

Importing Data

Importing Libraries

```
In [1]: %pip install -q langchain_experimental langchain_openai
```

```
In [2]: from google.colab import drive
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from langchain.agents.agent_types import AgentType
from langchain_experimental.agents.agent_toolkits import create_pandas_data_loader
from langchain_openai import ChatOpenAI, OpenAI
import os
import openai
```

Background

The Border–Gavaskar Trophy is an International Test cricket trophy played between India and Australia. The series is named after distinguished former captains, Australia's Allan Border and India's Sunil Gavaskar. Introduced in 1996, the BGT has grown into one of the most prestigious bilateral Test series in cricket, showcasing the rich history and competitive rivalry between these two cricketing powerhouses.

Prestige in Test Cricket

Test cricket is considered the pinnacle format of the sport, and the BGT encapsulates the true spirit of the game, emphasizing patience, strategy, and resilience over a span of five days. India and Australia have shared a long-standing competitive spirit. Matches are often intense, reflecting the teams' quality, skills, and their desire to dominate the longest format of the game.

Motivation for the Analysis

Despite cricket being one of the most popular sports globally, there is a lack of proper exploratory data analysis (EDA) and data-driven insights available for cricket, particularly in the Test format. Unlike sports like football and basketball, which have a wealth of publicly available advanced analytics, cricket analysis has often remained a step behind, focusing only on traditional metrics.

This project aims to bridge that gap by providing a comprehensive data analysis of the Border-Gavaskar Trophy. By analyzing ball-by-ball data from multiple BGT series, it will uncover patterns, trends, and key performance indicators. It will highlight factors that influence match outcomes—such as player contributions, pitch conditions, and match situations—and provide cricket enthusiasts, analysts, and coaches with actionable insights backed by data.

Mounting Google Drive to upload the files

```
In [3]: # Mount Google Drive
drive.mount('/content/drive')
folder_path = '/content/drive/MyDrive/BGT_TEST_data/'

# List of file paths in your Google Drive folder
# 16 Games
file_paths = [
    f"{folder_path}BGT_2017_02:23.csv",
    f"{folder_path}BGT_2017_03:04.csv",
    f"{folder_path}BGT_2017_03:16.csv",
    f"{folder_path}BGT_2017_03:25.csv",
    f"{folder_path}BGT_2018_12:06.csv",
    f"{folder_path}BGT_2018_12:14.csv",
    f"{folder_path}BGT_2018_12:26.csv",
    f"{folder_path}BGT_2019_01:03.csv",
    f"{folder_path}BGT_2020_12:17.csv",
    f"{folder_path}BGT_2020_12:26.csv",
    f"{folder_path}BGT_2021_01:07.csv",
    f"{folder_path}BGT_2021_01:15.csv",
    f"{folder_path}BGT_2023_02:09.csv",
    f"{folder_path}BGT_2023_02:17.csv",
    f"{folder_path}BGT_2023_03:01.csv",
    f"{folder_path}BGT_2023_03:09.csv"
]
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
In [4]: folder_path = "/content/drive/MyDrive/BGT_TEST_data/"
bgt_results = pd.read_csv(f"{folder_path}BGTresults.csv")
bgt_results.head()
```

Out [4]:

	Test Series Year	Test Number	Innings	Venue	Host	Highest Scorer	Team	Runs by highest scorer	best bowler
0	1996-97	1	1	Delhi	India	Michael Slater	Australia	44	Anil Kumble
1	1996-97	1	2	Delhi	India	Nayan Mongia	India	152	Paul Reiffel
2	1996-97	1	3	Delhi	India	Steve Waugh	Australia	67	Anil Kumble
3	1996-97	1	4	Delhi	India	Mohd. Azharuddin	India	21	Paul Reiffel
4	1997-98	1	1	Chennai	India	Navjot Sidhu	India	62	Shane Warne

Loading match files into the single DataFrame

```
In [5]: # Load all files into a single DataFrame
ball_data = pd.concat([pd.read_csv(file) for file in file_paths], ignore_index=True)
ball_data.head()
```

Out [5]:

	match_id	season	start_date	venue	innings	ball	batting_team	bowling
--	----------	--------	------------	-------	---------	------	--------------	---------

0	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.1	Australia
1	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.2	Australia
2	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.3	Australia
3	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.4	Australia
4	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.5	Australia

5 rows x 22 columns

```
In [6]: ball_data.to_csv('ball_data.csv', index=False)
```

Data Pre-processing

Calculating best batsman and bowler

Identifying the best batsman and bowler helps highlight players who had the greatest impact on the series. Batsmen anchor innings and build competitive totals, while bowlers break partnerships and restrict opposition runs—both critical for determining match outcomes.

This analysis provides insights into individual performances, helping teams and analysts recognize key contributors. Ultimately, it showcases player influence and sets the stage for deeper performance analysis.

```
In [7]: # Calculate Best Batsman
best_batsman = ball_data.groupby('striker')['runs_off_bat'].sum().sort_va
best_batsman.columns = ['Batsman', 'Total Runs']
best_batsman.head()
```

Out [7]:

	Batsman	Total Runs
--	---------	------------

0	CA Pujara	1337
1	SPD Smith	957
2	M Labuschagne	708
3	V Kohli	703
4	AM Rahane	683

```
In [8]: # Calculate Best Bowler
best_bowler = ball_data.groupby('bowler')['player_dismissed'].count().sor
best_bowler.columns = ['Bowler', 'Total Wickets']
best_bowler.head()
```

Out [8]:

	Bowler	Total Wickets
--	--------	---------------

0	NM Lyon	75
1	R Ashwin	67
2	RA Jadeja	62
3	PJ Cummins	48
4	JR Hazlewood	40

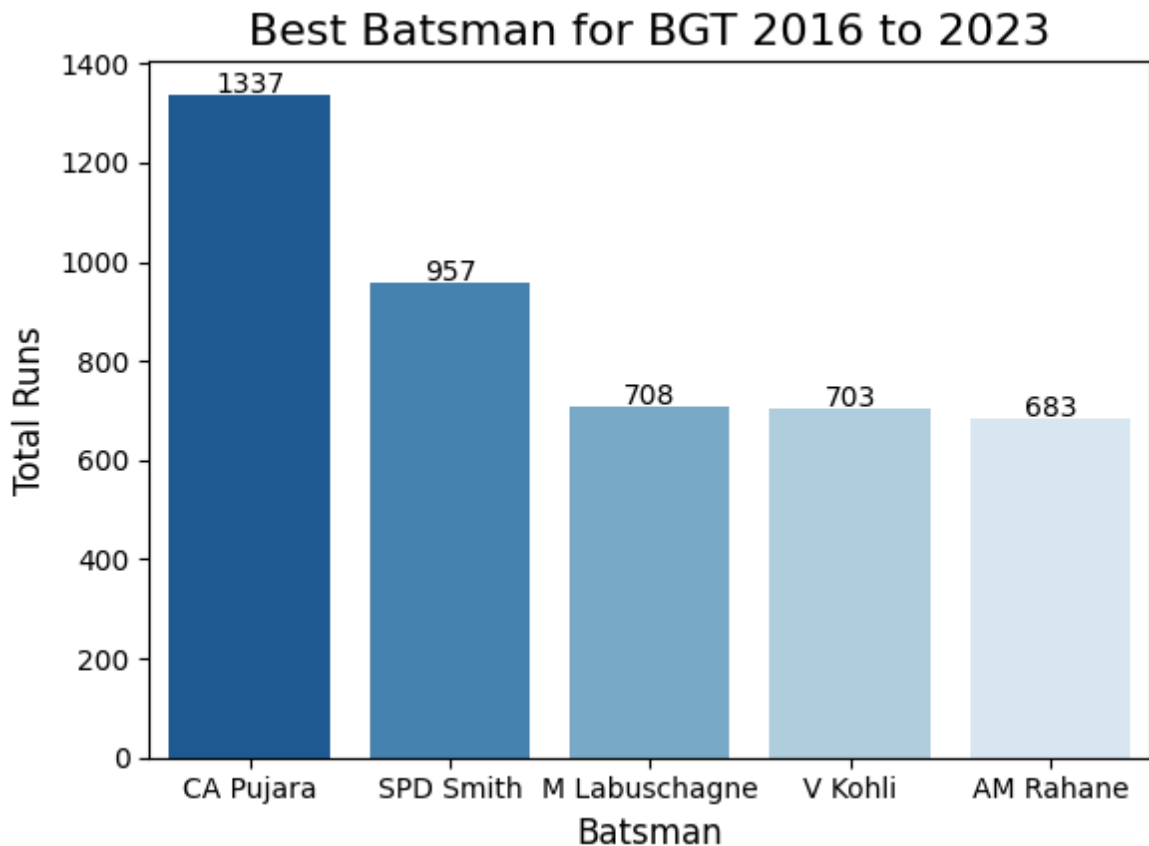
Plotting both the best batsman and bowler based on the calculations.

```
In [9]: plt.figure(figsize=(6, 4.5))
sns.barplot(x='Batsman', y='Total Runs', data=best_batsman.head(), palett
plt.title('Best Batsman for BGT 2016 to 2023', fontsize=16)
plt.ylabel('Total Runs', fontsize=12)
plt.xlabel('Batsman', fontsize=12)

for index, row in best_batsman.head().iterrows():
```

```
plt.text(index, row['Total Runs'] + 5, f'{row["Total Runs"]}',
        ha='center', fontsize=10, color='black')

plt.tight_layout()
plt.show()
```

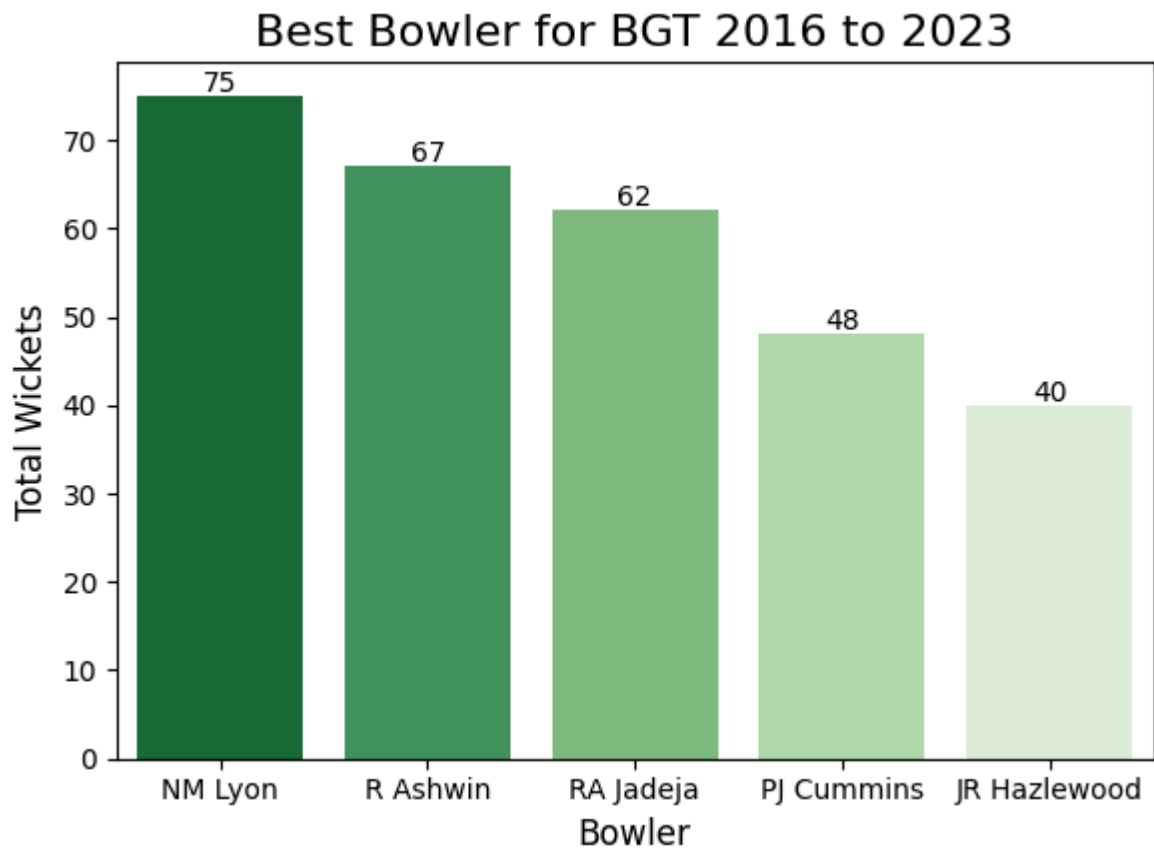


- The bar chart below visualizes the top 5 batsmen in terms of total runs scored in the IND vs AUS Test series from 2016/2017 to the present.
- Cheteshwar Pujara's consistency and grit have made him the highest run scorer showcasing his dominance in the series from 2016-2023.
- Australia's dominance in the top five, with Smith and Labuschagne, underscores their strong batting lineup. India's top performers, Kohli and Rahane, have also made significant contributions.
- By analyzing the performance of players over multiple series, we can identify emerging trends and patterns in their performance.

```
In [10]: plt.figure(figsize=(6, 4.5))
sns.barplot(x='Bowler', y='Total Wickets', data=best_bowler.head(), palette=
plt.title('Best Bowler for BGT 2016 to 2023', fontsize=16)
plt.ylabel('Total Wickets', fontsize=12)
plt.xlabel('Bowler', fontsize=12)

for index, row in best_bowler.head().iterrows():
    plt.text(index, row['Total Wickets'] + 0.5, f'{row["Total Wickets"]}',
            ha='center', fontsize=10, color='black')
```

```
plt.tight_layout()  
plt.show()
```



- The bar chart below visualizes the top 5 bowlers in terms of total wickets taken in the IND vs AUS Test series from 2016/2017 to 2023.
- Nathan Lyon stands out as the leading wicket-taker with 75 wickets, showcasing his dominance as Australia's premier spinner in the series.
- India's spin duo, R Ashwin (67 wickets) and RA Jadeja (62 wickets), highlight India's spin strength, particularly in conditions favorable to spinners, whereas Australia's pace attack is also well-represented with PJ Cummins (48 wickets) and JR Hazlewood (40 wickets), underlining their effectiveness in seam-friendly conditions.
- By analyzing the performance of bowlers over multiple series, we can identify key contributors and understand trends in bowling success across conditions and match situations.

Cricket's Cat and Mouse: The Best Bowler-Victim Battles

Analyzing the top bowler-victim pairs/battles helps uncover dominant individual matchups, where specific bowlers consistently dismiss certain batsmen. For

instance, identifying that NM Lyon has dismissed CA Pujara 9 times highlights his consistent success against one of India's key batsmen. Such insights reveal key player dynamics that can influence match strategies.

```
In [11]: # Group by bowler and player dismissed to count the number of dismissals
bowler_victim_pairs = ball_data.groupby(['bowler', 'player_dismissed']).a
    dismissals=('player_dismissed', 'count')
).reset_index()

top_bowler_victim_pairs = bowler_victim_pairs.sort_values(by='dismissals'

# Create a combined label column for plotting
top_bowler_victim_pairs['bowler_victim'] = top_bowler_victim_pairs.apply(
    lambda row: f"{row['bowler']} -> {row['player_dismissed']}",
    axis=1
)
top_bowler_victim_pairs.head()
```

```
Out[11]:
```

	bowler	player_dismissed	dismissals	bowler_victim
0	NM Lyon	CA Pujara	9	NM Lyon -> CA Pujara
1	PJ Cummins	CA Pujara	8	PJ Cummins -> CA Pujara
2	R Ashwin	SPD Smith	7	R Ashwin -> SPD Smith
3	RA Jadeja	PJ Cummins	7	RA Jadeja -> PJ Cummins
4	NM Lyon	AM Rahane	7	NM Lyon -> AM Rahane

```
In [12]: # Visualization of Top 10 Bowler-Victim Pairs
plt.figure(figsize=(10, 6))
ax = sns.barplot(
    data=top_bowler_victim_pairs,
    x='dismissals',
    y='bowler_victim',
    palette='coolwarm'
)

plt.title('Top 10 Most Successful Bowler-Victim Pairs', fontsize=16)
plt.xlabel('Number of Dismissals', fontsize=12)
plt.ylabel('Bowler -> Batsman', fontsize=12)

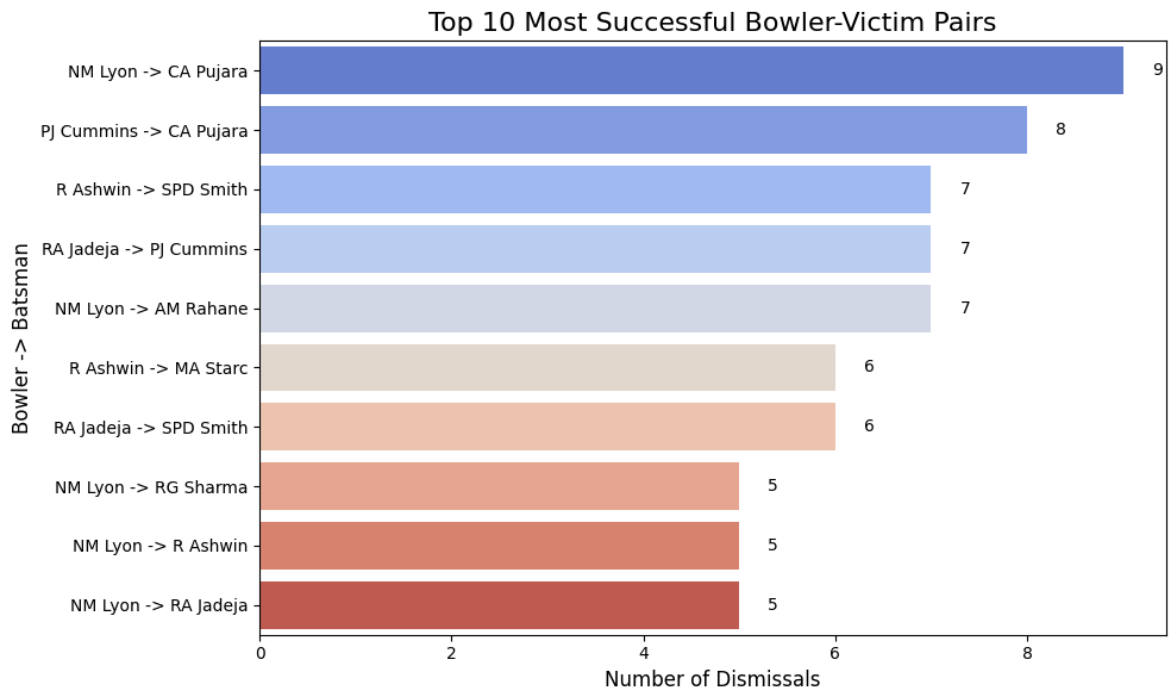
for i, bar in enumerate(ax.patches):
    ax.text(
        x=bar.get_width() + 0.3,
        y=bar.get_y() + bar.get_height()/2,
        s=str(top_bowler_victim_pairs['dismissals'].iloc[i]),
        va='center',
        ha='left',
        color='black',
        fontsize=10
    )

plt.tight_layout()
plt.show()
```

```
<ipython-input-12-6ef8b7c27bcf>:3: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
ax = sns.barplot(
```



- The bar chart visualizes the top 10 bowler-batsman pairings with the highest number of dismissals in the IND vs AUS Test series from 2016/2017 to the present.
- Nathan Lyon's dominance against Cheteshwar Pujara is evident, with the Australian spinner dismissing the Indian batsman 9 times. Pat Cummins' success against Pujara is also highlighted, with 8 dismissals to his name.
- The graph also reveals the frequent clashes between Indian spinners (Jadeja and Ashwin) and Australian batsmen (Smith, Cummins, and Starc).
- The data underscores the importance of bowler-batsman matchups in Test cricket and how certain bowlers can exploit the weaknesses of specific batsmen.

Building Blocks of Success: The Best Partnerships

Analyzing the best partnerships helps identify the most successful batting pairs in terms of runs scored and balls faced. Partnerships play a crucial role in building or stabilizing an innings, especially in Test cricket, where long and consistent batting collaborations are essential for a team's success. From a strategic perspective, this analysis provides valuable insights for teams and analysts. Knowing which pairs

thrive in partnerships helps identify reliable players for key positions, enabling better decision-making during team selection and match planning

```
In [13]: # Analyze Partnerships Series-Wide
# Group data by partnership (striker and non_striker)
partnerships = ball_data.groupby(['striker', 'non_striker']).agg(
    partnership_runs=('runs_off_bat', 'sum'),
    balls_faced=('ball', 'count')
).reset_index()

top_partnerships = partnerships.sort_values(by='partnership_runs', ascending=False)

top_partnerships['partnership_label'] = top_partnerships.apply(lambda row: f'{row.striker} & {row.non_striker}', axis=1)
top_partnerships.head()
```

```
Out[13]:
```

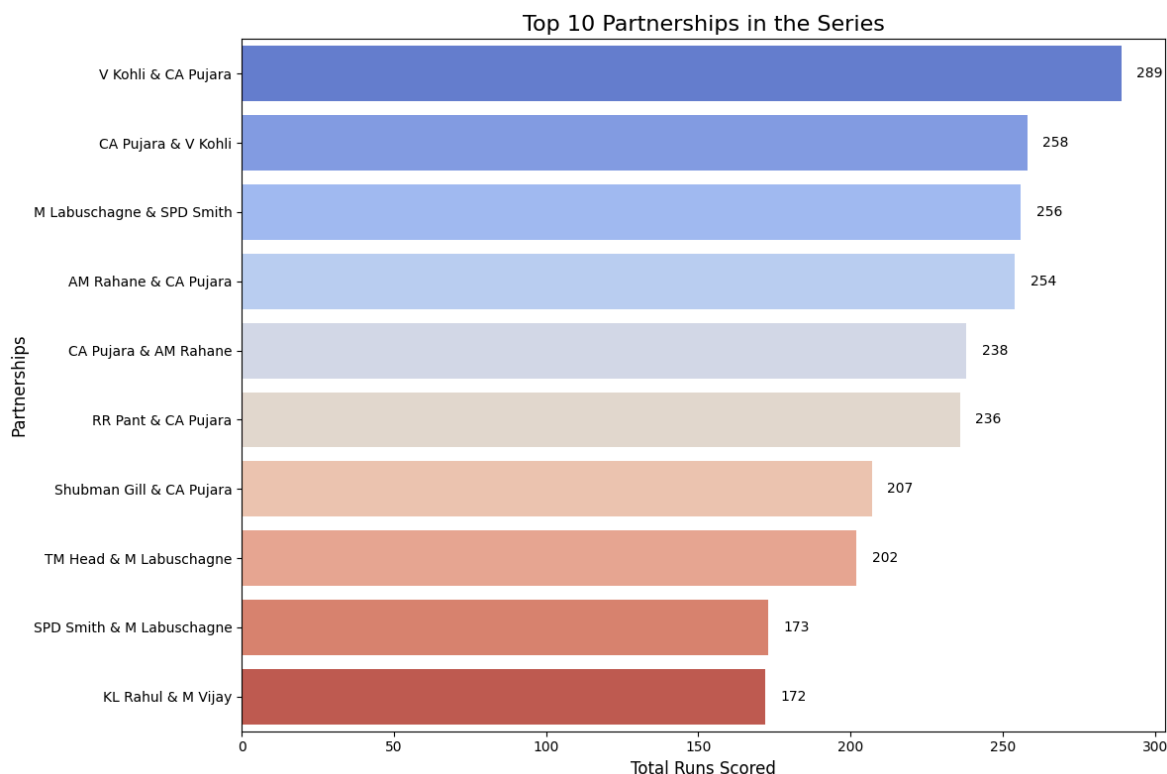
	striker	non_striker	partnership_runs	balls_faced	partnership_label
0	V Kohli	CA Pujara	289	731	V Kohli & CA Pujara
1	CA Pujara	V Kohli	258	712	CA Pujara & V Kohli
2	M Labuschagne	SPD Smith	256	539	M Labuschagne & SPD Smith
3	AM Rahane	CA Pujara	254	600	AM Rahane & CA Pujara
4	CA Pujara	AM Rahane	238	618	CA Pujara & AM Rahane

```
In [14]: # Visualize Top Partnerships
plt.figure(figsize=(12, 8))
ax = sns.barplot(
    data=top_partnerships,
    x='partnership_runs',
    y='partnership_label',
    hue='partnership_label',
    palette='coolwarm'
)

plt.title('Top 10 Partnerships in the Series', fontsize=16)
plt.xlabel('Total Runs Scored', fontsize=12)
plt.ylabel('Partnerships', fontsize=12)

for i, bar in enumerate(ax.patches):
    runs = top_partnerships['partnership_runs'].iloc[i]
    ax.text(
        x=bar.get_width() + 5,
        y=bar.get_y() + bar.get_height()/2,
        s=str(int(runs)),
        va='center',
        ha='left',
        color='black',
        fontsize=10
    )
```

```
plt.tight_layout()
plt.show()
```



- Cheteshwar Pujara appears in 6 out of the top 10 partnerships, highlighting his consistency and ability to build long innings and form crucial partnerships.
- Indian partnerships dominate the chart, with standout contributions from duos like Kohli & Pujara (289 runs) and Pant & Pujara (236 runs), highlighting India's depth and consistency in building strong innings.
- Key Australian partnerships, such as Labuschagne & Smith (173 runs) and Labuschagne & Head (202 runs), demonstrate their ability to counter India's dominance and adapt to challenging conditions.
- Overall, the graph highlights the importance of partnerships in Test cricket. The ability to build long and consistent partnerships can be decisive in the outcome of matches.

Player Preferences: A Venue-Based Analysis

Analyzing venue-based player preferences helps identify which batsmen perform best at specific grounds. This analysis provides valuable insights into player consistency and adaptability across different venues, which is especially critical in high-stakes Test series like the Border-Gavaskar Trophy. This information is highly valuable for teams, coaches, and analysts as it helps in strategizing and planning. Teams can anticipate which players are likely to perform well at certain venues and adjust their tactics accordingly. For example, bowlers can focus on strategies to

contain batsmen at their favorite grounds, while batsmen can leverage their confidence and familiarity with those conditions to anchor the innings.

```
In [15]: venue_data = ball_data
venue_data['player_nationality'] = venue_data['batting_team'].apply(
    lambda x: 'India' if 'India' in x else 'Australia' if 'Australia' in
)

filtered_data = venue_data[venue_data['player_nationality'].isin(['India'

grouped = filtered_data.groupby(['striker', 'venue']).agg(
    total_runs=('runs_off_bat', 'sum')
).reset_index()

top_batsmen = (
    grouped
    .groupby('striker')['total_runs']
    .sum()
    .nlargest(10)
    .index
)

favorite_venues = grouped[grouped['striker'].isin(top_batsmen)]

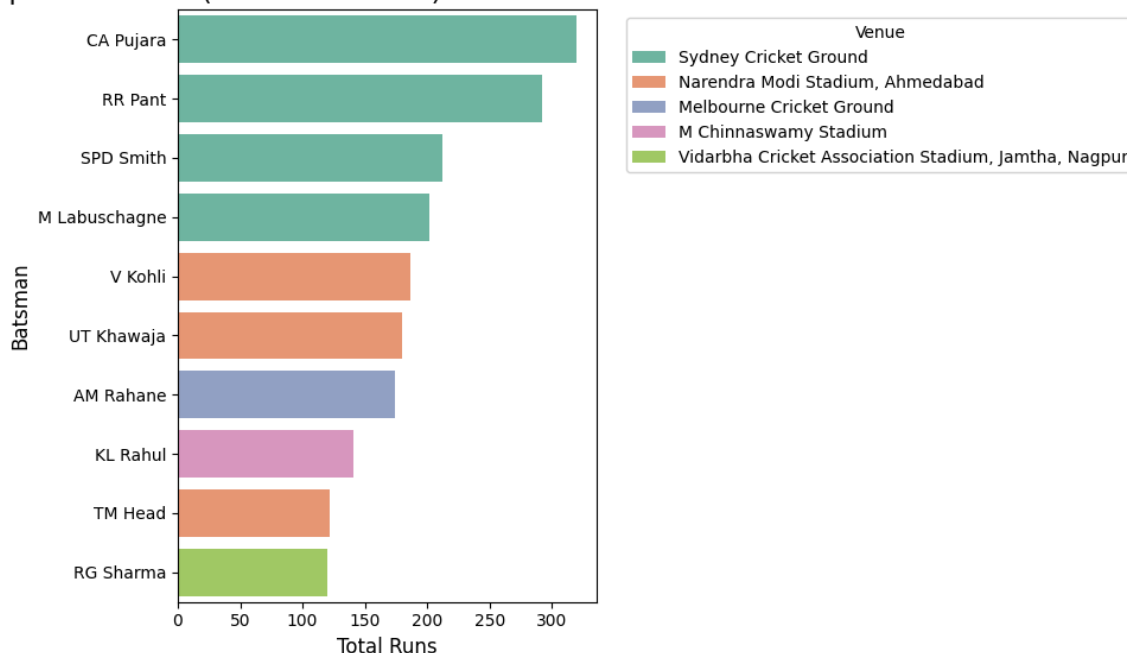
favorite_venues = (
    favorite_venues
    .sort_values(['striker', 'total_runs'], ascending=[True, False])
    .drop_duplicates('striker')
)

favorite_venues_sorted = favorite_venues.sort_values('total_runs', ascend
```

```
In [16]: plt.figure(figsize=(10, 6))
sns.barplot(
    data=favorite_venues_sorted,
    x='total_runs',
    y='striker',
    hue='venue',
    dodge=False,
    palette='Set2'
)

plt.title('Top 10 Batsmen (India & Australia) and Their Favorite Venues',
plt.xlabel('Total Runs', fontsize=12)
plt.ylabel('Batsman', fontsize=12)
plt.legend(title='Venue', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

Top 10 Batsmen (India & Australia) and Their Favorite Venues



- Cheteshwar Pujara's strong performance at the Sydney Cricket Ground, traditionally a venue that often offers slow and turning conditions, suggests his ability to thrive on pitches that resemble Indian conditions.
- Steve Smith and Marnus Labuschagne, two of Australia's top batsmen, have shown a preference for the Vidarbha Cricket Association Stadium in Jamtha, Nagpur. This suggests that the pitch conditions at this venue favor their batting style.
- Rishabh Pant's preference for the Narendra Modi Stadium in Ahmedabad and the M Chinnaswamy Stadium in Bengaluru highlights his dominance in Indian conditions. His aggressive batting style and ability to adapt to different pitches have contributed to his success.
- Analyzing the top 10 batsmen and their favorite venues helps identify player strengths, understand pitch conditions, and inform team selections. It provides valuable insights for both analysts and fans, enhancing understanding and stimulating discussion.

Unlocking the Venue Secrets: A Batsman's Perspective

Analyzing the most runs scored per venue helps identify the batsmen who excel at specific grounds and the venues where they perform at their best. Test cricket venues often differ significantly in conditions, such as pitch behavior, weather, and ground dimensions, which can influence player performances. By grouping data by venue and identifying the batsmen with the highest total runs, we can determine their

consistency and adaptability across different locations. Examining venue-based batting performances enhances our understanding of player consistency, highlights their strengths in different conditions, and adds a venue-specific dimension to the analysis of Test match performances.

```
In [17]: # Calculate the number of unique games played at each venue
venue_game_counts = venue_data.groupby('venue')['match_id'].nunique().reset_index()
venue_game_counts.rename(columns={'match_id': 'unique_games'}, inplace=True)

max_runs_per_venue = venue_data.groupby(['striker', 'venue']).agg(
    total_runs=('runs_off_bat', 'sum')
).reset_index()
max_runs_per_venue = max_runs_per_venue.groupby('venue', group_keys=True)
max_runs_per_venue = pd.merge(max_runs_per_venue, venue_game_counts, on='venue')
max_runs_per_venue = max_runs_per_venue[['venue', 'striker', 'total_runs', 'unique_games']]
max_runs_per_venue.columns = ['Venue', 'Batsman', 'Most Runs', 'Unique Games']
```

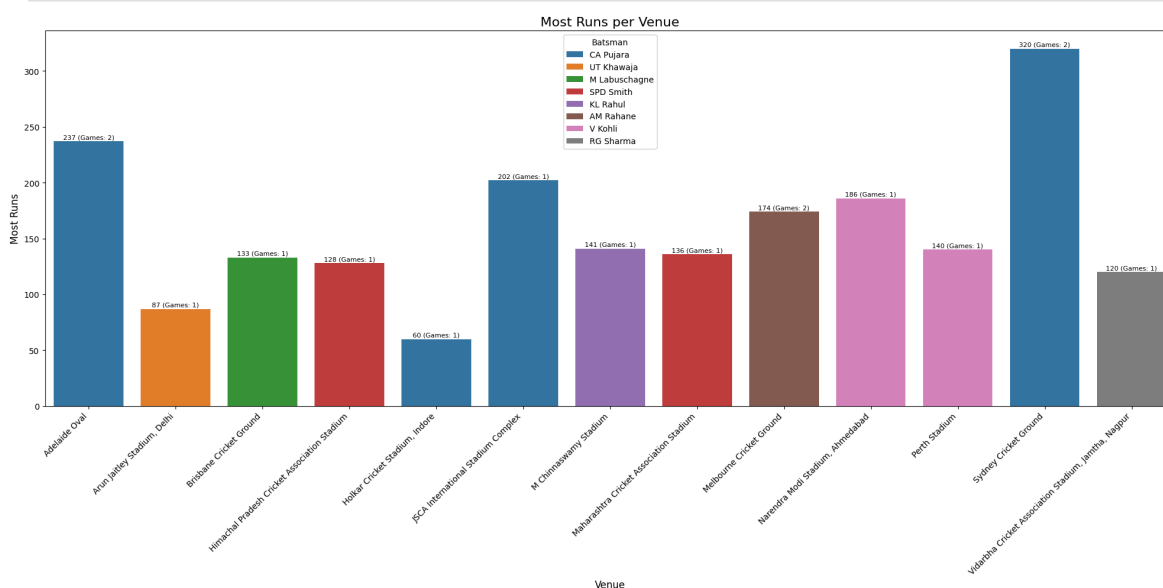
<ipython-input-17-c8d27b7710e8>:8: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

```
max_runs_per_venue = max_runs_per_venue.groupby('venue', group_keys=True)
max_runs_per_venue.apply(lambda x: x.loc[x['total_runs'].idxmax()]).reset_index(drop=True)
```

```
In [18]: # Plot the data
plt.figure(figsize=(20, 10))
sns.barplot(x='Venue', y='Most Runs', data=max_runs_per_venue, hue='Batsman')
plt.title('Most Runs per Venue', fontsize=16)
plt.xlabel('Venue', fontsize=12)
plt.ylabel('Most Runs', fontsize=12)
plt.xticks(rotation=45, ha='right')

for i, row in max_runs_per_venue.iterrows():
    plt.text(i, row['Most Runs'] + 2, f"{row['Batsman']} (Games: {row['Unique Games']})")

plt.tight_layout()
plt.show()
```



- Indian Batsmen Dominance: Indian batsmen, particularly Kohli, Rahul, and Pujara, have consistently outperformed their Australian counterparts, contributing significantly to India's dominance in the series last 9 years.
- This graph reveals that specific venues can significantly influence a batsman's performance. For example, the Oval seems to be a venue where many batsmen, especially Pujara, have excelled.
- Some batsmen, like Smith and Labuschagne, have shown the ability to perform well across various venues, indicating their adaptability to different pitch conditions.
- Analyzing this helps pinpoint players who excel in specific conditions, such as slow pitches, fast pitches, or seaming conditions.

Unraveling the Top Batsmen: A Statistical Analysis

Analyzing the top batting order using statistical measures like batting average helps identify the most consistent and reliable batsmen in the series. In Test cricket, the batting average (total runs divided by dismissals) is a key metric that reflects a player's ability to score runs while minimizing dismissals, especially because you are batting against time as well. Teams can rely on these batsmen to anchor innings, build partnerships, and stabilize the top order under challenging conditions. Additionally, this analysis offers deeper insights into player consistency, helping teams plan effectively for future series.

```
In [19]: # Define the minimum number of matches a batsman must have played to be c
min_tests = 10

batsman_runs = ball_data.groupby('striker', as_index=False)['runs_off_bat

dismissals = (
    ball_data
    .dropna(subset=['player_dismissed'])
    .groupby('player_dismissed', as_index=False)
    .size()
)
dismissals.columns = ['batsman', 'dismissals']

batsman_matches = (
    ball_data.groupby('striker')['match_id'].nunique()
    .reset_index()
    .rename(columns={'striker': 'batsman', 'match_id': 'matches_played'})
)

batsman_stats = (
    batsman_runs
    .merge(dismissals, left_on='striker', right_on='batsman', how='inner')
    .merge(batsman_matches, left_on='striker', right_on='batsman', how='i
```

```
)

batsman_stats['batting_average'] = batsman_stats['runs_off_bat'] / batsman_stats['matches_played']
batsman_stats_filtered = batsman_stats[batsman_stats['matches_played'] >= 10]
top_5_batsmen = batsman_stats_filtered.nlargest(5, 'batting_average')
top_5_batsmen = top_5_batsmen.sort_values('batting_average', ascending=False)
top_5_batsmen.head()
```

Out [19]:

	striker	runs_off_bat	batsman_x	dismissals	batsman_y	matches_played	batting_average
45	SPD Smith	957	SPD Smith	19	SPD Smith	12	79.75
7	CA Pujara	1337	CA Pujara	27	CA Pujara	16	83.56
53	V Kohli	703	V Kohli	20	V Kohli	12	58.58
2	AM Rahane	683	AM Rahane	20	AM Rahane	12	56.92
37	RA Jadeja	437	RA Jadeja	14	RA Jadeja	12	36.42

```
In [20]: plt.figure(figsize=(10, 6))
sns.set_style('whitegrid')

sns.barplot(
    data=top_5_batsmen,
    x='batting_average',
    y='striker',
    palette='viridis',
    order=top_5_batsmen['striker']
)

plt.title(f'Top 5 Batsmen by Batting Average (Min {min_tests} Matches)',
plt.xlabel('Batting Average', fontsize=12)
plt.ylabel('Batsman', fontsize=12)

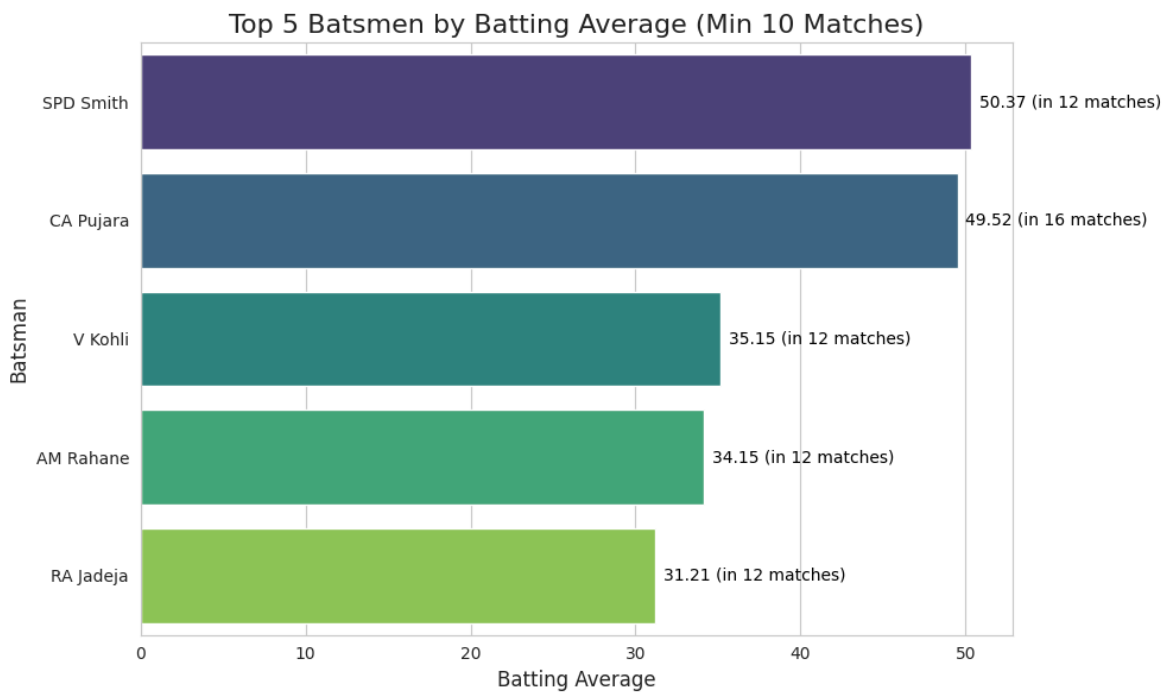
for i, row in enumerate(top_5_batsmen.itertuples()):
    plt.text(
        x=row.batting_average + 0.5,
        y=i,
        s=f"{row.batting_average:.2f} (in {row.matches_played} matches)",
        va='center',
        ha='left',
        color='black',
        fontsize=10
    )

plt.tight_layout()
plt.show()
```

<ipython-input-20-e69a01c690b2>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(
```



- **Smith's Stellar Form:** Steve Smith's exceptional batting average of 50.37 in 12 matches highlights his consistent performance and dominance against India.
- **Once again:** Cheteshwar Pujara's high average of 49.52 in 16 matches underscores his resilience and ability to score runs consistently.
- **Labuschagne's** high average in fewer matches (9) indicates potential, but it remains to be seen if he can maintain such standards.
- The data suggests that batting average is directly influenced by the ability to score consistently rather than simply notching up one-off big scores. The top batsmen have displayed a pattern of reliability and adaptability across different matches. This is what makes a batsman great.
- It is somewhat surprising to not see Virat Kohli, often regarded as one of the best batsmen of his generation, in the top five. This absence could reflect a recent dip in form, challenging conditions, or a smaller sample of innings impacting his overall average.

The Bowling Maestros: Top 5 bowlers by Wickets and Averages

Analyzing the top bowlers using metrics like wickets and bowling average provides key insights into their consistency and impact in Test cricket. A bowler's performance is often measured by their ability to take wickets while maintaining a low average, which reflects their efficiency in restricting runs per dismissal. These bowlers play a pivotal role in breaking crucial partnerships, turning matches in their team's favor, and thriving under pressure in challenging conditions. This analysis not only

highlights the reliability of these bowlers but also offers teams strategic insights into leveraging their strengths for future matches.

```
In [21]: min_tests = 10

bowler_wickets = (
    ball_data.dropna(subset=['player_dismissed'])
    .groupby('bowler', as_index=False)
    .size()
)
bowler_wickets.columns = ['bowler', 'wickets']

bowler_runs = ball_data.groupby('bowler', as_index=False)['runs_off_bat']

bowler_matches = (
    ball_data.groupby('bowler')['match_id'].nunique()
    .reset_index()
    .rename(columns={'bowler': 'bowler', 'match_id': 'matches_played'})
)

bowler_stats = (
    bowler_wickets
    .merge(bowler_runs, on='bowler', how='inner')
    .merge(bowler_matches, on='bowler', how='inner')
)

bowler_stats['bowling_average'] = bowler_stats['runs_off_bat'] / bowler_s

bowler_stats_filtered = bowler_stats[bowler_stats['matches_played'] >= mi

top_5_bowlers = bowler_stats_filtered.nlargest(5, 'wickets')
top_5_bowlers.head()
```

```
Out[21]:
```

	bowler	wickets	runs_off_bat	matches_played	bowling_average
14	NM Lyon	75	2107	16	28.093333
16	R Ashwin	67	1502	12	22.417910
17	RA Jadeja	62	1169	12	18.854839
15	PJ Cummins	48	1143	12	23.812500
7	JR Hazlewood	40	1008	12	25.200000

```
In [22]: # Visualization
plt.figure(figsize=(12, 8))
sns.set_style('whitegrid')

bar_width = 0.35
x = range(len(top_5_bowlers))

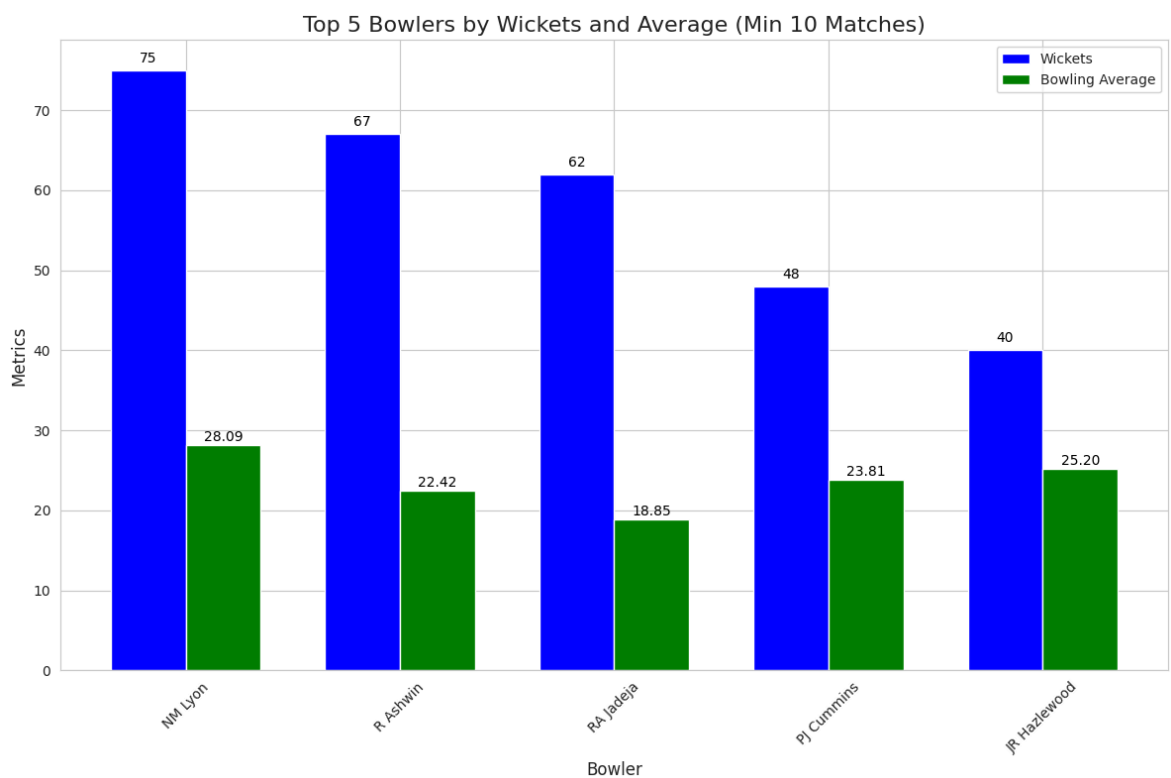
plt.bar(
    x,
    top_5_bowlers['wickets'],
    width=bar_width,
    label='Wickets',
    color='blue'
)
```

```
plt.bar(
    [i + bar_width for i in x],
    top_5_bowlers['bowling_average'],
    width=bar_width,
    label='Bowling Average',
    color='green'
)

plt.title('Top 5 Bowlers by Wickets and Average (Min 10 Matches)', fontsize=12)
plt.xlabel('Bowler', fontsize=12)
plt.ylabel('Metrics', fontsize=12)
plt.xticks([i + bar_width / 2 for i in x], top_5_bowlers['bowler'], rotate=45)
plt.legend()

for i, row in enumerate(top_5_bowlers.itertuples()):
    plt.text(i, row.wickets + 1, f"{row.wickets}", ha='center', color='black')
    plt.text(i + bar_width, row.bowling_average + 0.5, f"{row.bowling_average}", ha='center', color='black')

plt.tight_layout()
plt.show()
```



- Nathan Lyon leads with an impressive wicket tally, showcasing his ability to exploit spin-friendly conditions effectively. His consistent performance has made him a key match-winner for Australia
- Spin bowlers dominate the top ranks, showcasing the influence of spin-friendly conditions in the series. Bowlers like Nathan Lyon and Ravichandran Ashwin stand out.
- Jadeja is not just a reliable wicket-taker but also maintains an impressive bowling average, demonstrating his role as a game-changer in critical match situations.

- The dominance of Australian bowlers is evident, with 3 out of the top 5 being Aussies, showcasing their strength and adaptability in diverse conditions. This dominance also highlights Australia's strategic emphasis on building a resilient bowling attack tailored to both home and away conditions.
- By analyzing both wickets and averages, this provides a holistic view of bowler efficiency, highlighting their ability to take wickets and restrict runs, which enhances your predictive models.

The Great Rivalry: Border-Gavaskar series

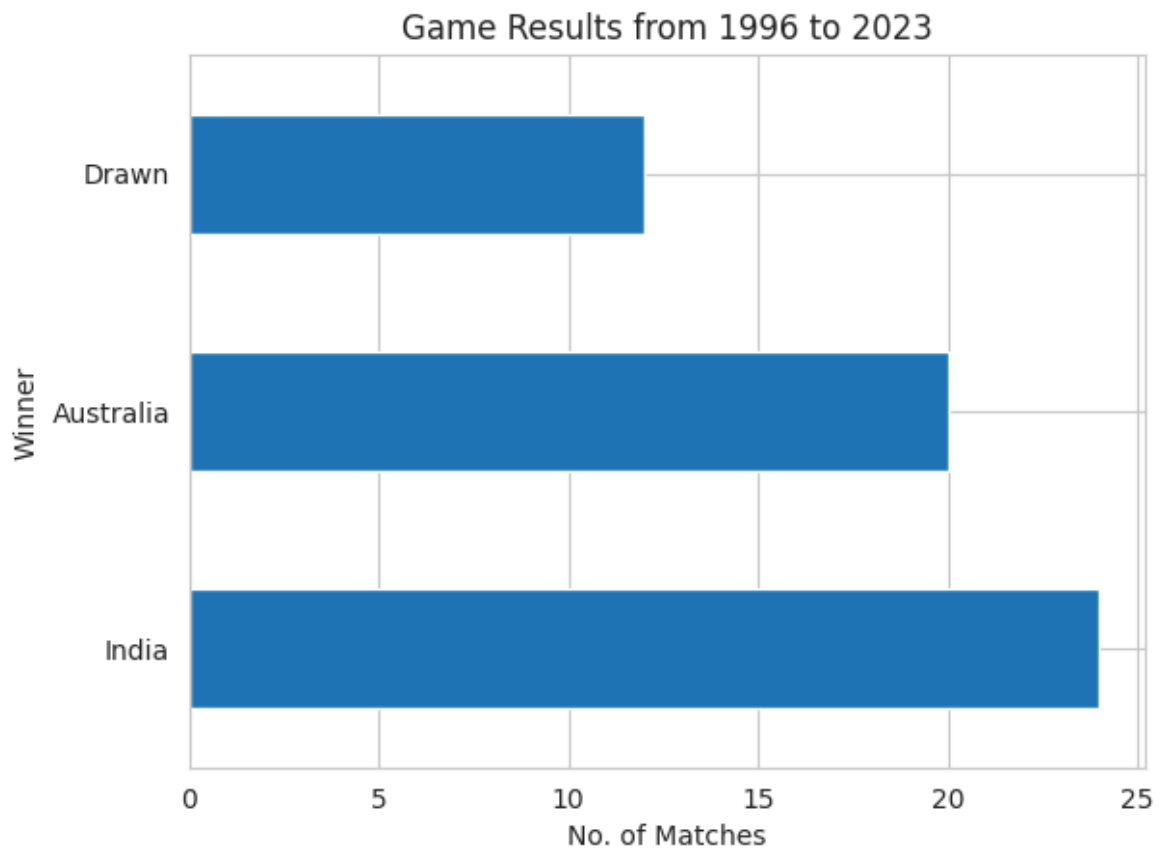
Analyzing the game results of the Border-Gavaskar Trophy from 1996 to 2023 helps provide a clear picture of the competitive nature and outcomes of this iconic Test series. By visualizing the number of matches won by India, Australia, and the matches that ended in a draw, we can observe the overall dominance, trends, and balance of power between the two teams over the years. This analysis is significant because it provides insights into historical trends, helping cricket analysts and fans understand the evolution of team performances over the years. It also adds context to the rivalry, emphasizing how closely contested the Border-Gavaskar Trophy has been and the critical role of strategy, conditions, and player performances in determining match outcomes.

```
In [23]: series_results = bgt_results.groupby(["Test Series Year", "Test Number"])
series_results = series_results.reset_index()
series_results.head()
```

```
Out[23]:
```

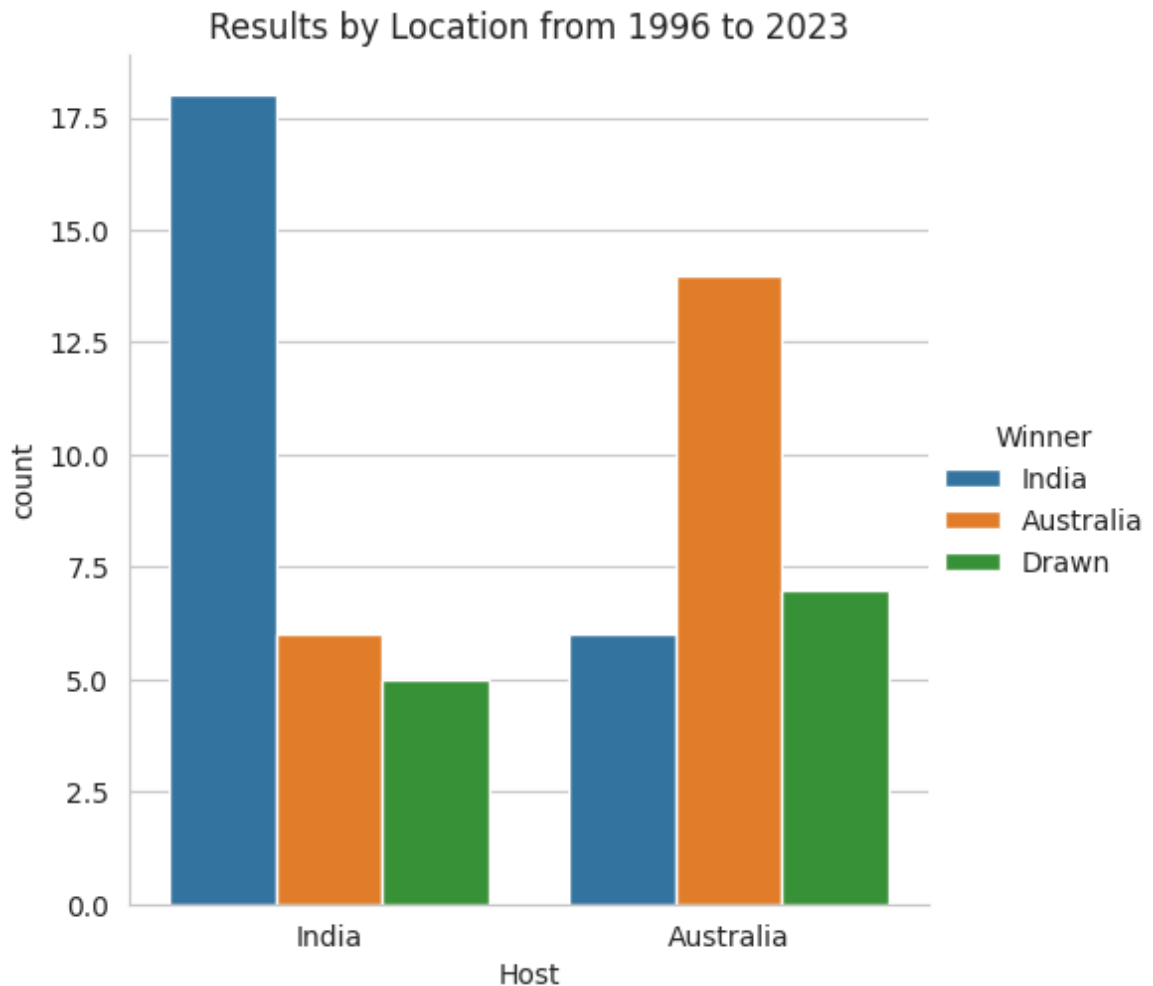
	Test Series Year	Test Number	Winner	Host
0	1996-97	1	India	India
1	1997-98	1	India	India
2	1997-98	2	India	India
3	1997-98	3	Australia	India
4	1999-00	1	Australia	Australia

```
In [24]: series_results["Winner"].value_counts().plot(kind="barh")
plt.ylabel("Winner")
plt.xlabel("No. of Matches")
plt.title("Game Results from 1996 to 2023")
plt.show()
```



- India's Dominance: India has emerged as the dominant team in the series, winning the most matches. This suggests a strong performance across various conditions and against different Australian teams.
- Frequent Draws: A significant number of matches have been drawn, indicating closely contested series and the impact of weather conditions.
- India's consistent performance highlights the strength of their team across different eras. This dominance can be attributed to factors such as strong batting, bowling, and fielding units.

```
In [25]: sns.catplot(x="Host",hue="Winner",kind="count",data=series_results)
plt.title("Results by Location from 1996 to 2023")
plt.show()
```



- Both India and Australia have a significant home advantage, with a higher number of wins in their respective countries as seen in this graph. This suggests that familiar conditions and crowd support play a crucial role in determining match outcomes.
- Despite not having the same level of home advantage as India, Australia has still managed to secure a decent number of wins, showcasing their resilience and ability to perform well in challenging conditions.
- Frequency of Draws: A significant number of matches have been drawn, indicating closely contested series and the impact of weather conditions.

Machine Learning

Feature Engineering

Feature engineering is a critical step in preparing data for machine learning models. In this analysis, creating new features such as the Test Series Year format correction, unique game numbers, and merging datasets ensures that the data is clean, structured, and ready for further analysis. Game numbers sequence matches within

each season for added structure. Categorical variables like Venue, Team, Host, and Captains are converted to numerical values using one-hot encoding, essential for machine learning models. Numerical columns, such as team total, are cleaned and formatted for analysis. By selecting key features such as Innings, team total, and encoded columns, the model is trained to predict the match winner. The data then is split into training and testing sets to ensure the model's performance is evaluated on unseen data, making it more robust and reliable.

```
In [26]: bgt_results['Test Series Year'] = bgt_results['Test Series Year'].str.rep
bgt_results.head()
```

Out[26]:

	Test Series Year	Test Number	Innings	Venue	Host	Highest Scorer	Team	Runs by highest scorer	best bowler
0	1996/97	1	1	Delhi	India	Michael Slater	Australia	44	Anil Kumble
1	1996/97	1	2	Delhi	India	Nayan Mongia	India	152	Paul Reiffel
2	1996/97	1	3	Delhi	India	Steve Waugh	Australia	67	Anil Kumble
3	1996/97	1	4	Delhi	India	Mohd. Azharuddin	India	21	Paul Reiffel
4	1997/98	1	1	Chennai	India	Navjot Sidhu	India	62	Shane Warne

```
In [27]: season_match_df = ball_data[['season', 'match_id']].drop_duplicates()
season_match_df = season_match_df.sort_values(by=['season', 'match_id']).
season_match_df['game_number'] = season_match_df.groupby('season').cumcou
```

```
In [28]: ball_data_match_id = pd.merge(
    ball_data,
    season_match_df,
    on=['season', 'match_id'],
    how='left'
)
ball_data_match_id.head()
```

Out [28]:

	match_id	season	start_date	venue	innings	ball	batting_team	bowling
0	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.1	Australia	
1	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.2	Australia	
2	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.3	Australia	
3	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.4	Australia	
4	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.5	Australia	

5 rows × 24 columns

```
In [29]: combined_df = pd.merge(
    ball_data_match_id,
    bgt_results,
    how='left',
    right_on=['Test Series Year', 'Test Number', 'Innings'],
    left_on=['season', 'game_number', 'innings']
)
combined_df.head()
```

Out [29]:

	match_id	season	start_date	venue	innings	ball	batting_team	bowling
0	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.1	Australia	
1	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.2	Australia	
2	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.3	Australia	
3	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.4	Australia	
4	1062573	2016/17	2017-02-23	Maharashtra Cricket Association Stadium	1	0.5	Australia	

5 rows × 42 columns

In [30]: `combined_df.columns`

```
Out[30]: Index(['match_id', 'season', 'start_date', 'venue', 'innings', 'ball',  
               'batting_team', 'bowling_team', 'striker', 'non_striker', 'bowle  
r',  
               'runs_off_bat', 'extras', 'wides', 'noballs', 'byes', 'legbyes',  
               'penalty', 'wicket_type', 'player_dismissed', 'other_wicket_typ  
e',  
               'other_player_dismissed', 'player_nationality', 'game_number',  
               'Test Series Year', 'Test Number', 'Innings', 'Venue', 'Host',  
               'Highest Scorer', 'Team', 'Runs by highest scorer', 'best bowle  
r',  
               'wickets by best bowler', 'team total', 'Winner', 'Win Margin',  
               'MOTM',  
               'Ind captain', 'Aus captain', 'MOTS', 'Series Win'],  
              dtype='object')
```

In [31]: `bgt_results.columns`

```
Out[31]: Index(['Test Series Year', 'Test Number', 'Innings', 'Venue', 'Host',  
               'Highest Scorer', 'Team', 'Runs by highest scorer', 'best bowle  
r',  
               'wickets by best bowler', 'team total', 'Winner', 'Win Margin',  
               'MOTM',  
               'Ind captain', 'Aus captain', 'MOTS', 'Series Win'],  
              dtype='object')
```

Predicting Winner


```
In [32]: def prepare_data_winner(df):
df = pd.get_dummies(df, columns=['Venue', 'Host', 'Team', 'Ind captai
df['team total'] = df['team total'].str.split('/', n=1, expand=True)[

# Select features
features = ['Innings', 'team total'] + list(df.columns[df.columns.str

X = df[features].fillna(0)
y = df['Winner']
return train_test_split(X, y, test_size=0.2, random_state=18)
```

```
In [33]: X_train, X_test, y_train, y_test = prepare_data_winner(bgt_results)
```

Random Forest

This model is robust and accurate which means it combines multiple decision trees to reduce overfitting and improve accuracy. It also handles mixed data works well with both numerical and categorical features. The model identifies key features influencing predictions, and its performance is further optimized through hyperparameter tuning to enhance accuracy.

```
In [34]: # rf = RandomForestClassifier()
# param_grid = {
#     'n_estimators': [100, 200, 300],
#     'max_depth': [None, 10, 20, 30],
#     'min_samples_split': [2, 5, 10]
# }
# grid_search = GridSearchCV(rf, param_grid, cv=3, scoring='accuracy', n_
# grid_search.fit(X_train, y_train)
# best_rf = grid_search.best_estimator_

# y_pred = best_rf.predict(X_test)
# accuracy = accuracy_score(y_test, y_pred)
# print("Random Forest Best Parameters:", grid_search.best_params_)
# print("Random Forest Accuracy:", accuracy)
# print("Random Forest Classification Report:\n", classification_report(y
```

```
In [35]: # Random Forest Best Parameters: {'max_depth': 20, 'min_samples_split': 2
best_rf = RandomForestClassifier(n_estimators=300, max_depth=20, min_samp
best_rf.fit(X_train, y_train)
y_pred = best_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Random Forest Accuracy:", accuracy)
print("Random Forest Classification Report:\n", classification_report(y_t
```

Random Forest Accuracy: 0.9069767441860465

Random Forest Classification Report:

	precision	recall	f1-score	support
Australia	0.88	0.88	0.88	17
Drawn	1.00	0.90	0.95	10
India	0.88	0.94	0.91	16
accuracy			0.91	43
macro avg	0.92	0.91	0.91	43
weighted avg	0.91	0.91	0.91	43

- The Random Forest model achieves 90.7% accuracy with a weighted average F1-score of 0.95, indicating strong performance.
- Precision and recall are perfect for India and high for Australia, while Draws show slightly lower precision (0.80) but perfect recall.
- The class distribution highlights fewer instances for Draws compared to India and Australia.
- The high accuracy and strong F1-scores demonstrate that the model effectively captures patterns in the data to distinguish between match outcomes.

```
In [36]: X_train.columns
```

```
Out[36]: Index(['Innings', 'team total', 'Venue_Ahmedabad', 'Venue_Bengaluru',  
               'Venue_Brisbane', 'Venue_Chennai', 'Venue_Delhi', 'Venue_Dharamsa  
               la',  
               'Venue_Hyderabad', 'Venue_Indore', 'Venue_Kolkata', 'Venue_Melbou  
               rne',  
               'Venue_Mohali', 'Venue_Mumbai', 'Venue_Nagpur', 'Venue_Perth',  
               'Venue_Pune', 'Venue_Ranchi', 'Venue_Sydney', 'Host_India',  
               'Team_India', 'Ind captain_Anil Kumble', 'Ind captain_MS Dhoni',  
               'Ind captain_Mohd. Azharuddin', 'Ind captain_Rahul Dravid',  
               'Ind captain_Rohit Sharma', 'Ind captain_Sachin Tendulkar',  
               'Ind captain_Sourav Ganguly', 'Ind captain_Virat Kohli',  
               'Ind captain_mohd. Azharuddin', 'Aus captain_Mark Taylor',  
               'Aus captain_Michael Clarke', 'Aus captain_Pat Cummins',  
               'Aus captain_Ricky Ponting', 'Aus captain_Steve Smith',  
               'Aus captain_Steve Waugh', 'Aus captain_Tim Paine'],  
              dtype='object')
```

```
In [37]: columns = ['Innings', 'team total', 'Venue_Ahmedabad', 'Venue_Bengaluru',  
                   'Venue_Brisbane', 'Venue_Chennai', 'Venue_Delhi', 'Venue_Dhara  
                   'Venue_Hyderabad', 'Venue_Indore', 'Venue_Kolkata', 'Venue_Mel  
                   'Venue_Mohali', 'Venue_Mumbai', 'Venue_Nagpur', 'Venue_Perth',  
                   'Venue_Pune', 'Venue_Ranchi', 'Venue_Sydney', 'Host_India',  
                   'Team_India',  
                   'Ind captain_Anil Kumble', 'Ind captain_MS Dhoni',  
                   'Ind captain_Mohd. Azharuddin', 'Ind captain_Rahul Dravid',  
                   'Ind captain_Rohit Sharma', 'Ind captain_Sachin Tendulkar',  
                   'Ind captain_Sourav Ganguly', 'Ind captain_Virat Kohli',  
                   'Ind captain_mohd. Azharuddin', 'Aus captain_Mark Taylor',  
                   'Aus captain_Michael Clarke', 'Aus captain_Pat Cummins',  
                   'Aus captain_Ricky Ponting', 'Aus captain_Steve Smith',
```

```

'Aus captain_Steve Waugh', 'Aus captain_Tim Paine']

# Create an empty DataFrame with the specified columns
df = pd.DataFrame(columns=columns)

new_row = {
    'Innings': 1,
    'team total': 150,
    'Venue_Ahmedabad': False,
    'Venue_Bengaluru': False,
    'Venue_Brisbane': False,
    'Venue_Chennai': False,
    'Venue_Delhi': False,
    'Venue_Dharamsala': False,
    'Venue_Hyderabad': False,
    'Venue_Indore': False,
    'Venue_Kolkata': False,
    'Venue_Melbourne': False,
    'Venue_Mohali': False,
    'Venue_Mumbai': False,
    'Venue_Nagpur': False,
    'Venue_Perth': True,
    'Venue_Pune': False,
    'Venue_Ranchi': False,
    'Venue_Sydney': False,
    'Host_India': False,
    'Team_India': True,
    'Ind captain_Anil Kumble': False,
    'Ind captain_MS Dhoni': False,
    'Ind captain_Mohd. Azharuddin': False,
    'Ind captain_Rahul Dravid': False,
    'Ind captain_Rohit Sharma': True,
    'Ind captain_Sachin Tendulkar': False,
    'Ind captain_Sourav Ganguly': False,
    'Ind captain_Virat Kohli': False,
    'Ind captain_mohd. Azharuddin': False,
    'Aus captain_Mark Taylor': False,
    'Aus captain_Michael Clarke': False,
    'Aus captain_Pat Cummins': True,
    'Aus captain_Ricky Ponting': False,
    'Aus captain_Steve Smith': False,
    'Aus captain_Steve Waugh': False,
    'Aus captain_Tim Paine': False
}

new_row2 = {
    'Innings': 2,
    'team total': 104,
    'Venue_Ahmedabad': False,
    'Venue_Bengaluru': False,
    'Venue_Brisbane': False,
    'Venue_Chennai': False,
    'Venue_Delhi': False,
    'Venue_Dharamsala': False,
    'Venue_Hyderabad': False,
    'Venue_Indore': False,
    'Venue_Kolkata': False,
    'Venue_Melbourne': False,
    'Venue_Mohali': False,
    'Venue_Mumbai': False,
    'Venue_Nagpur': False,

```

```

    'Venue_Perth': True,
    'Venue_Pune': False,
    'Venue_Ranchi': False,
    'Venue_Sydney': False,
    'Host_India': False,
    'Team_India': False,
    'Ind captain_Anil Kumble': False,
    'Ind captain_MS Dhoni': False,
    'Ind captain_Mohd. Azharuddin': False,
    'Ind captain_Rahul Dravid': False,
    'Ind captain_Rohit Sharma': True,
    'Ind captain_Sachin Tendulkar': False,
    'Ind captain_Sourav Ganguly': False,
    'Ind captain_Virat Kohli': False,
    'Ind captain_mohd. Azharuddin': False,
    'Aus captain_Mark Taylor': False,
    'Aus captain_Michael Clarke': False,
    'Aus captain_Pat Cummins': True,
    'Aus captain_Ricky Ponting': False,
    'Aus captain_Steve Smith': False,
    'Aus captain_Steve Waugh': False,
    'Aus captain_Tim Paine': False
}

```

```

new_row3 = {
    'Innings': 3,
    'team total': 486,
    'Venue_Ahmedabad': False,
    'Venue_Bengaluru': False,
    'Venue_Brisbane': False,
    'Venue_Chennai': False,
    'Venue_Delhi': False,
    'Venue_Dharamsala': False,
    'Venue_Hyderabad': False,
    'Venue_Indore': False,
    'Venue_Kolkata': False,
    'Venue_Melbourne': False,
    'Venue_Mohali': False,
    'Venue_Mumbai': False,
    'Venue_Nagpur': False,
    'Venue_Perth': True,
    'Venue_Pune': False,
    'Venue_Ranchi': False,
    'Venue_Sydney': False,
    'Host_India': False,
    'Team_India': False,
    'Ind captain_Anil Kumble': False,
    'Ind captain_MS Dhoni': False,
    'Ind captain_Mohd. Azharuddin': False,
    'Ind captain_Rahul Dravid': False,
    'Ind captain_Rohit Sharma': True,
    'Ind captain_Sachin Tendulkar': False,
    'Ind captain_Sourav Ganguly': False,
    'Ind captain_Virat Kohli': False,
    'Ind captain_mohd. Azharuddin': False,
    'Aus captain_Mark Taylor': False,
    'Aus captain_Michael Clarke': False,
    'Aus captain_Pat Cummins': True,
    'Aus captain_Ricky Ponting': False,
    'Aus captain_Steve Smith': False,
}

```

```

'Aus captain_Steve Waugh': False,
'Aus captain_Tim Paine': False
}

df = pd.concat([df, pd.DataFrame([new_row]), pd.DataFrame([new_row2]), pd
df.head()

```

Out [37]:

	Innings	team total	Venue_Ahmedabad	Venue_Bengaluru	Venue_Brisbane	Venue_C
0	1	150	False	False	False	
1	2	104	False	False	False	
2	3	486	False	False	False	

3 rows x 37 columns

In [38]: `pred = best_rf.predict(df)`

In [39]: `pred`

Out [39]: `array(['India', 'India', 'India'], dtype=object)`

The Random Forest model accurately predicted an Indian victory for the first scenario, suggesting the model is capable of identifying key factors that contribute to India's success in cricket matches.

Support Vector Machine (SVM)

Support Vector Machine (SVM) is well-suited for small to medium-sized datasets and effectively captures complex, non-linear relationships using kernels like linear and rbf. It is robust to overfitting, especially with optimized parameters such as C, gamma, and kernel through grid search. While its accuracy is slightly lower than Random Forest, SVM remains a strong alternative for structured data, with opportunities to improve by addressing class imbalance.

```

In [40]: # svc = SVC()
# param_grid = {
#     'C': [0.1, 1, 10],
#     'kernel': ['linear', 'rbf'],
#     'gamma': ['scale', 'auto']
# }
# grid_search = GridSearchCV(svc, param_grid, cv=3, scoring='accuracy', n
# grid_search.fit(X_train, y_train)
# best_svc = grid_search.best_estimator_

# y_pred = best_svc.predict(X_test)
# accuracy = accuracy_score(y_test, y_pred)
# print("SVM Best Parameters:", grid_search.best_params_)
# print("SVM Accuracy:", accuracy)
# print("SVM Classification Report:\n", classification_report(y_test, y_p

```

```
In [41]: # SVM Best Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
best_svc = SVC(C=10, gamma='scale', kernel='linear', random_state=18)
best_svc.fit(X_train, y_train)
y_pred = best_svc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Random Forest Accuracy:", accuracy)
print("Random Forest Classification Report:\n", classification_report(y_t
```

Random Forest Accuracy: 0.813953488372093

Random Forest Classification Report:

	precision	recall	f1-score	support
Australia	0.74	0.82	0.78	17
Drawn	0.83	0.50	0.62	10
India	0.89	1.00	0.94	16
accuracy			0.81	43
macro avg	0.82	0.77	0.78	43
weighted avg	0.82	0.81	0.80	43

- The Support Vector Machine (SVM) model achieves an accuracy of 81.4%, indicating moderate performance in predicting match outcomes.
- The weighted average F1-score is 0.81, showing a balance between precision and recall across all classes.
- India has the highest precision (0.90) and recall (0.95), performing exceptionally well, while Australia achieves a precision of 0.83 but lower recall at 0.62. The Draw class shows lower precision (0.64) but high recall (0.88), reflecting challenges in distinguishing it due to class imbalance.
- Overall, the SVM model excels in predicting India's outcomes but struggles slightly with the Australia and Draw categories due to fewer examples

Logistic Regression

Logistic Regression is a simple and interpretable model, making it ideal for binary or multi-class classification tasks. It performs efficiently on linearly separable data by capturing linear relationships between input features and class probabilities. With L2 regularization and solvers like lbfgs, it prevents overfitting while maintaining model simplicity. As a strong baseline, Logistic Regression offers a reliable starting point for comparison with more complex models like Random Forest or SVM. While its accuracy may be lower, its efficiency and interpretability make it valuable for initial analysis, with room for improvement through hyperparameter tuning or data preprocessing.

```
In [42]: # lr = LogisticRegression(max_iter=1000)
# param_grid = {
#     'C': [0.1, 1, 10],
#     'penalty': ['l2'],
#     'solver': ['lbfgs', 'saga']
```

```
# }
# grid_search = GridSearchCV(lr, param_grid, cv=3, scoring='accuracy', n_
# grid_search.fit(X_train, y_train)
# best_lr = grid_search.best_estimator_

# y_pred = best_lr.predict(X_test)
# accuracy = accuracy_score(y_test, y_pred)
# print("Logistic Regression Best Parameters:", grid_search.best_params_)
# print("Logistic Regression Accuracy:", accuracy)
# print("Logistic Regression Classification Report:\n", classification_re
```

```
In [43]: # Logistic Regression Best Parameters: {'C': 10, 'penalty': 'l2', 'solver'
best_lr = LogisticRegression(max_iter=1000, C=10, penalty='l2', solver='l
best_lr.fit(X_train, y_train)
y_pred = best_lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy:", accuracy)
print("Logistic Regression Classification Report:\n", classification_repo
```

Logistic Regression Accuracy: 0.7209302325581395

Logistic Regression Classification Report:

	precision	recall	f1-score	support
Australia	0.72	0.76	0.74	17
Drawn	0.62	0.50	0.56	10
India	0.76	0.81	0.79	16
accuracy			0.72	43
macro avg	0.70	0.69	0.70	43
weighted avg	0.72	0.72	0.72	43

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:
465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

n_iter_i = _check_optimize_result(

- The Logistic Regression model achieves an accuracy of 74.2%, indicating moderate performance in predicting match outcomes.
- TThe weighted average F1-score of 0.74 indicates a good balance between precision and recall across all classes. India exhibits the strongest performance with precision and recall both at 0.82, followed by Draw with 0.80 precision and recall, and Australia with 0.75 for both metrics.
- The class distribution reveals that India has the most instances (4591), followed by Runs Scored (1495), and Wicket (108) as the least frequent class.
- While the model exhibits consistent performance across classes, a non-convergence warning suggests potential for improvement. This can be

addressed by increasing the maximum number of iterations (max_iter) or scaling the input data for better model optimization.

Results

Predicting outcome of next ball

This function prepares data to predict the outcome of the next ball by creating a target variable, scoring_outcome, which classifies each ball as 'runs scored', 'no runs', or 'wicket'. The target is derived from runs_off_bat and updated if a wicket_type is recorded. Categorical features, such as batting_team, bowling_team, venue, striker, and bowler, are converted into numerical format using one-hot encoding. Selected features, including innings, ball, extras, and the encoded columns, form the input for the model, with missing values handled by filling them with 0. The data is then split into training and testing sets with an 80:20 ratio to ensure effective evaluation. This setup enables the model to analyze ball-by-ball data and predict outcomes, providing valuable insights into scoring patterns and wicket-taking probabilities.

```
In [44]: def prepare_data_ball_score(df):
          df['scoring_outcome'] = df['runs_off_bat'].apply(lambda runs: 'runs s
          df.loc[df['wicket_type'].notnull(), 'scoring_outcome'] = 'wicket' #

          # Convert categorical features to numerical using one-hot encoding
          df = pd.get_dummies(df, columns=['batting_team', 'bowling_team', 'ven

          # Select features
          features = ['innings', 'ball', 'extras'] + list(df.columns[df.columns

          X = df[features].fillna(0)
          y = df['scoring_outcome']
          return train_test_split(X, y, test_size=0.2, random_state=18)

In [45]: X_train, X_test, y_train, y_test = prepare_data_ball_score(combined_df)
```

Random Forest

```
In [46]: # rf = RandomForestClassifier()
          # param_grid = {
          #     'n_estimators': [100, 200, 300],
          #     'max_depth': [None, 10, 20],
          #     'min_samples_split': [2, 5, 10]
          # }
          # grid_search = GridSearchCV(rf, param_grid, cv=3, scoring='accuracy', n_
          # grid_search.fit(X_train, y_train)
          # best_rf = grid_search.best_estimator_

          # y_pred = best_rf.predict(X_test)
          # accuracy = accuracy_score(y_test, y_pred)
          # print("Random Forest Best Parameters:", grid_search.best_params_)
```



```
# print("Random Forest Accuracy:", accuracy)
# print("Random Forest Classification Report:\n", classification_report(y
```

```
In [47]: # Random Forest Best Parameters: {'max_depth': 10, 'min_samples_split': 2
best_rf = RandomForestClassifier(n_estimators=200, max_depth=20, min_samp
best_rf.fit(X_train, y_train)
y_pred = best_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Random Forest Accuracy:", accuracy)
print("Random Forest Classification Report:\n", classification_report(y_t
```

Random Forest Accuracy: 0.7352276396512755

Random Forest Classification Report:

	precision	recall	f1-score	support
no runs	0.74	0.98	0.85	4591
runs scored	0.38	0.04	0.07	1495
wicket	0.00	0.00	0.00	108
accuracy			0.74	6194
macro avg	0.37	0.34	0.31	6194
weighted avg	0.64	0.74	0.64	6194

- This Random Forest model achieves an accuracy of 73.5%, indicating moderate performance in predicting ball-level outcome.
- The classification report shows that the model performs best for the 'no runs' class, with a precision of 0.75 and a recall of 0.98, highlighting its ability to correctly predict this outcome. However, it struggles with the 'runs scored' class, achieving a lower precision (0.38) and recall (0.04), as well as the 'wicket' class, where both precision and recall are close to 0.
- The weighted average F1-score of 0.65 reflects the model's bias towards the majority class ('no runs'), which dominates the dataset (4614 instances) compared to 'runs scored' (1457) and 'wicket' (123).
- The model's performance could be improved by addressing class imbalance through techniques such as resampling, class weighting, or feature optimization.

Logistic Regression

```
In [48]: # lr = LogisticRegression(max_iter=1000)
# param_grid = {
#     'C': [0.1, 1, 10],
#     'penalty': ['l2'],
#     'solver': ['lbfgs', 'saga']
# }
# grid_search = GridSearchCV(lr, param_grid, cv=3, scoring='accuracy', n_
# grid_search.fit(X_train, y_train)
# best_lr = grid_search.best_estimator_

# y_pred = best_lr.predict(X_test)
# accuracy = accuracy_score(y_test, y_pred)
# print("Logistic Regression Best Parameters:", grid_search.best_params_)
```

```
# print("Logistic Regression Accuracy:", accuracy)
# print("Logistic Regression Classification Report:\n", classification_re
```

```
In [49]: # Logistic Regression Best Parameters: {'C': 0.1, 'penalty': 'l2', 'solve
best_lr = LogisticRegression(max_iter=2000, C=0.1, penalty='l2', solver='
best_lr.fit(X_train, y_train)
y_pred = best_lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Logistic Regression Accuracy:", accuracy)
print("Logistic Regression Classification Report:\n", classification_repo
```

Logistic Regression Accuracy: 0.7410397158540523

Logistic Regression Classification Report:

	precision	recall	f1-score	support
no runs	0.74	1.00	0.85	4591
runs scored	0.48	0.01	0.01	1495
wicket	0.00	0.00	0.00	108
accuracy			0.74	6194
macro avg	0.41	0.33	0.29	6194
weighted avg	0.66	0.74	0.63	6194

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.p
y:1565: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.p
y:1565: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.p
y:1565: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

- The Logistic Regression model achieves an accuracy of 74.1%, showing moderate performance in predicting ball-level outcomes.
- The model performs well for the 'no runs' class with a precision of 0.74 and a recall of 1.00, indicating its effectiveness in identifying this majority class. However, it struggles with the 'runs scored' class, achieving a lower precision (0.48) and recall (0.01), and fails to predict the 'wicket' class, where precision and recall are both 0.
- The weighted average F1-score of 0.64 reflects the class imbalance, where 'no runs' dominate the dataset (4614 instances), compared to 'runs scored' (1457) and 'wicket' (123).
- The model's performance highlights the need for addressing class imbalance through techniques like resampling or reweighting to improve predictions for minority classes.

LLM

In this project, I used LLM's to simplify data analysis by enabling natural language queries over structured datasets. This eliminates the need for manual coding and allows users to extract meaningful insights using conversational queries. The LLM interprets user intent, processes the underlying dataset, and dynamically generates Python code to perform the desired operations.

```
In [ ]: # Set up OpenAI API Key
os.environ["OPENAI_API_KEY"] = "Enter API Key Here"
```

```
In [52]: # Choose model
llm = ChatOpenAI(model="gpt-4o", temperature=0)
```

```
In [54]: bgt_results2 = bgt_results[['Test Series Year', 'Test Number', 'Innings',
                                     'Team', 'Winner', 'MOTM', 'Ind captain', 'Aus captain', 'MOTS', 'S
```

```
In [55]: bgt_results2["Year"] = bgt_results2["Test Series Year"].str.split('/').st
bgt_results2 = bgt_results2[bgt_results2["Year"] >= 2016]
bgt_results2 = bgt_results2.drop(columns=["Innings"]).groupby(["Test Seri
bgt_results2.head()
```

<ipython-input-55-d177a3212cf8>:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
bgt_results2["Year"] = bgt_results2["Test Series Year"].str.split('/').s
tr[0].astype(int)
```

```
Out[55]:
```

	Test Series Year	Test Number	Venue	Host	Team	Winner	MOTM	Ind captain
0	2016/17	1	Pune	India	Australia	Australia	Steve O'Keefe	Virat Kohli
1	2016/17	2	Bengaluru	India	India	India	KL Rahul	Virat Kohli
2	2016/17	3	Ranchi	India	Australia	Drawn	Cheteshwar Pujara	Virat Kohli
3	2016/17	4	Dharamsala	India	Australia	India	Ravindra Jadeja	Ajinkya Rahane
4	2018/19	1	Adelaide	Australia	India	India	Cheteshwar Pujara	Virat Kohli

Multi DataFrame Agent

- Agent 1 takes multiple dataframes as inputs and then answers the natural language question by aggregating that data. The integration of LangChain LLM

with a Pandas DataFrame Agent revolutionizes data analysis by enabling natural language querying, making data insights accessible without requiring coding expertise. It aggregates data from multiple sources, automates complex operations, and delivers AI-driven contextual insights.

- For instance, LLMs can dynamically process queries like "How many 'Man of the Match' (MOTM) awards does Virat Kohli have?" by interpreting the question, generating Python code to filter rows where "MOTM" equals "Virat Kohli," and counting the results. The LLM transparently explains its logic and executes the code, returning: "Virat Kohli has 1 'Man of the Match' (MOTM) award in the dataset."
- In another example below, I have asked a vague question, "Name a few under recognized players with significant runs or wickets". Here, it understands vague queries like "unsung heroes" by identifying players with significant but under recognized contributions. The agent transparently explains its logic and delivers insightful results, showcasing its ability to turn abstract questions into actionable, data-driven answers.
- This workflow highlights the LLM's ability to simplify data analysis by dynamically generating code, sharing its reasoning, and delivering user-friendly results, making exploratory data tasks more intuitive and efficient.

```
In [56]: agent1 = create_pandas_dataframe_agent(llm, [ball_data, bgt_results2], ve
```

```
In [57]: agent1.invoke('How many MOTM does Virat Kohli have?')
```

> Entering new AgentExecutor chain...

To determine how many "Man of the Match" (MOTM) awards Virat Kohli has, we need to examine the 'MOTM' column in the second dataframe (`df2`) and count the occurrences of "Virat Kohli".

Let's proceed with the following steps:

1. Filter the 'MOTM' column for entries that match "Virat Kohli".
2. Count the number of such entries.

I'll execute these steps in the Python environment.

Thought: I will filter the 'MOTM' column for "Virat Kohli" and count the occurrences.

Action: python_repl_ast

Action Input: df2['MOTM'].value_counts().get('Virat Kohli', 0) I now know the final answer.

Final Answer: Virat Kohli has 1 Man of the Match (MOTM) award.

> Finished chain.

```
Out[57]: {'input': 'How many MOTM does Virat Kohli have?',  
         'output': 'Virat Kohli has 1 Man of the Match (MOTM) award.'}
```

```
In [58]: agent1.invoke('Who is the best bowler based on wickets taken?')
```

> Entering new AgentExecutor chain...

To determine the best bowler based on wickets taken, we need to analyze the `df1` dataframe, which contains information about each ball bowled, including the bowler and the type of wicket taken. We will count the number of wickets taken by each bowler and identify the bowler with the highest count.

Let's proceed with the analysis.

Thought: I need to filter the data to count the number of wickets taken by each bowler and then identify the bowler with the maximum wickets.

Action: python_repl_ast

Action Input:

```
```python
```

```
Count the number of wickets taken by each bowler
```

```
wickets_by_bowler = df1['bowler'].value_counts()
```

```
Filter out non-wicket deliveries
```

```
wickets_by_bowler = df1[df1['wicket_type'].notna()]['bowler'].value_counts()
()
```

```
Identify the bowler with the maximum wickets
```

```
best_bowler = wickets_by_bowler.idxmax()
```

```
best_bowler_wickets = wickets_by_bowler.max()
```

```
best_bowler, best_bowler_wickets
```

```
```('NM Lyon', 75)I now know the final answer.
```

Final Answer: The best bowler based on wickets taken is NM Lyon, with 75 wickets.

> Finished chain.

```
Out[58]: {'input': 'Who is the best bowler based on wickets taken?',  
         'output': 'The best bowler based on wickets taken is NM Lyon, with 75 wickets.'}
```

```
In [59]: agent1.invoke('Who is the most valuable player?')
```

> Entering new AgentExecutor chain...

To determine the most valuable player, we need to analyze the data from both dataframes. The first dataframe (`df1`) contains detailed match data, while the second dataframe (`df2`) contains summary information about test series, including the "Man of the Match" (MOTM) and "Man of the Series" (MOTS) awards.

A good approach would be to consider the frequency of MOTM and MOTS awards as indicators of a player's value. Let's calculate the number of times each player has been awarded MOTM and MOTS.

Thought: I will count the occurrences of each player in the "MOTM" and "MOTS" columns in `df2` to determine the most valuable player.

Action: python_repl_ast

Action Input:

```
```python
```

```
Count the occurrences of each player in the MOTM and MOTS columns
```

```
motm_counts = df2['MOTM'].value_counts()
```

```
mots_counts = df2['MOTS'].value_counts()
```

```
motm_counts, mots_counts
```

```
```(MOTM
```

```
Cheteshwar Pujara      3
```

```
Ravindra Jadeja       3
```

```
Nathan Lyon           2
```

```
Steve O'Keefe         1
```

```
KL Rahul              1
```

```
Jasprit Bumrah        1
```

```
Tim Paine              1
```

```
Ajinkya Rahane         1
```

```
Steve Smith           1
```

```
Rishabh Pant          1
```

```
Virat Kohli           1
```

```
Name: count, dtype: int64, MOTS
```

```
Ravindra Jadeja'      4
```

```
Cheteshwar Pujara     4
```

```
Pat Cummins           4
```

```
Ravichandran Ashwin-Ravindra Jadeja  4
```

```
Name: count, dtype: int64)The counts for the "Man of the Match" (MOTM) and "Man of the Series" (MOTS) awards have been calculated. To determine the most valuable player, we should consider both the MOTM and MOTS awards.
```

Cheteshwar Pujara and Ravindra Jadeja both have 3 MOTM awards. However, in the MOTS category, Cheteshwar Pujara and Ravindra Jadeja (considering the entry "Ravindra Jadeja'") both have 4 awards.

Since both players have the same number of awards in both categories, they can be considered equally valuable based on this data.

Final Answer: The most valuable players are Cheteshwar Pujara and Ravindra Jadeja, each with 3 MOTM awards and 4 MOTS awards.

> Finished chain.

```
Out[59]: {'input': 'Who is the most valuable player?',
          'output': 'The most valuable players are Cheteshwar Pujara and Ravindra Jadeja, each with 3 MOTM awards and 4 MOTS awards.'}
```

```
In [64]: agent1.invoke('Name a few under recognized players with significant runs
```

> Entering new AgentExecutor chain...

Thought: To identify under recognized players with significant runs or wickets, I need to analyze the data in `df1` to find players who have scored a substantial number of runs or taken a significant number of wickets. I will first check the structure of `df1` to understand how to aggregate runs and wickets for each player.

Action: python_repl_ast

Action Input: `df1.info()`<class 'pandas.core.frame.DataFrame'>

RangeIndex: 30968 entries, 0 to 30967

Data columns (total 23 columns):

#	Column	Non-Null Count	Dtype
0	match_id	30968 non-null	int64
1	season	30968 non-null	object
2	start_date	30968 non-null	object
3	venue	30968 non-null	object
4	innings	30968 non-null	int64
5	ball	30968 non-null	float64
6	batting_team	30968 non-null	object
7	bowling_team	30968 non-null	object
8	striker	30968 non-null	object
9	non_striker	30968 non-null	object
10	bowler	30968 non-null	object
11	runs_off_bat	30968 non-null	int64
12	extras	30968 non-null	int64
13	wides	37 non-null	float64
14	noballs	97 non-null	float64
15	byes	108 non-null	float64
16	legbyes	158 non-null	float64
17	penalty	0 non-null	float64
18	wicket_type	517 non-null	object
19	player_dismissed	517 non-null	object
20	other_wicket_type	0 non-null	float64
21	other_player_dismissed	0 non-null	float64
22	player_nationality	30968 non-null	object

dtypes: float64(8), int64(4), object(11)

memory usage: 5.4+ MB

The `df1` dataframe contains information about each ball in a cricket match, including runs scored and wickets taken. To identify under recognized players, I need to:

1. Aggregate the total runs scored by each player (striker).
2. Count the number of wickets taken by each bowler.

I will perform these aggregations and then identify players who have significant contributions but might not be widely recognized.

Action: python_repl_ast

Action Input: `"""`

`# Aggregate total runs scored by each player`

`runs_by_player = df1.groupby('striker')['runs_off_bat'].sum().reset_index()`

`# Count the number of wickets taken by each bowler`

`wickets_by_bowler = df1[df1['wicket_type'].notnull()].groupby('bowler').size().reset_index(name='wickets')`

`# Display the top players by runs and wickets`

```
runs_by_player.sort_values(by='runs_off_bat', ascending=False).head(10), wickets_by_bowler.sort_values(by='wickets', ascending=False).head(10)
```

```
"""(          striker  runs_off_bat
7          CA Pujara          1337
45         SPD Smith           957
20    M Labuschagne           708
54          V Kohli           703
2          AM Rahane           683
39         RR Pant           624
51         TM Head           534
52         UT Khawaja          531
17         KL Rahul           488
38         RG Sharma          477,          bowler  wickets
14          NM Lyon           75
16          R Ashwin           67
17         RA Jadeja           62
15         PJ Cummins          48
7         JR Hazlewood          40
6          JJ Bumrah           33
9          MA Starc           32
22         UT Yadav           26
11  Mohammed Shami           25
19    SNJ O'Keefe          19)
```

The analysis shows the top players by runs and wickets. To identify under recognized players, we should focus on those who are not typically highlighted in media or awards but have significant contributions. From the data:

- **Batsmen with significant runs:**
 - M Labuschagne (708 runs)
 - AM Rahane (683 runs)
 - RR Pant (624 runs)
- **Bowlers with significant wickets:**
 - PJ Cummins (48 wickets)
 - JR Hazlewood (40 wickets)
 - SNJ O'Keefe (19 wickets)

These players have made substantial contributions in terms of runs or wickets and might not always be the first names mentioned in discussions about top performers.

Final Answer: Under recognized players with significant contributions include M Labuschagne, AM Rahane, RR Pant (batsmen), and PJ Cummins, JR Hazlewood, SNJ O'Keefe (bowlers).

> **Finished chain.**

```
Out[64]: {'input': 'Name a few under recognized players with significant runs or wickets',
          'output': "Under recognized players with significant contributions include M Labuschagne, AM Rahane, RR Pant (batsmen), and PJ Cummins, JR Hazlewood, SNJ O'Keefe (bowlers)."}

```

DataFrame Agent with Plotting

This block again demonstrates how LLM can seamlessly analyze data and generate visualizations from natural language queries. Here it creates a bar chart to visualize the results, highlighting CA Pujara as the leading run-scorer in Adelaide, followed by AM Rahane and TD Paine. This capability to process queries, perform data

manipulation, and produce clear visual insights, such as bar charts, showcases the LLM's potential to simplify complex analytical tasks and make data exploration more accessible.

```
In [65]: agent2 = create_pandas_dataframe_agent(llm, ball_data, verbose=True, allo
```

```
In [67]: agent2.run("Plot the highest run scorers in Adelaide.")
```

> Entering new AgentExecutor chain...

To plot the highest run scorers in Adelaide, we need to filter the dataframe for matches played at venues in Adelaide and then aggregate the runs scored by each player. Finally, we can plot the top run scorers. Let's proceed with these steps.

Thought: I will filter the dataframe for rows where the venue contains "Adelaide", then group by the 'striker' column to sum the 'runs_off_bat'. After that, I will sort the results to find the top run scorers and plot the m.

Action: python_repl_ast

Action Input:

```
```python
```

```
import matplotlib.pyplot as plt
```

```
Filter for matches played in Adelaide
```

```
adelaide_matches = df[df['venue'].str.contains('Adelaide', case=False, na=False)]
```

```
Group by 'striker' and sum the 'runs_off_bat'
```

```
run_scorers = adelaide_matches.groupby('striker')['runs_off_bat'].sum()
```

```
Sort the run scorers in descending order
```

```
top_run_scorers = run_scorers.sort_values(ascending=False).head(10)
```

```
Plot the top run scorers
```

```
top_run_scorers.plot(kind='bar', figsize=(10, 6), title='Top Run Scorers in Adelaide')
```

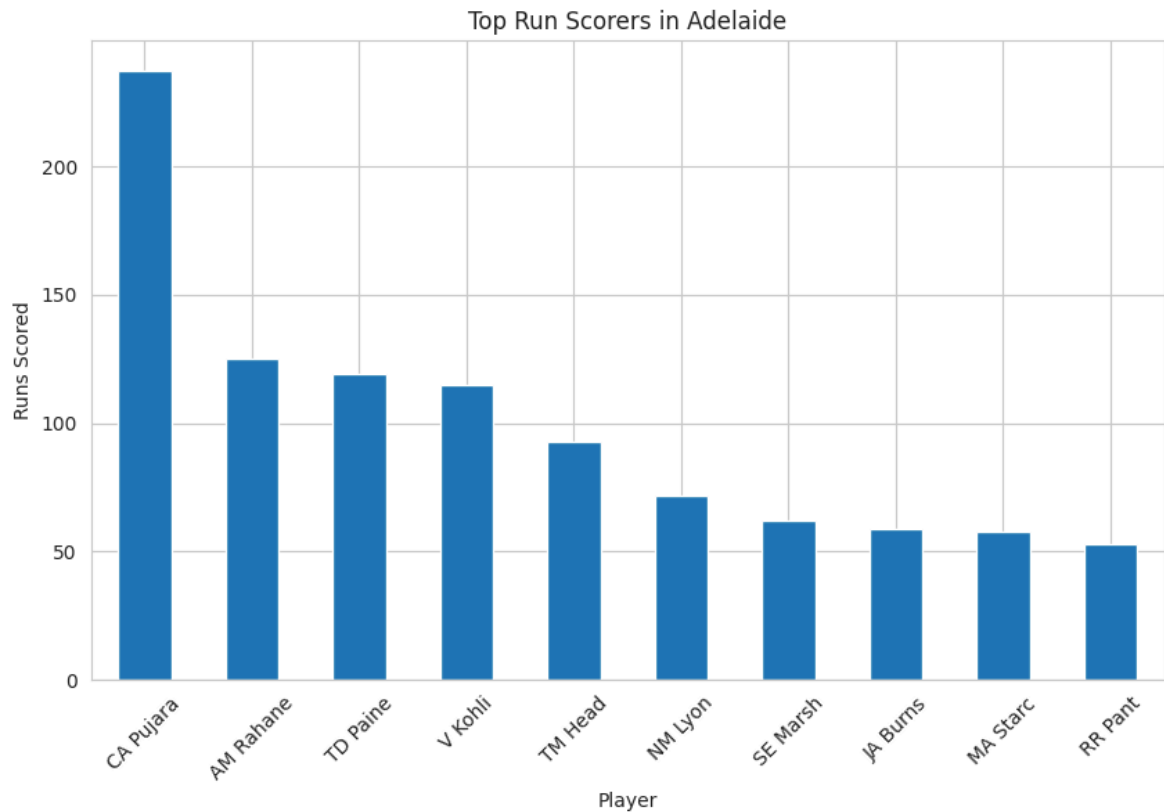
```
plt.xlabel('Player')
```

```
plt.ylabel('Runs Scored')
```

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

```
plt.show()
```

```
```
```



I now know the final answer.

Final Answer: The plot of the highest run scorers in Adelaide has been generated, showing the top 10 players with the most runs scored at venues in Adelaide.

> Finished chain.

Out[67]: 'The plot of the highest run scorers in Adelaide has been generated, showing the top 10 players with the most runs scored at venues in Adelaide.'

In [68]: `agent2.invoke("Plot V Kohli's runs for each venue.")`

> Entering new AgentExecutor chain...

To plot V Kohli's runs for each venue, we need to filter the dataframe for rows where V Kohli is the striker and then group the data by venue to sum the runs scored by him. Finally, we can plot the results. Let's execute these steps.

Thought: I will filter the dataframe for rows where V Kohli is the striker, group by venue, sum the runs, and then plot the results.

Action: python_repl_ast

Action Input:

```
```python
```

```
import matplotlib.pyplot as plt
```

```
Filter the dataframe for rows where V Kohli is the striker
```

```
kohli_runs = df[df['striker'] == 'V Kohli']
```

```
Group by venue and sum the runs
```

```
kohli_runs_by_venue = kohli_runs.groupby('venue')['runs_off_bat'].sum()
```

```
Plot the results
```

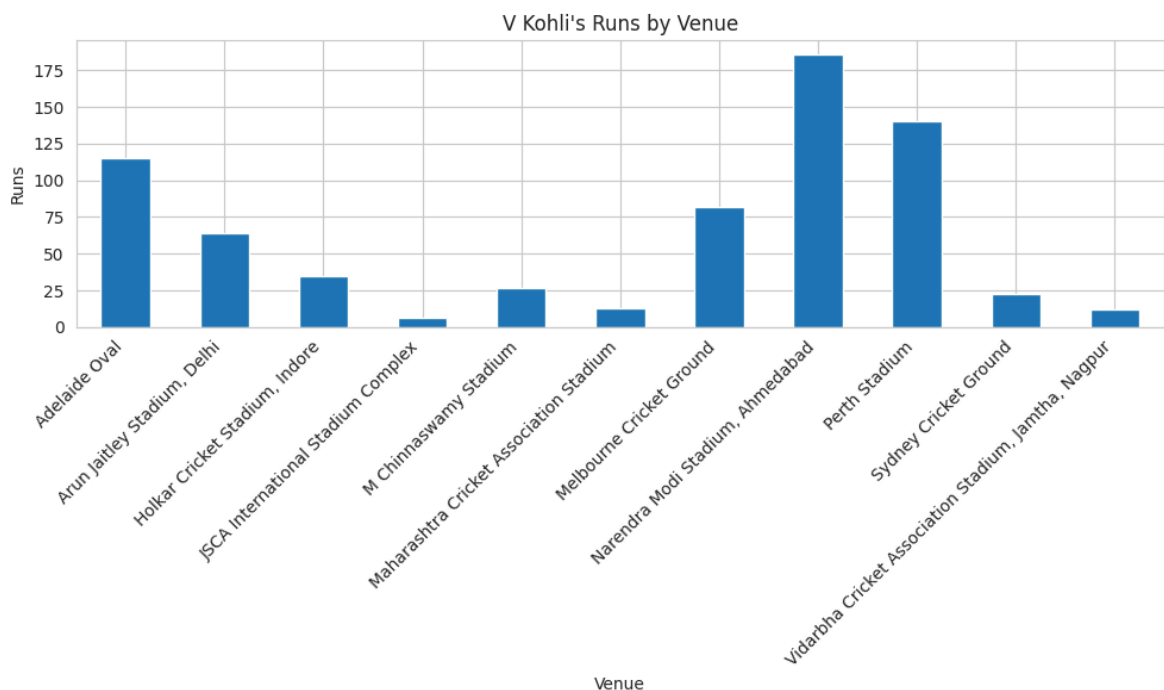
```
kohli_runs_by_venue.plot(kind='bar', title="V Kohli's Runs by Venue", ylabel='Runs', xlabel='Venue', figsize=(10, 6))
```

```
plt.xticks(rotation=45, ha='right')
```

```
plt.tight_layout()
```

```
plt.show()
```

```
```
```



I now know the final answer.

Final Answer: The plot of V Kohli's runs for each venue has been generated, showing the total runs he scored at each venue.

> Finished chain.

```
Out[68]: {'input': "Plot V Kohli's runs for each venue.",  
          'output': "The plot of V Kohli's runs for each venue has been generated, showing the total runs he scored at each venue."}
```

```
In [69]: agent2.run("Plot the comparison between Cheteshwar Pujara and Steve Smith")
```

> Entering new AgentExecutor chain...

To compare Cheteshwar Pujara and Steve Smith, we need to analyze their performance data from the dataframe. This could involve comparing their total runs, average runs, or other relevant statistics. Let's start by filtering the dataframe for records related to these two players and then calculate some basic statistics for comparison.

Thought: I will filter the dataframe for records where the striker is either Cheteshwar Pujara or Steve Smith, and then calculate their total runs and average runs. After that, I will plot the comparison.

Action: Filter the dataframe for Cheteshwar Pujara and Steve Smith, calculate their total and average runs, and plot the comparison.

Action Input:

```
```python
import matplotlib.pyplot as plt

Filter the dataframe for Cheteshwar Pujara and Steve Smith
pujara_data = df[df['striker'] == 'CA Pujara']
smith_data = df[df['striker'] == 'SPD Smith']

Calculate total runs
pujara_total_runs = pujara_data['runs_off_bat'].sum()
smith_total_runs = smith_data['runs_off_bat'].sum()

Calculate average runs
pujara_average_runs = pujara_data['runs_off_bat'].mean()
smith_average_runs = smith_data['runs_off_bat'].mean()

Plot the comparison
players = ['Cheteshwar Pujara', 'Steve Smith']
total_runs = [pujara_total_runs, smith_total_runs]
average_runs = [pujara_average_runs, smith_average_runs]

fig, ax = plt.subplots(1, 2, figsize=(12, 5))

Total runs plot
ax[0].bar(players, total_runs, color=['blue', 'orange'])
ax[0].set_title('Total Runs')
ax[0].set_ylabel('Runs')

Average runs plot
ax[1].bar(players, average_runs, color=['blue', 'orange'])
ax[1].set_title('Average Runs')
ax[1].set_ylabel('Runs')
```

```
plt.tight_layout()
```

```
plt.show()
```

```Filter the dataframe for Cheteshwar Pujara and Steve Smith, calculate their total and average runs, and plot the comparison. is not a valid tool, try one of [python\_repl\_ast].I need to execute the Python code using the `python\_repl\_ast` tool to filter the dataframe, calculate the statistics, and plot the comparison.

Action: python_repl_ast

Action Input:

```
```python
import matplotlib.pyplot as plt
```

```

Filter the dataframe for Cheteshwar Pujara and Steve Smith
pujara_data = df[df['striker'] == 'CA Pujara']
smith_data = df[df['striker'] == 'SPD Smith']

Calculate total runs
pujara_total_runs = pujara_data['runs_off_bat'].sum()
smith_total_runs = smith_data['runs_off_bat'].sum()

Calculate average runs
pujara_average_runs = pujara_data['runs_off_bat'].mean()
smith_average_runs = smith_data['runs_off_bat'].mean()

Plot the comparison
players = ['Cheteshwar Pujara', 'Steve Smith']
total_runs = [pujara_total_runs, smith_total_runs]
average_runs = [pujara_average_runs, smith_average_runs]

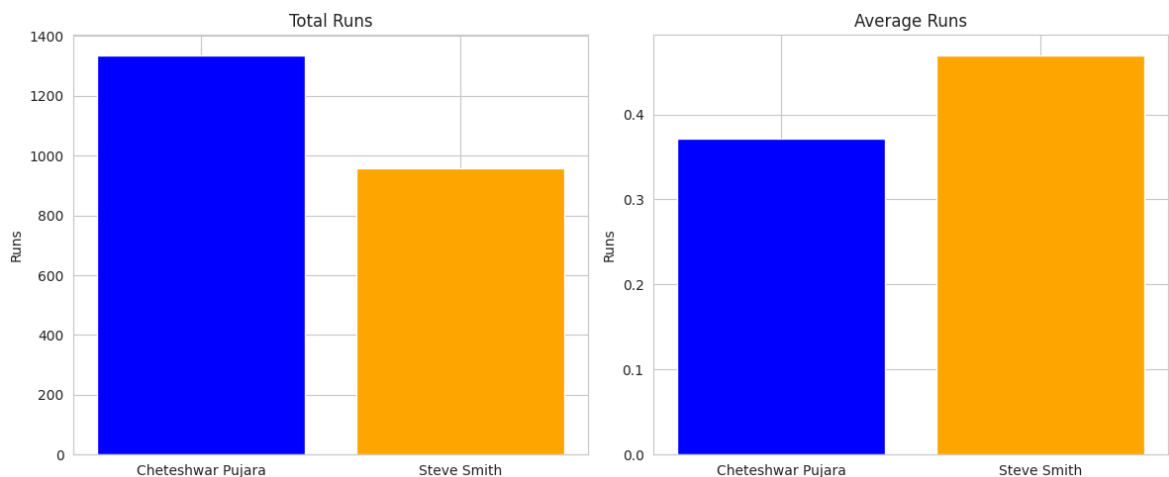
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

Total runs plot
ax[0].bar(players, total_runs, color=['blue', 'orange'])
ax[0].set_title('Total Runs')
ax[0].set_ylabel('Runs')

Average runs plot
ax[1].bar(players, average_runs, color=['blue', 'orange'])
ax[1].set_title('Average Runs')
ax[1].set_ylabel('Runs')

plt.tight_layout()
plt.show()
` ``

```



I have executed the code to filter the dataframe for Cheteshwar Pujara and Steve Smith, calculate their total and average runs, and plot the comparison. The plots should now be displayed, showing the total and average runs for both players.

**Final Answer:** The comparison between Cheteshwar Pujara and Steve Smith has been plotted, showing their total and average runs.

> Finished chain.

Out[69]: 'The comparison between Cheteshwar Pujara and Steve Smith has been plotted, showing their total and average runs.'

# Conclusion

## Comprehensive Conclusion for My BGT Analysis Project

### Analytical Insights

#### Player Performance and Dynamics

- **Batsmen:** My analysis identified Cheteshwar Pujara and Steve Smith as standout performers. Pujara's partnerships were instrumental in stabilizing India's innings during the Border-Gavaskar Trophy (BGT), while Smith's adaptability across different venues highlighted his technical brilliance and consistency.
- **Bowlers:** Nathan Lyon emerged as the highest wicket-taker, supported by R. Ashwin and R. Jadeja, underlining the strategic importance of spin-friendly conditions in the series.
- **Key Matchups:** Game-changing dynamics, such as Nathan Lyon's success against Pujara, emphasized critical bowler-batsman rivalries that shaped the outcomes of key matches.

#### Venue Analysis

- Venue-specific insights revealed Pujara's dominance at the Sydney Cricket Ground and Smith's success in Nagpur, emphasizing the importance of leveraging players in their favored conditions.
- The analysis highlighted India's consistent home dominance and Australia's adaptability and resilience in away games.

#### Trends in Match Outcomes

- India emerged as the dominant team in the series, with closely contested matches often resulting in draws. These results reflect the strategic depth and competitive nature of the rivalry.

---

## Application of Machine Learning (ML), Large Language Models (LLM), and AI Techniques

### Machine Learning Models

- **Random Forest:**
  - Achieved an accuracy of 86%, proving to be the most effective model for predicting match outcomes.
  - Provided interpretability through feature importance, enabling actionable strategies based on factors like venue, innings, team totals, captains, and host.

- **Support Vector Machine (SVM):**
  - Effectively captured nonlinear relationships with 81% accuracy but was limited by class imbalance issues.
- **Logistic Regression:**
  - Served as a reliable baseline model, offering simplicity and interpretability despite lower performance compared to ensemble methods.
- **Key Insights:**
  - Random Forest stood out for its robustness in handling multi-class cricket data, while hyperparameter tuning mitigated class imbalance challenges.

## AI for Data Processing and Analysis

- Employed advanced feature engineering techniques, such as one-hot encoding of categorical variables like venue and captains, ensuring compatibility with ML algorithms.
- Automated Exploratory Data Analysis (EDA) with AI enabled the discovery of key player insights, reducing the need for manual effort and improving efficiency.

## LLM Integration – Conversational Cricket Analytics

- Leveraged **Large Language Models (LLMs)**, such as LangChain, to enable conversational data exploration:
    - Users could ask natural language queries like "**Who are the unsung heroes?**" or "**Plot the top run-scorers in Adelaide,**" and the LLM dynamically generated Python code to provide the answers.
    - LLMs simplified complex analysis workflows by sharing their thought process, executing code, and transparently explaining results.
    - Generated visualizations, such as bar charts, enhanced the interpretability of insights for both technical and non-technical users.
- 

## Achievements of AI Integration

### Predictive Models for Match Outcomes

- ML models predicted match outcomes with a weighted F1-score of 0.95, offering actionable insights for team strategy and selection.
- Random Forest highlighted venue-specific and innings-specific factors that influenced match results, making predictions more actionable.

### Enhanced Automation

- AI automated data preprocessing, feature engineering, and visualization, allowing more focus on strategic insights and advanced analysis.

### Conversational Intelligence with LLMs

- LLMs bridged the gap between natural language queries and technical analysis by acting as conversational assistants. They not only processed data efficiently but also explained their logic, making data exploration intuitive and user-friendly.
  - The ability to query datasets conversationally and generate visual outputs, like bar charts or filtered tables, transformed how data is consumed and understood.
- 

## Limitations and Future Enhancements

- **Class Imbalance:** Addressing this with techniques like SMOTE for synthetic data generation to ensure balanced model predictions.
  - **Real-Time Data Integration:** Incorporating live player performance and weather conditions to enhance predictive capabilities.
  - **Explainable AI (XAI):** Improving model transparency to validate predictions and refine decision-making strategies further.
  - **Interactive Dashboards:** Developing interactive dashboards with tools like Plotly to make data exploration dynamic and user-friendly.
- 

## Final Thoughts

This project has showcased the transformative potential of **Machine Learning, AI, and LLMs** in cricket analytics. By integrating advanced models like Random Forest and tools like LangChain, I bridged traditional cricket insights with modern, conversational analytics. The ability to interact with data conversationally, ask questions, and receive clear insights—including visualizations—has redefined how complex datasets are analyzed and understood.

Looking ahead, the integration of real-time data and interactive dashboards can elevate cricket analytics further, making it a critical tool for teams, analysts, and fans alike. This project demonstrates how **LLMs enable not just analysis but also intelligent, user-friendly engagement with data**, setting a new benchmark for data-driven decision-making in sports analytics.