CRICKET DATA ANALYSIS PROJECT USING WEB SCRAPING, PYTHON, PANDAS & POWER-BI

A PROJECT REPORT

Submitted by

MONALIKA GHOSH

[EC2332251010130]

Under the Guidance of

Dr. Umamaheswari. Km

(Assistant Professor of Online Education)

in partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

DIRECTORATE OF ONLINE EDUCATION
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR- 603 203
DECEMBER 2024

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

DIRECTORATE OF ONLINE EDUCATION

SRM INSTITUTE OF SCIENCE AND TECHNOLOGYKATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

This Project Work Report titled **"Cricket Data Analysis Project Using Web Scraping, Python, Pandas, and Power BI"** of **"MONALIKA GHOSH [EC2332251010130]"**, who carried out the Project Work under my supervision along with the company mentor. Certified further, that to the best of my knowledge the work reported herein does not form any other internship report or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate

SRM
UNIVERSITY
(Under section 3 of UGC Act 1956)

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to Director DOE, SRM Institute of Science and Technology, Prof. **Dr Manoranjan Pon Ram,** for his invaluable support. SRM Institute of Science and Technology, for her support throughout the project work. We want to convey our thanks to Programme Coordinator **Dr. G. Babu**, Directorate of online Education, SRM Institute of Science and Technology, for their inputs during the project reviews and support. Our inexpressible respect and thanks to my guide, **Dr. Umamaheswari. Km.**, Assistant Professor of online Education, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under her mentorship. She provided me with the freedom and support to explore the research topics of my interest. Her passion for solving problems and making a difference in the world has always been inspiring. We sincerely thank the Directorate of online Education, staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

MONALIKA GHOSH

SRM UNIVERSITY (Under section 3 of UGC Act 1956)

CONTENTS

1.ABSTRACT

In recent years, the availability of sports data has dramatically increased, providing new opportunities for in-depth analysis and insights. The Indian Premier League (IPL), which debuted in 2008 and is currently the most well-known T20 league worldwide, is played in India. Cricket is a tremendously fluid sport that evolves with each ball. People enjoy and watch cricket the most, and the Twenty20 format in particular since no one can predict the winner until the final ball of the final over. India’s most popular and frequently played sport, with the largest fan following, is cricket.This research project, titled "Cricket Data Analysis Project Using Web Scraping, Python, Pandas, and Power BI," aims to harness the potential of data science and visualization tools to explore and analyze cricket data. The primary objective of this project is to develop a comprehensive framework for extracting, processing, analyzing, and visualizing cricket data from various online sources.

The first phase of the project involves web scraping, where relevant cricket data, including match statistics, player performance metrics, and historical records, are collected from reliable online platforms. Python, a versatile programming language, is employed for web scraping, leveraging libraries such as BeautifulSoup and Selenium to automate data extraction.

Once the data is collected, it undergoes cleaning, transformation, and analysis using Pandas, a powerful Python library for data manipulation. This phase includes handling missing data, normalizing data formats, and conducting exploratory data analysis (EDA) to uncover patterns, trends, and correlations within the cricket datasets.

The final phase of the project focuses on data visualization and reporting using Power BI, a robust business intelligence tool. Through interactive dashboards and visual reports, key insights derived from the cricket data are presented, allowing for a deeper understanding of player performances, match outcomes, and other critical aspects of the game. These visualizations are designed to be intuitive, enabling users to explore the data and make data-driven decisions.

This research project not only demonstrates the application of web scraping, Python, Pandas, and Power BI in sports analytics but also provides a scalable framework that can be adapted to other sports or domains. The 20 overs formula used in the Indian Premier League is quite unexpected. Therefore, we decided to do an exploratory data analysis to research and evaluate the IPL games. The findings of the analytics applied reveal several insights from the IPL dataset that can be used to better optimize both team and individual performance.

The findings from this project could benefit teams, coaches, analysts, and fans by offering a more nuanced view of cricket data.

Keywords: Cricket data analysis, IPL, Exploratory Data Analysis, Data Visualization, web scraping, Python, Pandas, Power BI, sports analytics, player performance, match statistics.

2.INTRODUCTION

Cricket, a sport with a rich history and massive global following, generates an immense amount of data across various formats, leagues, and international tournaments. From individual player performances to team strategies and match outcomes, cricket data holds significant potential for analysis, providing valuable insights that can influence decision-making, performance improvement, and fan engagement. However, this data often resides across multiple online platforms and in unstructured formats, making it challenging to collect, process, and analyze efficiently. This research project, titled *"Cricket Data Analysis Project Using Web Scraping, Python, Pandas, and Power BI,"* seeks to address this challenge by building a comprehensive framework for cricket data extraction, analysis, and visualization.

The project is anchored in the field of data science, where the ability to gather, process, and analyze large datasets is paramount. The rise of web scraping as a technique allows us to gather large amounts of data directly from websites, bypassing the limitations of pre-structured datasets. In the context of cricket, web scraping facilitates the extraction of real-time and historical match statistics, player metrics, and other valuable information from platforms like ESPN Cricinfo, Cricbuzz, and other cricket-related databases. These sources typically do not provide an easy way to download raw data in a usable format, making web scraping an essential component of this project.

Python, one of the most widely used programming languages in the data science domain, plays a central role in the implementation of this project. Libraries such as BeautifulSoup and Selenium are utilized for web scraping, while Pandas serves as the primary tool for data cleaning, transformation, and analysis. Python's flexibility and powerful libraries enable the development of a dynamic workflow that handles everything from raw data extraction to complex data manipulations. Pandas, in particular, is crucial for transforming messy, unstructured data into organized formats that can be analyzed for meaningful insights.

The project moves beyond raw data analysis by incorporating Power BI, a leading business intelligence tool, to create rich, interactive data visualizations. Visualization is critical in conveying insights effectively, allowing stakeholders such as coaches, analysts, and cricket enthusiasts to interpret the data in a more intuitive and accessible manner. By integrating Power BI, this project aims to produce dynamic reports and dashboards that users can interact with, enabling them to explore key cricket statistics, trends, and performance metrics with ease.

By leveraging the combination of web scraping, Python, Pandas, and Power BI, this project intends to provide a structured and scalable approach to cricket data analysis. The insights generated from this analysis can be used to evaluate player performance, assess team strategies, and predict match outcomes. Moreover, this framework can be expanded and adapted to other domains, making it highly versatile and valuable across multiple sports and industries.

In summary, the "Cricket Data Analysis Project Using Web Scraping, Python, Pandas, and Power BI" addresses the increasing demand for data-driven insights in cricket. By building an automated data pipeline that integrates web scraping, advanced data manipulation, and dynamic visualization, this project demonstrates the power of modern data science techniques in sports analytics.

2.1INTRODUCTION TO DATA ANALYSIS

Data analysis is an essential aspect of modern decision-making processes across various sectors, including business, healthcare, finance, and academia. As organizations generate massive amounts of data daily, understanding how to extract meaningful insights from this data becomes crucial.

What Do You Mean by Data Analysis?

In today's data-driven world, organizations rely on data analysis to uncover patterns, trends, and relationships within their data. Whether it's for optimizing operations, improving customer satisfaction, or forecasting future trends, effective data analysis helps stakeholders make informed decisions. The term data analysis refers to the systematic application of statistical and logical techniques to describe, summarize, and evaluate data. This process can involve transforming raw data into a more understandable format, identifying significant patterns, and drawing conclusions based on the findings.

When we ask, "What do you mean by data analysis?" it essentially refers to the practice of examining datasets to draw conclusions about the information they contain. The process can be broken down into several steps, including:

1. **Data Collection**: Gathering relevant data from various sources, which could be databases, surveys, sensors, or web scraping.
2. **Data Cleaning**: Identifying and correcting inaccuracies or inconsistencies in the data to ensure its quality and reliability.
3. **Data Transformation**: Modifying data into a suitable format for analysis, which may involve normalization, aggregation, or creating new variables.
4. **Data Analysis**: Applying statistical methods and algorithms to explore the data, identify trends, and extract meaningful insights.
5. **Data Interpretation**: Translating the findings into actionable recommendations or conclusions that inform decision-making.

By employing these steps, organizations can transform raw data into a valuable asset that guides strategic planning and enhances operational efficiency.

To solidify our understanding, let's define data analysis with an example. Imagine a retail company looking to improve its sales performance. The company collects data on customer purchases, demographics, and seasonal trends.

Data Analysis in Data Science

The field of data science relies heavily on data analysis to derive insights from large datasets. Data analysis in data science refers to the methods and processes used to manipulate data, identify trends, and generate predictive models that aid in decision-making.

Data scientists employ various analytical techniques, such as:

- **Statistical Analysis**: Applying statistical tests to validate hypotheses or understand relationships between variables.
- **Machine Learning**: Using algorithms to enable systems to learn from data patterns and make predictions.
- **Data Visualization**: Creating graphical representations of data to facilitate understanding and communication of insights.

These techniques play a vital role in enabling organizations to leverage their data effectively, ensuring they remain competitive and responsive to market changes.

Why Data Analysis is important?

Data analysis is crucial for informed decision-making, revealing patterns, trends, and insights within datasets. It enhances strategic planning, identifies opportunities and challenges, improves efficiency, and fosters a deeper understanding of complex phenomena across various industries and fields.

1. **Informed Decision-Making: A**nalysis of data provides a basis for informed decision-making by offering insights into past performance, current trends, and potential future outcomes.
2. **Business Intelligence:** Analyzed data helps organizations gain a competitive edge by identifying market trends, customer preferences, and areas for improvement.
3. **Problem Solving:** It aids in identifying and solving problems within a system or process by revealing patterns or anomalies that require attention.
4. **Performance Evaluation:** Analysis of data enables the assessment of performance metrics, allowing organizations to measure success, identify areas for improvement, and set realistic goals.
5. **Risk Management:** Understanding patterns in data helps in predicting and managing risks, allowing organizations to mitigate potential challenges.
6. **Optimizing Processes:** Data analysis identifies inefficiencies in processes, allowing for optimization and cost reduction.

The Process of Data Analysis

A Data analysis has the ability to transform raw available data into meaningful insights for your business and your decision-making. While there are several different ways of collecting and interpreting this data, most data-analysis processes follow the same six general steps.

1. **Define Objectives and Questions:** Clearly define the goals of the analysis and the specific questions you aim to answer. Establish a clear understanding of what insights or decisions the analyzed data should inform.
2. **Data Collection:** Gather relevant data from various sources. Ensure data integrity, quality, and completeness. Organize the data in a format suitable for analysis. There are two types of data: qualititative and quantitative data.
3. **Data Cleaning and Preprocessing:** Address missing values, handle outliers, and transform the data into a usable format. Cleaning and preprocessing steps are crucial for ensuring the accuracy and reliability of the analysis.
4. **Exploratory Data Analysis (EDA):** Conduct exploratory analysis to understand the characteristics of the data. Visualize distributions, identify patterns, and calculate summary statistics. EDA helps in formulating hypotheses and refining the analysis approach.
5. **Statistical Analysis or Modeling:** Apply appropriate statistical methods or modeling techniques to answer the defined questions. This step involves testing hypotheses, building predictive models, or performing any analysis required to derive meaningful insights from the data.
6. **Interpretation and Communication:** Interpret the results in the context of the original objectives. Communicate findings through reports, visualizations, or presentations. Clearly articulate insights, conclusions, and recommendations based on the analysis to support informed decision-making.

Analyzing Data: Techniques and Methods

When discussing analyzing data, several methods can be employed depending on the nature of the data and the questions being addressed. These methods can be broadly categorized into three types:

There are various data analysis methods, each tailored to specific goals and types of data. The major Data Analysis methods are:

1. Descriptive Analysis
A Descriptive Analysis is foundational as it provides the necessary insights into past performance. Understanding what has happened is crucial for making informed decisions in data analysis. For instance, data analysis in data science often begins with descriptive techniques to summarize and visualize data trends.

2. Diagnostic Analysis
Diagnostic analysis works hand in hand with Descriptive Analysis. As descriptive Analysis finds out what happened in the past, diagnostic Analysis, on the other hand, finds out why did that happen or what measures were taken at that time, or how frequently it has happened. By analyzing data thoroughly, businesses can address the question, “what do you mean by data analysis?” They can assess what factors contributed to specific outcomes, providing a clearer picture of their operational efficiency and effectiveness.

3. Predictive Analysis
By forecasting future trends based on historical data, Predictive analysis predictive analysis enables organizations to prepare for upcoming opportunities and challenges. This analysis type answers the inquiry of what is data science analysis by leveraging data trends to predict future behaviors and trends. This capability is vital for strategic planning and risk management in business operations.

4. Prescriptive Analysis
Prescriptive Analysis is an advanced method that takes Predictive Analysis insights and offers actionable recommendations, guiding decision-makers toward the best course of action. It extends beyond merely analyzing data to suggesting optimal solutions based on potential future scenarios, thus addressing the need for a structured approach to decision-making.

5. Statistical Analysis
Statistical Analysis is essential for summarizing data, helping in identifying key characteristics and understanding relationships within datasets. This analysis can reveal significant patterns that inform broader strategies and policies, thereby allowing analysts to provide a robust review of data analytics practices within an organization.

6. Regression Analysis
Regression analysis is a statistical method extensively used in data analysis to model the relationship between a dependent variable and one or more independent variables. This method is particularly useful in establishing the relationship between variables, making it vital for forecasting and strategic planning, as analysts often define data analysis with examples that utilize regression techniques to illustrate these concepts.

7. Cohort Analysis
By examining specific groups over time, cohort analysis aids in understanding customer behavior and improving retention strategies. This approach allows businesses to tailor their services to different segments, thereby effectively utilizing data storage and analysis in big data to enhance customer engagement and satisfaction.

8. Time Series Analysis
Time series analysis is crucial for any domain where data points are collected over time, allowing for trend identification and forecasting. Businesses can utilize this method to analyze seasonal trends and predict future sales, addressing the question of what do you understand by

data analysis in the context of temporal data

.

9. Factor Analysis

Factor analysis is a statistical method that explores underlying relationships among a set of observed variables. It identifies latent factors that contribute to observed patterns, simplifying complex data structures. This technique is invaluable in reducing dimensionality, revealing hidden patterns, and aiding in the interpretation of large datasets.

10. Text Analysis

Text analysis involves extracting valuable information from unstructured textual data. Utilizing natural language processing and machine learning techniques, it enables the extraction of sentiments, key themes, and patterns within large volumes of text. analyze customer feedback, social media sentiment, and more, showcasing the practical applications of analyzing data in real-world scenarios.

Tools for Data Analysis

Several tools are available to facilitate effective data analysis. These tools can range from simple spreadsheet applications to complex statistical software. Some popular tools include:

- **SAS** :SAS was a programming language developed by the SAS Institute for performed advanced analytics, multivariate analyses, business intelligence, data management, and predictive analytics. , SAS was developed for very specific uses and powerful tools are not added every day to the extensive already existing collection thus making it less scalable for certain applications.
- **Microsoft Excel** :It is an important spreadsheet application that can be useful for recording expenses, charting data, and performing easy manipulation and lookup and or generating pivot tables to provide the desired summarized reports of large datasets that contain significant data findings. It is written in C#, C++, and .NET Framework, and its stable version was released in 2016.
- **R** :It is one of the leading programming languages for performing complex statistical computations and graphics. It is a free and open-source language that can be run on various *UNIX platforms, Windows, and macOS*. It also has a command-line interface that is easy to use. However, it is tough to learn especially for people who do not have prior knowledge about programming.
- **Python**: It is a powerful high-level programming language that is used for general-purpose programming. Python supports both structured and functional programming methods. Its extensive collection of libraries make it very useful in data analysis. Knowledge of Tensorflow, Theano, Keras, Matplotlib, Scikit-learn, and Keras can get you a lot closer to your dream of becoming a machine learning engineer.
- **Tableau Public**: Tableau Public is free software developed by the public company “Tableau Software” that allows users to connect to any spreadsheet or file and create interactive data visualizations. It can also be used to create maps, dashboards along with real-time updation for easy presentation on the web. The results can be shared through social media sites or directly with the client making it very convenient to use.
- **Knime** :Knime, the Konstanz Information Miner is a free and open-source data analytics software. It is also used as a reporting and integration platform. It involves the integration of various components for Machine Learning and data mining through the modular data-pipe lining. It is written in Java and developed by KNIME.com AG. It can be operated in various operating systems such as Linux, OS X, and Windows.
- **Power BI**: A business analytics service that provides interactive visualizations and

business intelligence capabilities with a simple interface.

In conclusion, data analysis is a vital process that involves examining, cleaning, transforming, and modeling data to extract meaningful insights that drive decision-making. With the vast amounts of data generated daily, organizations must harness the power of data analysis to remain competitive and responsive to market trends.
Understanding the different types of data analysis, the tools available, and the methods employed in this field is essential for professionals aiming to leverage data effectively. As we move further into the digital age, the significance of data analysis will continue to grow, shaping the future of industries and influencing strategic decisions across the globe.

3.PROBLEM STATEMENT

The world of cricket produces a vast amount of data in the form of match statistics, player performance metrics, and historical records. This data is often scattered across various websites and stored in formats that are not immediately accessible for analysis. For coaches, analysts, teams, and enthusiasts, gaining valuable insights from this data can be a challenge, as it requires efficient methods to collect, process, and visualize the information.
The primary problem this project addresses is the lack of a streamlined and automated system to gather and analyze cricket data. Manually collecting and analyzing data is time-consuming, error-prone, and inefficient, especially when dealing with large datasets. The solution lies in creating a data pipeline that automates the extraction of data, processes it for analysis, and generates visual reports that offer actionable insights into cricket performance.
This project will use web scraping to collect cricket data from online platforms, Python and Pandas to clean and analyze it, and Power BI to create interactive visualizations. This comprehensive system will make cricket data more accessible and allow for in-depth analysis that can benefit teams, analysts, and fans alike.

The *Cricket Data Analysis Project* focuses on analyzing player data from the IPL 2022 auction, including both newly purchased and retained players across various franchises. This data includes comprehensive details on each player’s performance metrics, such as matches played, runs scored, wickets taken, batting and bowling averages, strike rates, catches, runouts, stumpings, and other relevant statistics. By leveraging these data points, the project aims to identify and create an optimal playing XI from the available pool of players, using insights from player performance data to guide selection.
The goal is to construct a team that mirrors successful strategies, such as those observed in the ICC World Cup winning squads and previous IPL champions, focusing on the ideal mix of batters, bowlers, and all-rounders. This approach provides an analytical foundation for building an effective team based on past data-driven squad formations.

3.1 DESIGN

The design of the *Cricket Data Analysis Project* integrates multiple technologies and follows a modular, step-by-step approach to ensure efficient data collection, processing, analysis, and visualization. This section outlines the system’s architecture and design principles, focusing on the structure of each module and how they interact to create a cohesive data analysis pipeline.

1. System Architecture

The system is designed as a multi-module architecture with clearly defined roles for each component. It includes the following key stages:

- **Data Collection Module (Web Scraping):** The first step is to automate the extraction of cricket data from online sources. Python scripts, using libraries such as BeautifulSoup and Selenium, scrape data from websites like ESPN Cricinfo and Cricbuzz. These scripts are designed to run periodically or on-demand, ensuring the latest data is always available. The extracted data is stored in structured formats such as CSV or JSON.

- **Data Processing Module (Python & Pandas):** After data collection, the raw data is cleaned and structured using Pandas. This module handles common data issues, such as missing or inconsistent values, and transforms the scraped data into a format that is ready for analysis. It also prepares datasets for visualizations by ensuring all data is standardized.

- **Exploratory Data Analysis (EDA) Module:** In this stage, Python's data visualization libraries like Matplotlib and Seaborn are used for preliminary analysis. These tools help identify trends, patterns, and outliers within the data, which can inform more detailed analysis in the subsequent steps.

- **Data Visualization Module (Power BI):** The cleaned and analyzed data is imported into Power BI for the creation of interactive dashboards. Power BI allows users to filter and explore the data visually, offering insights into various cricket statistics, including player performance and team trends.

2. Data Flow

The project's design follows a logical flow of data from the collection phase to the visualization stage. The key design principles are:

- **Data Pipeline:** The project is designed as a data pipeline where data flows from one module to the next. Each module has a specific function and processes data before passing it to the subsequent stage. This design ensures modularity, allowing each phase to be independently developed and optimized.

- **Automation:** To ensure the system is efficient and up-to-date, web scraping is automated through scheduled scripts. The automation process is designed to handle updates in cricket statistics and player performance after each match or tournament, ensuring real-time data analysis.

- **Error Handling:** The design incorporates robust error-handling mechanisms in the web scraping phase to manage potential issues such as missing data, website structure changes, or CAPTCHA blocks. This helps maintain the integrity of the collected data.

3. User Interface (Power BI Dashboards)

The final component of the design is the user interface, created using Power BI. The goal of this interface is to provide stakeholders (such as analysts, coaches, and fans) with easy access to detailed cricket insights through interactive dashboards.

- **Dashboard Design:** The dashboards are designed to be user-friendly, allowing users to explore data through various filters and drill-down options. Users can select specific players, teams, or match dates to focus on particular aspects of performance or compare trends over time.

- **Interactivity:** Power BI's interactivity is a critical design feature, enabling users to engage with the data without needing technical expertise. Dynamic charts, graphs, and tables provide an intuitive way to explore cricket statistics.

4. Scalability and Flexibility

The system is designed to be scalable. It can handle increasing amounts of data as the project grows to include more cricket matches, tournaments, and player data. The modular design ensures that additional data sources or analysis features can be added without disrupting the existing system.

- **Cloud Integration (Optional):** For larger datasets or real-time updates, the design can be expanded to include cloud-based storage and processing, ensuring high performance even as data volume increases.

The design of the *Cricket Data Analysis Project* follows a structured, modular approach that streamlines the entire process from data collection to visualization. It ensures real-time data updates, robust error handling, and user-friendly dashboards, making it a powerful tool for cricket analysts and enthusiasts. The scalability and flexibility of the design also allow the system to grow and adapt over time, handling more data and more complex analysis as needed.

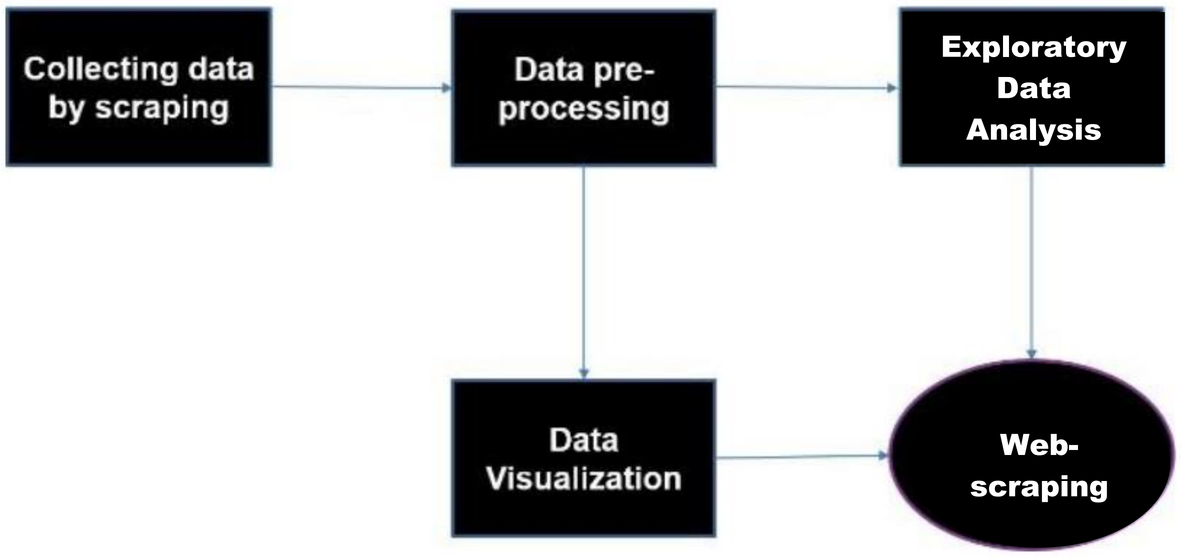


Fig1: Architecture Diagram

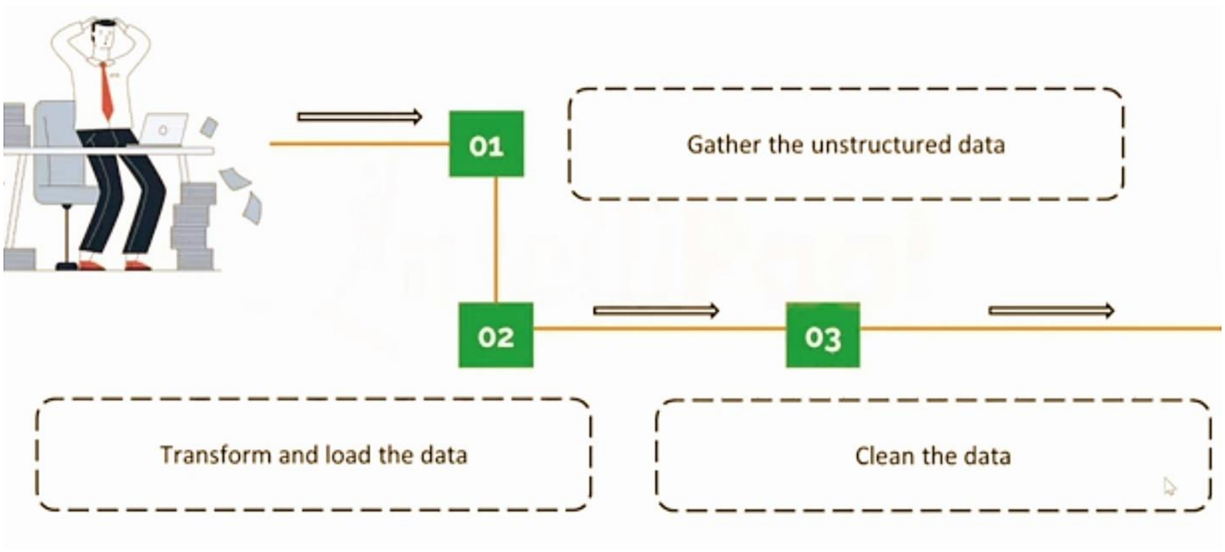


Fig2: Data Flow Diagram

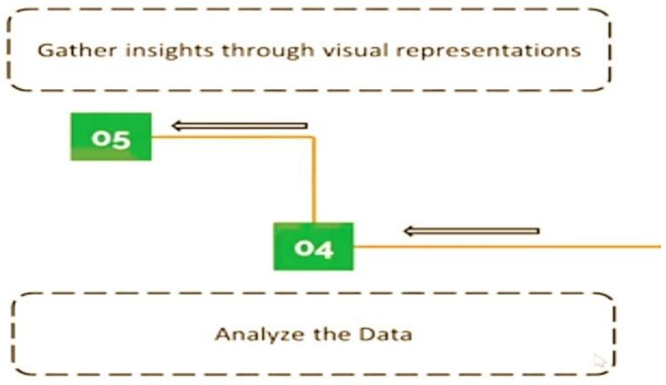


Fig2.1: Data Flow Diagram

4.MODULES DESCRIPTION

The *Cricket Data Analysis Project* is divided into four key modules: Data Collection (Web Scraping), Data Processing (Python and Pandas), Exploratory Data Analysis (EDA), and Data Visualization (Power BI). Each module addresses a specific part of the problem and contributes to the overall objective of developing a complete cricket data analysis pipeline.

1. Data Collection Module (Web Scraping)

The data collection module is responsible for gathering cricket-related data from various online sources. Websites like ESPN Cricinfo and Cricbuzz provide a wealth of information on player statistics, match outcomes, and team performances. However, this data is often displayed as web content, making it difficult to download directly.
In this module, Python libraries such as BeautifulSoup and Selenium will be used to extract the necessary data from HTML pages. BeautifulSoup will parse static content, while Selenium will be used to handle dynamic content that loads with JavaScript. The scraped data will include match statistics (runs, wickets, scores), player profiles, and historical match records.
Key Functions:

- Automating data extraction from cricket websites.
- Handling both static and dynamic content.
- Storing the extracted data in a structured format (CSV, JSON).

2. Data Processing Module (Python & Pandas)

Once the raw data is collected, it must be cleaned and organized before analysis. This module focuses on transforming the unstructured, often messy, scraped data into a structured format suitable for analysis. Using Pandas, a Python library designed for data manipulation, this module will handle tasks like removing duplicate entries, handling missing values, and standardizing data formats.
This module will also prepare the data for further analysis by filtering relevant statistics, normalizing player names and match details, and ensuring data consistency across different datasets. Pandas' powerful DataFrame structure will allow the data to be stored and manipulated efficiently.
Key Functions:

- Data cleaning (handling missing values, removing duplicates).
- Standardizing and organizing the data into DataFrames.

- Preparing the data for exploratory and statistical analysis.

3. Exploratory Data Analysis (EDA) Module

Exploratory Data Analysis (EDA) is essential for understanding the structure and trends within the data. This module will focus on identifying key insights, such as player performance over time, team statistics, and patterns in match outcomes. Various statistical methods and visualizations will be employed to explore the data.

Python libraries like Matplotlib and Seaborn will be used for preliminary visualization, helping identify correlations, outliers, and trends in the dataset. This module serves as the foundation for more detailed analysis and will guide the design of the Power BI dashboards.

Key Functions:

- Summarizing the data through descriptive statistics.
- Identifying trends and patterns (e.g., performance over time).
- Visualizing data using Python's Matplotlib or Seaborn libraries.

4. Data Visualization Module (Power BI)

The final module focuses on transforming the processed data into interactive dashboards and reports using Power BI. Visualization is crucial for conveying insights effectively, especially to non-technical users such as coaches or cricket fans. Power BI allows for real-time, interactive exploration of the data through intuitive dashboards.

The visualizations will display various cricket statistics, such as player performance charts, team comparisons, and win-loss ratios. Filters and interactive elements will enable users to dive deeper into specific aspects of the data, such as focusing on a particular player, team, or time period.

Key Functions:

- Importing the processed data into Power BI for visualization.
- Creating dynamic, interactive dashboards.
- Allowing user interactivity through filters and drill-down options.

5. Selection of the Best Playing XI

This module synthesizes insights from previous analyses to formulate an ideal cricket team of 11 players. Drawing from historical data of ICC World Cup-winning formations and successful IPL teams, this module selects players based on performance metrics and team balance, including a mix of batters, bowlers, and all-rounders. The goal is to optimize team composition for a balanced squad that can perform well across various game situations, enhancing the team's overall potential for success.

Key Functions:

- **Performance Evaluation:** Assess individual players' statistics to rank batters, bowlers, and all-rounders.
- **Role-Based Selection:** Choose players based on specific roles and contributions, ensuring a balanced team.

- **Historical Benchmarking:** Reference successful team structures from past tournaments to guide selection.
- **Squad Optimization:** Formulate a playing XI with a strategic blend of skills and experience, maximizing chances of winning.

Thus, the *Cricket Data Analysis Project* aims to solve the challenge of extracting and analyzing cricket data by dividing the task into four interconnected modules: Data Collection, Data Processing, Exploratory Data Analysis, and Data Visualization. Each module addresses a specific challenge, from gathering unstructured data to transforming it into meaningful insights through powerful visualizations.

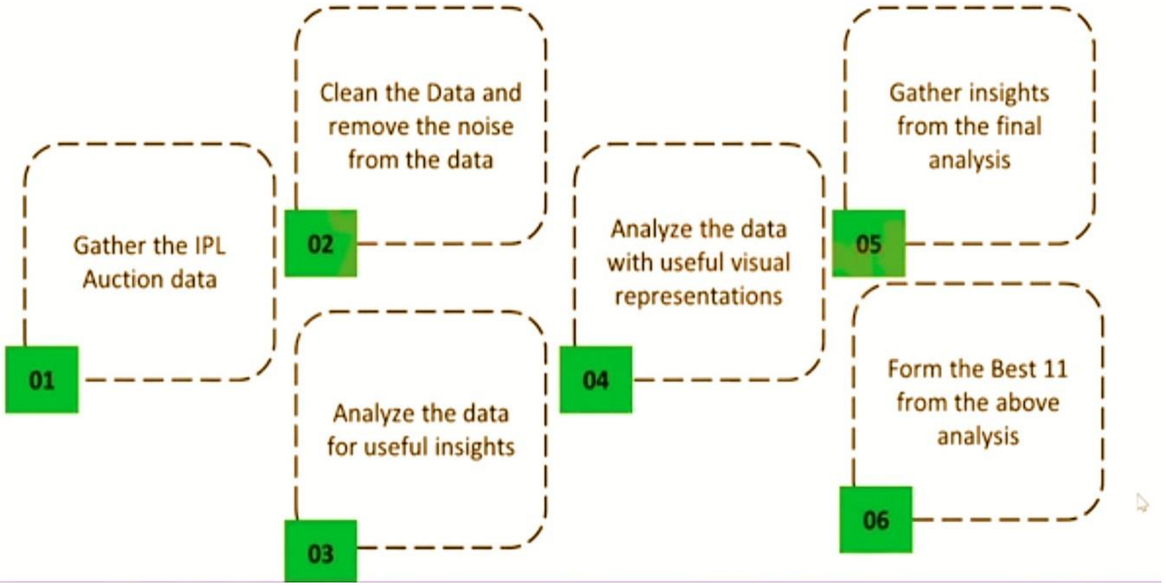

Gather the IPL Auction data
01
Clean the Data and remove the noise from the data
02
Analyze the data for useful insights
03
Analyze the data with useful visual representations
04
Gather insights from the final analysis
05
Form the Best 11 from the above analysis
06

Fig 3:Steps for IPL Data Analysis

5.ANALYSIS AND REQUIREMENT:

The *Cricket Data Analysis Project Using Web Scraping, Python, Pandas, and Power BI* integrates data collection, processing, analysis, and visualization to extract meaningful insights from cricket data. This section outlines the core project analysis and necessary requirements for its successful implementation.

1. Project Overview

The project aims to create a data pipeline that automates cricket data collection, processes it for analysis, and visualizes insights. This will enable analysts, coaches, and fans to derive actionable insights through streamlined data extraction, transformation, and presentation.

2. Data Collection (Web Scraping)

Analysis:

Cricket data, such as match statistics and player performance, is available online but not in easy-to-download formats. Web scraping will automate data extraction from sites like ESPN Cricinfo.

Requirements:

- **Tools:** Python libraries (BeautifulSoup, Selenium) for parsing and automating data extraction.
- **Data Sources:** ESPN Cricinfo, Cricbuzz.

- **Infrastructure:** Python-enabled systems, stable internet connection.
- **Error Handling:** Mechanisms for issues like missing data or CAPTCHA.
- **Compliance:** Adherence to website terms of service.

3. Data Processing (Python & Pandas)
Analysis:
Scraped data is often unstructured and requires cleaning and transformation. Using Pandas, this data will be organized and standardized for analysis.
Requirements:

- **Data Cleaning:** Remove duplicates, handle missing data, and standardize formats.
- **Data Transformation:** Convert raw data into structured formats like DataFrames.
- **Handling Large Data:** Efficient management of historical data.
- **Dependencies:** Python libraries (Pandas, NumPy, Matplotlib).

4. Exploratory Data Analysis (EDA)
Analysis:
EDA helps uncover insights such as performance trends and anomalies in cricket data, using statistical methods and visualizations.
Requirements:

- **Data Summarization:** Tools for calculating basic statistics (mean, median, etc.).
- **Trend Analysis:** Tools for observing performance trends over time.
- **Tools for Visualization:** Matplotlib, Seaborn for initial graphs.

5. Data Visualization (Power BI)
Analysis:
Power BI will visualize analyzed cricket data, providing interactive dashboards for exploring performance, match outcomes, and more.
Requirements:

- **Data Integration:** Import processed data into Power BI using CSV or database connectors.
- **Dashboard Design:** Create intuitive, user-friendly dashboards.
- **User Interactivity:** Enable filtering and exploration of data based on specific parameters.
- **Reporting:** Generate automated reports summarizing insights.

6.TOOLS AND TECHNOLOGIES:

The *Cricket Data Analysis Project* requires a mix of hardware and software to ensure smooth execution, scalability, and efficiency across data collection, processing, and visualization tasks. Below are the recommended specifications.

1. Hardware Specifications
Basic Setup (Small to Medium Datasets)

- **Processor:** Intel Core i5 or AMD (4 cores), supporting parallel tasks for web scraping, processing, and visualization.

- **RAM:** 8 GB DDR4 for handling medium datasets.
- **Storage:** 256 GB SSD to ensure fast data access.
- **Graphics Card:** Integrated (dedicated GPU optional for intensive Power BI visualizations).
- **Internet:** Stable, high-speed connection for web scraping.

2. Software Specifications
Operating System
- **Windows 10/11, macOS, or Linux:** Suitable for Python and Power BI; Linux is ideal for cloud-based, headless scraping.

Development Environment
- **Python 3.x:** Primary language for scraping, processing, and analysis.
- **Jupyter Notebooks:** For interactive analysis, testing, and debugging.
- **Power BI Desktop:** Essential for building and testing visualizations.

Python Libraries.
- **Pandas & NumPy:** For data manipulation, transformation, and advanced calculations.
- **Matplotlib & Seaborn:** For preliminary visualizations and EDA.

Data Storage
- **Formats:** CSV and JSON for storing raw/processed data.
- **SQL Database (Optional):** MySQL or PostgreSQL for efficient storage and querying of large datasets, easily connected to Power BI.

Power BI Tools
- **Power BI Desktop:** Core tool for designing dashboards and reports.
- **Power BI Service (Cloud):** For sharing reports and real-time data visualization.

Version Control & Collaboration
- **Git/GitHub/GitLab:** For version control and team collaboration.

6.1 INTRODUCTION TO PYTHON

Python provides incredible flexibility in implementing a machine learning model and working on data preprocessing and exploratory data analysis tasks, making it the most preferred language for loan eligibility prediction projects.

Python Libraries used for IPL Data analysis :

Since we are working on a fixed dataset to compare the performance of multiple algorithms and get started with a loan prediction project, we can use some popular libraries commonly used in Python.

Python Libraries for Data Manipulation and Analysis

1. NumPy
NumPy is a free Python software library for numerical computing on data that can be in the form of large arrays and multi-dimensional matrices. These multidimensional matrices are the main objects in NumPy where their dimensions are called axes and the number of axes is called a rank.
Key Features:
- N-dimensional array objects
- Broadcasting functions

- Linear algebra, Fourier transforms, and random number capabilities

2. Pandas

Pandas is one of the best libraries for Python, which is a free software library for data analysis and data handling. In short, Pandas is perfect for quick and easy data manipulation, data aggregation, reading, and writing the data and data visualization.

Key Features:

- DataFrame manipulation
- Grouping, joining, and merging datasets
- Time series data handling
- Data cleaning and wrangling

Python Libaries for Data Visualization

3. Matplotlib

Matplotlib is one of the oldest and most widely used libraries for creating static, animated, and interactive visualizations in Python. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, etc.

Key Features:

- Support for 2D plotting
- Extensive charting options (line plots, histograms, scatter plots, etc.)
- Fully customizable plots

4. Seaborn

Seaborn is a powerful Python data visualization library built on top of Matplotlib, designed to make it easier to create attractive and informative statistical graphics. Seaborn is widely used by data scientists due to its ease of use, intuitive syntax, and integration with Pandas, which allows seamless plotting directly from DataFrames.

Key Features:

- High-level interface for drawing statistical plots
- Supports themes for better aesthetics
- Integrates with Pandas DataFrames

6.2MICROSOFT POWER BI – FOR CREATING DASHBOARD

Power BI is a **Data Visualization**, and **Business Intelligence** tool which helps to convert data from different data sources into interactive dashboards and BI reports. It provides interactive visualizations with self-service business intelligence capabilities where end users can create reports and dashboards by themselves, without having to depend on information technology staff or database administrators.

Power BI provides multiple connectors, software, and services. These services based on the **SaaS** and mobile Power BI apps which are available for different platforms. These set of services are used by business users to consume data and to build BI reports.

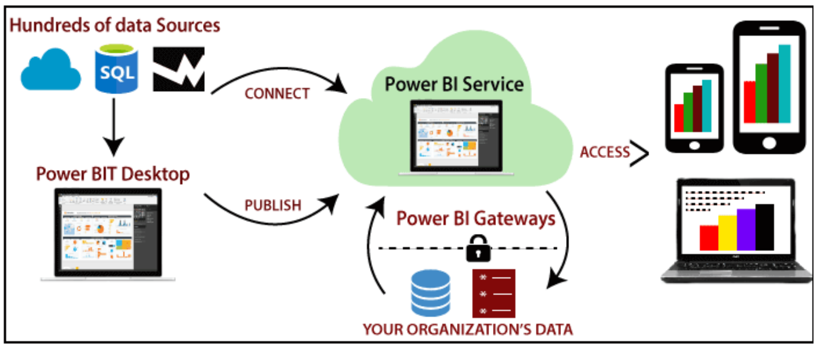


Fig 4: Microsoft Power BI

Power BI desktop app is used to create reports, while Power BI Service (Software as a Service - SaaS) is used to publish those reports. And Power BI mobile app is used to view the reports and dashboards.

Different Power BI version like Desktop, Service-based (SaaS), and mobile Power BI apps are used in different platforms.

Here are some significant reasons to use the Power BI tool:

- It allows real-time dashboard updates.
- It provides secure and reliable connections to the data sources in the cloud.
- It allows data exploration using a natural language query.
- Power BI provides a hybrid configuration, quick deployment, and secure environment.
- It provides features for dashboard visualization regularly updated with the community.
- It provides pre-built dashboards and reports for SaaS solutions.

Power BI Architecture

The architecture of Power BI is shown as below:

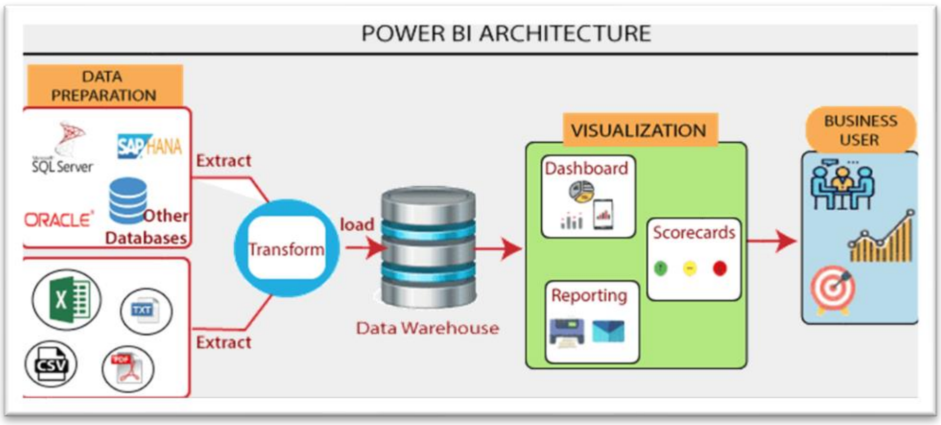


Fig 5: Power BI architecture

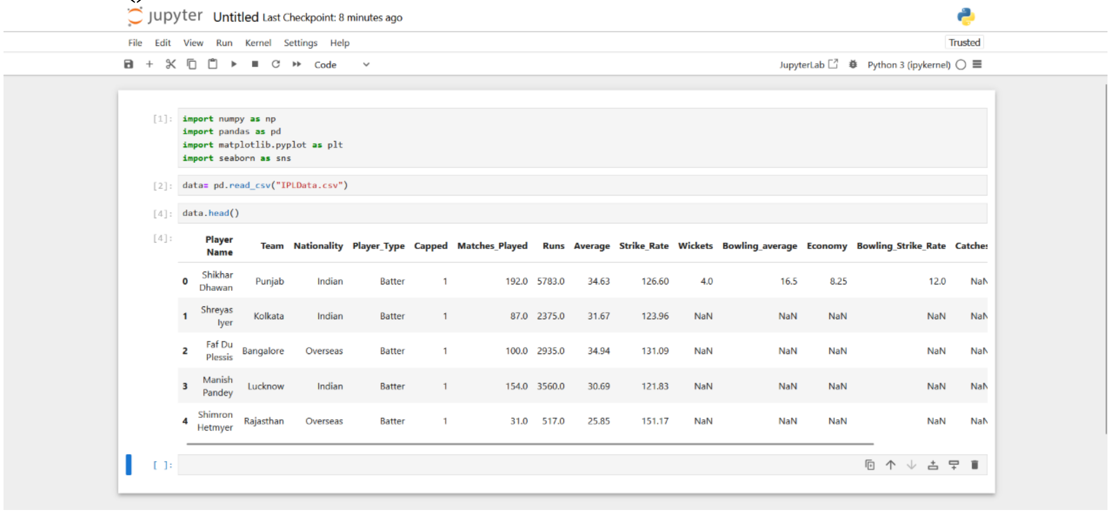
Power BI architecture has three phases. The first two phases use ETL (extract, transform, and load) process to handle the data.

1. **Data Integration:** An organization needs to deal with the data that comes from different sources. First, extract the data from different sources which can be your separate database, servers, etc. Then the data is integrated into a standard format and stored at a common area that's called staging area.
2. **Data Processing:** Still, the integrated data is not ready for visualization because the data needs processing before it can be presented. This data is pre-processed. **For example,** the missing values or redundant values will be removed from the data sets .After that, the business rules will be applied to the data, and it transforms into presentable data. Then this data will be loaded into the data warehouse.
3. **Data presentation:** Once the data is loaded and processed, then it can be visualized much better with the use of various visualization that Power BI offers. By using of dashboard and reports, we represent the data more intuitively. These visual reports help business end-users to take business decision based on the insights.

7.IMPLEMENTATION APPROACH

SOURCE CODE:

```
# In[90]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# In[91]:
data= pd.read_csv("IPLData.csv")

# In[92]:
data.head()
```

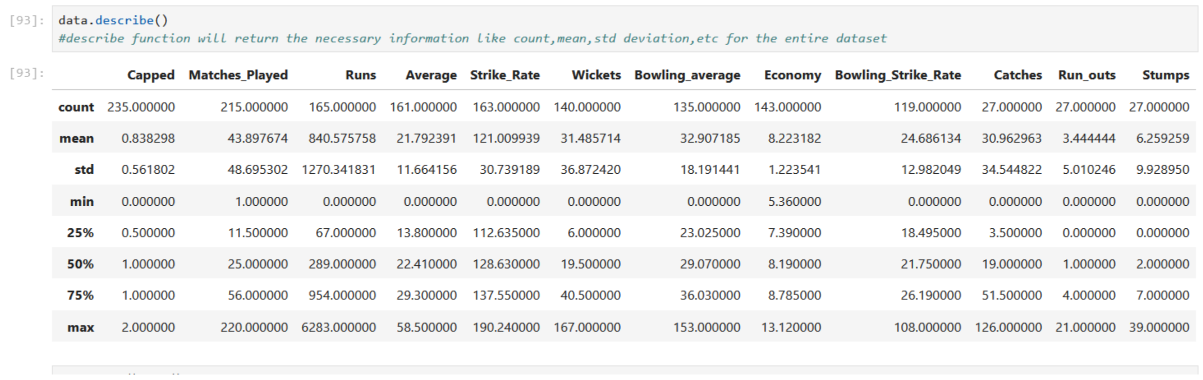

jupyter Untitled Last Checkpoint: 8 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

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

[2]: data= pd.read_csv("IPLData.csv")

[4]: data.head()

[4]:
```

	Player Name	Team	Nationality	Player_Type	Capped	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate	Catche
0	Shikhar Dhawan	Punjab	Indian	Batter	1	192.0	5783.0	34.63	126.60	4.0	16.5	8.25	12.0	NaN
1	Shreyas Iyer	Kolkata	Indian	Batter	1	87.0	2375.0	31.67	123.96	NaN	NaN	NaN	NaN	NaN
2	Faf Du Plessis	Bangalore	Overseas	Batter	1	100.0	2935.0	34.94	131.09	NaN	NaN	NaN	NaN	NaN
3	Manish Pandey	Lucknow	Indian	Batter	1	154.0	3560.0	30.69	121.83	NaN	NaN	NaN	NaN	NaN
4	Shimron Hetmyer	Rajasthan	Overseas	Batter	1	31.0	517.0	25.85	151.17	NaN	NaN	NaN	NaN	NaN

```
[ ]:
```

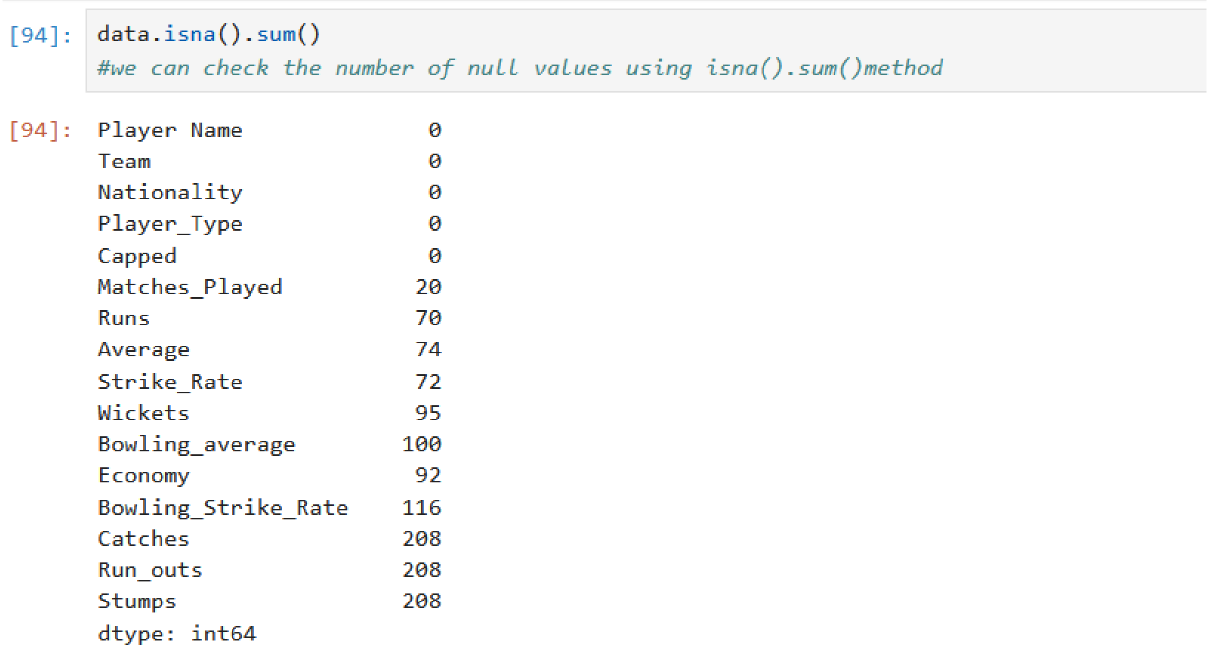
```
# In[93]:
data.describe()
#describe function will return the necessary information like count,mean,std deviation,etc for the entire dataset
```



```
[93]: data.describe()
      #describe function will return the necessary information like count,mean,std deviation,etc for the entire dataset

[93]:
```

	Capped	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate	Catches	Run_outs	Stumps
count	235.000000	215.000000	165.000000	161.000000	163.000000	140.000000	135.000000	143.000000	119.000000	27.000000	27.000000	27.000000
mean	0.838298	43.897674	840.575758	21.792391	121.009939	31.485714	32.907185	8.223182	24.686134	30.962963	3.444444	6.259259
std	0.561802	48.695302	1270.341831	11.664156	30.739189	36.872420	18.191441	1.223541	12.982049	34.544822	5.010246	9.928950
min	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.360000	0.000000	0.000000	0.000000	0.000000
25%	0.500000	11.500000	67.000000	13.800000	112.635000	6.000000	23.025000	7.390000	18.495000	3.500000	0.000000	0.000000
50%	1.000000	25.000000	289.000000	22.410000	128.630000	19.500000	29.070000	8.190000	21.750000	19.000000	1.000000	2.000000
75%	1.000000	56.000000	954.000000	29.300000	137.550000	40.500000	36.030000	8.785000	26.190000	51.500000	4.000000	7.000000
max	2.000000	220.000000	6283.000000	58.500000	190.240000	167.000000	153.000000	13.120000	108.000000	126.000000	21.000000	39.000000

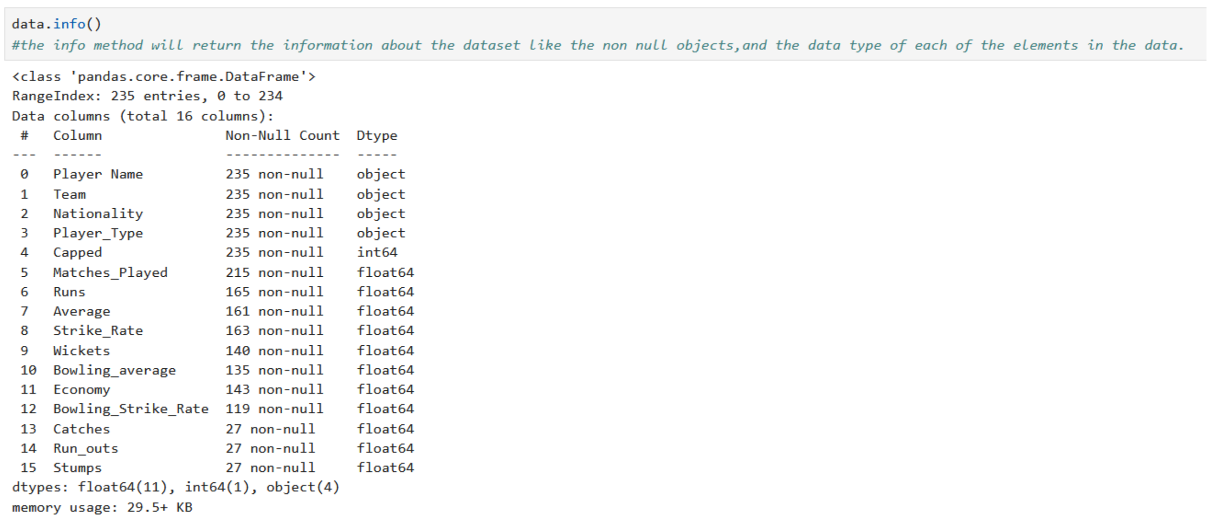
```
# In[94]:
data.isna().sum()
#we can check the number of null values using isna().sum()method
```



```
[94]: data.isna().sum()
      #we can check the number of null values using isna().sum()method
```

```
[94]: Player Name             0
      Team                    0
      Nationality             0
      Player_Type             0
      Capped                  0
      Matches_Played         20
      Runs                   70
      Average                74
      Strike_Rate            72
      Wickets                95
      Bowling_average       100
      Economy                92
      Bowling_Strike_Rate   116
      Catches               208
      Run_outs              208
      Stumps                208
      dtype: int64
```

```
# In[95]:
data.info()
#the info method will return the information about the dataset like the non null objects,and the
data type of each of the elements in the data.
```



```
data.info()
#the info method will return the information about the dataset like the non null objects,and the data type of each of the elements in the data.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 235 entries, 0 to 234
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Player Name          235 non-null    object 
 1   Team                 235 non-null    object 
 2   Nationality          235 non-null    object 
 3   Player_Type          235 non-null    object 
 4   Capped               235 non-null    int64  
 5   Matches_Played       215 non-null    float64
 6   Runs                 165 non-null    float64
 7   Average              161 non-null    float64
 8   Strike_Rate          163 non-null    float64
 9   Wickets              140 non-null    float64
 10  Bowling_average      135 non-null    float64
 11  Economy              143 non-null    float64
 12  Bowling_Strike_Rate  119 non-null    float64
 13  Catches              27 non-null     float64
 14  Run_outs             27 non-null     float64
 15  Stumps               27 non-null     float64
dtypes: float64(11), int64(1), object(4)
memory usage: 29.5+ KB
```

```
# In[96]:
#segregating Data Capped Batters

#we have kept the batting parameters to restrict the analysis for batter specific tendencies.
batters= data.loc[(data["Player_Type"] == "Batter")]

batters_new= batters.loc[(batters["Capped"] == 1)]

Capped_Batters= batters_new[['Player Name',

'Team',
```

'Nationality',

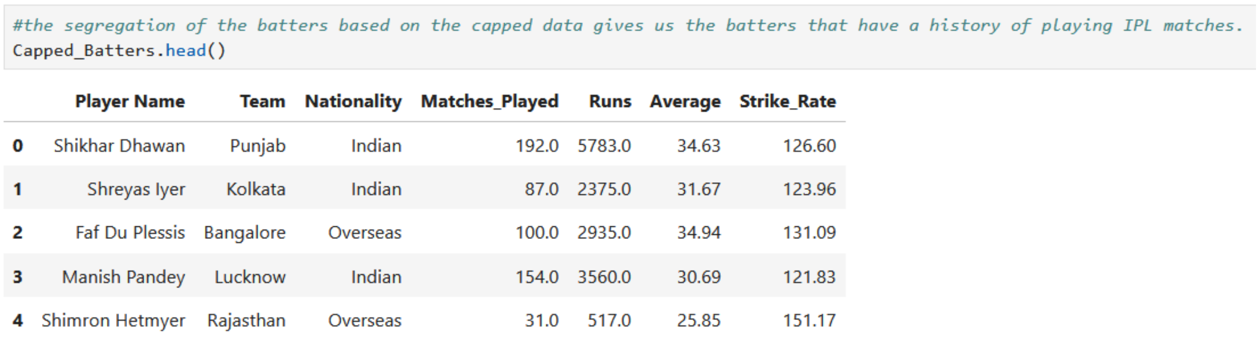
'Matches_Played',

'Runs',

'Average',

'Strike_Rate']]

# In[97]:
#the segregation of the batters based on the capped data gives us the batters that have a history of playing IPL matches.
Capped_Batters.head()



```
#the segregation of the batters based on the capped data gives us the batters that have a history of playing IPL matches.
Capped_Batters.head()
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate
0	Shikhar Dhawan	Punjab	Indian	192.0	5783.0	34.63	126.60
1	Shreyas Iyer	Kolkata	Indian	87.0	2375.0	31.67	123.96
2	Faf Du Plessis	Bangalore	Overseas	100.0	2935.0	34.94	131.09
3	Manish Pandey	Lucknow	Indian	154.0	3560.0	30.69	121.83
4	Shimron Hetmyer	Rajasthan	Overseas	31.0	517.0	25.85	151.17

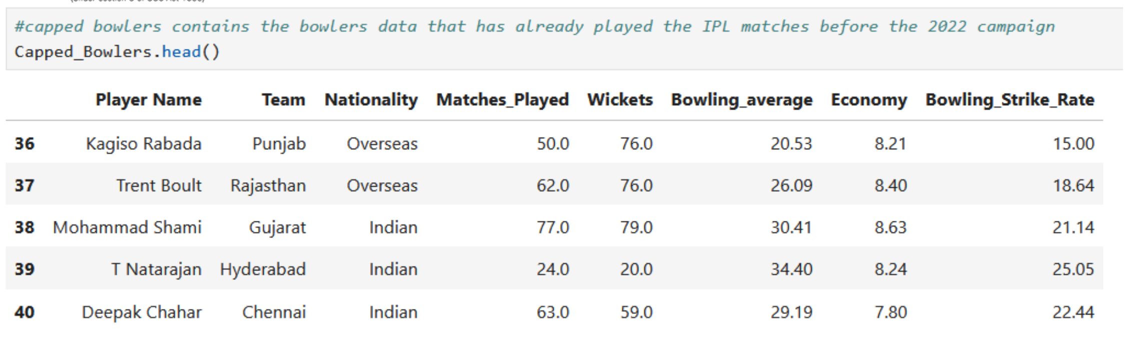
# In[98]:
#segregating Data -Capped Bowlers

#we have only kept the bowling parameters, since we will purely judge the bowlers on their bowling statistics.
bowlers = data.loc[(data["Player_Type"] == "Bowler ")]

bowlers_new= bowlers.loc[(bowlers["Capped"]== 1)]

Capped_Bowlers= bowlers_new[['Player Name',
'Team',
'Nationality',
'Matches_Played',
'Wickets',
'Bowling_average',
'Economy',
'Bowling_Strike_Rate']]

# In[99]:
#capped bowlers contains the bowlers data that has already played the IPL matches before the 2022 campaign
Capped_Bowlers.head()



```
#capped bowlers contains the bowlers data that has already played the IPL matches before the 2022 campaign
Capped_Bowlers.head()
```

	Player Name	Team	Nationality	Matches_Played	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
36	Kagiso Rabada	Punjab	Overseas	50.0	76.0	20.53	8.21	15.00
37	Trent Boult	Rajasthan	Overseas	62.0	76.0	26.09	8.40	18.64
38	Mohammad Shami	Gujarat	Indian	77.0	79.0	30.41	8.63	21.14
39	T Natarajan	Hyderabad	Indian	24.0	20.0	34.40	8.24	25.05
40	Deepak Chahar	Chennai	Indian	63.0	59.0	29.19	7.80	22.44

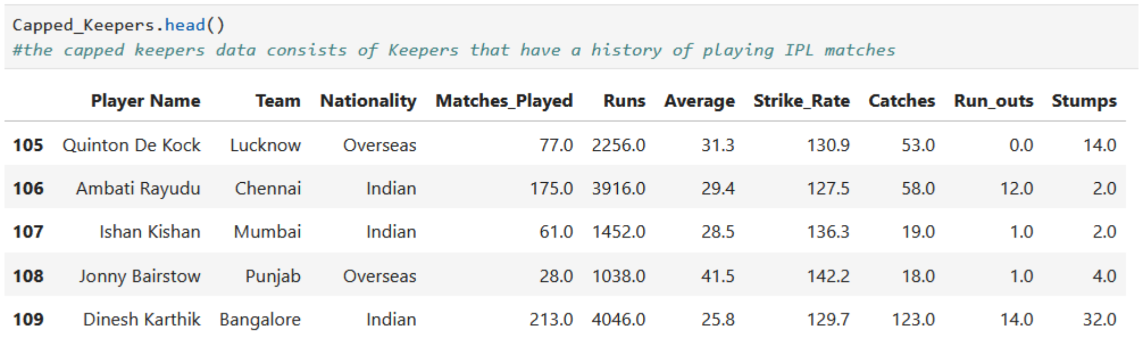
# In[100]:
#segregating Data Capped Keepers

#We are only keeping the relavant parameters for the keepers, since bowling figures are not needed for the wicket-keepers.
Keepers= data.loc[(data["Player_Type"] == "Keeper")]

Keepers_new = Keepers.loc[(Keepers["Capped"] == 1)]

Capped_Keepers =Keepers_new[['Player Name',
'Team',
'Nationality',
'Matches_Played',
'Runs',
'Average',
'Strike_Rate',
'Catches',
'Run_outs',
'Stumps']]

# In[101]:
Capped_Keepers.head()
#the capped keepers data consists of Keepers that have a history of playing IPL matches



```
Capped_Keepers.head()
#the capped keepers data consists of Keepers that have a history of playing IPL matches
```

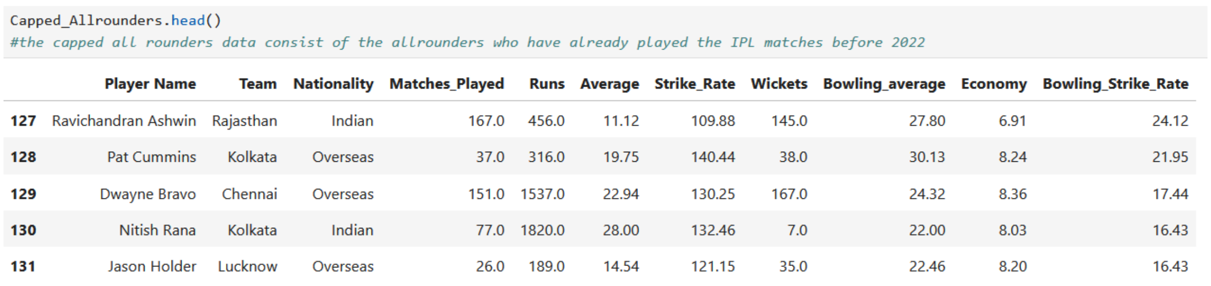
	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.3	130.9	53.0	0.0	14.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.4	127.5	58.0	12.0	2.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.5	136.3	19.0	1.0	2.0
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.5	142.2	18.0	1.0	4.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.8	129.7	123.0	14.0	32.0

# In[102]:
#we have kept the bowling and batting parameters for the allrounders to analyze all round performance.
#segregating Data - capped Allrounders
Allrounders = data.loc[(data["Player_Type"] == "Allrounder")]

Allrounders_new = Allrounders.loc[(Allrounders["Capped"] == 1)]

Capped_Allrounders = Allrounders_new[['Player Name',
'Team',
'Nationality',
'Matches_Played',
'Runs',
'Average',
'Strike_Rate',
'Wickets',
'Bowling_average',
'Economy',
'Bowling_Strike_Rate']]

# In[103]:
Capped_Allrounders.head()
#the capped all rounders data consist of the allrounders who have already played the IPL matches before 2022



```
Capped_Allrounders.head()
#the capped all rounders data consist of the allrounders who have already played the IPL matches before 2022
```

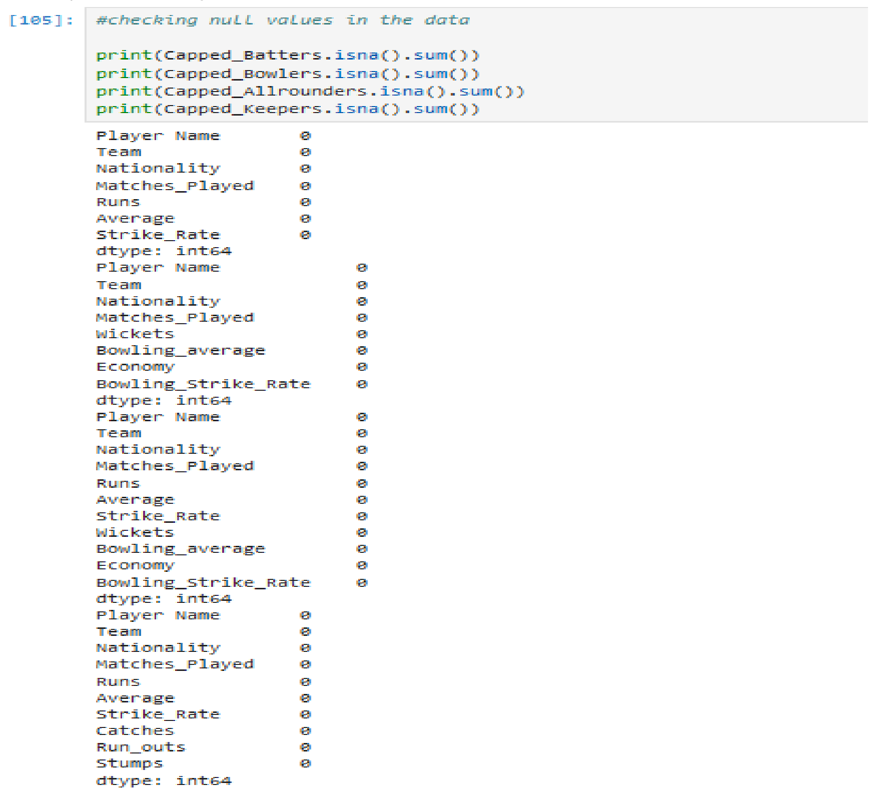
	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
127	Ravichandran Ashwin	Rajasthan	Indian	167.0	456.0	11.12	109.88	145.0	27.80	6.91	24.12
128	Pat Cummins	Kolkata	Overseas	37.0	316.0	19.75	140.44	38.0	30.13	8.24	21.95
129	Dwayne Bravo	Chennai	Overseas	151.0	1537.0	22.94	130.25	167.0	24.32	8.36	17.44
130	Nitish Rana	Kolkata	Indian	77.0	1820.0	28.00	132.46	7.0	22.00	8.03	16.43
131	Jason Holder	Lucknow	Overseas	26.0	189.0	14.54	121.15	35.0	22.46	8.20	16.43

# In[104]:
#Cleaning the data by making the null or NAN values 0.

Capped_Batters= Capped_Batters.fillna(0)
Capped_Bowlers= Capped_Bowlers.fillna(0)
Capped_Allrounders= Capped_Allrounders.fillna(0)
Capped_Keepers= Capped_Keepers.fillna(0)

# In[105]:
#checking null values in the data

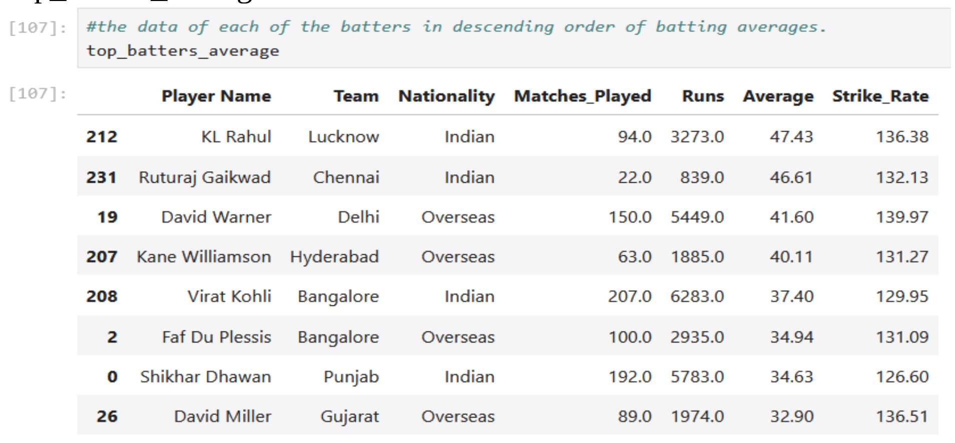
print(Capped_Batters.isna().sum())
print(Capped_Bowlers.isna().sum())
print(Capped_Allrounders.isna().sum())
print(Capped_Keepers.isna().sum())



```
[105]: #checking null values in the data

       print(Capped_Batters.isna().sum())
       print(Capped_Bowlers.isna().sum())
       print(Capped_Allrounders.isna().sum())
       print(Capped_Keepers.isna().sum())
```

```
Player Name       0
Team              0
Nationality       0
Matches_Played    0
Runs              0
Average           0
Strike_Rate       0
dtype: int64
Player Name            0
Team                   0
Nationality            0
Matches_Played         0
Wickets                0
Bowling_average        0
Economy                0
Bowling_Strike_Rate    0
dtype: int64
Player Name            0
Team                   0
Nationality            0
Matches_Played         0
Runs                   0
Average                0
Strike_Rate            0
Wickets                0
Bowling_average        0
Economy                0
Bowling_Strike_Rate    0
dtype: int64
Player Name       0
Team              0
Nationality       0
Matches_Played    0
Runs              0
Average           0
Strike_Rate       0
Catches           0
Run_outs          0
Stumps            0
dtype: int64
```

```
# In[106]:
#Analyzing the Batters Data

#Here we have narrowed our analysis to batters who have a batting average more the 32.0
top_batters = Capped_Batters.loc[(Capped_Batters ["Average"] >= 32.0)]

#Sorting the data in descending order with respect to each parameter
top_batters_average= top_batters.sort_values('Average', ascending=False)
top_batters_strike_rate = top_batters.sort_values('Strike_Rate', ascending=False)
top_batters_runs = top_batters.sort_values('Runs', ascending=False)
top_batters_matches = top_batters.sort_values('Matches_Played', ascending=False)

# In[107]:
#the data of each of the batters in descending order of batting averages.
top_batters_average
```



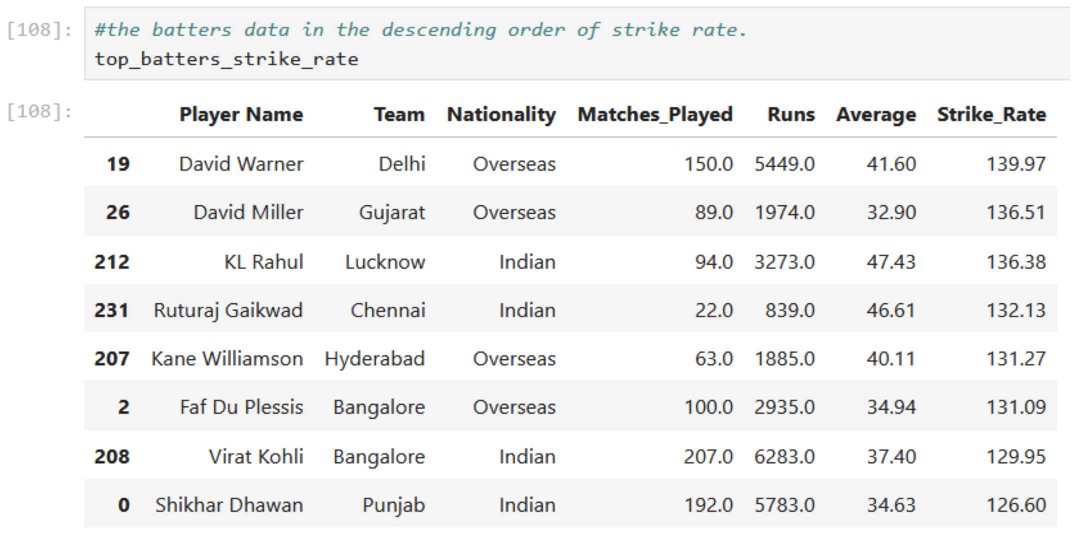
```
[107]: #the data of each of the batters in descending order of batting averages.
       top_batters_average
```

[107]:

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate
212	KL Rahul	Lucknow	Indian	94.0	3273.0	47.43	136.38
231	Ruturaj Gaikwad	Chennai	Indian	22.0	839.0	46.61	132.13
19	David Warner	Delhi	Overseas	150.0	5449.0	41.60	139.97
207	Kane Williamson	Hyderabad	Overseas	63.0	1885.0	40.11	131.27
208	Virat Kohli	Bangalore	Indian	207.0	6283.0	37.40	129.95
2	Faf Du Plessis	Bangalore	Overseas	100.0	2935.0	34.94	131.09
0	Shikhar Dhawan	Punjab	Indian	192.0	5783.0	34.63	126.60
26	David Miller	Gujarat	Overseas	89.0	1974.0	32.90	136.51

# In[108]:
#the batters data in the descending order of strike rate.

top_batters_strike_rate



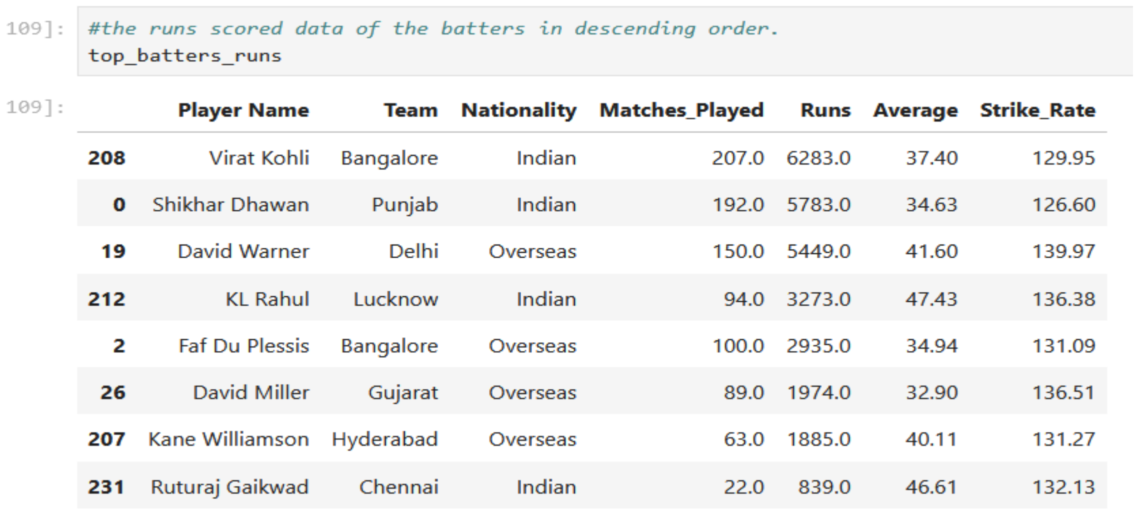
```
[108]: #the batters data in the descending order of strike rate.
       top_batters_strike_rate
```

[108]:

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate
19	David Warner	Delhi	Overseas	150.0	5449.0	41.60	139.97
26	David Miller	Gujarat	Overseas	89.0	1974.0	32.90	136.51
212	KL Rahul	Lucknow	Indian	94.0	3273.0	47.43	136.38
231	Ruturaj Gaikwad	Chennai	Indian	22.0	839.0	46.61	132.13
207	Kane Williamson	Hyderabad	Overseas	63.0	1885.0	40.11	131.27
2	Faf Du Plessis	Bangalore	Overseas	100.0	2935.0	34.94	131.09
208	Virat Kohli	Bangalore	Indian	207.0	6283.0	37.40	129.95
0	Shikhar Dhawan	Punjab	Indian	192.0	5783.0	34.63	126.60

# In[109]:
#the runs scored data of the batters in descending order.

top_batters_runs



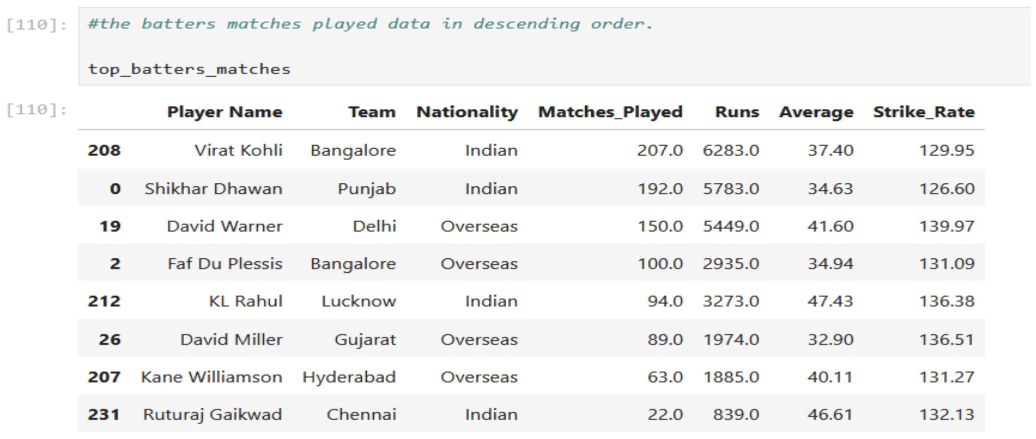
```
109]: #the runs scored data of the batters in descending order.
      top_batters_runs
```

109]:

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate
208	Virat Kohli	Bangalore	Indian	207.0	6283.0	37.40	129.95
0	Shikhar Dhawan	Punjab	Indian	192.0	5783.0	34.63	126.60
19	David Warner	Delhi	Overseas	150.0	5449.0	41.60	139.97
212	KL Rahul	Lucknow	Indian	94.0	3273.0	47.43	136.38
2	Faf Du Plessis	Bangalore	Overseas	100.0	2935.0	34.94	131.09
26	David Miller	Gujarat	Overseas	89.0	1974.0	32.90	136.51
207	Kane Williamson	Hyderabad	Overseas	63.0	1885.0	40.11	131.27
231	Ruturaj Gaikwad	Chennai	Indian	22.0	839.0	46.61	132.13

# In[110]:
#the batters matches played data in descending order.

top_batters_matches

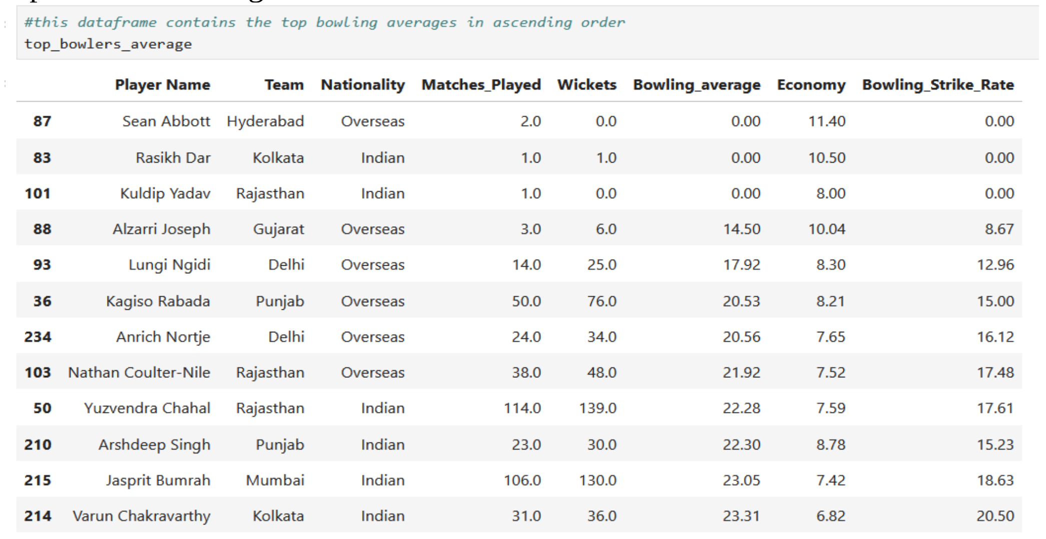


```
[110]: #the batters matches played data in descending order.

       top_batters_matches
```

[110]:

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate
208	Virat Kohli	Bangalore	Indian	207.0	6283.0	37.40	129.95
0	Shikhar Dhawan	Punjab	Indian	192.0	5783.0	34.63	126.60
19	David Warner	Delhi	Overseas	150.0	5449.0	41.60	139.97
2	Faf Du Plessis	Bangalore	Overseas	100.0	2935.0	34.94	131.09
212	KL Rahul	Lucknow	Indian	94.0	3273.0	47.43	136.38
26	David Miller	Gujarat	Overseas	89.0	1974.0	32.90	136.51
207	Kane Williamson	Hyderabad	Overseas	63.0	1885.0	40.11	131.27
231	Ruturaj Gaikwad	Chennai	Indian	22.0	839.0	46.61	132.13

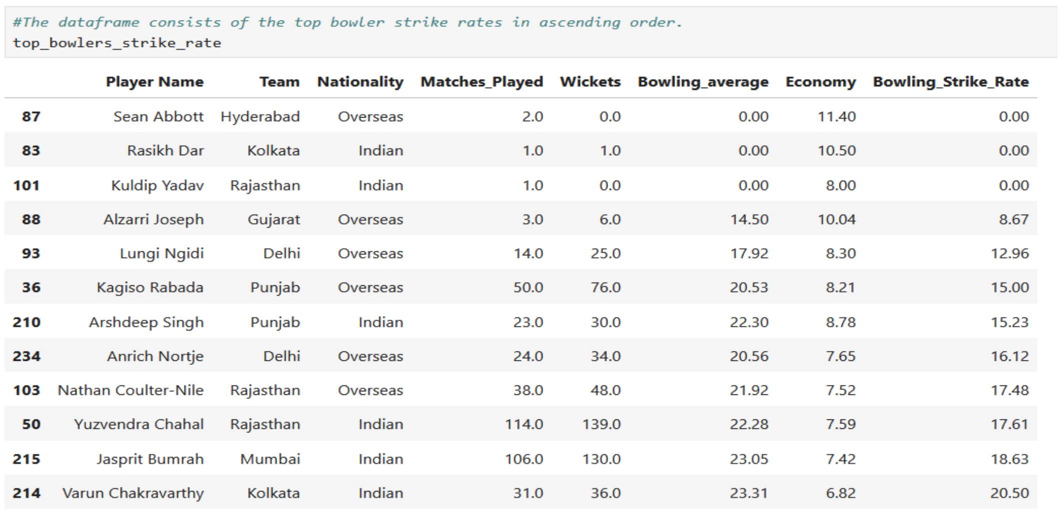
```
# In[111]:
#Analyzing the Bowlers Data

#here we have narrowed our analysis based on the bowling averages of the players to be less than 24.0
top_bowlers=Capped_Bowlers.loc[(Capped_Bowlers["Bowling_average"] <= 24.0)]
top_bowlers_average=top_bowlers.sort_values('Bowling_average')
top_bowlers_strike_rate=top_bowlers.sort_values('Bowling_Strike_Rate')
top_bowlers_wickets=top_bowlers.sort_values('Wickets', ascending=False)
top_bowlers_economy=top_bowlers.sort_values('Economy')
top_bowlers_matches=top_bowlers.sort_values('Matches_Played', ascending=False)

# In[112]:
#this dataframe contains the top bowling averages in ascending order
top_bowlers_average
```



```
#this dataframe contains the top bowling averages in ascending order
top_bowlers_average
```

	Player Name	Team	Nationality	Matches_Played	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
87	Sean Abbott	Hyderabad	Overseas	2.0	0.0	0.00	11.40	0.00
83	Rasikh Dar	Kolkata	Indian	1.0	1.0	0.00	10.50	0.00
101	Kuldip Yadav	Rajasthan	Indian	1.0	0.0	0.00	8.00	0.00
88	Alzarri Joseph	Gujarat	Overseas	3.0	6.0	14.50	10.04	8.67
93	Lungi Ngidi	Delhi	Overseas	14.0	25.0	17.92	8.30	12.96
36	Kagiso Rabada	Punjab	Overseas	50.0	76.0	20.53	8.21	15.00
234	Anrich Nortje	Delhi	Overseas	24.0	34.0	20.56	7.65	16.12
103	Nathan Coulter-Nile	Rajasthan	Overseas	38.0	48.0	21.92	7.52	17.48
50	Yuzvendra Chahal	Rajasthan	Indian	114.0	139.0	22.28	7.59	17.61
210	Arshdeep Singh	Punjab	Indian	23.0	30.0	22.30	8.78	15.23
215	Jasprit Bumrah	Mumbai	Indian	106.0	130.0	23.05	7.42	18.63
214	Varun Chakravarthy	Kolkata	Indian	31.0	36.0	23.31	6.82	20.50

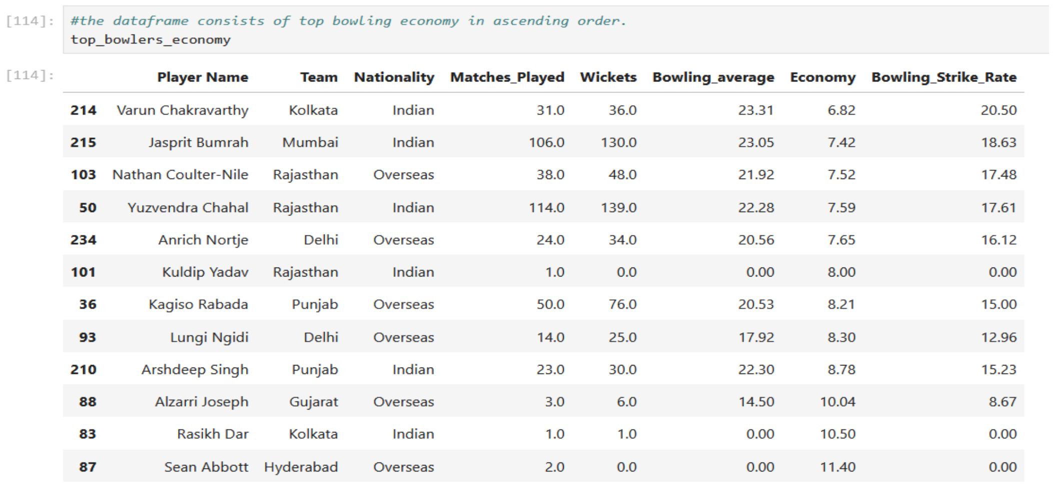
```
# In[113]:
#The dataframe consists of the top bowler strike rates in ascending order.
top_bowlers_strike_rate
```



```
#The dataframe consists of the top bowler strike rates in ascending order.
top_bowlers_strike_rate
```

	Player Name	Team	Nationality	Matches_Played	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
87	Sean Abbott	Hyderabad	Overseas	2.0	0.0	0.00	11.40	0.00
83	Rasikh Dar	Kolkata	Indian	1.0	1.0	0.00	10.50	0.00
101	Kuldip Yadav	Rajasthan	Indian	1.0	0.0	0.00	8.00	0.00
88	Alzarri Joseph	Gujarat	Overseas	3.0	6.0	14.50	10.04	8.67
93	Lungi Ngidi	Delhi	Overseas	14.0	25.0	17.92	8.30	12.96
36	Kagiso Rabada	Punjab	Overseas	50.0	76.0	20.53	8.21	15.00
210	Arshdeep Singh	Punjab	Indian	23.0	30.0	22.30	8.78	15.23
234	Anrich Nortje	Delhi	Overseas	24.0	34.0	20.56	7.65	16.12
103	Nathan Coulter-Nile	Rajasthan	Overseas	38.0	48.0	21.92	7.52	17.48
50	Yuzvendra Chahal	Rajasthan	Indian	114.0	139.0	22.28	7.59	17.61
215	Jasprit Bumrah	Mumbai	Indian	106.0	130.0	23.05	7.42	18.63
214	Varun Chakravarthy	Kolkata	Indian	31.0	36.0	23.31	6.82	20.50

# In[114]:
#the dataframe consists of top bowling economy in ascending order.
top_bowlers_economy

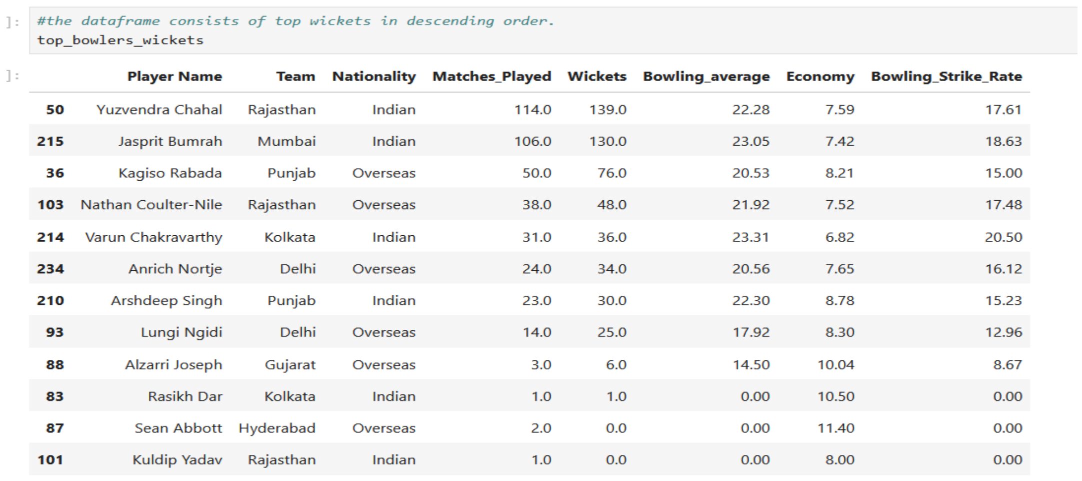


```
[114]: #the dataframe consists of top bowling economy in ascending order.
       top_bowlers_economy
```

[114]:

	Player Name	Team	Nationality	Matches_Played	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
214	Varun Chakravarthy	Kolkata	Indian	31.0	36.0	23.31	6.82	20.50
215	Jasprit Bumrah	Mumbai	Indian	106.0	130.0	23.05	7.42	18.63
103	Nathan Coulter-Nile	Rajasthan	Overseas	38.0	48.0	21.92	7.52	17.48
50	Yuzvendra Chahal	Rajasthan	Indian	114.0	139.0	22.28	7.59	17.61
234	Anrich Nortje	Delhi	Overseas	24.0	34.0	20.56	7.65	16.12
101	Kuldip Yadav	Rajasthan	Indian	1.0	0.0	0.00	8.00	0.00
36	Kagiso Rabada	Punjab	Overseas	50.0	76.0	20.53	8.21	15.00
93	Lungi Ngidi	Delhi	Overseas	14.0	25.0	17.92	8.30	12.96
210	Arshdeep Singh	Punjab	Indian	23.0	30.0	22.30	8.78	15.23
88	Alzarri Joseph	Gujarat	Overseas	3.0	6.0	14.50	10.04	8.67
83	Rasikh Dar	Kolkata	Indian	1.0	1.0	0.00	10.50	0.00
87	Sean Abbott	Hyderabad	Overseas	2.0	0.0	0.00	11.40	0.00

# In[115]:
#the dataframe consists of top wickets in descending order.
top_bowlers_wickets

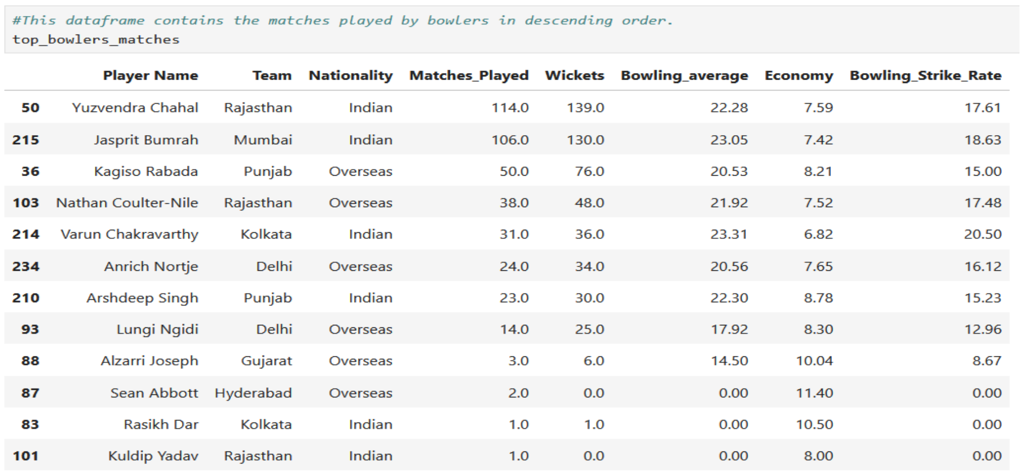


```
]: #the dataframe consists of top wickets in descending order.
   top_bowlers_wickets
```

]:

	Player Name	Team	Nationality	Matches_Played	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
50	Yuzvendra Chahal	Rajasthan	Indian	114.0	139.0	22.28	7.59	17.61
215	Jasprit Bumrah	Mumbai	Indian	106.0	130.0	23.05	7.42	18.63
36	Kagiso Rabada	Punjab	Overseas	50.0	76.0	20.53	8.21	15.00
103	Nathan Coulter-Nile	Rajasthan	Overseas	38.0	48.0	21.92	7.52	17.48
214	Varun Chakravarthy	Kolkata	Indian	31.0	36.0	23.31	6.82	20.50
234	Anrich Nortje	Delhi	Overseas	24.0	34.0	20.56	7.65	16.12
210	Arshdeep Singh	Punjab	Indian	23.0	30.0	22.30	8.78	15.23
93	Lungi Ngidi	Delhi	Overseas	14.0	25.0	17.92	8.30	12.96
88	Alzarri Joseph	Gujarat	Overseas	3.0	6.0	14.50	10.04	8.67
83	Rasikh Dar	Kolkata	Indian	1.0	1.0	0.00	10.50	0.00
87	Sean Abbott	Hyderabad	Overseas	2.0	0.0	0.00	11.40	0.00
101	Kuldip Yadav	Rajasthan	Indian	1.0	0.0	0.00	8.00	0.00

```
# In[116]:
#This dataframe contains the matches played by bowlers in descending order.
top_bowlers_matches
```



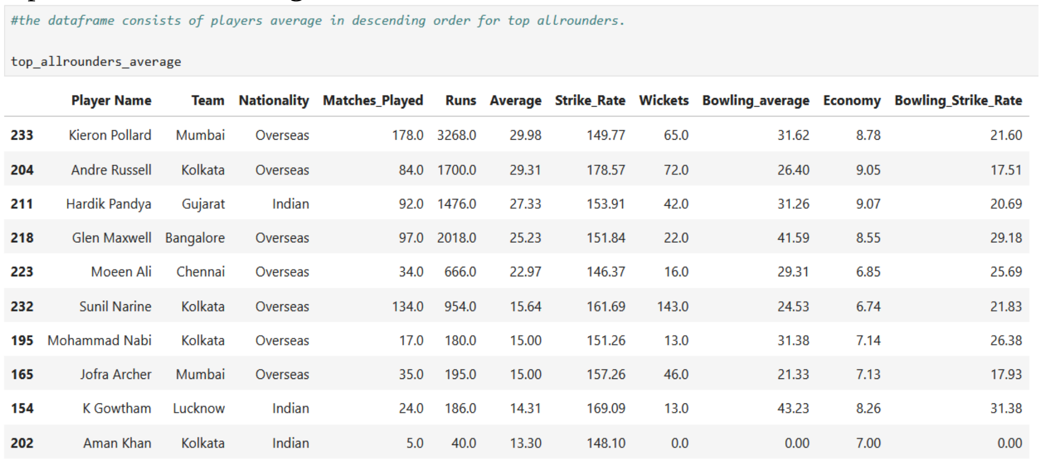
```
#This dataframe contains the matches played by bowlers in descending order.
top_bowlers_matches
```

	Player Name	Team	Nationality	Matches_Played	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
50	Yuzvendra Chahal	Rajasthan	Indian	114.0	139.0	22.28	7.59	17.61
215	Jasprit Bumrah	Mumbai	Indian	106.0	130.0	23.05	7.42	18.63
36	Kagiso Rabada	Punjab	Overseas	50.0	76.0	20.53	8.21	15.00
103	Nathan Coulter-Nile	Rajasthan	Overseas	38.0	48.0	21.92	7.52	17.48
214	Varun Chakravarthy	Kolkata	Indian	31.0	36.0	23.31	6.82	20.50
234	Anrich Nortje	Delhi	Overseas	24.0	34.0	20.56	7.65	16.12
210	Arshdeep Singh	Punjab	Indian	23.0	30.0	22.30	8.78	15.23
93	Lungi Ngidi	Delhi	Overseas	14.0	25.0	17.92	8.30	12.96
88	Alzarri Joseph	Gujarat	Overseas	3.0	6.0	14.50	10.04	8.67
87	Sean Abbott	Hyderabad	Overseas	2.0	0.0	0.00	11.40	0.00
83	Rasikh Dar	Kolkata	Indian	1.0	1.0	0.00	10.50	0.00
101	Kuldip Yadav	Rajasthan	Indian	1.0	0.0	0.00	8.00	0.00

```
# ### From the above analysis, if we rank down the parameters from 1-10. The top bowling options are as follows:
#
# 1. Kagiso Rabada
#
# 2. Jasprit Bumrah
#
# 3. Yuzvendra Chahal
#
# 4. Nathan Coulter-Nile

# In[117]:
#We have narrowed our analysis by further segregating the allrounders based on strike rate equal to or more than 140.0.

top_allrounders=Capped_Allrounders.loc[(Capped_Allrounders ["Strike_Rate"] >= 140.8)]

top_allrounders_average=top_allrounders.sort_values('Average', ascending=False)

top_alrounders_strike_rate = top_allrounders.sort_values('Strike_Rate', ascending=False)

top_allrounders_runs=top_allrounders.sort_values('Runs', ascending=False)

top_allrounders_matches=top_allrounders.sort_values('Matches_Played', ascending=False)

top_allrounders_bowling_average=top_allrounders.sort_values('Bowling_average')
top_allrounders_bowling_strike_rate=top_allrounders.sort_values('Bowling_Strike_Rate')

top_allrounders_wickets=top_allrounders.sort_values('Wickets', ascending=False)

top_allrounders_economy=top_allrounders.sort_values('Economy')

top_allrounders_matches= top_allrounders.sort_values('Matches_Played', ascending=False)
```

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

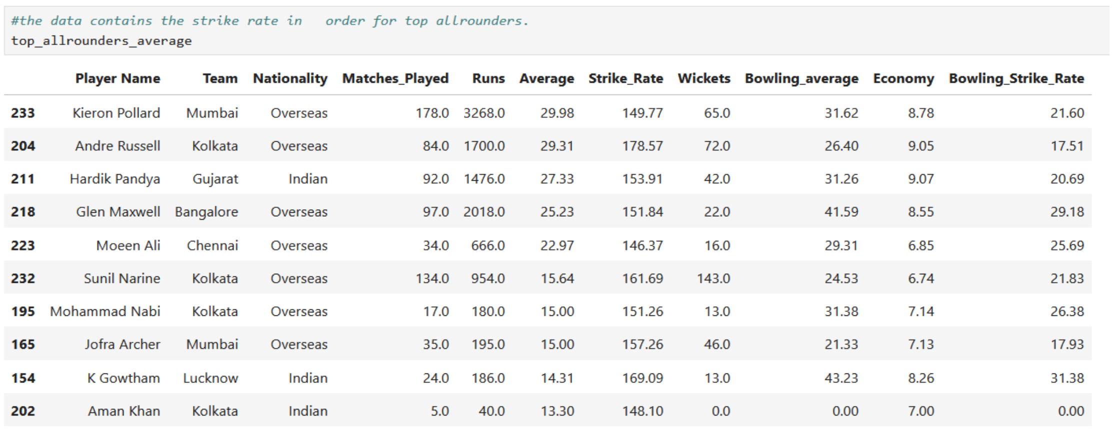
# In[118]:
#the dataframe consists of players average in descending order for top allrounders.
top_allrounders_average



```
#the dataframe consists of players average in descending order for top allrounders.

top_allrounders_average
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00

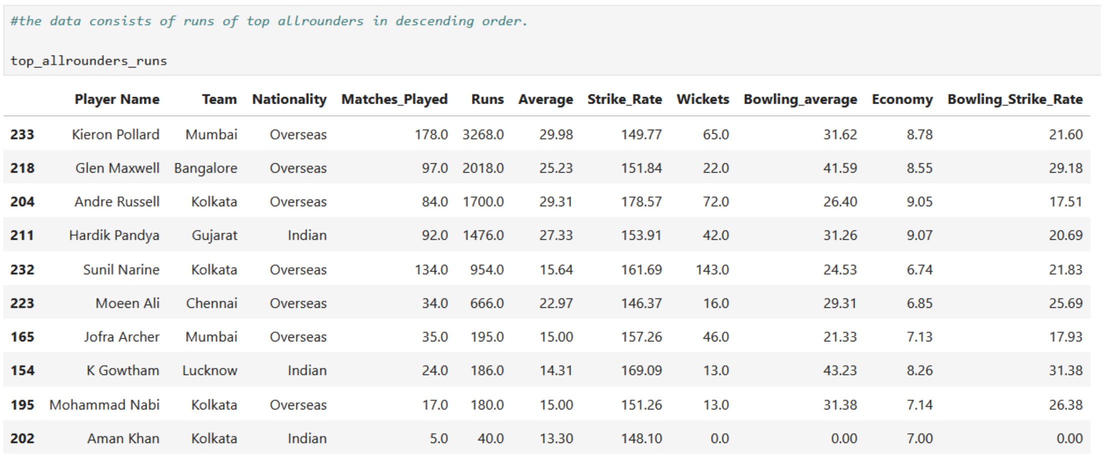
# In[119]:
#the data contains the strike rate in   order for top allrounders.
top_allrounders_average



```
#the data contains the strike rate in   order for top allrounders.
top_allrounders_average
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00

# In[120]:
#the data consists of runs of top allrounders in descending order.
top_allrounders_runs

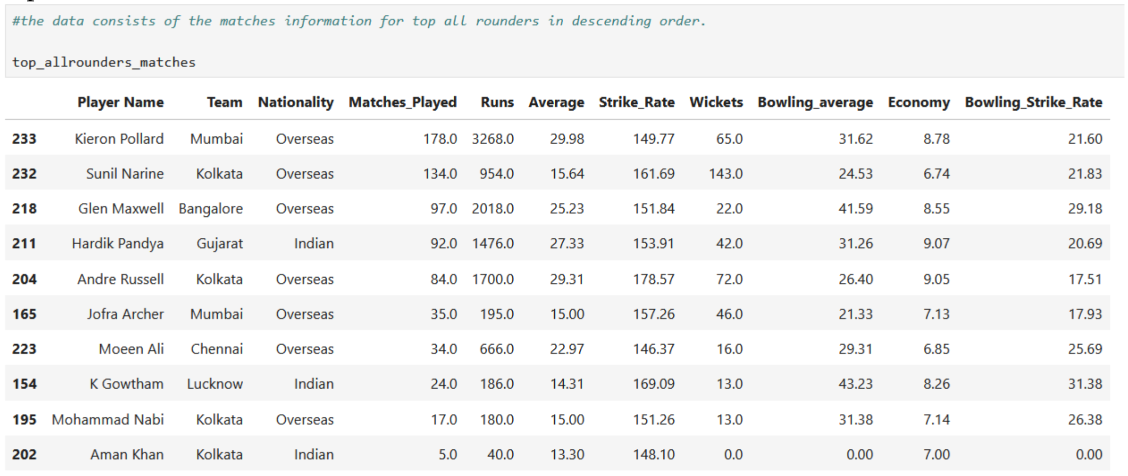


```
#the data consists of runs of top allrounders in descending order.

top_allrounders_runs
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

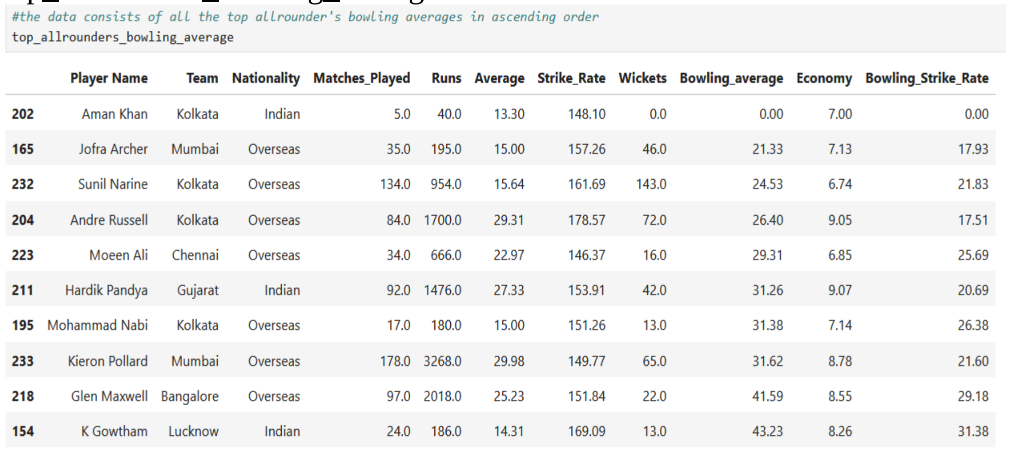
# In[121]:
#the data consists of the matches information for top all rounders in descending order.
top_allrounders_matches



```
#the data consists of the matches information for top all rounders in descending order.

top_allrounders_matches
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00

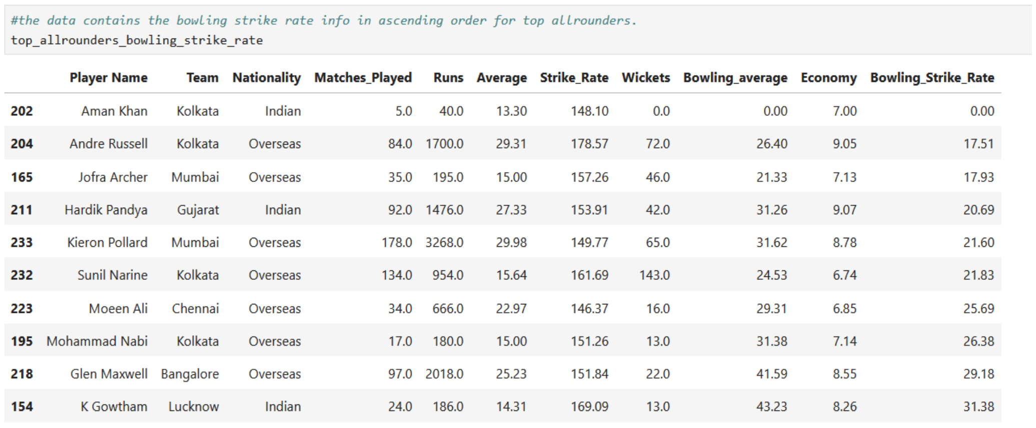
# In[122]:
#the data consists of all the top allrounder's bowling averages in ascending order
top_allrounders_bowling_average



```
#the data consists of all the top allrounder's bowling averages in ascending order
top_allrounders_bowling_average
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38

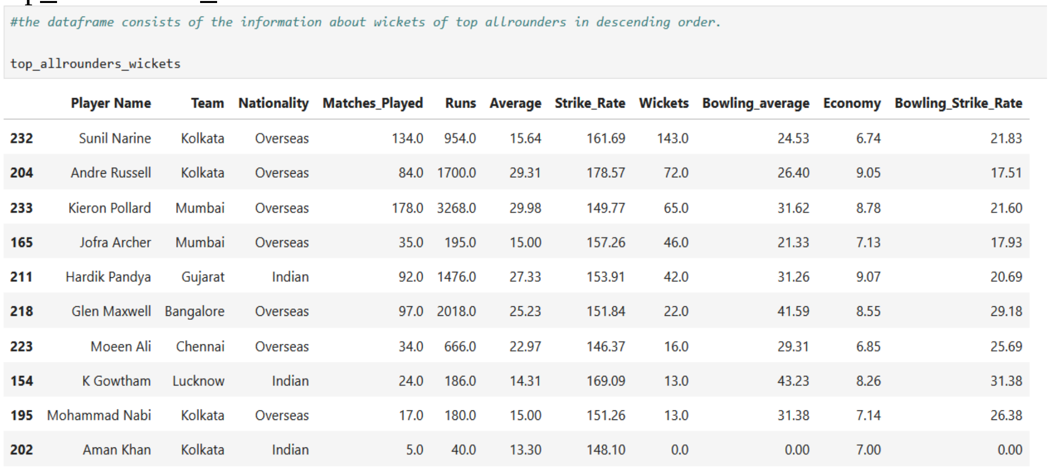
# In[123]:
#the data contains the bowling strike rate info in ascending order for top allrounders.
top_allrounders_bowling_strike_rate



```
#the data contains the bowling strike rate info in ascending order for top allrounders.
top_allrounders_bowling_strike_rate
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38

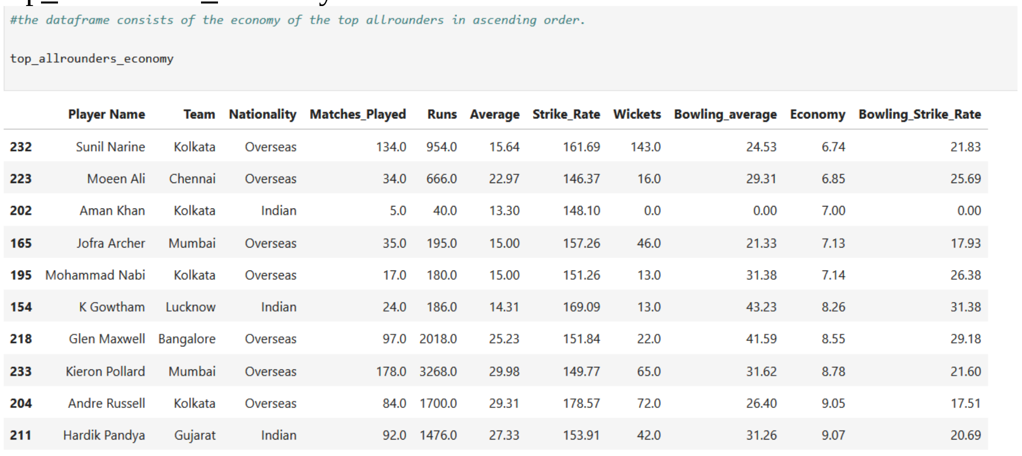
# In[124]:
#the dataframe consists of the information about wickets of top allrounders in descending order.
top_allrounders_wickets



```
#the dataframe consists of the information about wickets of top allrounders in descending order.

top_allrounders_wickets
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00

# In[125]:
#the dataframe consists of the economy of the top allrounders in ascending order.
top_allrounders_economy



```
#the dataframe consists of the economy of the top allrounders in ascending order.

top_allrounders_economy
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Wickets	Bowling_average	Economy	Bowling_Strike_Rate
232	Sunil Narine	Kolkata	Overseas	134.0	954.0	15.64	161.69	143.0	24.53	6.74	21.83
223	Moeen Ali	Chennai	Overseas	34.0	666.0	22.97	146.37	16.0	29.31	6.85	25.69
202	Aman Khan	Kolkata	Indian	5.0	40.0	13.30	148.10	0.0	0.00	7.00	0.00
165	Jofra Archer	Mumbai	Overseas	35.0	195.0	15.00	157.26	46.0	21.33	7.13	17.93
195	Mohammad Nabi	Kolkata	Overseas	17.0	180.0	15.00	151.26	13.0	31.38	7.14	26.38
154	K Gowtham	Lucknow	Indian	24.0	186.0	14.31	169.09	13.0	43.23	8.26	31.38
218	Glen Maxwell	Bangalore	Overseas	97.0	2018.0	25.23	151.84	22.0	41.59	8.55	29.18
233	Kieron Pollard	Mumbai	Overseas	178.0	3268.0	29.98	149.77	65.0	31.62	8.78	21.60
204	Andre Russell	Kolkata	Overseas	84.0	1700.0	29.31	178.57	72.0	26.40	9.05	17.51
211	Hardik Pandya	Gujarat	Indian	92.0	1476.0	27.33	153.91	42.0	31.26	9.07	20.69

# In[126]:
#From the above analysis, if we rank down the allrounders from 1-10 on various parameters. The top allrounder options are as follows:

#1. Andre Russell

#2. Sunil Narine

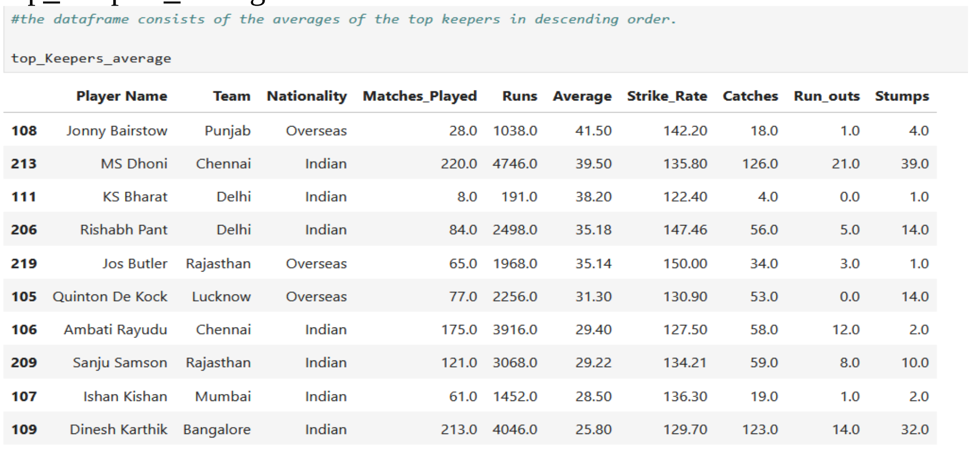
#3. Hardik Pandya

#4. Jofra Arche

# In[127]:
#Analyzing the Keepers Data

#we have narrowed our analysis down to keepers averaging more than 25.0

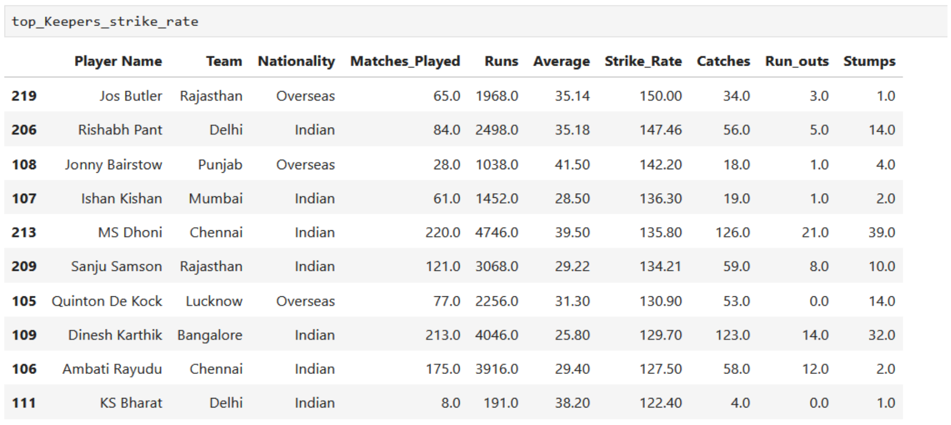
```
top_keepers=Capped_Keepers.loc[(Capped_Keepers ["Average"] >= 25.0)]

#Sorting the data in descending order- respect to each parameter.
top_Keepers_average=top_keepers.sort_values('Average', ascending=False)

top_Keepers_strike_rate = top_keepers.sort_values('Strike_Rate', ascending=False)

top_Keepers_runs = top_keepers.sort_values('Runs', ascending=False)

top_Keepers_matches = top_keepers.sort_values('Matches_Played', ascending=False)

top_Keepers_catches= top_keepers.sort_values('Catches',ascending=False)

top_Keepers_runouts=top_keepers.sort_values('Run_outs', ascending=False)
top_Keepers_stumps=top_keepers.sort_values('Stumps', ascending=False)

# In[128]:
#the dataframe consists of the averages of the top keepers in descending order.
top_Keepers_average
```



```
#the dataframe consists of the averages of the top keepers in descending order.

top_Keepers_average
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.50	142.20	18.0	1.0	4.0
213	MS Dhoni	Chennai	Indian	220.0	4746.0	39.50	135.80	126.0	21.0	39.0
111	KS Bharat	Delhi	Indian	8.0	191.0	38.20	122.40	4.0	0.0	1.0
206	Rishabh Pant	Delhi	Indian	84.0	2498.0	35.18	147.46	56.0	5.0	14.0
219	Jos Butler	Rajasthan	Overseas	65.0	1968.0	35.14	150.00	34.0	3.0	1.0
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.30	130.90	53.0	0.0	14.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.40	127.50	58.0	12.0	2.0
209	Sanju Samson	Rajasthan	Indian	121.0	3068.0	29.22	134.21	59.0	8.0	10.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.50	136.30	19.0	1.0	2.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.80	129.70	123.0	14.0	32.0

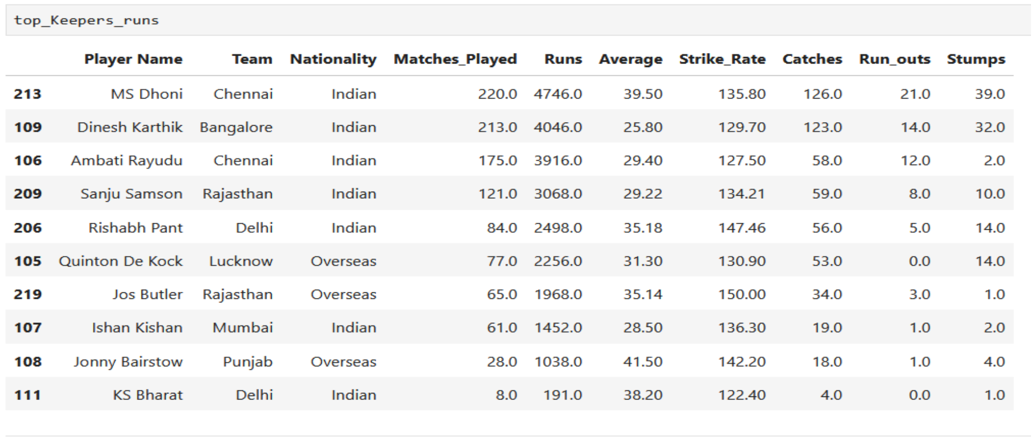
```
# In[129]:
top_Keepers_strike_rate
```



```
top_Keepers_strike_rate
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
219	Jos Butler	Rajasthan	Overseas	65.0	1968.0	35.14	150.00	34.0	3.0	1.0
206	Rishabh Pant	Delhi	Indian	84.0	2498.0	35.18	147.46	56.0	5.0	14.0
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.50	142.20	18.0	1.0	4.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.50	136.30	19.0	1.0	2.0
213	MS Dhoni	Chennai	Indian	220.0	4746.0	39.50	135.80	126.0	21.0	39.0
209	Sanju Samson	Rajasthan	Indian	121.0	3068.0	29.22	134.21	59.0	8.0	10.0
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.30	130.90	53.0	0.0	14.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.80	129.70	123.0	14.0	32.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.40	127.50	58.0	12.0	2.0
111	KS Bharat	Delhi	Indian	8.0	191.0	38.20	122.40	4.0	0.0	1.0

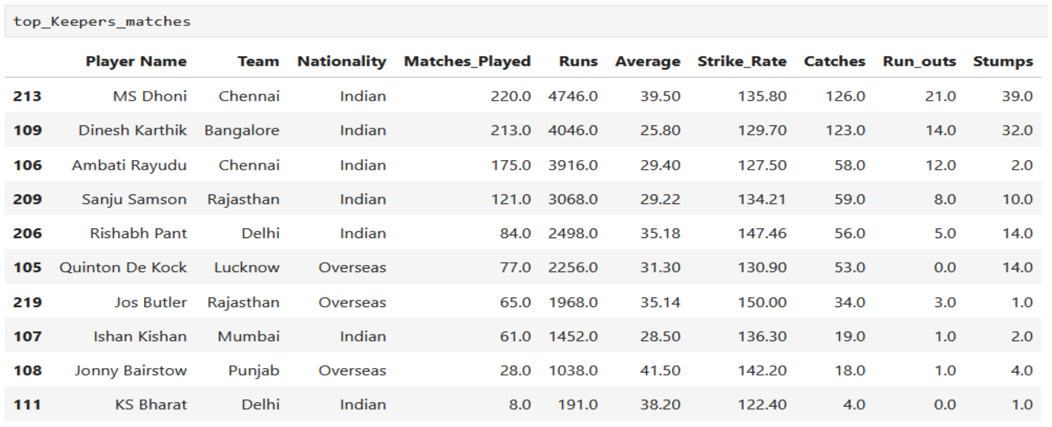
# In[130]:
top_Keepers_runs



```
top_Keepers_runs
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
213	MS Dhoni	Chennai	Indian	220.0	4746.0	39.50	135.80	126.0	21.0	39.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.80	129.70	123.0	14.0	32.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.40	127.50	58.0	12.0	2.0
209	Sanju Samson	Rajasthan	Indian	121.0	3068.0	29.22	134.21	59.0	8.0	10.0
206	Rishabh Pant	Delhi	Indian	84.0	2498.0	35.18	147.46	56.0	5.0	14.0
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.30	130.90	53.0	0.0	14.0
219	Jos Butler	Rajasthan	Overseas	65.0	1968.0	35.14	150.00	34.0	3.0	1.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.50	136.30	19.0	1.0	2.0
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.50	142.20	18.0	1.0	4.0
111	KS Bharat	Delhi	Indian	8.0	191.0	38.20	122.40	4.0	0.0	1.0

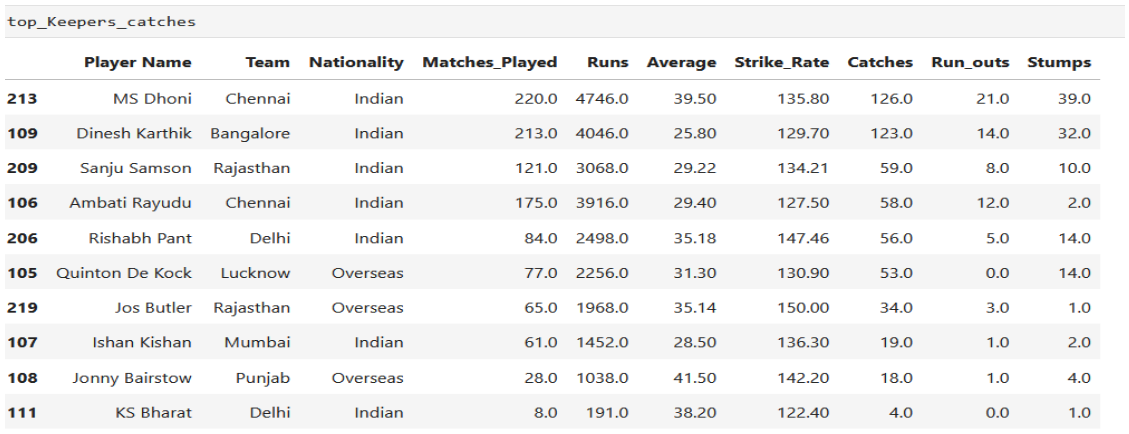
# In[131]:
top_Keepers_matches



```
top_Keepers_matches
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
213	MS Dhoni	Chennai	Indian	220.0	4746.0	39.50	135.80	126.0	21.0	39.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.80	129.70	123.0	14.0	32.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.40	127.50	58.0	12.0	2.0
209	Sanju Samson	Rajasthan	Indian	121.0	3068.0	29.22	134.21	59.0	8.0	10.0
206	Rishabh Pant	Delhi	Indian	84.0	2498.0	35.18	147.46	56.0	5.0	14.0
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.30	130.90	53.0	0.0	14.0
219	Jos Butler	Rajasthan	Overseas	65.0	1968.0	35.14	150.00	34.0	3.0	1.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.50	136.30	19.0	1.0	2.0
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.50	142.20	18.0	1.0	4.0
111	KS Bharat	Delhi	Indian	8.0	191.0	38.20	122.40	4.0	0.0	1.0

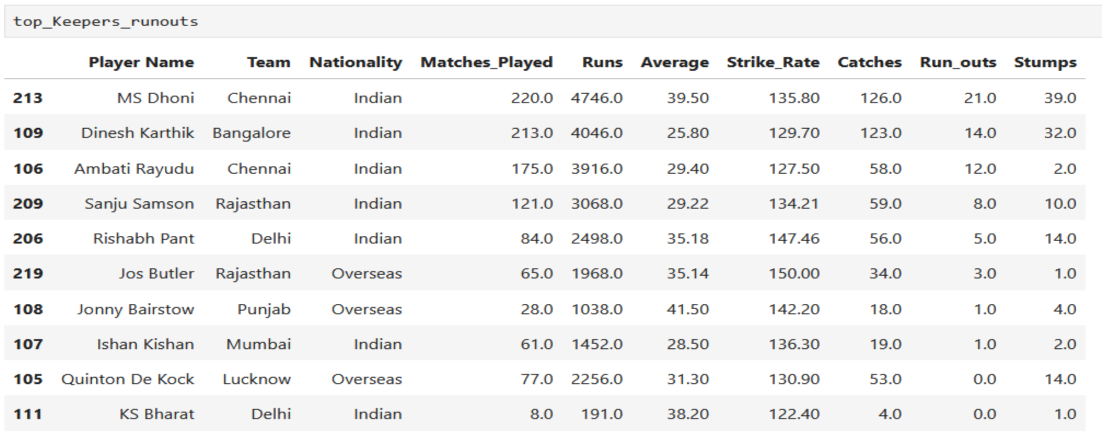
# In[132]:
top_Keepers_catches



```
top_Keepers_catches
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
213	MS Dhoni	Chennai	Indian	220.0	4746.0	39.50	135.80	126.0	21.0	39.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.80	129.70	123.0	14.0	32.0
209	Sanju Samson	Rajasthan	Indian	121.0	3068.0	29.22	134.21	59.0	8.0	10.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.40	127.50	58.0	12.0	2.0
206	Rishabh Pant	Delhi	Indian	84.0	2498.0	35.18	147.46	56.0	5.0	14.0
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.30	130.90	53.0	0.0	14.0
219	Jos Butler	Rajasthan	Overseas	65.0	1968.0	35.14	150.00	34.0	3.0	1.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.50	136.30	19.0	1.0	2.0
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.50	142.20	18.0	1.0	4.0
111	KS Bharat	Delhi	Indian	8.0	191.0	38.20	122.40	4.0	0.0	1.0

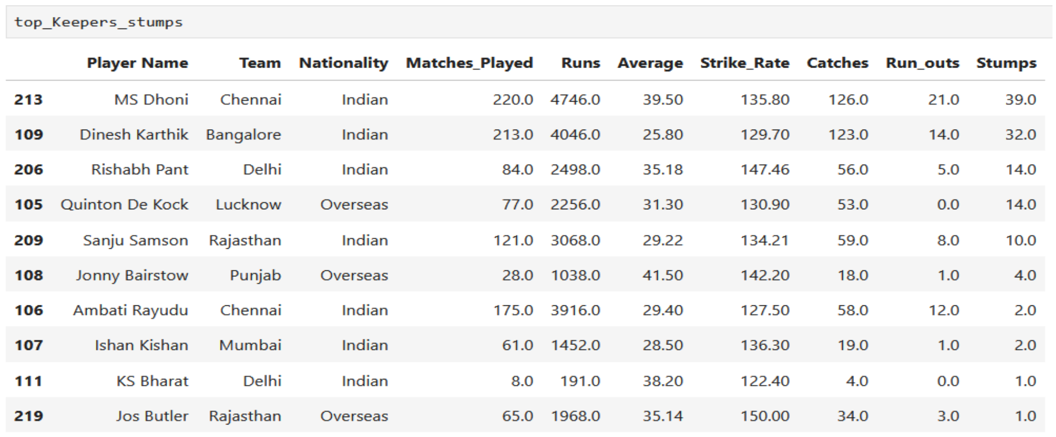
# In[133]:
top_Keepers_runouts



```
top_Keepers_runouts
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
213	MS Dhoni	Chennai	Indian	220.0	4746.0	39.50	135.80	126.0	21.0	39.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.80	129.70	123.0	14.0	32.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.40	127.50	58.0	12.0	2.0
209	Sanju Samson	Rajasthan	Indian	121.0	3068.0	29.22	134.21	59.0	8.0	10.0
206	Rishabh Pant	Delhi	Indian	84.0	2498.0	35.18	147.46	56.0	5.0	14.0
219	Jos Butler	Rajasthan	Overseas	65.0	1968.0	35.14	150.00	34.0	3.0	1.0
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.50	142.20	18.0	1.0	4.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.50	136.30	19.0	1.0	2.0
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.30	130.90	53.0	0.0	14.0
111	KS Bharat	Delhi	Indian	8.0	191.0	38.20	122.40	4.0	0.0	1.0

# In[134]:
top_Keepers_stumps



```
top_Keepers_stumps
```

	Player Name	Team	Nationality	Matches_Played	Runs	Average	Strike_Rate	Catches	Run_outs	Stumps
213	MS Dhoni	Chennai	Indian	220.0	4746.0	39.50	135.80	126.0	21.0	39.0
109	Dinesh Karthik	Bangalore	Indian	213.0	4046.0	25.80	129.70	123.0	14.0	32.0
206	Rishabh Pant	Delhi	Indian	84.0	2498.0	35.18	147.46	56.0	5.0	14.0
105	Quinton De Kock	Lucknow	Overseas	77.0	2256.0	31.30	130.90	53.0	0.0	14.0
209	Sanju Samson	Rajasthan	Indian	121.0	3068.0	29.22	134.21	59.0	8.0	10.0
108	Jonny Bairstow	Punjab	Overseas	28.0	1038.0	41.50	142.20	18.0	1.0	4.0
106	Ambati Rayudu	Chennai	Indian	175.0	3916.0	29.40	127.50	58.0	12.0	2.0
107	Ishan Kishan	Mumbai	Indian	61.0	1452.0	28.50	136.30	19.0	1.0	2.0
111	KS Bharat	Delhi	Indian	8.0	191.0	38.20	122.40	4.0	0.0	1.0
219	Jos Butler	Rajasthan	Overseas	65.0	1968.0	35.14	150.00	34.0	3.0	1.0

# In[135]:
###If we rank the keepers in order of 1-10 on the above parameters. The top 3 keepers will be

##1. MS Dhoni

##2. Dinesh Karthik

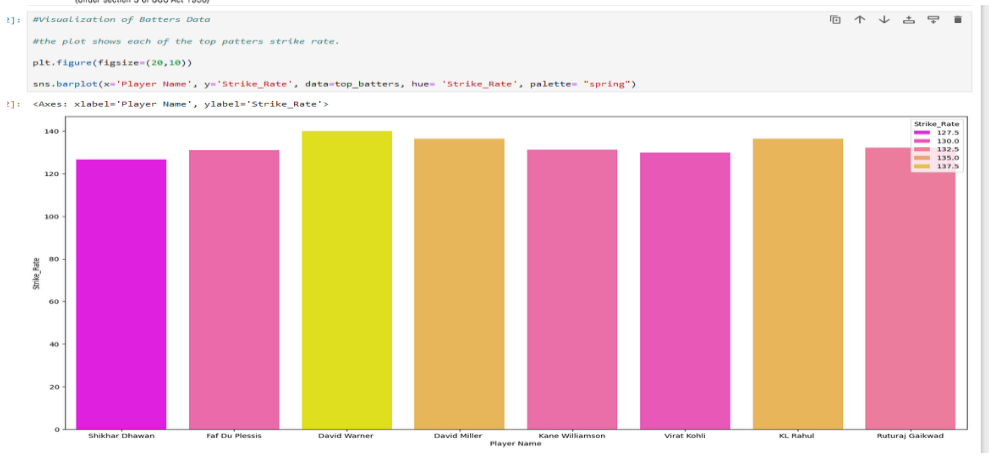
#33. Rishabh Pant

# In[136]:
#Visualization of Batters Data

#the plot shows each of the top patters strike rate.

plt.figure(figsize=(20,10))

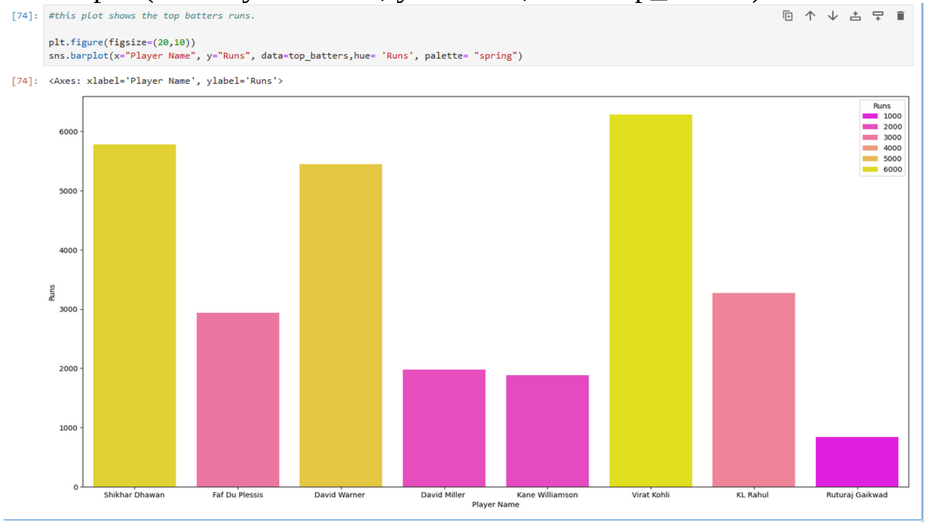
sns.barplot(x='Player Name', y='Strike_Rate', data=top_batters)



```
[ ]: #Visualization of Batters Data

     #the plot shows each of the top patters strike rate.

     plt.figure(figsize=(20,10))

     sns.barplot(x='Player Name', y='Strike_Rate', data=top_batters, hue= 'Strike_Rate', palette= "spring")

[ ]: <Axes: xlabel='Player Name', ylabel='Strike_Rate'>
```

# In[137]:
#this plot shows the top batters runs.

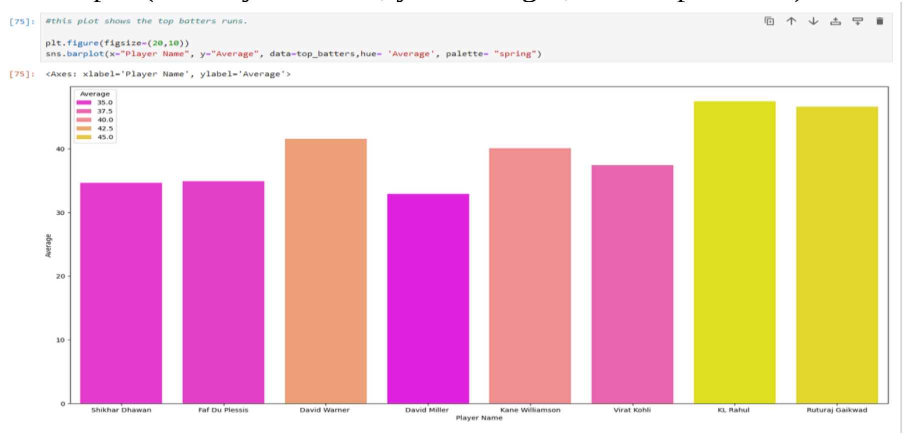
plt.figure(figsize=(20,10))
sns.barplot(x="Player Name", y="Runs", data=top_batters)



```
[74]: #this plot shows the top batters runs.

      plt.figure(figsize=(20,10))
      sns.barplot(x="Player Name", y="Runs", data=top_batters,hue= 'Runs', palette= "spring")

[74]: <Axes: xlabel='Player Name', ylabel='Runs'>
```

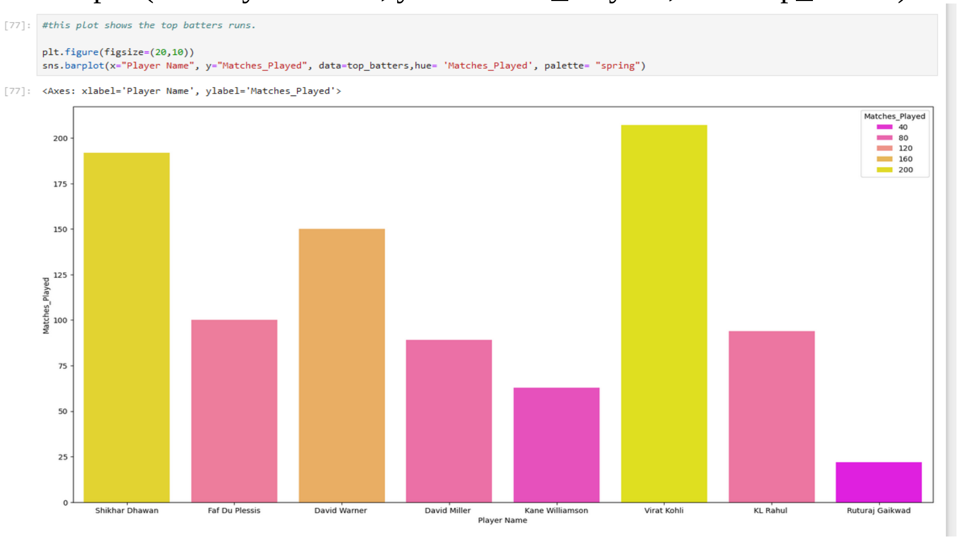
# In[138]:
#this plot shows the top batters runs.

plt.figure(figsize=(20,10))
sns.barplot(x="Player Name", y="Average", data=top_batters)



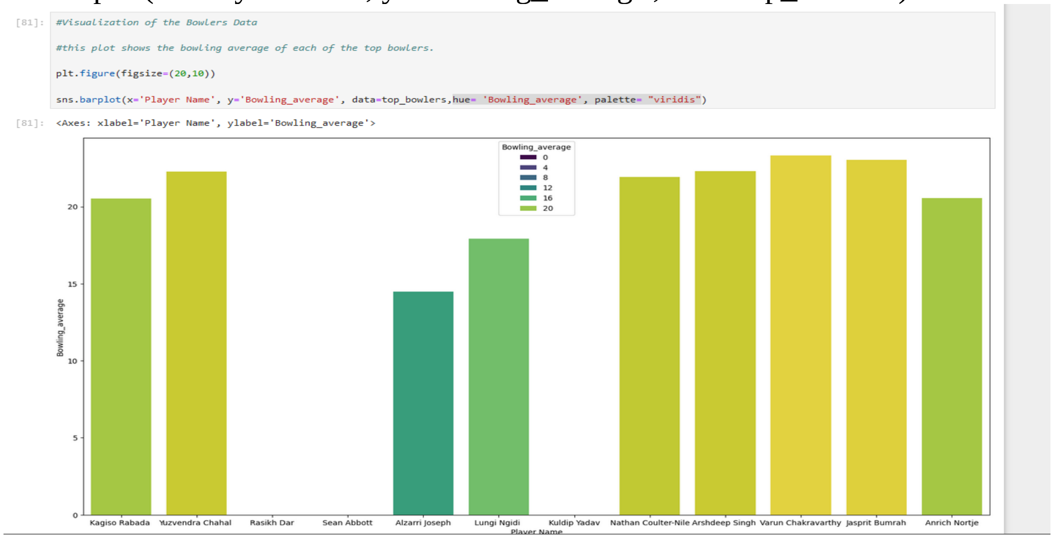
```
[75]: #this plot shows the top batters runs.

      plt.figure(figsize=(20,10))
      sns.barplot(x="Player Name", y="Average", data=top_batters,hue= 'Average', palette= "spring")

[75]: <Axes: xlabel='Player Name', ylabel='Average'>
```

```
# In[140]:
#this plot shows the top batters runs.
plt.figure(figsize=(20,10))
sns.barplot(x="Player Name", y="Matches_Played", data=top_batters)
```



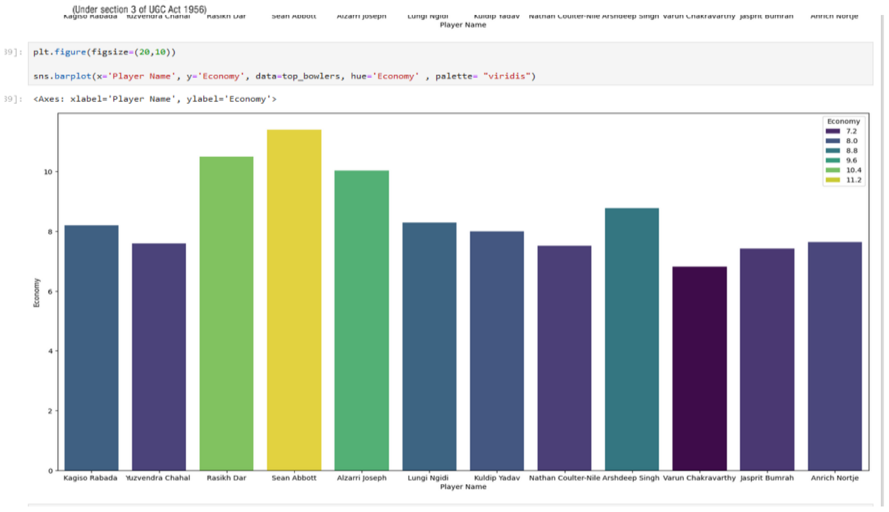
```
[77]: #this plot shows the top batters runs.

      plt.figure(figsize=(20,10))
      sns.barplot(x="Player Name", y="Matches_Played", data=top_batters,hue= 'Matches_Played', palette= "spring")

[77]: <Axes: xlabel='Player Name', ylabel='Matches_Played'>
```

```
# In[141]:
#Visualization of the Bowlers Data

#this plot shows the bowling average of each of the top bowlers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Bowling_average', data=top_bowlers)
```



```
[81]: #Visualization of the Bowlers Data

      #this plot shows the bowling average of each of the top bowlers.

      plt.figure(figsize=(20,10))

      sns.barplot(x='Player Name', y='Bowling_average', data=top_bowlers,hue= 'Bowling_average', palette= "viridis")

[81]: <Axes: xlabel='Player Name', ylabel='Bowling_average'>
```

```
# In[142]:
plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Economy', data=top_bowlers)
```

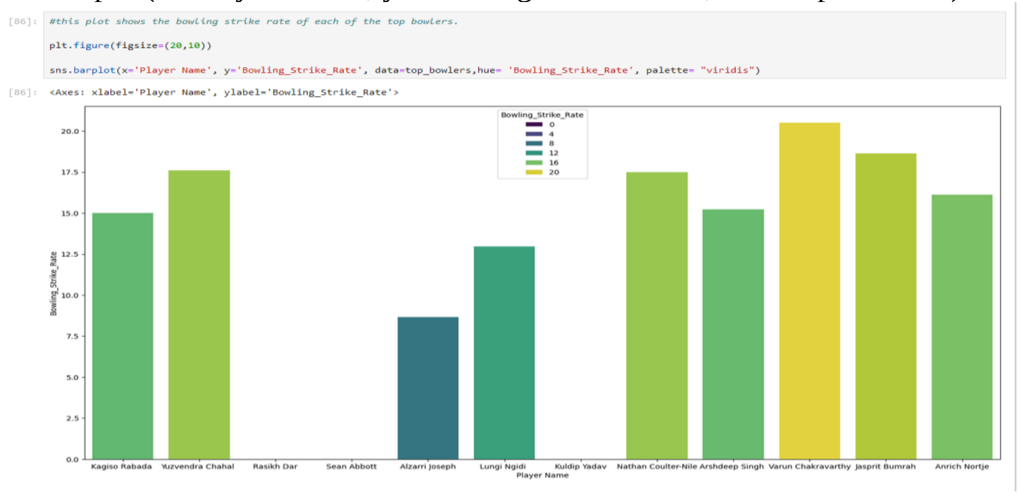


```
39]: plt.figure(figsize=(20,10))

     sns.barplot(x='Player Name', y='Economy', data=top_bowlers, hue='Economy' , palette= "viridis")

39]: <Axes: xlabel='Player Name', ylabel='Economy'>
```

# In[143]:
#this plot shows the bowling strike rate of each of the top bowlers.

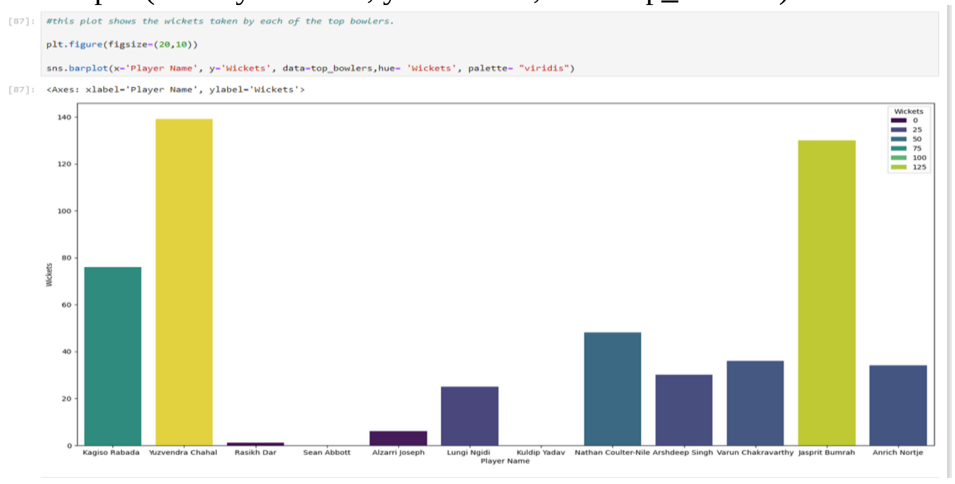
plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Bowling_Strike_Rate', data=top_bowlers)



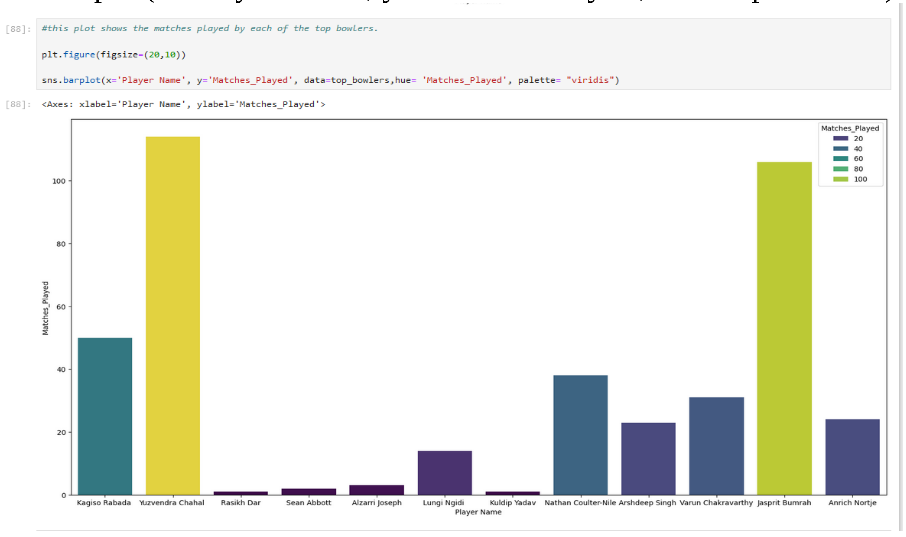
```
[86]: #this plot shows the bowling strike rate of each of the top bowlers.

      plt.figure(figsize=(20,10))

      sns.barplot(x='Player Name', y='Bowling_Strike_Rate', data=top_bowlers,hue= 'Bowling_Strike_Rate', palette= "viridis")

[86]: <Axes: xlabel='Player Name', ylabel='Bowling_Strike_Rate'>
```

# In[144]:
#this plot shows the wickets taken by each of the top bowlers.
plt.figure(figsize=(20,10))
sns.barplot(x='Player Name', y='Wickets', data=top_bowlers)



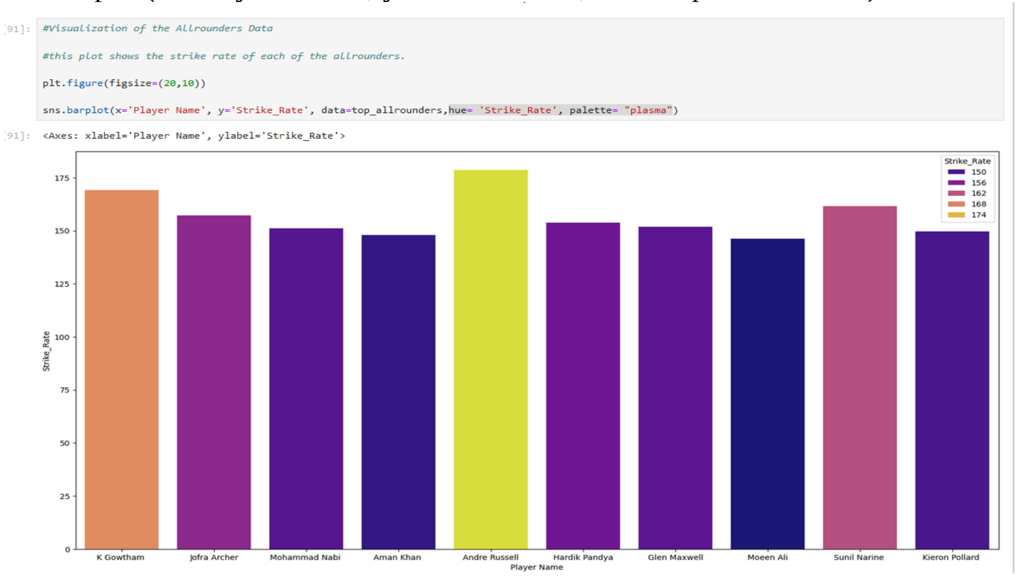
```
[87]: #this plot shows the wickets taken by each of the top bowlers.

      plt.figure(figsize=(20,10))

      sns.barplot(x='Player Name', y='Wickets', data=top_bowlers,hue= 'Wickets', palette= "viridis")

[87]: <Axes: xlabel='Player Name', ylabel='Wickets'>
```

```
# In[145]:
#this plot shows the matches played by each of the top bowlers.
plt.figure(figsize=(20,10))
sns.barplot(x='Player Name', y='Matches_Played', data=top_bowlers)
```



```
[88]: #this plot shows the matches played by each of the top bowlers.

      plt.figure(figsize=(20,10))

      sns.barplot(x='Player Name', y='Matches_Played', data=top_bowlers,hue= 'Matches_Played', palette= "viridis")

[88]: <Axes: xlabel='Player Name', ylabel='Matches_Played'>
```

```
# In[146]:
#Visualization of the Allrounders Data

#this plot shows the strike rate of each of the allrounders.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Strike_Rate', data=top_allrounders)
```

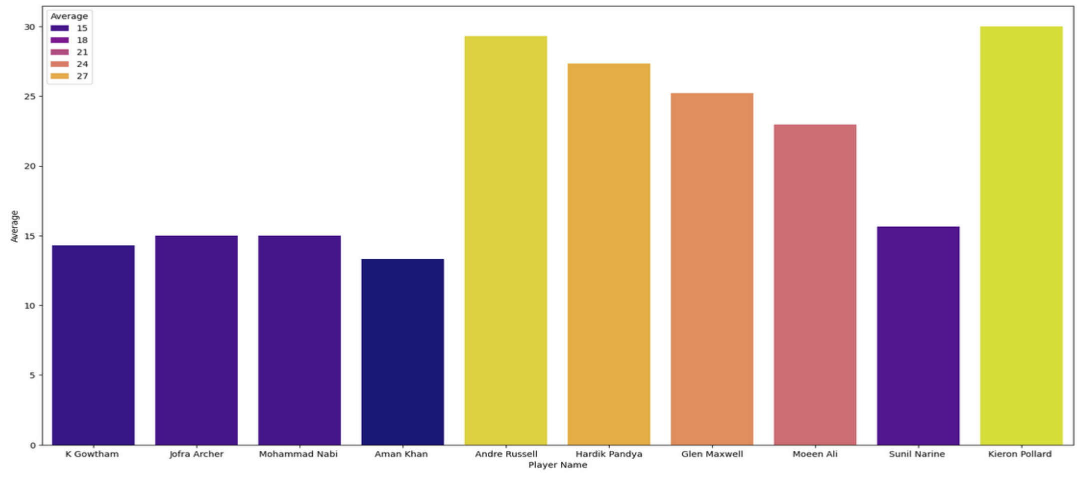


```
[91]: #Visualization of the Allrounders Data

      #this plot shows the strike rate of each of the allrounders.

      plt.figure(figsize=(20,10))

      sns.barplot(x='Player Name', y='Strike_Rate', data=top_allrounders,hue= 'Strike_Rate', palette= "plasma")

[91]: <Axes: xlabel='Player Name', ylabel='Strike_Rate'>
```

```
# In[147]:
#this plot shows the bowling average of each of the allrounders.
plt.figure(figsize=(20,10))
sns.barplot(x='Player Name', y='Average', data=top_allrounders)
```

```
#this plot shows the bowling average of each of the allrounders.
plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Average', data=top_allrounders,hue= 'Average', palette= "plasma")
```

```
<Axes: xlabel='Player Name', ylabel='Average'>
```



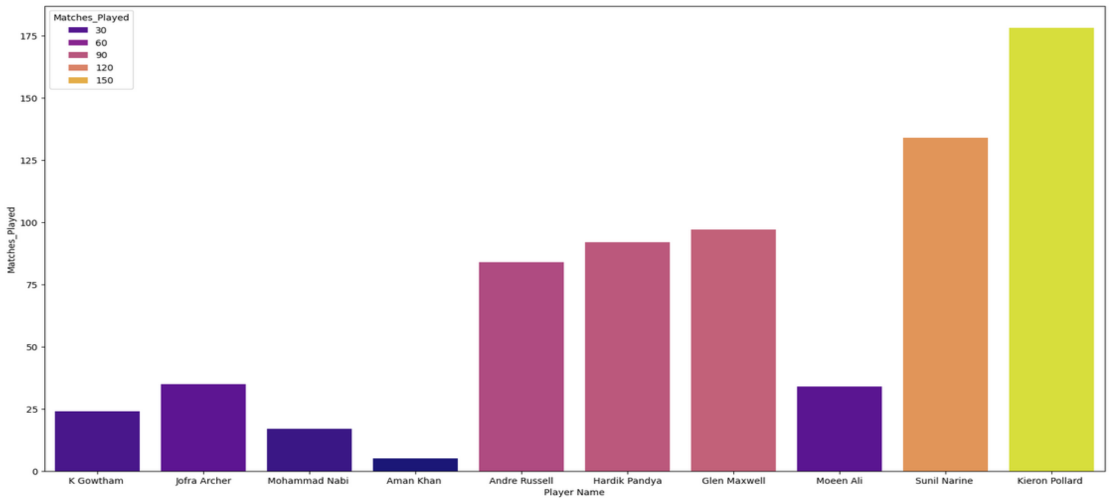
# In[148]:

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Matches_Played', data=top_allrounders)

```
plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Matches_Played', data=top_allrounders,hue= 'Matches_Played', palette= "plasma")
```

```
<Axes: xlabel='Player Name', ylabel='Matches_Played'>
```

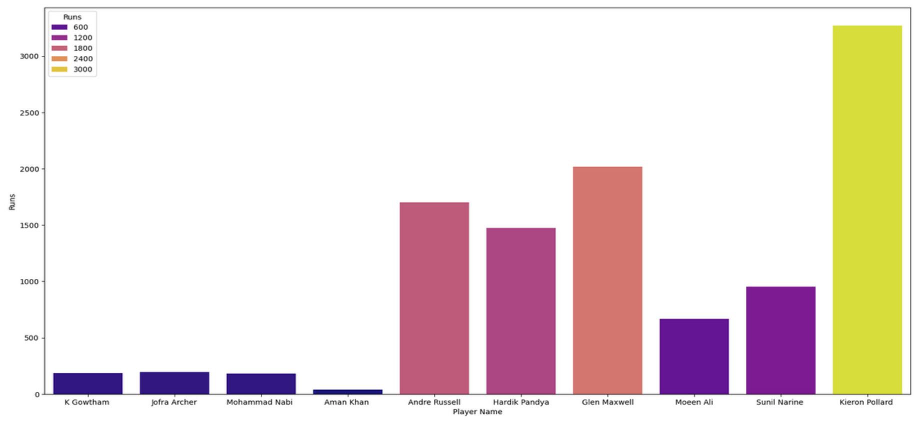


# In[149]:
#this plot shows the top allrounder's runs.

plt.figure(figsize=(20,10))

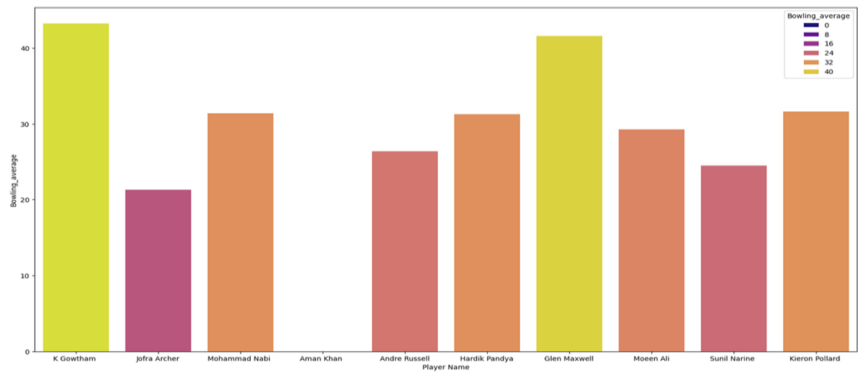
sns.barplot(x='Player Name', y='Runs', data=top_allrounders)

```
]: #this plot shows the top allrounder's runs.
   plt.figure(figsize=(20,10))

   sns.barplot(x='Player Name', y='Runs', data=top_allrounders,hue= 'Runs', palette= "plasma")

]: <Axes: xlabel='Player Name', ylabel='Runs'>
```



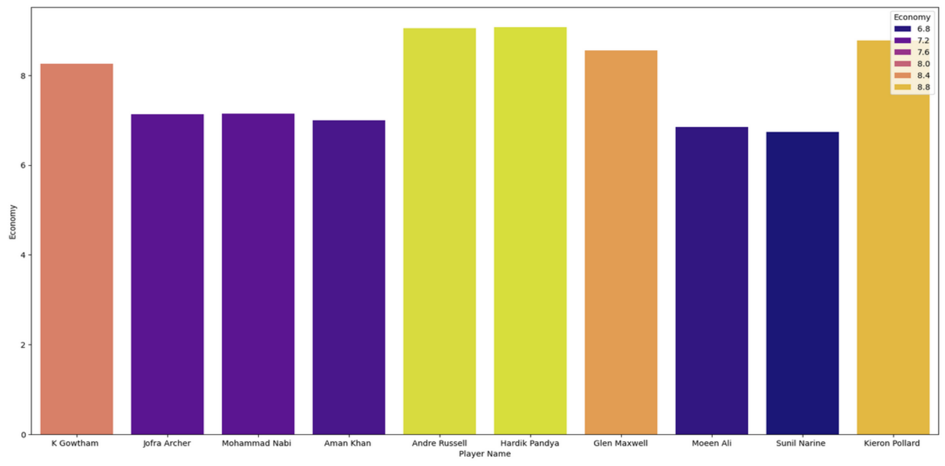
# In[150]:
plt.figure(figsize=(20,10))
sns.barplot(x='Player Name', y='Bowling_average', data=top_allrounders)

```
]: plt.figure(figsize=(20,10))

   sns.barplot(x='Player Name', y='Bowling_average', data=top_allrounders,hue= 'Bowling_average', palette= "plasma")

]: <Axes: xlabel='Player Name', ylabel='Bowling_average'>
```

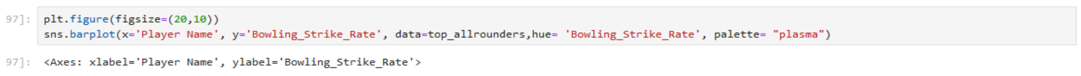


# In[151]:
plt.figure(figsize=(20,10))
sns.barplot(x='Player Name', y='Economy', data=top_allrounders)

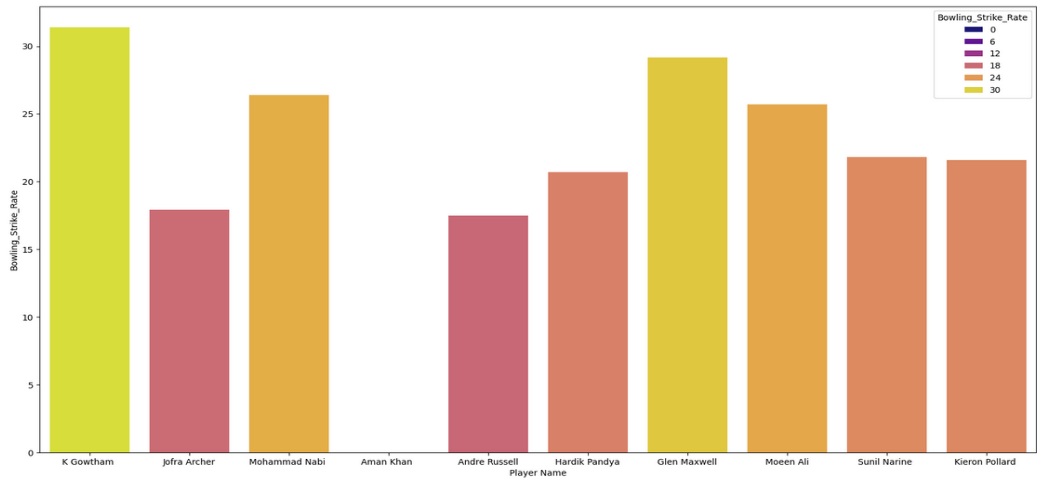
```
]: plt.figure(figsize=(20,10))

   sns.barplot(x='Player Name', y='Economy', data=top_allrounders, hue= 'Economy', palette= "plasma")

]: <Axes: xlabel='Player Name', ylabel='Economy'>
```



# In[152]:
plt.figure(figsize=(20,10))
sns.barplot(x='Player Name', y='Bowling_Strike_Rate', data=top_allrounders)

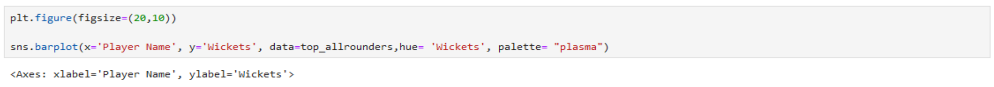


```
97]:  plt.figure(figsize=(20,10))
      sns.barplot(x='Player Name', y='Bowling_Strike_Rate', data=top_allrounders,hue= 'Bowling_Strike_Rate', palette= "plasma")

97]:  <Axes: xlabel='Player Name', ylabel='Bowling_Strike_Rate'>
```

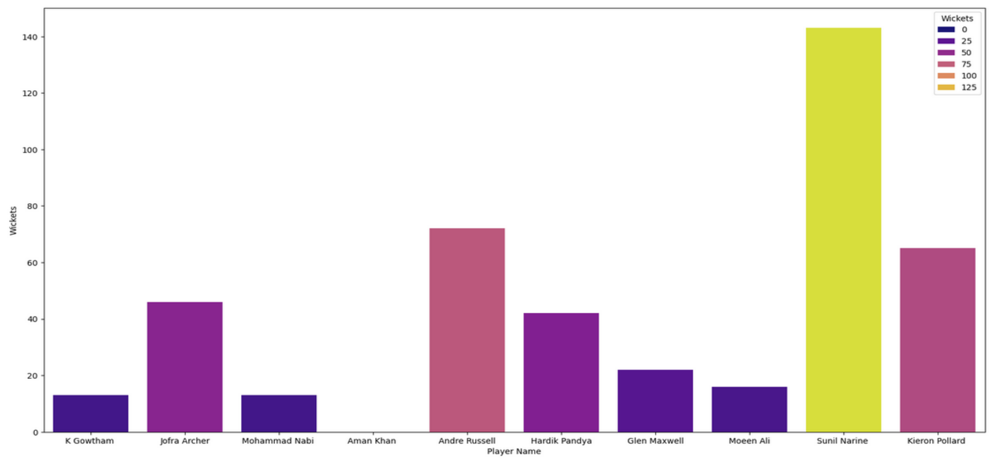


# In[153]:
plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Wickets', data=top_allrounders)



```
plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Wickets', data=top_allrounders,hue= 'Wickets', palette= "plasma")

<Axes: xlabel='Player Name', ylabel='Wickets'>
```



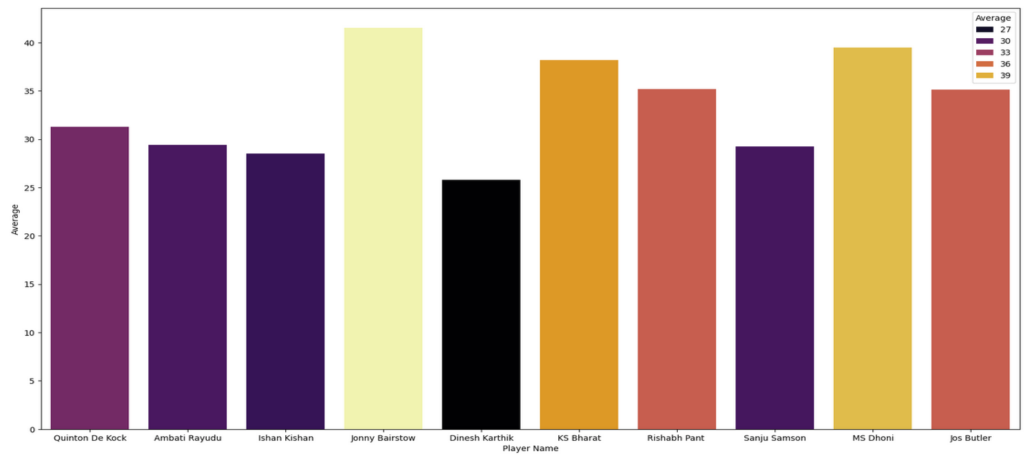
# In[154]:
#Visualization of the Keepers Data

#this plot shows the average of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Average', data=top_keepers)

```
67]: #Visualization of the Keepers Data

     #this plot shows the average of top keepers.

     plt.figure(figsize=(20,10))

     sns.barplot(x='Player Name', y='Average', data=top_keepers, hue= 'Average', palette= "inferno")

67]: <Axes: xlabel='Player Name', ylabel='Average'>
```



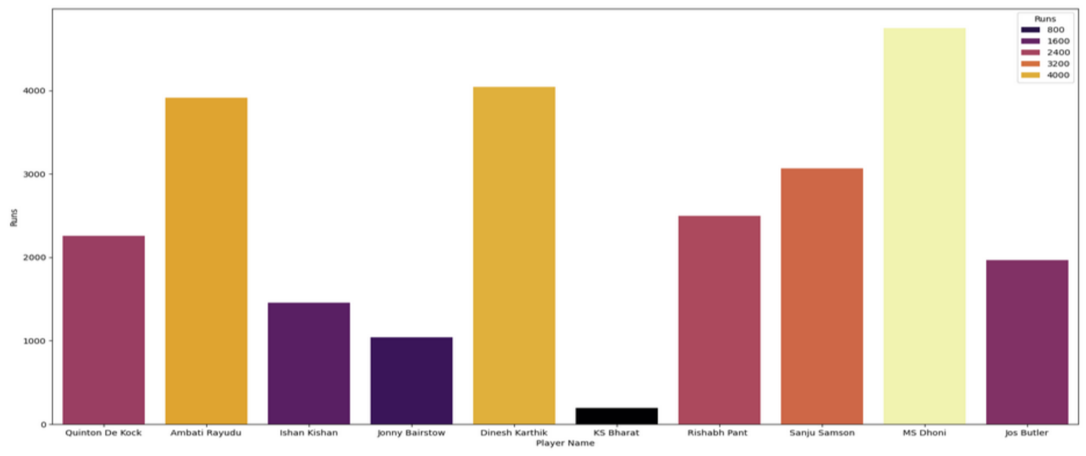
# In[155]:
#Visualization of the Keepers Data

#this plot shows the runs of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Runs', data=top_keepers)

```
: #Visualization of the Keepers Data

  #this plot shows the runs of top keepers.

  plt.figure(figsize=(20,10))

  sns.barplot(x='Player Name', y='Runs', data=top_keepers,hue= 'Runs', palette= "inferno")

: <Axes: xlabel='Player Name', ylabel='Runs'>
```



# In[156]:
#Visualization of the Keepers Data

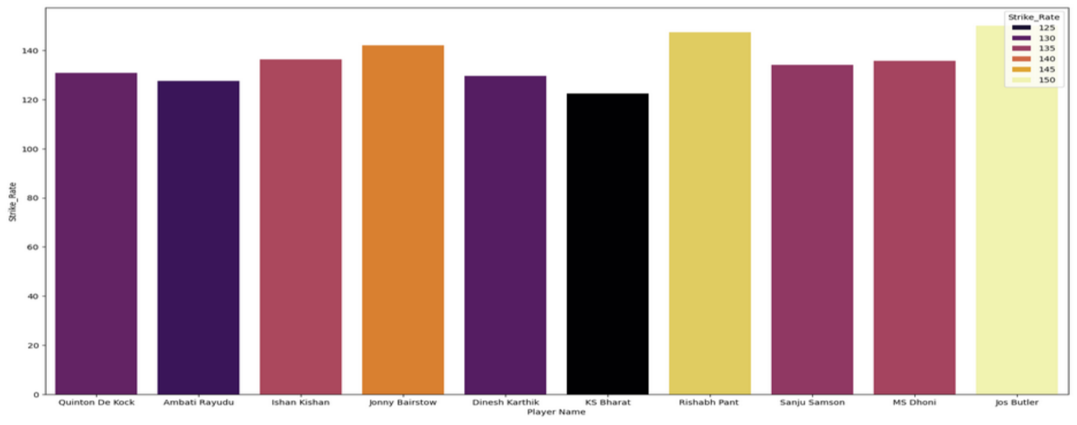
#this plot shows the average of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Strike_Rate', data=top_keepers)

```
#Visualization of the Keepers Data

#this plot shows the average of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Strike_Rate', data=top_keepers,hue= 'Strike_Rate', palette= "inferno")
```

```
<Axes: xlabel='Player Name', ylabel='Strike_Rate'>
```



# In[157]:

#Visualization of the Keepers Data

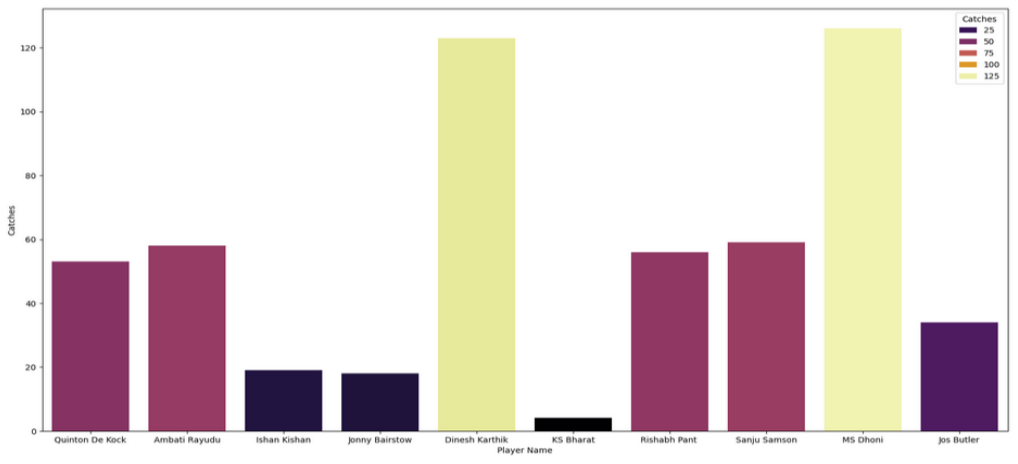
#this plot shows the catches of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Catches', data=top_keepers)

```
#Visualization of the Keepers Data

#this plot shows the catches of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Catches', data=top_keepers,hue= 'Catches', palette= "inferno")
```

```
<Axes: xlabel='Player Name', ylabel='Catches'>
```



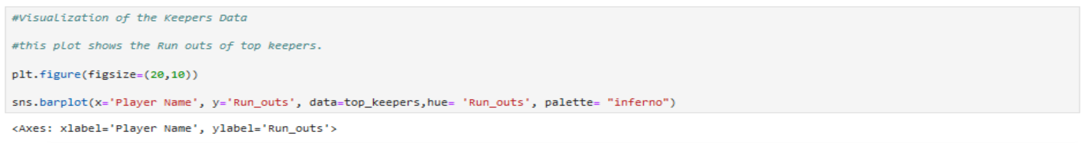
# In[158]:

#Visualization of the Keepers Data

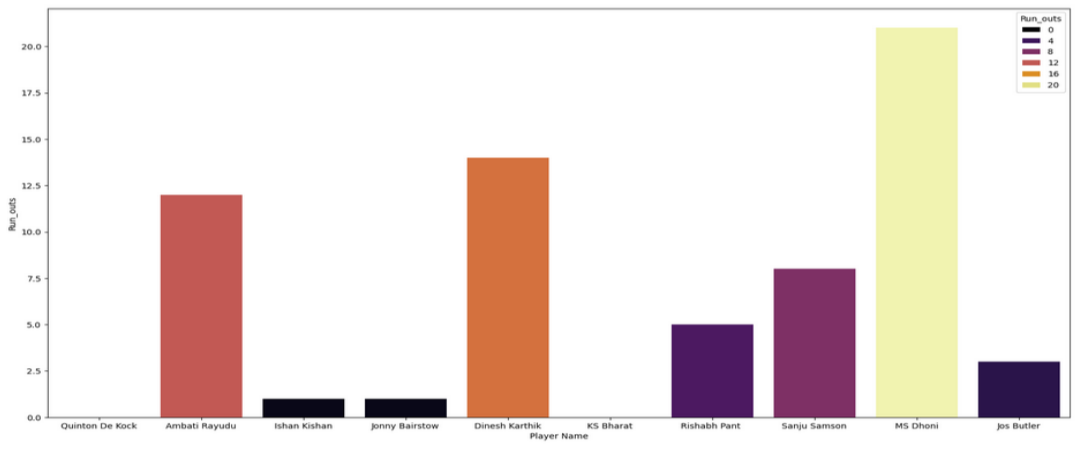
#this plot shows the Run outs of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Run_outs', data=top_keepers)



```
#Visualization of the Keepers Data

#this plot shows the Run outs of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Run_outs', data=top_keepers,hue= 'Run_outs', palette= "inferno")

<Axes: xlabel='Player Name', ylabel='Run_outs'>
```

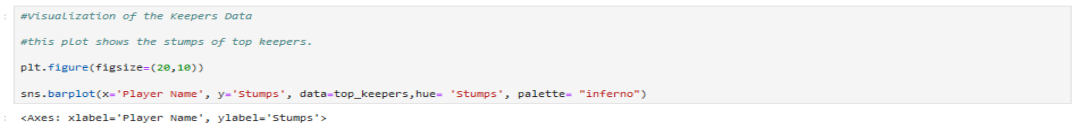


# In[159]:
#Visualization of the Keepers Data

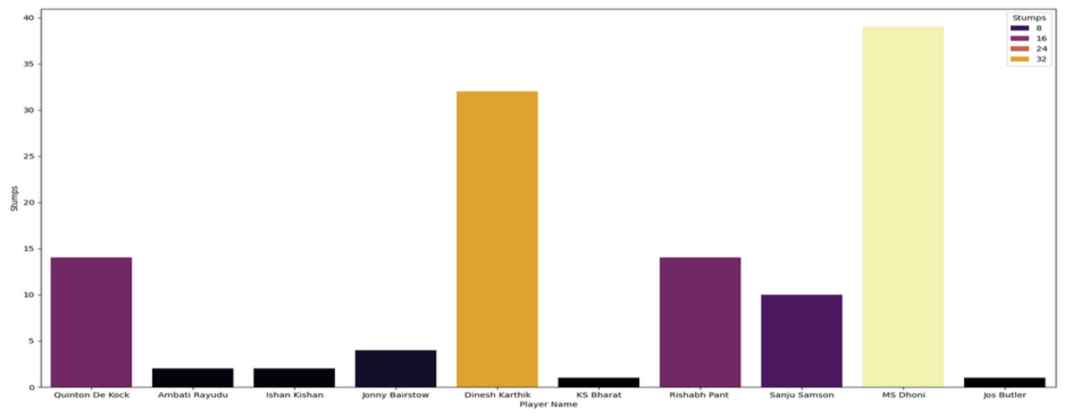
#this plot shows the stumps of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Stumps', data=top_keepers)



```
#Visualization of the Keepers Data

#this plot shows the stumps of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Stumps', data=top_keepers,hue= 'Stumps', palette= "inferno")

<Axes: xlabel='Player Name', ylabel='Stumps'>
```



# In[160]:
#Visualization of the Keepers Data

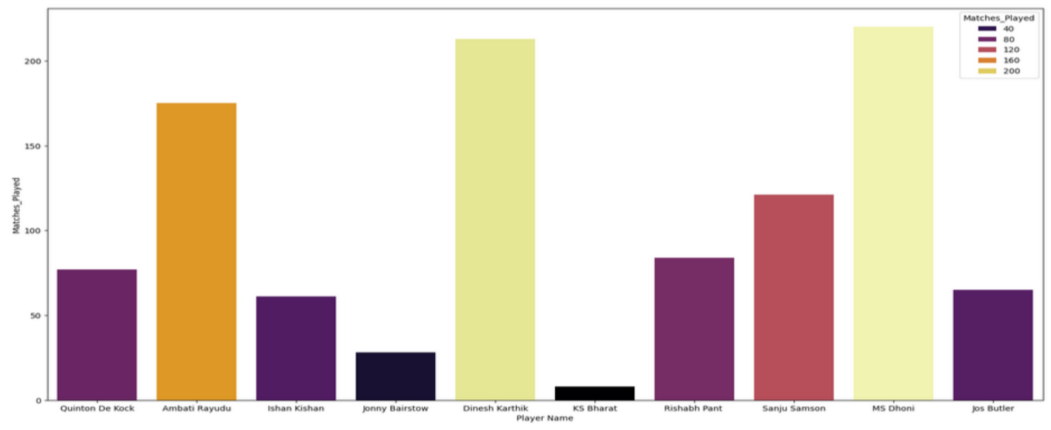
#this plot shows the matches played of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Matches_Played', data=top_keepers)

```
#Visualization of the Keepers Data

#this plot shows the matches played of top keepers.

plt.figure(figsize=(20,10))

sns.barplot(x='Player Name', y='Matches_Played', data=top_keepers,hue= 'Matches_Played', palette= "inferno")
```

```
<Axes: xlabel='Player Name', ylabel='Matches_Played'>
```



# In[161]:
#Forming Our Best 11 for the Campaign based on the above analysis

##1. We will consider the number of players from each category that the 120 world cup winning and the last year's IPL winning team played in
#their Final matches.
#2.The Australia squad consisted of-3 Batters, 3 Allrounders, 4 Bowlers with 1 spin option and 1 wicket keeper.
#3.The Chennai Squad Consisted of-4 Batters, 3 Allrounders, 3 Bowlers and 1 Wicket Keeper.
#4.For our final analysis we will consider the ratio of players in the best 11 as follows:

#1.3 Batters

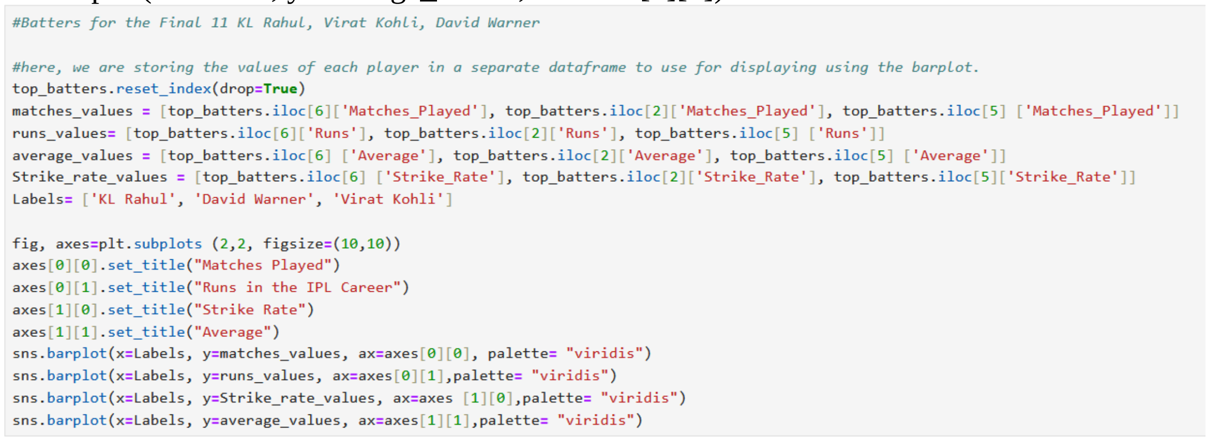
#2. 3 Allrounders

#3. 4 Bowlers with 2 Spin Options

#4.1 Wicket Keeper
# In[162]:
#Batters for the Final 11 KL Rahul, Virat Kohli, David Warner

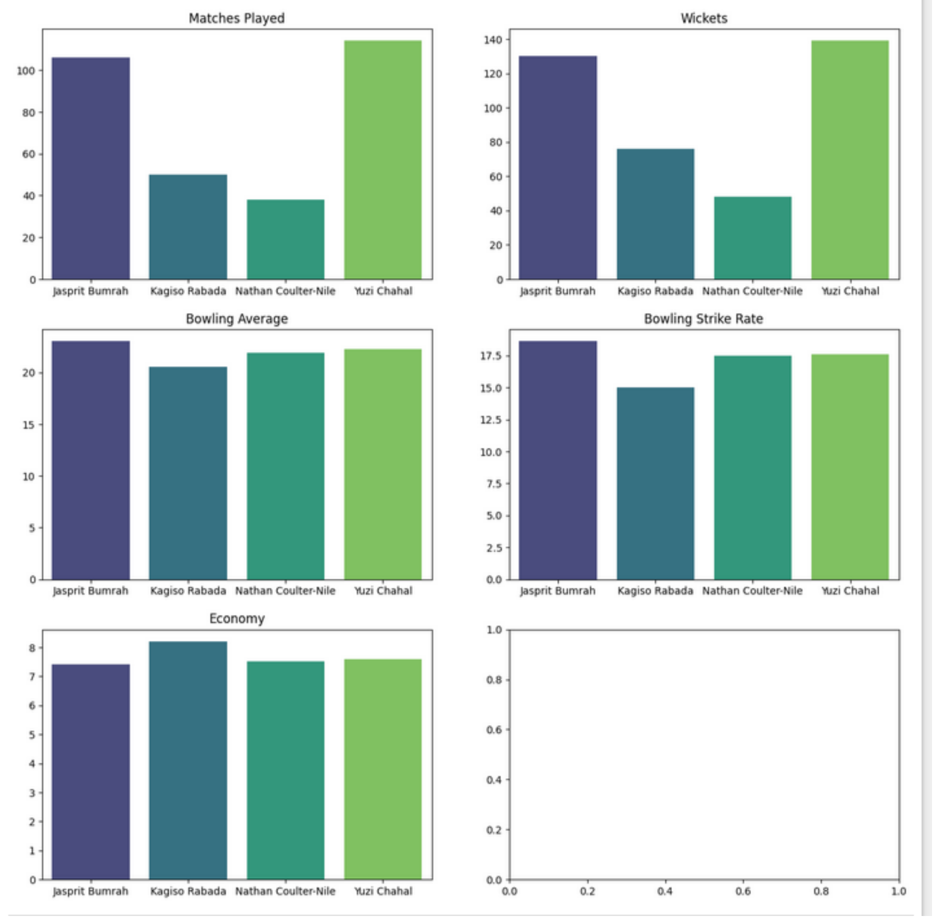
#here, we are storing the values of each player in a separate dataframe to use for displaying using the barplot.
top_batters.reset_index(drop=True)
matches_values = [top_batters.iloc[6]['Matches_Played'], top_batters.iloc[2]['Matches_Played'], top_batters.iloc[5] ['Matches_Played']]
runs_values= [top_batters.iloc[6]['Runs'], top_batters.iloc[2]['Runs'], top_batters.iloc[5] ['Runs']]
average_values = [top_batters.iloc[6] ['Average'], top_batters.iloc[2]['Average'], top_batters.iloc[5] ['Average']]
Strike_rate_values = [top_batters.iloc[6] ['Strike_Rate'], top_batters.iloc[2]['Strike_Rate'], top_batters.iloc[5]['Strike_Rate']]
Labels= ['KL Rahul', 'David Warner', 'Virat Kohli']

```
fig, axes=plt.subplots (2,2, figsize=(10,10))
axes[0][0].set_title("Matches Played")
axes[0][1].set_title("Runs in the IPL Career")
axes[1][0].set_title("Strike Rate")
axes[1][1].set_title("Average")
sns.barplot(x=Labels, y=matches_values, ax=axes[0][0])
sns.barplot(x=Labels, y=runs_values, ax=axes[0][1])
sns.barplot(x=Labels, y=Strike_rate_values, ax=axes [1][0])
sns.barplot(x=Labels, y=average_values, ax=axes[1][1])
```



```
#Batters for the Final 11 KL Rahul, Virat Kohli, David Warner

#here, we are storing the values of each player in a separate dataframe to use for displaying using the barplot.
top_batters.reset_index(drop=True)
matches_values = [top_batters.iloc[6]['Matches_Played'], top_batters.iloc[2]['Matches_Played'], top_batters.iloc[5] ['Matches_Played']]
runs_values= [top_batters.iloc[6]['Runs'], top_batters.iloc[2]['Runs'], top_batters.iloc[5] ['Runs']]
average_values = [top_batters.iloc[6] ['Average'], top_batters.iloc[2]['Average'], top_batters.iloc[5] ['Average']]
Strike_rate_values = [top_batters.iloc[6] ['Strike_Rate'], top_batters.iloc[2]['Strike_Rate'], top_batters.iloc[5]['Strike_Rate']]
Labels= ['KL Rahul', 'David Warner', 'Virat Kohli']

fig, axes=plt.subplots (2,2, figsize=(10,10))
axes[0][0].set_title("Matches Played")
axes[0][1].set_title("Runs in the IPL Career")
axes[1][0].set_title("Strike Rate")
axes[1][1].set_title("Average")
sns.barplot(x=Labels, y=matches_values, ax=axes[0][0], palette= "viridis")
sns.barplot(x=Labels, y=runs_values, ax=axes[0][1],palette= "viridis")
sns.barplot(x=Labels, y=Strike_rate_values, ax=axes [1][0],palette= "viridis")
sns.barplot(x=Labels, y=average_values, ax=axes[1][1],palette= "viridis")
```

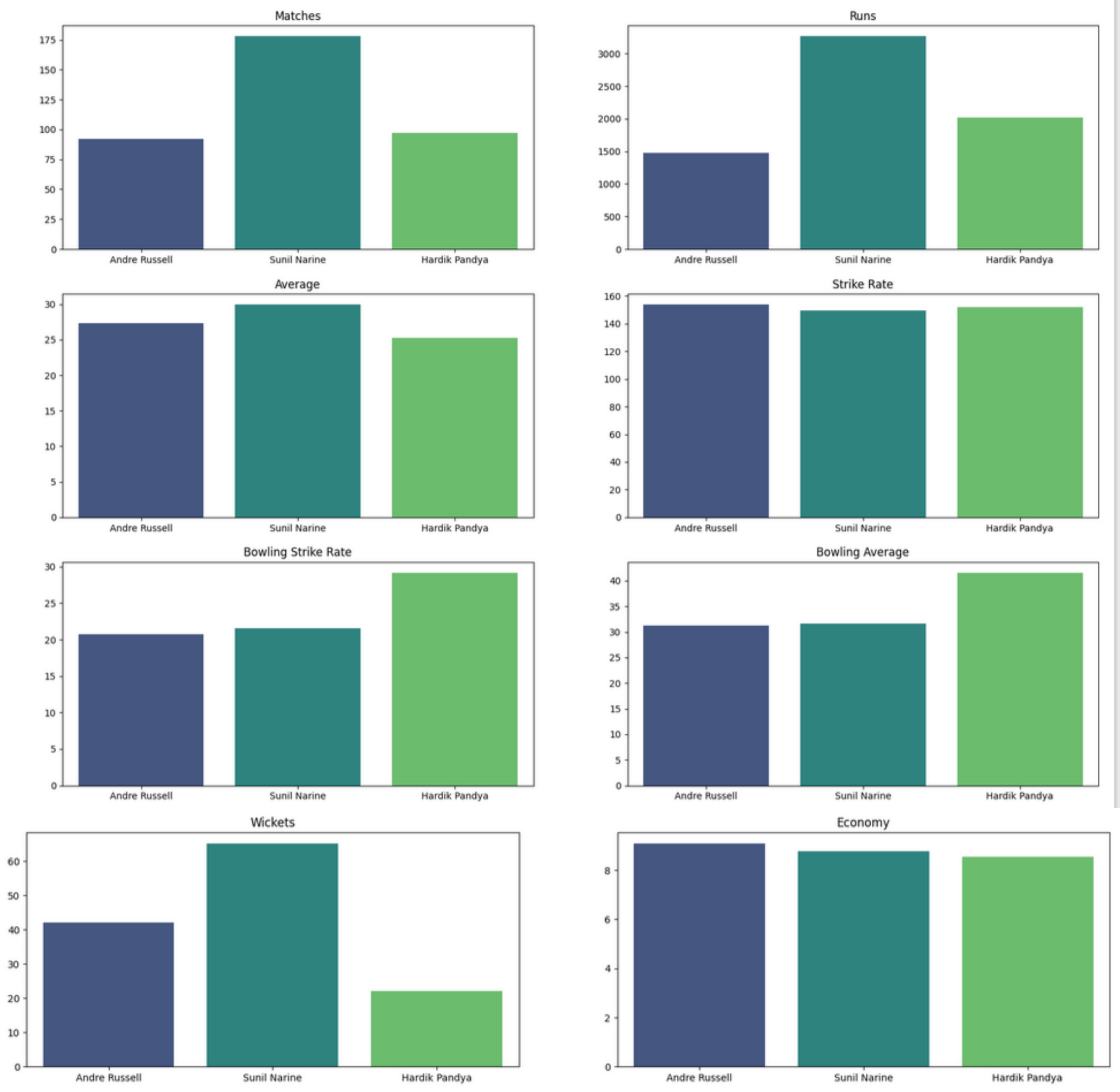
```
<Axes: title={'center': 'Economy'}>
```



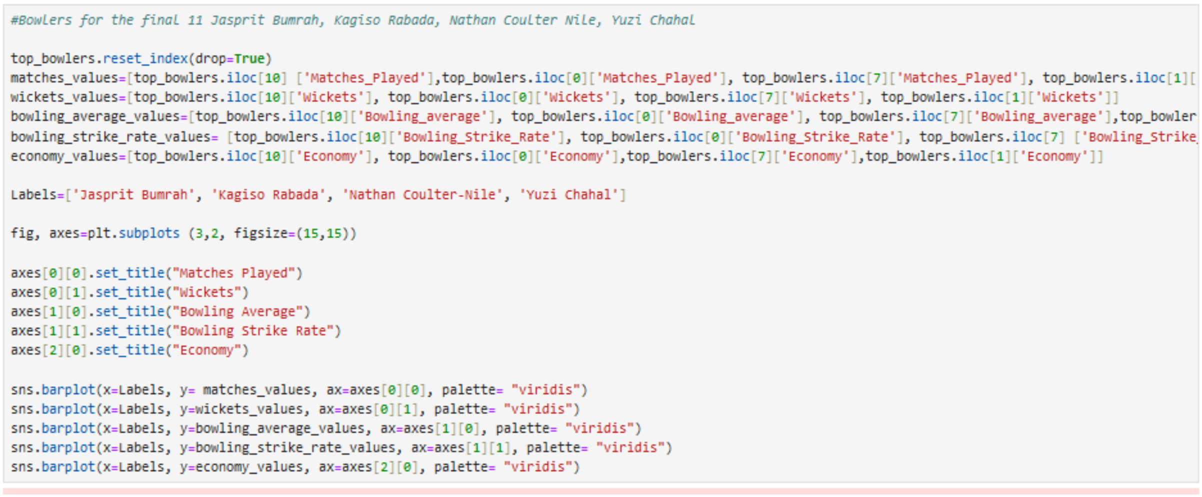
```
# In[163]:
#Allrounders for the final 11 - Andre Russell, Sunil Narine, Hardik Pandya

top_allrounders.reset_index(drop=True)

matches_values=[top_allrounders.iloc[5]['Matches_Played'],top_allrounders.iloc[9]['Matches
_Played'], top_allrounders.iloc[6]['Matches_Played']]
runs_values=[top_allrounders.iloc[5]['Runs'],                    top_allrounders.iloc[9]['Runs'],
top_allrounders.iloc[6]['Runs']]
average_values=[top_allrounders.iloc[5]['Average'],          top_allrounders.iloc[9]['Average'],
top_allrounders.iloc[6] ['Average']]
strike_rate_values=[top_allrounders.iloc[5]['Strike_Rate'],
top_allrounders.iloc[9]['Strike_Rate'], top_allrounders.iloc[6]['Strike_Rate']]
bowling_strike_rate_values=[top_allrounders.iloc[5]['Bowling_Strike_Rate'],top_allrounders
.iloc[9]['Bowling_Strike_Rate'],top_allrounders.iloc[6]['Bowling_Strike_Rate']]
bowling_average_values=[top_allrounders.iloc[5]['Bowling_average'],
top_allrounders.iloc[9]['Bowling_average'], top_allrounders.iloc[6]['Bowling_average']]
wickets_values=[top_allrounders.iloc[5]                                        ["Wickets"],
top_allrounders.iloc[9]['Wickets'],top_allrounders.iloc[6] ['Wickets']]
economy_values=                                      [top_allrounders.iloc[5]['Economy'],
top_allrounders.iloc[9]["Economy"],top_allrounders.iloc[6]['Economy']]

Labels=[ 'Andre Russell', 'Sunil Narine', 'Hardik Pandya']

fig, axes=plt.subplots(4,2, figsize=(20,20))
axes[0][0].set_title("Matches")
axes[0][1].set_title("Runs")
axes[1][0].set_title("Average")
axes[1][1].set_title("Strike Rate")
axes[2][0].set_title("Bowling Strike Rate")
axes[2][1].set_title("Bowling Average")
axes[3][0].set_title("Wickets")
axes[3][1].set_title("Economy")

sns.barplot(x=Labels, y=matches_values, ax=axes[0][0])
sns.barplot(x=Labels, y=runs_values, ax=axes[0][1])
sns.barplot(x=Labels, y=average_values, ax=axes[1][0])
sns.barplot(x=Labels, y=strike_rate_values, ax=axes [1][1])
sns.barplot(x=Labels, y=bowling_strike_rate_values, ax=axes [2][0])
sns.barplot(x=Labels, y=bowling_average_values, ax=axes [2][1])
sns.barplot(x=Labels, y=wickets_values, ax=axes[3][0])
sns.barplot(x=Labels, y=economy_values, ax=axes[3][1])
```

```
#Allrounders for the final 11 - Andre Russell, Sunil Narine, Hardik Pandya

top_allrounders.reset_index(drop=True)

matches_values=[top_allrounders.iloc[5]['Matches_Played'],top_allrounders.iloc[9]['Matches_Played'], top_allrounders.iloc[6]['Matches_Played']]
runs_values=[top_allrounders.iloc[5]['Runs'], top_allrounders.iloc[9]['Runs'], top_allrounders.iloc[6]['Runs']]
average_values=[top_allrounders.iloc[5]['Average'], top_allrounders.iloc[9]['Average'], top_allrounders.iloc[6] ['Average']]
strike_rate_values=[top_allrounders.iloc[5]['Strike_Rate'], top_allrounders.iloc[9]['Strike_Rate'], top_allrounders.iloc[6]['Strike_Rate']]
bowling_strike_rate_values=[top_allrounders.iloc[5]['Bowling_Strike_Rate'],top_allrounders.iloc[9]['Bowling_Strike_Rate'],top_allrounders.iloc[6]['Bowling
bowling_average_values=[top_allrounders.iloc[5]['Bowling_average'], top_allrounders.iloc[9]['Bowling_average'], top_allrounders.iloc[6]['Bowling_average']]
wickets_values=[top_allrounders.iloc[5] ["Wickets"], top_allrounders.iloc[9]['Wickets'],top_allrounders.iloc[6] ['Wickets']]
economy_values= [top_allrounders.iloc[5]['Economy'], top_allrounders.iloc[9]["Economy"],top_allrounders.iloc[6]['Economy']]

Labels=[ 'Andre Russell', 'Sunil Narine', 'Hardik Pandya']

fig, axes=plt.subplots(4,2, figsize=(20,20))
axes[0][0].set_title("Matches")
axes[0][1].set_title("Runs")
axes[1][0].set_title("Average")
axes[1][1].set_title("Strike Rate")
axes[2][0].set_title("Bowling Strike Rate")
axes[2][1].set_title("Bowling Average")
axes[3][0].set_title("Wickets")
axes[3][1].set_title("Economy")

sns.barplot(x=Labels, y=matches_values, ax=axes[0][0], palette= "viridis")
sns.barplot(x=Labels, y=runs_values, ax=axes[0][1], palette= "viridis")
sns.barplot(x=Labels, y=average_values, ax=axes[1][0], palette= "viridis")
sns.barplot(x=Labels, y=strike_rate_values, ax=axes [1][1], palette= "viridis")
sns.barplot(x=Labels, y=bowling_strike_rate_values, ax=axes [2][0], palette= "viridis")
sns.barplot(x=Labels, y=bowling_average_values, ax=axes [2][1], palette= "viridis")
sns.barplot(x=Labels, y=wickets_values, ax=axes[3][0], palette= "viridis")
sns.barplot(x=Labels, y=economy_values, ax=axes[3][1], palette= "viridis")
```

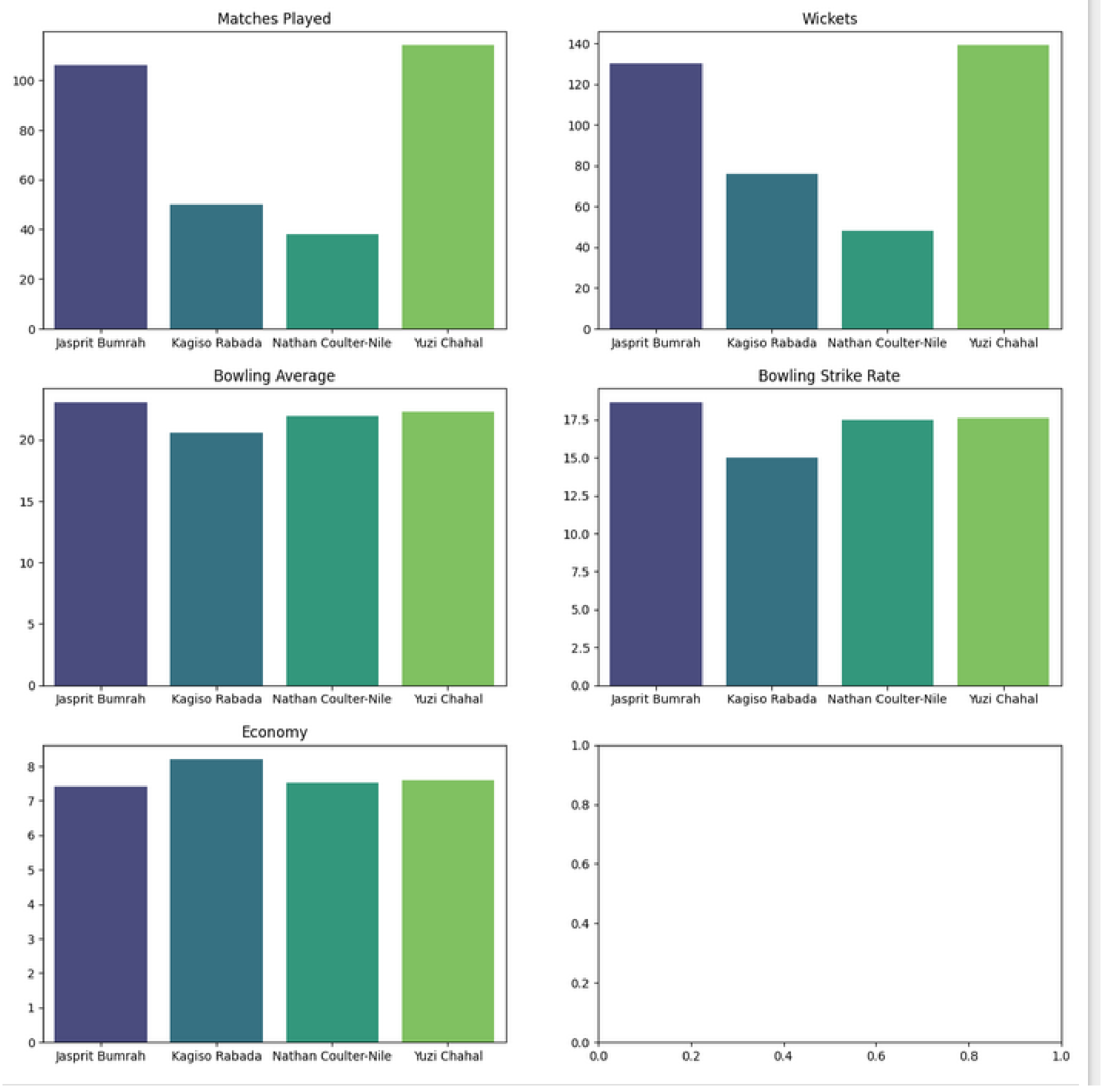
```
[161]: <Axes: title={'center': 'Economy'}>
```



```
# In[164]:
#Bowlers for the final 11 Jasprit Bumrah, Kagiso Rabada, Nathan Coulter Nile, Yuzi Chahal

top_bowlers.reset_index(drop=True)
matches_values=[top_bowlers.iloc[10]
['Matches_Played'],top_bowlers.iloc[0]['Matches_Played'],
top_bowlers.iloc[7]['Matches_Played'], top_bowlers.iloc[1]['Matches_Played']]
wickets_values=[top_bowlers.iloc[10]['Wickets'],                    top_bowlers.iloc[0]['Wickets'],
top_bowlers.iloc[7]['Wickets'], top_bowlers.iloc[1]['Wickets']]
bowling_average_values=[top_bowlers.iloc[10]['Bowling_average'],
top_bowlers.iloc[0]['Bowling_average'],
top_bowlers.iloc[7]['Bowling_average'],top_bowlers.iloc[1]['Bowling_average']]
bowling_strike_rate_values=                        [top_bowlers.iloc[10]['Bowling_Strike_Rate'],
top_bowlers.iloc[0]['Bowling_Strike_Rate'],                              top_bowlers.iloc[7]
['Bowling_Strike_Rate'],top_bowlers.iloc[1]['Bowling_Strike_Rate']]
economy_values=[top_bowlers.iloc[10]['Economy'],
top_bowlers.iloc[0]['Economy'],top_bowlers.iloc[7]['Economy'],top_bowlers.iloc[1]['Econo
my']]

Labels=['Jasprit Bumrah', 'Kagiso Rabada', 'Nathan Coulter-Nile', 'Yuzi Chahal']

fig, axes=plt.subplots (3,2, figsize=(15,15))

axes[0][0].set_title("Matches Played")
axes[0][1].set_title("Wickets")
axes[1][0].set_title("Bowling Average")
axes[1][1].set_title("Bowling Strike Rate")
axes[2][0].set_title("Economy")

sns.barplot(x=Labels, y= matches_values, ax=axes[0][0])
sns.barplot(x=Labels, y=wickets_values, ax=axes[0][1])
sns.barplot(x=Labels, y=bowling_average_values, ax=axes[1][0])
sns.barplot(x=Labels, y=bowling_strike_rate_values, ax=axes[1][1])
sns.barplot(x=Labels, y=economy_values, ax=axes[2][0])
```



```
#Bowlers for the final 11 Jasprit Bumrah, Kagiso Rabada, Nathan Coulter Nile, Yuzi Chahal

top_bowlers.reset_index(drop=True)
matches_values=[top_bowlers.iloc[10] ['Matches_Played'],top_bowlers.iloc[0]['Matches_Played'], top_bowlers.iloc[7]['Matches_Played'], top_bowlers.iloc[1][
wickets_values=[top_bowlers.iloc[10]['Wickets'], top_bowlers.iloc[0]['Wickets'], top_bowlers.iloc[7]['Wickets'], top_bowlers.iloc[1]['Wickets']]
bowling_average_values=[top_bowlers.iloc[10]['Bowling_average'], top_bowlers.iloc[0]['Bowling_average'], top_bowlers.iloc[7]['Bowling_average'],top_bowler
bowling_strike_rate_values= [top_bowlers.iloc[10]['Bowling_Strike_Rate'], top_bowlers.iloc[0]['Bowling_Strike_Rate'], top_bowlers.iloc[7] ['Bowling_Strike
economy_values=[top_bowlers.iloc[10]['Economy'], top_bowlers.iloc[0]['Economy'],top_bowlers.iloc[7]['Economy'],top_bowlers.iloc[1]['Economy']]

Labels=['Jasprit Bumrah', 'Kagiso Rabada', 'Nathan Coulter-Nile', 'Yuzi Chahal']

fig, axes=plt.subplots (3,2, figsize=(15,15))

axes[0][0].set_title("Matches Played")
axes[0][1].set_title("Wickets")
axes[1][0].set_title("Bowling Average")
axes[1][1].set_title("Bowling Strike Rate")
axes[2][0].set_title("Economy")

sns.barplot(x=Labels, y= matches_values, ax=axes[0][0], palette= "viridis")
sns.barplot(x=Labels, y=wickets_values, ax=axes[0][1], palette= "viridis")
sns.barplot(x=Labels, y=bowling_average_values, ax=axes[1][0], palette= "viridis")
sns.barplot(x=Labels, y=bowling_strike_rate_values, ax=axes[1][1], palette= "viridis")
sns.barplot(x=Labels, y=economy_values, ax=axes[2][0], palette= "viridis")
]:  <Axes: title={'center': 'Economy'}>
```



# In[165]:
#Wicket Keeper For The Final 11-MS Dhoni
matches_values = [top_keepers.iloc[8] ['Matches_Played'], top_keepers.iloc[8] ['Runs']]
average_values = [top_keepers.iloc[8]['Average'], top_keepers.iloc[8] ['Strike_Rate']]
keeping_values = [top_keepers.iloc[8]['Catches'], top_keepers.iloc[8] ['Stumps'], top_keepers.iloc[8]['Run_outs']]

labell= ['Matches', 'Runs']

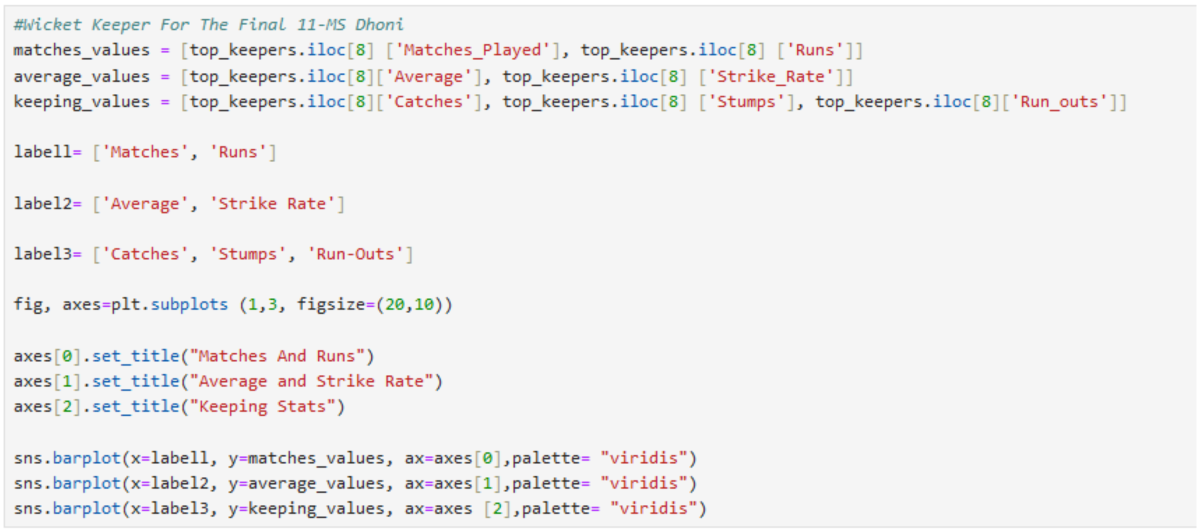
label2= ['Average', 'Strike Rate']

label3= ['Catches', 'Stumps', 'Run-Outs']

fig, axes=plt.subplots (1,3, figsize=(20,10))

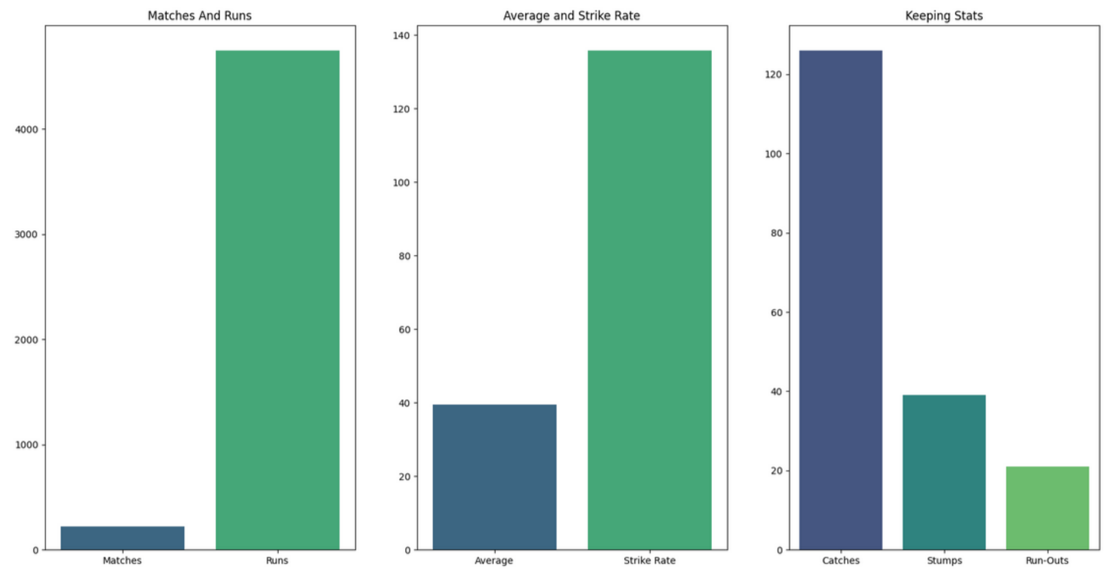
axes[0].set_title("Matches And Runs")
axes[1].set_title("Average and Strike Rate")
axes[2].set_title("Keeping Stats")

sns.barplot(x=labell, y=matches_values, ax=axes[0])
sns.barplot(x=label2, y=average_values, ax=axes[1])
sns.barplot(x=label3, y=keeping_values, ax=axes [2])



```
#Wicket Keeper For The Final 11-MS Dhoni
matches_values = [top_keepers.iloc[8] ['Matches_Played'], top_keepers.iloc[8] ['Runs']]
average_values = [top_keepers.iloc[8]['Average'], top_keepers.iloc[8] ['Strike_Rate']]
keeping_values = [top_keepers.iloc[8]['Catches'], top_keepers.iloc[8] ['Stumps'], top_keepers.iloc[8]['Run_outs']]

labell= ['Matches', 'Runs']

label2= ['Average', 'Strike Rate']

label3= ['Catches', 'Stumps', 'Run-Outs']

fig, axes=plt.subplots (1,3, figsize=(20,10))

axes[0].set_title("Matches And Runs")
axes[1].set_title("Average and Strike Rate")
axes[2].set_title("Keeping Stats")

sns.barplot(x=labell, y=matches_values, ax=axes[0],palette= "viridis")
sns.barplot(x=label2, y=average_values, ax=axes[1],palette= "viridis")
sns.barplot(x=label3, y=keeping_values, ax=axes [2],palette= "viridis")
```

```
[163]:  <Axes: title={'center': 'Keeping Stats'}>
```



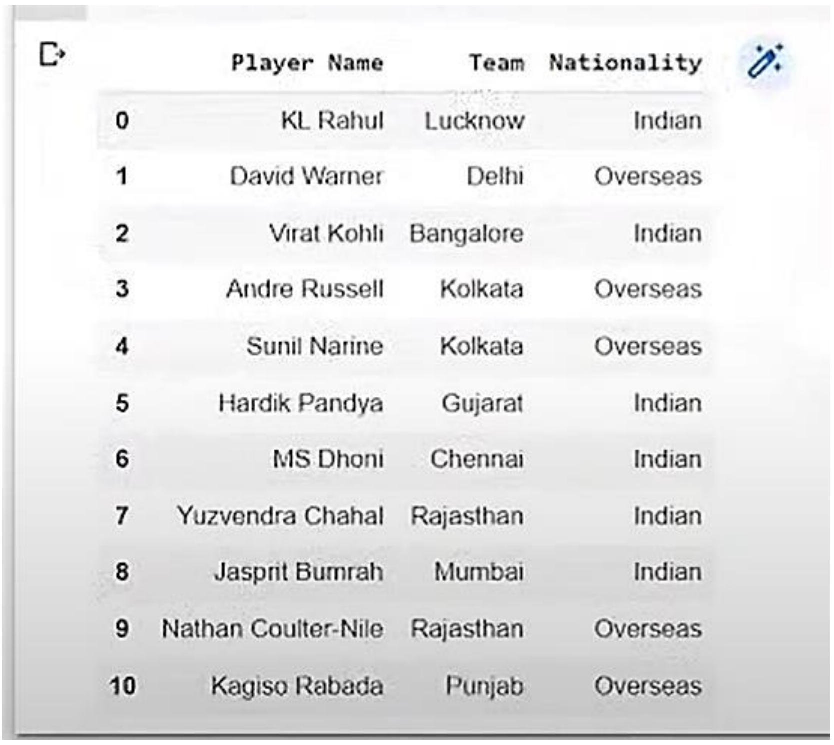
# In[173]:
batterl=top_batters.loc[(top_batters["Player Name"] =='KL Rahul ')]
batter2=top_batters.loc[(top_batters["Player Name"] =='David Warner')]
batter3=top_batters.loc[(top_batters["Player Name"] =='Virat Kohli')]

bowlerl=top_bowlers.loc[(top_bowlers["Player Name"] =='Yuzvendra Chahal')]
bowler2=top_bowlers.loc[(top_bowlers["Player Name"] =='Jasprit Bumrah')]
bowler3=top_bowlers.loc[(top_bowlers["Player Name"] =='Kagiso Rabada ')]
bowler4= top_bowlers.loc[(top_bowlers["Player Name"]=='Nathan Coulter-Nile')]

allrounderl=top_allrounders.loc[(top_allrounders ["Player Name"] =='Andre Russell')]
allrounder2=top_allrounders.loc[(top_allrounders ["Player Name"]=='Sunil Narine')]
allrounder3=top_allrounders.loc[(top_allrounders ["Player Name"] =='Hardik Pandya')]

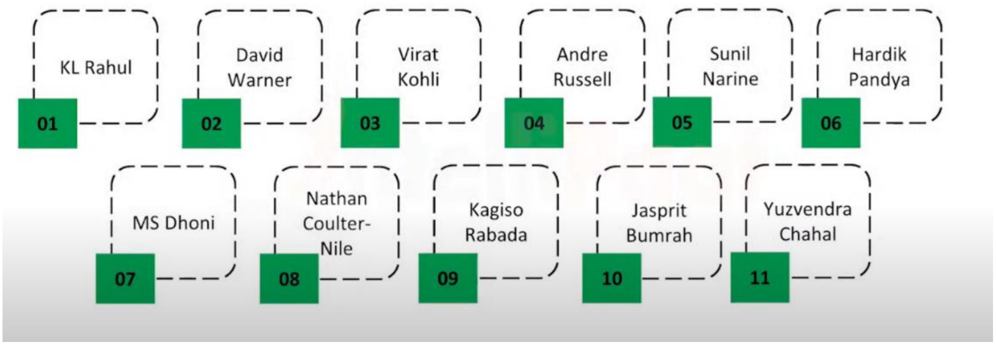
keeper= top_keepers.loc[(top_keepers["Player Name"]==' MS Dhoni')]

# In[174]:
final= [batterl, batter2, batter3,allrounderl, allrounder2, allrounder3, keeper, bowlerl, bowler2, bowler3, bowler4]
final_team=pd.concat(final)
final_team=final_team.drop(labels=['Matches_Played', 'Runs', 'Average', 'Strike_Rate', 'Wickets', 'Bowling_average', 'Economy', 'Bowling_Strike_Rate',
'Catches', 'Run_outs', 'Stumps'], axis=1)
final_team.reset_index(drop=True)

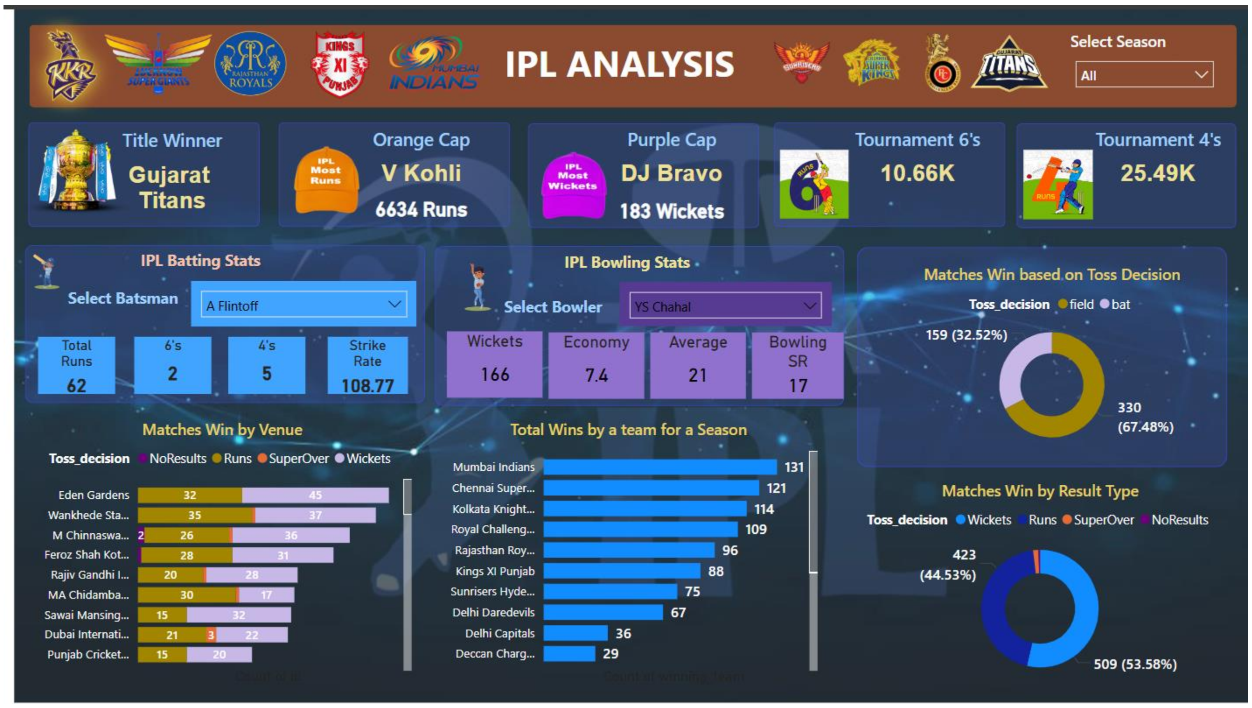


	Player Name	Team	Nationality
0	KL Rahul	Lucknow	Indian
1	David Warner	Delhi	Overseas
2	Virat Kohli	Bangalore	Indian
3	Andre Russell	Kolkata	Overseas
4	Sunil Narine	Kolkata	Overseas
5	Hardik Pandya	Gujarat	Indian
6	MS Dhoni	Chennai	Indian
7	Yuzvendra Chahal	Rajasthan	Indian
8	Jasprit Bumrah	Mumbai	Indian
9	Nathan Coulter-Nile	Rajasthan	Overseas
10	Kagiso Rabada	Punjab	Overseas

RESULT:



Power-BI Dashboard:



About this Dashboard:

The IPL Analysis Dashboard is an innovative project aimed at providing comprehensive insights and analysis of the Indian Premier League (IPL) cricket tournament. This dashboard serves as a valuable tool for various stakeholders, including coaches, IPL cricket teams, and the Board of Control for Cricket in India (BCCI), by offering key performance indicators (KPIs) and statistical data that drive strategic decision-making and enhance team performance. Key Features and KPIs:

1. Title Winner: The dashboard highlights the winner of each IPL season, allowing stakeholders to track the historical performance of teams and identify trends.
2. Purple Cap and Orange Cap Holder: The Purple Cap is awarded to the leading wicket-taker, while the Orange Cap is given to the top run-scorer of the tournament. These metrics are crucial in assessing individual player performances.
3. Tournament 6's and 4's by Individuals: The dashboard presents data on the number of sixes and fours hit by individual players, providing valuable insights into their batting prowess and scoring capabilities.
4. Batting and Bowling Statistics: Detailed statistical analysis of batting and bowling performances of players allows coaches and team managers to evaluate player strengths, weaknesses, and overall contributions to the team.
5. Matches Won based on Toss Decision: This feature reveals the outcomes of matches based on the toss decision (batting or bowling first). Understanding the impact of the toss can help teams devise winning strategies.
6. Match Win by Venues: The dashboard offers an overview of match results based on different venues, enabling teams to identify patterns and adjust strategies accordingly.
7. Total Wins by Teams for a Season: This metric displays the total number of wins for each IPL team in a specific season, aiding in assessing team performance and setting realistic goals.

8. Match Win by Result Type: The dashboard categorizes match results into different types, such as by runs, wickets, super overs, or run rate, providing a comprehensive overview of match outcomes.

By leveraging the IPL Analysis Dashboard, coaches, IPL cricket teams, and the BCCI can gain valuable insights into team and player performances. This information facilitates data-driven decision-making, player selection, strategy formulation, and overall improvement in team performance. With its user-friendly interface and comprehensive data visualization, the dashboard serves as an indispensable tool for IPL stakeholders, enabling them to maximize their chances of success in the tournament.

Insights Drawn:

- Player Performance: Unearth standout performers such as V Kohli, amassing an impressive 6634 runs, and DJ Bravo, boasting a remarkable 193 wickets, showcasing the essence of IPL cricket.
- Crucial Metrics: Gain insights into vital stats like total sixes (10.66K) and total fours (25.49K) that underscore the electrifying gameplay of IPL tournaments.
- Detailed Analysis: Access player-specific details; for instance, AB de Villiers' noteworthy stats, including 5181 runs, 253 sixes, and a striking 148.58 strike rate.
- Bowling Brilliance: Analyze the bowling prowess of players like YS Chahal, with an average of 21.07, an economy rate of 7.40, and a bowling strike rate of 17.09.
- Winning Dynamics: Uncover the significance of toss wins, win percentages based on match outcomes (32.52% wins by runs, 44.53% by wickets), and the distribution of victories across venues.
- Seasonal Dominance: Gain insights into team performance, highlighting Mumbai Indians with 131 wins, Chennai Super Kings with 121, and Kolkata Knight Riders with 114, shaping the tournament's legacy.

CONCLUSION:

The “Cricket Data Analysis Project” represents a comprehensive approach to understanding and visualizing cricket data by integrating modern data analysis technologies and methods. This project addresses the growing need for data-driven insights in the sports industry, specifically in cricket, where performance metrics, match outcomes, and player statistics are of great interest to analysts, teams, coaches, and fans alike. By utilizing web scraping, Python, Pandas, and Power BI, this project successfully develops a streamlined data pipeline that automates data collection, cleans and structures the data, analyzes key trends, and generates interactive visualizations for easy interpretation.

Through the initial stages of web scraping, data from reputable cricket websites, such as ESPN Cricinfo and Cricbuzz, is collected in real-time. The project overcomes challenges associated with dynamic content and unstructured data by leveraging Python libraries like BeautifulSoup and Selenium, making data collection efficient and reliable. This raw data is then processed using Pandas, which offers a robust framework for cleaning, organizing, and transforming the data into usable formats. This preparation stage ensures data consistency, accuracy, and usability for subsequent analysis.

The Exploratory Data Analysis (EDA) phase, enabled by libraries such as Matplotlib and Seaborn, provides deep insights into player performance, team strategies, match trends, and various other metrics. By visualizing these aspects, analysts can uncover patterns that may influence match outcomes and player decisions. The EDA stage forms the foundation for the visualizations created in Power BI, where interactive dashboards bring the data to life, allowing users to interact with the data, filter specific attributes, and drill down into areas of interest. The result is an intuitive and accessible dashboard that communicates valuable insights to both technical and non-technical stakeholders.

This project not only demonstrates the power of data analysis in sports but also highlights the essential tools and methodologies required to turn raw, unstructured web data into actionable insights. Using a combination of programming (Python), data manipulation (Pandas), and visualization (Power BI), this data pipeline is adaptable to other sports or fields where data analysis is essential. The integration of real-time data scraping, efficient data processing, and user-friendly dashboards demonstrates how modern data tools can transform the way we interpret and utilize sports data.

In conclusion, the “Cricket Data Analysis Project” is a valuable endeavor that bridges the gap between raw data and practical insights in cricket. It underscores the importance of data literacy in sports, providing an accessible platform for stakeholders to explore, analyze, and understand cricket data. This research project, therefore, contributes to the field of sports analytics and can serve as a model for similar applications in other sports and domains where data can enhance decision-making and performance analysis.

REFERENCES:

[1]. Daniel MagoVistro, Faizan Rasheed, Leo Gertrude David, “*The Cricket Winner Prediction With Application of Machine Learning And Data Analytics*” International Journal of Scientific & Technology Research (2019)
[2]. Madan Gopal Jhanwar and VikramPudi, “*Predicting the Outcome of ODI Cricket Matches: A Team Composition Based Approach*” International Institution of Information Technology (2017)
[3]. I. P. Wickramasingheet. al, *"Predicting the performance of batsmen in test cricket," Journal of Human Sport & Exercise*”, vol. 9, no. 4, pp. (2017)
[4]. R. P. Schumaker, O. K. Solieman and H. Chen, "*Predictive Modeling for Sports and Gaming*” in Sports Data Mining, vol. 26, Boston, Massachusetts: Springer, (2016)
[5]. J. McCullagh, "*Data Mining in Sport: A Neural Network Approach*," International Journal of Sports Science and Engineering, vol. 4, no. 3 (2016)
[6]. Bunker, Rory &Thabtah, Fadi. “*A Machine Learning Framework for Sport Result Prediction. Applied Computing and Informatics*”. (2017)
[7] Kulkarni, V. & Sinha, P., n.d. Effective Learning and Classification using Random Forest Algorithm. International Journal of Engineering and Innovative Technology (IJEIT).
[8] Lokhande, A., Chawan, R. &. &Pramila&, S., 2018. Prediction of Live Cricket Score and Winning. Computer and IT Dept, VeermataJeejabai Technological Institute, Mumbai, India, 5(4)(2394-9333).
[9] Mitchel, M. T., 1997. Machine learning. Burr Ridge, IL: McGraw Hill, 45, 1997.
[10] Murphy, K. P., 2006. Naive bayes classifiers. University of British Columbia.
[11] Nasteski&Vladmir, 2007. An Overview of the Supervised Machine Learning Methods. Faculty of Information and Technology. Faculty of Information and communication Technologies.
[12] Available at: https://medium.com/machine-learning-101/chapter2-svm-support-vectormachine-theory-f0812effc72
[13] Shah, P. & Shah, M., 2015. Predicting ODI Cricket Result. ISSN (Paper) 2312-5187 ISSN (Online) 2312-5179 An International Peer-reviewed Journal, Volume 5.
[14] Asare-Frempong, J. and Jayabalan, M., 2017. Predicting customer response to bank direct telemarketing campaign. In 2017 International Conference on Engineering Technology and Technopreneurship (ICE2T) (pp. 1-4). IEEE.
[15] Yasir, M. et al., 2017. Ongoing Match Prediction in T20 International. IJCSNS International Journal of Computer Science and Network Security. [16] Python https://www.python.org/