Cricket Performance Insights

Abstract

Cricket, one of the most widely followed sports globally, generates a wealth of performance data that can be analyzed to gain valuable insights into player and team performance. This project, "Cricket Performance Insights: Visualizing Trends and Patterns Using Pandas, Matplotlib, and Power BI," aims to analyze cricket batting and bowling statistics using data analysis and visualization techniques. The primary objective is to extract actionable insights that can aid in player performance evaluation, team strategy optimization, and selection of the Best Playing XI.

Through the use of Python’s pandas and matplotlib libraries, we preprocess the data, handle missing values, remove outliers, and create visualizations to understand the underlying patterns in player and team performance. The project also utilizes Power BI to design interactive dashboards, allowing dynamic selection of the Best Playing XI based on key performance metrics such as batting average, strike rate, economy rate, and wickets taken. These dashboards are built with filters that provide insights into specific player categories, such as openers, middle order, all-rounders, and bowlers.

The expected outcomes of this project include identifying the most influential factors impacting player performance, visualizing trends such as wins by ground, day, and team, and presenting a comprehensive Best Playing XI selection tool. These insights aim to enhance decision-making processes in cricket team selection and strategy formulation. Ultimately, this project provides a structured approach to leveraging cricket data to derive meaningful insights without relying on machine learning, ensuring data-driven decisions for optimal team performance.

Table of Content S.NO	TOPIC
1.	Title of Project
2.	Introduction
3.	Objective
4.	Features Implemented
5.	Scope of Work
6.	Errors Faced & Solutions
7.	Timeline of the project
8.	Tools and Technologies
9.	Work Flow
10.	Code Implementation
11.	Explanation of Code Implementation
12.	Performance Analysis
13.	Challenges & Future Enhancements
14.	Conclusion
15.	References

1. Introduction

Cricket is one of the most popular sports worldwide, and data analysis plays a crucial role in understanding player performance, match trends, and team statistics. This project focuses on analyzing cricket batting and bowling data using pandas and matplotlib to generate meaningful visualizations. Additionally, Power BI is used to create an interactive dashboard for selecting the Best Playing XI based on performance metrics. By leveraging data analysis techniques, we can draw meaningful insights that improving their decision-making processes.

2. Objectives

The primary objectives of this project are:

- To Identify top-performing players.
- To perform data preprocessing, including handling missing values and outliers.
- To Understand match trends.
- To Select the Best Playing XI based on data-driven decisions.
- To visualize the results and present actionable insights..

3. Features Implemented

This project is designed to extract valuable insights from cricket performance data to assist in team selection and performance strategy. The analysis begins with the collection of cricket data from ESPNcricinfo, which is available in JSON format. This data is processed using Python's pandas library, which converts the raw JSON data into a structured CSV format for easier analysis. The first task is to clean the dataset by handling missing values through imputation techniques, removing outliers, and standardizing performance metrics. This ensures the data is accurate and ready for further analysis.

Exploratory Data Analysis (EDA) is conducted next, which involves summarizing the dataset using descriptive statistics, such as mean, median, and variance, to gain an initial understanding of the trends and patterns in player performance. Various visualizations, including bar charts, pie charts, scatter plots, and histograms, are created using Matplotlib to further explore relationships between different performance metrics. These visualizations help uncover trends such as the number of matches won by each team, runs scored by top players, and the distribution of wickets. The goal of EDA is to identify patterns and make the data more interpretable for decision-makers.

The project also involves developing performance models for both batting and bowling, which are used to evaluate players based on their respective roles. For batting, metrics like batting average, strike rate, consistency score, and boundary percentage are calculated to assess player performance. Similarly, bowling performance is evaluated using metrics such as bowling average, economy rate, strike rate, and dot ball percentage. These models allow for a comprehensive comparison of players' performances and help identify the most impactful factors that influence team success. The models are integrated into Power BI, where interactive dashboards are built to dynamically select the Best Playing XI based on a variety of performance criteria.

Finally, the analysis is validated by comparing the selected Best Playing XI with historical data, ensuring the model’s consistency and accuracy. The dashboards allow users to filter players based on their performance in different roles such as openers, middle-order batsmen, finishers, all-rounders, and bowlers. The final output of the project includes a comprehensive report that documents the entire analysis process, the visualizations created, and the key insights derived. These insights are used to make informed decisions about team selection and strategy, providing actionable recommendations that can help improve performance and decision-making in cricket matches.

4. Scope of Work

The project involves:

- Data Exploration: Understanding the dataset, including the features and target variable.
- Data Preprocessing: Cleaning the dataset by handling missing values, removing outliers, and normalizing/standardizing the data.
- Feature Selection: Identifying the most significant features influencing cricket performance.
- Data Visualization: Using plots and graphs to visualize the relationship between features and cricket performance. Using interactive dashboards that selects the Best Playing XI based on batting and bowling performances.
- Model Building: Building models to Cricket Performance Insights using Pandas, matplotlib, power bi.
- Interpretation of Results: Analyzing the output of the models and drawing conclusions.
- Reporting: Documenting the findings and preparing a final report..

5. Errors Faced & Solutions

1. Data Quality Issues: Missing values and inconsistent data entries were found.

   Solution: Imputation techniques (mean, median, mode) were applied, and data normalization was used to ensure consistency.

2. Outlier Detection: Extreme outliers were identified in player performance data.

   Solution: Outliers were detected using Z-scores and IQR methods and either removed or capped to maintain data integrity.

3. Data Integration: JSON to CSV conversion led to formatting issues due to varied structures.

   Solution: Custom Python scripts were developed to clean and standardize the JSON files before converting them into CSV format.

4. Power BI Integration: Dynamic filters in Power BI were not updating visualizations as expected.

   Solution: Table relationships were reviewed, and DAX formulas were adjusted to ensure filters provided real-time updates.

5. Model Accuracy: The performance models did not fully align with expected outcomes.

   Solution: Additional metrics like boundary and dot ball percentages were incorporated, and models were validated against historical data for accuracy.

6. Timeline of Project

Task No.	Task Name	Start Date	End Date	Total no.of Days
1.	Gathering Requirements	24/02/2025	24/02/2025	1
2.	Feasibility Study	25/02/2025	26/02/2025	2
3.	System Analysis	27/02/2025	28/02/2025	2
4.	Design/Approach	1/03/2025	2/03/2025	1
5.	Coding	2/03/2025	2/03/2025	1
6.	Testing & adding new features	03/03/2025	4/03/2025	2
			Toatal	12 Days

7. Tools and Technologies

The project will utilize the following tools and technologies:

- Programming Language: Python (For data preprocessing, analysis, and visualization).
- Libraries: Pandas (For handling and manipulating structured data), Matplotlib (For generating various statistical and trend-based visualizations).
- IDE: Jupyter Notebook (For interactive data analysis and visualization)
- Data Source: ESPNcricinfo (Source of cricket performance data), JSON (Data formats used for storing and processing cricket data), Excel (For additional data cleaning and manual inspection)
- Power BI: Dashboards Creation (Designing interactive dashboards for performance comparison and team selection).

8. Work Flow

Data Collection:

- Collect raw cricket performance data from ESPNcricinfo in JSON format.
- Use Python (Pandas) to convert the raw data into a structured CSV format for easier analysis.

Data Preprocessing:

- **Handle Missing Values**: Apply imputation techniques (mean, median, mode) for numerical and categorical data.
- **Remove Outliers**: Identify outliers using statistical methods (Z-scores, IQR) and either remove or cap them.
- **Data Normalization**: Standardize numerical features to ensure consistency across different players and seasons.

Exploratory Data Analysis (EDA):

- Perform descriptive statistics (mean, median, variance) to summarize the dataset.
- Visualize trends and patterns using Matplotlib (bar charts, line plots, scatter plots).
- Create interactive visualizations and dashboards with Power BI for data exploration.

Feature Selection:

- Identify key performance metrics such as batting average, strike rate, economy rate, etc.
- Select the most significant features influencing cricket performance using statistical methods.

Model Building:

- Develop performance models for batting (e.g., batting average, consistency score) and bowling (e.g., bowling average, economy rate).
- Implement these models in Power BI to allow dynamic selection of the Best Playing XI based on player form and performance metrics.

Evaluation & Interpretation:

- Analyze the accuracy and relevance of the models by comparing them to historical performance data.
- Validate the selected Best Playing XI using historical data to ensure it aligns with past performances.

Visualization:

- Generate various visualizations (bar charts, pie charts, scatter plots, histograms) using Matplotlib.
- Develop interactive Power BI dashboards with filters to dynamically select the Best Playing XI.

- Display key insights through visualizations like top run-scorers, strike rates, and match statistics.

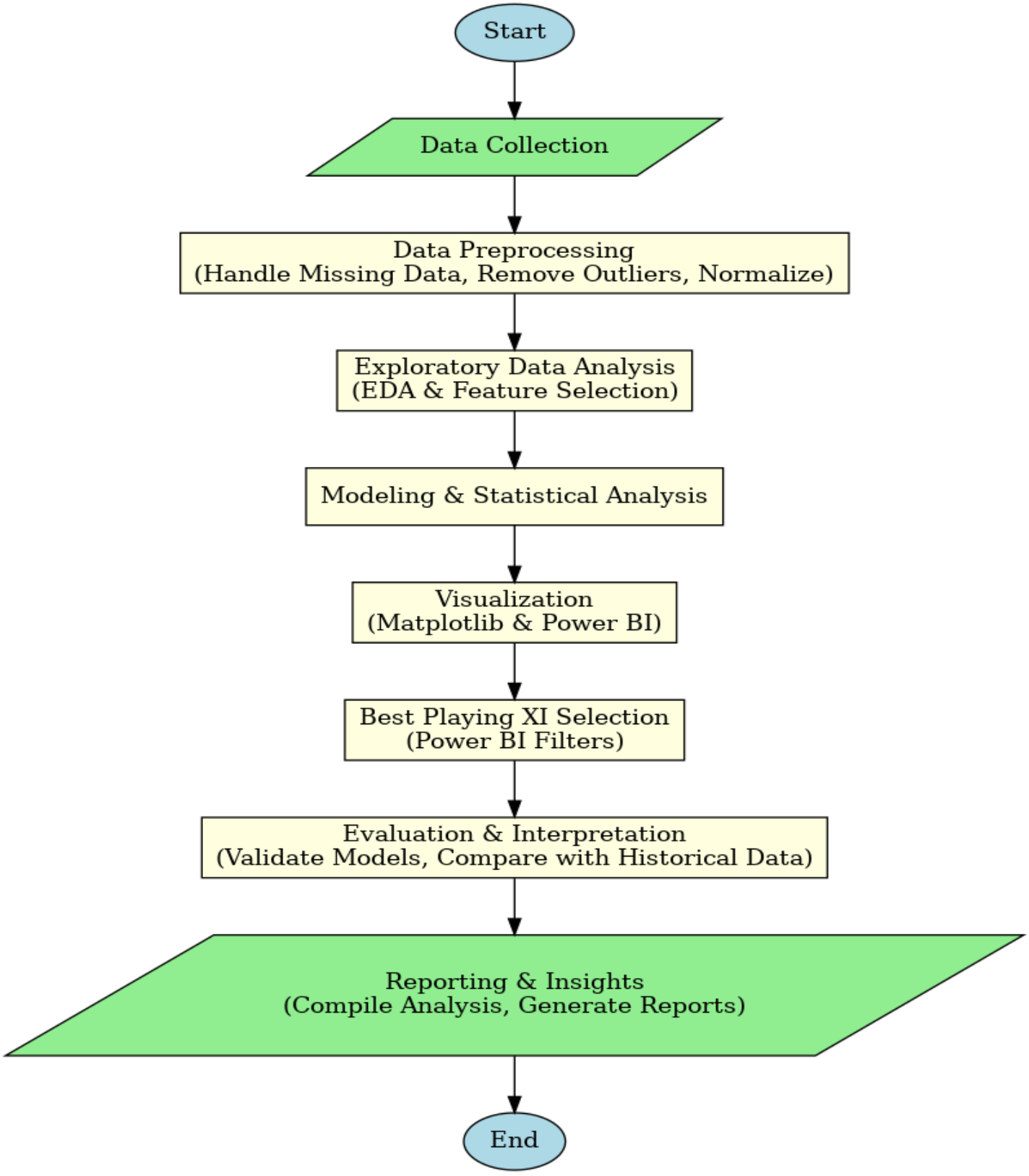
Reporting:

- Document the data analysis process, methodologies, and results.
- Prepare a comprehensive final report detailing insights, findings, and recommendations for improving decision-making in team selection.

Final Submission:

- Deliver the final report along with the visualizations and Power BI dashboard for review and implementation.

Flowchart:


Start
Data Collection
Data Preprocessing
(Handle Missing Data, Remove Outliers, Normalize)
Exploratory Data Analysis
(EDA & Feature Selection)
Modeling & Statistical Analysis
Visualization
(Matplotlib & Power BI)
Best Playing XI Selection
(Power BI Filters)
Evaluation & Interpretation
(Validate Models, Compare with Historical Data)
Reporting & Insights
(Compile Analysis, Generate Reports)
End

9. Code Implementation and Explanation of Code Implementation

Code:

Data Collection

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset from a CSV file
df1 = pd.read_csv("csv/match_summary.csv")
df1.head()
```

Key Functionalities:

Load CSV Data: Loads data from the CSV file match_summary.csv into a Pandas DataFrame (df1).

Inspect Data: Displays the first five rows of the DataFrame using .head() to provide a quick overview of the data.

Data Preprocessing

```
import pandas as pd
import matplotlib.pyplot as plt

# Check for missing values
df1.isnull().sum()

# Remove particular row if column as Nan
df1.dropna(subset=['margin'],inplace=True)

# Checking for any duplicates
df1.duplicated().sum()

# Remove any duplicate rows
df1.drop_duplicates(inplace=True)
```

Key functionalities of the code:

1. **Check for Missing Values**: Identifies and sums up the missing (NaN) values in the DataFrame using df1.isnull().sum().
2. **Remove Rows with Missing Values**: Removes rows where the margin column has NaN values using df1.dropna().
3. **Check for Duplicates**: Identifies and counts any duplicate rows in the DataFrame using df1.duplicated().sum().
4. **Remove Duplicate Rows**: Removes any duplicate rows from the DataFrame using df1.drop_duplicates().

Matches Won by Each Team (Bar Chart)

```
# Count matches won by each team
team_wins = df1['winner'].value_counts()

# Generate random colors for each bar
colors = [plt.cm.Paired(i) for i in range(len(team_wins))]

# Plot bar chart
plt.figure(figsize=(10,5))
plt.bar(team_wins.index,team_wins.values,color=colors,edgecolor="black")

# Add labels and title
plt.title("Number of Matches Won by Each Team")
plt.xlabel("Teams")
plt.ylabel("Number of Wins")
plt.xticks(rotation=45) # Rotate x-axis labels for better readability
plt.show()
```

Key functionalities of the code:

1. **Count Matches Won by Each Team**: Uses value_counts() to count how many matches each team has won from the winner column.
2. **Generate Random Colors**: Creates a list of random colors for each bar using plt.cm.Paired.
3. **Plot Bar Chart**: Creates a bar chart with teams on the x-axis and their

corresponding wins on the y-axis.

4. **Customize Plot**: Adds a title, labels for the x and y axes, and rotates the x-axis labels for better readability.
5. **Display Plot**: Displays the generated bar chart using plt.show().

Wins by Ground (Pie Chart)

```
ground_counts = df1["ground"].value_counts()

# pie plot
plt.figure(figsize=(8,5))
plt.plot(ground_counts.index,ground_counts.values,marker="o",linestyle="-",color="purple")

# Add title
plt.title("Number of Matches Played at Each Ground")
plt.xlabel("Ground")
plt.ylabel("Matches Played")
plt.xticks(rotation=90)

plt.grid(True)
plt.show()
```

Key functionalities of the code:

1. **Count Matches Played at Each Ground**: Uses value_counts() to count the number of matches played at each ground from the ground column.
2. **Plot Line Chart**: Creates a line chart where the x-axis represents grounds, and the y-axis represents the number of matches played at each ground.
3. **Customize Plot**: Adds a title, labels for the x and y axes, and rotates the x-axis labels for better readability.
4. **Enable Grid**: Displays grid lines on the plot for better visual clarity using plt.grid(True).
5. **Display Plot**: Displays the line plot using plt.show().

wins Over Day (Scatter Chart)

```
df1["matchDate"] = pd.to_datetime(df1["matchDate"]) # Convert to datetime format

# Format date
df1["FormattedDate"] = df1["matchDate"].dt.strftime("%b %d, %Y")

# Extract day for trend analysis
df1['Day'] = df1['matchDate'].dt.day

# Count matches per day
matches_per_day = df1["Day"].value_counts().sort_index()

plt.figure(figsize=(10,5))
plt.scatter(matches_per_day.index,matches_per_day.values, color="red",s=100,alpha=0.7,edgecolors="black

# Add labels and title
plt.title("Number of Matches Played Each Day")
plt.xlabel("Day")
plt.ylabel("Matches Played")
plt.grid(True,linestyle="--",alpha=0.6)
plt.show()
```

Key functionalities of the code:

1. **Convert to Datetime Format**: Converts the matchDate column to a Pandas datetime format using pd.to_datetime().
2. **Format Date**: Creates a new column FormattedDate with the date formatted as "Month Day, Year" using .dt.strftime().
3. **Extract Day**: Extracts the day of the month from the matchDate and stores it in a new column Day.
4. **Count Matches Per Day**: Counts the number of matches played on each day using value_counts() and sorts the results by day.
5. **Plot Scatter Plot**: Creates a scatter plot with days on the x-axis and the number of matches on the y-axis.
6. **Customize Plot**: Adds a title, axis labels, gridlines, and customizes the appearance of the scatter points.
7. **Display Plot**: Displays the scatter plot using plt.show()

BATTING SCORE

Data Collection

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset from a CSV file
df2 = pd.read_csv("csv/batting_score.csv")
df2.head()
```

Key functionalities of the code:

1. **Load CSV Data**: Loads data from the CSV file batting_score.csv into a Pandas DataFrame (df2).
2. **Inspect Data**: Displays the first five rows of the DataFrame using .head() to get an overview of the dataset.

Data Preprocessing

```
import pandas as pd
import matplotlib.pyplot as plt

# Check for missing values
df2.isnull().sum()

# Checking for any duplicates
df2.duplicated().sum()

# Remove any duplicate rows
df2.drop_duplicates(inplace=True)
```

Key functionalities of the code:

1. **Check for Missing Values**: Identifies and sums up any missing (NaN) values in the DataFrame using df2.isnull().sum().

2. **Check for Duplicates**: Identifies and counts any duplicate rows in the DataFrame using df2.duplicated().sum().
3. **Remove Duplicate Rows**: Removes any duplicate rows from the DataFrame using df2.drop_duplicates().

Top 5 Run Scorers, Top 5 Six Hitters (Sub plots)

```
# Aggregate total runs and sixes by batsmab
top_scores = df2.groupby("batsmanName")["runs"].sum().nlargest(5)
top_six_hitters = df2.groupby("batsmanName")["6s"].sum().nlargest(5)

# Create subplots
fig,axes = plt.subplots(1,2,figsize=(12,5))

# Bar chart for Top 5 Run Scorers
axes[0].bar(top_scores.index,top_scores.values,color="blue",edgecolor="black")
axes[0].set_title("Top 5 Run Scorers")
axes[0].set_xlabel("Batsman")
axes[0].set_ylabel("Total Runs")
axes[0].tick_params(axis="x",rotation=45)

# Bar chart for Top 5 Six Hitters
axes[1].bar(top_six_hitters.index,top_six_hitters.values,color="green",edgecolor="black")
axes[1].set_title("Top 5 Six Scorers")
axes[1].set_xlabel("Batsman")
axes[1].set_ylabel("Total Six")
axes[1].tick_params(axis="x",rotation=45)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

Key functionalities of the code:

1. **Aggregate Total Runs and Sixes**: Groups the data by batsmanName and calculates the total runs (sum()) and total sixes hit (sum()) for each batsman. It then selects the top 5 players using nlargest(5).
2. **Create Subplots**: Sets up two subplots using plt.subplots() for displaying two bar

charts side by side.

3. **Plot Top 5 Run Scorers**: Plots a bar chart for the top 5 run scorers with batsmen on the x-axis and total runs on the y-axis.
4. **Plot Top 5 Six Hitters**: Plots a bar chart for the top 5 six hitters with batsmen on the x-axis and total sixes on the y-axis.
5. **Customize Plot**: Adds titles, axis labels, and rotates the x-axis labels for readability. Adjusts the layout with plt.tight_layout() to ensure proper spacing.
6. **Display Plot**: Displays the combined bar charts using plt.show().

Highest Strike Rate (Horizontal Bar Chart)

```
# Convert 'sr' and 'balls' columns to numeric
df2["SR"]=pd.to_numeric(df2["SR"],errors="coerce") # Convert SR to numeric
df2["balls"]=pd.to_numeric(df2["balls"],errors="coerce") # Convert SR to numeric

# Filter batsman who faced atleast 25 balls
df2_filtered = df2[df2["balls"] >= 50]

# Get top 5 batsmen with highest strike rate
top_strike_rate = df2_filtered.groupby("batsmanName")["SR"].mean().sort_values(ascending=False).head(5)

# Plot horizontal bar chart
plt.figure(figsize=(8,5))
plt.barh(top_strike_rate.index,top_strike_rate.values,color=["blue","green","red","purple","orange"],edgecolor="black")

# Add labels and title
plt.title("Top 5  Batsmen with Highest Strike Rate (Min 50 Balls)")
plt.xlabel("Strike Rate")
plt.ylabel("Batsman")
plt.gca().invert_yaxis() # Invert y-axis to show highest SR at the top
plt.grid(axis="x",linestyle="--",alpha=0.7)

# Show plot
plt.show()
```

Key functionalities of the code:

1. **Convert Columns to Numeric**: Converts the SR (Strike Rate) and balls columns to numeric values using pd.to_numeric(), with any errors coerced into NaN.
2. **Filter Batsmen**: Filters the dataset to include only batsmen who have faced at least 50 balls (df2["balls"] >= 50).
3. **Get Top 5 Batsmen by Strike Rate**: Groups the data by batsmanName, calculates

the mean strike rate for each batsman, and selects the top 5 with the highest strike rates.

4. **Plot Horizontal Bar Chart**: Plots a horizontal bar chart with the top 5 batsmen and their corresponding strike rates.
5. **Customize Plot**: Adds a title, labels for the x and y axes, inverts the y-axis to show the highest strike rate at the top, and adds gridlines to the x-axis.
6. **Display Plot**: Displays the horizontal bar chart using plt.show()

Out vs Not Out Ratio (Pie Chart)

```
out_counts = df2["out/not_out"].value_counts()

plt.figure(figsize=(6,6))
plt.pie(out_counts,labels=out_counts.index,autopct="%1.1f%%",colors=["red","blue"],startangle=90)
plt.title("Out vs Not Out Ratio")
plt.show()
```

Key functionalities of the code:

1. **Count "Out" vs "Not Out"**: Uses value_counts() to count the occurrences of "out" and "not_out" in the out/not_out column.
2. **Create Pie Chart**: Generates a pie chart showing the distribution of "Out" vs "Not Out" using plt.pie(), with labels and percentage values displayed.
3. **Customize Plot**: Adds a title to the pie chart and sets the starting angle for better visual presentation.
4. **Display Plot**: Displays the pie chart using plt.show().

BOWLING SCORE

Data Collection

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset from a CSV file
df3 = pd.read_csv("csv/bowling_score.csv")
df3.head()
```

Key functionalities of the code:

1. **Load CSV Data**: Reads the bowling_score.csv file into a Pandas DataFrame (df3).
2. **Inspect Data**: Displays the first five rows of the dataset using .head() to provide a quick overview

Data Preprocessing

```
import pandas as pd
import matplotlib.pyplot as plt

# Check for missing values
df3.isnull().sum()

# Remove particular row if column as Nan
df3.dropna(subset=['bowlerName','overs','maiden','runs','wickets','economy','0s','4s','6s','wides','noBalls'],inplace=True)

# Checking for any duplicates
df3.duplicated().sum()

# Remove any duplicate rows
df3.drop_duplicates(inplace=True)
```

```
#Convert relevant columns to numeric error to Nan
numeric_cols = ["wickets","economy","wides","noBalls"]
df3[numeric_cols]=df3[numeric_cols].apply(pd.to_numeric,errors="coerce")

# Drop NaN values
df3=df3.dropna(subset=numeric_cols)
```

Key functionalities of the code:

1. **Check for Missing Values**: Identifies and sums up missing (NaN) values in the DataFrame using df3.isnull().sum().
2. **Remove Rows with Missing Data**: Drops rows where key columns (bowlerName, overs, maiden, runs, wickets, economy, etc.) contain NaN values.
3. **Check for Duplicates**: Identifies duplicate rows in the dataset using df3.duplicated().sum().
4. **Remove Duplicate Rows**: Deletes duplicate rows using df3.drop_duplicates(inplace=True).
5. **Convert Columns to Numeric**: Converts wickets, economy, wides, and noBalls to numeric format, coercing errors to NaN.

6. **Drop Remaining NaN Values**: Removes rows where the specified numeric columns still contain NaN values after conversion.

TOP 5 Wicket-Takers (Bar Chart), Top 5 Most Economical Bowlers (Bar Chart), Wides & No-Balls Distribution (Scatter Plot)

```
# Aggregate total wickets by bowler
top_wickets_takers = df3.groupby("bowlerName")["wickets"].sum().nlargest(5)

# Aggregate economy rates (lowest 5 best economy)
top_economy_bowlers = df3.groupby("bowlerName")["economy"].mean().nsmallest(5)

# Scatter plot for wides vs no-balls
wides_noballs = df3.groupby("bowlerName")[["wides","noBalls"]].sum()

# Create subplots (1 row,3 column)
fig,axes = plt.subplots(1,3,figsize=(15,3))

# Bar Chart: Top 5 Wicket-Takers
axes[0].bar(top_wickets_takers.index,top_wickets_takers.values,color="blue",edgecolor="black")
axes[0].set_title("Top 5 Wicket-Takers")
axes[0].set_xlabel("Bowler")
axes[0].set_ylabel("Total Wickets")
axes[0].tick_params(axis="x",rotation = 45)

# Bar Chart: Top 5 Most Economical Bowlers
axes[1].bar(top_economy_bowlers.index,top_economy_bowlers.values,color="green",edgecolor="black")
axes[1].set_title("Top 5 Most Economical Bowlers")
axes[1].set_xlabel("Bowler")
axes[1].set_ylabel("Economy Rate")
axes[1].tick_params(axis="x",rotation = 45)

# Scatter Plot: Wides vs No Balls
axes[2].scatter(wides_noballs["wides"],wides_noballs["noBalls"],color="red")
axes[2].set_title("Wides vs. No-Balls")
axes[2].set_xlabel("Wides")
axes[2].set_ylabel("No-Balls")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

Key functionalities of the code:

1. **Aggregate Total Wickets**: Groups data by bowlerName and sums up wickets to identify the top 5 wicket-takers.
2. **Aggregate Economy Rates**: Calculates the average economy rate for each bowler and selects the top 5 with the lowest economy rates.

3. **Summarize Wides and No-Balls**: Groups data by bowlerName and sums up wides and no-balls for analysis.
4. **Create Subplots**: Generates a 1-row, 3-column subplot layout to visualize different metrics.
5. **Plot Top 5 Wicket-Takers**: Displays a bar chart of the top 5 bowlers with the most wickets.
6. **Plot Most Economical Bowlers**: Displays a bar chart of the top 5 bowlers with the lowest economy rates.
7. **Scatter Plot for Wides vs. No-Balls**: Plots a scatter chart comparing wides and no-balls for bowlers.
8. **Customize and Display Plots**: Adjusts layout, adds titles, axis labels, and rotates x-axis labels for clarity before displaying the plots.

Wickets Distribution(Histogram)

```
plt.figure(figsize=(8,5))
plt.hist(df3["wickets"],bins=10,color="purple",edgecolor="black",alpha=0.7)

# Add labels and title
plt.title("Distribution of Wickets Taken by Bowlers")
plt.xlabel("Number of Wickets")
plt.ylabel("Frequency(Number of Bowlers)")
plt.grid(axis="y",linestyle="--",alpha=0.7)

# Show plot
plt.show()
```

Key functionalities of the code:

1. **Plot Histogram**: Creates a histogram to visualize the distribution of wickets taken by bowlers.
2. **Set Number of Bins**: Divides the data into 10 bins for better representation of frequency distribution.
3. **Customize Appearance**: Uses purple color, black edges, and adjusts transparency (alpha=0.7) for better visibility.

4. **Add Labels and Title**: Sets a title and labels for the x-axis (Number of Wickets) and y-axis (Frequency (Number of Bowlers)).
5. **Enable Grid**: Adds grid lines along the y-axis for better readability.
6. **Display Plot**: Shows the histogram using plt.show().

Best Playing XI Dashboards (Power BI)

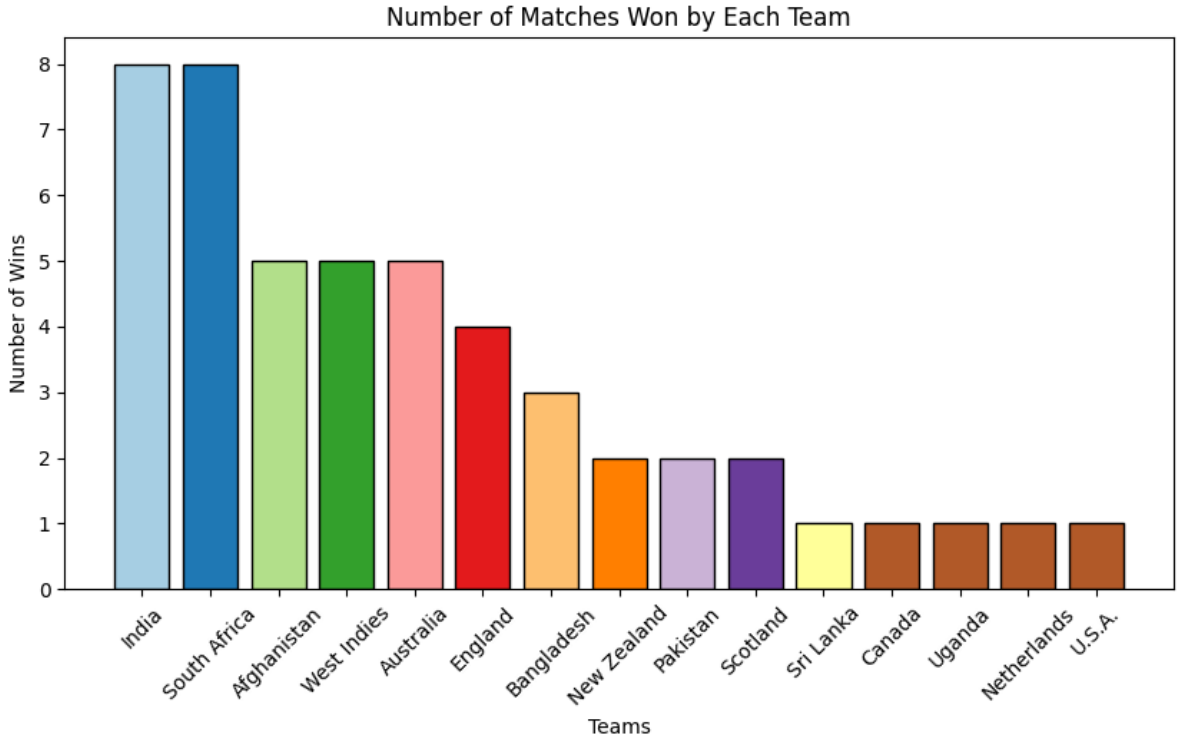
1. Power Hitter/ Openers Page
2. Anchors/ Middle Order Page
3. Finisher/ Lower Order Anchor Page
4. All Rounders/ Lower Middle Order Page
5. Specialist Fast Bowlers/ Tail End Page
6. Best Playing XI Page

Key Functionalities of Best Playing XI Dashboards (Power BI)

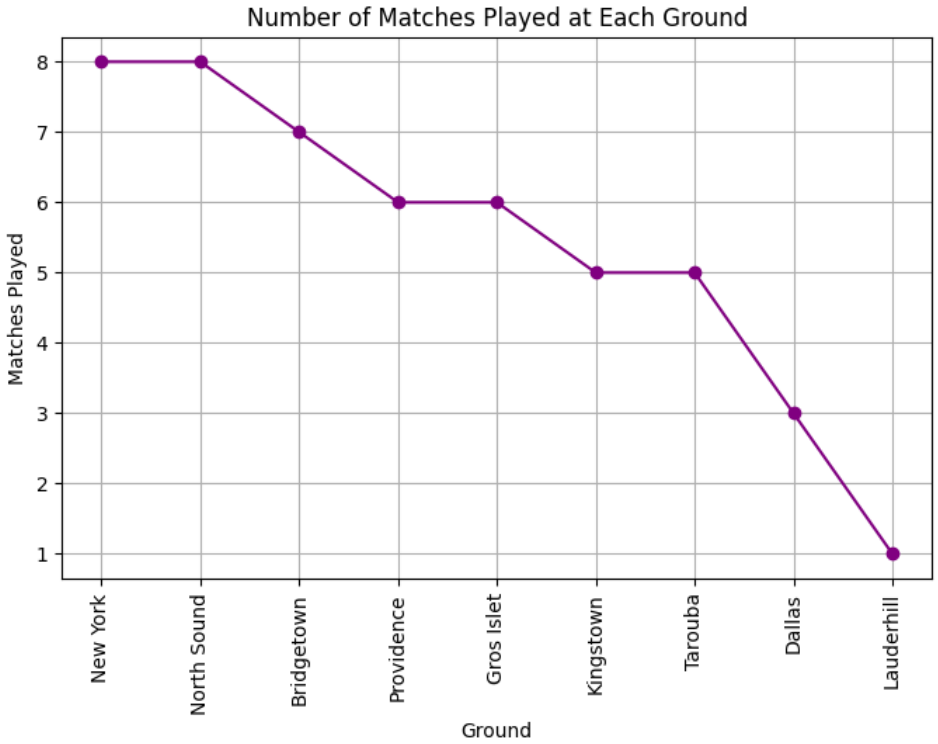
1. **Power Hitter/Openers Page**
   - Identifies explosive batsmen with high strike rates and boundary-hitting ability.
   - Compares performance metrics like runs, strike rate, and powerplay efficiency.
2. **Anchors/Middle Order Page**
   - Highlights consistent run-scorers who stabilize the innings.
   - Analyzes average, balls faced per innings, and partnerships.
3. **Finisher/Lower Order Anchor Page**
   - Showcases players who excel in death overs with high strike rates.
   - Evaluates finishing ability using death-over statistics and pressure performance.
4. **All-Rounders/Lower Middle Order Page**
   - Displays all-rounders contributing with both bat and ball.
   - Tracks runs, wickets, economy rate, and impact in crucial phases.
5. **Specialist Fast Bowlers/Tail End Page**
   - Focuses on fast bowlers' wicket-taking ability and economy rates.
   - Analyzes pace, swing, and death-over bowling effectiveness.
6. **Best Playing XI Page**
   - Combines the best performers from all categories to form the ideal XI.
   - Uses weighted metrics to optimize team balance and selection.

10. Output

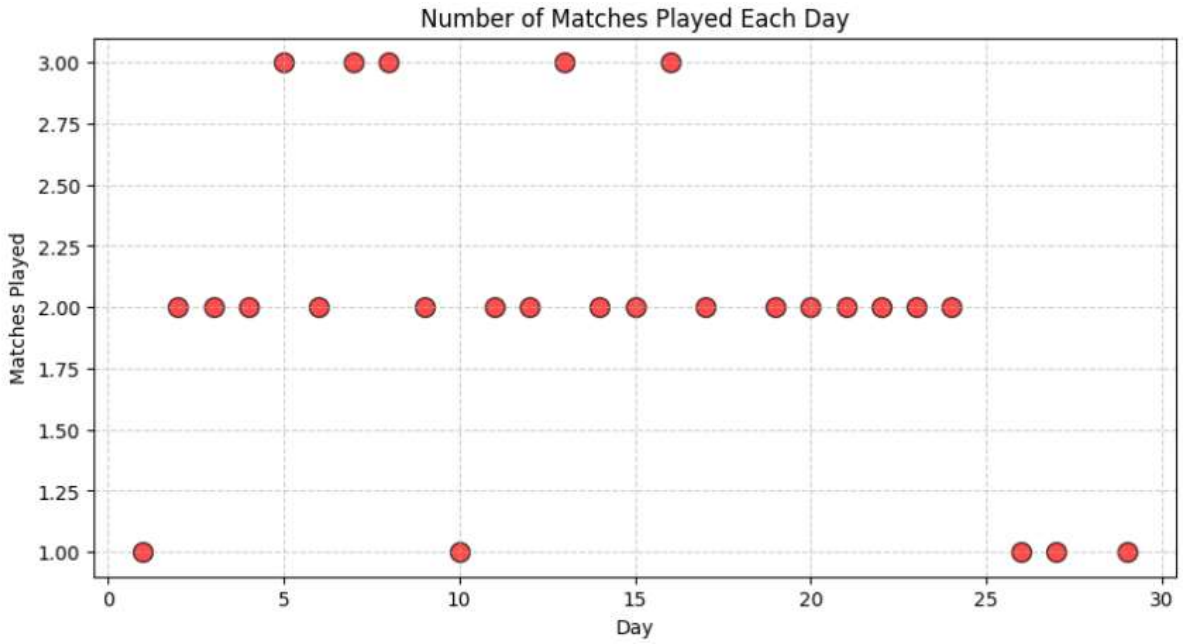
- Matches Won by Each Team (Bar Chart)



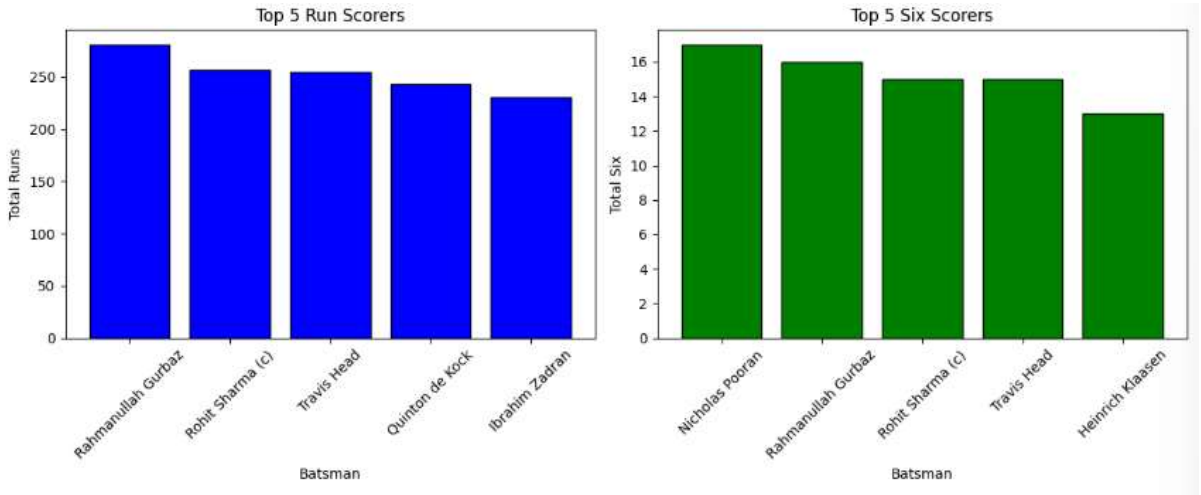
- Wins by Ground (Pie Chart)



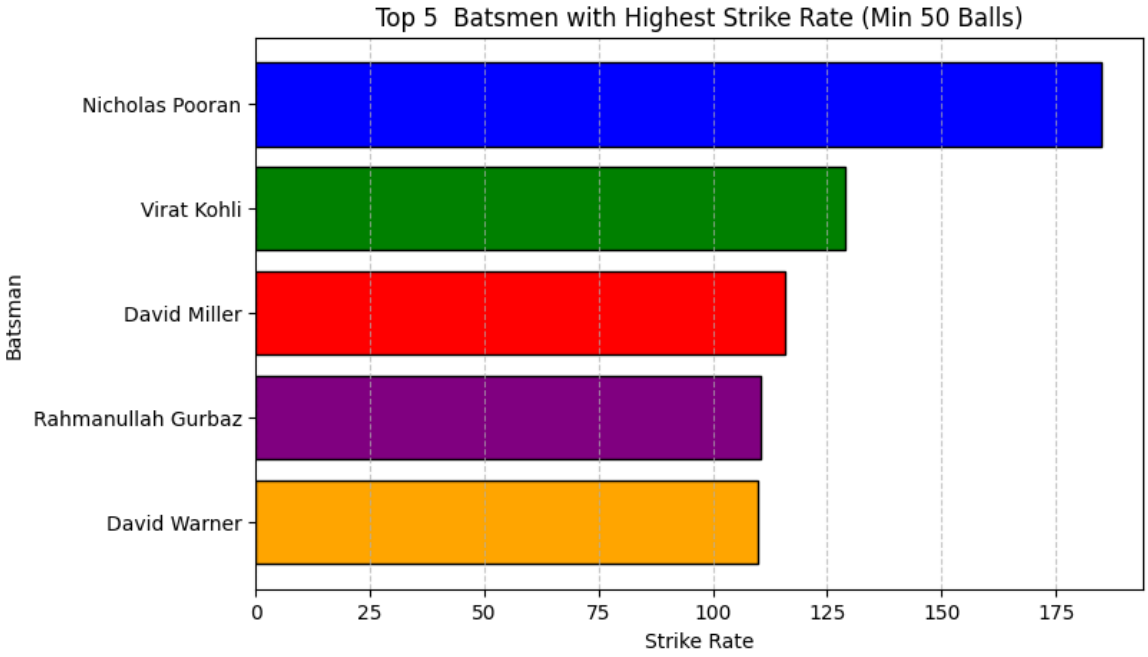
- Wins Over Day (Scatter Chart)



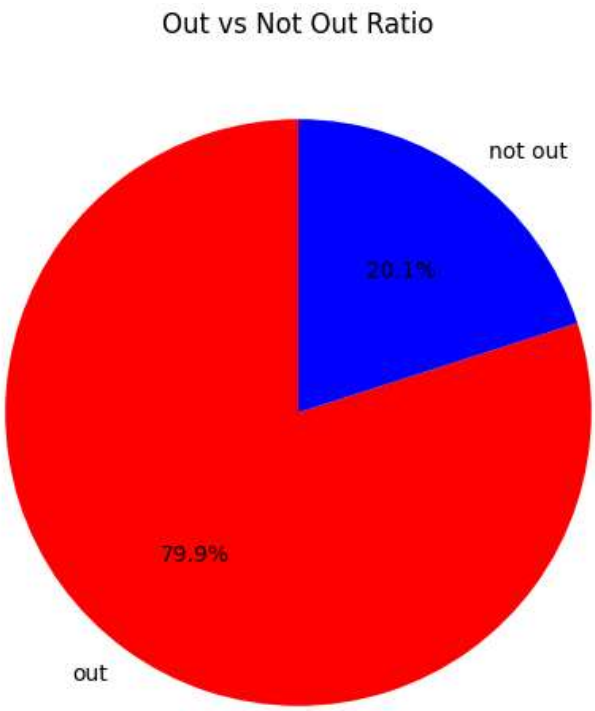
- Top 5 Run Scorers, Top 5 Six Hitters (Subplots)



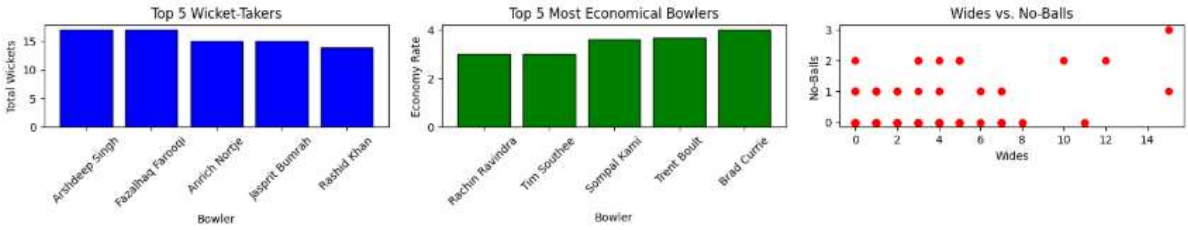
- Highest Strike Rate (Horizontal Bar Chart)



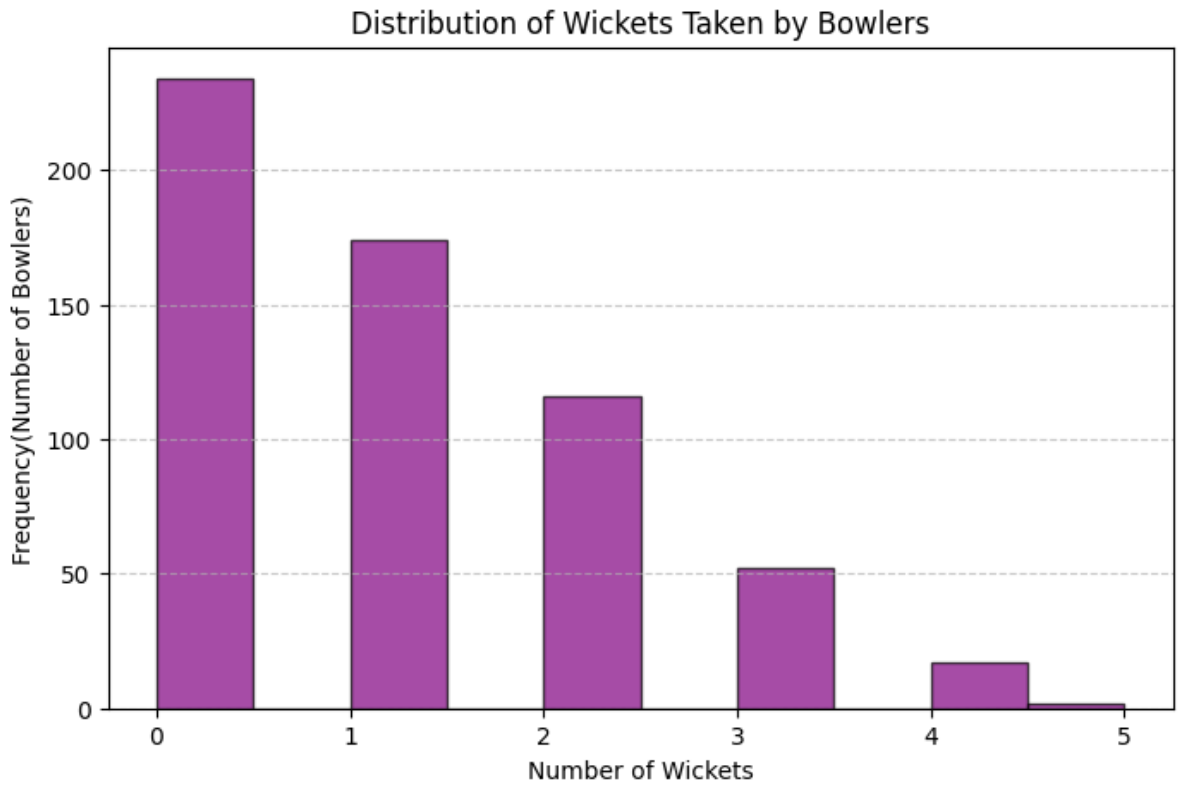
- Out vs Not Out Ratio (Pie Chart)



- Subplots (Bar chart, Scatter Chart)



- Wickets Distribution (Histogram)



Best Playing XI Dashboards (Power BI)

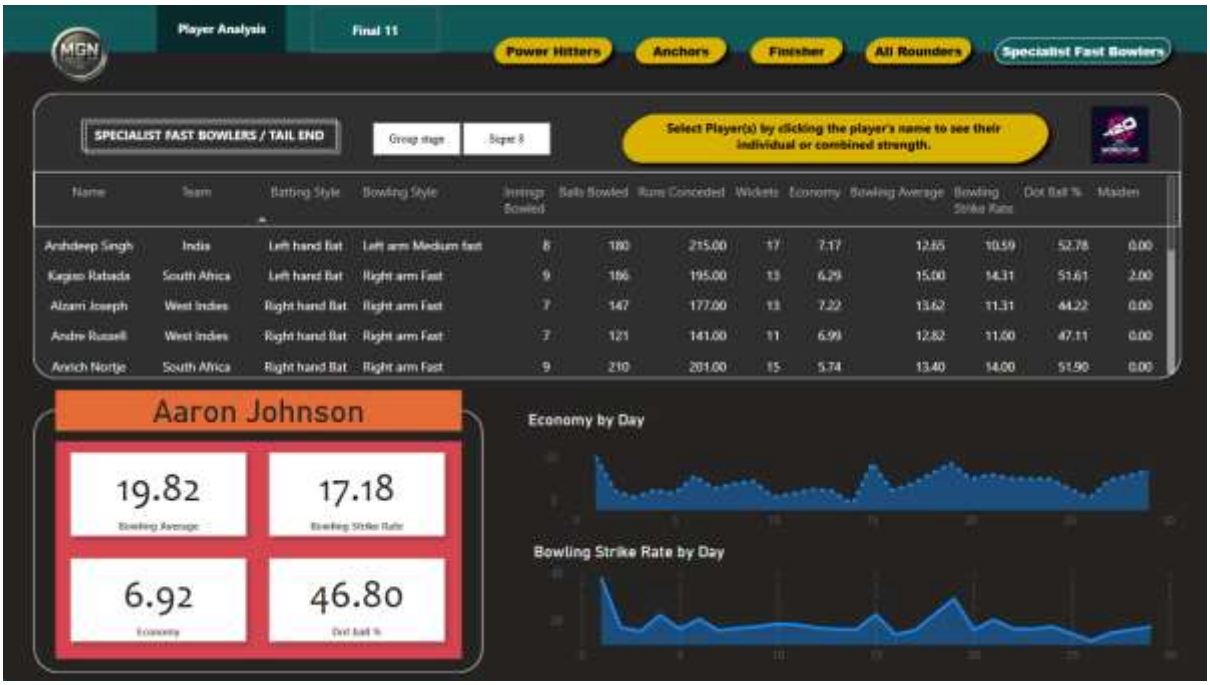
7. Power Hitter/ Openers Page

8. Anchors/ Middle Order Page

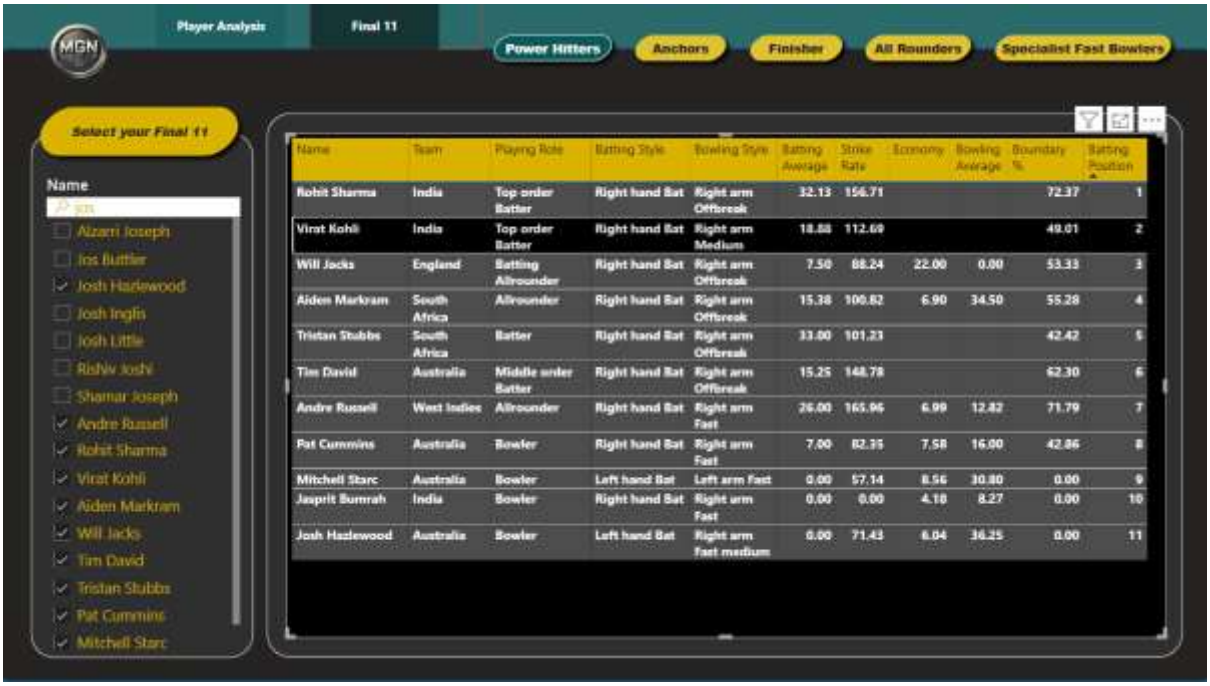
9. Finisher/ Lower Order Anchor Page

10. All Rounders/ Lower Middle Order Page

11. Specialist Fast Bowlers/ Tail End Page



12. Best Playing XI Page



11. Performance Analysis

- **Top Performers Identification** – Successfully analyzed batting and bowling data to identify key players based on strike rate, economy, and consistency.
- **Data-Driven Decision Making** – Utilized statistical models and Power BI dashboards to select the Best Playing XI dynamically.
- **Insightful Visualizations** – Created bar charts, scatter plots, and histograms to highlight trends in player performance and match outcomes.
- **Optimized Team Selection** – Ensured a balanced team composition by evaluating metrics such as boundary percentage, dot ball rate, and wicket-taking ability.

12. Challenges & Future Enhancements

Challenges:

- **Data Quality Issues** – Handling missing values, inconsistent formats, and outliers required extensive preprocessing.
- **Dynamic Performance Variations** – Player form fluctuates, making it difficult to maintain an optimal Best Playing XI.
- **Complexity in Metrics Calculation** – Combining multiple performance parameters to create meaningful rankings was challenging.
- **Limited Real-Time Data Access** – Dependency on pre-existing datasets restricted real-time analysis and live match insights.

Future Enhancements:

- **Integration with Live Data Feeds** – Incorporating real-time match updates for more accurate player performance tracking.
- **Advanced Predictive Analytics** – Using predictive models to forecast player and team performances based on historical trends.
- **Enhanced Visualization & UI** – Improving Power BI dashboards with more interactive and user-friendly visual elements.

- **Automated Team Selection** – Developing an AI-powered recommendation system for selecting the Best Playing XI based on live statistics.

13. Conclusion

This project successfully analyzes cricket performance data using **Pandas, Matplotlib, and Power BI**, providing **valuable insights into player and team performance**. By visualizing key statistics and trends, it enables data-driven decision-making for selecting the **Best Playing XI**. The use of **interactive dashboards** enhances user engagement, making performance evaluation more efficient. While challenges such as data quality and real-time updates exist, future enhancements like **live data integration and predictive analytics** can further refine insights. Overall, this project demonstrates the power of **data-driven cricket analysis** without relying on machine learning.

14. References

ESPNcricinfo. (n.d.). Cricket Data. Retrieved from https://www.espncricinfo.com/

McKinney, W. (2018). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media. ISBN: 978-1491957660.

Microsoft. (2021). Power BI Documentation. Microsoft. Retrieved from https://docs.microsoft.com/en-us/power-bi/