STATISTICAL ANALYSIS OF
INDIAN PREMIER LEAGUE(IPL) 2008-2023

A Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of
M.Sc. Statistics
(With Specialization in Industrial Statistics)

Submitted by

Mr. Wankhede Mithilesh Bhausaheb (379387)

Ms. Mandage Akshada Ravindra (379357)

Ms. Patil Poonam Mangesh (379370)

Under the Guidance of

Asst. Prof. MANOJ C. PATIL

in the

Department of Statistics, School of Mathematical Sciences,

Kavayitri Bahinabai Chaudhari North Maharashtra

University, Jalgaon-425001.

(Academic Year : 2023-2024)

DEPARTMENT OF STATISTICS

KAVAYITRI BAHINABAI CHAUDHARI

NORTH MAHARASHTRA UNIVERSITY, JALGAON

CERTIFICATE

This is to certify that **Mr. Wankhede Mithilesh Bhausaheb, Ms. Mandage Akshada Ravindra** and **Ms. Patil Poonam Mangesh** are the student of M.Sc. Statistics (with specialization in Industrial Statistics) at Department of Statistics, School of Mathematical Sciences, Kavayitri Bahinabai Chaudhari North Maharashtra University, Jalgaon have successfully completed their project entitled **"Statistical Analysis of Indian Premier League (IPL) 2008-2023"** under my guidance and supervision during the academic year 2023-2024.

Place :- Jalgaon
Date :- May 30, 2024

Asst. Prof. Manoj C. Patil
(Project Guide)
Department Of Statistics
Kavayitri Bahinabai Chaudhari
North Maharashtra University,
Jalgaon.

ACKNOWLEDGEMENT

Upon the completion of this project, we wish to extend our heartfelt gratitude to **Dr. R. L. Shinde**, Head of the Department of Statistics, School of Mathematical Sciences, Kavayitri Bahinabai Chaudhari North Maharashtra University, Jalgaon, for granting us the necessary permission to undertake this project.

We also take this opportunity to express our profound appreciation to our project guide, **Asst. Prof. Manoj C. Patil**, for his invaluable guidance, immense support, motivation, and encouragement, which were instrumental in the successful completion of our project work.

Our sincere thanks go to **Dr. K. K. Kamalja** for his unwavering support throughout the project. Additionally, we are deeply grateful to **Dr. R. D. Koshti** for his valuable guidance and insights that significantly contributed to our work.

We extend our heartfelt thanks to our parents, friends, and classmates for their moral support. We are also appreciative of everyone who directly or indirectly assisted us in completing our project work.

Mr. Wankhede Mithilesh Bhausaheb (Seat No.379387)

Ms. Mandage Akshada Ravindra (Seat No.379357)

Ms. Patil Poonam Mangesh (Seat No.379370)

Contents

Chapter 1

Introduction

1.1 Motivation

You see, cricket isn't just a game for us Indians; it's part of our culture, our passion. And the IPL, well, it's like the heartbeat of cricket, pulsating with excitement and energy. Now, as statisticians, diving into IPL data is like uncovering hidden gems. We're not just crunching numbers; we're decoding the essence of the game.

This project? It's about using stats to understand cricket better, to help teams make smarter decisions and players reach their full potential. Plus, it's not just about cricket; it's about the stories behind the game—how it connects people, reflects our society, and even influences trends.

So, by analyzing IPL data, we're not just playing with numbers; we're unraveling the magic of cricket and adding our bit to its incredible journey. It's a chance to blend our love for the game with our passion for numbers, all while celebrating the spirit of cricket that runs deep in our veins.

1.2 Objectives of the Project

The objectives of selecting these datasets for the project include:

◇ **Exploratory Data Analysis (EDA)**

Perform EDA to uncover patterns, trends, and insights from the IPL data, including run distributions, wicket distributions, and team performance across seasons.

◇ **Examine Dependencies and the Significance Differences**

Conduct hypothesis tests to examine dependencies and the significance of factors such as the impact of winning the toss on match outcomes and significant difference between group means.

◇ **Player's Ranking by PCA**

Perform Principal Component Analysis (PCA) to rank players based on their overall performance metrics, combining batting and bowling statistics.

◇ **Prediction of Winning Team**

Develop predictive models to forecast the winning team based on historical match data and in-game statistics.

1.3 About IPL

The Indian Premier League (IPL), also known as the TATA IPL for sponsorship reasons, is a men's Twenty20 (T20) cricket league held annually in India. Founded by the BCCI in 2007, the league features ten city-based franchise teams. The IPL usually takes place during the summer, between March and May each year. It has an exclusive window in the ICC Future Tours Programme, resulting in fewer international cricket tours occurring during the IPL seasons.

The IPL is the most popular cricket league in the world; in 2014, it ranked sixth in average attendance among all sports leagues. In 2010, the IPL became the first sporting event to be broadcast live on YouTube. Inspired by the success of the IPL, other Indian sports leagues have been established. In 2022, the league's brand value was estimated at 90,038 crore (US $11 billion). According to the BCCI, the 2015 IPL season contributed 1,150 crore (US$140 million) to India's GDP. In December 2022, the IPL achieved a valuation of US $10.9 billion, becoming a decacorn and registering a 75% growth in dollar terms since 2020 when it was valued at $6.2 billion, according to a report by the consulting firm D and P Advisory. Its 2023 final became the most streamed live event on the internet, with 32 million viewers

In 2023, the league sold its media rights for the next 4 Seasons for US $6.4 billion to Viacom18 and Star Sports, meaning each IPL match was valued at $13.4 million. As of 2023, there have been sixteen seasons of the tournament. The current champions are the Chennai Super Kings, who won the 2023 season after defeating the Gujarat Titans in the final.

1.4 History of IPL

1.4.1 Inspired by a rival

In 2007, Zee Entertainment Enterprises founded the Indian Cricket League (ICL). The ICL was not recognized by the Board of Control for Cricket in India (BCCI) or the International Cricket Council (ICC). Moreover, the BCCI was unhappy about its committee members joining the ICL executive board. In response, the BCCI increased the prize money for its domestic tournaments and imposed lifetime bans on players who joined the rival league, which it considered a rebel league.

1.4.2 Foundation

On 13 September 2007, following India's victory at the 2007 T20 World Cup, the BCCI announced a franchise based Twenty20 cricket competition known as the Indian Premier League. The inaugural season was scheduled to start in April 2008, commencing with a "high-profile ceremony" in New Delhi. BCCI Vice-president Lalit Modi, who led the IPL initiative, provided details of the tournament, including its format, prize money, franchise revenue system, and squad composition rules. The league, to be managed by a seven-man governing council, would also serve as the qualifying mechanism for that year's Champions League Twenty20.

To determine team ownership, an auction for the franchises was held on 24 January 2008. The reserve prices for the eight franchises totalled $400 million, but the auction ultimately raised $723.59 million. The league officially commenced in April 2008, featuring Chennai Super Kings (CSK), Mumbai Indians (MI), Delhi Daredevils (DD), Kings XI Punjab (KXIP), Deccan Chargers (DC), Rajasthan Royals (RR), Kolkata Knight Riders (KKR), and Royal Challengers Bangalore (RCB). Following the ban on players who chose to participate in the ICL, the rival league shut down in 2009.

Expansions and terminations

New franchises, Pune Warriors India and Kochi Tuskers Kerala, joined the league before the fourth season in 2011. The Sahara Adventure Sports Group purchased the Pune franchise for $370 million, while Rendezvous Sports World bought the Kochi franchise for $333.3 million. The Kochi franchise was terminated after just one season due to their failure to pay the BCCI the 10% bank guarantee element of the franchise fee.

In September 2012, the Deccan Chargers franchise agreement was terminated after the BCCI failed to find new owners. In October, an auction was held for a replacement franchise; Sun TV Network won the bid for what became the Hyderabad franchise; the team was named Sunrisers Hyderabad.

Pune Warriors India withdrew from the IPL in May 2013 due to financial differences with the BCCI. The BCCI officially terminated the franchise in October, and the league reverted to eight teams.

In June 2015, the two-time champions Chennai Super Kings and the inaugural season champions Rajasthan Royals were suspended for two seasons following their involvement in a spot-fixing and betting scandal. The two teams were replaced for two seasons by franchises based in Pune and Rajkot.

Due to the COVID-19 pandemic, the venue for the 2020 season was moved and games were played in the United Arab Emirates. In August 2021, the BCCI announced two new franchises, based in two of six shortlisted cities, would join the league in the 2022 season. In closed bidding held in October, the RPSG Group and CVC Capital won the bids for the

teams, paying 7,000 crore (US $880 million) and 5,200 crore (US $650 million), respectively. The teams were subsequently named Lucknow Super Giants and Gujarat Titans.

Several IPL franchise owners have expanded their business by acquiring teams in other franchise leagues, such as the Caribbean Premier League (CPL), South Africa's SA20, the UAE's International League T20 (ILT) and the USA's Major League Cricket (MLC). These teams have been branded with similar names to their parent IPL franchises.

IPL	CPL	SA20	ILT	MLC
CSK		Joburg Super Kings		Texas Super Kings
DC		Pretoria Capitals	Dubai Capitals	
GT				
KKR	Trinbago Knight Riders		Abu Dhabi Knight Riders	Los Angeles Knight Riders
LSG		Durban's Super Giants		
MI		MI Cape Town	MI Emirates	MI New York
PK	Saint Lucia Kings			
RR	Barbados Royals	Paarl Royals		
RCB				
SRH		Sunrisers Eastern Cape		

Table 1.1: Comparison of Cricket Teams Across Leagues

1.5 Organization

The IPL's headquarters are located in the Cricket Centre, next to the Wankhede Stadium in Churchgate, Mumbai. The Governing Council is responsible for the league's functions, including the organization of tournaments. As of April 2023, its members included:

- Arun Singh Dhumal - Chairman
- Jay Shah - Secretary of the BCCI
- Ashish Shelar - Treasurer, BCCI
- Avishek Dalmiya
- Pragyan Ojha - Indian Cricketers' Association's representative
- Alka Rehani Bhardwaj - Comptroller and Auditor General of India nominee

1.5.1 Player acquisition, squad composition, and salaries

A team can acquire players through the annual player auction, trading with other teams during trading windows, and signing replacements for unavailable players. Players sign up for the auction and set their base price and are bought by the highest-bidding franchise. Players unsold at the auction are eligible to be signed as replacement signings. In the trading windows, a player can only be traded with consent; the franchise pays any difference between the old and new contracts. If the new contract is worth more

than the old one, the player and the selling franchise share the difference. There are generally three trading windows – two before the auction and one between the auction and the start of the tournament. Players cannot be traded outside the trading windows or during the tournament, but replacements can be signed before or during the tournament.

Other notable rules, as of the 2020 season, include:

- The salary cap of the entire squad must not exceed 85 crore (US $11 million).
- Under-19 players cannot be picked unless they have previously played first-class or List A cricket.

Player contracts run for one year but can be extended by one or two years if the franchises take up the option. Since the 2014 season, player contracts have been denominated in the Indian rupee, before which the contracts were in the US dollar. Overseas players can be remunerated in the currency of the player's choice, at the exchange rate on either the contract due date or the actual payment date. Before the 2014 season, Indian domestic players were not included in the player auction pool. They could be signed up by franchises at a discrete amount while a fixed sum of 10 lakh (US $13,000) to 30 lakh (US $38,000) would be deducted per signing from the franchise's salary purse. This received significant opposition from franchise owners, who complained richer franchises were "luring players with under-the-table deals." The IPL later decided to include domestic players in the player auction.

The BCCI gives 10% of foreign players' salaries to their country's national cricket board. According to a 2015 survey by Sporting Intelligence and ESPN The Magazine, the average IPL salary when pro-rated is US$4.33 million per year, the second-highest of sports leagues in the world. Because players in the IPL are contracted only for the duration of the tournament - less than two months - the weekly IPL salaries are extrapolated pro data to obtain an average annual salary, unlike other sports leagues in which players are contracted by a single team for the entire year.

According to a report by The Telegraph, IPL players are paid 18% of the revenue, which is the lowest amount compared to other major sports leagues, in which players receive at least 50% of the revenue. The Federation of International Cricketers' Associations said that IPL players must be paid fairly.

1.5.2 Prize Money

The 2022 season of the IPL offered total prize money of 46.5 crore (equivalent to 49 crore or US$6.2 million in 2023), with the winning team netting 20 crore (equivalent to 21 crore or US$2.7 million in 2023) and the second-placed team 13 crore (equivalent to 14 crore or US$1.7 million in 2023). League rules mandate that half of any prize money must be distributed amongst the franchise's players.

1.6 Unique Rules and Variations

- IPL games incorporate television timeouts. Each team is given a two-and-a-half-minute "strategic time-out" during each innings. One must be taken by the bowling team between the seventh and ninth overs and the other by the batting team between the 14th and 16th overs. A penalty may be imposed if umpires find teams misusing this privilege.
- Since the 2018 season, the Decision Review System (DRS) has been used in all IPL matches, allowing each team two opportunities each innings to review an on-field umpire's decision. From the 2023 season, this was extended to allow the review of wides and no-balls.
- If the bowling team does not complete its overs in the allocated time, it may place only four fielders outside of the fielding restrictions circle for the remainder of the innings, or the match referee may impose financial sanctions on the bowling team after the match, with players fined a proportion of their match fee.
- Teams can use a substitute, termed an "impact player", from a list of five players named as possible substitutes. The substitution can be made before the start of the innings, when a wicket falls when a batter retires or at the end of an over. Both teams can introduce a substitute once per match.
- Teams can declare their playing eleven to the match referee before or after the toss.
- A five-run penalty is imposed if a fielder or wicket-keeper makes an unfair movement while the bowler is bowling and the ball is designated as dead ball.
- Teams can include four overseas players in their playing eleven.
- Teams must include 25 players, with a maximum of eight overseas players.
- From the 2024 season, bowlers will be allowed to deliver two bouncers an over. This change in playing conditions was trialled during the 2023–24 Syed Mushtaq Ali Trophy, India's domestic T20 tournament.

1.7 Awards

1.7.1 Orange Cap

The Orange Cap, introduced in 2008, is awarded to the highest run-scorer at the end of each season. It is an ongoing competition with the current highest-run scorer wearing the cap whilst fielding. The eventual winner keeps the cap for the season. Brendon McCullum was the first player to wear the Orange Cap and Shaun Marsh the inaugural winner of the award. Australian batsman David Warner has won the award three times, more than any other player. Shubman Gill of Gujarat Titans, who scored 890 runs during the 2023 season, is the most recent winner of the award.

1.7.2 Purple Cap

The Purple Cap is awarded to the highest wicket-taker at the end of each season. It is an ongoing competition and the bowler who is the leading wicket-taker wears a purple cap whilst fielding. The leading wicket-taker at the end of the season wins the award. Bhuvneshwar Kumar and Dwayne Bravo are the only players to have won the award twice.

1.7.3 Most Valuable Player

The Most Valuable Player award, called the "Man of the Tournament" until the 2012 season, is awarded using a ratings system introduced in 2013. Shubman Gill won the award in 2023.

1.7.4 Fair Play Award

The Fair Play Award is given after each season to the team considered to have the best fair play record. After each match, the two on-field umpires and the third umpire score the performance of both teams, with the highest-scoring team at the end of the season receiving the award. The 2023 winners were Delhi Capitals.

1.7.5 Emerging Player Award

The Emerging Player Award was presented to the best under-19 player in 2008 and the best under-23 player in 2009 and 2010. In 2011 and 2012, the award was known as "Rising Star of the Year," and in 2013 the "Best Young Player of the Season." Since 2014, the award has been called the Emerging Player of the Year. Mustafizur Rahman is the only foreign player to win this award. The 2023 winner was Yashasvi Jaiswal.

1.7.6 Maximum Sixes Award

The Maximum Sixes Award is presented to the player who hits the most sixes at the end of the season.

1.8 Title Sponsorship

From 2008 to 2012, the IPL title sponsor was DLF, India's largest real estate developer, which bid 200 crore (US$25 million) for the rights for five seasons. After the 2012 season, PepsiCo bought the title sponsorship rights for 397 crore (US$50 million) for the next five seasons but terminated the deal in October 2015, two years before the expiry of the contract, due to the two-season suspension of the Chennai and Rajasthan franchises from the league. The BCCI transferred the title sponsorship rights for the remaining two seasons of the contract to Chinese smartphone manufacturer Vivo for 200 crore (US$25 million).

In June 2017, Vivo retained the rights for the next five seasons from 2018 to 2022 with a winning bid of 2,199 crore (US$280 million). On 4 August 2020, Vivo canceled the title sponsorship rights due to the military stand-off between India and China at the Line of Actual Control in July 2020. The withdrawal was also a result of Vivo's market losses due

to the ongoing COVID-19 pandemic; Vivo intended to return as the title sponsor for the following three years. Dream11 became the title sponsors for the 2020 IPL for an amount of 222 crore (equivalent to 261 crore or US$33 million in 2023). Vivo returned as the title sponsor for the 2021 IPL season but withdrew again, and was replaced by the Tata Group for the next two seasons. InsideSport reported the BCCI would receive 498 crore (US$62 million) for the 2022 and 2023 seasons from title sponsors. Vivo had previously agreed to pay a higher amount for the last two seasons of its sponsorship contract due to the league's expansion from the 2022 season. According to InsideSport, due to the new deal's structure, Tata would pay 335 crore (US$42 million) per year while Vivo would pay the deficit of 163 crore (US$20 million) per season. Saudi Aramco brought the rights to advertise on the Purple and Orange caps in 2022.

Sponsor	**Period**	**Estimated Annual Sponsorship Fees**
DLF	2008–2012	40 crore (US$5.0 million)
Pepsi	2013–2015	79.2 crore (US$9.9 million)
Vivo	2016, 2017	100 crore (US$12.5 million)
	2018, 2019, 2021	440 crore (US$55.1 million)
Dream11	2020	222 crore (US$27.8 million)
Tata	2022–2023	335 crore (US$42.0 million)
	2024–2028	500 crore (US$62.6 million)

Table 1.2: IPL Title Sponsorship Details

Chapter 2

Data Description

2.1 Data Sources

For this project, two primary datasets were collected from Kaggle:

- **IPL Ball-by-Ball Dataset** (2008 to 2023): This dataset contains detailed information about every ball bowled in IPL matches from 2008 to 2022. It has 243,815 entries with 19 columns. https://www.kaggle.com/datasets/dgsports/ipl-ball-by-ball-2008-to-2022
- **IPL Matches Dataset** (2008 to 2023): This dataset provides information about all IPL matches played from 2008 to 2023.https://www.kaggle.com/datasets/vora1011/ipl-2008-to-2021-all-match-dataset?select=IPL_Matches_2008_2022.csv

2.2 Ball-by-Ball Dataset

The ball-by-ball dataset includes the following variables:

- **ID**: Unique identifier for each match.
- **season**: The season or year in which the match was played.
- **start date**: The date when the match was played.
- **venue**: The stadium or venue where the match took place.
- **Inns**: The innings number (1 for first innings, 2 for second innings).
- **overs**: The over number within the innings.
- **ball number**: The ball number within the over.
- **Batting Team**: The team that is batting.
- **bowling team**: The team that is bowling.
- **batter**: The batsman facing the delivery.
- **non striker**: The batsman at the non striker end.
- **bowler**: The bowler delivering the ball.

- **batsman run**: Runs scored by the batsman on that ball.
- **extras run**: Extra runs conceded (wides, no-balls, etc.).
- **total run**: Total runs scored on that ball (batsman runs + extras).
- **extra type**: Type of extra run (wide, no ball, etc.), if any.
- **isWicketDelivery**: Indicator if the delivery resulted in a wicket (1 if true, 0 if false).
- **wicket type**: The type of wicket (caught, bowled, etc.), if any.
- **player dismissed**: The player who was dismissed, if any.

2.3 Derived Datasets

Player Performance Dataset

This dataset aggregates individual player performances across matches and derived from IPL ball by ball dataset:

- **Player**: Name of the player.
- **Matches**: Number of matches played by the player.
- **innings bat**: Number of batting innings played.
- **Runs**: Total runs scored by the player.
- **Balls played**: Total balls faced by the player.
- **Outs**: Number of times the player got out.
- **SR**: Strike rate (Runs/Balls played * 100).
- **Avg**: Batting average (Runs/Outs).
- **Centuries**: Number of centuries scored.
- **Fifties**: Number of fifties scored.
- **Fours**: Number of boundaries (fours) hit.
- **Sixes**: Number of sixes hit.
- **innings bowl**: Number of innings bowled.
- **Wickets taken**: Total wickets taken.
- **Total balls**: Total balls bowled.
- **Total runs given**: Total runs conceded by the player.
- **Economy**: Economy rate (Total runs given/Total balls * 6).
- **Bowling avg**: Bowling average (Total runs given/Wickets taken).
- **Bowling SR**: Bowling strike rate (Total balls/Wickets taken).

Runs and Wickets Dataset

This dataset summarizes runs and wickets data by match innings and phases:

- **Total Runs 1st Inns**: Total runs scored in the first innings.
- **Total Runs 2nd Inns**: Total runs scored in the second innings.
- **Total Wkts 1st Inns**: Total wickets taken in the first innings.
- **Total Wkts 2nd Inns**: Total wickets taken in the second innings.
- **Power Play Runs 1st Inns**: Runs scored in the power play of the first innings.
- **Middle Overs Runs 1st Inns**: Runs scored in the middle overs of the first innings.
- **Death Overs Runs 1st Inns**: Runs scored in the death overs of the first innings.
- **Power Play Runs 2nd Inns**: Runs scored in the power play of the second innings.
- **Middle Overs Runs 2nd Inns**: Runs scored in the middle overs of the second innings.
- **Death Overs Runs 2nd Inns**: Runs scored in the death overs of the second innings.
- **Power Play Wkts 1st Inns**: Wickets taken in the power play of the first innings.
- **Middle Overs Wkts 1st Inns**: Wickets taken in the middle overs of the first innings.
- **Death Overs Wkts 1st Inns**: Wickets taken in the death overs of the first innings.
- **Power Play Wkts 2nd Inns**: Wickets taken in the power play of the second innings.
- **Middle Overs Wkts 2nd Inns**: Wickets taken in the middle overs of the second innings.
- **Death Overs Wkts 2nd Inns**: Wickets taken in the death overs of the second innings.

2.4 Merged Match Dataset

This dataset merges the runs and wickets data with IPL Matches dataset:

- **ID**: Unique identifier for each match.
- **City**: The city where the match was played.
- **Date** : The date of the match.
- **Season**: The season or year of the match.
- **Match Number**: The match number in the season.
- **Team1**: The first team playing the match.
- **Team2**: The second team playing the match.
- **Venue**: The venue where the match was held.
- **Toss Winner**: The team that won the toss.
- **Toss Decision**: The decision made by the toss-winning team (bat or bowl).
- **Winning Team**: The team that won the match.

2.5 Data Cleaning and Missing Values

Data cleaning is a critical step to ensure the quality and reliability of the analysis. The following steps were taken:

◇ **Handling Missing Values**

- In the ball by ball dataset, missing values in columns such as `extra type`, `wicket type`, and `player dismissed` were left as is, as these fields are only applicable for certain types of deliveries.
- Other missing values were imputed or removed as necessary to maintain data integrity.

◇ **Data Consistency and Formatting**

- Standardized date formats across datasets and necessarily created dummy variables.
- Checked for consistency in team and player names to avoid discrepancies.

2.6 Project Data

The project data we used is contained on the next pages.

ID	City	Date	Season	Match No.	Team1	Team2	Venue	TossWinner	TossDecision	WinningTeam
335982	Bangalore	18-04-2008	2008	1	RCB	KKR	M Chinnaswamy	RCB	field	KKR
335983	Chandigarh	19-04-2008	2008	2	PK	CSK	Punjab Cricket	CSK	bat	CSK
335984	Delhi	19-04-2008	2008	3	DC	RR	Feroz Shah Kotla	RR	bat	DC
335985	Mumbai	20-04-2008	2008	4	MI	RCB	Wankhede	MI	bat	RCB
335986	Kolkata	20-04-2008	2008	5	KKR	SRH	Eden Gardens	SRH	bat	KKR
335987	Jaipur	21-04-2008	2008	6	RR	PK	Sawai Mansingh	PK	bat	RR
335988	Hyderabad	22-04-2008	2008	7	SRH	DC	Rajiv Gandhi	SRH	bat	DC
335989	Chennai	23-04-2008	2008	8	CSK	MI	MA Chidambaram	MI	field	CSK
335990	Hyderabad	24-04-2008	2008	9	SRH	RR	Rajiv Gandhi	RR	field	RR
335991	Chandigarh	25-04-2008	2008	10	PK	MI	Punjab Cricket	MI	field	PK
335992	Bangalore	26-04-2008	2008	11	RCB	RR	M Chinnaswamy	RR	field	RR
335993	Chennai	26-04-2008	2008	12	CSK	KKR	MA Chidambaram	KKR	bat	CSK
335994	Mumbai	27-04-2008	2008	13	MI	SRH	Dr DY Patil Academy	SRH	field	SRH
335995	Chandigarh	27-04-2008	2008	14	PK	DC	Punjab Cricket	DC	bat	PK
335996	Bangalore	28-04-2008	2008	15	RCB	CSK	M Chinnaswamy	CSK	bat	CSK
335997	Kolkata	29-04-2008	2008	16	KKR	MI	Eden Gardens	KKR	bat	MI
335998	Delhi	30-04-2008	2008	17	DC	RCB	Feroz Shah Kotla	RCB	field	DC
335999	Hyderabad	01-05-2008	2008	18	SRH	PK	Rajiv Gandhi	PK	field	PK
336000	Jaipur	01-05-2008	2008	19	RR	KKR	Sawai Mansingh	RR	bat	RR
336001	Chennai	02-05-2008	2008	20	CSK	DC	MA Chidambaram	CSK	bat	DC
336002	Hyderabad	25-05-2008	2008	50	SRH	RCB	Rajiv Gandhi	SRH	bat	RCB
336003	Chandigarh	03-05-2008	2008	21	PK	KKR	Punjab Cricket	PK	bat	PK
336004	Mumbai	04-05-2008	2008	23	MI	DC	Dr DY Patil Academy	DC	field	MI
336005	Jaipur	04-05-2008	2008	24	RR	CSK	Sawai Mansingh	CSK	bat	RR
336006	Bangalore	05-05-2008	2008	25	RCB	PK	M Chinnaswamy	PK	field	PK

Table 2.1: project data

Total Runs 1st Inns	Total Runs 2nd Inns	Total Wkts 1st Inns	Total Wkts 2nd Inns	Power Play Runs 1st Inns	Middle Overs Runs 1st Inns	Death Overs Runs 1st Inns	Power Play Runs 2nd Inns	Middle Overs Runs 2nd Inns	Death Overs Runs 2nd Inns	Power Play Wkts 1st Inns	Middle Overs Wkts 1st Inns	Death Overs Wkts 1st Inns	Power Play Wkts 2nd Inns	Middle Overs Wkts 2nd Inns	Death Overs Wkts 2nd Inns
222	82	3	10	68	90	64	33	49	0	1	1	1	4	6	0
240	207	5	4	62	107	71	69	107	31	2	3	0	1	2	1
129	132	8	1	44	62	23	59	73	0	3	4	1	1	0	0
165	166	7	5	54	64	47	46	80	40	3	1	3	1	3	1
110	112	10	5	45	37	28	33	54	25	2	4	4	3	1	1
166	168	8	4	59	69	38	60	92	16	2	2	4	3	1	0
142	143	8	1	34	48	60	55	88	0	2	3	3	1	0	0
208	202	5	7	59	99	50	57	90	55	2	1	2	2	4	1
214	217	5	7	61	99	54	82	94	41	2	1	2	1	2	4
182	116	10	9	70	76	36	46	59	11	1	3	6	2	6	1
135	138	8	3	64	44	27	52	78	8	3	4	1	2	1	0
147	152	9	1	63	45	39	55	84	13	3	3	3	0	1	0
154	155	7	0	35	79	40	83	72	0	3	2	2	0	0	0
158	162	8	6	54	68	36	62	71	29	3	2	3	2	2	2
178	165	5	10	43	73	62	56	84	25	2	1	2	1	3	6
137	138	8	3	47	46	44	43	78	17	3	4	1	3	0	0
191	181	5	5	71	88	32	59	81	41	1	1	3	3	1	1
164	167	8	3	54	65	45	46	94	27	2	4	2	2	1	0
196	151	7	10	53	103	40	54	82	15	2	2	3	3	2	5
169	172	6	2	54	73	42	83	64	25	1	2	3	0	2	0

Table 2.2: project data

Chapter 3

Exploratory Data Analysis(EDA)

3.1 What is EDA?

Exploratory Data Analysis (EDA) is a process in data analysis that involves exploring and examining datasets to gain a better understanding of their main characteristics. It focuses on summarizing the main features of the data, uncovering patterns, and identifying potential relationships between variables. EDA typically involves techniques such as data visualization, descriptive statistics, and basic statistical tests. Its goal is to generate insights, discover hidden patterns or trends, and formulate hypotheses that can guide further analysis or modeling. EDA is often the first step in the data analysis process and helps analysts make informed decisions about data preprocessing, feature engineering, and modeling strategies. We had used following plots in our project.

3.2 Objective of the EDA

The objective of the EDA is to uncover answers to the following questions:

- Descriptive statistics of runs and wickets in different phases of match .
- How many matches have been played in IPL so far by season?
- How many matches has the teams played in all seasons?
- What is the overall win and loss probability of each team?
- What is the total number of matches the team has won or lost throughout the season?
- What is the win and loss probability team by team?
- What is the Win-Loss ratio and winning streaks of each team?
- What are average runs and wickets at each venue?
- What is the minimum 1st innings score for 95% winning probability by venue?
- Top 10 run scorer and top 10 wicket taker against each team.

3.3 Descriptive Statistics

3.3.1 Descriptive statistics of runs

Stats	Total Runs 1st Inns	Total Runs 2nd Inns	Power Play Runs 1st Inns	Middle Overs Runs 1st Inns	Death Overs Runs 1st Inns	Power Play Runs 2nd Inns	Middle Overs Runs 2nd Inns	Death Overs Runs 2nd Inns
Count	1024	1024	1024	1024	1024	1024	1024	1024
Mean	163.88	150.08	52.25	71.10	40.53	53.79	68.42	27.87
SD	30.90	31.97	12.43	17.52	14.12	13.66	18.92	16.32
Min	67	49	20	16	0	17	0	0
25%	144	133	44	59	32	44	58	17
50%	164	152	52	70	41	53	69	29
75%	185	171	60	82	49	62	79.25	39
Max	263	226	96	144	89	107	142	79

Table 3.1: Descriptive statistics of runs in different phases

3.3.2 Descriptive statistics of wickets

Stats	Total Wkts 1st Inns	Total Wkts 2nd Inns	Power Play Wkts 1st Inns	Middle Overs Wkts 1st Inns	Death Overs Wkts 1st Inns	Power Play Wkts 2nd Inns	Middle Overs Wkts 2nd Inns	Death Overs Wkts 2nd Inns
Count	1024	1024	1024	1024	1024	1024	1024	1024
Mean	6.09	5.66	1.57	2.34	2.18	1.67	2.46	1.53
SD	2.11	2.75	1.12	1.28	1.41	1.22	1.54	1.51
Min	0	0	0	0	0	0	0	0
25%	4	4	1	1	1	1	1	0
50%	6	6	1	2	2	2	2	1
75%	8	8	2	3	3	2	3	2
Max	10	10	7	7	7	6	7	7

Table 3.2: Descriptive statistics of wickets in different phases

◇ Interpretation

- The descriptive statistics above display the average runs and wickets for different phases of the 1st and 2nd innings, with some noticeable variation.

3.3.3 Correlation between runs and wickets

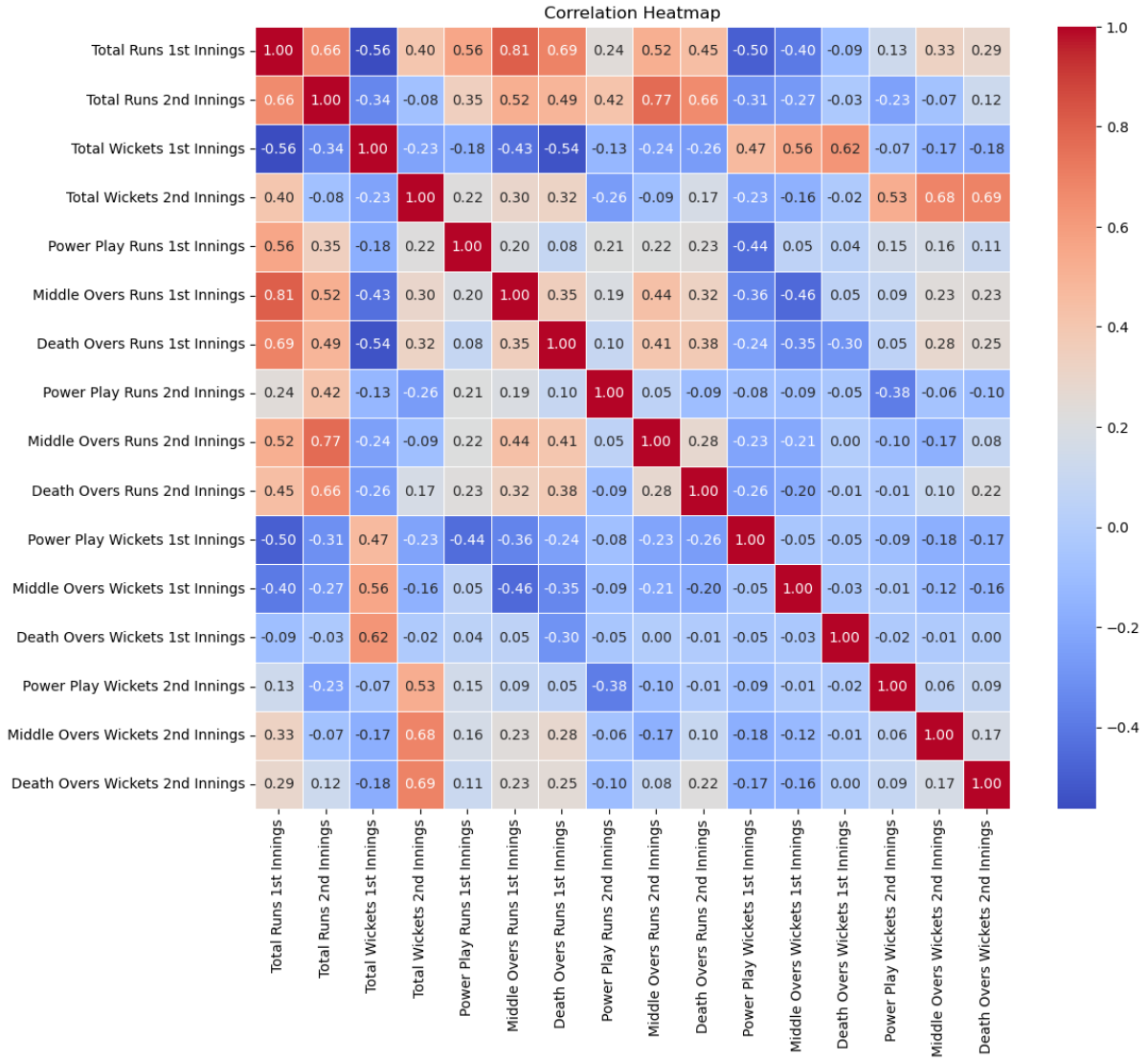


Figure 3.1: Count of final played by teams and win

◇ Interpretation

- The correlation below heatmap provides insights into the relationships between different phases of innings in cricket matches.
- **Runs Correlation:** 1st Innings Total Runs, Highly correlated with runs scored in the middle overs (0.81) and death overs (0.69). Indicates that scoring heavily in these phases is crucial for a high total. 2nd Innings Total Runs, Strong correlation with runs in the death overs (0.77) and middle overs (0.52). Suggests that finishing strong in the latter phases is important for chasing.
- **Wickets Correlation:** 1st Innings Wickets, Negatively correlated with runs across all phases, especially middle overs (-0.56), indicating that losing wickets hampers run accumulation. 2nd Innings Wickets, Shows similar trends, with higher negative correlations in the middle (-0.53) and death overs (-0.60).

3.4 Team Wise Analysis

3.4.1 Matches Played by Season and Teams Short Form

Season	Matches
IPL-2008	58
IPL-2009	57
IPL-2010	60
IPL-2011	72
IPL-2012	74
IPL-2013	76
IPL-2014	60
IPL-2015	57
IPL-2016	60
IPL-2017	59
IPL-2018	60
IPL-2019	59
IPL-2020	60
IPL-2021	60
IPL-2022	74
IPL-2023	73
Grand Total	**1032**

Figure 3.2: Matches Played by Seasons

Team	Short form Team
Delhi Daredevils	DC
Pune Warriors	PW
Rajasthan Royals	RR
Gujarat Lions	GL
Delhi Capitals	DC
Kings XI Punjab	PK
Lucknow Super Giants	LSG
Chennai Super Kings	CSK
Gujarat Titans	GT
Kolkata Knight Riders	KKR
Punjab Kings	PK
Rising Pune Supergiants	RPS
Mumbai Indians	MI
Deccan Charges	SRH
Sunrisers Hyderabad	SRH
Royal Challengers Bangalore	RCB
Kochi Tuskers Kerala	KTK

Figure 3.3: Teams Short Form

◇ **Interpretation**

- A total of 1032 matches have been played across all seasons.
- In 2013, "Deccan Chargers" changed its name to "Sunrisers Hyderabad".
- In 2017, "Pune Warriors" changed its name to "Rising Pune Supergiant".
- In 2019, "Delhi Daredevils" changed its name to "Delhi Capitals".
- In 2021, "Kings XI Punjab" changed its name to "Punjab Kings".

In all analyses, we exclusively utilize the most recent team names.

3.4.2 Matches Played Team by Team

Team	CSK	PW	RPS	SRH	DC	RR	KTK	GL	RCB	LSG	GT	KKR	PK	MI	Total
CSK	0	6	0	29	29	28	2	0	30	3	5	28	28	36	224
PW	6	0	0	6	6	5	1	0	5	0	0	5	6	6	46
RPS	0	0	0	4	4	0	0	4	4	0	0	4	4	6	30
SRH	29	6	4	0	34	27	1	5	33	3	3	34	31	31	241
DC	29	6	4	34	0	27	2	4	30	3	3	31	32	33	238
RR	28	5	0	27	27	0	2	0	29	3	5	27	26	27	206
KTK	2	1	0	1	2	2	0	0	2	0	0	2	1	1	14
GL	0	0	4	5	4	0	0	0	5	0	0	4	4	4	30
RCB	30	5	4	33	30	29	2	5	0	4	3	32	31	32	240
LSG	3	0	0	3	3	3	0	0	4	0	4	3	3	4	30
GT	5	0	0	3	3	5	0	0	3	4	0	3	3	4	33
KKR	28	5	4	34	31	27	2	4	32	3	3	0	32	32	237
PK	28	6	4	31	32	26	1	4	31	3	3	32	0	31	232
MI	36	6	6	31	33	27	1	4	32	4	4	32	31	0	247
Total	224	46	30	241	238	206	14	30	240	30	33	237	232	247	2048

Table 3.3: Total matches played in All seasons

◇ Interpretation

The **"Mumbai Indians"** have played the highest number of matches, totaling 247, participating in every season and featuring in the majority of playoffs matches throughout history. In contrast, the **"Kochi Tuskers Kerala"** only played 14 matches, as they were part of just one season in 2011.

3.4.3 Overall Win and Loss probability of each team

- Winning Probability of team = Number of win matches/ Total matches of team
- Losing Probability of team = Number of loss matches/ Total matches of team

Team	MI	SRH	RCB	DC	KKR	PK	CSK	RR	PW	GT	GL	RPS	LSG	KTK	Tie
Total Matches	247	241	240	238	237	232	224	206	46	33	30	30	30	14	0
Wins	140	108	116	108	120	107	131	103	12	23	13	15	17	6	5
Winning Probability	0.57	0.45	0.48	0.45	0.51	0.46	0.58	0.50	0.26	0.70	0.43	0.50	0.57	0.43	inf
Losses	107	133	124	130	117	125	93	103	34	10	17	15	13	8	0
Losing Probability	0.43	0.55	0.52	0.55	0.49	0.54	0.42	0.50	0.74	0.30	0.57	0.50	0.43	0.57	

Table 3.4: Overall Win and Loss probability of each team?

- **Gujrat Titans** has the highest winning probability i.e 70% while **Pune Warriors** has Lowest i.e 0.26%

3.4.4 Matches win/loss team by team

Team		Number of loss matches														Total win matches
		RCB	PK	DC	MI	KKR	RR	SRH	CSK	KTK	PW	GL	RPS	LSG	GT	
Number of win matches	RCB	0	14	18	14	14	15	15	10	2	5	3	2	3	1	116
	PK	17	0	16	15	11	11	14	13	1	3	2	2	1	1	107
	DC	11	16	0	15	15	13	18	10	1	3	3	2	0	1	108
	MI	18	16	18	0	23	15	18	20	0	5	2	2	1	2	140
	KKR	18	21	16	9	0	14	23	10	0	4	1	3	0	1	120
	RR	12	15	14	12	13	0	16	13	1	4	0	0	2	1	103
	SRH	18	17	16	13	11	11	0	9	1	5	5	1	0	1	108
	CSK	20	15	19	16	18	15	20	0	1	4	0	0	1	2	131
	KTK	0	0	1	1	2	1	0	1	0	0	0	0	0	0	6
	PW	0	3	2	1	1	1	1	2	1	0	0	0	0	0	12
	GL	2	2	1	2	3	0	0	0	0	0	0	3	0	0	13
	RPS	2	2	2	4	1	0	3	0	0	0	1	0	0	0	15
	LSG	1	2	3	3	3	1	3	1	0	0	0	0	0	0	17
	GT	2	2	2	2	2	4	2	3	0	0	0	0	4	0	23
	Total loss matches	121	125	128	107	117	101	133	92	8	33	17	15	12	10	1019

Table 3.5: Total win/loss in All seasons

◇ **Interpretation:**

- **for win matches read row wise:**

  win matches of RCB with PK, DC, MI... is 14, 18, 14...

- **for loss matches read column wise:**

  loss matches of RCB with PK, DC, MI... is 17, 11, 18...

- **Mumbai Indians** win the most matches, with **Chennai Super Kings** coming in second, suggesting that these teams consistently perform at their best across all seasons.

- **Sunrisers Hyderabad** have lost the most matches, with **Delhi Capitals** coming in second, indicating their consistent lowest performance across all seasons.

3.4.5 Win and Loss probability team by team?

Teams		Probability Of Winning												
		CSK	DC	GL	GT	KKR	KTK	LSG	MI	PK	PW	RCB	RPS	RR	SRH
Probability of Losing	CSK	0.00	0.34	0.00	0.60	0.36	0.50	0.33	0.56	0.46	0.33	0.33	0.00	0.46	0.31
	DC	0.66	0.00	0.25	0.67	0.52	0.50	1.00	0.55	0.50	0.33	0.60	0.50	0.52	0.47
	GL	0.00	0.75	0.00	0.00	0.25	0.00	0.00	0.50	0.50	0.00	0.60	0.25	0.00	1.00
	GT	0.40	0.33	0.00	0.00	0.33	0.00	0.00	0.50	0.33	0.00	0.33	0.00	0.20	0.33
	KKR	0.64	0.48	0.75	0.67	0.00	1.00	1.00	0.72	0.34	0.20	0.44	0.25	0.48	0.32
	KTK	0.50	0.50	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.50	1.00
	LSG	0.33	0.00	0.00	1.00	0.00	0.00	0.00	0.25	0.33	0.00	0.75	0.00	0.67	0.00
	MI	0.44	0.45	0.50	0.50	0.28	1.00	0.75	0.00	0.48	0.17	0.44	0.67	0.44	0.42
	PK	0.54	0.50	0.50	0.67	0.66	0.00	0.67	0.52	0.00	0.50	0.45	0.50	0.58	0.55
	PW	0.67	0.50	0.00	0.00	0.80	0.00	0.00	0.83	0.50	0.00	1.00	0.00	0.80	0.83
	RCB	0.67	0.37	0.40	0.67	0.56	0.00	0.25	0.56	0.55	0.00	0.00	0.50	0.41	0.55
	RPS	0.00	0.50	0.75	0.00	0.75	0.00	0.00	0.33	0.50	0.00	0.50	0.00	0.00	0.25
	RR	0.54	0.48	0.00	0.80	0.52	0.50	0.33	0.56	0.42	0.20	0.52	0.00	0.00	0.41
	SRH	0.69	0.53	0.00	0.67	0.68	0.00	1.00	0.58	0.45	0.17	0.45	0.75	0.59	0.00

Table 3.6: Rounded Valu

◇ Interpretation

- **For winning probability read column wise:**

  Probability of win matches of CSK with DC, GL, GT... is 0.6552 , 0, 0.4...

- **For losing probability read row wise:**

  Probability of loss matches of CSK with DC, GL, GT... is 0.3448, 0, 0.6...

- The red color highlights the highest probabilities of winning and losing of each team against other team.
- A probability of 1 for winning or losing for corresponding teams indicates that those teams have won or lost every match against each other.
- A win probability of 1 for GT against LSG indicates that GT has won every match against LSG, while a loss probability of 1 for LSG against GT indicates that LSG has lost every match against GT.

3.4.6 Win-Loss ratio and winning streaks of each team

- **Win-Loss Ratios:** This ratio provides a measure of a team's success by comparing the number of matches won to the number of matches lost. It's calculated as:

  Win-Loss Ratio = (Number of Matches Won) / (Number of Matches Lost)

  A ratio greater than 1 indicates that the team has won more matches than it has lost, while a ratio less than 1 indicates the opposite.

- **Winning Streaks:** A winning streak occurs when a team wins consecutive matches without any losses in between. By identifying these streaks, you can understand periods of dominance or exceptional performance by a team.

Team	Win-Loss Ratio	Winning Streaks
PW	0.352941	[2, 2, 2, 1, 1, 1, 1, 2]
RPS	1	[1, 1, 1, 3, 3, 4, 2]
MI	1.308411	[6, 2, 1, 1, 2, 2, 5, 3, 1, 2, 3,3, 2, 1, 2, 1, 3, 2, 1, 3, 3, 5, 2, 2, 1, 4, 1, 5, 4, 1, 1, 3, 1, 1, 6, 3, 1,2, 1, 1, 3, 1, 1, 3, 2, 1, 4, 1, 5, 1, 2, 2, 2, 2, 1, 2, 2, 1, 1, 3, 2, 2, 2]
RR	1	[5, 6, 2, 1, 1, 3, 1, 4, 2, 2, 3,3, 2, 2, 1, 2, 2, 2, 4, 1, 1, 4, 1, 1, 5, 1, 1, 2, 1, 3, 1, 1, 1, 1, 2, 2, 1,1, 2, 1, 1, 2, 1, 2, 1, 3, 1, 2, 1, 1, 3, 1, 1, 1]
KTK	0.75	[3, 2, 1]
GL	0.764706	[3, 3, 1, 2, 1, 1, 1, 1]
PK	0.856	[5, 4, 1, 3, 1, 1, 2, 1, 1, 2, 3,4, 2, 1, 2, 2, 1, 1, 1, 2, 1, 8, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2,1, 4, 1, 1, 2, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1]
RCB	0.935484	[1, 1, 2, 1, 3, 5, 4, 1, 1, 1, 2,7, 1, 1, 1, 3, 3, 1, 1, 2, 3, 1, 1, 3, 1, 2, 1, 2, 1, 2, 1, 1, 1, 1, 2, 5, 1,1, 1, 1, 1, 1, 3, 1, 3, 2, 2, 2, 2, 4, 1, 3, 1, 3, 2, 2, 2, 1, 1, 2, 1, 2]
LSG	1.307692	[3, 1, 4, 1, 1, 2, 1, 1, 3]
CSK	1.408602	[4, 2, 1, 2, 1, 5, 1, 1, 2, 3, 1,4, 1, 4, 3, 2, 1, 1, 2, 1, 3, 2, 2, 7, 1, 1, 1, 6, 2, 2, 3, 3, 1, 1, 1, 1, 2,3, 1, 1, 1, 6, 4, 1, 1, 1, 1, 1, 1, 3, 5, 4, 2, 1, 1, 1, 1, 2, 3, 2, 3]
GT	2.3	[3, 5, 2, 4, 1, 3, 2, 2, 1]
KKR	1.025641	[2, 3, 1, 1, 4, 1, 1, 1, 2, 3, 3,1, 1, 2, 6, 5, 1, 1, 1, 2, 1, 1, 10, 2, 1, 3, 1, 3, 2, 1, 1, 1, 3, 3, 1, 1,1, 2, 2, 4, 2, 2, 2, 2, 2, 1, 1, 2, 1, 2, 1, 4, 1, 2, 1, 2, 2, 1, 2, 1]
DC	0.830769	[2, 2, 1, 2, 3, 1, 4, 1, 1, 2, 4,1, 1, 1, 1, 1, 1, 3, 4, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 3, 2, 1, 1, 2, 2, 2,1, 1, 1, 3, 1, 3, 3, 2, 2, 3, 2, 1, 1, 1, 3, 4, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1]
SRH	0.81203	[1, 1, 4, 1, 1, 1, 2, 3, 5, 1, 1,1, 3, 1, 1, 4, 1, 2, 2, 1, 2, 1, 1, 2, 2, 1, 1, 2, 3, 3, 4, 1, 5, 2, 2, 2, 3,6, 1, 3, 2, 1, 2, 1, 1, 4, 1, 1, 1, 5, 1, 2, 1, 1]

Table 3.7: Win-Loss Ratio and Winning Streaks of Each Team

◇ Interpretation

- **Win-Loss Ratio :** The win-loss ratio for Kolkata Knight Riders is **1.025641.**
  This means that, on average, they have won slightly more matches than they have lost. Since the ratio is greater than 1, it indicates that Kolkata Knight Riders have won more matches than they have lost over the analyzed period.

- **Winning Streaks :** The winning streaks for Kolkata Knight Riders are listed as a sequence of numbers:

  {2, 3, 1, 1, 4, 1, 1, 1, 2, 3, 3, 1, 1, 2, 6, . . . }
  Each number represents the length of a winning streak.
  For example, the number 2 indicates a winning streak of 2 matches, the number 3 indicates a winning streak of 3 matches, and so on.

3.4.7 Count of win matches by toss decision

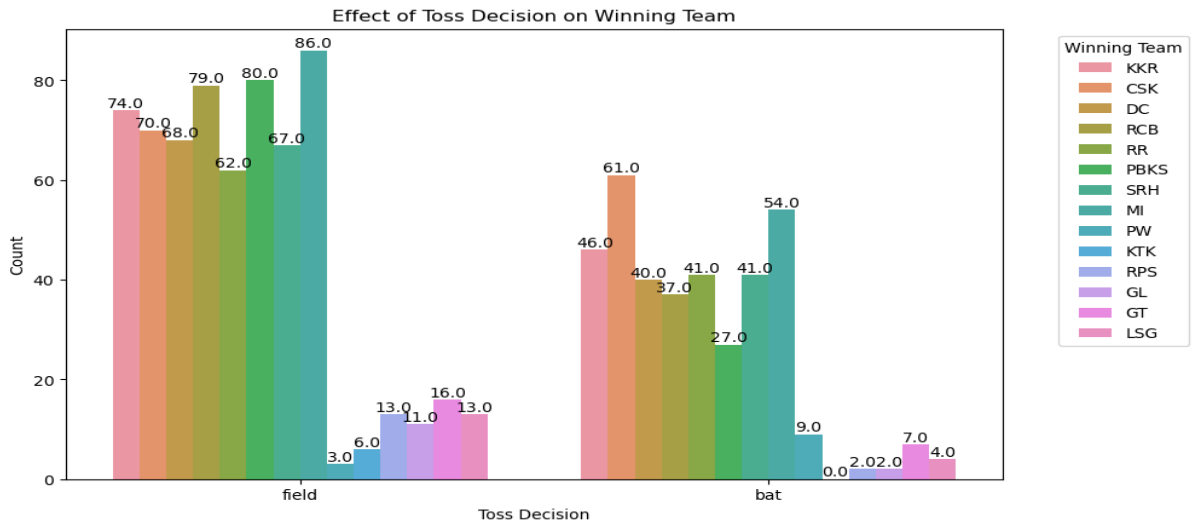


Figure 3.4: Count of win matches by toss decision

◇ Interpretation

- It's evident that most teams have won more matches when opting to field first. Therefore, choosing to field after winning the toss significantly enhances a team's likelihood of winning.

3.4.8 Count of final played by teams and win

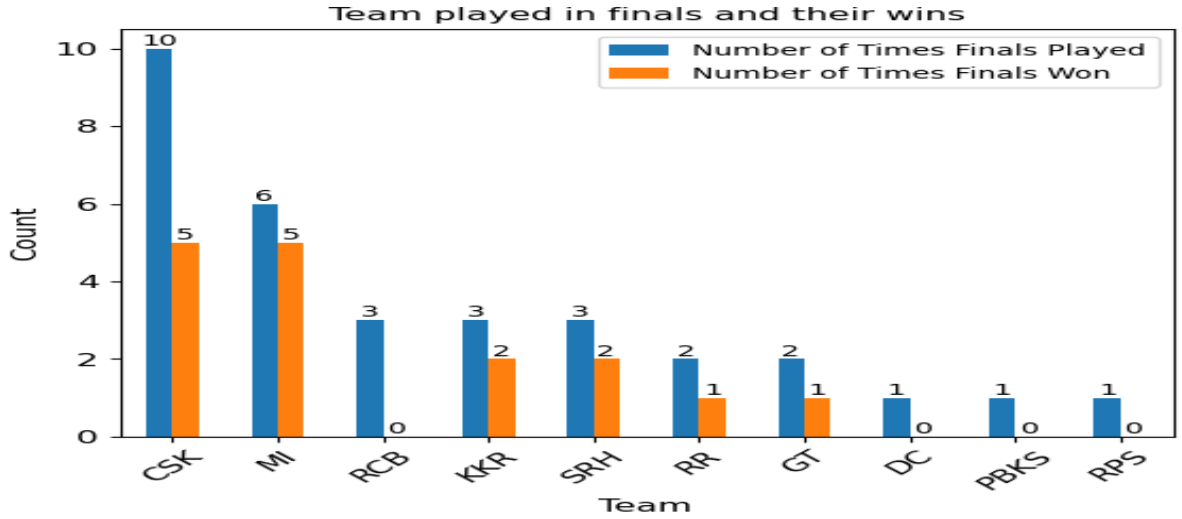


Figure 3.5: Count of final played by teams and win

◇ Interpretation

- **CSK** has appeared in 10 finals, winning half of them, showcasing consistent success in reaching this stage. **MI** boasts an impressive conversion rate, winning 5 out of 6 finals, underscoring their dominance in crucial matches.

3.5 Venue Wise Analysis

3.5.1 Average runs in 1st and 2nd innings

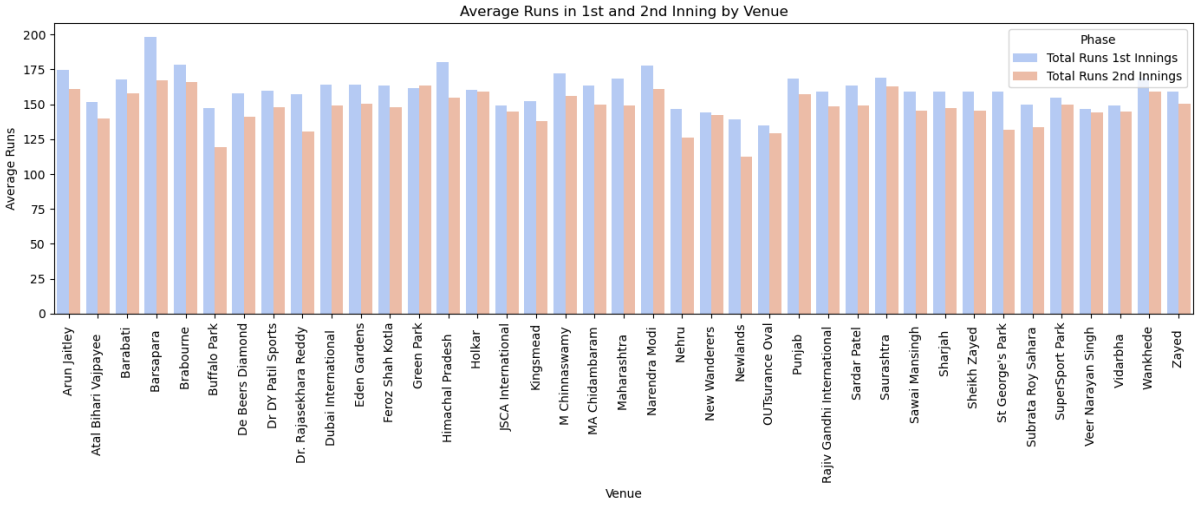


Figure 3.6: Average Runs in 1st and 2nd Innings by Venue

◇ Interpretation

- The average runs scored in the first innings are consistently higher than those scored in the second innings at each venue.

3.5.2 Average wickets in 1st and 2nd innings

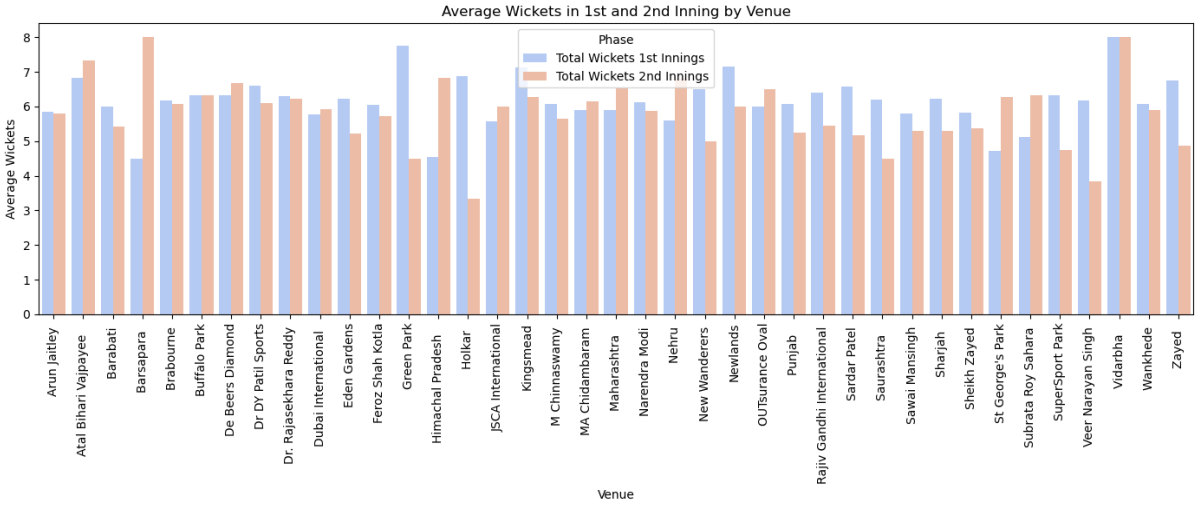


Figure 3.7: Average Wickets in 1st and 2nd Innings by Venue

◇ Interpretation

- The average wickets in both the first and second innings exhibit moderate behavior at each venue.

3.5.3 Average Runs in Different Phases of 1st Inning

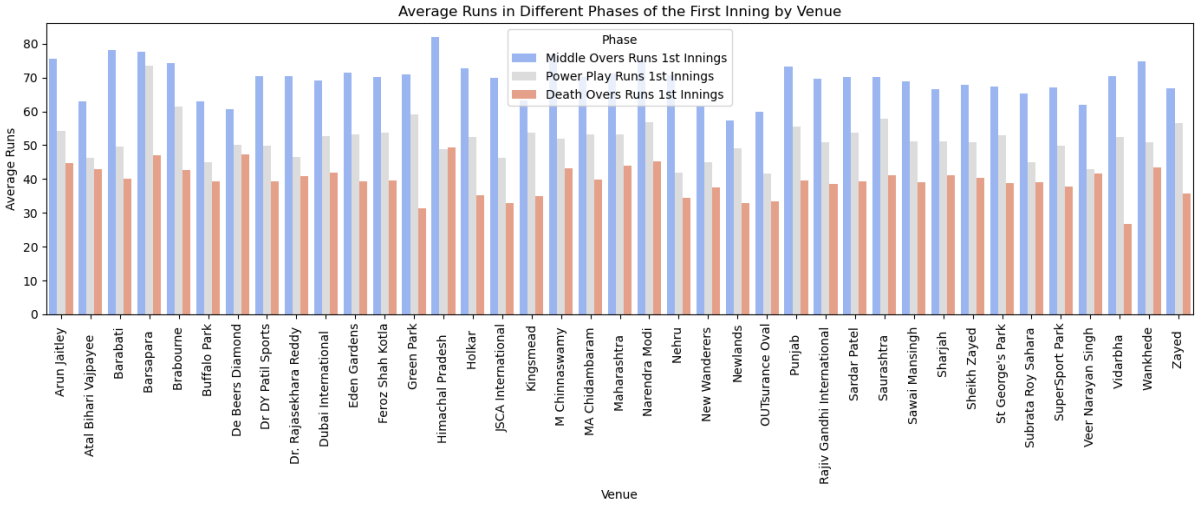


Figure 3.8: Average Runs in Different Phases of the First Inning by Venue

◇ Interpretation

- The highest average runs in 1st innings are scored during the middle overs, followed by the power play, with the lowest average runs in the death overs at each venue. This is because the middle overs consist of 9 overs, while the death overs have only 5 overs.

3.5.4 Average runs in different phases of 2nd inning

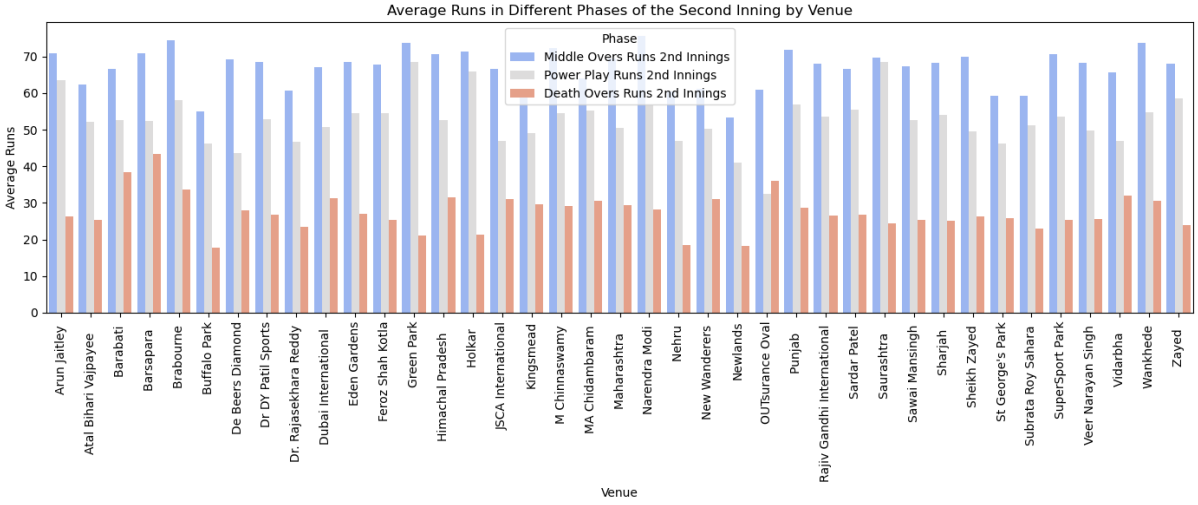


Figure 3.9: Average Runs in Different Phases of the Second Inning by Venue

◇ Interpretation

- This pattern is consistently observed in the second innings runs in different phases across all venues as well.

3.5.5 Average wickets in different phases of 1st inning

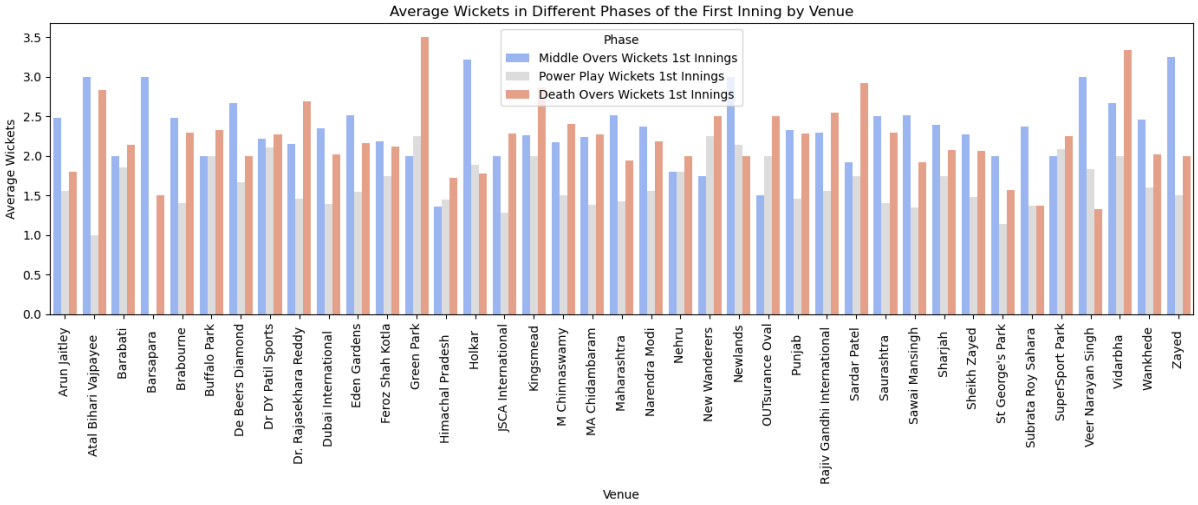


Figure 3.10: Average Wickets in Different Phases of the First Inning by Venue

◇ Interpretation

- At each venue, there is a consistent trend where the maximum average number of wickets falls during the middle overs and death overs in the first innings, attributed to the heightened pressure during these phases, while the fewest wickets fall during the power play.

3.5.6 Average wickets in different phases of 2nd inning

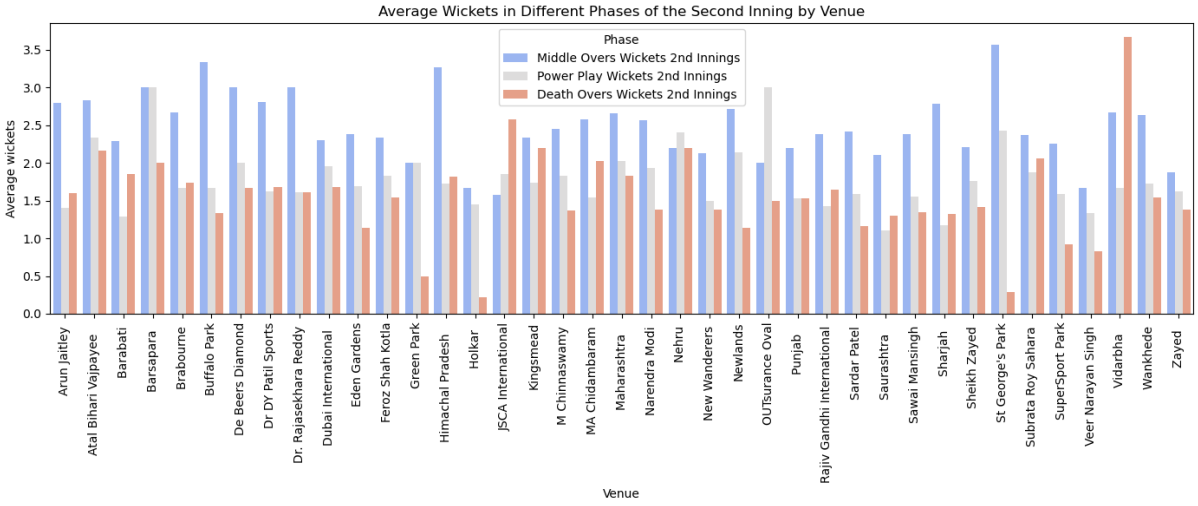


Figure 3.11: Average Wickets in Different Phases of the Second Inning by Venue

◇ Interpretation

- In the second innings at each venue, the maximum average number of wickets falls during the middle overs, followed by the death overs, with fewest in power play.

3.5.7 Minimum 1st innings score for 95% winning probability by venue

Venue	Min Score for 95% Probability of Winning
Arun Jaitley Stadium	221
Barabati Stadium	232
Barsapara Cricket Stadium	200
Shri Atal Bihari Stadium	194
Brabourne Stadium	218
Buffalo Park	117
De Beers Diamond Oval	141
Dr DY Patil Sports Academy	217
Dr. Y.S. Rajasekhara Reddy Stadium	207
Dubai International Cricket Stadium	220
Eden Gardens	236
Feroz Shah Kotla	232
Green Park	173
Himachal Pradesh Association	214
Holkar Cricket Stadium	246
JSCA International Stadium Complex	149
Kingsmead	169
M Chinnaswamy Stadium	264
MA Chidambaram Stadium	218
Maharashtra Cricket Association Stadium	212
Narendra Modi Stadium	234
Nehru Stadium	162
New Wanderers Stadium	164
Newlands	105
OUTsurance Oval	120
Punjab Cricket Association Stadium	258
Rajiv Gandhi International Stadium	232
Sardar Patel Stadium	202
Saurashtra Cricket Association Stadium	181
Sawai Mansingh Stadium	215
Shaheed Veer Narayan Singh Stadium	165
Sharjah Cricket Stadium	229
Sheikh Zayed Stadium	206
St George's Park	154
Subrata Roy Sahara Stadium	156
SuperSport Park	189
Vidarbha Cricket Association Stadium	152
Wankhede Stadium	223
Zayed Cricket Stadium	172

Table 3.8: Minimum 1st innings score for 95% winning probability by venue

3.5.8 Number of matches at each venue

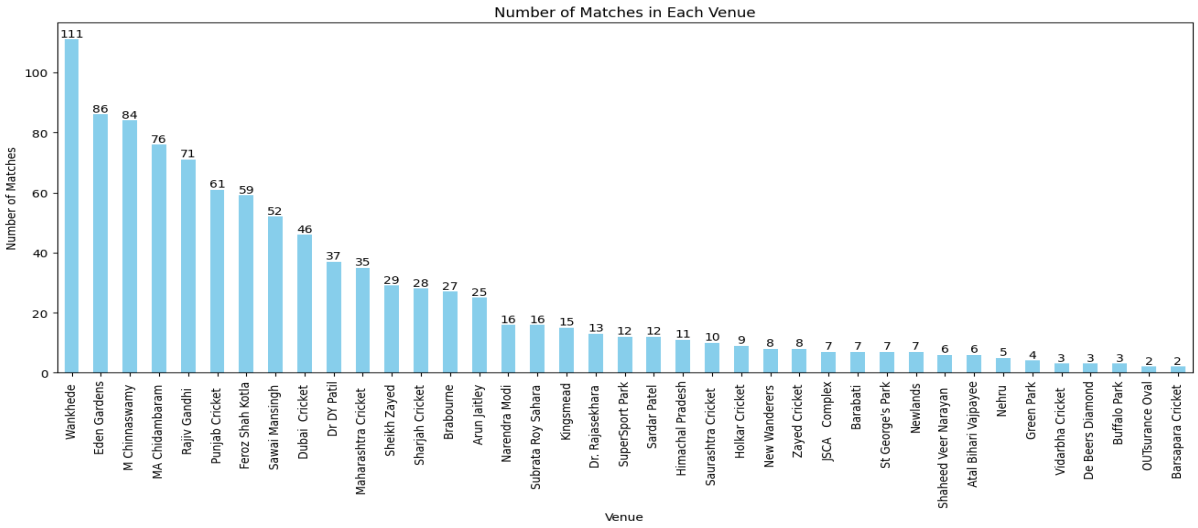


Figure 3.12: Number of matches played at each venue

◇ Interpretation

- Wankhede Stadium has hosted the highest number of matches, with a total of 111 games played, followed by Eden Gardens, which has hosted 86 matches, the second most

3.5.9 Count of match wins by toss decision at each venue

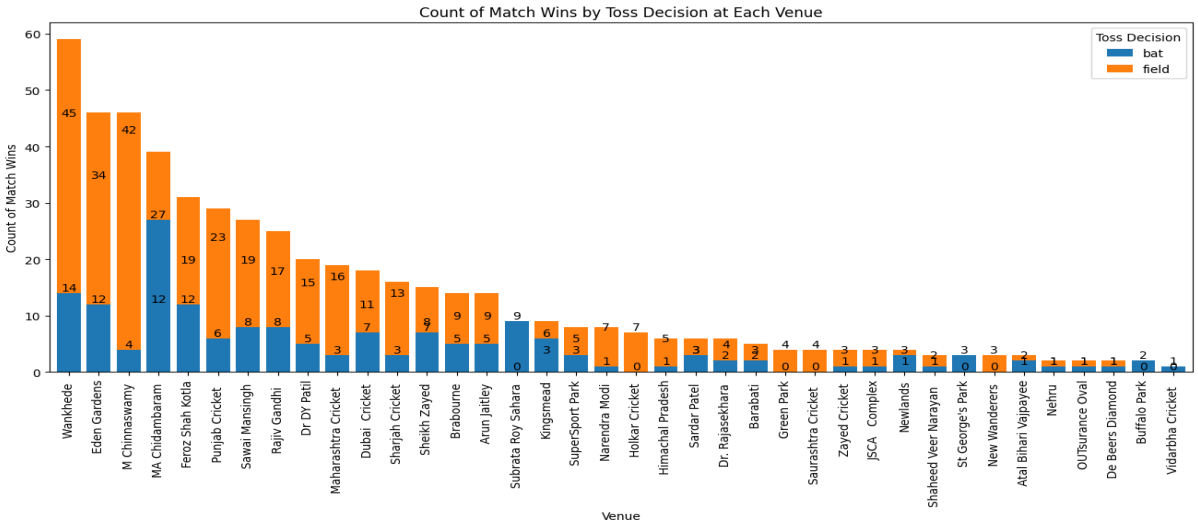


Figure 3.13: Count of match wins by toss decision at each venue

◇ Interpretation

- With the exception of MA Chidambaram Stadium, most venues exhibit a chasing-friendly nature.

3.6 Players Statistics

3.6.1 Top 10 run scorer against each team

Player	KKR	Player	RCB	Player	CSK	Player	PK
DA Warner	1075	DA Warner	861	S Dhawan	1057	DA Warner	1105
RG Sharma	1040	MS Dhoni	839	V Kohli	985	S Dhawan	894
S Dhawan	907	RG Sharma	793	RG Sharma	791	V Kohli	861
V Kohli	861	AT Rayudu	728	KD Karthik	675	SK Raina	830
SK Raina	829	SK Raina	702	DA Warner	644	RG Sharma	814
CH Gayle	700	RV Uthappa	687	KA Pollard	583	CH Gayle	797
AM Rahane	565	AM Rahane	685	RV Uthappa	547	F du Plessis	796
MS Dhoni	555	S Dhawan	679	AB de Villiers	532	RV Uthappa	772
AB de Villiers	522	G Gambhir	647	SR Watson	480	AB de Villiers	736
SV Samson	517	KL Rahul	628	KL Rahul	455	G Gambhir	730

Player	RR	Player	DC	Player	MI	Player	SRH
S Dhawan	679	V Kohli	1030	S Dhawan	901	V Kohli	980
AB de Villiers	652	RG Sharma	977	KL Rahul	871	SV Samson	791
KD Karthik	630	AM Rahane	813	V Kohli	857	MS Dhoni	769
SK Raina	630	RV Uthappa	742	SK Raina	824	SK Raina	763
V Kohli	618	MS Dhoni	672	AB de Villiers	791	AB de Villiers	693
KL Rahul	579	SK Raina	661	MS Dhoni	748	RV Uthappa	662
MS Dhoni	553	AT Rayudu	629	MK Pandey	736	AT Rayudu	657
CH Gayle	552	AB de Villiers	588	DA Warner	717	SR Watson	646
RG Sharma	541	S Dhawan	557	CH Gayle	715	YK Pathan	598
DA Warner	534	KD Karthik	543	AM Rahane	707	RG Sharma	584

Player	KTK	Player	PW	Player	RPS	Player	GL
SR Tendulkar	100	CH Gayle	383	V Kohli	271	DA Warner	336
V Sehwag	95	V Sehwag	220	BB McCullum	186	V Kohli	283
EJG Morgan	76	S Badrinath	199	RG Sharma	178	AB de Villiers	233
KD Karthik	69	G Gambhir	182	S Dhawan	138	RR Pant	190
SK Raina	69	AM Rahane	173	KK Nair	137	HM Amla	169
SR Watson	69	R Dravid	165	SV Samson	134	SPD Smith	153
KC Sangakkara	65	SR Watson	165	PA Patel	129	N Rana	142
AB de Villiers	54	CL White	162	AB de Villiers	116	BA Stokes	128
AT Rayudu	53	SR Tendulkar	152	M Vijay	112	S Dhawan	127
TM Dilshan	53	AB de Villiers	149	DR Smith	110	G Gambhir	122

Player	LSG	Player	GT
F du Plessis	219	RD Gaikwad	304
Shubman Gill	157	V Kohli	232
HH Pandya	135	SA Yadav	200
WP Saha	133	JC Buttler	190
V Kohli	117	SV Samson	162
RK Singh	113	Abhishek Sharma	112
RM Patidar	112	VR Iyer	111
RA Tripathi	98	SO Hetmyer	107
GJ Maxwell	95	S Dhawan	105
Ishan Kishan	95	RK Singh	102

Table 3.9: Top 10 run scorer against each team

3.6.2 Top 10 wicket taker against each team

Player	KKR	Player	RCB	Player	CSK	Player	PK
B Kumar	33	Sandeep Sharma	30	SL Malinga	37	SP Narine	36
YS Chahal	28	JJ Bumrah	29	Harbhajan Singh	24	UT Yadav	36
DJ Bravo	25	RA Jadeja	28	SP Narine	24	DJ Bravo	31
R Ashwin	24	Harbhajan Singh	25	PP Ojha	23	YS Chahal	31
R Vinay Kumar	24	A Nehra	24	PP Chawla	22	B Kumar	28
CH Morris	23	DJ Bravo	24	R Vinay Kumar	18	SL Malinga	27
SL Malinga	23	SP Narine	23	YS Chahal	18	Harbhajan Singh	24
JJ Bumrah	21	B Kumar	22	P Kumar	17	A Mishra	23
Z Khan	21	R Ashwin	22	UT Yadav	17	JJ Bumrah	23
A Mishra	20	A Mishra	19	Z Khan	16	R Ashwin	23

Player	RR	Player	DC	Player	MI	Player	SRH
A Mishra	33	R Ashwin	29	DJ Bravo	36	SL Malinga	33
PP Chawla	23	SP Narine	29	MM Sharma	33	DJ Bravo	32
RA Jadeja	22	PP Chawla	27	PP Chawla	28	YS Chahal	28
Mohammed Shami	21	Harbhajan Singh	25	R Ashwin	27	A Mishra	24
R Ashwin	20	JJ Bumrah	25	YS Chahal	26	PP Chawla	22
HV Patel	18	DJ Bravo	24	A Mishra	25	R Bhatia	20
JJ Bumrah	18	RA Jadeja	23	SP Narine	25	UT Yadav	20
YS Chahal	18	SL Malinga	23	B Kumar	22	AD Russell	19
DJ Bravo	17	Rashid Khan	22	HV Patel	22	MM Sharma	19
B Kumar	16	Sandeep Sharma	21	Rashid Khan	21	Harbhajan Singh	18

Player	KTK	Player	PW	Player	RPS	Player	GL
I Sharma	5	SL Malinga	12	JJ Bumrah	8	B Kumar	12
RE van der Merwe	5	R Vinay Kumar	11	A Nehra	7	Imran Tahir	8
M Morkel	4	A Mishra	10	SR Watson	6	YS Chahal	8
SK Trivedi	4	MM Patel	8	AJ Tye	5	CJ Jordan	7
WD Parnell	4	PP Ojha	8	MG Johnson	5	AR Patel	6
YK Pathan	4	DE Bollinger	7	MM Sharma	5	JJ Bumrah	6
DE Bollinger	3	Harbhajan Singh	7	Sandeep Sharma	5	MJ McClenaghan	6
DL Vettori	3	L Balaji	7	Z Khan	5	Rashid Khan	6
DW Steyn	3	AD Mascarenhas	6	A Mishra	4	Sandeep Sharma	6
R Ashwin	3	NLTC Perera	6	B Kumar	4	Mohammed Shami	5

Player	LSG	Player	GT
JR Hazlewood	9	B Kumar	8
K Rabada	8	K Rabada	7
MM Sharma	8	M Pathirana	7
Rashid Khan	8	Umran Malik	6
HV Patel	7	AD Russell	5
Akash Madhwal	6	Akash Madhwal	5
MM Ali	6	KK Ahmed	5
Mohammed Siraj	6	PP Chawla	5
SN Thakur	6	RA Jadeja	5
Mohammed Shami	5	Avesh Khan	4

Table 3.10: Top 10 wicket taker against each team

3.6.3 Best batsman and bowler by season

◇ **Orange cap holders**

Season	Batsman	Total_Run
2008	SE Marsh	644
2009	ML Hayden	626
2010	SR Tendulkar	664
2011	CH Gayle	638
2012	CH Gayle	762
2013	CH Gayle	778
2014	RV Uthappa	681
2015	DA Warner	589
2016	V Kohli	998
2017	DA Warner	670
2018	KS Williamson	747
2019	DA Warner	727
2020	KL Rahul	696
2021	RD Gaikwad	658
2022	JC Buttler	894
2023	Shubman Gill	923

Table 3.11: Orange cap holders

◇ **Purple cap holders**

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

Table 3.12: Purple cap holders

◇ **Interpretation**

- The red colors highlight the extremes in runs and wickets achieved by the purple cap and orange cap holders.

Chapter 4

Statistical Analysis

4.1 Objective of the Analysis

While analyzing IPL data, several questions arise about the dependencies and interrelations between the variables. Specifically, we are interested in the following:

- Is there any significant association between toss decision and match outcome?
- Is there any significant difference between means or medians of runs and wickets of 1st inning and second inning?
- Is there any significant difference between means or medians of groups of runs and wickets in power play, middle overs and death overs of 1st inning and second inning?

To answer these questions we had gone through the following approaches

4.2 Statistical Tools Used

4.2.1 Chi-Square test for independence of attributes

A chi-square test of independence, also known as a chi-square test of association, is used to determine whether two categorical variables are related. If two variables are related, the probability of one variable having a certain value depends on the value of the other variable. The test compares the observed frequencies to the frequencies you would expect if the two variables were unrelated. When the variables are unrelated, the observed and expected frequencies will be similar.

◇ **Hypothesis**

- **Null hypothesis (H0)**: There is no association between the two categorical variables (they are independent).
- **Alternative hypothesis (H1)**: There is an association between the two categorical variables (they are not independent).

◇ **Test statistic formula**

The Chi-Square statistic is calculated using the formula:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Where:

- O_i is the observed frequency.
- E_i is the expected frequency, calculated under the assumption that the null hypothesis is true.

◇ **Degrees of freedom**

The degrees of freedom (df) for the test are calculated as:

$$\text{df} = (r-1) \times (c-1)$$

where r is the number of rows and c is the number of columns in the contingency table.

◇ **Interpretation**

- p-value < 0.05: Reject the null hypothesis, suggesting a significant association between the variables.
- p-value ≥ 0.05: Fail to reject the null hypothesis, suggesting no significant association between the variables.

4.2.2 Anderson-Darling test for normality

The Anderson-Darling test is a statistical test used to assess whether a given sample of data comes from a specified distribution, most commonly a normal distribution. It is an enhancement of the Kolmogorov-Smirnov test and gives more weight to the tails of the distribution, making it particularly sensitive to deviations from normality.

◇ **Hypotheses**

- **Null hypothesis (H0)**: The data follows a normal distribution.
- **Alternative hypothesis (H1)**: The data does not follow a normal distribution.

◇ **Test statistic**

The Anderson-Darling test statistic (A^2) is calculated using the following formula:

$$A^2 = -N - S$$

where:

$$S = \sum_{i=1}^{N} \frac{2i-1}{N} [\ln(F(Y_i)) + \ln(1 - F(Y_{N+1-i}))]$$

and:

- N is the sample size.
- Y_i are the ordered data points.
- F is the cumulative distribution function (CDF) of the specified distribution (for normality, the normal CDF).

◇ **Interpretation**

- Compare the test statistic A^2 with critical values from the Anderson-Darling distribution table for normality.
- If A^2 is greater than the critical value at a given significance level (e.g., 0.05), reject the null hypothesis.

◇ **Decision rule**

- p-value < 0.05: Reject the null hypothesis, indicating that the data does not follow a normal distribution.
- p-value ≥ 0.05: Fail to reject the null hypothesis, suggesting that the data follows a normal distribution.

4.2.3 Wilcoxon signed-rank test

The Wilcoxon signed-rank test is a non-parametric statistical test used to compare two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ. It is the non-parametric alternative to the paired t-test and is used when the differences between pairs are not normally distributed.

◇ **Hypotheses**

- **Null hypothesis (H0)**: The median difference between pairs is zero.
- **Alternative hypothesis (H1)**: The median difference between pairs is not zero.

◇ **Test statistic**

The Wilcoxon signed-rank test statistic is calculated as follows:

- Calculate the differences between each pair of observations.
- Rank the absolute differences, ignoring the signs.
- Assign the signs of the differences to the ranks.

- Calculate the test statistic W as the sum of the ranks of the positive differences (or equivalently, the sum of the ranks of the negative differences, as W and W' are related by $W + W' = \frac{N(N+1)}{2}$, where N is the number of non-zero differences).

$$W = \sum_{i:d_i>0} R_i$$

where:

- d_i is the difference between each pair of observations.
- R_i is the rank of the absolute difference $|d_i|$.

◇ **Interpretation**

- p-value < 0.05: Reject the null hypothesis, indicating that there is a significant difference between the paired samples.
- p-value ≥ 0.05: Fail to reject the null hypothesis, suggesting that there is no significant difference between the paired samples.

4.2.4 Levene's test for eqaulity of variances

Levene's test is a statistical procedure used to assess the equality of variances for a variable across two or more groups. It is important in the context of analysis of variance (ANOVA) and other statistical analyses that assume homogeneity of variances across groups.

◇ **Hypotheses**

- **Null hypothesis (H0)**: The variances are equal across the groups.
- **Alternative hypothesis (H1)**: At least one group's variance is different from the others.

◇ **Test statistic**

Levene's test uses the following formula to calculate the test statistic:

$$W = \frac{(N-k)}{(k-1)} \cdot \frac{\sum_{i=1}^{k} N_i (Z_{i\cdot} - Z_{\cdot\cdot})^2}{\sum_{i=1}^{k} \sum_{j=1}^{N_i} (Z_{ij} - Z_{i\cdot})^2}$$

Where:

- N is the total number of observations.
- k is the number of groups.
- N_i is the number of observations in group i.
- Z_{ij} is the absolute deviation of the j-th observation in group i from the group median.
- $Z_{i\cdot}$ is the mean of Z_{ij} for group i.
- $Z_{\cdot\cdot}$ is the overall mean of Z_{ij}.

◇ **Interpretation**

- p-value < 0.05: Reject the null hypothesis, indicating that there is significant evidence that the variances are different across the groups.
- p-value ≥ 0.05: Fail to reject the null hypothesis, indicating no significant evidence of differences in variances.

4.2.5 Kruskal-Wallis rank rum test

The Kruskal-Wallis test is a non-parametric statistical test used to determine whether there are significant differences between the distributions of three or more independent groups. It is an extension of the Mann-Whitney U test to multiple groups and serves as an alternative to the one-way ANOVA when the assumptions of normality and homogeneity of variances are not met.

The primary purpose of the Kruskal-Wallis test is to compare the medians of three or more independent groups to ascertain if they come from the same distribution. This test is useful when the data are ordinal, not normally distributed, or when the sample sizes are small.

◇ **Hypotheses**

- **Null hypothesis (H0)**: The distributions of the groups are the same (i.e., the medians are equal).
- **Alternative hypothesis (H1)**: At least one group's distribution is different from the others (i.e., at least one median is different).

◇ **Test statistic**

The Kruskal-Wallis test statistic (H) is calculated using the following formula:

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(N+1)$$

where:

- N is the total number of observations across all groups.
- k is the number of groups.
- n_i is the number of observations in the i-th group.
- R_i is the sum of the ranks for the i-th group.

◇ **Degrees of freedom**

The degrees of freedom for the Kruskal-Wallis test are calculated as: df = k - 1

where k is the number of groups.

◇ **Interpretation**

- p-value < 0.05: Reject the null hypothesis, indicating that there is a significant difference in the distributions of at least one pair of groups.
- p-value ≥ 0.05: Fail to reject the null hypothesis, suggesting that there is no significant difference in the distributions of the groups.

4.3 Statistical Analysis

4.3.1 Significant association between toss decision and match outcome

◇ **Chi-Square test for independence**

The Chi-Square test for independence was conducted to determine if there is an association between the toss decision and match outcome.

◇ **Hypotheses**

- H_0**:** There is no association between toss decision and match outcome.
- H_1**:** There is an association between toss decision and match outcome.

◇ **Contingency table**

Decision	Wins	Losses	Total
Bat	170	202	372
Field	353	299	652
Total	523	501	1024

Table 4.1: Observed frequencies

Decision	Wins	Losses
Bat	189.996	182.004
Field	333.004	318.996

Table 4.2: Expected frequencies

◇ **Results**

- **Chi-Square statistic**: 6.4219
- **Degrees of freedom**: 1
- **p-value**: 0.0113

◇ **Interpretation**

- Since the p-value is less than 0.05, we reject the null hypothesis. This indicates that there is a significant association between toss decision and match outcome.
- Means choosing between the toss decision (to bat or field first) will significantly impact the match outcome (winning or losing).

4.3.2 Comparison between more than three group means

For further analysis of comparing means or medians across multiple groups, our variables of interest includes:

'Total Runs 1st Inns', 'Total Runs 2nd Inns', 'Total Wkts 1st Inns', 'Total Wkts 2nd Inns', 'Power Play Runs 1st Inns', 'Middle Overs Runs 1st Inns', 'Death Overs Runs 1st Inns', 'Power Play Runs 2nd Inns', 'Middle Overs Runs 2nd Inns', 'Death Overs Runs 2nd Inns', 'Power Play Wkts 1st Inns', 'Middle Overs Wkts 1st Inns', 'Death Overs Wkts 1st Inns', 'Power Play Wkts 2nd Inns', 'Middle Overs Wkts 2nd Inns', 'Death Overs Wkts 2nd Inns'.

To compare the means of two or more groups, we utilize both parametric and non-parametric approaches. For comparing the means of two groups, the parametric approach is the Paired t-test, while the non-parametric alternative is the Wilcoxon signed-rank test. To compare the means of more than two groups, the parametric method used is Analysis of Variance (ANOVA), with the Kruskal-Wallis rank-sum test serving as the non-parametric alternative.

The choice between these approaches depends on the verification of underlying assumptions. The primary assumptions for selecting parametric methods are normality and equality of variances. To test the assumption of normality, we employ the Anderson-Darling test. To assess the assumption of equality of variances, we use Levene's test. Verifying these assumptions is a crucial step before proceeding with the chosen statistical approach.

Levene's test for equality of variances

◇ **Hypotheses**

- H_0**:** The variances are equal across the groups.
- H_0**:** At least one group's variance is different from the others.

◇ **Levene's test results**

Variable	Statistic	P-value	Equal Variances
Runs 1st Inns	46.82209	9.32E-21	**No**
Runs 2nd Inns	27.8113	1.07E-12	**No**
Wkts 1st Inns	12.34571	4.57E-06	**No**
Wkts 2nd Inns	17.98362	1.72E-08	**No**

◇ **Interpretation**

- Our variable of interest does not meet the assumption of equal variances.

◇ Anderson-Darling test for normality

◇ Hypotheses

- H_0**:** The data follows a normal distribution.
- H_1**:** The data does not follow a normal distribution.

Normality Test Results:

Variables	Anderson-Darling statistic	Anderson-Darling critical values	Is Normally Distributed
Total Runs 1st Inns	0.7455	0.784	**TRUE**
Total Runs 2nd Inns	3.8773	0.784	**FALSE**
Total Wkts 1st Inns	11.9713	0.784	**FALSE**
Total Wkts 2nd Inns	12.6761	0.784	**FALSE**
Power Play Runs 1st Inns	0.9072	0.784	**FALSE**
Middle Overs Runs 1st Inns	1.6005	0.784	**FALSE**
Death Overs Runs 1st Inns	2.623	0.784	**FALSE**
Power Play Runs 2nd Inns	1.2496	0.784	**FALSE**
Middle Overs Runs 2nd Inns	5.5502	0.784	**FALSE**
Death Overs Runs 2nd Inns	8.5553	0.784	**FALSE**
Power Play Wkts 1st Inns	38.9325	0.784	**FALSE**
Middle Overs Wkts 1st Inns	28.008	0.784	**FALSE**
Death Overs Wkts 1st Inns	23.7312	0.784	**FALSE**
Power Play Wkts 2nd Inns	33.8795	0.784	**FALSE**
Middle Overs Wkts 2nd Inns	20.7957	0.784	**FALSE**
Death Overs Wkts 2nd Inns	43.5742	0.784	**FALSE**

Table 4.3: Anderson Darling Test Results

◇ Interpretation

- In the variables of interest shown in the table above, 'Total Runs 1st Inns' follows a normal distribution, whereas the other variables do not. Therefore, non-parametric approaches are more suitable for further analysis.

4.3.3 Significant difference between medians two groups

◇ **Wilcoxon signed-rank test**

The Wilcoxon signed-rank test is a non-parametric statistical test used to compare two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ. It is the non-parametric alternative to the paired t-test and is used when the differences between pairs are not normally distributed.

Here our variable of interest are as follows:

- Runs 1st inning Runs 2nd innings.
- Wickets 1st inning Wickets 2nd innings.

◇ **Total runs**

- H_0**:** There is no difference in total runs scored between 1st and 2nd Innings.
- H_1**:** There is a difference in total runs scored between 1st and 2nd Innings.

◇ **Test statistic formula**

Wilcoxon Statistic = Sum of ranks of differences between paired observations.

◇ **Results**

- **Statistic**: 154151.0000
- **P-value**: 0.0000

◇ **Total wickets**

- H_0**:** There is no difference in total wickets taken between 1st and 2nd Innings
- H_1**:** There is a difference in total wickets taken between 1st and 2nd Innings

◇ **Results**

- **Statistic**: 190510.0000
- **P-value**: 0.0021

◇ **Interpretation**

- For total runs: Reject the null hypothesis, There is a significant difference in total runs scored between 1st and 2nd Innings This suggests that the differences observed are statistically significant and not due to random variation.
- For total wickets: Reject the null hypothesis, There is a difference in the total wickets taken between the 1st and 2nd Innings This suggests that the differences observed are statistically significant and not due to random variation.

4.3.4 Significant difference between medians of three or more groups

◇ **Kruskal-Wallis rank-sum test**

Here our variable of interest are as follows:

- 1st inning runs in power play, middle over and death over.
- 2nd inning runs in power play, middle over and death over.
- 1st inning wickets in power play, middle over and death over.
- 2nd inning runs in power play, middle over and death over.

◇ **Hypotheses**

- H_0**:** The distributions of the groups are the same (i.e., the medians are equal).
- H_0**:** At least one group's distribution is different from the others (i.e., at least one median is different).

◇ **Kruskal-Wallis test results:**

Variable	Statistic	P-value
Runs 1st Inns	1330.33	0.00
Runs 2nd Inns	1689.347	0.00
Wkts 1st Inns	200.4353	0.00
Wkts 2nd Inns	232.5534	0.00

Table 4.4: Kruskal-Wallis test results for runs and wickets in the 1st and 2nd innings

◇ **Interpretation**

- The Kruskal-Wallis test results indicate significant differences in the medians of the groups for all four variables: Runs in the 1st Inns, Runs in the 2nd Inns, Wkts in the 1st Inns, and Wkts in the 2nd Innings Since the p-values for all these variables are less than the significance level of 0.05, we reject the null hypothesis in each case.
- This implies that the central tendencies of these variables are not the same across the groups studied. The significant differences in the medians suggest the presence of underlying differences between the groups that may need further investigation to understand their causes. Identifying these differences could provide insights into the factors influencing the outcomes for runs and wickets in cricket matches.

Chapter 5

Player's Ranking by PCA

5.1 Introduction to PCA

As the number of features or dimensions in a dataset increases, the amount of data required to obtain a statistically significant result increases exponentially. This can lead to issues such as overfitting, increased computation time, and reduced accuracy of machine learning models this is known as the curse of dimensionality problems that arise while working with high-dimensional data.

As the number of dimensions increases, the number of possible combinations of features increases exponentially, which makes it computationally difficult to obtain a representative sample of the data and it becomes expensive to perform tasks such as clustering or classification because it becomes. Additionally, some machine learning algorithms can be sensitive to the number of dimensions, requiring more data to achieve the same level of accuracy as lower-dimensional data.

To address the curse of dimensionality, Feature engineering techniques are used which include feature selection and feature extraction. Dimensionality reduction is a type of feature extraction technique that aims to reduce the number of input features while retaining as much of the original information as possible.

In this article, we will discuss one of the most popular dimensionality reduction techniques i.e. Principal Component Analysis(PCA).

5.1.1 What is Principal Component Analysis(PCA)?

Principal Component Analysis(PCA) technique was introduced by the mathematician **Karl Pearson** in 1901. It works on the condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.

- **Principal Component Analysis (PCA)** is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables.PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover,

- Principal Component Analysis (PCA) is an **unsupervised learning** algorithm technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.
- The main goal of Principal Component Analysis (PCA) is to reduce the dimensionality of a dataset while preserving the most important patterns or relationships between the variables without any prior knowledge of the target variables.

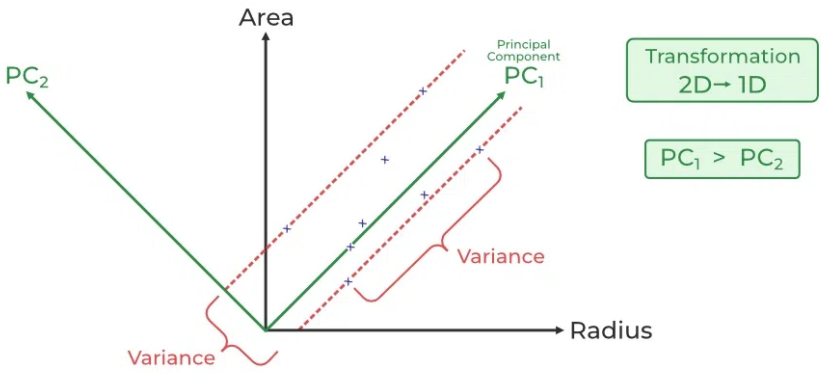


Figure 5.1: Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is used to reduce the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables, retaining most of the sample's information, and useful for the **regression and classification** of data.
Principal Component Analysis (PCA) is a dimensionality reduction technique that identifies a set of orthogonal axes, known as principal components, to capture the maximum variance within data. These components are linear combinations of original variables and are ranked by importance, with the first component capturing the most variation and subsequent components capturing orthogonal variances. PCA serves various purposes including data visualization, feature selection, and data compression. It leverages the assumption that variance in features signifies information, making it valuable for tasks such as plotting high-dimensional data, identifying important variables, and reducing dataset size while preserving critical information.

5.1.2 Step-By-Step Explanation of PCA

Step 1: Standardization
First, we need to standardize our dataset to ensure that each variable has a mean of 0 and a standard deviation of 1.

$$Z = \frac{X - \mu}{\sigma}$$

Here,

μ is the mean of independent features $\mu = \{\mu_1, \mu_2, \cdots, \mu_m\}$

σ is the standard deviation of independent features $\sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$.

Step 2: Covariance Matrix Computation
Covariance measures the strength of joint variability between two or more variables, indicating how much they change in relation to each other. To find the covariance we can use the formula:

$$\text{cov}(x_1, x_2) = \frac{\sum_{i=1}^{n}(x_{1_i} - \bar{x}_1)(x_{2_i} - \bar{x}_2)}{n-1}$$

The value of covariance can be positive, negative, or zero.

- Positive: As x_1 increases, x_2 also increases.
- Negative: As x_1 increases, x_2 decreases.
- Zero: No direct relation.

Step 3: Compute Eigenvalues and Eigenvectors of Covariance Matrix to Identify Principal Components
Let A be a square $n \times n$ matrix and X be a non-zero vector for which

$$AX = \lambda X$$

for some scalar values λ. Then λ is known as the eigenvalue of matrix A and X is known as the eigenvector of matrix A for the corresponding eigenvalue.

It can also be written as:

$$AX - \lambda X = 0$$
$$(A - \lambda I)X = 0$$

where I is the identity matrix of the same shape as matrix A. And the above conditions will be true only if $(A - \lambda I)$ will be non-invertible (i.e., a singular matrix). That means,

$$|A - \lambda I| = 0$$

From the above equation, we can find the eigenvalues λ, and therefore, the corresponding eigenvector can be found using the equation $AX = \lambda X$.

5.2 Objective

Rationale:
The IPL involves diverse player performances in batting, bowling, and all-round skills. Traditional rankings may not capture overall performance effectively. PCA reduces dataset dimensionality while preserving significant variance, offering a holistic approach to ranking players.

5.3 Analysis Of Batsmen

This Section includes the analysis of the Batsman's ranking based on their overall performance. Dataset needed for batsman's ranking has variables as 'Matches played', 'Innings played', 'Runs scored', 'Balls played', 'Number of times got out', 'Strike rate', 'Average', 'Number of hundreds', 'Number of fifties', 'Number of Fours' and 'Number of sixes'.

5.3.1 Correlation heatmap

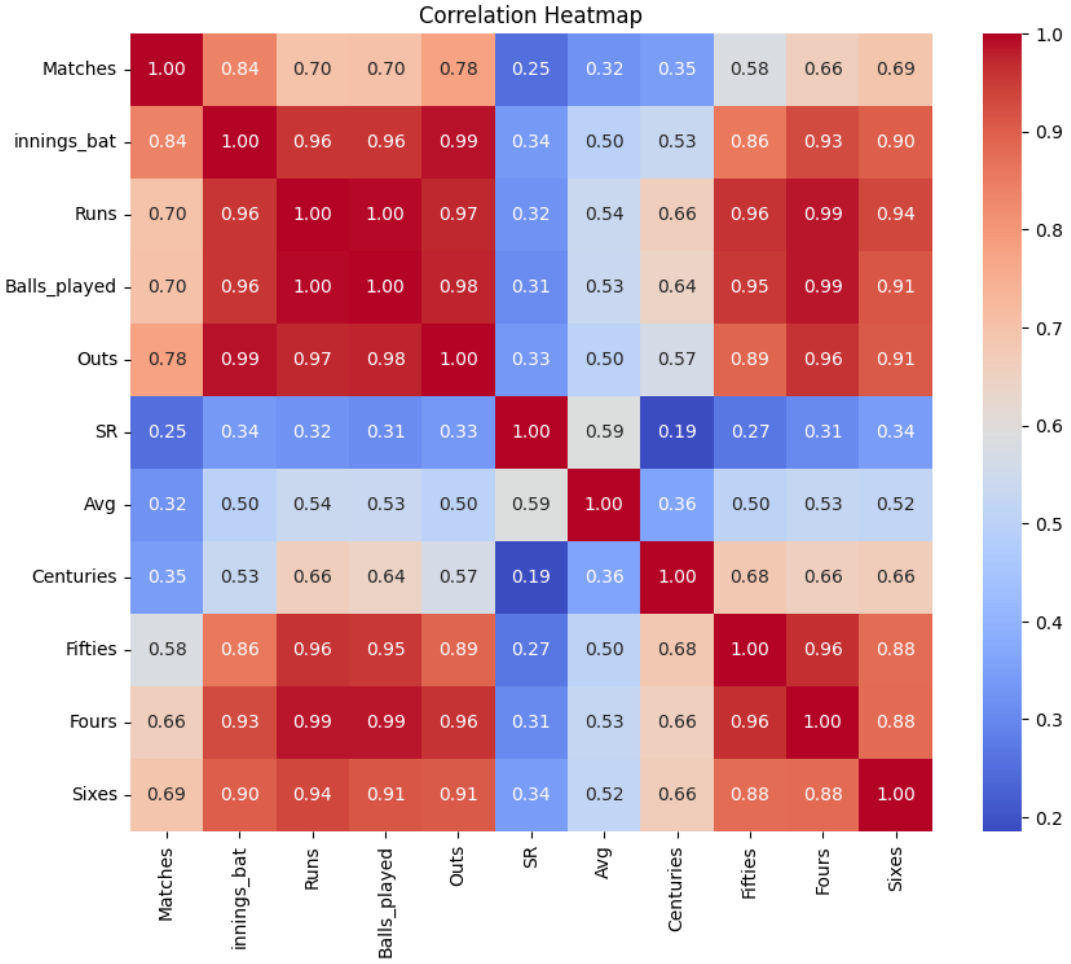


Figure 5.2: Correlation heatmap of IPL batsmen statistics

Interpretation

- High correlations between ***innings_bat***, ***Runs***, ***Balls_played***, ***Outs***, ***Fifties***, and ***Fours*** indicate that these metrics move together, reflecting overall player performance.
- ***SR (Strike Rate)*** and ***Avg (Batting Average)*** show moderate correlations with other variables, suggesting that while important, they are influenced by different factors compared to raw performance metrics like runs and balls faced.
- Performance metrics such as ***Centuries***, ***Fifties***, ***Fours***, and ***Sixes*** are interconnected, reflecting the aggressive scoring capability of players.

5.3.2 Proportion of explained variance for batsmen

Below Scree plot shows how much variation each PC captured from data. PC1 captures most variation, PC2 captures second most and so on....

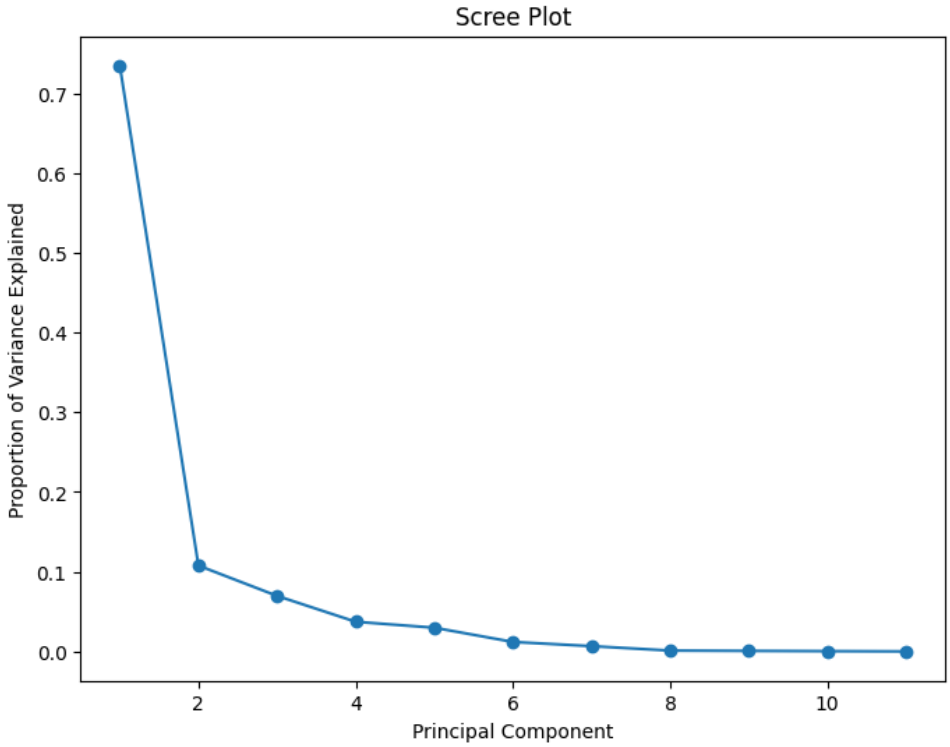


Figure 5.3: Proportion of explained variance of principal components

The first principal component (PC1) explains approximately 70% of the total variance in the dataset.

5.3.3 Eigen Analysis

Percentage of variance explained

$$\text{Percentage of variance explained} = \left(\frac{\text{Eigenvalue}}{\text{Sum of eigenvalues}}\right) \times 100$$

Ordered eigen values and proportion of total variability for Batsmen

PC's	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
Eigen_Values	8.08454	1.18484	0.76642	0.40929	0.32738	0.13086	0.07297	0.01168	0.00754	0.00395	0.00053
Proportion explain	0.73496	0.10771	0.06967	0.03721	0.02976	0.0119	0.00663	0.00106	0.00069	0.00036	0.00005
Cumulative proportion	0.73496	0.84267	0.91235	0.94955	0.97932	0.99121	0.99785	0.99891	0.99959	0.99995	1

Table 5.1: Ordered eigen values and proportion of total variability for Batsmen

5.3.4 PC's eigen vector of covariance matrix

PC's	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
Matches	0.2643	-0.1119	-0.5901	0.3774	-0.5022	0.1810	-0.3378	-0.0886	0.0825	0.1106	-0.0319
Innings	0.3393	-0.0715	-0.2560	-0.0078	-0.0088	-0.0029	0.3582	0.3852	-0.1437	-0.6901	0.2012
Runs	0.3483	-0.0752	0.0343	-0.1022	0.1273	-0.0211	0.0292	0.1870	0.3569	-0.0295	-0.8250
Balls	0.3466	-0.0839	0.0126	-0.1384	0.1295	0.1141	0.1147	0.3963	0.4459	0.4951	0.4578
Outs	0.3428	-0.0782	-0.1603	-0.0555	0.1041	0.0424	0.4356	-0.1864	-0.6254	0.4450	-0.1462
SR	0.1395	0.7645	-0.1178	0.4415	0.4253	0.0650	-0.0397	0.0160	0.0192	0.0077	0.0038
Avg	0.2096	0.5943	0.1424	-0.5019	-0.5738	-0.0012	0.0068	-0.0130	-0.0346	0.0040	-0.0040
100's	0.2386	-0.0871	0.6913	0.5587	-0.3166	0.1370	0.1432	0.0473	-0.0575	-0.0181	0.0074
50's	0.3303	-0.0997	0.2012	-0.1852	0.2400	0.1572	-0.7168	0.1866	-0.4089	-0.0703	0.0554
4's	0.3420	-0.0822	0.0744	-0.1536	0.1892	0.3205	0.0480	-0.7337	0.2825	-0.2493	0.1615
6's	0.3313	-0.0405	0.0338	0.0891	0.0100	-0.8953	-0.1245	-0.2034	0.0579	0.0057	0.1328

Table 5.2: PC's eigen vector of correlation matrix for batsmen

As we know, PC1 captures the majority of the variability present in the original variables. We will use it to rank players.

5.3.5 Rankings of Batsmen

◇ **Ranking equation for the batsmen:**

PC1_Score = (0.2643 × **Matches**) + (0.3393 × **Innings**) + (0.3483 × **Runs**) + (0.3466 × **Balls**) + (0.3428 × **Outs**) + (0.1395 × **Strike Rate**) + (0.2096 × **Average**) + (0.2386 × **Hundreds**) + (0.3303 × **Fifties**) + (0.3420 × **Fours**) + (0.3313 × **Sixes**)

Top 100 batsmen According to Principal Component 1 with above 70% explained variation.

Player	Mat	Inns	Runs	Balls	Outs	SR	Avg	100's	50's	4's	6's	PC1	Rank
V Kohli	255	229	7273	5739	205	126.73	35.48	7	51	646	235	**18.86**	**1**
DA Warner	177	176	6399	4719	156	135.6	41.02	4	61	646	226	**16.14**	**2**
S Dhawan	222	216	6617	5358	189	123.5	35.01	2	50	750	149	**15.7**	**3**
RG Sharma	269	237	6213	4890	219	127.06	28.37	1	42	554	258	**15.36**	**4**
CH Gayle	179	141	4997	3516	128	142.12	39.04	6	32	408	359	**14.09**	**5**
SK Raina	269	200	5536	4177	168	132.54	32.95	1	39	506	204	**13.3**	**6**
AB de Villiers	170	170	5181	3487	125	148.58	41.45	3	41	414	253	**12.84**	**7**
RV Uthappa	197	197	4954	3927	184	126.15	26.92	0	27	481	182	**11.42**	**8**
MS Dhoni	217	217	5082	3865	146	131.49	34.81	0	24	349	239	**11.3**	**9**
SR Watson	246	141	3880	2892	126	134.16	30.79	4	21	377	190	**10.77**	**10**

Continued on next page

Player	Mat	Inns	Runs	Balls	Outs	SR	Avg	100's	50's	4's	6's	PC1	Rank
KD Karthik	220	220	4517	3492	180	129.35	25.09	0	20	439	139	**10.49**	**11**
AM Rahane	160	159	4400	3658	143	120.28	30.77	2	30	455	96	**10.36**	**12**
KL Rahul	108	108	4169	3180	92	131.1	45.32	4	33	355	168	**10.28**	**13**
AT Rayudu	185	185	4348	3490	152	124.58	28.61	1	22	359	173	**10.18**	**14**
SV Samson	147	147	3888	2903	131	133.93	29.68	3	20	304	182	**9.63**	**15**
G Gambhir	151	151	4217	3524	135	119.67	31.24	0	36	492	59	**9.39**	**16**
F du Plessis	124	123	4133	3159	110	130.83	37.57	0	33	375	145	**8.84**	**17**
KA Pollard	275	168	3437	2447	129	140.46	26.64	0	16	221	224	**8.69**	**18**
MK Pandey	157	157	3817	3255	132	117.27	28.92	1	22	333	109	**8.67**	**19**
JC Buttler	95	95	3224	2253	86	143.1	37.49	5	19	320	149	**8.64**	**20**
YK Pathan	235	153	3222	2334	111	138.05	29.03	1	14	263	161	**8.03**	**21**
SA Yadav	125	124	3249	2313	102	140.47	31.85	1	21	349	112	**7.48**	**22**
BB McCullum	109	109	2882	2272	106	126.85	27.19	2	13	293	130	**6.97**	**23**
Q de Kock	96	96	2910	2216	90	131.32	32.33	2	20	287	114	**6.91**	**24**
V Sehwag	119	104	2728	1833	98	148.83	27.84	2	16	334	106	**6.84**	**25**
GJ Maxwell	192	119	2720	1796	102	151.45	26.67	0	18	227	158	**6.68**	**26**
RA Jadeja	366	169	2692	2187	108	123.09	24.93	0	2	193	99	**6.66**	**27**
Shubman Gill	87	87	2790	2137	75	130.56	37.2	3	18	273	80	**6.61**	**28**
Yuvraj Singh	199	126	2754	2207	109	124.78	25.27	0	13	218	149	**6.57**	**29**
WP Saha	134	134	2798	2248	112	124.47	24.98	1	13	278	84	**6.48**	**30**
PA Patel	136	136	2848	2442	127	116.63	22.43	0	13	365	49	**6.33**	**31**
RR Pant	97	97	2851	2000	85	142.55	33.54	1	15	262	129	**6.22**	**32**
M Vijay	109	105	2619	2208	97	118.61	27	2	13	247	91	**6.21**	**33**
MA Agarwal	117	117	2605	2022	112	128.83	23.26	1	13	257	97	**6.09**	**34**
DA Miller	114	114	2714	2022	77	134.22	35.25	1	12	187	126	**5.89**	**35**
N Rana	123	99	2594	1980	92	131.01	28.2	0	18	229	131	**5.85**	**36**
SS Iyer	101	100	2780	2295	93	121.13	29.89	0	19	238	99	**5.79**	**37**
AD Russell	193	95	2266	1374	78	164.92	29.05	0	10	151	193	**5.59**	**38**
DR Smith	134	88	2385	1803	81	132.28	29.44	0	17	245	117	**5.47**	**39**
JH Kallis	184	95	2427	2291	89	105.94	27.27	0	17	255	44	**5.39**	**40**
HH Pandya	196	115	2318	1660	77	139.64	30.1	0	10	172	126	**5.38**	**41**
SE Marsh	69	69	2489	1913	65	130.11	38.29	1	20	269	78	**5.37**	**42**
AC Gilchrist	81	80	2069	1555	76	133.05	27.22	2	11	239	92	**5.1**	**43**
SPD Smith	94	93	2495	1999	78	124.81	31.99	1	11	226	60	**5.01**	**44**
Ishan Kishan	85	85	2324	1769	77	131.37	30.18	0	15	220	103	**4.84**	**45**
SR Tendulkar	82	78	2333	2043	71	114.19	32.86	1	13	296	29	**4.78**	**46**
AJ Finch	95	90	2092	1696	84	123.35	24.9	0	15	214	78	**4.55**	**47**

Continued on next page

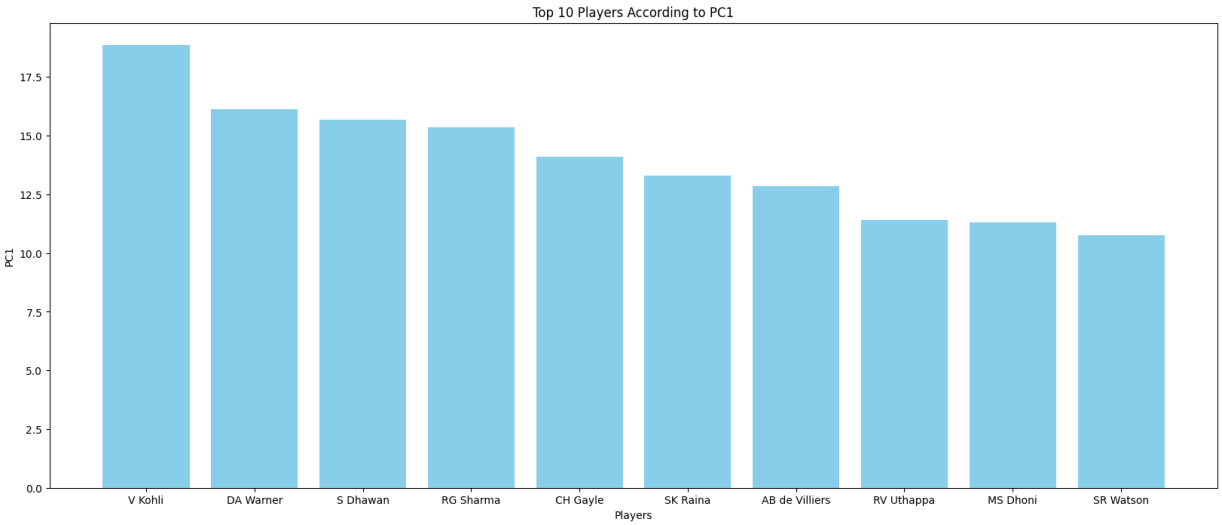
Player	Mat	Inns	Runs	Balls	Outs	SR	Avg	100's	50's	4's	6's	PC1	Rank
KS Williamson	76	74	2105	1707	57	123.32	36.93	0	18	183	64	**4.18**	**48**
RA Tripathi	88	87	2071	1529	76	135.45	27.25	0	11	205	78	**4.18**	**49**
R Dravid	82	82	2174	1918	77	113.35	28.23	0	11	269	28	**4.09**	**50**
JP Duminy	124	75	2029	1680	49	120.77	41.41	0	14	126	79	**4.09**	**51**
DJ Bravo	268	110	1560	1247	69	125.1	22.61	0	5	120	66	**4.05**	**52**
MEK Hussey	58	58	1977	1648	52	119.96	38.02	1	15	198	52	**4.04**	**53**
Jayawardene	78	78	1808	1522	64	118.79	28.25	1	10	200	40	**3.81**	**54**
RD Gaikwad	51	51	1797	1356	45	132.52	39.93	1	14	159	73	**3.74**	**55**
PP Shaw	71	71	1694	1196	69	141.64	24.55	0	13	208	56	**3.51**	**56**
Mandeep Singh	99	97	1706	1418	85	120.31	20.07	0	6	176	38	**3.39**	**57**
AR Patel	234	100	1418	1135	69	124.93	20.55	0	1	88	69	**3.38**	**58**
KH Pandya	201	96	1514	1175	69	128.85	21.94	0	1	136	56	**3.34**	**59**
SP Narine	253	92	1046	692	71	151.16	14.73	0	4	114	64	**3.28**	**60**
NV Ojha	94	94	1554	1360	74	114.26	21	0	6	121	79	**3.27**	**61**
MK Tiwary	89	83	1695	1489	61	113.83	27.79	0	7	156	40	**3.1**	**62**
MP Stoinis	129	74	1478	1094	55	135.1	26.87	0	7	111	75	**3.08**	**63**
D Padikkal	57	57	1521	1263	56	120.43	27.16	1	9	167	42	**3.06**	**64**
Sangakkara	68	68	1687	1424	69	118.47	24.45	0	10	195	27	**3.06**	**65**
KK Nair	68	68	1496	1192	65	125.5	23.02	0	10	161	39	**2.82**	**66**
DJ Hooda	118	87	1321	1063	69	124.27	19.14	0	7	80	58	**2.79**	**67**
SS Tiwary	73	73	1494	1291	50	115.72	29.88	0	8	111	50	**2.68**	**68**
S Badrinath	65	65	1441	1245	48	115.74	30.02	0	11	154	28	**2.64**	**69**
EJG Morgan	74	74	1406	1181	61	119.05	23.05	0	5	112	64	**2.59**	**70**
JM Bairstow	39	39	1291	935	38	138.07	33.97	1	9	133	55	**2.58**	**71**
Harbhajan	248	88	833	636	56	130.97	14.88	0	1	79	42	**2.45**	**72**
BJ Hodge	83	63	1400	1153	46	121.42	30.43	0	6	122	43	**2.42**	**73**
CA Lynn	42	42	1329	974	39	136.45	34.08	0	10	132	66	**2.4**	**74**
IK Pathan	183	82	1150	985	53	116.75	21.7	0	1	88	38	**2.39**	**75**
SC Ganguly	76	56	1349	1326	54	101.73	24.98	0	7	137	42	**2.39**	**76**
YBK Jaiswal	38	37	1172	807	35	145.23	33.49	1	8	145	48	**2.38**	**77**
N Pooran	58	58	1270	847	48	149.94	26.46	0	6	78	91	**2.36**	**78**
DJ Hussey	87	61	1322	1101	49	120.07	26.98	0	5	90	60	**2.31**	**79**
LMP Simmons	30	29	1079	878	27	122.89	39.96	1	11	109	44	**2.21**	**80**
KM Jadhav	81	81	1208	1012	54	119.37	22.37	0	4	102	40	**2.18**	**81**
R Ashwin	275	81	714	622	53	114.79	13.47	0	1	56	24	**2.12**	**82**
BA Stokes	81	43	935	718	41	130.22	22.8	2	2	81	32	**2.07**	**83**
JA Morkel	154	67	975	712	40	136.94	24.38	0	3	61	55	**2.06**	**84**

Continued on next page

Player	Mat	Inns	Runs	Balls	Outs	SR	Avg	100's	50's	4's	6's	PC1	Rank
TM Dilshan	75	50	1153	1047	41	110.12	28.12	0	9	140	24	**2.06**	**85**
MM Ali	100	52	1034	744	44	138.98	23.5	0	5	88	59	**1.97**	**86**
SO Hetmyer	56	56	1130	782	35	144.5	32.29	0	4	67	75	**1.94**	**87**
A Symonds	66	36	974	781	26	124.71	37.46	1	5	74	41	**1.87**	**88**
S Dube	60	47	1106	818	39	135.21	28.36	0	6	58	73	**1.87**	**89**
PP Chawla	260	80	609	565	53	107.79	11.49	0	0	55	19	**1.83**	**90**
VR Iyer	44	36	956	762	34	125.46	28.12	1	7	86	42	**1.82**	**91**
KP Pietersen	49	36	1001	758	29	132.06	34.52	1	4	91	40	**1.8**	**92**
ML Hayden	32	32	1107	839	28	131.94	39.54	0	8	121	44	**1.78**	**93**
MC Henriques	114	54	1000	801	38	124.84	26.32	0	5	87	28	**1.75**	**94**
V Shankar	73	51	1032	815	38	126.63	27.16	0	6	73	43	**1.69**	**95**
M Vohra	51	51	1083	863	48	125.49	22.56	0	3	104	43	**1.61**	**96**
STR Binny	129	66	880	700	44	125.71	20	0	0	66	35	**1.53**	**97**
LRPL Taylor	56	54	1017	847	41	120.07	24.8	0	3	66	46	**1.46**	**98**
Venugopal Rao	72	52	985	865	42	113.87	23.45	0	3	77	37	**1.45**	**99**
CL White	51	45	971	774	35	125.45	27.74	0	6	76	38	**1.44**	**100**

Table 5.3: Ranking of batsmen according to PC1

◇ Top 10 Batsmen according to PC1



captionTop 10 batsmen according to PC1

5.4 Analysis of Bowlers

This Section includes the analysis of the Bowler's ranking based on their overall performance. Dataset needed for batsman's ranking has variables as 'Matches played', 'Innings played', 'Number of Wickets Taken', 'Total balls bowled', 'Number of runs given', 'Economy rate', 'Bowling average', 'Bowling 'Strike rate'.

5.4.1 Correlation heatmap

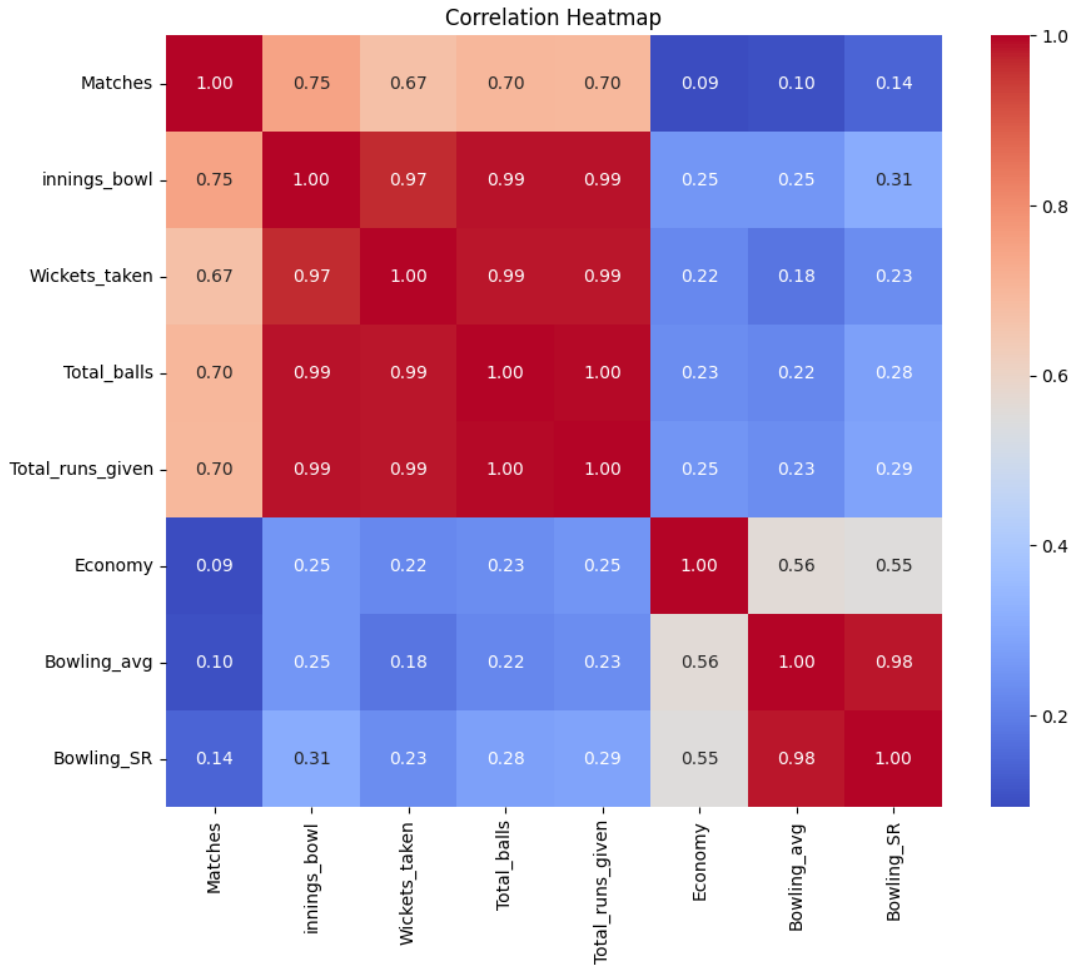


Figure 5.4: Correlation Heatmap of IPL Bowler Statistics

Interpretation

- ***Innings Bowled, Wickets Taken, Total Balls, Total Runs Given*** Very strong positive relationships (near 1). As one increases, the others also tend to increase.
- ***Bowling Average and Bowling Strike Rate*** Strong positive relationship (0.98), indicating that a better (lower) average is associated with a better (lower) strike rate.
- ***Bowling Average and Economy (0.56)*** Indicates that a better (lower) economy rate tends to be associated with a better (lower) bowling average.
- ***Matches with Bowling Average and Bowling Strike Rate*** Weak positive relationships (0.10 and 0.14), suggesting very little correlation.

5.4.2 Proportion of explained variance

Below Scree plot shows how much variation each PC captured from data. PC1 captures most variation, PC2 captures second most and so on....

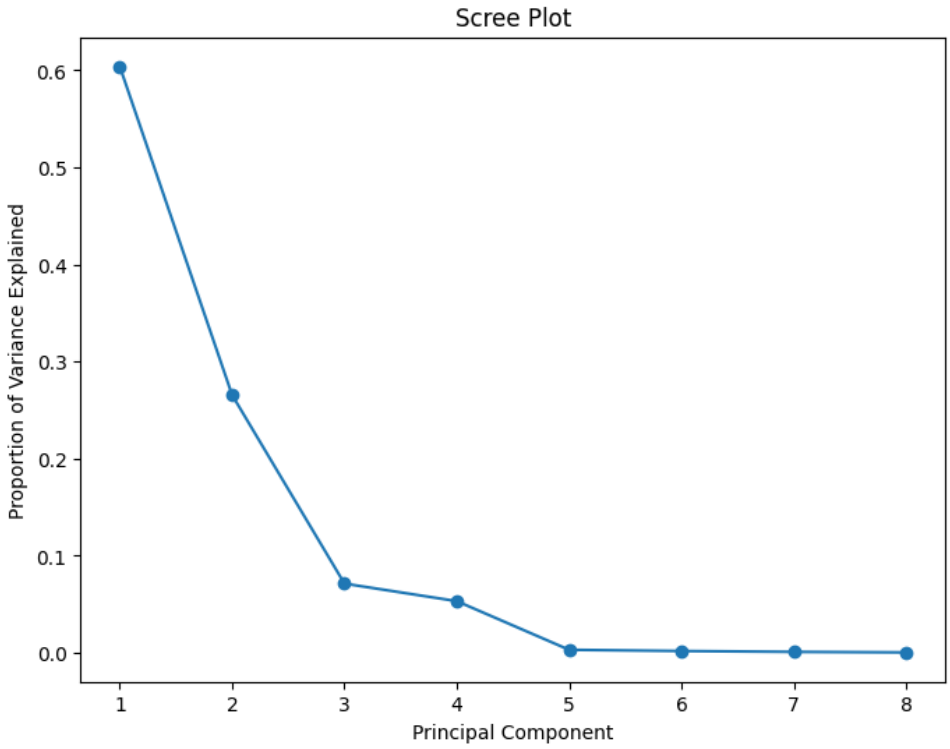


Figure 5.5: Proportion of explained variance of principal components

The first principal component (PC1) explains approximately 60% of the total variance in the dataset.

5.4.3 Eigen analysis

By using same formula as we use to calculate the explain proportion in analysis of batsmen

PC's	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Eigen values	4.8311	2.1241	0.5700	0.4254	0.0244	0.0144	0.0075	0.0031
Explain roportion	0.6039	0.2655	0.0712	0.0532	0.0031	0.0018	0.0009	0.0004
Cumulative Proportion	0.6039	0.8694	0.9406	0.9938	0.9969	0.9987	0.9996	1

Table 5.4: Ordered eigen values and proportion of total variability for bowler

5.4.4 PC's eigen vector of covariance matrix

PC's	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Matches	0.340452	0.192806	0.225146	-0.88324	-0.10937	-0.02843	-0.05786	-0.00219
Innings	0.443965	0.124781	0.00054	0.078461	0.601053	0.122318	0.636343	-0.00319
Wickets	0.430571	0.166065	-0.07672	0.256126	-0.71457	-0.27107	0.362562	-0.00127
Balls	0.439308	0.145801	-0.02989	0.207088	0.137341	-0.02005	-0.48296	0.699609
RunsGiven	0.440693	0.135058	-0.04335	0.205797	0.054522	0.234607	-0.45933	-0.68871
Economy	0.176968	-0.44336	-0.84392	-0.24351	0.008814	-0.01898	-0.01042	0.007828
SR	0.192087	-0.59531	0.327176	0.031152	-0.24615	0.643354	0.088114	0.135793
Avg	0.214894	-0.57378	0.348429	0.051308	0.183456	-0.66411	-0.09033	-0.13307

Table 5.5: PC's eigen vector of correlation matrix for bowlers

As we know, PC1 captures the majority of the variability present in the original variables. We will use it to rank players.

5.4.5 Ranking of bowlers

◇ **Ranking equation for the bowler:**

PC1_Score = (0.340452 × **Matches**) + (0.443965 × **Innings**) + (0.430571 × **Wickets**) + (0.439308 × **Balls**) + (0.440693 × **RunsGiven**) + (0.176968 × **Economy**) + (0.192087 × **Bowling SR**) + (0.214894 × **Bowling Avg**)

Top 100 bowlers According to Principal Component 1 with above 60% explained variation.

Player	Mat	Inns	Wickets	Balls	RunsGiven	Economy	SR	Avg	PC1	Rank
Player	**Mat**	**Inn**	**Wickets**	**Balls**	**RunsGiven**	**Economy**	**Avg**	**SR**	**PC1**	**Rank**
R Ashwin	275	194	189	4333	4966	6.88	26.28	22.93	**11.11**	**1**
RA Jadeja	366	197	161	3610	4546	7.56	28.24	22.42	**10.71**	**2**
PP Chawla	260	180	188	3680	4867	7.94	25.89	19.57	**10.32**	**3**
DJ Bravo	268	158	207	3296	4436	8.08	21.43	15.92	**9.73**	**4**
SP Narine	253	161	182	3812	4296	6.76	23.6	20.95	**9.68**	**5**
B Kumar	220	160	184	3703	4496	7.28	24.43	20.12	**9.54**	**6**
A Mishra	216	161	182	3432	4173	7.3	22.93	18.86	**9.13**	**7**
Harbhajan	248	160	161	3496	4101	7.04	25.47	21.71	**9.11**	**8**
YS Chahal	161	144	194	3266	4133	7.59	21.3	16.84	**8.53**	**9**
UT Yadav	186	140	154	3061	4231	8.29	27.47	19.88	**8.19**	**10**
AR Patel	234	134	120	2887	3484	7.24	29.03	24.06	**7.52**	**11**
SL Malinga	142	122	188	2974	3486	7.03	18.54	15.82	**7.44**	**12**
JJ Bumrah	142	120	161	2857	3499	7.35	21.73	17.75	**7.09**	**13**

Continued on next page

Player	Mat	Inns	Wickets	Balls	RunsGiven	Economy	SR	Avg	PC1	Rank
Sandeep Sharma	141	116	138	2689	3444	7.68	24.96	19.49	**6.67**	**14**
P Kumar	176	119	102	2637	3342	7.6	32.76	25.85	**6.57**	**15**
Mohammed Shami	133	110	144	2521	3486	8.3	24.21	17.51	**6.51**	**16**
Rashid Khan	159	109	147	2639	2968	6.75	20.19	17.95	**6.41**	**17**
SR Watson	246	105	107	2137	2742	7.7	25.63	19.97	**6.07**	**18**
R Vinay Kumar	146	104	127	2186	3041	8.35	23.94	17.21	**5.84**	**19**
IK Pathan	183	101	99	2113	2711	7.7	27.38	21.34	**5.54**	**20**
Z Khan	126	99	119	2276	2860	7.54	24.03	19.13	**5.52**	**21**
MM Sharma	126	100	131	2063	2850	8.29	21.76	15.75	**5.49**	**22**
I Sharma	120	101	84	2247	2989	7.98	35.58	26.75	**5.37**	**23**
KA Pollard	275	107	81	1586	2200	8.32	27.16	19.58	**5.35**	**24**
KH Pandya	201	105	74	1978	2386	7.24	32.24	26.73	**5.25**	**25**
DW Steyn	128	95	105	2282	2583	6.79	24.6	21.73	**5.18**	**26**
HV Patel	122	89	121	1967	2730	8.33	22.56	16.26	**5.08**	**27**
JD Unadkat	115	93	100	1994	2898	8.72	28.98	19.94	**5.07**	**28**
TA Boult	106	88	112	2095	2838	8.13	25.34	18.71	**5.05**	**29**
AD Russell	193	98	106	1583	2387	9.05	22.52	14.93	**5.03**	**30**
A Nehra	105	88	121	1974	2537	7.71	20.97	16.31	**4.83**	**31**
JH Kallis	184	89	74	1799	2348	7.83	31.73	24.31	**4.77**	**32**
JA Morkel	154	87	96	1807	2409	8	25.09	18.82	**4.72**	**33**
DS Kulkarni	112	92	91	1876	2513	8.04	27.62	20.62	**4.62**	**34**
PP Ojha	109	90	99	1945	2399	7.4	24.23	19.65	**4.59**	**35**
CH Morris	130	81	107	1801	2377	7.92	22.21	16.83	**4.54**	**36**
SN Thakur	115	83	95	1782	2593	8.73	27.29	18.76	**4.54**	**37**
RP Singh	110	82	100	1874	2417	7.74	24.17	18.74	**4.46**	**38**
R Bhatia	138	91	82	1661	2059	7.44	25.11	20.26	**4.23**	**39**
Mohammed Siraj	98	79	81	1731	2387	8.27	29.47	21.37	**4.11**	**40**
K Rabada	94	69	115	1645	2266	8.27	19.7	14.3	**4.04**	**41**
HH Pandya	196	81	57	1259	1807	8.61	31.7	22.09	**3.88**	**42**
YK Pathan	235	82	46	1184	1443	7.31	31.37	25.74	**3.76**	**43**
M Morkel	90	70	88	1699	2136	7.54	24.27	19.31	**3.76**	**44**
AB Dinda	91	75	82	1589	2103	7.94	25.65	19.38	**3.70**	**45**
L Balaji	86	73	85	1574	2083	7.94	24.51	18.52	**3.64**	**46**
DL Chahar	85	73	74	1600	2041	7.65	27.58	21.62	**3.56**	**47**
Kuldeep Yadav	98	71	74	1519	2037	8.05	27.53	20.53	**3.55**	**48**
Shakib Al Hasan	122	70	71	1515	1864	7.38	26.25	21.34	**3.53**	**49**
SK Trivedi	89	75	73	1557	1944	7.49	26.63	21.33	**3.50**	**50**

Continued on next page

Player	Mat	Inns	Wickets	Balls	RunsGiven	Economy	SR	Avg	PC1	Rank
KV Sharma	109	73	75	1371	1874	8.2	24.99	18.28	**3.42**	**51**
SK Raina	269	69	30	930	1139	7.35	37.97	31	**3.42**	**52**
RD Chahar	88	68	66	1508	1885	7.5	28.56	22.85	**3.29**	**53**
S Nadeem	90	70	54	1444	1800	7.48	33.33	26.74	**3.20**	**54**
JP Faulkner	105	60	76	1287	1849	8.62	24.33	16.93	**3.15**	**55**
M Muralitharan	75	66	67	1581	1765	6.7	26.34	23.6	**3.14**	**56**
MM Patel	75	63	82	1382	1733	7.52	21.13	16.85	**3.01**	**57**
GJ Maxwell	192	73	34	867	1182	8.18	34.76	25.5	**2.95**	**58**
MJ McClenaghan	76	56	75	1346	1839	8.2	24.52	17.95	**2.94**	**59**
Imran Tahir	67	59	86	1340	1729	7.74	20.1	15.58	**2.90**	**60**
Yuvraj Singh	199	73	39	882	1091	7.42	27.97	22.62	**2.87**	**61**
MG Johnson	82	54	66	1301	1740	8.02	26.36	19.71	**2.79**	**62**
TG Southee	73	54	57	1262	1774	8.43	31.12	22.14	**2.72**	**63**
S Kaul	64	55	63	1238	1772	8.59	28.13	19.65	**2.68**	**64**
CV Varun	66	56	64	1315	1631	7.44	25.48	20.55	**2.64**	**65**
M Prasidh Krishna	58	51	54	1231	1741	8.49	32.24	22.8	**2.54**	**66**
SK Warne	81	54	60	1223	1465	7.19	24.42	20.38	**2.48**	**67**
MC Henriques	114	60	46	979	1302	7.98	28.3	21.28	**2.43**	**68**
M Kartik	69	55	39	1182	1418	7.2	36.36	30.31	**2.39**	**69**
Arshdeep Singh	61	51	64	1127	1583	8.43	24.73	17.61	**2.37**	**70**
V Kohli	255	26	5	264	371	8.43	74.2	52.8	**2.33**	**71**
SB Jakati	62	57	50	1101	1474	8.03	29.48	22.02	**2.32**	**72**
Ravi Bishnoi	62	51	58	1190	1477	7.45	25.47	20.52	**2.31**	**73**
Washington Sundar	93	56	38	1034	1262	7.32	33.21	27.21	**2.28**	**74**
VR Aaron	62	50	46	1065	1527	8.6	33.2	23.15	**2.25**	**75**
JO Holder	73	46	59	1049	1489	8.52	25.24	17.78	**2.23**	**76**
SM Curran	81	45	46	980	1502	9.2	32.65	21.3	**2.22**	**77**
MP Stoinis	129	55	40	744	1173	9.46	29.32	18.6	**2.19**	**78**
Mustafizur Rahman	54	48	56	1128	1468	7.81	26.21	20.14	**2.16**	**79**
T Natarajan	49	47	56	1069	1532	8.6	27.36	19.09	**2.14**	**80**
Avesh Khan	54	47	59	1044	1477	8.49	25.03	17.69	**2.11**	**81**
R Tewatia	108	52	36	866	1117	7.74	31.03	24.06	**2.05**	**82**
PJ Cummins	71	42	48	995	1405	8.47	29.27	20.73	**2.03**	**83**
DT Christian	90	49	40	922	1218	7.93	30.45	23.05	**2.03**	**84**
S Gopal	70	48	50	958	1291	8.09	25.82	19.16	**1.97**	**85**
KK Ahmed	48	43	61	1021	1425	8.37	23.36	16.74	**1.96**	**86**
CH Gayle	179	38	19	584	755	7.76	39.74	30.74	**1.90**	**87**

Continued on next page

Player	Mat	Inns	Wickets	Balls	RunsGiven	Economy	SR	Avg	PC1	Rank
MS Gony	59	44	39	925	1317	8.54	33.77	23.72	**1.87**	**88**
JP Duminy	124	49	23	701	847	7.25	36.83	30.48	**1.84**	**89**
AD Mathews	85	44	28	807	1095	8.14	39.11	28.82	**1.82**	**90**
STR Binny	129	63	28	611	763	7.49	27.25	21.82	**1.81**	**91**
RG Sharma	269	32	16	349	462	7.94	28.88	21.81	**1.78**	**92**
A Nortje	51	40	55	957	1316	8.25	23.93	17.4	**1.78**	**93**
S Sreesanth	56	44	43	947	1221	7.74	28.4	22.02	**1.76**	**94**
JC Archer	63	40	49	987	1210	7.36	24.69	20.14	**1.76**	**95**
Iqbal Abdulla	61	48	45	943	1125	7.16	25	20.96	**1.74**	**96**
PJ Sangwan	56	42	44	904	1266	8.4	28.77	20.55	**1.74**	**97**
M Ashwin	56	44	35	898	1182	7.9	33.77	25.66	**1.72**	**98**
R Sharma	63	44	42	935	1100	7.06	26.19	22.26	**1.67**	**99**
AB Agarkar	60	42	33	820	1174	8.59	35.58	24.85	**1.67**	**100**

Table 5.6: Ranking of batsmen according to PC1

◇ Top 10 bowlers according to PC1

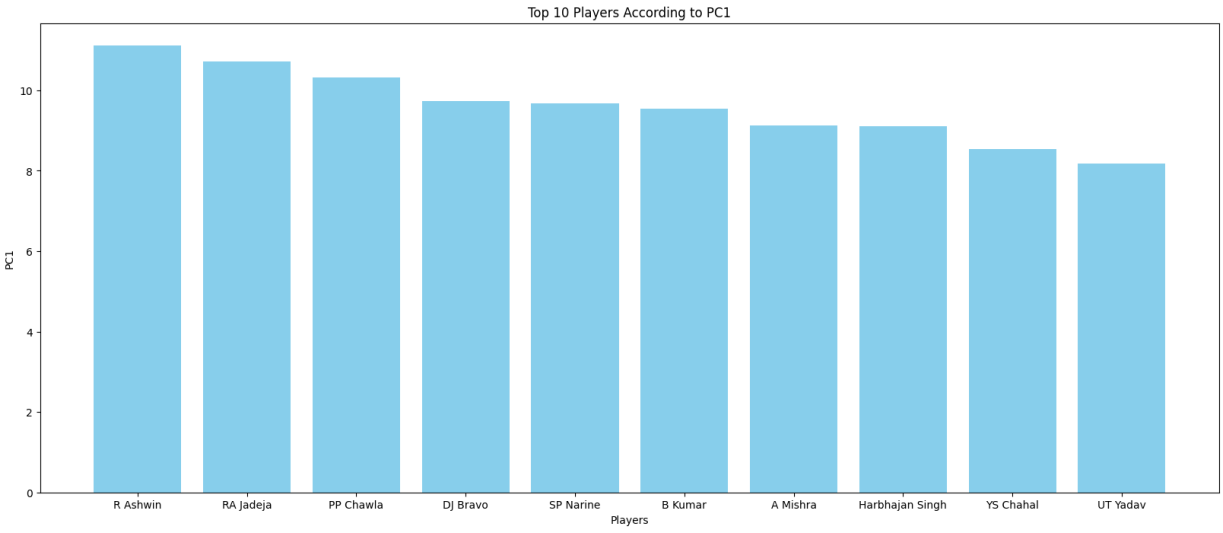


Figure 5.6: Top 10 bowlers according to PC1

Chapter 6

Prediction of Winning Team

6.1 Objective: Prediction of Winning Team

Objective:

- Develop predictive models for forecasting the winning team in cricket matches based on historical match data and in-game statistics using machine learning algorithms.

Key Components:

- Utilize historical match data including teams, venue, and outcomes.
- Extract in-game statistics such as team batting and bowling averages.
- Perform feature engineering to prepare the dataset for model training.
- Train machine learning models using algorithms like logistic regression, KNN, random forest, and decision tree.
- Evaluate model performance using metrics like accuracy, precision, recall, and F1-score.
- Deploy models for making predictions on new cricket matches and implement monitoring mechanisms for model performance.

6.2 Machine Learning Algorithms Overview

6.2.1 Logistic Regression

Overview:

- Logistic Regression is a statistical method for predicting binary outcomes (e.g., win/lose) based on one or more predictor variables.
- It models the probability of the default class (e.g., the winning team) and uses the logistic function to squeeze the output between 0 and 1.

Key Characteristics:

- **Linear Model**: Despite its name, logistic regression is a linear model for classification rather than regression.
- **Probability Estimates**: Provides probabilities for the class labels.
- **Interpretability**: Coefficients indicate the relationship between each feature and the log-odds of the outcome.
- **Assumptions**: Assumes linearity between the predictor variables and the log-odds of the response variable.

Use Cases:

- Binary classification problems such as spam detection, disease diagnosis, and customer churn prediction.

Advantages:

- Simple and easy to implement.
- Efficient and performs well with linearly separable data.
- Outputs probabilities for class membership.

Disadvantages:

- Assumes a linear relationship between features and log-odds.
- Not suitable for non-linear problems unless feature engineering is used.

6.2.2 K-Nearest Neighbors (KNN)

Overview:

- KNN is a simple, instance-based learning algorithm where the classification of a sample is determined by the majority class among its k nearest neighbors.

Key Characteristics:

- **Lazy Learner**: KNN does not learn a discriminative function from the training data but memorizes the training dataset instead.
- **Distance Metric**: Typically uses Euclidean distance to find the closest neighbors.
- **No Training Phase**: All computation is deferred until classification.

Use Cases:

- Classification tasks in recommendation systems, image recognition, and anomaly detection.

Advantages:

- Simple to understand and implement.
- No assumptions about the data distribution.
- Naturally handles multi-class classification.

Disadvantages:

- Computationally intensive for large datasets.
- Sensitive to the choice of k and the distance metric.
- Poor performance with high-dimensional data due to the curse of dimensionality.

6.2.3 Decision Tree

Overview:

- A decision tree is a non-parametric supervised learning algorithm used for classification and regression tasks. It splits the data into subsets based on the value of input features.

Key Characteristics:

- **Tree Structure**: Consists of nodes (tests on features), branches (outcome of the tests), and leaves (final decision or class).
- **Greedy Algorithm**: Uses algorithms like CART (Classification and Regression Tree) to create splits based on feature importance.

Use Cases:

- Widely used in various domains like finance (credit risk analysis), healthcare (disease prediction), and marketing (customer segmentation).

Advantages:

- Easy to understand and visualize.
- Requires little data preparation (e.g., no need for feature scaling).
- Handles both numerical and categorical data.

Disadvantages:

- Prone to overfitting, especially with noisy data.
- Can create biased trees if some classes dominate.
- Instability: Small changes in the data can result in a completely different tree.

6.2.4 Random Forest

Overview:

- Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the class that is the mode of the classes of the individual trees.

Key Characteristics:

- **Ensemble Method**: Combines the predictions of multiple decision trees.
- **Bootstrap Aggregating (Bagging)**: Each tree is trained on a bootstrap sample of the data.
- **Random Feature Selection**: At each split in the tree, a random subset of features is considered.

Use Cases:

- Classification tasks such as credit scoring, medical diagnosis, and stock market prediction.

Advantages:

- Robust to overfitting due to the averaging of multiple trees.
- Handles both classification and regression tasks.
- Works well with high-dimensional data.

Disadvantages:

- Can be computationally intensive and memory consuming.
- Less interpretable than a single decision tree.
- Performance can degrade with noisy data.

6.3 Analysis Using ML Algorithms

◇ Data Used

ID	City	Team1	Team2	Venue	Toss Winner	Toss Decision	Team1 BallAvg	Team1 BatAvg	Team2 BatAvg	Team2 BallAvg	Winning Team
335982	Bangalore	RCB	KKR	M Chinnaswamy	RCB	field	13.83	20.34	12.38	11.70	KKR
335983	Chandigarh	PK	CSK	Punjab Cricket	CSK	bat	9.99	10.24	35.10	27.65	CSK
335984	Delhi	DC	RR	Feroz Shah Kotla	RR	bat	24.93	33.60	16.89	12.54	DC
335985	Mumbai	MI	RCB	Wankhede Stadium	MI	bat	9.72	9.61	25.73	18.51	RCB
335986	Kolkata	KKR	SRH	Eden Gardens	SRH	bat	6.67	5.91	16.69	12.71	KKR
335987	Jaipur	RR	PK	Sawai Mansingh	PK	bat	13.9	18.40	13.60	13.30	RR
335988	Hyderabad	SRH	DC	Rajiv Gandhi	SRH	bat	9.04	11.67	39.06	26.49	DC
335989	Chennai	CSK	MI	MA Chidambaram	MI	field	22.82	28.77	13.24	12.19	CSK
335990	Hyderabad	SRH	RR	Rajiv Gandhi	RR	field	5.47	8.37	19.91	16.63	RR
335991	Chandigarh	PK	MI	Punjab Cricket	MI	field	12.23	13.08	17.57	16.49	PK

Table 6.1: Data(head) used for prediction

◇ Encoding of categorical variables

City	City_encoded	City	City_encoded	City	City_encoded
Abu Dhabi	0	Dubai	11	Mumbai	22
Ahmedabad	1	Durban	12	Nagpur	23
Bangalore	2	East London	13	Port Elizabeth	24
Bloemfontein	3	Hyderabad	14	Pune	25
Cape Town	4	Indore	15	Raipur	26
Centurion	5	Jaipur	16	Rajkot	27
Chandigarh	6	Johannesburg	17	Ranchi	28
Chennai	7	Kanpur	18	Sharjah	29
Cuttack	8	Kimberley	19	Visakhapatnam	30
Delhi	9	Kochi	20		
Dharamsala	10	Kolkata	21		

Table 6.2: Encoding of city

Teams	Team_encoded	Teams	Team_encoded
CSK	0	MI	7
DC	1	PK	8
GL	2	PW	9
GT	3	RCB	10
KKR	4	RPS	11
KTK	5	RR	12
LSG	6	SRH	13

Table 6.3: Encoding of teams

TossDecision	TossDecision_encoded
bat	0
field	1

Table 6.4: Encoding toss decision

Venue	Venue_encoded	Venue	Venue_encoded	Venue	Venue_encoded
Arun Jaitley	0	JSCA Complex	13	Sardar Patel	25
Barabati	1	Kingsmead	14	Saurashtra Cricket	26
Brabourne	2	M Chinnaswamy	15	Sawai Mansingh	27
Buffalo Park	3	MA Chidambaram	16	Shaheed Veer Narayan Singh	28
De Beers Diamond	4	Maharashtra Cricket	17	Sharjah Cricket	29
Dr DY Patil Academy	5	Narendra Modi	18	Sheikh Zayed	30
Dr. Rajasekhara Reddy	6	Nehru	19	St George's Park	31
Dubai Cricket	7	New Wanderers	20	Subrata Roy Sahara	32
Eden Gardens	8	Newlands	21	SuperSport Park	33
Feroz Shah Kotla	9	OUTsurance Oval	22	Vidarbha Cricket	34
Green Park	10	Punjab Cricket	23	Wankhede	35
Himachal Pradesh Cricket	11	Rajiv Gandhi	24	Zayed Cricket	36
Holkar Cricket	12				

Table 6.5: Encoding of Venue

◇ Accuracy Measures

From appendix

Model	Training Accuracy	Testing Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.3056	0.2632	0.23	0.26	0.23
K-Nearest Neighbors	0.5794	0.3737	0.36	0.37	0.36
Random Forest	0.8876	0.4368	0.44	0.44	0.42
Decision Tree	0.8876	0.4316	0.46	0.43	0.43

Table 6.6: Performance metrics for different models

◇ Interpretation

Random Forest Model Performance Summary

Training Accuracy: 0.8876

- **Explanation:** The Random Forest model accurately predicted approximately 88.76% of the outcomes in the training dataset. This high accuracy indicates that the model has effectively learned the patterns present in the training data.

Test Accuracy: 0.4368

- **Explanation:** The Random Forest model accurately predicted approximately 43.68% of the outcomes in the test dataset. The notably lower test accuracy compared to the training accuracy suggests that the model may be overfitting.

Overfitting:

- The substantial gap between the training accuracy (88.76%) and the test accuracy (43.68%) strongly indicates overfitting. This implies that the model has become too tailored to the specifics of the training data and does not generalize well to new, unseen data.

List of Tables

List of Figures

References

[1] https://en.wikipedia.org/wiki/Indian_Premier_League

[2] https://www.cuemath.com/data/non-parametric-test/

[3] https://www.analyticsvidhya.com/blog/2020/12/an-end-to-end-comprehensive-guide-for-pca/

[4] https://www.geeksforgeeks.org/principal-component-analysis-pca/

[5] https://github.com/Poonampatil27

[6] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*, John Wiley & Sons, 1984.

[7] V. Rajgopalan, *Selected Statistical Tests*, New Age International Publishers, 2006

[8] G. James, D. Witten, T. Hastie, R. Tibshirani, J. Taylor, *Introduction to Statistical Learning with Applications in Python*, 2021.

[9] Python Software Foundation, *Python*, Python Software Foundation, https://www.python.org/.

[10] Microsoft Corporation, *Microsoft Excel*, Microsoft Corporation, https://www.microsoft.com/en-us/microsoft-365/excel.

[11] LaTeX Project, *LaTeX*, LaTeX Project, https://www.latex-project.org/.

Appendix

IPL Ball by Ball Analysis

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

[2]:
```
# Set option to display all columns
pd.set_option('display.max_columns', None)
ipl=pd.read_csv("IPL_ball_by_ball_updated 2.csv")
ipl
```

[72]:
```
ipl.info()
```

[73]:
```
ipl.columns
```

[74]:
```
ipl['BattingTeam'].unique()
```

[75]:
```
# Dictionary mapping full team names to short forms
team_short_forms = {
    'Kolkata Knight Riders': 'KKR',
    'Royal Challengers Bangalore': 'RCB',
    'Chennai Super Kings': 'CSK',
    'Kings XI Punjab': 'PK',
    'Rajasthan Royals': 'RR',
    'Delhi Daredevils': 'DC',
    'Mumbai Indians': 'MI',
    'Deccan Chargers': 'SRH',
    'Kochi Tuskers Kerala': 'KTK',
    'Pune Warriors': 'PW',
    'Sunrisers Hyderabad': 'SRH',
    'Rising Pune Supergiants': 'RPS',
    'Gujarat Lions': 'GL',
    'Rising Pune Supergiant': 'RPS',
    'Delhi Capitals': 'DC',
    'Punjab Kings': 'PK',
    'Lucknow Super Giants': 'LSG',
    'Gujarat Titans': 'GT'
}

# Replace full team names with short forms for 'BattingTeam' column
ipl['BattingTeam'] = ipl['BattingTeam'].replace(team_short_forms)

# Replace full team names with short forms for 'FieldingTeam' column
ipl['bowling_team'] = ipl['bowling_team'].replace(team_short_forms)

# Display the updated DataFrame
ipl.head()
```

[76]:
```
pivot_table=pd.pivot_table(ipl,
                           index='batter',
                           values=['batsman_run','isWicketDelivery','ID'],
```

```
                          aggfunc={'ID': 'nunique','batsman_run':'sum','batter':
 ↪'count','isWicketDelivery':'sum'},
                          fill_value=0)
pivot_table.columns=['innings_bat','Runs', 'Balls_played','Outs']
pivot_table['SR']=pivot_table.Runs/pivot_table.Balls_played*100
pivot_table['Avg']=pivot_table.Runs/pivot_table.Outs

pivot_table.reset_index(inplace=True)
pivot_table.rename(columns={'batter': 'Player'}, inplace=True)

pivot_table
```

[77]:
```
# Calculation
centuries_count = {}
fifties_count = {}

for player in np.unique(ipl['batter']):
    df = ipl[ipl['batter'] == player]
    df2 = pd.pivot_table(df,
                         index='ID',
                         values='batsman_run',
                         aggfunc={'batsman_run': 'sum'},
                         fill_value=0)
    centuries = sum(df2['batsman_run'] >= 100)
    fifties = sum((df2['batsman_run'] >= 50) & (df2['batsman_run'] < 100))
    centuries_count[player] = centuries
    fifties_count[player] = fifties

# Convert to DataFrame
centuries_df = pd.DataFrame(list(centuries_count.items()), columns=['Player', 'Centuries'])
centuries_df['Fifties'] = centuries_df['Player'].map(fifties_count)

print(centuries_df)
```

[78]:
```
boundaries_count = {}

for player in np.unique(ipl['batter']):
    df = ipl[ipl['batter'] == player]
    fours = sum(df['batsman_run'] == 4)
    sixes = sum(df['batsman_run'] == 6)
    boundaries_count[player] = {'Fours': fours, 'Sixes': sixes}

# Convert to DataFrame
boundaries_df = pd.DataFrame.from_dict(boundaries_count, orient='index', columns=['Fours',␣
 ↪'Sixes']).reset_index()
boundaries_df.rename(columns={'index': 'Player'}, inplace=True)

print(boundaries_df)
```

[79]:
```
# Merge all DataFrames at once
merged_df = pd.merge(pivot_table, centuries_df, on='Player', how='left').merge(boundaries_df,␣
 ↪on='Player', how='left')
merged_df
```

[80]:
```
# Pivot table calculation for bowlers
bowling_pivot_table = pd.pivot_table(ipl,
                                     index='bowler',
                                     values=['ID', 'total_run', 'isWicketDelivery'],
                                     aggfunc={'ID': 'nunique',
                                              'isWicketDelivery': 'sum',
                                              'total_run': ['count','sum']},  # Calculate␣
 ↪total runs given
                                     fill_value=0)           # Calculate total wickets taken


# Rename columns
bowling_pivot_table.columns = ['innings_bowl', 'Wickets_taken',␣
 ↪'Total_balls','Total_runs_given']
```

```
# Calculate additional statistics

bowling_pivot_table['Economy'] = bowling_pivot_table['Total_runs_given'] /␣
 ↪(bowling_pivot_table['Total_balls'] / 6)
bowling_pivot_table['Bowling_avg'] = bowling_pivot_table['Total_runs_given'] /␣
 ↪bowling_pivot_table['Wickets_taken']
bowling_pivot_table['Bowling_SR'] = bowling_pivot_table['Total_balls'] /␣
 ↪bowling_pivot_table['Wickets_taken']

# Reset index and rename columns
bowling_pivot_table.reset_index(inplace=True)
bowling_pivot_table.rename(columns={'bowler': 'Player'}, inplace=True)

bowling_pivot_table
```

[81]:
```
players_performace = pd.merge(merged_df, bowling_pivot_table, on='Player', how='left')
players_performace
```

[82]:
```
# Fill NaN values in the last 7 columns
columns_to_fill = ['innings_bowl','Wickets_taken', 'Total_balls', 'Total_runs_given',␣
 ↪'Economy', 'Bowling_avg', 'Bowling_SR']
players_performace[columns_to_fill] = players_performace[columns_to_fill].fillna(0)

# Create a new column 'Matches' which is the sum of 'innings_bat' and 'innings_bowl'
players_performace.insert(1, 'Matches', players_performace['innings_bat'] +␣
 ↪players_performace['innings_bowl'])
```

[83]:
```
# Round decimal values to two decimal places
players_performace = players_performace.round({'SR': 2, 'Avg': 2, 'Economy': 2, 'Bowling_avg':
 ↪ 2, 'Bowling_SR': 2})
players_performace
```

[84]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
file_name = "players_performace_2008_2023.csv"

# Save the DataFrame to a CSV file in the current directory
players_performace.to_csv(os.path.join(current_directory, file_name), index=False)

# Now, the DataFrame 'merged_data2_sorted' is saved as a CSV file in the current working␣
 ↪directory
```

[85]:
```
ipl.columns
```

[86]:
```
# Initialize an empty dictionary to store total runs scored by each batsman against each team
batsman_runs_against_team = {}

# Iterate over each row in the DataFrame
for index, row in ipl.iterrows():
    # Ensure both 'batter' and 'FieldingTeam' are not empty
    if row['batter'] != '' and row['bowling_team'] != '':
        # If the batsman is not in the dictionary, initialize with zero runs against each team
        if row['batter'] not in batsman_runs_against_team:
            batsman_runs_against_team[row['batter']] = {team: 0 for team in team_short_forms.
 ↪values()}

        # Add runs scored by the batsman against this team
        batsman_runs_against_team[row['batter']][row['bowling_team']] += row['batsman_run']

# Create a DataFrame from the dictionary
batsman_runs_df = pd.DataFrame.from_dict(batsman_runs_against_team, orient='index')

# Sort the DataFrame by player name (index)
batsman_runs_df_sorted = batsman_runs_df.sort_index()
```

```
# Calculate the total runs for each player
batsman_runs_df_sorted['TotalRuns'] = batsman_runs_df_sorted.sum(axis=1)

# Reset the index to include the 'Player' column
batsman_runs_df_sorted.reset_index(inplace=True)

# Rename the index column to 'Player'
batsman_runs_df_sorted.rename(columns={'index': 'Player'}, inplace=True)

# Display the updated DataFrame
batsman_runs_df_sorted
```

[89]:
```
# Get the current working directory
#current_directory = os.getcwd()

# Define the file name
#file_name = "batsman_runs_against_team4.csv"

# Save the DataFrame to a CSV file in the current directory
#batsman_runs_df_sorted.to_csv(os.path.join(current_directory, file_name), index=False)
```

[90]:
```
ipl.columns
```

[91]:
```
# Initialize an empty dictionary to store total wicket taken by each bowler against each team
bowlers_wickets_against_team = {}

# Iterate over each row in the DataFrame
for index, row in ipl.iterrows():
    if row['bowler'] != '' and row['BattingTeam'] != '':
        # If the batsman is not in the dictionary, initialize with zero runs against each team
        if row['bowler'] not in bowlers_wickets_against_team:
            bowlers_wickets_against_team[row['bowler']] = {team: 0 for team in␣
 ↪team_short_forms.values()}

        # Add runs scored by the batsman against this team
        bowlers_wickets_against_team[row['bowler']][row['BattingTeam']] +=␣
 ↪row['isWicketDelivery']

# Create a DataFrame from the dictionary
bowlers_wickets_df = pd.DataFrame.from_dict(bowlers_wickets_against_team, orient='index')

# Sort the DataFrame by player name (index)
bowlers_wickets_df_sorted = bowlers_wickets_df.sort_index()

# Calculate the total runs for each player
bowlers_wickets_df_sorted['TotalWickets'] = bowlers_wickets_df_sorted.sum(axis=1)

# Reset the index to include the 'Player' column
bowlers_wickets_df_sorted.reset_index(inplace=True)

# Rename the index column to 'Player'
bowlers_wickets_df_sorted.rename(columns={'index': 'Player'}, inplace=True)

# Display the updated DataFrame
bowlers_wickets_df_sorted
```

[ ]:
```
# Get the current working directory
#current_directory = os.getcwd()

# Define the file name
#file_name = "bowlers_wickets_against_team2.csv"

# Save the DataFrame to a CSV file in the current directory
#bowlers_wickets_df_sorted.to_csv(os.path.join(current_directory, file_name), index=False)
```

[92]:
```
# Group by batter and bowling_team, and sum the batsman_run
grouped = ipl.groupby(['batter', 'bowling_team'])['batsman_run'].sum().reset_index()
```

```
# Pivot the table so that each row corresponds to a batter and each column to a bowling_team
pivot_table = grouped.pivot(index='batter', columns='bowling_team', values='batsman_run').
 ↪fillna(0)

# Initialize a dictionary to store the top 10 players for each team
top_players_per_team = {}

# Iterate through each bowling_team and find the top 10 players
for team in pivot_table.columns:
    # Sort players by runs scored against the team in descending order and get the top 10
    top_players = pivot_table[team].nlargest(10)

    # Add the team column to the dictionary
    top_players_per_team[team] = top_players

# Display the top players for each team
for team, top_players in top_players_per_team.items():
    print(f"Top players against {team}:")
    print(top_players)
    print()
```

[96]:
```
# Group the data by season
season_groups = ipl.groupby('season')

# Initialize an empty list to store the results for all seasons
all_season_results = []

# Iterate over each season
for season, season_data in season_groups:
    # Group the data by match ID
    match_groups = season_data.groupby('ID')

    # Initialize an empty list to store the results for the current season
    season_results = []

    # Iterate over each match in the current season
    for match_id, match_data in match_groups:
        # Filter 1st and 2nd innings separately
        first_innings = match_data[match_data['innings'] == 1]
        second_innings = match_data[match_data['innings'] == 2]

        # Calculate total runs scored in 1st and 2nd innings
        total_runs_1st_innings = first_innings['total_run'].sum()
        total_runs_2nd_innings = second_innings['total_run'].sum()

        # Calculate wickets taken in 1st and 2nd innings
        total_wickets_1st_innings = first_innings['isWicketDelivery'].sum()
        total_wickets_2nd_innings = second_innings['isWicketDelivery'].sum()

        # Define function to calculate runs and wickets in power play, middle overs, and␣
 ↪death overs
        def calculate_runs_wickets(data):
            power_play_runs = data[data['overs'] <= 6]['total_run'].sum()
            middle_overs_runs = data[(data['overs'] > 6) & (data['overs'] <=␣
 ↪15)]['total_run'].sum()
            death_overs_runs = data[data['overs'] > 15]['total_run'].sum()

            power_play_wickets = data[data['overs'] <= 6]['isWicketDelivery'].sum()
            middle_overs_wickets = data[(data['overs'] > 6) & (data['overs'] <=␣
 ↪15)]['isWicketDelivery'].sum()
            death_overs_wickets = data[data['overs'] > 15]['isWicketDelivery'].sum()

            return power_play_runs, middle_overs_runs, death_overs_runs, power_play_wickets,␣
 ↪middle_overs_wickets, death_overs_wickets

        # Apply the function to each innings
        first_innings_power_play_runs, first_innings_middle_overs_runs,␣
 ↪first_innings_death_overs_runs, \
```

```
            first_innings_power_play_wickets, first_innings_middle_overs_wickets,␣
↪first_innings_death_overs_wickets = calculate_runs_wickets(first_innings)

        second_innings_power_play_runs, second_innings_middle_overs_runs,␣
↪second_innings_death_overs_runs, \
            second_innings_power_play_wickets, second_innings_middle_overs_wickets,␣
↪second_innings_death_overs_wickets = calculate_runs_wickets(second_innings)

        # Store the results for the current match
        match_results = {
            'Match ID': match_id,
            'Season': season,
            'Date': first_innings['date'].iloc[0],
            'Total Runs 1st Innings': total_runs_1st_innings,
            'Total Runs 2nd Innings': total_runs_2nd_innings,
            'Total Wickets 1st Innings': total_wickets_1st_innings,
            'Total Wickets 2nd Innings': total_wickets_2nd_innings,
            'Power Play Runs 1st Innings': first_innings_power_play_runs,
            'Middle Overs Runs 1st Innings': first_innings_middle_overs_runs,
            'Death Overs Runs 1st Innings': first_innings_death_overs_runs,
            'Power Play Runs 2nd Innings': second_innings_power_play_runs,
            'Middle Overs Runs 2nd Innings': second_innings_middle_overs_runs,
            'Death Overs Runs 2nd Innings': second_innings_death_overs_runs,
            'Power Play Wickets 1st Innings': first_innings_power_play_wickets,
            'Middle Overs Wickets 1st Innings': first_innings_middle_overs_wickets,
            'Death Overs Wickets 1st Innings': first_innings_death_overs_wickets,
            'Power Play Wickets 2nd Innings': second_innings_power_play_wickets,
            'Middle Overs Wickets 2nd Innings': second_innings_middle_overs_wickets,
            'Death Overs Wickets 2nd Innings': second_innings_death_overs_wickets
        }

        season_results.append(match_results)

    # Convert the results for the current season into a DataFrame and append it to the list␣
↪of results for all seasons
    season_results_df = pd.DataFrame(season_results)
    all_season_results.append(season_results_df)

# Concatenate the results for all seasons into a single DataFrame
final_results = pd.concat(all_season_results, ignore_index=True)

final_results
```

[97]:
```
# Get the current working directory
#current_directory = os.getcwd()

# Define the file name
#file_name = "Runs_and_wickets_all_matches_2008_2023.csv"

# Save the DataFrame to a CSV file in the current directory
#final_results.to_csv(os.path.join(current_directory, file_name), index=False)
```

[5]:
```
# Find the best batsman with the most runs for each season
best_batsmen = ipl.groupby(['season', 'batter']).agg({'total_run': 'sum'}).reset_index()
best_batsman_per_season = best_batsmen.groupby('season').apply(lambda x: x.nlargest(1,␣
↪'total_run')).reset_index(drop=True)
print("\nBest Batsman with Most Runs for Each Season:")
best_batsman_per_season
```

[7]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
file_name = "Purple cap.csv"

# Save the DataFrame to a CSV file in the current directory
best_batsman_per_season.to_csv(os.path.join(current_directory, file_name), index=False)
```

[6]:
```
# Find the best bowler with the most wickets for each season
best_bowlers = ipl[ipl['isWicketDelivery'] == 1].groupby(['season', 'bowler']).size().
 ↪reset_index(name='wickets')
best_bowler_per_season = best_bowlers.groupby('season').apply(lambda x: x.nlargest(1,␣
 ↪'wickets')).reset_index(drop=True)

print("\nBest Bowler with Most Wickets for Each Season:")
best_bowler_per_season
```

[8]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
file_name = "orange cap.csv"

# Save the DataFrame to a CSV file in the current directory
best_bowler_per_season.to_csv(os.path.join(current_directory, file_name), index=False)
```

Exploratory Data Analysis

[78]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[79]:
```
# Set display options to show all columns
pd.set_option('display.max_columns', None)
# Load all columns from the CSV file into a DataFrame
df = pd.read_csv(r"C:\Users\VISHAL\Desktop\project\2008-2023 IPL ANALYSIS.csv")
df
```

[80]:
```
df.head()
```

[81]:
```
df['Team1'].unique()
```

[82]:
```
import pandas as pd


# Dictionary mapping full team names to short forms
team_name_mapping = {
    'Royal Challengers Bangalore': 'RCB',
    'Kings XI Punjab': 'PBKS',
    'Delhi Daredevils': 'DC',
    'Mumbai Indians': 'MI',
    'Kolkata Knight Riders': 'KKR',
    'Rajasthan Royals': 'RR',
    'Deccan Chargers': 'SRH',
    'Chennai Super Kings': 'CSK',
    'Kochi Tuskers Kerala': 'KTK',
    'Pune Warriors': 'PW',
    'Sunrisers Hyderabad': 'SRH',
    'Gujarat Lions': 'GL',
    'Rising Pune Supergiants': 'RPS',
    'Delhi Capitals': 'DC',
    'Punjab Kings': 'PBKS',
    'Lucknow Super Giants': 'LSG',
    'Gujarat Titans': 'GT',
    'Tie': 'Tie'
}

# Check for team names in columns that are not in the mapping dictionary
for column in ['Team1', 'Team2', 'TossWinner', 'WinningTeam']:
    missing_teams = df.loc[~df[column].isin(team_name_mapping.keys()), column].unique()
    if len(missing_teams) > 0:
        print(f"Missing teams in column '{column}': {missing_teams}")

# Replace team names in DataFrame columns with their short forms
```

```
df['Team1'] = df['Team1'].map(team_name_mapping)
df['Team2'] = df['Team2'].map(team_name_mapping)
df['TossWinner'] = df['TossWinner'].map(team_name_mapping)
df['WinningTeam'] = df['WinningTeam'].map(team_name_mapping)

# Remove rows where WinningTeam is 'Tie'
df = df[df['WinningTeam'] != 'Tie']
df
```

[83]:
```
# Display basic information about the dataset
print("Dataset Information:")
df.info()
```

[84]:
```
# Drop irrelevant columns
df2=df.drop(['ID', 'MatchNumber','Season_x','Margin'], axis=1)
```

[85]:
```
# Display summary statistics
print("\nSummary Statistics:")
df2.describe()
```

[86]:
```
df2.columns
```

[87]:
```
# Assuming you already have venue_counts from previous code
venue_counts = df['Venue'].value_counts()

# Plotting the bar chart
plt.figure(figsize=(12, 6))
bar_chart = venue_counts.plot(kind='bar', color='skyblue')

# Annotate each bar with its count value
for index, value in enumerate(venue_counts):
    bar_chart.text(index, value + 0.1, str(value), ha='center', va='bottom', rotation=0)

plt.title('Number of Matches in Each Venue')
plt.xlabel('Venue')
plt.ylabel('Number of Matches')
plt.xticks(rotation=90)
plt.show()
```

[91]:
```
# Filter the DataFrame to include only matches where the winning team decided to bat or bowl␣
 ↪after winning the toss
winning_toss_decision = df[df['TossWinner'] == df['WinningTeam']]

# Group by 'Venue' and 'TossDecision' and count the occurrences
winning_toss_decision_counts = winning_toss_decision.groupby(['Venue', 'TossDecision']).
 ↪size().unstack(fill_value=0)

# Sort the DataFrame by the total count of match wins for each venue
winning_toss_decision_counts['Total'] = winning_toss_decision_counts.sum(axis=1)
winning_toss_decision_counts = winning_toss_decision_counts.sort_values(by='Total',␣
 ↪ascending=False).drop(columns='Total')

# Plotting
ax = winning_toss_decision_counts.plot(kind='bar', stacked=True, figsize=(12, 8), width=0.8) ␣
 ↪# Adjust width of bars
plt.title('Count of Match Wins by Toss Decision at Each Venue')
plt.xlabel('Venue')
plt.ylabel('Count of Match Wins')
plt.legend(title='Toss Decision')
plt.xticks(rotation=90)

# Adding counts on the bars
for p in ax.patches:
    ax.annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2., p.get_height()),␣
 ↪ha='center', va='center', xytext=(0, 5), textcoords='offset points')

plt.show()
```

[89]:
```
# Plotting the countplot
plt.figure(figsize=(10, 6))
count_plot = sns.countplot(x='City', data=df)

# Annotate each bar with its count value
for p in count_plot.patches:
    count_plot.annotate(format(p.get_height(), '.0f'),
                        (p.get_x() + p.get_width() / 2., p.get_height()),
                        ha = 'center', va = 'center',
                        xytext = (0, 5),
                        textcoords = 'offset points',
                        rotation=0)

plt.xticks(rotation=90)
plt.title('Distribution of Matches by City')
plt.show()
```

[90]:
```
# Plotting the countplot
plt.figure(figsize=(10, 6))
count_plot = sns.countplot(x='Season_x', data=df)

# Annotate each bar with its count value
for p in count_plot.patches:
    count_plot.annotate(format(p.get_height(), '.0f'),
                        (p.get_x() + p.get_width() / 2., p.get_height()),
                        ha = 'center', va = 'center',
                        xytext = (0, 5),
                        textcoords = 'offset points')

plt.title('Distribution of Matches by Season')
plt.show()
```

[46]:
```
# Calculate the number of wins and losses for teams batting second
num_of_wins = (df['TossDecision'] == 'field').sum()
num_of_loss = (df['TossDecision'] == 'bat').sum()

# Create labels and sizes for the pie chart
labels = ["Wins", "Loss"]
total = float(num_of_wins + num_of_loss)
sizes = [(num_of_wins / total) * 100, (num_of_loss / total) * 100]

# Define colors for the pie chart
colors = ['gold', 'lightskyblue']

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90)
plt.title("Win Percentage Batting Second")
plt.show()
```

[47]:
```
# Exclude non-numeric columns before calculating correlation
numeric_columns = df2.select_dtypes(include=np.number)

# Correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(numeric_columns.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

[48]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
ax = sns.countplot(x='Season_x', hue='TossDecision', data=df)

plt.title('Countplot of Toss Decision by Season')
plt.xlabel('Season')
plt.ylabel('Count')
plt.legend(title='Toss Decision')
```

```
# Add counts on each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.xticks(rotation=45)
plt.show()
```

[49]:
```
# Countplot of winning teams based on toss decision
plt.figure(figsize=(10, 6))
ax = sns.countplot(x='TossDecision', hue='WinningTeam', data=df)

plt.title('Effect of Toss Decision on Winning Team')
plt.xlabel('Toss Decision')
plt.ylabel('Count')
plt.legend(title='Winning Team', bbox_to_anchor=(1.05, 1), loc='upper left')

# Add counts on each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.show()
```

[50]:
```
# Combine 'Team1' and 'Team2' columns into one Series
teams = pd.concat([df['Team1'], df['Team2']])

# Count the occurrences of each team
team_counts = teams.value_counts()

# Plotting the bar chart
plt.figure(figsize=(12, 6))
ax = team_counts.plot(kind='bar', color='skyblue')

plt.title('Number of Matches Played by Each Team')
plt.xlabel('Team')
plt.ylabel('Number of Matches Played')
plt.xticks(rotation=45)

# Add counts on each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.show()
```

[51]:
```
# Identify the finals by keeping the last match of each season
finals = df.drop_duplicates('Season_x', keep='last')
finals = finals[['Season_x', 'Team1', 'Team2', 'WinningTeam']]

# Concatenate team1 and team2 to find teams that reached the final
teams_reaching_finals = pd.concat([finals['Team1'], finals['Team2']])
teams_reaching_finals_count = teams_reaching_finals.value_counts().reset_index()
teams_reaching_finals_count.columns = ['Team', 'Final_Count']

# Count the number of wins in the final for each team
finals_wins_count = finals['WinningTeam'].value_counts().reset_index()
finals_wins_count.columns = ['Team', 'Final_Wins']

# Merge the two dataframes to get the final result
finals_summary = teams_reaching_finals_count.merge(finals_wins_count, on='Team', how='outer').
 ↪fillna(0)
finals_summary.set_index('Team', inplace=True)
finals_summary.columns = ['Number of Times Finals Played', 'Number of Times Finals Won']
```

```
# Plotting the bar chart
plt.figure(figsize=(13, 7))
ax = finals_summary.plot(kind='bar', fontsize=12, title='Team played in finals and their␣
 ↪wins')

plt.xlabel('Team', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(rotation=45)

# Add counts on each bar
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                textcoords='offset points')

plt.show()
```

[68]:
```
# Calculate average runs for each phase for the first inning grouped by venue
average_runs_by_phase = df.groupby('Venue').agg({
    'Middle Overs Runs 1st Innings': 'mean',
    'Power Play Runs 1st Innings': 'mean',
    'Death Overs Runs 1st Innings': 'mean'
}).reset_index()

# Melt the DataFrame to have a long format for seaborn plotting
average_runs_melted = average_runs_by_phase.melt(id_vars='Venue', var_name='Phase',␣
 ↪value_name='Average Runs')

# Plotting the subdivided bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=average_runs_melted, x='Venue', y='Average Runs', hue='Phase',␣
 ↪palette='coolwarm')
plt.title('Average Runs in Different Phases of the First Inning by Venue')
plt.xlabel('Venue')
plt.ylabel('Average Runs')
plt.xticks(rotation=90)
plt.legend(title='Phase')
plt.tight_layout()
plt.show()
```

[69]:
```
# Calculate average runs for each phase for the second inning grouped by venue
average_runs_second_inning = df.groupby('Venue').agg({
    'Middle Overs Runs 2nd Innings': 'mean',
    'Power Play Runs 2nd Innings': 'mean',
    'Death Overs Runs 2nd Innings': 'mean'
}).reset_index()

# Melt the DataFrame to have a long format for seaborn plotting
average_runs_second_inning_melted = average_runs_second_inning.melt(id_vars='Venue',␣
 ↪var_name='Phase', value_name='Average Runs')

# Plotting the subdivided bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=average_runs_second_inning_melted, x='Venue', y='Average Runs', hue='Phase',␣
 ↪palette='coolwarm')
plt.title('Average Runs in Different Phases of the Second Inning by Venue')
plt.xlabel('Venue')
plt.ylabel('Average Runs')
plt.xticks(rotation=90)
plt.legend(title='Phase')
plt.tight_layout()
plt.show()
```

[75]:
```
# Calculate average runs for each phase for the second inning grouped by venue
average_Wickets_first_inning = df.groupby('Venue').agg({
    'Middle Overs Wickets 1st Innings': 'mean',
    'Power Play Wickets 1st Innings': 'mean',
    'Death Overs Wickets 1st Innings': 'mean'
}).reset_index()
```

```
# Melt the DataFrame to have a long format for seaborn plotting
average_Wickets_first_inning_melted = average_Wickets_first_inning.melt(id_vars='Venue',␣
 ↪var_name='Phase', value_name='Average Wickets')

# Plotting the subdivided bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=average_Wickets_first_inning_melted, x='Venue', y='Average Wickets',␣
 ↪hue='Phase', palette='coolwarm')
plt.title('Average Wickets in Different Phases of the First Inning by Venue')
plt.xlabel('Venue')
plt.ylabel('Average Wickets')
plt.xticks(rotation=90)
plt.legend(title='Phase')
plt.tight_layout()
plt.show()
```

[76]:
```
# Calculate average runs for each phase for the second inning grouped by venue
average_wickets_second_inning = df.groupby('Venue').agg({
    'Middle Overs Wickets 2nd Innings': 'mean',
    'Power Play Wickets 2nd Innings': 'mean',
    'Death Overs Wickets 2nd Innings': 'mean'
}).reset_index()

# Melt the DataFrame to have a long format for seaborn plotting
average_wickets_second_inning_melted = average_wickets_second_inning.melt(id_vars='Venue',␣
 ↪var_name='Phase', value_name='Average wickets')

# Plotting the subdivided bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=average_wickets_second_inning_melted, x='Venue', y='Average wickets',␣
 ↪hue='Phase', palette='coolwarm')
plt.title('Average Wickets in Different Phases of the Second Inning by Venue')
plt.xlabel('Venue')
plt.ylabel('Average wickets')
plt.xticks(rotation=90)
plt.legend(title='Phase')
plt.tight_layout()
plt.show()
```

[73]:
```
# Calculate average runs for each phase for the second inning grouped by venue
average_runs_second_inning = df.groupby('Venue').agg({
    'Total Wickets 1st Innings': 'mean',
    'Total Wickets 2nd Innings': 'mean'
}).reset_index()

# Melt the DataFrame to have a long format for seaborn plotting
average_runs_second_inning_melted = average_runs_second_inning.melt(id_vars='Venue',␣
 ↪var_name='Phase', value_name='Average Wickets')

# Plotting the subdivided bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=average_runs_second_inning_melted, x='Venue', y='Average Wickets',␣
 ↪hue='Phase', palette='coolwarm')
plt.title('Average Wickets in 1st and 2nd Inning by Venue')
plt.xlabel('Venue')
plt.ylabel('Average Wickets')
plt.xticks(rotation=90)
plt.legend(title='Phase')
plt.tight_layout()
plt.show()
```

[74]:
```
# Calculate average runs for each phase for the second inning grouped by venue
average_runs_second_inning = df.groupby('Venue').agg({
    'Total Runs 1st Innings': 'mean',
    'Total Runs 2nd Innings': 'mean'
}).reset_index()
```

```
# Melt the DataFrame to have a long format for seaborn plotting
average_runs_second_inning_melted = average_runs_second_inning.melt(id_vars='Venue',␣
 ↪var_name='Phase', value_name='Average Runs')

# Plotting the subdivided bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=average_runs_second_inning_melted, x='Venue', y='Average Runs', hue='Phase',␣
 ↪palette='coolwarm')
plt.title('Average Runs in 1st and 2nd Inning by Venue')
plt.xlabel('Venue')
plt.ylabel('Average Runs')
plt.xticks(rotation=90)
plt.legend(title='Phase')
plt.tight_layout()
plt.show()
```

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import os
import pypandoc
```

[33]:
```
# Set option to display all columns
pd.set_option('display.max_columns', None)
ipl=pd.read_csv("IPL_Matches_2008_2023.csv")
ipl.head(1)
```

[3]:
```
# Set option to display all columns
pd.set_option('display.max_columns', None)
ipl2=pd.read_csv("Runs_and_wickets_all_matches_2008_2023.csv")
ipl2.head(1)
```

[4]:
```
# Merge datasets on the 'ID' variable
ipl3 = pd.merge(ipl, ipl2, on='ID', how='outer')

ipl3.head(1) # Print the merged DataFrame to verify the result
```

[5]:
```
# Selecting the last 16 columns from the dataset
variables = ['Total Runs 1st Innings', 'Total Runs 2nd Innings', 'Total Wickets 1st Innings',
             'Total Wickets 2nd Innings', 'Power Play Runs 1st Innings', 'Middle Overs Runs␣
 ↪1st Innings',
             'Death Overs Runs 1st Innings', 'Power Play Runs 2nd Innings', 'Middle Overs␣
 ↪Runs 2nd Innings',
             'Death Overs Runs 2nd Innings', 'Power Play Wickets 1st Innings', 'Middle Overs␣
 ↪Wickets 1st Innings',
             'Death Overs Wickets 1st Innings', 'Power Play Wickets 2nd Innings', 'Middle␣
 ↪Overs Wickets 2nd Innings',
             'Death Overs Wickets 2nd Innings']

# Create normal probability plots for each variable
for variable in variables:
    plt.figure()
    stats.probplot(ipl3[variable], dist="norm", plot=plt)
    plt.title(f'Normal Probability Plot - {variable}')
    plt.show()
```

[6]:
```
#last_16_columns = ipl3.iloc[:, -16:]

# Create the pair plot
#sns.pairplot(last_16_columns)
#plt.suptitle('Pair Plot of Last 16 Columns', y=1.02)
#plt.show()
```

[7]:
```
ipl3.info()
```

[8]:
```
# Drop the specified variables
ipl3 = ipl3.drop(columns=['Team1Players', 'Team2Players', 'method', 'Season_y', 'Date_y',␣
 ↪'SuperOver'])

ipl3.head(1)  # Print the DataFrame to verify the changes
```

[9]:
```
# Get the current working directory
#current_directory = os.getcwd()

# Define the file name
#file_name = "2008-2023 IPL ANALYSIS.csv"

# Save the DataFrame to a CSV file in the current directory
#ipl3.to_csv(os.path.join(current_directory, file_name), index=False)

# Now, the DataFrame 'merged_data2_sorted' is saved as a CSV file in the current working␣
 ↪directory
```

[10]:
```
ipl3.columns
```

[11]:
```
ipl['Team1'].unique()
```

[12]:
```
last_16_columns = ipl3.iloc[:, -16:]
descriptive_stats = last_16_columns.describe()
descriptive_stats
```

[18]:
```
# Create a dictionary mapping team names to their short forms
team_name_mapping = {
    'Delhi Daredevils': 'DC',
    'Pune Warriors': 'PW',
    'Rajasthan Royals': 'RR',
    'Gujarat Lions': 'GL',
    'Delhi Capitals': 'DC',
    'Kings XI Punjab': 'PK',
    'Lucknow Super Giants': 'LSG',
    'Chennai Super Kings': 'CSK',
    'Gujarat Titans': 'GT',
    'Kolkata Knight Riders': 'KKR',
    'Punjab Kings': 'PK',
    'Rising Pune Supergiants': 'RPS',
    'Mumbai Indians': 'MI',
    'Deccan Chargers': 'SRH',
    'Sunrisers Hyderabad': 'SRH',
    'Royal Challengers Bangalore': 'RCB',
    'Kochi Tuskers Kerala': 'KTK'
}

# Replace team names with their short forms in the DataFrame
ipl3['Team1'] = ipl3['Team1'].replace(team_name_mapping)
ipl3['Team2'] = ipl3['Team2'].replace(team_name_mapping)
ipl3['WinningTeam'] = ipl3['WinningTeam'].replace(team_name_mapping)
ipl3['TossWinner'] = ipl3['TossWinner'].replace(team_name_mapping)

# Display the updated DataFrame
ipl3
```

[19]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
file_name = "ipl3.csv"

# Save the DataFrame to a CSV file in the current directory
ipl3.to_csv(os.path.join(current_directory, file_name), index=False)

# Now, the DataFrame 'merged_data2_sorted' is saved as a CSV file in the current working␣
 ↪directory
```

[20]:
```
# Drop the rows where 'result' column contains 'Tie'
ipl3[ipl3['WinningTeam'] != 'Tie']
#ipl4.info()
```

[21]:
```
def calculate_win_loss_ratio(team_name, ipl3):
    team_matches = ipl3[(ipl3['Team1'] == team_name) | (ipl3['Team2'] == team_name)]
    num_matches_won = len(team_matches[team_matches['WinningTeam'] == team_name])
    num_matches_lost = len(team_matches) - num_matches_won

    if num_matches_lost == 0:
        return float('inf')  # To handle cases where there are no losses

    return num_matches_won / num_matches_lost

def identify_winning_streaks(team_name, ipl):
    team_matches = ipl3[(ipl3['Team1'] == team_name) | (ipl3['Team2'] == team_name)]
    streaks = []
    current_streak = 0

    for index, match in team_matches.iterrows():
        if match['WinningTeam'] == team_name:
            current_streak += 1
        else:
            if current_streak > 0:
                streaks.append(current_streak)
            current_streak = 0

    # If the team is currently on a winning streak
    if current_streak > 0:
        streaks.append(current_streak)

    return streaks

# Get unique team names
teams = set(ipl3['Team1'].unique()).union(set(ipl3['Team2'].unique()))

# Create lists to store team metrics
team_names = []
win_loss_ratios = []
winning_streaks = []

# Calculate metrics for each team
for team in teams:
    win_loss_ratio = calculate_win_loss_ratio(team, ipl3)
    winning_streak = identify_winning_streaks(team, ipl3)
    team_names.append(team)
    win_loss_ratios.append(win_loss_ratio)
    winning_streaks.append(winning_streak)

# Create DataFrame from the lists
team_metrics_df = pd.DataFrame({
    'Team': team_names,
    'Win-Loss Ratio': win_loss_ratios,
    'Winning Streaks': winning_streaks
})

# Display the DataFrame
team_metrics_df
```

6.3.1 Total Matches Played by Each Team Against other Team

[22]:
```
def matches_between_teams(team1, team2, ipl3):
    matches = ipl3[((ipl3['Team1'] == team1) & (ipl3['Team2'] == team2)) | ((ipl3['Team1'] ==␣
 ↪team2) & (ipl3['Team2'] == team1))]
    return len(matches)

# Get unique team names
teams = ipl3['Team1'].unique().tolist() + ipl3['Team2'].unique().tolist()
```

```
teams = list(set(teams))  # Remove duplicates

# Create list of dictionaries to store matches for each team
team_data = []

# Calculate matches for each team against all other teams
for team in teams:
    row = {'Team': team}
    total_matches = 0
    for opponent in teams:
        if team != opponent:
            num_matches = matches_between_teams(team, opponent, ipl3)
            row[opponent] = num_matches
            total_matches += num_matches
    # Append total matches played by the team
    row['Total'] = total_matches
    team_data.append(row)

# Create DataFrame
df1 = pd.DataFrame(team_data)

# Set 'Team' column as index
df1.set_index('Team', inplace=True)

# Add a column for the total matches played against each team
df1['Total'] = df1.sum(axis=0)

# Move the 'Total' column to the last position
total_column = df1['Total']
df1.drop(columns=['Total'], inplace=True)
df1['Total'] = total_column

# Move the second last column to the first position
cols = list(df1.columns)
second_last_col = cols[-2]
cols.remove(second_last_col)
cols.insert(0, second_last_col)
df1 = df1[cols]

# Add a row for the totals of each team
df1.loc['Team Totals'] = df1.sum()

# Add a row for the grand total
#df.loc['Grand Total'] = df.sum(numeric_only=True)

# Display DataFrame
df1.fillna(0, inplace=True)
df1
```

6.3.2 Total wins and losses

[23]:
```
# Filter the data where Team1 and Team2 are not null
matches_df = ipl3.dropna(subset=["Team1", "Team2"])

# Initialize an empty dictionary to store the count of unique teams for each team
team_counts = {}

# Get unique teams in the order they appear
unique_teams = matches_df["Team1"].unique()

# Iterate through each team
for team in unique_teams:
    # Filter the matches where the current team is either Team1 or Team2
    team_matches = matches_df[(matches_df["Team1"] == team) | (matches_df["Team2"] == team)]

    # Get the unique count of teams in the WinningTeam column for the current team's matches
    unique_win_counts = team_matches["WinningTeam"].value_counts()

    # Set count for the same team to zero
```

```
    unique_win_counts[team] = 0

    # Store the count in the dictionary
    team_counts[team] = unique_win_counts

# Create a DataFrame from the dictionary with index and columns set to unique_teams
team_counts_df = pd.DataFrame(team_counts, index=unique_teams).fillna(0)

# Add row sums (total wins for each team)
team_counts_df["Total_Wins"] = team_counts_df.sum(axis=1)

# Add column sums (total losses against each team)
team_counts_df.loc["Total_Losses"] = team_counts_df.sum()

# Display the DataFrame
team_counts_df
```

6.3.3 overall winning and losing probabiity of each team

[24]:
```
# Get unique team names
teams = pd.concat([ipl3['Team1'], ipl3['Team2']]).unique()

# Calculate total matches played by each team
total_matches = pd.concat([ipl3['Team1'], ipl3['Team2']]).value_counts().reset_index()
total_matches.columns = ['Team', 'Total Matches']

# Calculate total wins for each team
wins = ipl3['WinningTeam'].value_counts().reset_index()
wins.columns = ['Team', 'Wins']

# Calculate total losses for each team
losses_team1 = ipl3[ipl3['WinningTeam'] != ipl3['Team1']]['Team1'].value_counts().
 ↪reset_index()
losses_team1.columns = ['Team', 'Losses']

losses_team2 = ipl3[ipl3['WinningTeam'] != ipl3['Team2']]['Team2'].value_counts().
 ↪reset_index()
losses_team2.columns = ['Team', 'Losses']

# Merge wins and losses data
wins_losses = pd.merge(wins, losses_team1, on='Team', how='outer').merge(losses_team2,␣
 ↪on='Team', how='outer').fillna(0)

# Merge with total matches
team_probabilities = pd.merge(total_matches, wins_losses, on='Team', how='outer').fillna(0)

# Calculate winning and losing probabilities
team_probabilities['Winning Probability'] = team_probabilities['Wins'] /␣
 ↪team_probabilities['Total Matches']

# Calculate losses based on wins and total matches
team_probabilities['Losses'] = team_probabilities['Total Matches'] -␣
 ↪team_probabilities['Wins']

# Calculate losing probability
team_probabilities['Losing Probability'] = team_probabilities['Losses'] /␣
 ↪team_probabilities['Total Matches']

# Display DataFrame
team_probabilities.T
```

[25]:
```
# Get unique team names
teams = pd.concat([ipl3['Team1'], ipl3['Team2']]).unique()

# Calculate total matches played by each team
total_matches = pd.concat([ipl3['Team1'], ipl3['Team2']]).value_counts().reset_index()
total_matches.columns = ['Team', 'Total Matches']
```

```
# Calculate total wins for each team
wins = ipl3['WinningTeam'].value_counts().reset_index()
wins.columns = ['Team', 'Wins']

# Calculate losses when the team plays as Team1
losses_team1 = ipl3[ipl3['WinningTeam'] != ipl3['Team1']]['Team1'].value_counts().
 ↪reset_index()
losses_team1.columns = ['Team', 'Losses_x']

# Calculate losses when the team plays as Team2
losses_team2 = ipl3[ipl3['WinningTeam'] != ipl3['Team2']]['Team2'].value_counts().
 ↪reset_index()
losses_team2.columns = ['Team', 'Losses_y']

# Merge with total matches
team_statistics = pd.merge(total_matches, wins, on='Team', how='outer').fillna(0)
team_statistics = pd.merge(team_statistics, losses_team1, on='Team', how='outer').fillna(0)
team_statistics = pd.merge(team_statistics, losses_team2, on='Team', how='outer').fillna(0)

# Calculate winning and losing probabilities
team_statistics['Winning Probability'] = team_statistics['Wins'] / team_statistics['Total␣
 ↪Matches']
team_statistics['Losses'] = team_statistics['Losses_x'] + team_statistics['Losses_y']
team_statistics['Losing Probability'] = team_statistics['Losses'] / team_statistics['Total␣
 ↪Matches']

# Drop intermediate columns
team_statistics.drop(['Losses_x', 'Losses_y'], axis=1, inplace=True)

# Display DataFrame
team_statistics.T
```

[27]:
```
# Filter the data where Team1, Team2, and WinningTeam are not null
matches_df = ipl3.dropna(subset=["Team1", "Team2", "WinningTeam"])

# Initialize dictionaries to store counts and probabilities
win_counts = {}
total_counts = {}

# Iterate through each match in the dataset
for index, match in matches_df.iterrows():
    team1 = match["Team1"]
    team2 = match["Team2"]
    winning_team = match["WinningTeam"]

    # Increment win count for the winning team
    if winning_team not in win_counts:
        win_counts[winning_team] = {}
    if team1 not in win_counts[winning_team]:
        win_counts[winning_team][team1] = 0
    if team2 not in win_counts[winning_team]:
        win_counts[winning_team][team2] = 0
    win_counts[winning_team][team1] += 1
    win_counts[winning_team][team2] += 1

    # Increment total count for the pair of teams
    if team1 != team2:  # Exclude same team pairs
        if (team1, team2) not in total_counts:
            total_counts[(team1, team2)] = 0
        if (team2, team1) not in total_counts:
            total_counts[(team2, team1)] = 0
        total_counts[(team1, team2)] += 1
        total_counts[(team2, team1)] += 1

# Calculate winning probabilities
winning_probabilities = {}
for winning_team, counts in win_counts.items():
    winning_probabilities[winning_team] = {}
    for team, count in counts.items():
```

```
        total_matches = total_counts.get((winning_team, team), 0)  # Get total matches, default to 0 if not found
        if total_matches != 0:
            winning_probabilities[winning_team][team] = round(count / total_matches, 4)
        else:
            winning_probabilities[winning_team][team] = 0.0

# Create DataFrame from winning probabilities with index and columns explicitly set
winning_probabilities_df = pd.DataFrame(winning_probabilities, index=sorted(win_counts.keys()))

# Sort columns based on index
winning_probabilities_df = winning_probabilities_df.sort_index(axis=1)
# Replace NaN values with 0
winning_probabilities_df.fillna(0, inplace=True)

# Display the DataFrame
win_prob=winning_probabilities_df.drop(['Tie'], axis=1)
win_prob
```

[28]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
file_name = "winning probabilities.csv"

# Save the DataFrame to a CSV file in the current directory
win_prob.to_csv(os.path.join(current_directory, file_name), index=False)

# Now, the DataFrame 'merged_data2_sorted' is saved as a CSV file in the current working directory
```

6.4 STADIUM WISE ANALYSIS

6.4.1 No. of matches Played on each stadium

[31]:
```
# Count the number of matches played on each venue
venue_match_count = ipl3['Venue'].value_counts()

# Plot the count of matches played on each venue
plt.figure(figsize=(10, 8))
venue_match_count.plot(kind='barh', color='skyblue')
plt.title('Number of Matches Played on Each Venue')
plt.xlabel('Number of Matches')
plt.ylabel('Venue')
plt.tight_layout()

# Annotate each bar with its count
for i, count in enumerate(venue_match_count):
    plt.text(count + 1, i, str(count), ha='right', va='center', fontsize=9)

plt.show()
```

6.4.2 match winning by toss decision at each venue

[34]:
```
# Filter the DataFrame to include only matches where the winning team decided to bat or bowl after winning the toss
winning_toss_decision = ipl3[ipl3['TossWinner'] == ipl3['WinningTeam']]

# Group by 'Venue' and 'TossDecision' and count the occurrences
winning_toss_decision_counts = winning_toss_decision.groupby(['Venue', 'TossDecision']).size().unstack(fill_value=0)

# Sort the DataFrame by the total count of match wins for each venue
winning_toss_decision_counts['Total'] = winning_toss_decision_counts.sum(axis=1)
winning_toss_decision_counts = winning_toss_decision_counts.sort_values(by='Total', ascending=False).drop(columns='Total')
```

```
# Plotting
ax = winning_toss_decision_counts.plot(kind='bar', stacked=True, figsize=(14, 6), width=0.8) ␣
 ↪# Adjust width of bars
plt.title('Count of Match Wins by Toss Decision at Each Venue')
plt.xlabel('Venue')
plt.ylabel('Count of Match Wins')
plt.legend(title='Toss Decision')
plt.xticks(rotation=90)

# Adding counts on the bars
for p in ax.patches:
    ax.annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2., p.get_height()),␣
 ↪ha='center', va='center', xytext=(0, 5), textcoords='offset points')

plt.show()
```

[75]:
```
# Extracting columns 22 to 37
inning_columns = ipl3.columns[16:,]

# Grouping by 'Venue' and calculating the mean for the specified columns
venue_avg_innings_scores = ipl3.groupby('Venue')[inning_columns].mean()
venue_avg_innings_scores_rounded = venue_avg_innings_scores.round(0)
sorted_df = venue_avg_innings_scores_rounded.sort_values(by=['Total Runs 1st Innings'],␣
 ↪ascending=False)
sorted_df
```

[76]:
```
# Group data by venue and calculate the minimum first innings score for each venue
venue_min_score = ipl3.groupby('Venue')['Total Runs 1st Innings'].min()

# Create an empty DataFrame to store the results
result_df = pd.DataFrame(columns=['Venue', 'Min Score for 95% Probability of Winning'])

# Determine the minimum first innings score for a 95% probability of winning for each venue
for venue, min_score in venue_min_score.items():
    # Filter data for the venue and minimum score
    venue_data = ipl3[(ipl3['Venue'] == venue) & (ipl3['Total Runs 1st Innings'] >=␣
 ↪min_score)]

    # Calculate the winning probability based on the team batting first
    total_matches = len(venue_data)
    toss_winner = venue_data['TossWinner']
    toss_decision = venue_data['TossDecision']
    bat_first_team = toss_winner.where(toss_decision == 'bat', venue_data['Team2'])
    win_count = len(venue_data[venue_data['WinningTeam'] == bat_first_team])

    # Handle division by zero
    if total_matches == 0:
        win_prob = 0
    else:
        win_prob = win_count / total_matches

    # Iterate through scores to find the minimum score for 95% probability of winning
    min_score_95 = min_score
    while win_prob < 0.95 and total_matches > 0:  # Check total_matches > 0 to prevent␣
 ↪division by zero
        min_score_95 += 1
        venue_data = ipl3[(ipl3['Venue'] == venue) & (ipl3['Total Runs 1st Innings'] >=␣
 ↪min_score_95)]
        total_matches = len(venue_data)
        toss_winner = venue_data['TossWinner']
        toss_decision = venue_data['TossDecision']
        bat_first_team = toss_winner.where(toss_decision == 'bat', venue_data['Team2'])
        win_count = len(venue_data[venue_data['WinningTeam'] == bat_first_team])

        # Handle division by zero
        if total_matches == 0:
            win_prob = 0
        else:
```

```
            win_prob = win_count / total_matches

    # Append the result to the DataFrame
    result_df = pd.concat([result_df, pd.DataFrame([{'Venue': venue, 'Min Score for 95%␣
 ↪Probability of Winning': min_score_95}])], ignore_index=True)

print("Minimum First Innings Score for 95% Probability of Winning at Each Venue:")
result_df
```

Hypothesis Testing

[77]:
```
from scipy.stats import chi2_contingency
import pandas as pd

# Assuming ipl3 is your DataFrame containing the data

# Count of toss-winning teams choosing to bat and their corresponding wins and losses
bat_decisions = ipl3[ipl3['TossDecision'] == 'bat']
bat_win_count = len(bat_decisions[bat_decisions['WinningTeam'] ==␣
 ↪bat_decisions['TossWinner']])
bat_loss_count = len(bat_decisions) - bat_win_count

# Count of toss-winning teams choosing to field and their corresponding wins and losses
field_decisions = ipl3[ipl3['TossDecision'] == 'field']
field_win_count = len(field_decisions[field_decisions['WinningTeam'] ==␣
 ↪field_decisions['TossWinner']])
field_loss_count = len(field_decisions) - field_win_count

# Create a contingency table
contingency_table = pd.DataFrame({
    'Decision': ['Bat', 'Field'],
    'Wins': [bat_win_count, field_win_count],
    'Losses': [bat_loss_count, field_loss_count]
})

# Perform chi-square test
chi2_statistic, p_value, dof, expected = chi2_contingency(contingency_table[['Wins',␣
 ↪'Losses']])

# Print test results
print("Chi-Square Test Results:")
print("Chi-Square Statistic:", chi2_statistic)
print("Degrees of Freedom:", dof)
print("p-value:", p_value)
print("Expected Frequencies Table:")
print(expected)

# Interpret results
alpha = 0.05
if p_value < alpha:
    conclusion = "Reject null hypothesis: There is a significant association between toss␣
 ↪decision and match outcome."
else:
    conclusion = "Fail to reject null hypothesis: There is no significant association between␣
 ↪toss decision and match outcome."

# Print contingency table
print("\nContingency Table:")
print(contingency_table)

# Null and Alternative Hypotheses
null_hypothesis = "H0: There is no association between toss decision and match outcome."
alternative_hypothesis = "H1: There is an association between toss decision and match outcome.
 ↪"

# Test statistic formula
test_statistic_formula = "Chi-Square Statistic = Sum((Observed - Expected)^2 / Expected)"
```

```
# Build the report
report = f"""
Chi-Square Test for Independence

Null Hypothesis (H0):
{null_hypothesis}

Alternative Hypothesis (H1):
{alternative_hypothesis}

Test Statistic Formula:
{test_statistic_formula}

Results:
Chi-Square Statistic: {chi2_statistic:.4f}
Degrees of Freedom: {dof}
p-value: {p_value:.4f}

Interpretation:
{conclusion}

Contingency Table:
{contingency_table}

Expected Frequencies Table:
{pd.DataFrame(expected, index=['Bat', 'Field'], columns=['Wins', 'Losses'])}
"""

print(report)
```

[78]:
```
from scipy import stats

# Perform normality tests for each variable
normality_tests = {}
for column in ipl3.iloc[:, -16:]:
    ad_stat, ad_crit_values, ad_sig_levels = stats.anderson(ipl3[column], dist='norm')

    # Check if p-values are less than 0.05 (significance level)
    is_normal = ad_stat < ad_crit_values[2]
    normality_tests[column] = {
        "Anderson-Darling statistic": round(ad_stat, 4),
        "Anderson-Darling critical values": round(ad_crit_values[2], 4),
        "Is Normally Distributed": is_normal
    }

# Create DataFrame to display results
results_df = pd.DataFrame.from_dict(normality_tests, orient='index')

# Display results
print("Normality Test Results:")
df2=results_df.rename(columns={'index': 'Variables'})
df2
```

[79]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
file_name = "Normality test.csv"

# Save the DataFrame to a CSV file in the current directory
results_df.to_csv(os.path.join(current_directory, file_name), index=False)

# Now, the DataFrame 'merged_data2_sorted' is saved as a CSV file in the current working␣
 ↪directory
```

[80]:
```
from scipy.stats import wilcoxon
import pandas as pd

# Assuming ipl3 is your DataFrame containing the data
```

```
# Selecting total runs and wickets columns for 1st and 2nd innings
runs_1st_innings = ipl3['Total Runs 1st Innings']
runs_2nd_innings = ipl3['Total Runs 2nd Innings']
wickets_1st_innings = ipl3['Total Wickets 1st Innings']
wickets_2nd_innings = ipl3['Total Wickets 2nd Innings']

# Perform Wilcoxon signed-rank test for total runs
runs_statistic, runs_p_value = wilcoxon(runs_1st_innings, runs_2nd_innings)

# Perform Wilcoxon signed-rank test for total wickets
wickets_statistic, wickets_p_value = wilcoxon(wickets_1st_innings, wickets_2nd_innings)

# Print results
print("Wilcoxon Signed-Rank Test Results:")
print("Total Runs:")
print("Statistic:", runs_statistic)
print("P-value:", runs_p_value)
print("Significant difference" if runs_p_value < 0.05 else "No significant difference")

print("\nTotal Wickets:")
print("Statistic:", wickets_statistic)
print("P-value:", wickets_p_value)
print("Significant difference" if wickets_p_value < 0.05 else "No significant difference")

# Null and Alternative Hypotheses
null_hypothesis_runs = "H0: There is no difference in total runs scored between 1st and 2nd␣
 ↪innings."
alternative_hypothesis_runs = "H1: There is a difference in total runs scored between 1st and␣
 ↪2nd innings."
null_hypothesis_wickets = "H0: There is no difference in total wickets taken between 1st and␣
 ↪2nd innings."
alternative_hypothesis_wickets = "H1: There is a difference in total wickets taken between␣
 ↪1st and 2nd innings."

# Test statistic formula
test_statistic_formula_runs = "Wilcoxon Statistic = Sum of ranks of differences between␣
 ↪paired observations."
test_statistic_formula_wickets = "Wilcoxon Statistic = Sum of ranks of differences between␣
 ↪paired observations."

# Build the report
report = f"""
Wilcoxon Signed-Rank Test Results:

Total Runs:
Null Hypothesis (H0):
{null_hypothesis_runs}

Alternative Hypothesis (H1):
{alternative_hypothesis_runs}

Test Statistic Formula:
{test_statistic_formula_runs}

Results:
Statistic: {runs_statistic:.4f}
P-value: {runs_p_value:.4f}

Interpretation:
{"Reject null hypothesis: There is a significant difference in total runs scored between 1st␣
 ↪and 2nd innings." if runs_p_value < 0.05 else "Fail to reject null hypothesis: There is no␣
 ↪significant difference in total runs scored between 1st and 2nd innings."}

Total Wickets:
Null Hypothesis (H0):
{null_hypothesis_wickets}

Alternative Hypothesis (H1):
```

```
{alternative_hypothesis_wickets}

Test Statistic Formula:
{test_statistic_formula_wickets}

Results:
Statistic: {wickets_statistic:.4f}
P-value: {wickets_p_value:.4f}

Interpretation:
{"Reject null hypothesis: There is a significant difference in total wickets taken between␣
 ↪1st and 2nd innings." if wickets_p_value < 0.05 else "Fail to reject null hypothesis:␣
 ↪There is no significant difference in total wickets taken between 1st and 2nd innings."}
"""

print(report)
```

[83]:
```
from scipy.stats import levene

# Assuming 'ipl' is your DataFrame containing the data
# Selecting columns for runs and wickets for each phase (Power Play, Middle Overs, Death␣
 ↪Overs)
power_play_runs_1st = ipl3['Power Play Runs 1st Innings']
middle_overs_runs_1st = ipl3['Middle Overs Runs 1st Innings']
death_overs_runs_1st = ipl3['Death Overs Runs 1st Innings']
power_play_runs_2nd = ipl3['Power Play Runs 2nd Innings']
middle_overs_runs_2nd = ipl3['Middle Overs Runs 2nd Innings']
death_overs_runs_2nd = ipl3['Death Overs Runs 2nd Innings']
power_play_wickets_1st = ipl3['Power Play Wickets 1st Innings']
middle_overs_wickets_1st = ipl3['Middle Overs Wickets 1st Innings']
death_overs_wickets_1st = ipl3['Death Overs Wickets 1st Innings']
power_play_wickets_2nd = ipl3['Power Play Wickets 2nd Innings']
middle_overs_wickets_2nd = ipl3['Middle Overs Wickets 2nd Innings']
death_overs_wickets_2nd = ipl3['Death Overs Wickets 2nd Innings']

# Perform Levene's test for equal variances for runs
statistic_runs_1st, p_value_runs_1st = levene(power_play_runs_1st, middle_overs_runs_1st,␣
 ↪death_overs_runs_1st)
statistic_runs_2nd, p_value_runs_2nd = levene(power_play_runs_2nd, middle_overs_runs_2nd,␣
 ↪death_overs_runs_2nd)

# Perform Levene's test for equal variances for wickets
statistic_wickets_1st, p_value_wickets_1st = levene(power_play_wickets_1st,␣
 ↪middle_overs_wickets_1st, death_overs_wickets_1st)
statistic_wickets_2nd, p_value_wickets_2nd = levene(power_play_wickets_2nd,␣
 ↪middle_overs_wickets_2nd, death_overs_wickets_2nd)

# Create a DataFrame to store the results
results = pd.DataFrame({
    'Variable': ['Runs 1st Innings', 'Runs 2nd Innings', 'Wickets 1st Innings', 'Wickets 2nd␣
 ↪Innings'],
    'Statistic': [statistic_runs_1st, statistic_runs_2nd, statistic_wickets_1st,␣
 ↪statistic_wickets_2nd],
    'P-value': [p_value_runs_1st, p_value_runs_2nd, p_value_wickets_1st, p_value_wickets_2nd],
    'Equal Variances': ['Yes' if p_value_runs_1st > 0.05 else 'No',
                        'Yes' if p_value_runs_2nd > 0.05 else 'No',
                        'Yes' if p_value_wickets_1st > 0.05 else 'No',
                        'Yes' if p_value_wickets_2nd > 0.05 else 'No']
})

# Print the results table
print("Levene's Test Results:")
results
```

[84]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
```

```
file_name = "Eqaulity of variances.csv"

# Save the DataFrame to a CSV file in the current directory
results.to_csv(os.path.join(current_directory, file_name), index=False)

# Now, the DataFrame 'merged_data2_sorted' is saved as a CSV file in the current working␣
 ↪directory
```

[85]:
```
from scipy.stats import kruskal

# Selecting columns for runs and wickets for each phase (Power Play, Middle Overs, Death␣
 ↪Overs)
power_play_runs_1st = ipl3['Power Play Runs 1st Innings']
middle_overs_runs_1st = ipl3['Middle Overs Runs 1st Innings']
death_overs_runs_1st = ipl3['Death Overs Runs 1st Innings']
power_play_runs_2nd = ipl3['Power Play Runs 2nd Innings']
middle_overs_runs_2nd = ipl3['Middle Overs Runs 2nd Innings']
death_overs_runs_2nd = ipl3['Death Overs Runs 2nd Innings']
power_play_wickets_1st = ipl3['Power Play Wickets 1st Innings']
middle_overs_wickets_1st = ipl3['Middle Overs Wickets 1st Innings']
death_overs_wickets_1st = ipl3['Death Overs Wickets 1st Innings']
power_play_wickets_2nd = ipl3['Power Play Wickets 2nd Innings']
middle_overs_wickets_2nd = ipl3['Middle Overs Wickets 2nd Innings']
death_overs_wickets_2nd = ipl3['Death Overs Wickets 2nd Innings']

# Perform Kruskal-Wallis test for runs
runs_statistic_1st, runs_p_value_1st = kruskal(power_play_runs_1st, middle_overs_runs_1st,␣
 ↪death_overs_runs_1st)
runs_statistic_2nd, runs_p_value_2nd = kruskal(power_play_runs_2nd, middle_overs_runs_2nd,␣
 ↪death_overs_runs_2nd)

# Perform Kruskal-Wallis test for wickets
wickets_statistic_1st, wickets_p_value_1st = kruskal(power_play_wickets_1st,␣
 ↪middle_overs_wickets_1st, death_overs_wickets_1st)
wickets_statistic_2nd, wickets_p_value_2nd = kruskal(power_play_wickets_2nd,␣
 ↪middle_overs_wickets_2nd, death_overs_wickets_2nd)

# Create a DataFrame to store the results
results = pd.DataFrame({
    'Variable': ['Runs 1st Innings', 'Runs 2nd Innings', 'Wickets 1st Innings', 'Wickets 2nd␣
 ↪Innings'],
    'Statistic': [runs_statistic_1st, runs_statistic_2nd, wickets_statistic_1st,␣
 ↪wickets_statistic_2nd],
    'P-value': [runs_p_value_1st, runs_p_value_2nd, wickets_p_value_1st, wickets_p_value_2nd]
})

# Hypothesis Testing
significance_level = 0.05
for index, row in results.iterrows():
    variable = row['Variable']
    p_value = row['P-value']
    print(f"\nHypothesis Testing for {variable}:")
    if p_value < significance_level:
        print("Reject the null hypothesis. There is sufficient evidence to conclude that at␣
 ↪least one group has a different population median.")
    else:
        print("Fail to reject the null hypothesis. There is not enough evidence to conclude␣
 ↪that there is a difference in population medians among the groups.")

# Print the results table
print("Kruskal-Wallis Test Results:")
results
```

[86]:
```
# Get the current working directory
current_directory = os.getcwd()

# Define the file name
file_name = "Kruskal-Wallis Results.csv"
```

```
# Save the DataFrame to a CSV file in the current directory
results.to_csv(os.path.join(current_directory, file_name), index=False)

# Now, the DataFrame 'merged_data2_sorted' is saved as a CSV file in the current working␣
 ↪directory
```

Principal Component Analysis

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

[2]:
```
# Set option to display all columns
pd.set_option('display.max_columns', None)
ipl=pd.read_csv("players_performace_2008_2023.csv")
ipl.head()
```

[3]:
```
col = ['innings_bowl', 'Wickets_taken', 'Total_balls', 'Total_runs_given', 'Economy',␣
 ↪'Bowling_avg', 'Bowling_SR']

# Dropping the specified columns
ipl1=ipl.drop(columns=col)
ipl1.head(10)
```

[ ]:
```
# Select only numeric variables
numeric_ipl1 = ipl1.select_dtypes(include=[np.number])

# Compute the correlation matrix
correlation_matrix = numeric_ipl1.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

[36]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Assuming ipl1 is already loaded in your environment
# Dropping the 'Player' column for PCA
df_numeric = ipl1.drop(columns=['Player'])

# Standardize the data
df_standardized = (df_numeric - df_numeric.mean()) / df_numeric.std()

# Compute the covariance matrix
cov_matrix = np.cov(df_standardized.T)

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Sort the eigenvalues and eigenvectors
sorted_index = np.argsort(eigenvalues)[::-1]
sorted_eigenvalues = eigenvalues[sorted_index]
sorted_eigenvectors = eigenvectors[:, sorted_index]

# Compute the explained variance ratio
explained_variance_ratio = sorted_eigenvalues / np.sum(sorted_eigenvalues)

# Compute the cumulative explained variance ratio
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Compute the first principal component
```

```
pc1 = np.dot(df_standardized, sorted_eigenvectors[:, 0])

# Add the Player column to the principal component DataFrame
principal_components_df = pd.DataFrame({'Player': ipl1['Player'], 'PC1': pc1})

# Create a DataFrame for players, standardized values, and the first principal component
standardized_values_with_pc1_df = df_standardized.copy()
standardized_values_with_pc1_df['Player'] = ipl1['Player']
standardized_values_with_pc1_df['PC1'] = pc1

# Reorder columns to have Player first
standardized_values_with_pc1_df = standardized_values_with_pc1_df[['Player'] +␣
 ↪list(df_standardized.columns) + ['PC1']]

eigenvectors_df = pd.DataFrame(sorted_eigenvectors,
                               index=df_numeric.columns,
                               columns=[f'PC{i+1}' for i in range(len(df_numeric.columns))])

# Scree Plot
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, marker='o',␣
 ↪linestyle='--', label='Explained Variance')
#plt.plot(range(1, len(cumulative_explained_variance) + 1), cumulative_explained_variance,␣
 ↪marker='o', linestyle='-', label='Cumulative Explained Variance')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.xticks(range(1, len(explained_variance_ratio) + 1))
plt.legend()
plt.show()
```

[37]:
```
eigenvectors_df
```

[22]:
```
# Get the current working directory
current_directory = os.getcwd()
# Define the file name
file_name = "eigen vectors bat.csv"
# Save the DataFrame to a CSV file in the current directory
eigenvectors_df.to_csv(os.path.join(current_directory, file_name), index=False)
```

[19]:
```
merged_df = pd.merge(principal_components_df, standardized_values_with_pc1_df, on='Player')

# Print the merged DataFrame
merged_df_sorted = merged_df.sort_values(by='PC1_y', ascending=True)
merged_df_sorted
```

[20]:
```
# Get the current working directory
current_directory = os.getcwd()
# Define the file name
file_name = "new batsman ranking.csv"
# Save the DataFrame to a CSV file in the current directory
merged_df_sorted.to_csv(os.path.join(current_directory, file_name), index=False)
```

[13]:
```
# Create a DataFrame to merge the eigenvalues, explained variance ratio, and cumulative␣
 ↪explained variance ratio
pca_summary_df = pd.DataFrame({
    'Eigenvalue': sorted_eigenvalues,
    'Explained Variance Ratio': explained_variance_ratio,
    'Cumulative Explained Variance': cumulative_explained_variance
})

# Print the merged DataFrame
summary=pca_summary_df.T
summary
```

[14]:
```
# Get the current working directory
current_directory = os.getcwd()
# Define the file name
```

```
file_name = "pca summary.csv"
# Save the DataFrame to a CSV file in the current directory
summary.to_csv(os.path.join(current_directory, file_name), index=False)
```

[25]:
```
col2=['innings_bat', 'Runs', 'Balls_played', 'Outs', 'SR', 'Avg',
       'Centuries', 'Fifties', 'Fours', 'Sixes',]
ipl2=ipl.drop(columns=col2)
ipl2
```

[ ]:
```
# Select only numeric variables
numeric_ipl2 = ipl2.select_dtypes(include=[np.number])

# Compute the correlation matrix
correlation_matrix = numeric_ipl2.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

[32]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Assuming ipl1 is already loaded in your environment
# Dropping the 'Player' column for PCA
df_numeric1 = ipl2.drop(columns=['Player'])

# Standardize the data
df_standardized1 = (df_numeric1 - df_numeric1.mean()) / df_numeric1.std()

# Compute the covariance matrix
cov_matrix1 = np.cov(df_standardized1.T)

# Compute eigenvalues and eigenvectors
eigenvalues1, eigenvectors1 = np.linalg.eig(cov_matrix1)

# Sort the eigenvalues and eigenvectors
sorted_index1 = np.argsort(eigenvalues1)[::-1]
sorted_eigenvalues1 = eigenvalues1[sorted_index1]
sorted_eigenvectors1 = eigenvectors1[:, sorted_index1]

# Compute the explained variance ratio
explained_variance_ratio1 = sorted_eigenvalues1 / np.sum(sorted_eigenvalues1)

# Compute the cumulative explained variance ratio
cumulative_explained_variance1 = np.cumsum(explained_variance_ratio1)

# Compute the first principal component
pc_1 = np.dot(df_standardized1, sorted_eigenvectors1[:, 0])

# Add the Player column to the principal component DataFrame
principal_components_df1 = pd.DataFrame({'Player': ipl2['Player'], 'PC1': pc_1})

# Create a DataFrame for players, standardized values, and the first principal component
standardized_values_with_pc1_df1 = df_standardized1.copy()
standardized_values_with_pc1_df1['Player'] = ipl2['Player']
standardized_values_with_pc1_df1['PC1'] = pc_1

# Reorder columns to have Player first
standardized_values_with_pc1_df1 = standardized_values_with_pc1_df1[['Player'] +␣
 ↪list(df_standardized1.columns) + ['PC1']]

eigenvectors_df1 = pd.DataFrame(sorted_eigenvectors1,
                                index=df_numeric1.columns,
                                columns=[f'PC{i+1}' for i in range(len(df_numeric1.columns))])

# Scree Plot
```

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio1) + 1), explained_variance_ratio1, marker='o',␣
 ↪linestyle='--', label='Explained Variance')
#plt.plot(range(1, len(cumulative_explained_variance) + 1), cumulative_explained_variance,␣
 ↪marker='o', linestyle='-', label='Cumulative Explained Variance')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.xticks(range(1, len(explained_variance_ratio1) + 1))
plt.legend()
plt.show()
```

[33]:
```
eigenvectors_df1
```

[34]:
```
# Get the current working directory
current_directory = os.getcwd()
# Define the file name
file_name = "eigen vectors ball.csv"
# Save the DataFrame to a CSV file in the current directory
eigenvectors_df1.to_csv(os.path.join(current_directory, file_name), index=False)
```

[38]:
```
merged_df1 = pd.merge(principal_components_df1, standardized_values_with_pc1_df1, on='Player')

# Print the merged DataFrame
merged_df_sorted1 = merged_df1.sort_values(by='PC1_y', ascending=False)
merged_df_sorted1
```

[39]:
```
# Get the current working directory
current_directory = os.getcwd()
# Define the file name
file_name = "new bowlers ranking.csv"
# Save the DataFrame to a CSV file in the current directory
merged_df_sorted1.to_csv(os.path.join(current_directory, file_name), index=False)
```

[41]:
```
# Create a DataFrame to merge the eigenvalues, explained variance ratio, and cumulative␣
 ↪explained variance ratio
pca_summary_df1 = pd.DataFrame({
    'Eigenvalue': sorted_eigenvalues1,
    'Explained Variance Ratio': explained_variance_ratio1,
    'Cumulative Explained Variance': cumulative_explained_variance1
})

# Print the merged DataFrame
summary1=pca_summary_df1.T
summary1
```

[42]:
```
# Get the current working directory
current_directory = os.getcwd()
# Define the file name
file_name = "pca summary ball.csv"
# Save the DataFrame to a CSV file in the current directory
summary1.to_csv(os.path.join(current_directory, file_name), index=False)
```

Machine Learning

[ ]:
```
# import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
# Suppress specific warning
warnings.filterwarnings("ignore")
import os
```

[106]:
```
# Assuming df is already loaded with the necessary data
X = df[['City', 'Team1', 'Team2', 'Venue', 'TossWinner', 'TossDecision','Team1BatAvg',
       'Team2BatAvg', 'Team1BallAvg', 'Team2BallAvg']].copy()
```

```
y = df['WinningTeam']
```

[107]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

[108]:
```
# Encoding only categorical variables
label_encoders = {}
categorical_columns = ['City', 'Team1', 'Team2', 'Venue', 'TossWinner', 'TossDecision']

# Encode categorical features
for column in categorical_columns:
    le = LabelEncoder()
    X[column] = le.fit_transform(X[column])
    label_encoders[column] = le
```

[109]:
```
# Encoding target variable
le_y = LabelEncoder()
y = le_y.fit_transform(y)
```

[110]:
```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
```

[113]:
```
# Logistic Regression
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)

# K-Nearest Neighbors (KNN)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
```

[114]:
```
# Logistic Regression
log_reg_train_accuracy = accuracy_score(y_train, log_reg.predict(X_train))
log_reg_test_accuracy = accuracy_score(y_test, y_pred_log_reg)
log_reg_report = classification_report(y_test, y_pred_log_reg)

# K-Nearest Neighbors (KNN)
knn_train_accuracy = accuracy_score(y_train, knn.predict(X_train))
knn_test_accuracy = accuracy_score(y_test, y_pred_knn)
knn_report = classification_report(y_test, y_pred_knn)

# Random Forest
rf_train_accuracy = accuracy_score(y_train, rf.predict(X_train))
rf_test_accuracy = accuracy_score(y_test, y_pred_rf)
rf_report = classification_report(y_test, y_pred_rf)

# Decision Tree
dt_train_accuracy = accuracy_score(y_train, dt.predict(X_train))
dt_test_accuracy = accuracy_score(y_test, y_pred_dt)
dt_report = classification_report(y_test, y_pred_dt)

# Print overall performance metrics
```

```
print("Overall Performance Metrics:")
print("Logistic Regression - Training Accuracy:", log_reg_train_accuracy, "Test Accuracy:",␣
 ↪log_reg_test_accuracy)
print("K-Nearest Neighbors - Training Accuracy:", knn_train_accuracy, "Test Accuracy:",␣
 ↪knn_test_accuracy)
print("Random Forest - Training Accuracy:", rf_train_accuracy, "Test Accuracy:",␣
 ↪rf_test_accuracy)
print("Decision Tree - Training Accuracy:", dt_train_accuracy, "Test Accuracy:",␣
 ↪dt_test_accuracy)
print("\nClassification Reports:")
print("Logistic Regression:")
print(log_reg_report)
print("K-Nearest Neighbors:")
print(knn_report)
print("Random Forest:")
print(rf_report)
print("Decision Tree:")
print(dt_report)
```