the names of two teams, A and B, and returns the win ratio of Team A vs Team B in the last 5 years.

```
    else:
      return round(selected_rows1["Match winner"].value_counts()[A]*100/selected_rows1.shape[0],2)
  else:
    selected_rows=pd.concat([selected_rows1,selected_rows2])
    if len(selected_rows["Match winner"].dropna().unique()) == 1:
        if selected_rows["Match winner"].dropna().unique()[0] == A:
          return 100
        else:
          return 0
    else:
      return round(selected_rows["Match winner"].value_counts()[A]*100/selected_rows.shape[0],2)
```

```
#Chase vs Defend wins at that stadium in last 5 years
```

```
def chasedefend(stadium): #returns win record of matches which happened at the stadium
    std_data = matchdata5[matchdata5["Venue"] == stadium] #returns matches which happened at the stadium
    runs = std_data[std_data["Won by"] == "Runs"].shape[0]
    wickets = std_data[std_data["Won by"] == "Wickets"].shape[0]
    superover = std_data[std_data["Won by"] == "Super Over"].shape[0]
    return [runs,wickets] #returns list of total number of games won by defending and total number of games won by chasing
```

The function "chasedefend" takes the stadium name as parameter and returns the number of games the defending team and the chasing team has won at that stadium.

```
#Momentum
```

```
def momentum(team): #returns win and loss record of last 3 games of a specific team
    if matchdata[(matchdata["Team1"] == team) | (matchdata["Team2"] == team)]["Match no."].unique().shape[0] >= 4:
        last4_matches = matchdata[(matchdata["Team1"] == team) | (matchdata["Team2"] == team)]["Match no."].unique()[-4:]
        if inseason(last4_matches) and whichseason(last4_matches) == int(allmatchdata.iloc[n]["Year"]):
            mntm = []
            for i in last4_matches:
                match_data = matchdata[(matchdata["Match no."] == i)]
                if match_data[match_data["Match winner"] == team].shape[0] == 1:
                    mntm.append(1)
                else:
                    mntm.append(0)
            return mntm #returns a list of 1s and 0s wher 1 dnote a win and 0 denotes a loss
        else:
            return None
```

The function "momentum" takes the team name as its parameter and returns the win records of the last 4 games the team has played. It returns

a list of 1s and 0s, where 1 if the team won the match and 0 if the team lost the match.

For e.g., if RCB has won their fourth last match, lost their third last match, lost their second last match, and won its last match then "momentum(Royal Challengers Bangalore)" will return [1,0,0,1].

```
match=allmatchdata[allmatchdata["Match no."]==n+1]
match
```

	Match no.	Match ID	Date	Month	Year	Venue	Team1	Team2	Team1 Player1	Team1 Player2	...	Second innings wicket	Match winner	Won by	Margin	Method	SuperOver	Player of the match
453	454	55	25	5	2014	Punjab Cricket Association Stadium, Mohali	Delhi Capitals	Punjab Kings	KP Pietersen	MA Agarwal	...	3.0	Punjab Kings	Wickets	7.0	NaN	N	Manan Vohra

1 rows × 46 columns

```
t1p1 = allmatchdata.loc[n]["Team1 Player1"]
t2p1 = match.loc[n]["Team2 Player1"]
#batvsball(t1p1,t2p1)
t1p1
```

```
'KP Pietersen'
```

```
allmatchdata.iloc[475]
```

```
Match no.                                476
Match ID                                  17
Date                                      20
Month                                      4
Year                                    2015
Venue                  Arun Jaitley Stadium, Delhi
Team1                            Delhi Capitals
Team2                    Kolkata Knight Riders
Team1 Player1                       MA Agarwal
Team1 Player2                          SS Iyer
Team1 Player3                       JP Duminy
Team1 Player4                       MK Tiwary
Team1 Player5                     Yuvraj Singh
Team1 Player6                       AD Mathews
Team1 Player7                        KM Jadhav
Team1 Player8                  NM Coulter-Nile
Team1 Player9                          A Mishra
Team1 Player10                     Imran Tahir
Team1 Player11                   DJ Muthuswami
Team2 Player1                       RV Uthappa
Team2 Player2                        G Gambhir
Team2 Player3                       MK Pandey
Team2 Player4                        SA Yadav
Team2 Player5                       YK Pathan
Team2 Player6                 RN ten Doeschate
Team2 Player7                       AD Russell
Team2 Player8                       PP Chawla
Team2 Player9                       SP Narine
Team2 Player10                       M Morkel
Team2 Player11                      UT Yadav
Toss winner              Kolkata Knight Riders
Toss decision                           Field
Stage                                   Group
First innings score                     146.0
First innings wicket                      8.0
Second innings score                    147.0
Second innings wicket                     4.0
Match winner             Kolkata Knight Riders
Won by                                Wickets
Margin                                    6.0
Method                                    NaN
SuperOver                                   N
Player of the match                 Umesh Yadav
POTM                                   UT Yadav
Umpire1                                 SD Fry
Umpire2                              CB Gaffaney
Name: 475, dtype: object
```

```
final_data=pd.DataFrame()

#batvsball
for i in range(1,12):
    for j in range(1,12):
        final_data[f"T1P{i} vs T2P{j} runs"] = batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"])[0] if batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) else None
        final_data[f"T1P{i} vs T2P{j} balls"] = batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"])[1] if batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) else None
        final_data[f"T1P{i} vs T2P{j} wickets"] = batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"])[2] if batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) else None
#columns with batting record of Team1 vs Team2

for i in range(1,12):
    for j in range(1,12):
        final_data[f"T2P{i} vs T1P{j} runs"] = batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"])[0] if batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) else None
        final_data[f"T2P{i} vs T1P{j} balls"] = batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"])[1] if batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) else None
        final_data[f"T2P{i} vs T1P{j} wickets"] = batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"])[2] if batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) else None
#columns with batting record of Team2 vs Team1
```

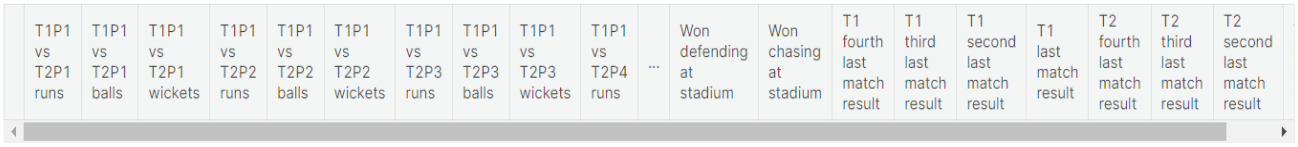
```
#batform
for i in range(1,12):
    final_data[f"T1P{i} last3_bat runs"] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"])[0] if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
    final_data[f"T1P{i} last3_bat balls"] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"])[1] if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
    final_data[f"T1P{i} last3_bat wickets"] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"])[2] if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
#columns with batting form of Team1

for i in range(1,12):
    final_data[f"T2P{i} last3_bat runs"] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"])[0] if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
    final_data[f"T2P{i} last3_bat balls"] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"])[1] if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
    final_data[f"T2P{i} last3_bat wickets"] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"])[2] if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
#columns with batting form of Team2
```

```
#bowlform
for i in range(1,12):
    final_data[f"T1P{i} last3_bowl runs"] = last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"])[0] if last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
    final_data[f"T1P{i} last3_bowl balls"] = last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"])[1] if last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
    final_data[f"T1P{i} last3_bowl wickets"] = last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"])[2] if last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None
#columns with bowling form of Team1

for i in range(1,12):
    final_data[f"T2P{i} last3_bowl runs"] = last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"])[0] if last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None
    final_data[f"T2P{i} last3_bowl balls"] = last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"])[1] if last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None
    final_data[f"T2P{i} last3_bowl wickets"] = last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"])[2] if last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None
#columns with bowling form of Team2
```

```
#win ratio at stadium
final_data["T1 wins at stadium"] = winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"])[0] if winlossst(allma
tchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"]) else None
final_data["T1 losses at stadium"] = winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"])[1] if winlossst(all
matchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"]) else None
#columns with wins and losses of Team1 at the stadium

final_data["T2 wins at stadium"] = winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"])[0] if winlossst(allma
tchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"]) else None
final_data["T2 losses at stadium"] = winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"])[1] if winlossst(all
matchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"]) else None
#columns with wins and losses of Team2 at the stadium



#win ratio
final_data["T1 vs T2 win ratio"] = win_ratio(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) if win_ratio(allmatch
data.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) else None
#column with win ratio of Team1 vs Team2

final_data["T2 vs T1 win ratio"] = win_ratio(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) if win_ratio(allmatch
data.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) else None
#column with win ratio of Team2 vs Team1
```

```
#chasedefend
final_data["Won defending at stadium"] = chasedefend(allmatchdata.iloc[n]["Venue"])[0] if chasedefend(allmatchdata.iloc[n]["Venu
e"]) else None
#column with total number of times defending team has won at the stadium

final_data["Won chasing at stadium"] = chasedefend(allmatchdata.iloc[n]["Venue"])[1] if chasedefend(allmatchdata.iloc[n]["Venu
e"]) else None
#column with total number of times chasing team has won at the stadium



#team momentum
final_data["T1 fourth last match result"] = momentum(allmatchdata.iloc[n]["Team1"])[0] if momentum(allmatchdata.iloc[n]["Team
1"]) else None
final_data["T1 third last match result"] = momentum(allmatchdata.iloc[n]["Team1"])[1] if momentum(allmatchdata.iloc[n]["Team1"])
else None
final_data["T1 second last match result"] = momentum(allmatchdata.iloc[n]["Team1"])[2] if momentum(allmatchdata.iloc[n]["Team
1"]) else None
final_data["T1 last match result"] = momentum(allmatchdata.iloc[n]["Team1"])[3] if momentum(allmatchdata.iloc[n]["Team1"]) else
None
#win loss record of Team1 in last 4 matches
```

```
final_data["T2 fourth last match result"] = momentum(allmatchdata.iloc[n]["Team2"])[0] if momentum(allmatchdata.iloc[n]["Team
2"]) else None
final_data["T2 third last match result"] = momentum(allmatchdata.iloc[n]["Team2"])[1] if momentum(allmatchdata.iloc[n]["Team2"])
else None
final_data["T2 second last match result"] = momentum(allmatchdata.iloc[n]["Team2"])[2] if momentum(allmatchdata.iloc[n]["Team
2"]) else None
final_data["T2 last match result"] = momentum(allmatchdata.iloc[n]["Team2"])[3] if momentum(allmatchdata.iloc[n]["Team2"]) else
None
#win loss record of Team2 in last 4 matches



final_data
#Creates a dataframe with given column names
```



T1P1 vs T2P1 runs	T1P1 vs T2P1 balls	T1P1 vs T2P1 wickets	T1P1 vs T2P2 runs	T1P1 vs T2P2 balls	T1P1 vs T2P2 wickets	T1P1 vs T2P3 runs	T1P1 vs T2P3 balls	T1P1 vs T2P3 wickets	T1P1 vs T2P4 runs	...	Won defending at stadium	Won chasing at stadium	T1 fourth last match result	T1 third last match result	T1 second last match result	T1 last match result	T2 fourth last match result	T2 third last match result	T2 second last match result

0 rows × 874 columns

All this code is for creating a data frame with the functions we defined above as column names.

```
#bat vs ball data
for i in range(1,12):
    for j in range(1,12):
        final_data.loc[0,final_data.columns[3*((i-1)*11+(j-1)):3*((i-1)*11+j)]] = batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) if batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) else None

for i in range(1,12):
    for j in range(1,12):
        final_data.loc[0,final_data.columns[363+3*((i-1)*11+(j-1)):363+3*((i-1)*11+j)]] = batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) if batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) else None
```

```
#bat form
for i in range(1,12):
    final_data.loc[0,final_data.columns[726+3*(i-1):726+3*i]] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None

for i in range(1,12):
    final_data.loc[0,final_data.columns[759+3*(i-1):759+3*i]] = last3_bat(allmatchdata.iloc[n][f"Team2 Player{i}"]) if last3_bat(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None
```

```
#bowl form
for i in range(1,12):
    final_data.loc[0,final_data.columns[792+3*(i-1):792+3*i]] = last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) if last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None

for i in range(1,12):
    final_data.loc[0,final_data.columns[825+3*(i-1):825+3*i]] = last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) if last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None
```

```
#wins and losses at that stadium
final_data.loc[0,final_data.columns[858:860]] = winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"]) if winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"]) else None

final_data.loc[0,final_data.columns[860:862]] = winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"]) if winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"]) else None
```

```
#winratio
final_data.loc[0,final_data.columns[862]] = win_ratio(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) if win_ratio
(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) else None
final_data.loc[0,final_data.columns[863]] = win_ratio(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) if win_ratio
(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) else None
```

```
#chase defend wins at that stadium
final_data.loc[0,final_data.columns[864:866]] = chasedefend(allmatchdata.iloc[n]["Venue"]) if chasedefend(allmatchdata.iloc[n]
["Venue"]) else None
```

```
#team momentum
final_data.loc[0, final_data.columns[866:870]] = momentum(allmatchdata.iloc[n]["Team1"]) if momentum(allmatchdata.iloc[n]["Team
1"]) else None

final_data.loc[0, final_data.columns[870:874]] = momentum(allmatchdata.iloc[n]["Team2"]) if momentum(allmatchdata.iloc[n]["Team
2"]) else None
```

```
col = final_data.columns
cols = col.to_list()
cols.insert(0,"Before match")
train_data = pd.DataFrame(columns = cols)
test_data = pd.DataFrame(columns = cols)
test_data
```

Before match	T1P1 vs T2P1 runs	T1P1 vs T2P1 balls	T1P1 vs T2P1 wickets	T1P1 vs T2P2 runs	T1P1 vs T2P2 balls	T1P1 vs T2P2 wickets	T1P1 vs T2P3 runs	T1P1 vs T2P3 balls	T1P1 vs T2P3 wickets	...	Won defending at stadium	Won chasing at stadium	T1 fourth last match result	T1 third last match result	T1 second last match result	T1 last match result	T2 fourth last match result	T2 third last match result	T2 second last match result

0 rows × 875 columns

```
for n in range(323, 758):
    data = alldata[alldata["match_no"]<=n]
    data5= data[data["year"]>=int(data.tail(1)["year"]-5)]

    matchdata = allmatchdata[allmatchdata["Match no."]<=n]
    matchdata5= matchdata[matchdata["Year"]>=int(matchdata.tail(1)["Year"]-5)]

    train_data.loc[n-323,train_data.columns[0]] = n+1

    #bat vs ball data
    for i in range(1,12):
        for j in range(1,12):
            train_data.loc[n-323,train_data.columns[3*((i-1)*11+(j-1))+1:3*((i-1)*11+j)+1]] = batvsball(allmatchdata.iloc[n][f"T
eam1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) if batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.
iloc[n][f"Team2 Player{j}"]) else None

    for i in range(1,12):
        for j in range(1,12):
            train_data.loc[n-323,train_data.columns[363+3*((i-1)*11+(j-1))+1:363+3*((i-1)*11+j)+1]] = batvsball(allmatchdata.ilo
c[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) if batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allma
tchdata.iloc[n][f"Team1 Player{j}"]) else None
```

```
    #bat form
    for i in range(1,12):
        train_data.loc[n-323,train_data.columns[727+3*(i-1):727+3*i]] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) if l
ast3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None

    for i in range(1,12):
        train_data.loc[n-323,train_data.columns[760+3*(i-1):760+3*i]] = last3_bat(allmatchdata.iloc[n][f"Team2 Player{i}"]) if l
ast3_bat(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None


    #bowl form
    for i in range(1,12):
        train_data.loc[n-323,train_data.columns[793+3*(i-1):793+3*i]] = last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) if
last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None

    for i in range(1,12):
        train_data.loc[n-323,train_data.columns[826+3*(i-1):826+3*i]] = last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) if
last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None
```

```
    #wins and losses at that stadium
    train_data.loc[n-323,train_data.columns[859:861]] = winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"])
if winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"]) else None

    train_data.loc[n-323,train_data.columns[861:863]] = winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"])
if winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"]) else None


    #winratio
    train_data.loc[n-323,train_data.columns[863]] = win_ratio(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) if w
in_ratio(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) else None
    train_data.loc[n-323,train_data.columns[864]] = win_ratio(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) if w
in_ratio(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) else None


    #chase defend wins at that stadium
    train_data.loc[n-323,train_data.columns[865:867]] = chasedefend(allmatchdata.iloc[n]["Venue"]) if chasedefend(allmatchdata.i
loc[n]["Venue"]) else None

    #team momentum
    train_data.loc[n-323, train_data.columns[867:871]] = momentum(allmatchdata.iloc[n]["Team1"]) if momentum(allmatchdata.iloc
[n]["Team1"]) else None

    train_data.loc[n-323, train_data.columns[871:875]] = momentum(allmatchdata.iloc[n]["Team2"]) if momentum(allmatchdata.iloc
[n]["Team2"]) else None
```

This code is for creating the train data using the feature extraction methods we implemented above.

```
train_data
```

```
train_data.to_csv("train_data.csv")
```

```
match_result=[]
for n in range(323,758):
    if allmatchdata.iloc[n]["Match winner"] == allmatchdata.iloc[n]["Team1"]:
        match_result.append("Team1")
    elif allmatchdata.iloc[n]["Match winner"] == allmatchdata.iloc[n]["Team2"]:
        match_result.append("Team2")
    else:
        match_result.append(None)
train_data["match result"] = match_result
```

```
for n in range(757, 951):
    data = alldata[alldata["match_no"]<=n]
    data5= data[data["year"]>=int(data.tail(1)["year"]-5)]

    matchdata = allmatchdata[allmatchdata["Match no."]<=n]
    matchdata5= matchdata[matchdata["Year"]>=int(matchdata.tail(1)["Year"]-5)]
```

This code is for creating the test data using the feature extraction methods we implemented above.

```
    test_data.loc[n-757,test_data.columns[0]] = n+1

    #bat vs ball data
    for i in range(1,12):
        for j in range(1,12):
            test_data.loc[n-757,test_data.columns[3*((i-1)*11+(j-1))+1:3*((i-1)*11+j)+1]] = batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) if batvsball(allmatchdata.iloc[n][f"Team1 Player{i}"], allmatchdata.iloc[n][f"Team2 Player{j}"]) else None

    for i in range(1,12):
        for j in range(1,12):
            test_data.loc[n-757,test_data.columns[363+3*((i-1)*11+(j-1))+1:363+3*((i-1)*11+j)+1]] = batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) if batvsball(allmatchdata.iloc[n][f"Team2 Player{i}"], allmatchdata.iloc[n][f"Team1 Player{j}"]) else None


    #bat form
    for i in range(1,12):
        test_data.loc[n-757,test_data.columns[727+3*(i-1):727+3*i]] = last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) if last3_bat(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None

    for i in range(1,12):
        test_data.loc[n-757,test_data.columns[760+3*(i-1):760+3*i]] = last3_bat(allmatchdata.iloc[n][f"Team2 Player{i}"]) if last3_bat(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None
```

```
    #bowl form
    for i in range(1,12):
        test_data.loc[n-757,test_data.columns[793+3*(i-1):793+3*i]] = last3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) if la
st3_bowl(allmatchdata.iloc[n][f"Team1 Player{i}"]) else None

    for i in range(1,12):
        test_data.loc[n-757,test_data.columns[826+3*(i-1):826+3*i]] = last3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) if la
st3_bowl(allmatchdata.iloc[n][f"Team2 Player{i}"]) else None


    #wins and losses at that stadium
    test_data.loc[n-757,test_data.columns[859:861]] = winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"]) if
winlossst(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Venue"]) else None

    test_data.loc[n-757,test_data.columns[861:863]] = winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"]) if
winlossst(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Venue"]) else None


    #winratio
    test_data.loc[n-757,test_data.columns[863]] = win_ratio(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) if win
_ratio(allmatchdata.iloc[n]["Team1"], allmatchdata.iloc[n]["Team2"]) else None
    test_data.loc[n-757,test_data.columns[864]] = win_ratio(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) if win
_ratio(allmatchdata.iloc[n]["Team2"], allmatchdata.iloc[n]["Team1"]) else None
```

```
    #chase defend wins at that stadium
    test_data.loc[n-757,test_data.columns[865:867]] = chasedefend(allmatchdata.iloc[n]["Venue"]) if chasedefend(allmatchdata.ilo
c[n]["Venue"]) else None

    #team momentum
    test_data.loc[n-757, test_data.columns[867:871]] = momentum(allmatchdata.iloc[n]["Team1"]) if momentum(allmatchdata.iloc[n]
["Team1"]) else None

    test_data.loc[n-757, test_data.columns[871:875]] = momentum(allmatchdata.iloc[n]["Team2"]) if momentum(allmatchdata.iloc[n]
["Team2"]) else None

    print(n, end=" ")

#Test data
```

Feature Extraction

This is the data we use to train the model. Columns are explained below. The rows show us the match number.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Match	T1P1 vs T2P1 r	T1P1 vs T2P1 b	T1P1 vs T2P1 w	T1P1 vs T2P2 r	T1P1 vs T2P2 b	T1P1 vs T2P2 w	T1P1 vs T2P3 r	T1P1 vs T2P3 b	T1P1 vs T2P3 w	T1P1 vs T2P4 r	T1P1 vs T2P4 b	T1P1 vs T2P4 w
2	324												
3	325												
4	326												
5	327												
6	328										2	6	0
7	329												
8	330												
9	331							15	11	1			
10	332												
11	333												

In the first three columns "T1P1 vs T2P1 runs", "T1P1 vs T2P1 balls", and "T1P1 vs T2P1 wickets", the batsman Team1 Player1 stats Vs the bowler Team2 Player2 are given (i.e. runs, balls, wickets). Likewise, we have taken the runs, balls, and wickets for every batsman in Team 1 vs every bowler in Team 2 (Total of 363 columns)

NA	NB	NC	ND	NE	NF	NG	NH	NI	NJ	NK	NL	NM
T2P1 vs T1P1 r	T2P1 vs T1P1 b	T2P1 vs T1P1 w	T2P1 vs T1P2 r	T2P1 vs T1P2 b	T2P1 vs T1P2 w	T2P1 vs T1P3 r	T2P1 vs T1P3 b	T2P1 vs T1P3 w	T2P1 vs T1P4 r	T2P1 vs T1P4 b	T2P1 vs T1P4 w	T2P1 vs T1P5 r
									4	6	0	
												9
12	12	1										
												3
						21	19	2	42	38	0	
												18
												9
						27	25	1				16

Same with Team 2 players batting Vs Team 1 players bowling (Total of 363 columns).

AAZ	ABA	ABB	ABC	ABD	ABE	ABF	ABG	ABH	ABI	ABJ	ABK	ABL
T1P1 last3_bat r	T1P1 last3_bat b	T1P1 last3_bat v	T1P2 last3_bat r	T1P2 last3_bat b	T1P2 last3_bat v	T1P3 last3_bat r	T1P3 last3_bat b	T1P3 last3_bat v	T1P4 last3_bat r	T1P4 last3_bat b	T1P4 last3_bat v	T1P5 last3_bat r
94	61	3	65	55	3	92	70	3	19	20	2	46
159	112	3				88	74	3	77	67	3	
34	49	3	24	21	2	183	110	3	93	71	2	75
99	67	2	42	36	3	174	118	2				
6	18	3	122	97	3	39	40	3	67	58	2	45
42	44	3	24	18	3	62	45	1	90	78	2	
58	48	3				36	44	2				
24	18	3	42	44	3	62	45	1	90	78	2	
138	97	3				84	80	3	34	36	3	
66	69	3				22	28	2	90	73	3	
			100	103	2							83
112	93	2	76	77	2	48	39	3	94	62	3	19
52	48	3	16	20	3				94	82	3	

In these columns, we are considering the form of the batsmen. We are taking the runs, balls played, and the number of times the batsman has been out. The above is for Team 1 and the below is for Team 2.

ACG	ACH	ACI	ACJ	ACK	ACL	ACM	ACN	ACO	ACP	ACQ	ACR	ACS
T2P1 last3_bat r	T2P1 last3_bat b	T2P1 last3_bat v	T2P2 last3_bat r	T2P2 last3_bat b	T2P2 last3_bat v	T2P3 last3_bat r	T2P3 last3_bat b	T2P3 last3_bat v	T2P4 last3_bat r	T2P4 last3_bat b	T2P4 last3_bat v	T2P5 last3_bat r
106	75	2	37	33	3	163	105	2				
62	62	3	41	41	3	54	57	3	59	61	1	67
			75	61	3	51	58	3				
160	120	2	97	90	2							
41	42	3	61	51	3	67	69	3				49
			100	103	2	80	80	3	15	19	3	83
			57	63	2	113	67	3				
30	33	3	134	113	3	57	47	3	100	64	2	60
102	75	2				186	121	2	101	69	2	
65	58	3	131	99	3							24
45	31	2	20	33	3	164	102	3	144	92	1	89
76	54	3				81	78	3	43	51	2	
			36	40	3	19	33	3	38	40	3	

AEE	AEF	AEG	AEH	AEI	AEJ	AEK	AEL	AEM	AEN	AEO	AEP	AEQ
T1P6 last3_bow	T1P7 last3_bow	T1P7 last3_bow	T1P7 last3_bow	T1P8 last3_bow	T1P8 last3_bow	T1P8 last3_bow	T1P9 last3_bow	T1P9 last3_bow	T1P9 last3_bow	T1P10 last3_bo	T1P10 last3_bo	T1P10 last3_bo
	69	60	0							79	72	0
0							72	66	0	78	60	0
	77	72	0				86	66	0			
							108	72	0	73	66	0
	54	48	0							59	72	0
	77	72	0	62	68	0	74	72	0	63	62	0
							74	48	0	63	66	0
0							62	68	0	63	62	0
	90	68	0				68	66	0	104	66	0
	112	66	0	85	66	0	35	24	0	58	72	0
				75	66	0						
							96	72	0	78	66	0
	79	72	0									

In these columns, we are considering the form of the bowlers. We are taking the runs, balls bowled, and the number of wickets he has taken. The above is for Team 1 and the below is for Team 2.

AFI	AFJ	AFK	AFL	AFM	AFN	AFO	AFP	AFQ	AFR	AFS	AFT	AFU	AFV
T2P5 last3_bow	T2P6 last3_bow	T2P6 last3_bow	T2P6 last3_bow	T2P7 last3_bow	T2P7 last3_bow	T2P7 last3_bow	T2P8 last3_bow	T2P8 last3_bow	T2P8 last3_bow	T2P9 last3_bow	T2P9 last3_bow	T2P9 last3_bow	T2P10 last3_bo
							99	72	0				80
	85	66	0	41	24	0	76	72	0	64	71	0	80
	56	48	0							76	60	0	57
				120	70	0				87	66	0	
0	41	24	0							72	72	0	66
							75	66	0	75	72	0	
				104	72	0							74
							64	48	0				77
							116	72	0				95
0	70	42	0							70	60	0	67
0							76	66	0	104	72	0	
				98	68	0				106	72	0	70
	97	72	0	35	24	0				62	66	0	76

AGB	AGC	AGD	AGE	AGF	AGG	AGH	AGI
T1 wins at stadiu	T1 losses at stac	T2 wins at stadiu	T2 losses at stac	T1 vs T2 win rati	T2 vs T1 win rati	Won defending a	Won chasing at
1	2	16	12	44.44	55.56	13	15
13	15	4	1	45.45	54.55	12	16
3	15	1	0	75	25	7	11
0	3	13	15	40	60	15	12
1	3	21	11	54.55	45.45	19	13
2	6	0	1	75	25	6	3
1	2	4	15	45.45	54.55	8	11
16	9	1	2	50	50	8	17
1	2	14	15	58.33	41.67	13	16
10	10	1	2	50	50	9	13
12	10	1	1	40	60	9	13
2	2	15	15	50	50	13	17
1	0	2	7	100		6	4
13	16	2	2	63.64	36.36	16	12
11	10	1	1	75	25	10	13
2	4	21	12	38.46	53.85	20	13
17	12	0	3	77.78	22.22	13	16
0	3	17	9	33.33	66.67	9	17
0	2	22	12	25	75	20	14
12	11	2	1	44.44	55.56	9	14
2	1	16	15	55.56	44.44	13	18
1	0	3	7	80	20	6	5

In the first four columns, we are recording Team 1 and Team 2 wins and losses at the stadium they are playing at.

The next two columns record the win ratio of Team 1 and Team 2 against each other.

The last two columns record the number of times the defending team and the chasing team have won at that stadium.

AGJ	AGK	AGL	AGM	AGN	AGO	AGP	AGQ	AGR
T1 fourth last ma	T1 third last mat	T1 second last n	T1 last match re	T2 fourth last ma	T2 third last mat	T2 second last n	T2 last match re	match result
								Team2
								Team1
								Team1
								Team1
								Team1
								Team2
								Team2
								Team1
								Team2
								Team1
								Team2
								Team2
								Team2
								Team2
								Team1
1	0	1	1					Team2
1	1	0	0	1	1	0	1	Team1
								Team2
0	0	1	0	0	0	1	1	Team1
				1	0	0	1	Team1
0	0	0	0	0	1	1	0	Team2
1	0	1	0	0	1	0	1	Team1
1	1	0	1	0	1	1	1	Team1
0	1	1	0	0	0	0	0	Team1
1	0	0	1	0	1	0	1	Team2

In these columns, we are taking the win-loss record of both the teams in the last 4 matches to obtain the momentum both teams have gained before their match.

Feature selection for our predictive model to predict the winning team

Model fitting for our predictive model:

Reading the dataset

```
import pandas as pd
```

```
train_new=pd.read_csv("/kaggle/input/momentum-data/Final train data momentum.csv")
test=pd.read_csv("/kaggle/input/momentum-test-data/Fianl test data momentum.csv")
train_new
```

	Match	T1P1 vs T2P6 runs	T1P1 vs T2P6 balls	T1P1 vs T2P6 wickets	T1P1 vs T2P7 runs	T1P1 vs T2P7 balls	T1P1 vs T2P7 wickets	T1P1 vs T2P8 runs	T1P1 vs T2P8 balls	T1P1 vs T2P8 wickets	...	Won chasing at stadium	T1 fourth last match result	T1 third last match result	T1 second last match result	T1 last match result	T2 fourth last match result	T2 third last match result	T2 second last match result	T2 last match result	match result
0	340	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	16	1	1	0	0	1	1	0	1	Team1
1	342	NaN	NaN	NaN	22.0	9.0	0.0	30.0	20.0	1.0	...	14	0	0	1	0	0	0	1	1	Team1
2	344	NaN	NaN	NaN	NaN	NaN	NaN	5.0	3.0	0.0	...	18	0	0	0	0	0	1	1	0	Team2
3	345	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	5	1	0	1	0	0	1	0	1	Team1
4	346	NaN	NaN	NaN	NaN	NaN	NaN	30.0	16.0	0.0	...	18	1	1	0	1	0	1	1	1	Team1

Data pre-processing for model building:

Data pre-processing for model building: Replacing all the null values of the test and training dataset null values with 0, since it would be more appropriate than average value for dataset.

```
train_new.fillna(0, inplace=True)
```

```
test.fillna(0, inplace=True)
```

+ Code + Markdown

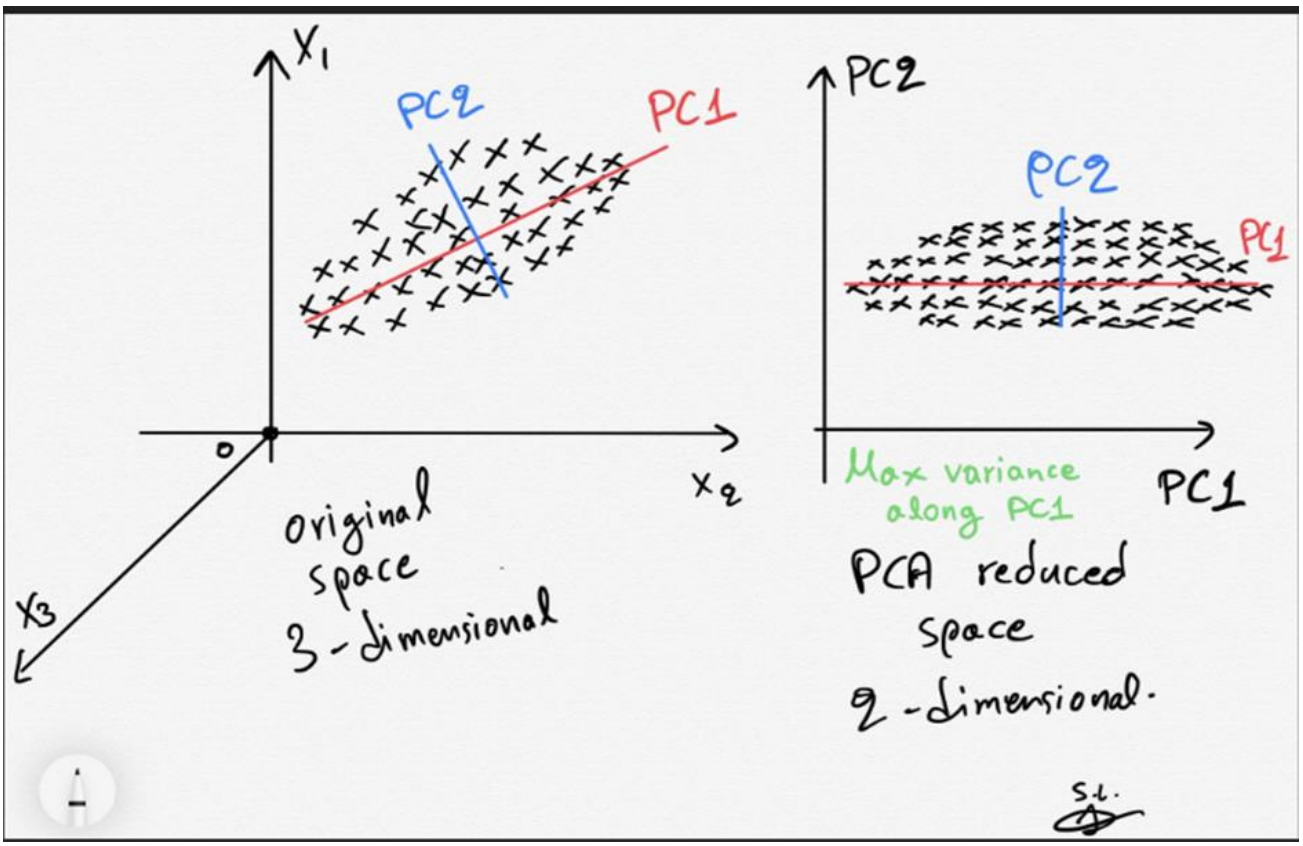
Deleting the column "Match" since the match number will not help to predict the winning team from both the test and train dataset.

```
del train_new["Match"]
```

+ Code + Markdown

```
[11]: del test["Match"]

[12]: train_new.shape
[12]: (313, 347)

[13]: test.shape
[13]: (142, 347)
```

Principal Component Analysis:



Principal Component Analysis (PCA) is a statistical technique used in data analysis and dimensionality reduction. It is a powerful tool that helps to identify underlying patterns and structures in high-dimensional data by

transforming the data into a lower-dimensional space. This transformation is done in a way that minimizes the loss of information from the original data set.

PCA is widely used in various fields, such as image processing, speech recognition, finance, biology, and many others. It is also commonly used in machine learning algorithms as a pre-processing step to reduce the number of features in a dataset, which can improve the performance of the model and reduce overfitting.

The basic idea behind PCA is to find a new set of variables, known as principal components, that capture the most variation in the original data. These principal components are linear combinations of the original variables, and they are orthogonal to each other. The first principal component captures the most variation in the data, and each subsequent principal component captures the most variation that is orthogonal to the previous components.

The process of performing PCA involves the following steps:

Standardizing the data: PCA requires that the data is standardized so that each variable has a mean of zero and a standard deviation of one. This step is important to ensure that each variable is equally important in the analysis.

Computing the covariance matrix: The covariance matrix is a matrix that describes the relationships between the variables in the data. It is computed by multiplying the transpose of the data matrix by the data matrix itself.

Computing the eigenvectors and eigenvalues: The eigenvectors and eigenvalues of the covariance matrix are computed. The eigenvectors are

the principal components, and the eigenvalues represent the amount of variation that is captured by each component.

Selecting the principal components: The principal components are selected based on their corresponding eigenvalues. The components with the highest eigenvalues capture the most variation in the data.

Transforming the data: The original data is transformed into a new space using the selected principal components.

PCA has several advantages, such as:

1.Dimensionality reduction: PCA can reduce the number of features in a dataset while preserving most of the information. This can simplify the analysis and improve the performance of machine learning algorithms.

2.Noise reduction: PCA can remove noise and unwanted variations in the data, which can improve the accuracy of the analysis.

3.Visualization: PCA can be used to visualize high-dimensional data in a lower-dimensional space. This can help to identify patterns and relationships in the data that may not be apparent in the original space.

However, PCA also has some limitations, such as:

1.Interpretability: The principal components generated by PCA may not have a clear interpretation in the original space, which can make it difficult to understand the relationships between variables.

2.Information loss: PCA may lead to information loss if the number of principal components selected is too low. This can result in a loss of important details in the data.

3.Sensitivity to outliers: PCA is sensitive to outliers in the data, which can affect the results of the analysis.

In summary, PCA is a powerful technique for analyzing high-dimensional data and reducing the number of features in a dataset. It has numerous applications in various fields and can improve the performance of machine learning algorithms. However, it also has some limitations that need to be considered when applying it to a dataset.

Concatenating the train and test dataset to apply PCA:

```
df=pd.concat([train_new,test],axis=0)
df
```

	T1P1 vs T2P6 runs	T1P1 vs T2P6 balls	T1P1 vs T2P6 wickets	T1P1 vs T2P7 runs	T1P1 vs T2P7 balls	T1P1 vs T2P7 wickets	T1P1 vs T2P8 runs	T1P1 vs T2P8 balls	T1P1 vs T2P8 wickets	T1P1 vs T2P9 runs	...	Won chasing at stadium	T1 fourth last match result	T1 third last match result	T1 second last match result	T1 last match result	T2 fourth last match result	T2 third last match result	T2 second last match result	T2 last match result	match result
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	...	16	1	1	0	0	1	1	0	1	Team1
1	0.0	0.0	0.0	22.0	9.0	0.0	30.0	20.0	1.0	0.0	...	14	0	0	1	0	0	0	1	1	Team1
2	0.0	0.0	0.0	0.0	0.0	0.0	5.0	3.0	0.0	0.0	...	18	0	0	0	0	0	1	1	0	Team2
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	...	5	1	0	1	0	0	1	0	1	Team1
4	0.0	0.0	0.0	0.0	0.0	0.0	30.0	16.0	0.0	0.0	...	18	1	1	0	1	0	1	1	1	Team1

Applying PCA using sklearn library with components (100)

```
#performing pca
from sklearn.decomposition import PCA
pca= PCA(n_components=100)
X_princple_components=pca.fit_transform(X)
X_princple_components
```

```
array([[-9.10222445e+01,  5.59216310e+01,  1.21583158e+01, ...,
        -1.41586307e+01, -1.09366821e+01, -2.78008284e+00],
       [-1.56698910e+01, -3.20719302e+01, -8.77757013e+01, ...,
         2.20463605e+00, -8.95802318e+00,  3.29712474e+00],
       [ 7.54345832e+01, -1.11552874e+02, -1.32421249e+02, ...,
         1.22904132e+01, -1.09895171e+01,  5.10878097e+00],
       ...
```

Scaling the feature:

Scaling the features is an important step in preparing data for machine learning algorithms. It involves transforming the features in a dataset so that they have a similar scale and range. There are several reasons why scaling is necessary:

Improving the performance of the models: Machine learning algorithms, such as linear regression, logistic regression, and k-nearest neighbors, are sensitive to the scale of the features. If the features are not scaled, the algorithms may give more importance to features with larger scales, even if they are not as important as features with smaller scales. Scaling the features can improve the performance of the models by ensuring that all features are equally important in the analysis.

Reducing the impact of outliers: Scaling can also help to reduce the impact of outliers in the data. Outliers are extreme values that can skew the analysis and affect the performance of the models. Scaling the features can help to reduce the impact of outliers by bringing the values of the features within a similar range.

Speeding up the convergence of algorithms: Some machine learning algorithms, such as gradient descent, converge faster when the features are scaled. This is because the algorithm can take larger steps towards the optimal solution, instead of getting stuck in local minima.

One of the most commonly used scaling techniques is MinMaxScaler. MinMaxScaler is a technique used to scale the features in a dataset to a specific range, usually between 0 and 1. It works by subtracting the minimum value of each feature from each observation and then dividing by the range of that feature.

The process of using MinMaxScaler involves the following steps:

Importing the library: The first step is to import the MinMaxScaler library from the scikit-learn package in Python.

Creating the scalar object: Next, we create an instance of the MinMaxScaler object.

Fitting the scalar to the data: The scalar object is then fitted to the data by passing the data to the fit () method.

Transforming the data: Once the scalar object is fitted to the data, we can transform the data by passing it to the transform () method. This will scale the features in the data to a specific range.

MinMaxScaler has several benefits, such as:

Improved performance: MinMaxScaler can improve the performance of machine learning models by ensuring that all features are equally important in the analysis.

Simplicity: MinMaxScaler is easy to use and can be applied to both numerical and categorical data.

Preserving the shape of the distribution: MinMaxScaler preserves the shape of the distribution of the features in the data, which can be useful for some algorithms that rely on the distribution of the data.

However, MinMaxScaler also has some limitations, such as:

Sensitivity to outliers: MinMaxScaler can be sensitive to outliers in the data, which can affect the results of the analysis.

Limited impact: MinMaxScaler may have limited impact on the analysis if the data is already well-scaled and normalized.

Inability to handle missing data: MinMaxScaler cannot handle missing data, and missing values need to be imputed before scaling the data.

In summary, scaling the features is an important step in preparing data for machine learning algorithms. MinMaxScaler is a commonly used scaling technique that can improve the performance of machine learning models by ensuring that all features are equally important in the analysis. However, it is important to be aware of its limitations and how to use it appropriately.

Using MinMaxScaler to scale the feature:

[145]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
X_princple_components=scaler.fit_transform(X_princple_components)
```

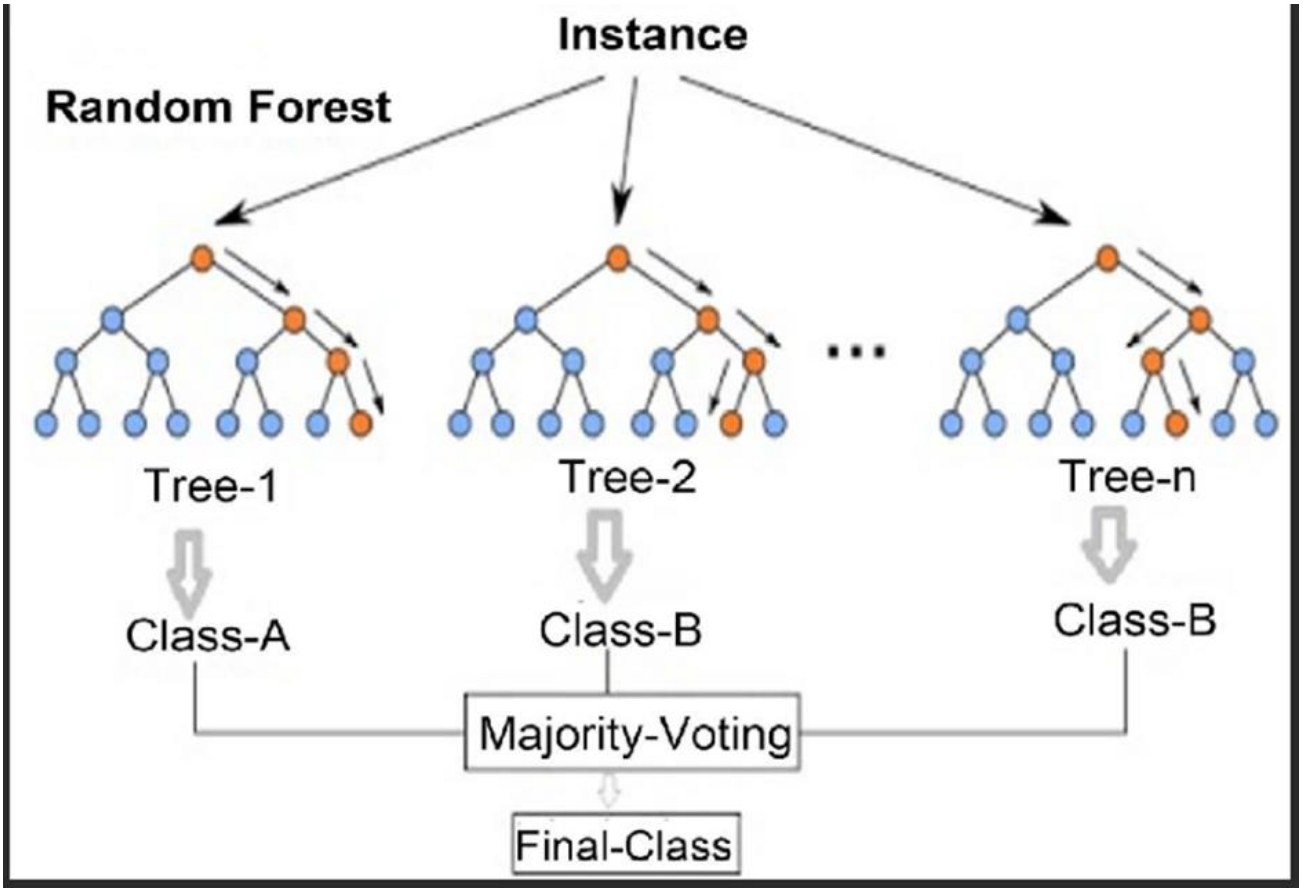
Splitting the data into train test split using test size of 20%

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train , y_test = train_test_split(X_princple_components,y, test_size=0.20,random_state=44)
```

Model for predicting the winning team :

We are using the random forest Classifier for predicting the winning team.

Random Forest Classifier:



Random Forest Classifier is a popular machine learning algorithm used for classification tasks. It is an ensemble learning method that combines multiple decision trees to make predictions. The name "Random Forest" comes from the fact that each tree in the ensemble is built using a

random subset of the features in the data. This randomization helps to reduce overfitting and improve the generalization of the model.

The Random Forest Classifier has several parameters that can be tuned to optimize the performance of the model. Some of the key parameters are:

n_estimators: This parameter specifies the number of trees in the ensemble. Increasing the number of trees can improve the accuracy of the model but can also increase the training time and memory requirements.

max_depth: This parameter controls the depth of each tree in the ensemble. A higher value for max_depth can increase the complexity of the model and improve the accuracy, but can also increase the risk of overfitting.

min_samples_split: This parameter specifies the minimum number of samples required to split an internal node. Increasing the value of min_samples_split can help to reduce overfitting and improve the generalization of the model.

min_samples_leaf: This parameter specifies the minimum number of samples required to be in a leaf node. Increasing the value of min_samples_leaf can help to reduce overfitting and improve the generalization of the model.

max_features: This parameter specifies the number of features to consider when looking for the best split at each node. The default value is "sqrt", which means that the square root of the total number of features is used. Increasing the value of max_features can increase the complexity of the model and improve the accuracy, but can also increase the risk of overfitting.

criterion: This parameter specifies the function used to measure the quality of a split. The default value is "gini", which uses the Gini impurity measure. The alternative option is "entropy", which uses the information gain measure.

bootstrap: This parameter specifies whether or not to use bootstrapping when building the trees in the ensemble. The default value is "True", which means that bootstrap sampling is used. Setting bootstrap to "False" can reduce the complexity of the model and improve the generalization, but can also reduce the accuracy.

These parameters can be tuned using techniques such as Grid Search or Random Search to find the optimal combination of values that maximize the performance of the model. It is important to note that the optimal values for the parameters can vary depending on the specific dataset and problem at hand.

In summary, the Random Forest Classifier is a powerful machine learning algorithm that can be used for classification tasks. It combines multiple decision trees to make predictions, and has several parameters that can be tuned to optimize its performance. Understanding the role of these parameters and how to tune them is important for building accurate and generalizable models.

Hyperparameter tuning: Hyperparameter tuning is the process of selecting the optimal values for the hyperparameters of a machine learning algorithm. Hyperparameters are parameters that are set before training the model, and they control the behavior of the algorithm during the training process. Examples of hyperparameters include the learning

rate, number of hidden layers, and number of nodes in each layer of a neural network.

Hyperparameter tuning is important because the performance of a machine learning algorithm is highly dependent on the values of the hyperparameters. Selecting the optimal values can lead to significant improvements in the accuracy and generalization of the model.

There are several methods for hyperparameter tuning, including grid search, random search, and Bayesian optimization. Grid search involves specifying a range of values for each hyperparameter and testing all possible combinations of values. Random search randomly samples from the range of values for each hyperparameter and tests a fixed number of combinations. Bayesian optimization uses a probabilistic model to guide the search for the optimal hyperparameters.

Hyperparameter tuning can be a time-consuming and computationally expensive process, especially for complex models with many hyperparameters. However, it is an important step in the machine learning pipeline and can lead to significant improvements in the performance of the model.

Code of hyperparameter tuning

[147]:
```
param_grid = {'n_estimators': [100, 300, 500, 800, 1200],
              'max_depth': [5, 10, 15, 20, 25, 30, None],
              'min_samples_split': [2, 5, 10, 15, 100],
              'min_samples_leaf': [1, 2, 4, 10],
              'max_features': ['sqrt', 'log2', None]}
```

[148]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestRegressor(random_state=42)

from sklearn.model_selection import RandomizedSearchCV

rf_random = RandomizedSearchCV(estimator=rf, param_distributions=param_grid, n_iter=100, cv=5, random_state=42, n_jobs=-1)
```

Best parameter we get after hyperparameter we get and fitting the model on Training dataset

```
print(rf_random.best_params_)
```

```
{'n_estimators': 500, 'min_samples_split': 15, 'min_samples_leaf': 10, 'max_features': None, 'max_depth': 20}
```

+ Code + Markdown

12]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators= 500, min_samples_split= 15, min_samples_leaf= 10, max_features= None, max_depth= 20)
```

13]:
```
rf.fit(X_train,y_train)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  """Entry point for launching an IPython kernel.
```

13.
```
RandomForestClassifier(max_depth=20, max_features=None, min_samples_leaf=10,
                       min_samples_split=15, n_estimators=500)
```

+ Code + Markdown

Predicting the values on testing dataset

```
[154]: y_pred = rf.predict(X_test)
       y_pred=pd.DataFrame(y_pred)
       y_pred
```

[154…

	0
0	2
1	2
2	2
3	2
4	2
...	...
86	1
87	1
88	2
89	2
90	2

91 rows × 1 columns

Metrics used for evaluating the model

Confusion Matrix: A confusion matrix is a table that shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for a binary classification problem. It is used to evaluate the performance of a classification model by calculating various metrics such as accuracy, precision, recall, and F1 score.

Accuracy: Accuracy is the most commonly used metric for evaluating a classification model. It measures the proportion of correct predictions out of the total number of predictions. It is calculated as (TP + TN) / (TP + TN + FP + FN). However, accuracy can be misleading when the classes are imbalanced or when the cost of false positives and false negatives are not equal.

Precision: Precision measures the proportion of positive predictions that are true positives. It is calculated as TP / (TP + FP). A high precision indicates that the model is good at predicting positive cases, but it may miss some positive cases.

Recall: Recall measures the proportion of actual positive cases that are correctly identified by the model. It is calculated as TP / (TP + FN). A high recall indicates that the model is good at identifying positive cases, but it may also predict some false positives.

F1 Score: F1 score is the harmonic mean of precision and recall, which gives equal weight to both measures. It is calculated as 2 * (precision * recall) / (precision + recall). F1 score is a better metric than accuracy when the classes are imbalanced or when the cost of false positives and false negatives are not equal.

Results :

Results from the dataset:

```
[155]: from sklearn.metrics import accuracy_score,confusion_matrix,f1_score, precision_score, recall_score
```

```
[159]: accuracy_score(y_test, y_pred)
```

```
0.65281
```

+ Code + Markdown

```
[162]: recall_score(y_test, y_pred)
```

```
0.6
```

```
[161]: precision_score(y_test, y_pred)
```

```
0.818
```

```
f1_score(y_test, y_pred)
```

```
0.694
```

+ Code + Markdown

```
]: confusion_matrix(y_test,y_pred)
```

```
… array([[23,  8],
       [24, 36]])
```

```
:
```

Recall=0.6

F1 score=0.694

Accuracy=0.65281

Precision = 0.818

SUMMARIZE PROJECT REPORT

In this project, we performed exploratory data analysis (EDA) on the IPL dataset, extracted relevant features, and used a random forest classifier to predict the winning team in a given match.

The project started by importing the IPL dataset and performing basic data cleaning and preprocessing tasks, such as handling missing values and encoding categorical variables. We then performed EDA on the dataset to gain insights into various factors that can affect the outcome of a match, such as team performance, player statistics, venue, and toss. The EDA helped us to identify the most important features for predicting the winning team, such as the number of runs scored, wickets taken, and the team's win-loss record in the current season.

Next, we used feature engineering techniques to extract relevant features from the dataset, such as the team's previous match performance, the average run rate, and the number of wins in the last five matches.

We also created new features, such as the difference in runs scored and wickets taken between the two teams, and the percentage of toss wins by a team in a given season. These features were used to train the random forest classifier, which is a powerful machine learning algorithm for classification tasks.

We split the dataset into training and testing sets, and trained the random forest classifier. We evaluated the performance of the classifier using various metrics, such as accuracy, precision, recall, and F1 score. The results showed that the random forest classifier was able to predict the winning team with an accuracy of around 65%, which is a significant improvement over random guessing.

We also performed feature importance analysis to identify the most important features for predicting the winning team using PCA (principal component analysis).

In conclusion, this project demonstrated how EDA, feature engineering, and machine learning techniques can be used to extract relevant features and predict the winning team in IPL matches. The project also highlighted the importance of various factors that can affect the outcome of a match, such as team and player statistics, venue, and toss. This information can be useful for cricket teams, coaches, and analysts to improve their performance and increase their chances of winning matches.