data-premier-league

February 3, 2024

#Data Premier League

0.0.1 Data Visualization and Prediction

Scenario: The 2024 Franchise Cricket Championship Prediction Challenge Get ready for the ultimate challenge in cricket analysis – the "2024 Franchise Cricket Championship Prediction Challenge"! The cricketing world is abuzz with anticipation as ten top-tier teams are set to battle it out in a round-robin format. Your task is to predict and answer the burning question on everyone's minds:

0.0.2 "Who will win the 2024 Cricket Season ?"

1 The 2024 Franchise Cricket Championship Prediction Challenge

From the given problem statement and the datasets we started by exploring the various attributes and records that are associated with respect to data represented in it. The given data are basically of three different categories like Batsman, Bowlers and All Rounders for various teams.

Initially before proceeding with the given problem statement we pre-processed our data and removed the empty rows and columns and in certain cases we removed the null values and in some other cases we filled the null entries with mean value respectively. And for our easier computation we created a folder called Teams and put all the data of each team inside the folder. Further while performing the analysis we automated it. For handling these data files we used pandas and for computations we used numpy and for the visualization we used matplotlib libraries wherever it is necessary.

2 Problem Statement: "Who will win the 2024 Cricket Season ?"

Participants are invited to employ their imagination, prediction skills, and visualization techniques with the given dataset. As you explore the intricacies of the cricket data through the questions provided, envision and predict compelling insights. Let your creativity and analytical prowess shine as you navigate through the diverse dimensions of the cricketing world encapsulated in this dataset.

[71]:
```
import pandas as pd
import matplotlib.pyplot as plt
import math
import numpy as np
import os
from sklearn.ensemble import RandomForestRegressor
```

[3]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

3 Data Processing (Exploration and Cleaning)

Removing Missing Values: Rows containing missing values (NaN) are dropped from the DataFrame. Removing Duplicate Rows: Duplicate rows in the DataFrame, based on the 'Match' column, are removed, retaining only the first occurrence. Converting 'Date' to Datetime: The 'Date' column in the DataFrame is converted to datetime format. If any errors occur during this conversion, they are handled by coercing them to NaT (Not a Time) values.

The cleaned DataFrame is then returned by the function. This code is designed to preprocess batting data, ensuring data integrity by handling missing values, removing duplicates, and converting the 'Date' column to a consistent datetime format.

[92]:
```
def clean_batting_data(df):
    df = df.dropna()
    df = df.drop_duplicates(subset='Match')
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
    return df
```

[93]:
```
def clean_bowling_data(df):
    df = df.dropna()
    df = df.drop_duplicates(subset='Match')
    return df
```

[94]:
```
def load_and_clean_data(folder_path):
    team_dfs = {}

    for team_folder in os.listdir(folder_path):
        team_path = os.path.join(folder_path, team_folder)
        if os.path.isdir(team_path):
            team_dfs[team_folder] = {}

            for role_folder in os.listdir(team_path):
                role_path = os.path.join(team_path, role_folder)
                if os.path.isdir(role_path):
                    team_dfs[team_folder][role_folder] = {}

                    for player_file in os.listdir(role_path):
                        player_name, file_extension = os.path.
 ↪splitext(player_file)
                        if file_extension.lower() == '.xlsx':
                            player_path = os.path.join(role_path, player_file)

                            xls = pd.ExcelFile(player_path)
```

```
                            player_dfs = {}

                            for sheet_name in xls.sheet_names:
                                df = pd.read_excel(xls, sheet_name=sheet_name)

                                if role_folder.lower() == 'batsmen' or␣
↪(role_folder.lower() == 'allrounders' and 'a' in sheet_name):
                                    df = clean_batting_data(df)

                                elif role_folder.lower() == 'bowlers' or␣
↪(role_folder.lower() == 'allrounders' and 'b' in sheet_name):
                                    df = clean_bowling_data(df)

                                player_dfs[sheet_name] = df

                            team_dfs[team_folder][role_folder][player_name] =␣
↪player_dfs

    return team_dfs
```

[95]:
```
folder_path = '/content/drive/MyDrive/cyberlab/Team'
cleaned_data = load_and_clean_data(folder_path)
print(cleaned_data)
```

```
{'CSK': {'ALLROUNDERS': {'CSK_ALLROUNDERS': {'Ravindra Jadeja a': Empty
DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Ravindra Jadeja b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mitchell Santner a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Mitchell Santner b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Moeen Ali a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Moeen Ali b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Shivam Dube a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Shivam Dube b': Empty DataFrame
```

Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Shardul Thakur a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Shardul Thakur b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Daryl Mitchell a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Daryl Mitchell b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'CSK_BATSMEN': {'MS Dhoni': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Devon Conway': Empty DataFrame
Columns: [Match, Match.1, Innings, Date, M/Inns, Posn, Versus, Ground, How
Dismissed, Runs, B/F, S/R, Unnamed: 12, Aggr, Avg, S/RC]
Index: [], 'Ruturaj Gaikwad': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Ajinkya Rahane': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'CSK_BOWLERS': {'Rajvardhan Hangargekar': Empty
DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Deepak Chahar': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Maheesh Theekshana': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mukesh Choudhary': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mustafizur Rahman': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Prashant Solanki': Empty DataFrame
Columns: [Unnamed: 0, Unnamed: 1, Unnamed: 2, Unnamed: 3, Unnamed: 4, Unnamed:
5, Unnamed: 6, Unnamed: 7, Unnamed: 8, Unnamed: 9, Unnamed: 10, Unnamed: 11,
Unnamed: 12]
Index: [], 'Simarjeet Singh': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,

Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Tushar Deshpande': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Matheesha Pathirana': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'DC': {'ALLROUNDERS': {'DC_ALLROUNDERS': {'Axar Patel a': Empty
DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Axar Patel b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Lalit Yadav a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Lalit Yadav b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mitchell Marsh a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Mitchell Marsh b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'DC_BATSMEN': {'Rishabh Pant': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'David Warner': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Prithvi Shaw': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Yash Dhull': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Abishek Porel': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Harry Brook': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Tristan Stubbs': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'DC_BOWLERS': {'Praveen Dubey': Empty DataFrame

Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Anrich Nortje': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Kuldeep Yadav': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Khaleel Ahmed': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Lungi Ngidi': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Ishant Sharma': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mukesh Kumar': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Jhye Richardson': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Rasikh Salam': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'GT': {'ALLROUNDERS': {'GT_ALLROUNDERS': {'Abhinav Manohar a':
Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Sai Sudharsan a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Darshan Nalkande a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Darshan Nalkande b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Vijay Shankar a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Vijay Shankar b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Shahrukh Khan a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]

Index: []}}, 'BATSMEN': {'GT_BATSMEN': {'David Miller': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Shubman Gill': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Mathew Wade': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Wriddhiman Saha': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Kane Williamson': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'GT_BOWLERS': {'Jayant Yadav': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Rahul Tewatia': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mohammed Shami': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Kartik Tyagi': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Noor Ahmad': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Sai Kishore': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Umesh Yadhav': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Rashid Khan': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Joshua Little': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mohit Sharma': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'KKR': {'ALLROUNDERS': {'KKR_ALLROUNDERS': {'Anukul Roy a': Empty
DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,

Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Anukul Roy b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Ramandeep Singh a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Ramandeep Singh b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Andre Russell a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Andre Russell b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Venkatesh Iyer a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Venkatesh Iyer b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'KKR_BATSMEN': {'Nitish Rana': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rinku Singh': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rahmanullah Gurbaz': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Shreyas Iyer': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Sherfane Rutherford': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'KS Bharat': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Manish Pandey': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Jason Roy': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'KKR_BOWLERS': {'Suyash Sharma': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,

Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mujeeb Ur Rahman': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Harshit Rana': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Sunil Narine': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Vaibhav Arora': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Varun Chakravarthy': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mitchell Starc': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Chetan Sakariya': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'LSG': {'ALLROUNDERS': {'LSG_ALLROUNDERS': {'Ayush Badoni a':
Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Ayush Badoni b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Deepak Hooda a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Deepak Hooda b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Krishnappa Gowtham a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Krishnappa Gowtham b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Krunal Pandya a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Krunal Pandya b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Kyle Mayers a': Empty DataFrame

Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Kyle Mayers b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Marcus Stoinis a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Marcus Stoinis b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Arshad Khan a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Arshad Khan b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Prerak Mankad a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Yudhvir Singh a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Yudhvir Singh b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'David Willey a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'David Willey b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'LSG_BATSMEN': {'KL Rahul': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Devdutt Padikkal': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Quinton de Kock': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Nicholas Pooran': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Ashton Turner': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'LSG_BOWLERS': {'Shivam Mavi': Empty DataFrame

Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mark Wood': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mohsin Khan': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Ravi Bishnoi': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Yash Thakur': Empty DataFrame
Columns: [Unnamed: 0, Unnamed: 1, Unnamed: 2, Unnamed: 3, Unnamed: 4, Unnamed:
5, Unnamed: 6, Unnamed: 7, Unnamed: 8, Unnamed: 9, Unnamed: 10, Unnamed: 11,
Unnamed: 12]
Index: [], 'Amit Mishra': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'MI': {'ALLROUNDERS': {'MI_ALLROUNDERS': {'Mohammad Nabi a': Empty
DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Mohammad Nabi b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Arjun Tendulkar a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Arjun Tendulkar b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Romario Shepherd a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Romario Shepherd b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Nehal Wadhera a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Hardik Pandya a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Hardik Pandya b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'MI_BATSMEN': {'Rohit Sharma': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,

Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Dewald Brevis': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Suryakumar Yadav': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Ishan Kishan': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Tilak Varma': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Tim David': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Vishnu Vinod': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'MI_BOWLERS': {'Jasprit Bumrah': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Piyush Chawla': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Kumar Kartikeya': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Akash Madhwal': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Jason Behrendorff': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Shreyas Gopal': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'PBKS': {'ALLROUNDERS': {'PBKS_ALLROUNDERS': {'Chris Woakes a':
Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Chris Woakes b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Atharva Taide a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rishi Dhawan a': Empty DataFrame

Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rishi Dhawan b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Sam Curran a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Sam Curran b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Sikandar Raza a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Sikandar Raza b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'PBKS_BATSMEN': {'Shikhar Dhawan': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Jitesh Sharma': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Jonny Bairstow': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Prabhsimran Singh': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Liam Livingstone': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rilee Rossouw': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Shashank Singh': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'PBKS_BOWLERS': {'Harpreet Brar': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Arshdeep Singh': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Kagiso Rabada': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Nathan Ellis': Empty DataFrame

Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Rahul Chahar': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Harshal Patel': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'RCB': {'ALLROUNDERS': {'RCB_ALLROUNDERS': {'Glenn maxwell a':
Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Glenn maxwell b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mahipal Lomror a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Mahipal Lomror b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Karn sharma a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Karn sharma b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Cameron green a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Cameron green b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Swapnil singh a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Swapnil singh b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'RCB_BATSMEN': {'Faf Duplesis': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rajat Patidar': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Virat Kohli': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]

Index: [], 'Anuj Rawat': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Dinesh Karthik': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Suyash Prabhudessai': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'RCB_BOWLERS': {'Akash Deep ': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Alzarri Joseph': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Lockie Ferguson': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mohammed Siraj': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Yash Dayal ': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Tom Curran': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Reece Topley': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'RR': {'ALLROUNDERS': {'RR_ALLROUNDERS': {'Ravichandran Ashwin a':
Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Ravichandran Ashwin b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}, 'BATSMEN': {'RR_BATSMEN': {'Sanju Samson': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Jos Buttler': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Shimron Hetmeyer': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Yashasvi Jaiswal': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,

Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Dhruv Jurel': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Riyan Parag': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rovman Powell': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'RR_BOWLERS': {'Avesh Khan': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Kuldeep Sen': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Navdeep Saini': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Prasidh Krishna': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Sandeep Sharma': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Trent Boult': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Yuzvendra Chahal': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Adam Zampa': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}, 'SRH': {'ALLROUNDERS': {'SRH_ALLROUNDERS': {'Abhishek Sharma a':
Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Abhishek Sharma b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Marco Jansen a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Marco Jansen b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Washington Sundar a': Empty DataFrame

Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Washington Sundar b': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Sanvir Singh a': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BATSMEN': {'SRH_BATSMEN': {'Abdul Samad': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Anmolpreet Singh': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Heinrich Klaasen': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Rahul Tripathi': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Mayank Agarwal': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Travis Head': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: [], 'Aiden Markram': Empty DataFrame
Columns: [Match, Innings, Date, M/Inns, Posn, Versus, Ground, How Dismissed,
Runs, B/F, S/R, Unnamed: 11, Aggr, Avg, S/RC]
Index: []}}, 'BOWLERS': {'SRH_BOWLERS': {'Akash Singh': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Shahbaz Ahmed': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Bhuvneshwar Kumar': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Fazalhaq Farooqi': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Jaydev Unadkat ': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Thangarasu Natarajan': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Umran Malik': Empty DataFrame

```
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Pat Cummins': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: [], 'Mayank Markande': Empty DataFrame
Columns: [Match, Date, M/Inns, Versus, Ground, Batsman Dismissed, Overs,
Wickets, Unnamed: 8, S/R, E/R, Wkts, Avg]
Index: []}}}}
```

4 1. Innings Distribution:

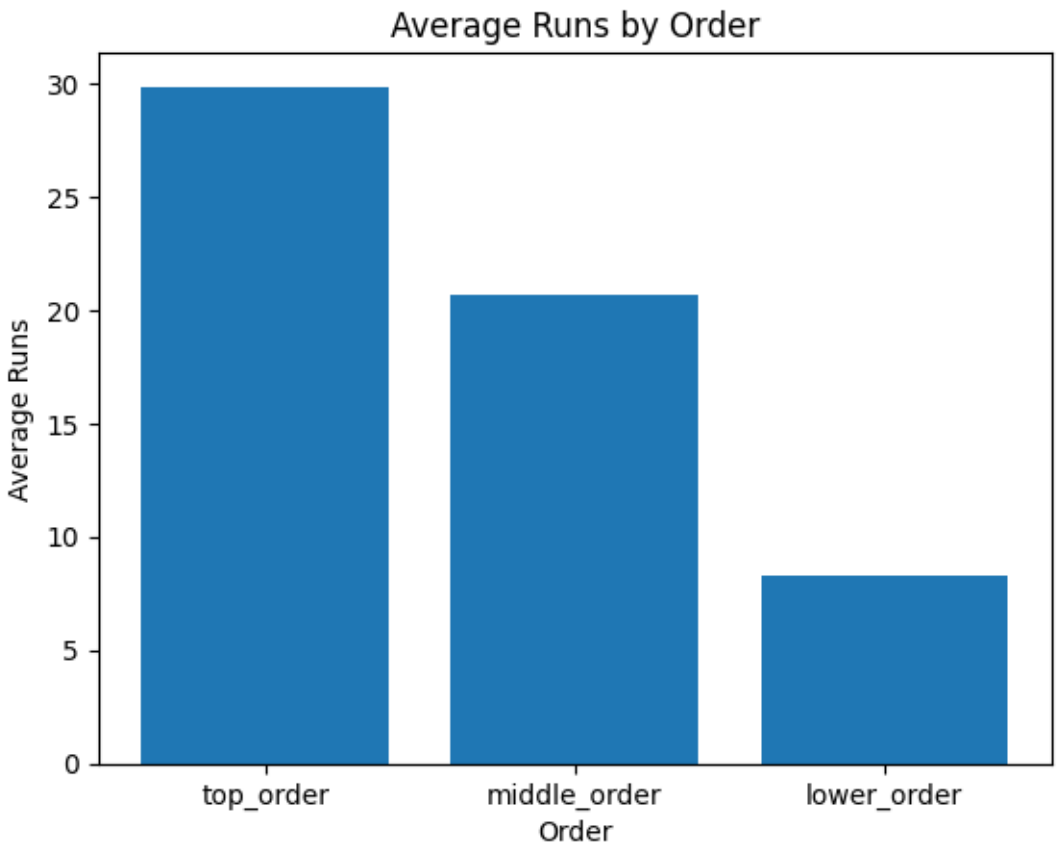
How is the distribution of innings for players in the dataset? Are they more inclined towards top-order, middle-order, or lower-order positions?

The code processes batting statistics for cricket teams, extracting data from Excel files for both batsmen and all-rounders. It computes the average runs for players in different batting positions, categorizing them into 'top_order,' 'middle_order,' and 'lower_order.' The script then identifies the batting order with the highest average runs and visualizes the results using a bar chart. The core concept revolves around data aggregation, analysis, and visualization to determine the most productive batting order in terms of average runs.

[96]:
```
a={}
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    dfs = {}
    for sheet in xls.sheet_names:
        dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)

    for i in dfs:
        for index, row in dfs[i].iterrows():
            if row['Posn']!='-':
                row["Runs"]=str(row["Runs"]).replace("*","")
                if row['Posn'] not in a:
                    a[row['Posn']]=[int(row['Runs']),1]
                else:
                    a[row['Posn']]=[ a[row['Posn']][0]+int(row['Runs']),␣
 ↪a[row['Posn']][1]+1]

teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/ALLROUNDERS/
 ↪{i}_ALLROUNDERS.xlsx')
    dfs = {}
    for sheet in xls.sheet_names:
```

```
        if sheet.split()[-1] =='a':
            dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)

    for i in dfs:
        for index, row in dfs[i].iterrows():
            if row['Posn']!='-':
                row["Runs"]=str(row["Runs"]).replace("*","")
                if row['Posn'] not in a:
                    a[row['Posn']]=[int(row['Runs']),1]
                else:
                    a[row['Posn']]=[ a[row['Posn']][0]+int(row['Runs']),␣
 ↪a[row['Posn']][1]+1]
order={'top_order':0,'middle_order':0,'lower_order':0}
for i in a:
    if i<=3:
        order['top_order']+=a[i][0]/a[i][1]
    elif i<=6:
        order['middle_order']+=a[i][0]/a[i][1]
    else:
        order['lower_order']+=a[i][0]/a[i][1]
for i in order:
    if i=='top_order':
        order[i]=order[i]/3
    elif i=='middle_order':
        order[i]=order[i]/3
    else:
        order[i]=order[i]/4

max_order = max(order, key=order.get)
print(f"The maximum average runs is from the {max_order} with␣
 ↪{order[max_order]} runs.")

plt.bar(order.keys(), order.values())
plt.xlabel('Order')
plt.ylabel('Average Runs')
plt.title('Average Runs by Order')
plt.show()
```

```
The maximum average runs is from the top_order with 29.84183541956652 runs.
```



5 2. Versus Opponent Analysis:

Which opponent has proven to be the most challenging for the players in terms of average runs scored and strike rate?

This code analyzes the performance of cricket batsmen against different teams. It calculates a combined score for each batsman against each opponent, considering both average runs and strike rate. The code then identifies the team against which each batsman performs the worst and the best. Additionally, it visualizes the average runs and strike rates for each batsman against different teams using a bar chart and a line plot. The key concept involves evaluating and comparing batsmen's performances against various opponents based on multiple performance metrics.

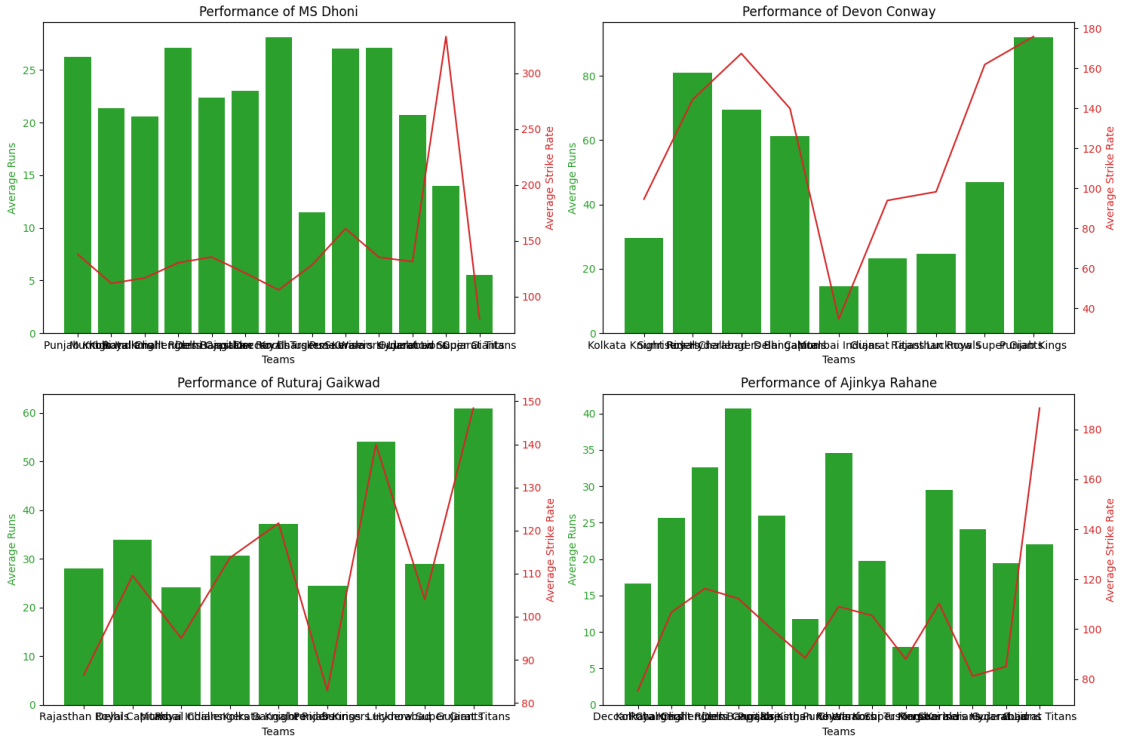
[97]:
```
worst={}
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    dfs = {}
    worst={}
    for sheet in xls.sheet_names:
```

```
        dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)

    for i in dfs:
        a={}
        for index, row in dfs[i].iterrows():
            if row['Versus']!='-':
                if row['Runs'] != '-':
                    if row['S/R'] != '-':
                        row["Runs"]=str(row["Runs"]).replace("*","")
                        if row['Versus'] not in a:
                            a[row['Versus']]=[int(row['Runs']),int(row['S/
↪R']),1]
                        else:
                            a[row['Versus']]=[␣
↪a[row['Versus']][0]+int(row['Runs']), a[row['Versus']][1]+int(row['S/R']),␣
↪a[row['Versus']][2]+1]
        worst[i]=a

    for i in worst:
        for j in worst[i]:
            worst[i][j]=[worst[i][j][0]/worst[i][j][2],worst[i][j][1]/
↪worst[i][j][2]]

    weight_runs = 0.7
    weight_sr = 0.3

    for batsman, data in worst.items():
        scores = {team: (weight_runs * stats[0] + weight_sr * stats[1], stats)␣
↪for team, stats in data.items()}

        worst_team = min(scores.items(), key=lambda x: x[1][0])
        best_team = max(scores.items(), key=lambda x: x[1][0])
        print(f"{batsman} performs worst against {worst_team[0]} with a score␣
↪of {worst_team[1][0]} (runs: {worst_team[1][1][0]}, strike rate:␣
↪{worst_team[1][1][1]}).")
        print(f"{batsman} performs best against {best_team[0]} with a score of␣
↪{best_team[1][0]} (runs: {best_team[1][1][0]}, strike rate:␣
↪{best_team[1][1][1]}).")
        print("\n")

    num_players = len(worst)
    num_cols = 2
    num_rows = math.ceil(num_players / num_cols)

    fig, axs = plt.subplots(num_rows, num_cols, figsize=(15, num_rows*5))
```

```
    for idx, (player, data) in enumerate(worst.items()):
        ax1 = axs[idx // num_cols, idx % num_cols]

        teams = list(data.keys())
        avg_runs = [team_data[0] for team_data in data.values()]
        avg_strike_rate = [team_data[1] for team_data in data.values()]

        color = 'tab:Green'
        ax1.set_xlabel('Teams')
        ax1.set_ylabel('Average Runs', color=color)
        ax1.bar(teams, avg_runs, color=color)
        ax1.tick_params(axis='y', labelcolor=color)

        ax2 = ax1.twinx()
        color = 'tab:red'
        ax2.set_ylabel('Average Strike Rate', color=color)
        ax2.plot(teams, avg_strike_rate, color=color)
        ax2.tick_params(axis='y', labelcolor=color)

        ax1.set_title(f'Performance of {player}')

    if num_players % num_cols != 0:
        for idx in range(num_players, num_rows * num_cols):
            fig.delaxes(axs.flatten()[idx])

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

```
MS Dhoni performs worst against Gujarat Titans with a score of 27.85 (runs: 5.5,
strike rate: 80.0).
MS Dhoni performs best against Lucknow Super Giants with a score of
109.69999999999999 (runs: 14.0, strike rate: 333.0).


Devon Conway performs worst against Mumbai Indians with a score of
20.666666666666664 (runs: 14.666666666666666, strike rate: 34.666666666666664).
Devon Conway performs best against Punjab Kings with a score of
117.19999999999999 (runs: 92.0, strike rate: 176.0).


Ruturaj Gaikwad performs worst against Punjab Kings with a score of 42.0 (runs:
24.5, strike rate: 82.83333333333333).
Ruturaj Gaikwad performs best against Gujarat Titans with a score of 87.08
(runs: 60.8, strike rate: 148.4).


Ajinkya Rahane performs worst against Kochi Tuskers Kerala with a score of 32.0
```

(runs: 8.0, strike rate: 88.0).
Ajinkya Rahane performs best against Gujarat Titans with a score of 71.94999999999999 (runs: 22.0, strike rate: 188.5).



Rishabh Pant performs worst against Rising Pune Supergiant with a score of 40.675 (runs: 18.25, strike rate: 93.0).
Rishabh Pant performs best against Gujarat Lions with a score of 86.8 (runs: 47.5, strike rate: 178.5).

David Warner performs worst against Kochi Tuskers Kerala with a score of 24.799999999999997 (runs: 8.0, strike rate: 64.0).
David Warner performs best against Gujarat Lions with a score of 92.34 (runs: 67.2, strike rate: 151.0).

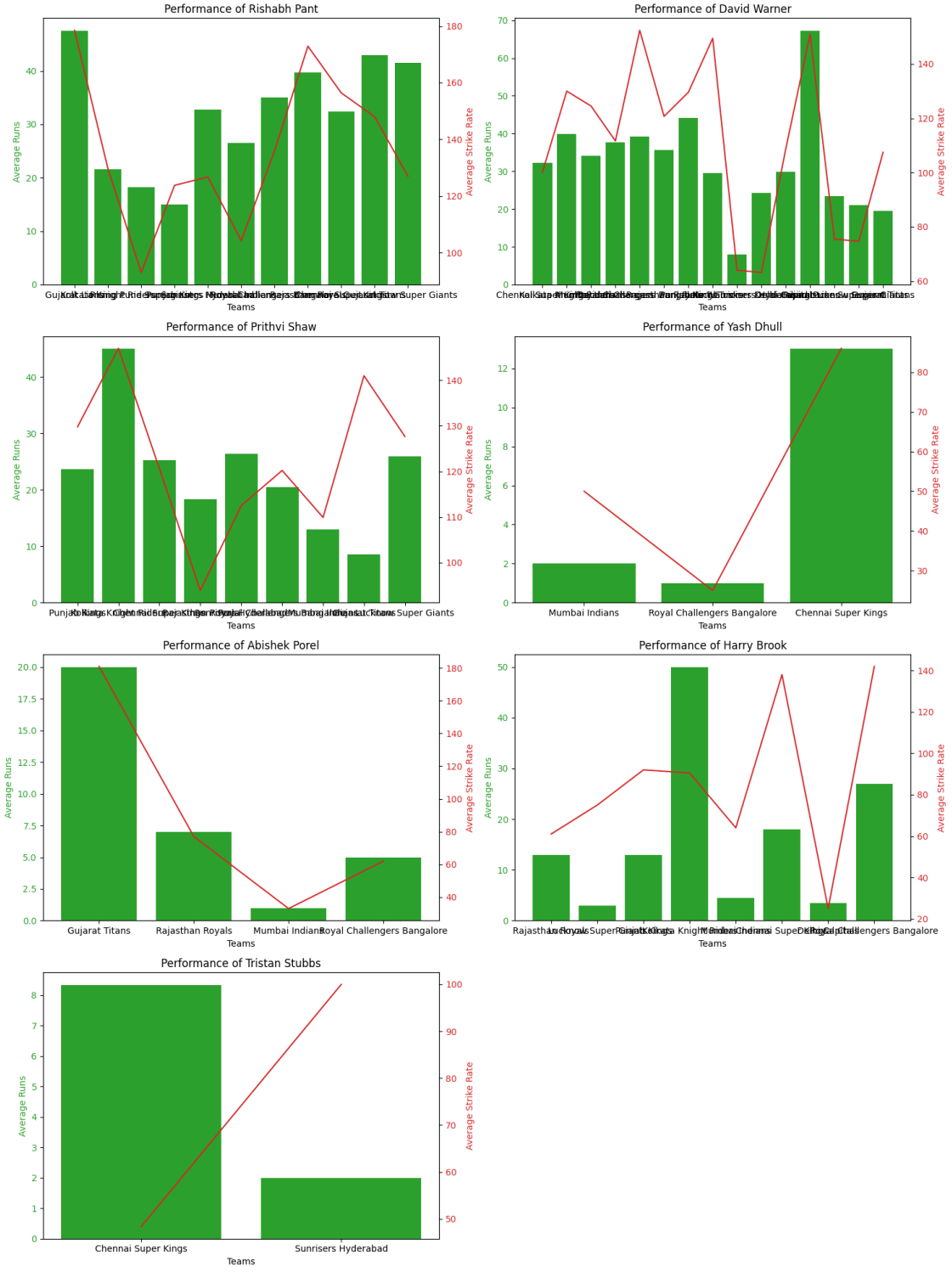
Prithvi Shaw performs worst against Rajasthan Royals with a score of 41.0 (runs: 18.333333333333332, strike rate: 93.88888888888889).
Prithvi Shaw performs best against Kolkata Knight Riders with a score of 75.6 (runs: 45.0, strike rate: 147.0).

Yash Dhull performs worst against Royal Challengers Bangalore with a score of 8.2 (runs: 1.0, strike rate: 25.0).
Yash Dhull performs best against Chennai Super Kings with a score of 34.9 (runs: 13.0, strike rate: 86.0).

Abishek Porel performs worst against Mumbai Indians with a score of 10.6 (runs: 1.0, strike rate: 33.0).
Abishek Porel performs best against Gujarat Titans with a score of 68.3 (runs: 20.0, strike rate: 181.0).

Harry Brook performs worst against Delhi Capitals with a score of 9.95 (runs: 3.5, strike rate: 25.0).
Harry Brook performs best against Kolkata Knight Riders with a score of 62.15 (runs: 50.0, strike rate: 90.5).

Tristan Stubbs performs worst against Chennai Super Kings with a score of 20.333333333333332 (runs: 8.333333333333334, strike rate: 48.333333333333336).
Tristan Stubbs performs best against Sunrisers Hyderabad with a score of 31.4 (runs: 2.0, strike rate: 100.0).



David Miller performs worst against Punjab Kings with a score of 40.133333333333326 (runs: 11.333333333333334, strike rate: 107.33333333333333). David Miller performs best against Pune Warriors with a score of 114.5 (runs:

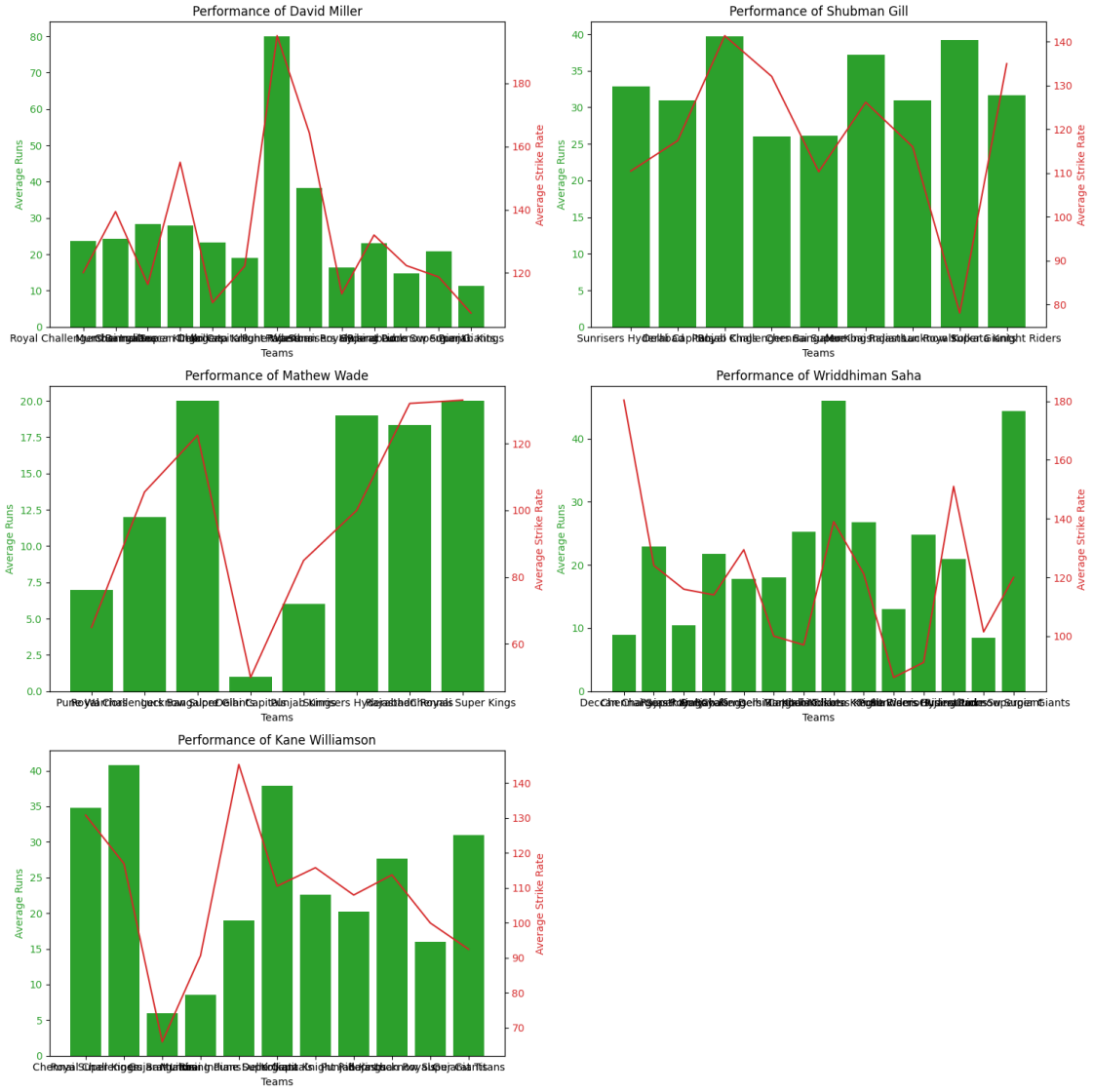
80.0, strike rate: 195.0).

Shubman Gill performs worst against Lucknow Super Giants with a score of 50.875 (runs: 39.25, strike rate: 78.0).
Shubman Gill performs best against Punjab Kings with a score of 70.21000000000001 (runs: 39.7, strike rate: 141.4).

Mathew Wade performs worst against Delhi Capitals with a score of 15.7 (runs: 1.0, strike rate: 50.0).
Mathew Wade performs best against Chennai Super Kings with a score of 53.9 (runs: 20.0, strike rate: 133.0).

Wriddhiman Saha performs worst against Pune Warriors with a score of 34.9 (runs: 13.0, strike rate: 86.0).
Wriddhiman Saha performs best against Kochi Tuskers Kerala with a score of 73.89999999999999 (runs: 46.0, strike rate: 139.0).

Kane Williamson performs worst against Gujarat Lions with a score of 24.0 (runs: 6.0, strike rate: 66.0).
Kane Williamson performs best against Royal Challengers Bangalore with a score of 63.644444444444446 (runs: 40.77777777777778, strike rate: 117.0).



Nitish Rana performs worst against Rising Pune Supergiant with a score of 36.05 (runs: 18.5, strike rate: 77.0).
Nitish Rana performs best against Kolkata Knight Riders with a score of 86.6 (runs: 50.0, strike rate: 172.0).

Rinku Singh performs worst against Mumbai Indians with a score of 42.725 (runs: 12.5, strike rate: 113.25).
Rinku Singh performs best against Lucknow Super Giants with a score of 79.26666666666667 (runs: 37.666666666666664, strike rate: 176.33333333333334).

Rahmanullah Gurbaz performs worst against Sunrisers Hyderabad with a score of

0.0 (runs: 0.0, strike rate: 0.0).
Rahmanullah Gurbaz performs best against Gujarat Titans with a score of 83.39999999999999 (runs: 48.0, strike rate: 166.0).

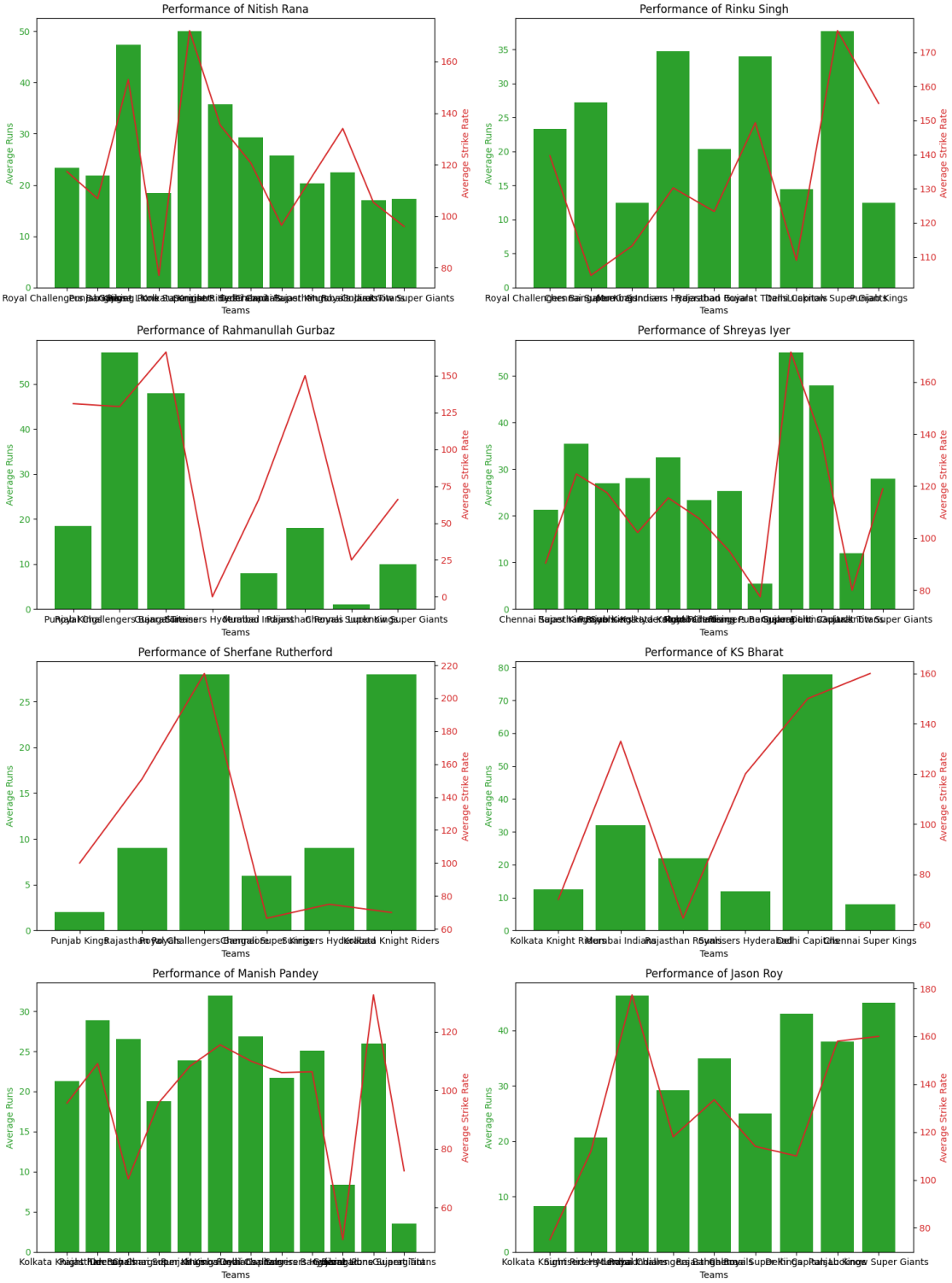
Shreyas Iyer performs worst against Rising Pune Supergiant with a score of 27.1 (runs: 5.5, strike rate: 77.5).
Shreyas Iyer performs best against Gujarat Lions with a score of 89.94999999999999 (runs: 55.0, strike rate: 171.5).

Sherfane Rutherford performs worst against Chennai Super Kings with a score of 24.15 (runs: 6.0, strike rate: 66.5).
Sherfane Rutherford performs best against Royal Challengers Bangalore with a score of 84.1 (runs: 28.0, strike rate: 215.0).

KS Bharat performs worst against Kolkata Knight Riders with a score of 29.75 (runs: 12.5, strike rate: 70.0).
KS Bharat performs best against Delhi Capitals with a score of 99.6 (runs: 78.0, strike rate: 150.0).

Manish Pandey performs worst against Gujarat Lions with a score of 20.53333333333333 (runs: 8.333333333333334, strike rate: 49.0).
Manish Pandey performs best against Rising Pune Supergiant with a score of 57.95 (runs: 26.0, strike rate: 132.5).

Jason Roy performs worst against Kolkata Knight Riders with a score of 28.333333333333332 (runs: 8.333333333333334, strike rate: 75.0).
Jason Roy performs best against Mumbai Indians with a score of 85.63333333333333 (runs: 46.333333333333336, strike rate: 177.33333333333334).



KL Rahul performs worst against Gujarat Titans with a score of 33.83333333333333 (runs: 25.333333333333332, strike rate: 53.666666666666664).
KL Rahul performs best against Mumbai Indians with a score of 75.82499999999999

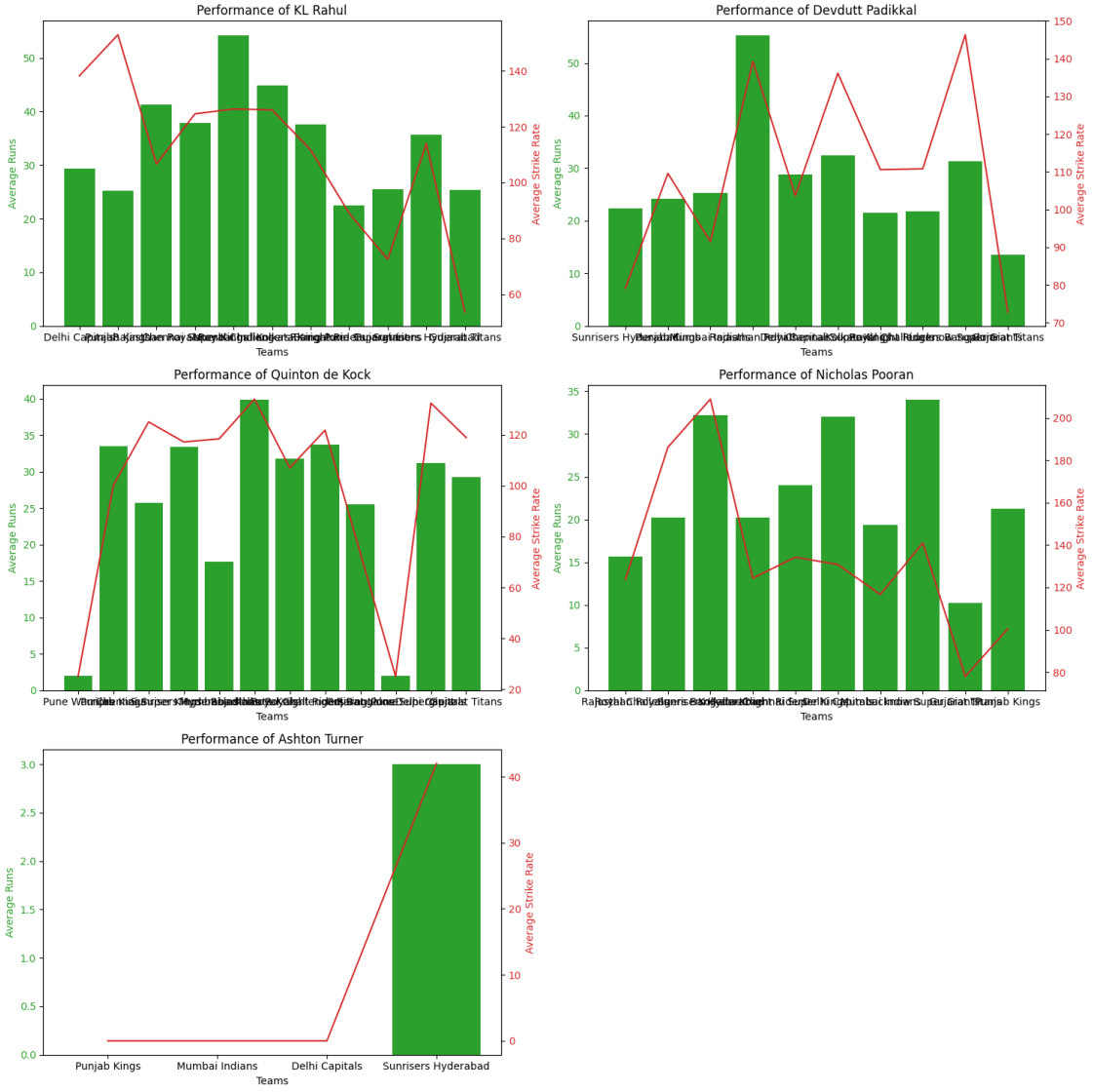
(runs: 54.1875, strike rate: 126.3125).

Devdutt Padikkal performs worst against Gujarat Titans with a score of 31.36 (runs: 13.6, strike rate: 72.8).
Devdutt Padikkal performs best against Rajasthan Royals with a score of 80.44999999999999 (runs: 55.25, strike rate: 139.25).

Quinton de Kock performs worst against Pune Warriors with a score of 8.9 (runs: 2.0, strike rate: 25.0).
Quinton de Kock performs best against Rajasthan Royals with a score of 68.15555555555555 (runs: 39.888888888888886, strike rate: 134.11111111111111).

Nicholas Pooran performs worst against Gujarat Titans with a score of 30.575 (runs: 10.25, strike rate: 78.0).
Nicholas Pooran performs best against Sunrisers Hyderabad with a score of 85.16666666666666 (runs: 32.166666666666664, strike rate: 208.83333333333334).

Ashton Turner performs worst against Punjab Kings with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Ashton Turner performs best against Sunrisers Hyderabad with a score of 14.7 (runs: 3.0, strike rate: 42.0).



Rohit Sharma performs worst against Pune Warriors with a score of 42.24999999999999 (runs: 22.5, strike rate: 88.33333333333333).
Rohit Sharma performs best against Deccan Chargers with a score of 65.35 (runs: 43.75, strike rate: 115.75).

Dewald Brevis performs worst against Chennai Super Kings with a score of 19.9 (runs: 4.0, strike rate: 57.0).
Dewald Brevis performs best against Punjab Kings with a score of 93.1 (runs: 49.0, strike rate: 196.0).

Suryakumar Yadav performs worst against Pune Warriors with a score of 0.0 (runs:

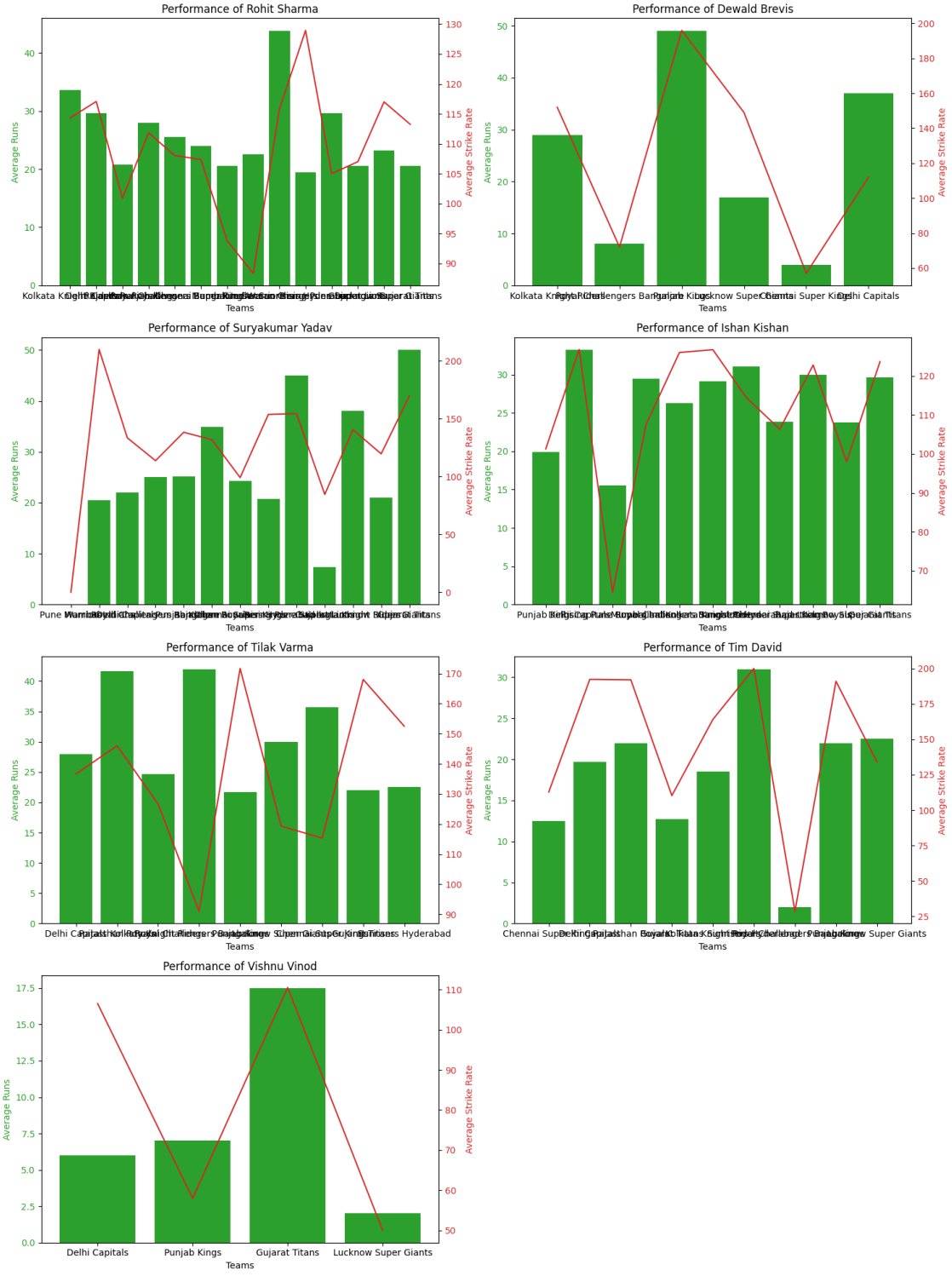
0.0, strike rate: 0.0).
Suryakumar Yadav performs best against Gujarat Titans with a score of 85.925 (runs: 50.0, strike rate: 169.75).

Ishan Kishan performs worst against Rising Pune Supergiant with a score of 30.199999999999996 (runs: 15.5, strike rate: 64.5).
Ishan Kishan performs best against Delhi Capitals with a score of 61.285714285714285 (runs: 33.214285714285715, strike rate: 126.78571428571429).

Tilak Varma performs worst against Kolkata Knight Riders with a score of 55.266666666666666 (runs: 24.666666666666668, strike rate: 126.66666666666667).
Tilak Varma performs best against Rajasthan Royals with a score of 72.96666666666667 (runs: 41.666666666666664, strike rate: 146.0).

Tim David performs worst against Royal Challengers Bangalore with a score of 9.95 (runs: 2.0, strike rate: 28.5).
Tim David performs best against Sunrisers Hyderabad with a score of 81.7 (runs: 31.0, strike rate: 200.0).

Vishnu Vinod performs worst against Lucknow Super Giants with a score of 16.4 (runs: 2.0, strike rate: 50.0).
Vishnu Vinod performs best against Gujarat Titans with a score of 45.4 (runs: 17.5, strike rate: 110.5).



Shikhar Dhawan performs worst against Lucknow Super Giants with a score of 14.549999999999999 (runs: 3.0, strike rate: 41.5).
Shikhar Dhawan performs best against Rising Pune Supergiant with a score of

60.75 (runs: 34.5, strike rate: 122.0).

Jitesh Sharma performs worst against Lucknow Super Giants with a score of 39.53333333333333 (runs: 9.333333333333334, strike rate: 110.0).
Jitesh Sharma performs best against Mumbai Indians with a score of 98.06666666666666 (runs: 34.666666666666664, strike rate: 246.0).

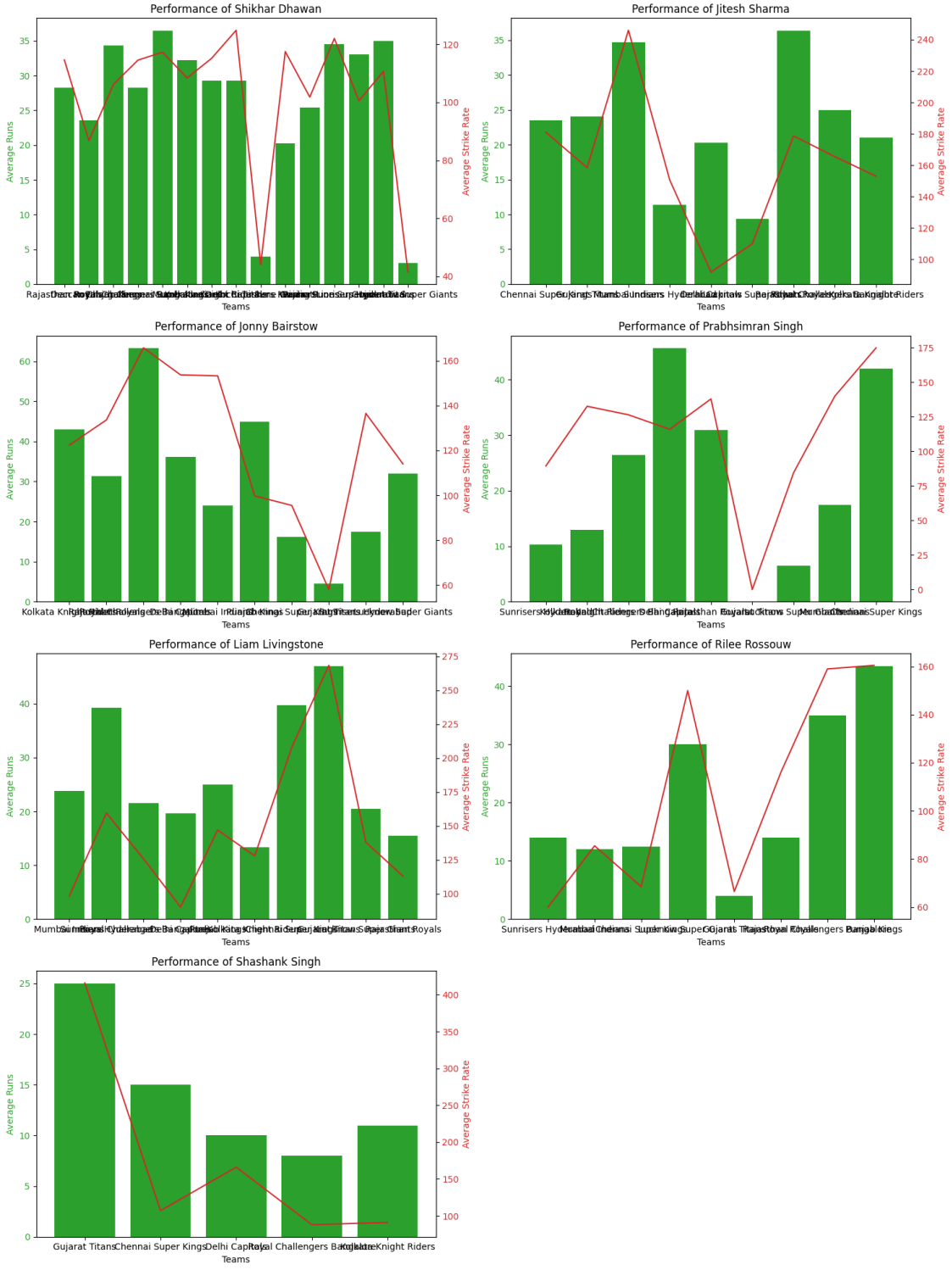
Jonny Bairstow performs worst against Gujarat Titans with a score of 20.549999999999997 (runs: 4.5, strike rate: 58.0).
Jonny Bairstow performs best against Royal Challengers Bangalore with a score of 94.0 (runs: 63.25, strike rate: 165.75).

Prabhsimran Singh performs worst against Gujarat Titans with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Prabhsimran Singh performs best against Chennai Super Kings with a score of 81.9 (runs: 42.0, strike rate: 175.0).

Liam Livingstone performs worst against Delhi Capitals with a score of 40.81666666666667 (runs: 19.666666666666668, strike rate: 90.16666666666667).
Liam Livingstone performs best against Gujarat Titans with a score of 113.44999999999999 (runs: 47.0, strike rate: 268.5).

Rilee Rossouw performs worst against Gujarat Titans with a score of 22.75 (runs: 4.0, strike rate: 66.5).
Rilee Rossouw performs best against Punjab Kings with a score of 78.6 (runs: 43.5, strike rate: 160.5).

Shashank Singh performs worst against Royal Challengers Bangalore with a score of 32.0 (runs: 8.0, strike rate: 88.0).
Shashank Singh performs best against Gujarat Titans with a score of 142.3 (runs: 25.0, strike rate: 416.0).



Sanju Samson performs worst against Chennai Super Kings with a score of 35.02142857142857 (runs: 15.285714285714286, strike rate: 81.07142857142857). Sanju Samson performs best against Gujarat Titans with a score of 67.32 (runs:

32.4, strike rate: 148.8).

Jos Buttler performs worst against Lucknow Super Giants with a score of 37.63333333333333 (runs: 18.333333333333332, strike rate: 82.66666666666667).
Jos Buttler performs best against Mumbai Indians with a score of 84.2125 (runs: 60.625, strike rate: 139.25).

Shimron Hetmeyer performs worst against Chennai Super Kings with a score of 44.71666666666667 (runs: 18.166666666666668, strike rate: 106.66666666666667).
Shimron Hetmeyer performs best against Punjab Kings with a score of 75.83333333333333 (runs: 24.333333333333332, strike rate: 196.0).

Yashasvi Jaiswal performs worst against Gujarat Titans with a score of 30.625 (runs: 10.0, strike rate: 78.75).
Yashasvi Jaiswal performs best against Chennai Super Kings with a score of 74.83999999999999 (runs: 40.4, strike rate: 155.2).

Dhruv Jurel performs worst against Lucknow Super Giants with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Dhruv Jurel performs best against Delhi Capitals with a score of 85.39999999999999 (runs: 8.0, strike rate: 266.0).

Riyan Parag performs worst against Delhi Capitals with a score of 33.04285714285714 (runs: 10.714285714285714, strike rate: 85.14285714285714).
Riyan Parag performs best against Royal Challengers Bangalore with a score of 54.55 (runs: 26.5, strike rate: 120.0).

Rovman Powell performs worst against Royal Challengers Bangalore with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Rovman Powell performs best against Sunrisers Hyderabad with a score of 104.19999999999999 (runs: 67.0, strike rate: 191.0).

Faf Duplesis performs worst against Gujarat Titans with a score of 43.0 (runs: 24.0, strike rate: 87.33333333333333).
Faf Duplesis performs best against Gujarat Lions with a score of 96.3 (runs:

69.0, strike rate: 160.0).

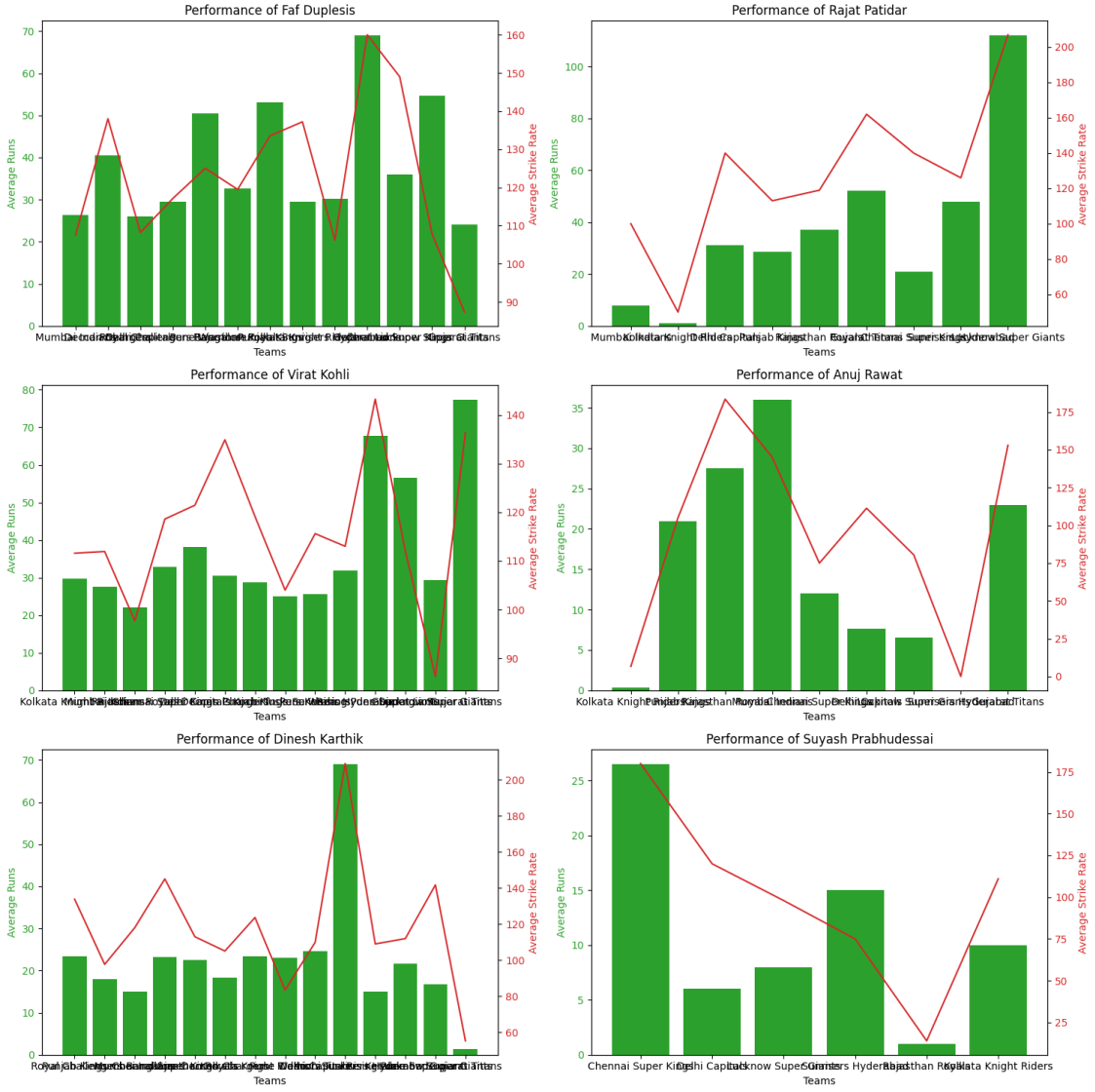
Rajat Patidar performs worst against Kolkata Knight Riders with a score of 15.7 (runs: 1.0, strike rate: 50.0).
Rajat Patidar performs best against Lucknow Super Giants with a score of 140.5 (runs: 112.0, strike rate: 207.0).

Virat Kohli performs worst against Rajasthan Royals with a score of 44.753571428571426 (runs: 22.071428571428573, strike rate: 97.67857142857143).
Virat Kohli performs best against Gujarat Titans with a score of 95.03333333333333 (runs: 77.33333333333333, strike rate: 136.33333333333334).

Anuj Rawat performs worst against Sunrisers Hyderabad with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Anuj Rawat performs best against Rajasthan Royals with a score of 74.3 (runs: 27.5, strike rate: 183.5).

Dinesh Karthik performs worst against Gujarat Titans with a score of 17.533333333333335 (runs: 1.3333333333333333, strike rate: 55.333333333333336).
Dinesh Karthik performs best against Kochi Tuskers Kerala with a score of 111.0 (runs: 69.0, strike rate: 209.0).

Suyash Prabhudessai performs worst against Rajasthan Royals with a score of 4.9 (runs: 1.0, strike rate: 14.0).
Suyash Prabhudessai performs best against Chennai Super Kings with a score of 72.55 (runs: 26.5, strike rate: 180.0).



Abdul Samad performs worst against Royal Challengers Bangalore with a score of 5.3 (runs: 0.5, strike rate: 16.5).
Abdul Samad performs best against Delhi Capitals with a score of 65.9 (runs: 25.25, strike rate: 160.75).

Anmolpreet Singh performs worst against Delhi Capitals with a score of 32.0 (runs: 8.0, strike rate: 88.0).
Anmolpreet Singh performs best against Lucknow Super Giants with a score of 61.25 (runs: 33.5, strike rate: 126.0).

Heinrich Klaasen performs worst against Chennai Super Kings with a score of 39.3

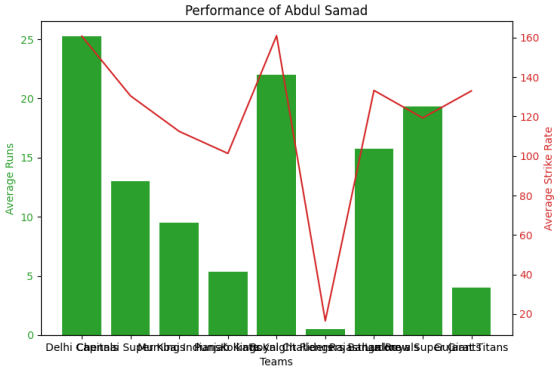
(runs: 12.0, strike rate: 103.0).
Heinrich Klaasen performs best against Royal Challengers Bangalore with a score of 100.85 (runs: 68.0, strike rate: 177.5).

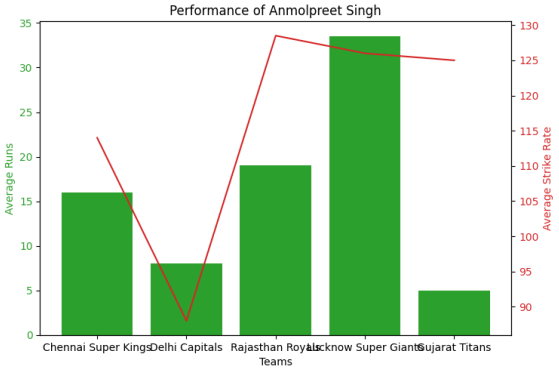
Rahul Tripathi performs worst against Gujarat Titans with a score of 44.33333333333333 (runs: 11.333333333333334, strike rate: 121.33333333333333).
Rahul Tripathi performs best against Kolkata Knight Riders with a score of 69.56 (runs: 30.8, strike rate: 160.0).

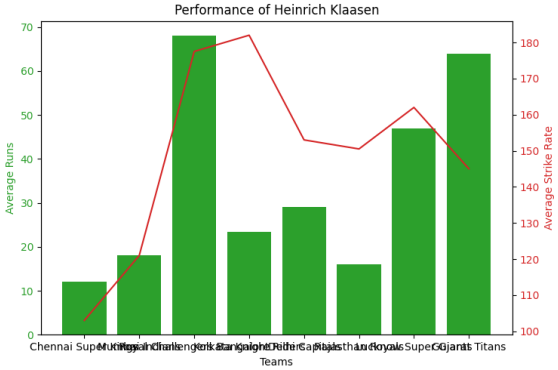
Mayank Agarwal performs worst against Gujarat Titans with a score of 20.0 (runs: 5.0, strike rate: 55.0).
Mayank Agarwal performs best against Kochi Tuskers Kerala with a score of 64.2 (runs: 33.0, strike rate: 137.0).

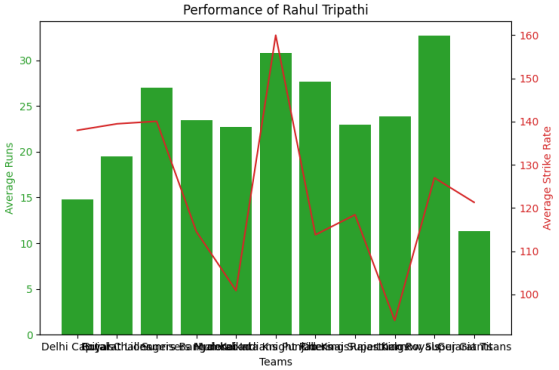
Travis Head performs worst against Rising Pune Supergiant with a score of 18.7 (runs: 4.0, strike rate: 53.0).
Travis Head performs best against Kolkata Knight Riders with a score of 100.19999999999999 (runs: 75.0, strike rate: 159.0).

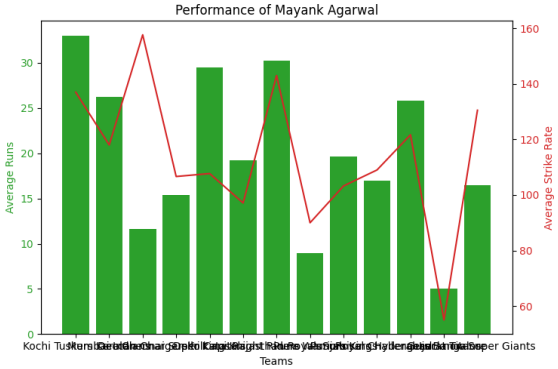
Aiden Markram performs worst against Lucknow Super Giants with a score of 31.833333333333336 (runs: 13.333333333333334, strike rate: 75.0).
Aiden Markram performs best against Kolkata Knight Riders with a score of 72.58 (runs: 41.8, strike rate: 144.4).

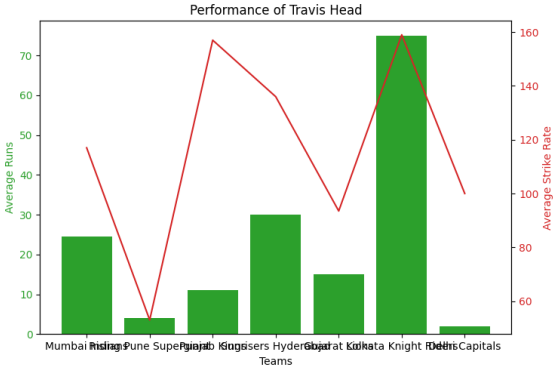

Performance of Abdul Samad
Average Runs
Average Strike Rate
Teams

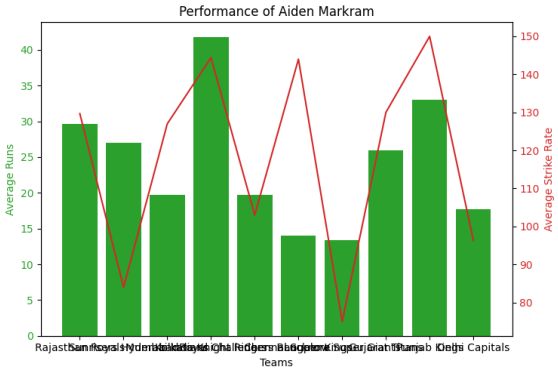

Performance of Anmolpreet Singh
Average Runs
Average Strike Rate
Chennai Super Kings
Delhi Capitals
Rajasthan Royals
Lucknow Super Giants
Gujarat Titans
Teams


Performance of Heinrich Klaasen
Average Runs
Average Strike Rate
Teams


Performance of Rahul Tripathi
Average Runs
Average Strike Rate
Teams


Performance of Mayank Agarwal
Average Runs
Average Strike Rate
Teams


Performance of Travis Head
Average Runs
Average Strike Rate
Mumbai Indians
Pune Supergiant
Punjab Kingsers
Hyderabad
Gujarat
Kolkata Knight Riders
Delhi Capitals
Teams


Performance of Aiden Markram
Average Runs
Average Strike Rate
Teams

6 3. Ground Impact:

Does the batting performance vary significantly based on the ground? Identify grounds where players tend to perform exceptionally well or struggle.

This code analyzes the batting performance of cricket players based on their runs and strike rates at different cricket grounds. It processes Excel files for batsmen from various teams, calculating the average runs and strike rates for each player at different grounds. The code then identifies the ground where each batsman performs the worst and the best, considering both metrics. Additionally, it visualizes the average runs and strike rates for each batsman at different grounds using a bar chart and a line plot. The core concept involves evaluating and comparing batsmen's performances at various grounds, providing insights into their strengths and weaknesses in different playing environments.

[98]:
```
worst={}
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    dfs = {}
    worst={}
    for sheet in xls.sheet_names:
        dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)

    for i in dfs:
        a={}
        for index, row in dfs[i].iterrows():
            if row['Ground']!='-':
                if row['Runs'] != '-':
                    if row['S/R'] != '-':
                        row["Runs"]=str(row["Runs"]).replace("*","")
                        if row['Ground'] not in a:
                            a[row['Ground']]=[int(row['Runs']),int(row['S/
 ↪R']),1]
                        else:
                            a[row['Ground']]=[␣
 ↪a[row['Ground']][0]+int(row['Runs']), a[row['Ground']][1]+int(row['S/R']),␣
 ↪a[row['Ground']][2]+1]
        worst[i]=a

    for i in worst:
        for j in worst[i]:
            worst[i][j]=[worst[i][j][0]/worst[i][j][2],worst[i][j][1]/
 ↪worst[i][j][2]]

    weight_runs = 0.7
    weight_sr = 0.3
```

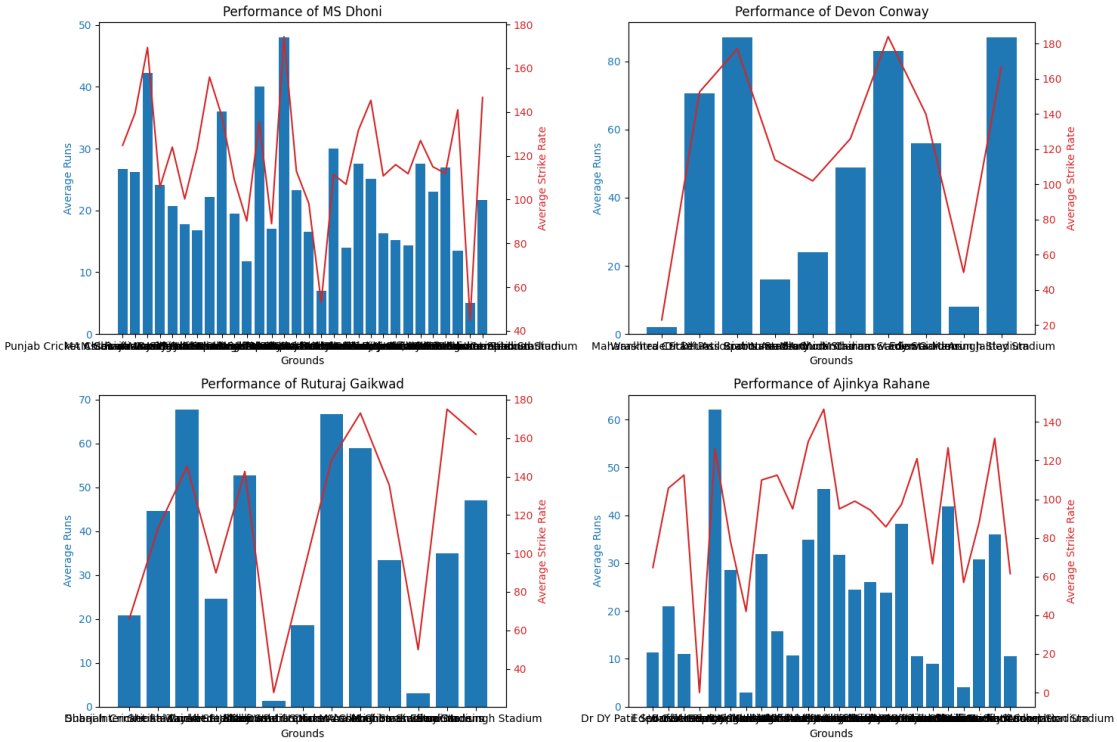
```
    for batsman, data in worst.items():
        scores = {ground: (weight_runs * stats[0] + weight_sr * stats[1],␣
↪stats) for ground, stats in data.items()}
        worst_ground = min(scores.items(), key=lambda x: x[1][0])
        best_ground = max(scores.items(), key=lambda x: x[1][0])
        print(f"{batsman} performs worst at {worst_ground[0]} with a score of␣
↪{worst_ground[1][0]} (runs: {worst_ground[1][1][0]}, strike rate:␣
↪{worst_ground[1][1][1]}).")  # replace 'against' with 'at'
        print(f"{batsman} performs best at {best_ground[0]} with a score of␣
↪{best_ground[1][0]} (runs: {best_ground[1][1][0]}, strike rate:␣
↪{best_ground[1][1][1]}).")  # replace 'against' with 'at'
        print("\n")

    num_players = len(worst)
    num_cols = 2
    num_rows = math.ceil(num_players / num_cols)

    fig, axs = plt.subplots(num_rows, num_cols, figsize=(15, num_rows*5))

    for idx, (player, data) in enumerate(worst.items()):
        ax1 = axs[idx // num_cols, idx % num_cols]

        grounds = list(data.keys())
        avg_runs = [ground_data[0] for ground_data in data.values()]
        avg_strike_rate = [ground_data[1] for ground_data in data.values()]

        color = 'tab:blue'
        ax1.set_xlabel('Grounds')
        ax1.set_ylabel('Average Runs', color=color)
        ax1.bar(grounds, avg_runs, color=color)
        ax1.tick_params(axis='y', labelcolor=color)

        ax2 = ax1.twinx()
        color = 'tab:red'
        ax2.set_ylabel('Average Strike Rate', color=color)
        ax2.plot(grounds, avg_strike_rate, color=color)
        ax2.tick_params(axis='y', labelcolor=color)

        ax1.set_title(f'Performance of {player}')

    if num_players % num_cols != 0:
        for idx in range(num_players, num_rows * num_cols):
            fig.delaxes(axs.flatten()[idx])

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

MS Dhoni performs worst at Holkar Cricket Stadium with a score of 17.0 (runs: 5.0, strike rate: 45.0).
MS Dhoni performs best at SuperSport Park with a score of 85.94999999999999 (runs: 48.0, strike rate: 174.5).

Devon Conway performs worst at Wankhede Stadium with a score of 8.299999999999999 (runs: 2.0, strike rate: 23.0).
Devon Conway performs best at Dr DY Patil Sports Academy with a score of 114.0 (runs: 87.0, strike rate: 177.0).

Ruturaj Gaikwad performs worst at Brabourne Stadium with a score of 9.233333333333334 (runs: 1.3333333333333333, strike rate: 27.666666666666668).
Ruturaj Gaikwad performs best at Narendra Modi Stadium with a score of 93.19999999999999 (runs: 59.0, strike rate: 173.0).

Ajinkya Rahane performs worst at Wanderers Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Ajinkya Rahane performs best at St George's Park with a score of 81.19999999999999 (runs: 62.0, strike rate: 126.0).



Rishabh Pant performs worst at Barabati Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Rishabh Pant performs best at Saurashtra Cricket Association Stadium with a score of 99.9 (runs: 69.0, strike rate: 172.0).

David Warner performs worst at Nehru Stadium (Kochi) with a score of 14.7 (runs: 3.0, strike rate: 42.0).
David Warner performs best at JSCA International Stadium Complex with a score of 123.0 (runs: 90.0, strike rate: 200.0).

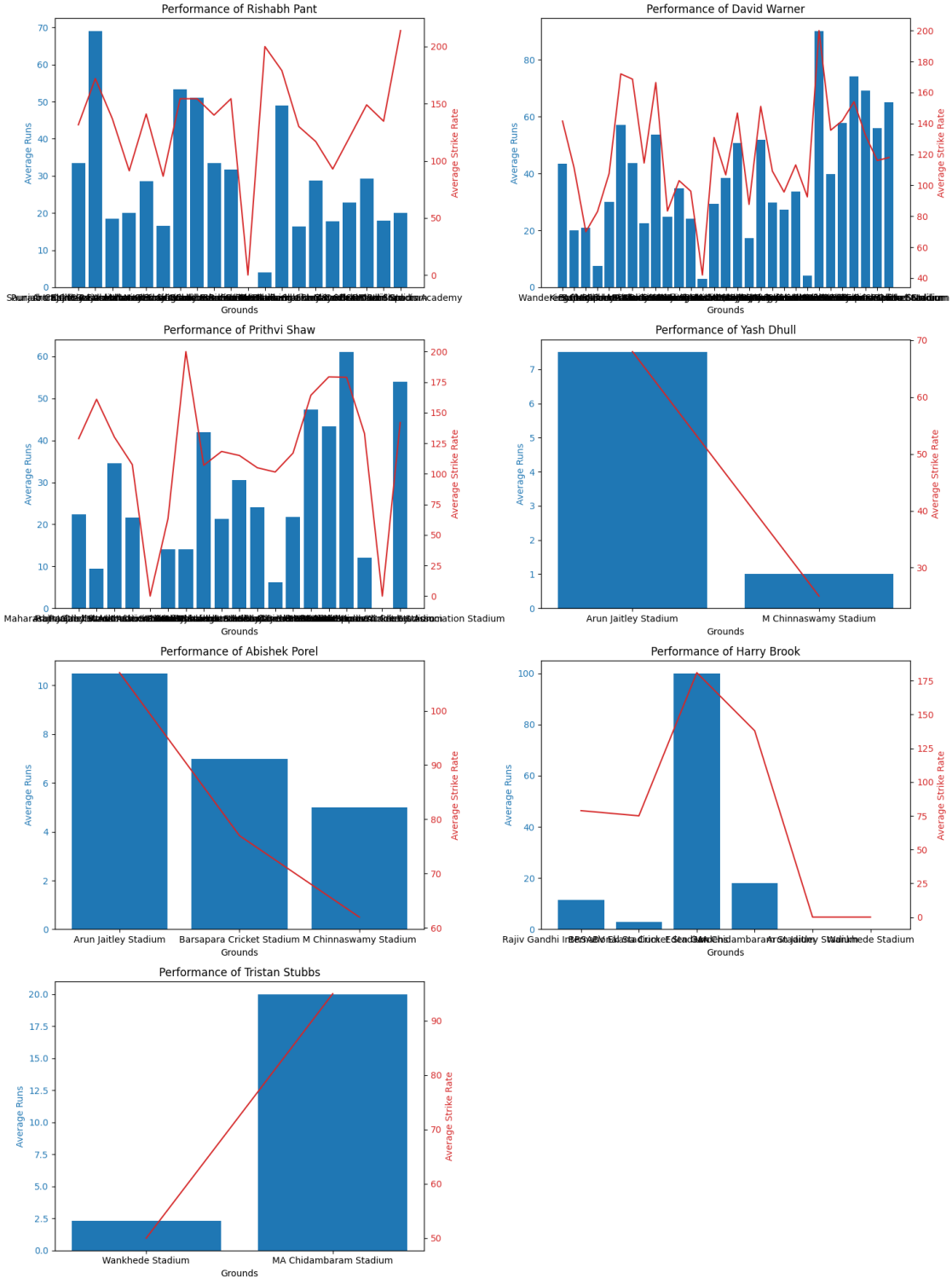
Prithvi Shaw performs worst at Punjab Cricket Association IS Bindra Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Prithvi Shaw performs best at Dr DY Patil Sports Academy with a score of 96.39999999999999 (runs: 61.0, strike rate: 179.0).

Yash Dhull performs worst at M Chinnaswamy Stadium with a score of 8.2 (runs: 1.0, strike rate: 25.0).
Yash Dhull performs best at Arun Jaitley Stadium with a score of 25.65 (runs: 7.5, strike rate: 68.0).

Abishek Porel performs worst at M Chinnaswamy Stadium with a score of 22.099999999999998 (runs: 5.0, strike rate: 62.0).
Abishek Porel performs best at Arun Jaitley Stadium with a score of 39.45 (runs: 10.5, strike rate: 107.0).

Harry Brook performs worst at Arun Jaitley Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Harry Brook performs best at Eden Gardens with a score of 124.3 (runs: 100.0, strike rate: 181.0).

Tristan Stubbs performs worst at Wankhede Stadium with a score of 16.633333333333333 (runs: 2.3333333333333335, strike rate: 50.0).
Tristan Stubbs performs best at MA Chidambaram Stadium with a score of 42.5 (runs: 20.0, strike rate: 95.0).



David Miller performs worst at BRSABV Ekana Cricket Stadium with a score of 19.2 (runs: 6.0, strike rate: 50.0).
David Miller performs best at Eden Gardens with a score of 64.82499999999999

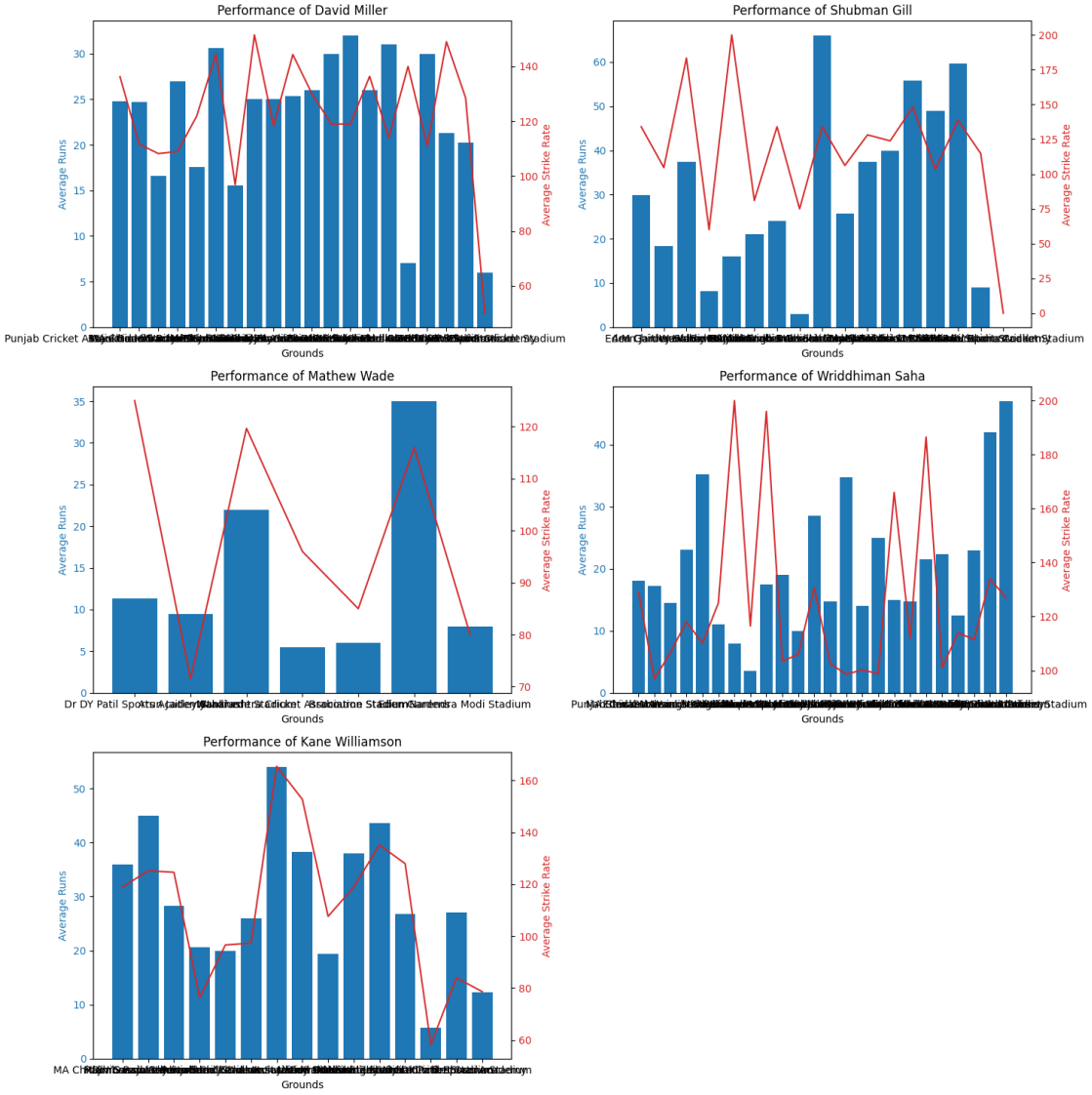
(runs: 30.625, strike rate: 144.625).

Shubman Gill performs worst at BRSABV Ekana Cricket Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Shubman Gill performs best at Punjab Cricket Association IS Bindra Stadium with a score of 86.39999999999999 (runs: 66.0, strike rate: 134.0).

Mathew Wade performs worst at Arun Jaitley Stadium with a score of 28.099999999999998 (runs: 9.5, strike rate: 71.5).
Mathew Wade performs best at Eden Gardens with a score of 59.3 (runs: 35.0, strike rate: 116.0).

Wriddhiman Saha performs worst at Wanderers Stadium with a score of 37.4 (runs: 3.5, strike rate: 116.5).
Wriddhiman Saha performs best at SuperSport Park with a score of 71.05 (runs: 17.5, strike rate: 196.0).

Kane Williamson performs worst at Sharjah Cricket Stadium with a score of 21.499999999999996 (runs: 5.75, strike rate: 58.25).
Kane Williamson performs best at Punjab Cricket Association IS Bindra Stadium with a score of 87.44999999999999 (runs: 54.0, strike rate: 165.5).



Nitish Rana performs worst at Arun Jaitley Stadium with a score of 25.133333333333333 (runs: 4.333333333333333, strike rate: 73.66666666666667). Nitish Rana performs best at Green Park with a score of 107.19999999999999 (runs: 70.0, strike rate: 194.0).

Rinku Singh performs worst at Maharashtra Cricket Association Stadium with a score of 25.299999999999997 (runs: 5.5, strike rate: 71.5).
Rinku Singh performs best at Narendra Modi Stadium with a score of 101.99999999999999 (runs: 48.0, strike rate: 228.0).

Rahmanullah Gurbaz performs worst at Rajiv Gandhi International Stadium with a

score of 0.0 (runs: 0.0, strike rate: 0.0).
Rahmanullah Gurbaz performs best at Punjab Cricket Association IS Bindra Stadium with a score of 56.5 (runs: 22.0, strike rate: 137.0).

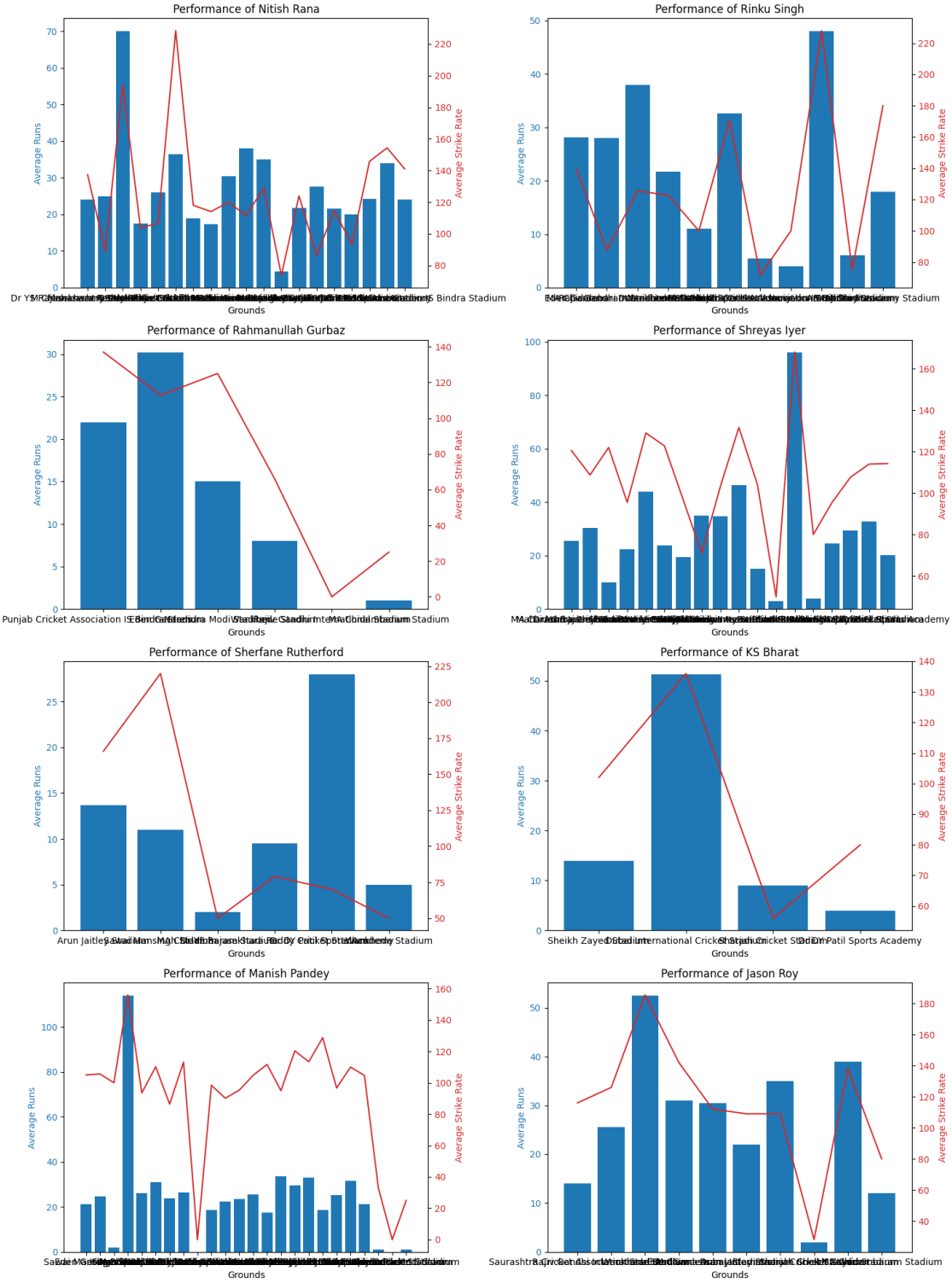
Shreyas Iyer performs worst at Barabati Stadium with a score of 17.1 (runs: 3.0, strike rate: 50.0).
Shreyas Iyer performs best at Green Park with a score of 117.6 (runs: 96.0, strike rate: 168.0).

Sherfane Rutherford performs worst at MA Chidambaram Stadium with a score of 16.4 (runs: 2.0, strike rate: 50.0).
Sherfane Rutherford performs best at Sawai Mansingh Stadium with a score of 73.7 (runs: 11.0, strike rate: 220.0).

KS Bharat performs worst at Sharjah Cricket Stadium with a score of 23.1 (runs: 9.0, strike rate: 56.0).
KS Bharat performs best at Dubai International Cricket Stadium with a score of 76.73333333333332 (runs: 51.333333333333336, strike rate: 136.0).

Manish Pandey performs worst at Vidarbha Cricket Association Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Manish Pandey performs best at SuperSport Park with a score of 126.6 (runs: 114.0, strike rate: 156.0).

Jason Roy performs worst at Sharjah Cricket Stadium with a score of 9.8 (runs: 2.0, strike rate: 28.0).
Jason Roy performs best at Wankhede Stadium with a score of 92.4 (runs: 52.5, strike rate: 185.5).



KL Rahul performs worst at Dr YS Rajasekhara Reddy Cricket Stadium with a score of 21.466666666666665 (runs: 8.666666666666666, strike rate: 51.333333333333336).

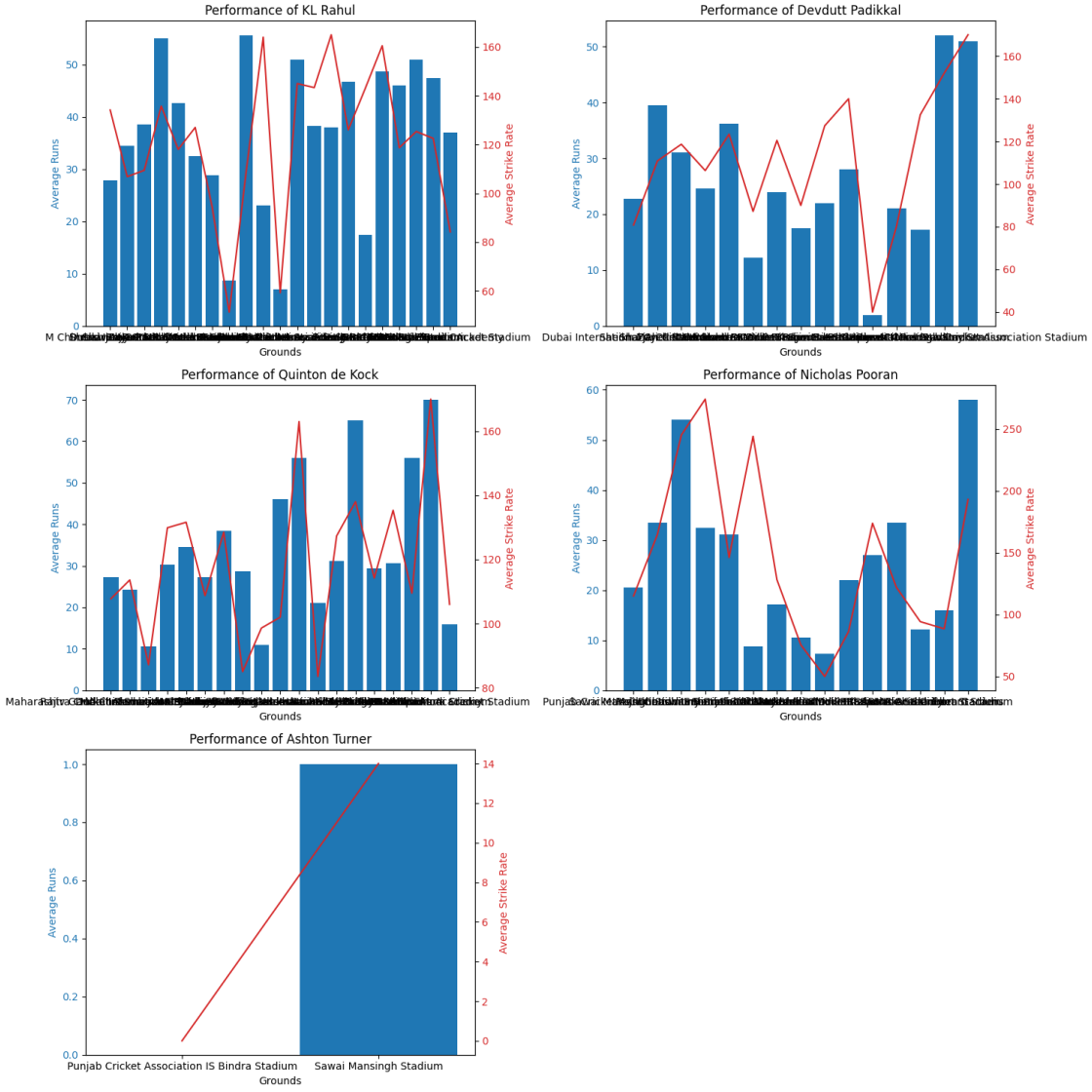
KL Rahul performs best at Holkar Cricket Stadium with a score of 82.275 (runs: 48.75, strike rate: 160.5).

Devdutt Padikkal performs worst at Rajiv Gandhi International Stadium with a score of 13.4 (runs: 2.0, strike rate: 40.0).
Devdutt Padikkal performs best at Himachal Pradesh Cricket Association Stadium with a score of 86.69999999999999 (runs: 51.0, strike rate: 170.0).

Quinton de Kock performs worst at MA Chidambaram Stadium with a score of 33.542857142857144 (runs: 10.571428571428571, strike rate: 87.14285714285714).
Quinton de Kock performs best at Narendra Modi Stadium with a score of 100.0 (runs: 70.0, strike rate: 170.0).

Nicholas Pooran performs worst at Narendra Modi Stadium with a score of 20.133333333333333 (runs: 7.333333333333333, strike rate: 50.0).
Nicholas Pooran performs best at M Chinnaswamy Stadium with a score of 111.3 (runs: 54.0, strike rate: 245.0).

Ashton Turner performs worst at Punjab Cricket Association IS Bindra Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Ashton Turner performs best at Sawai Mansingh Stadium with a score of 4.9 (runs: 1.0, strike rate: 14.0).



Rohit Sharma performs worst at Saurashtra Cricket Association Stadium with a score of 14.9 (runs: 5.0, strike rate: 38.0).
Rohit Sharma performs best at Himachal Pradesh Cricket Association Stadium with a score of 74.25 (runs: 46.5, strike rate: 139.0).

Dewald Brevis performs worst at Dr DY Patil Sports Academy with a score of 19.9 (runs: 4.0, strike rate: 57.0).
Dewald Brevis performs best at Brabourne Stadium with a score of 93.1 (runs: 31.0, strike rate: 238.0).

Suryakumar Yadav performs worst at JSCA International Stadium Complex with a

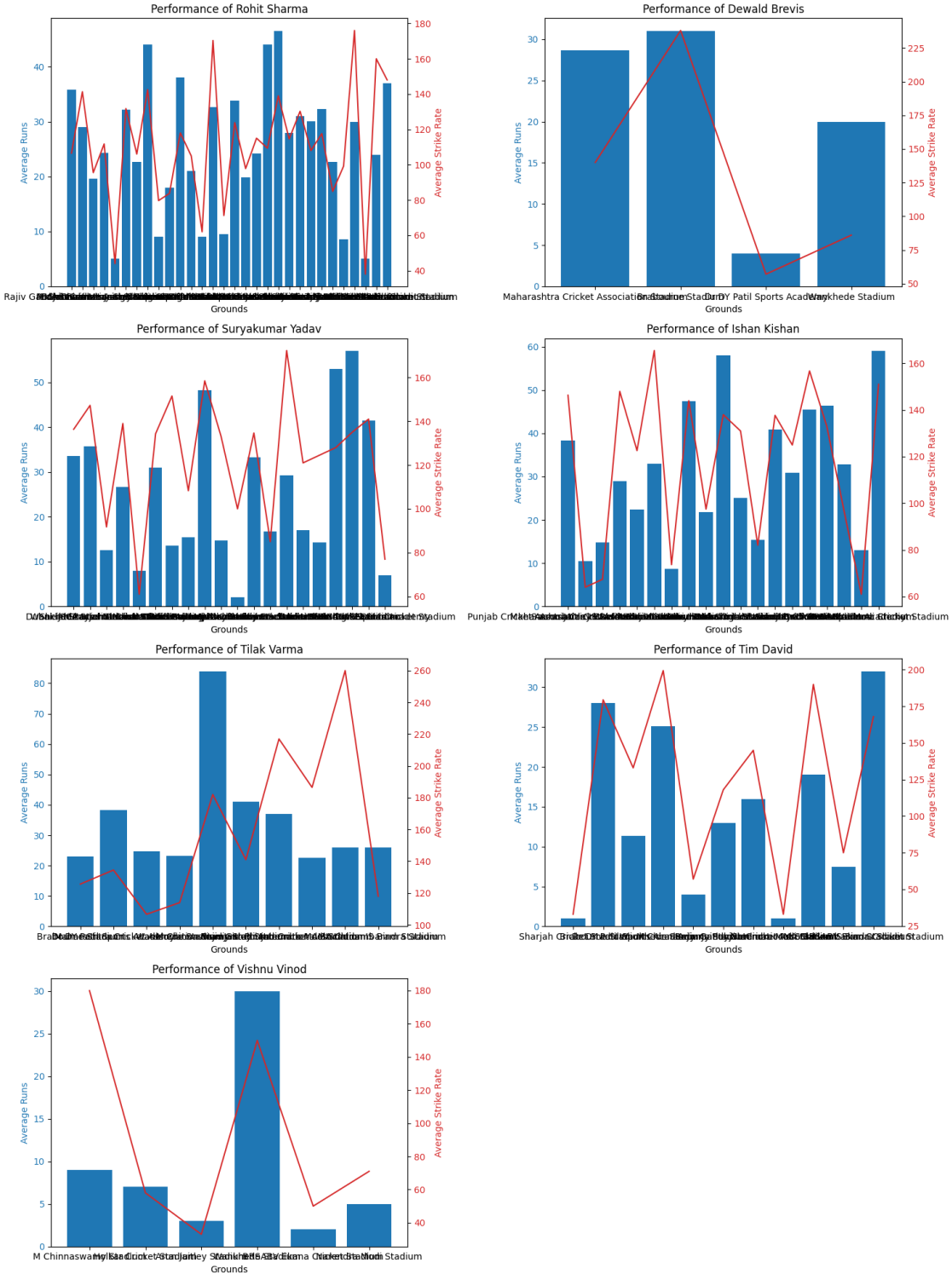
score of 23.9 (runs: 8.0, strike rate: 61.0).
Suryakumar Yadav performs best at Maharashtra Cricket Association Stadium with a score of 81.38333333333333 (runs: 48.333333333333336, strike rate: 158.5).

Ishan Kishan performs worst at Arun Jaitley Stadium with a score of 26.549999999999997 (runs: 10.5, strike rate: 64.0).
Ishan Kishan performs best at BRSABV Ekana Cricket Stadium with a score of 86.6 (runs: 59.0, strike rate: 151.0).

Tilak Varma performs worst at Maharashtra Cricket Association Stadium with a score of 49.266666666666666 (runs: 24.666666666666668, strike rate: 106.66666666666667).
Tilak Varma performs best at M Chinnaswamy Stadium with a score of 113.4 (runs: 84.0, strike rate: 182.0).

Tim David performs worst at Sharjah Cricket Stadium with a score of 10.6 (runs: 1.0, strike rate: 33.0).
Tim David performs best at Wankhede Stadium with a score of 77.41111111111111 (runs: 25.11111111111111, strike rate: 199.44444444444446).

Vishnu Vinod performs worst at Arun Jaitley Stadium with a score of 12.0 (runs: 3.0, strike rate: 33.0).
Vishnu Vinod performs best at Wankhede Stadium with a score of 66.0 (runs: 30.0, strike rate: 150.0).



Shikhar Dhawan performs worst at Kingsmead with a score of 14.7 (runs: 3.0, strike rate: 42.0).
Shikhar Dhawan performs best at Barsapara Cricket Stadium with a score of 106.1

(runs: 86.0, strike rate: 153.0).

Jitesh Sharma performs worst at Rajiv Gandhi International Stadium with a score of 16.0 (runs: 4.0, strike rate: 44.0).
Jitesh Sharma performs best at Wankhede Stadium with a score of 103.03333333333333 (runs: 27.333333333333332, strike rate: 279.6666666666667).

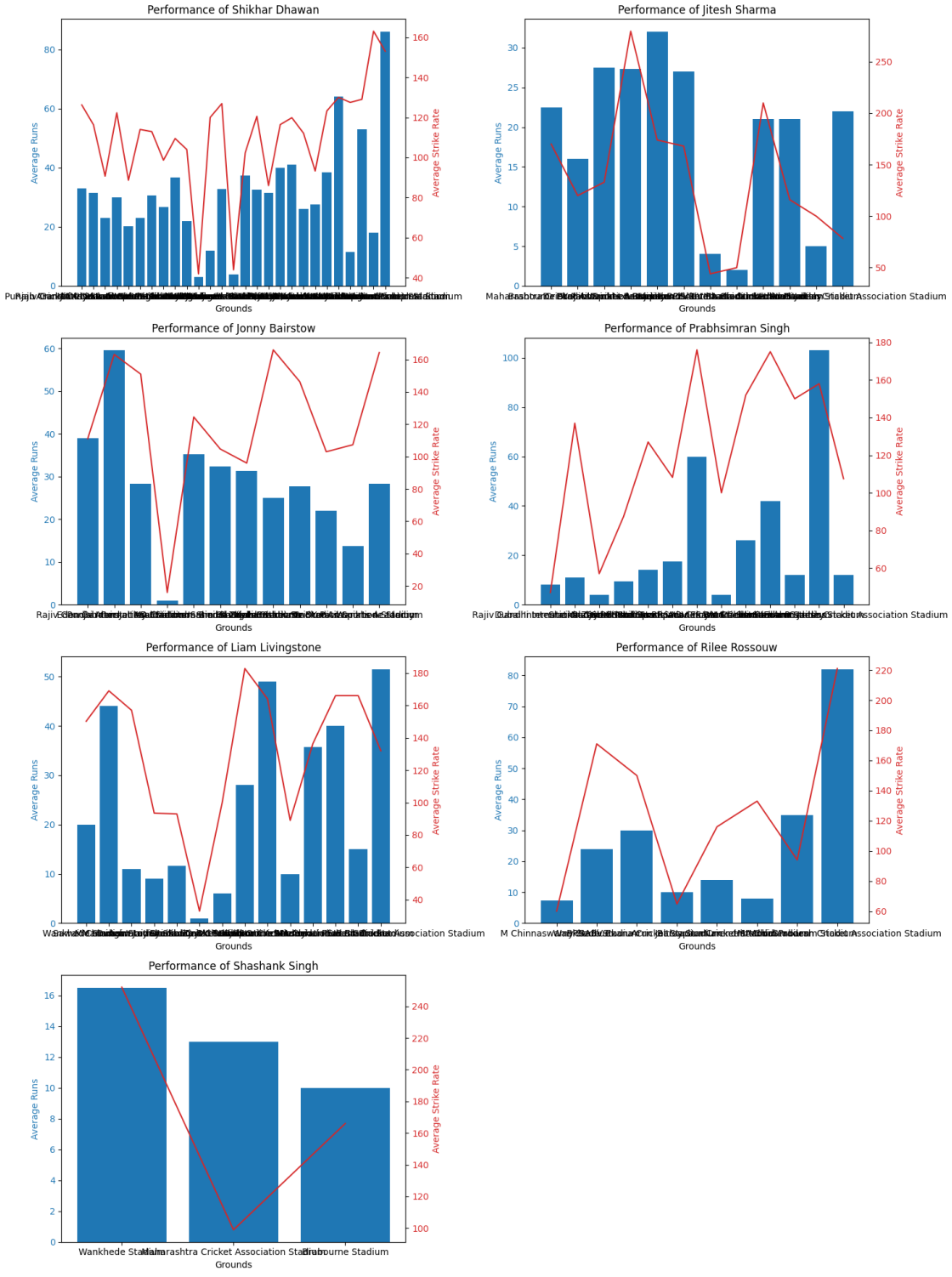
Jonny Bairstow performs worst at Punjab Cricket Association IS Bindra Stadium with a score of 5.5 (runs: 1.0, strike rate: 16.0).
Jonny Bairstow performs best at Rajiv Gandhi International Stadium with a score of 90.6 (runs: 59.5, strike rate: 163.16666666666666).

Prabhsimran Singh performs worst at Rajiv Gandhi International Stadium with a score of 19.7 (runs: 8.0, strike rate: 47.0).
Prabhsimran Singh performs best at Arun Jaitley Stadium with a score of 119.5 (runs: 103.0, strike rate: 158.0).

Liam Livingstone performs worst at Sheikh Zayed Stadium with a score of 10.6 (runs: 1.0, strike rate: 33.0).
Liam Livingstone performs best at Brabourne Stadium with a score of 83.5 (runs: 49.0, strike rate: 164.0).

Rilee Rossouw performs worst at M Chinnaswamy Stadium with a score of 23.075 (runs: 7.25, strike rate: 60.0).
Rilee Rossouw performs best at Himachal Pradesh Cricket Association Stadium with a score of 123.69999999999999 (runs: 82.0, strike rate: 221.0).

Shashank Singh performs worst at Maharashtra Cricket Association Stadium with a score of 38.8 (runs: 13.0, strike rate: 99.0).
Shashank Singh performs best at Wankhede Stadium with a score of 87.14999999999999 (runs: 16.5, strike rate: 252.0).



Sanju Samson performs worst at Barabati Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Sanju Samson performs best at Rajiv Gandhi International Stadium with a score of

73.48333333333333 (runs: 47.833333333333336, strike rate: 133.33333333333334).

Jos Buttler performs worst at Himachal Pradesh Cricket Association Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Jos Buttler performs best at Holkar Cricket Stadium with a score of 95.5 (runs: 64.0, strike rate: 169.0).

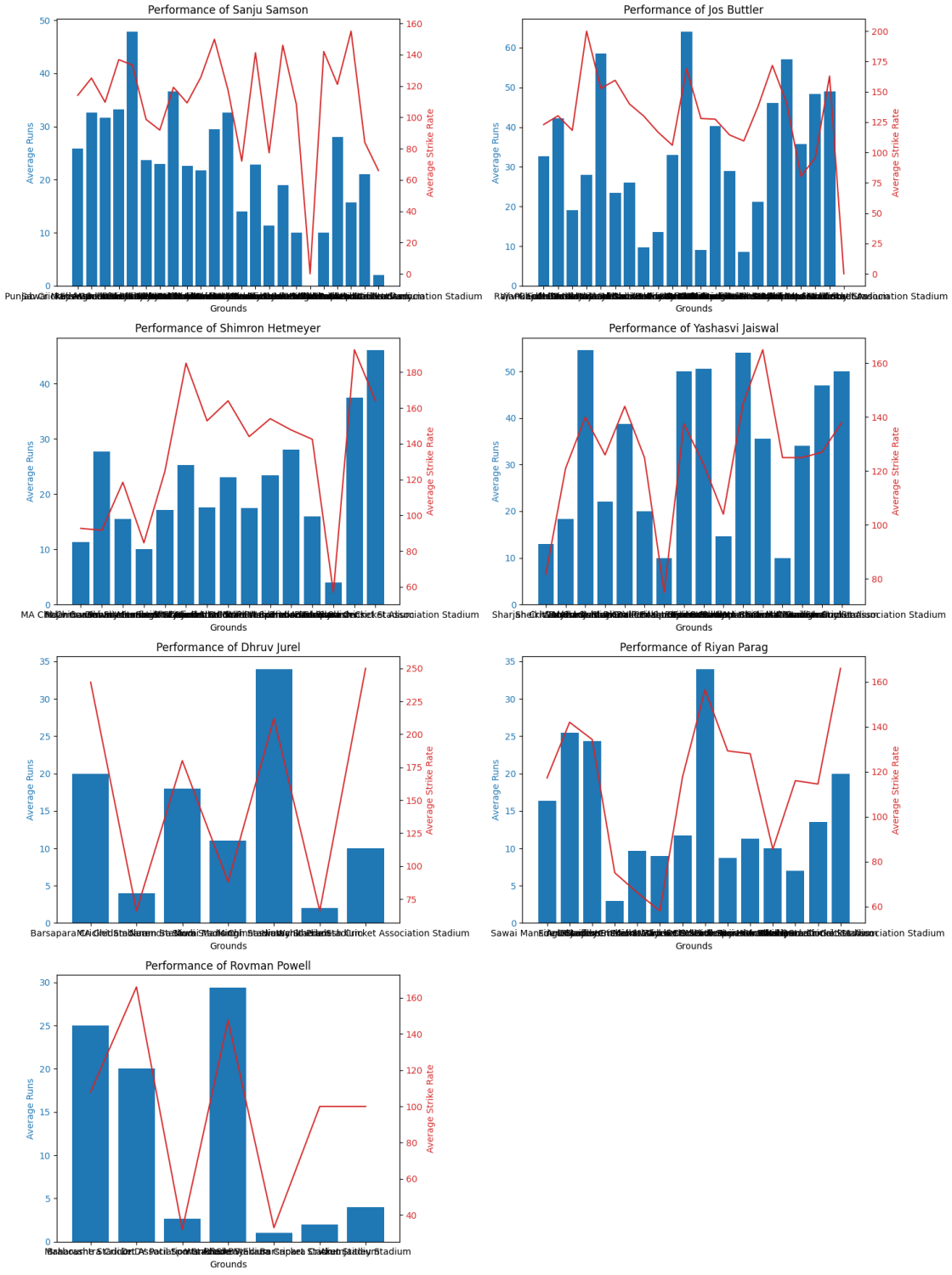
Shimron Hetmeyer performs worst at Eden Gardens with a score of 19.9 (runs: 4.0, strike rate: 57.0).
Shimron Hetmeyer performs best at Barsapara Cricket Stadium with a score of 84.0 (runs: 37.5, strike rate: 192.5).

Yashasvi Jaiswal performs worst at Dr DY Patil Sports Academy with a score of 29.5 (runs: 10.0, strike rate: 75.0).
Yashasvi Jaiswal performs best at Rajiv Gandhi International Stadium with a score of 81.3 (runs: 54.0, strike rate: 145.0).

Dhruv Jurel performs worst at Wankhede Stadium with a score of 21.2 (runs: 2.0, strike rate: 66.0).
Dhruv Jurel performs best at M Chinnaswamy Stadium with a score of 87.39999999999999 (runs: 34.0, strike rate: 212.0).

Riyan Parag performs worst at Sheikh Zayed Stadium with a score of 23.7 (runs: 9.0, strike rate: 58.0).
Riyan Parag performs best at Maharashtra Cricket Association Stadium with a score of 70.75 (runs: 34.0, strike rate: 156.5).

Rovman Powell performs worst at BRSABV Ekana Cricket Stadium with a score of 10.6 (runs: 1.0, strike rate: 33.0).
Rovman Powell performs best at Wankhede Stadium with a score of 64.85999999999999 (runs: 29.4, strike rate: 147.6).



Faf Duplesis performs worst at Narendra Modi Stadium with a score of 27.85 (runs: 13.0, strike rate: 62.5).
Faf Duplesis performs best at Barabati Stadium with a score of 98.8 (runs: 52.0,

strike rate: 208.0).

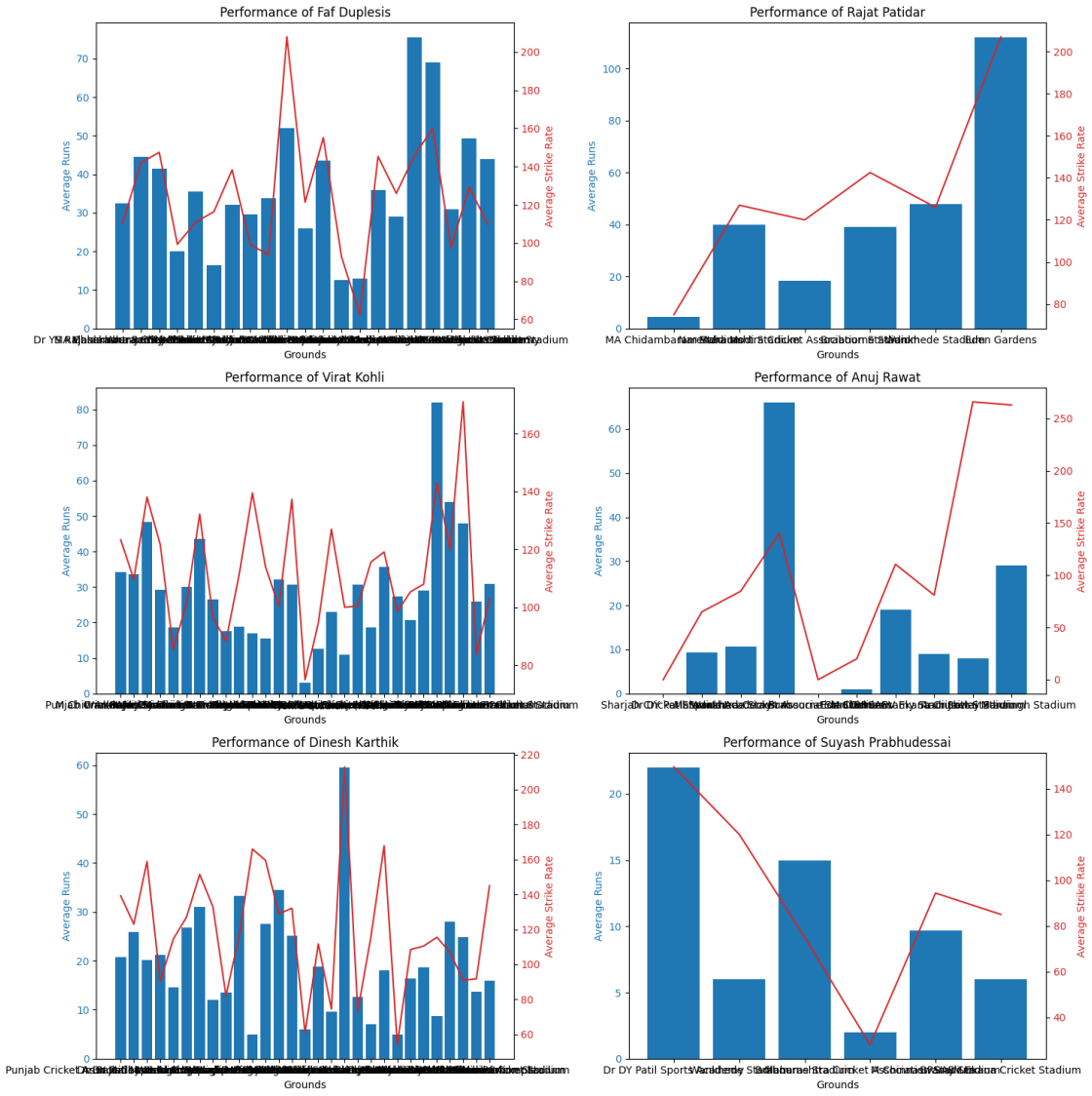
Rajat Patidar performs worst at MA Chidambaram Stadium with a score of 25.65 (runs: 4.5, strike rate: 75.0).
Rajat Patidar performs best at Eden Gardens with a score of 140.5 (runs: 112.0, strike rate: 207.0).

Virat Kohli performs worst at Vidarbha Cricket Association Stadium with a score of 24.6 (runs: 3.0, strike rate: 75.0).
Virat Kohli performs best at Saurashtra Cricket Association Stadium with a score of 100.3 (runs: 82.0, strike rate: 143.0).

Anuj Rawat performs worst at Sharjah Cricket Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Anuj Rawat performs best at Sawai Mansingh Stadium with a score of 99.19999999999999 (runs: 29.0, strike rate: 263.0).

Dinesh Karthik performs worst at Sharjah Cricket Stadium with a score of 19.7 (runs: 5.0, strike rate: 54.0).
Dinesh Karthik performs best at Holkar Cricket Stadium with a score of 105.55 (runs: 59.5, strike rate: 213.0).

Suyash Prabhudessai performs worst at Maharashtra Cricket Association Stadium with a score of 9.8 (runs: 2.0, strike rate: 28.0).
Suyash Prabhudessai performs best at Dr DY Patil Sports Academy with a score of 60.25 (runs: 22.0, strike rate: 149.5).



Abdul Samad performs worst at Dr DY Patil Sports Academy with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Abdul Samad performs best at Sawai Mansingh Stadium with a score of 84.5 (runs: 17.0, strike rate: 242.0).

Anmolpreet Singh performs worst at Brabourne Stadium with a score of 32.0 (runs: 8.0, strike rate: 88.0).
Anmolpreet Singh performs best at Rajiv Gandhi International Stadium with a score of 65.1 (runs: 36.0, strike rate: 133.0).

Heinrich Klaasen performs worst at M Chinnaswamy Stadium with a score of 29.7

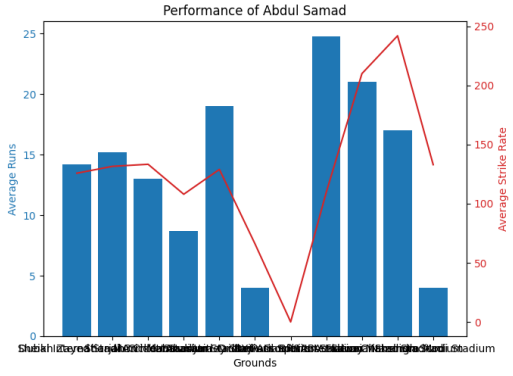
(runs: 6.0, strike rate: 85.0).
Heinrich Klaasen performs best at Rajiv Gandhi International Stadium with a score of 91.53999999999999 (runs: 50.8, strike rate: 186.6).

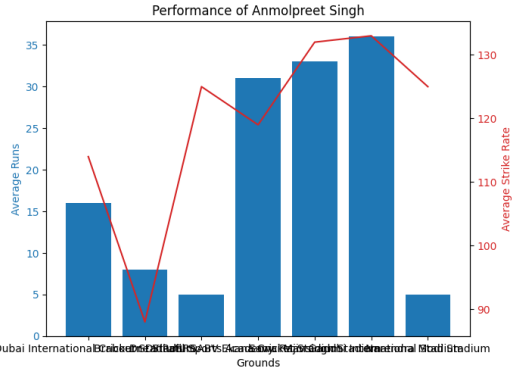
Rahul Tripathi performs worst at Holkar Cricket Stadium with a score of 32.9 (runs: 11.0, strike rate: 84.0).
Rahul Tripathi performs best at Saurashtra Cricket Association Stadium with a score of 81.3 (runs: 33.0, strike rate: 194.0).

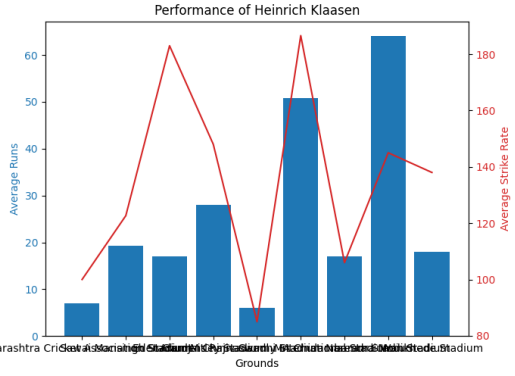
Mayank Agarwal performs worst at Holkar Cricket Stadium with a score of 17.46 (runs: 3.0, strike rate: 51.2).
Mayank Agarwal performs best at Narendra Modi Stadium with a score of 77.39999999999999 (runs: 49.0, strike rate: 143.66666666666666).

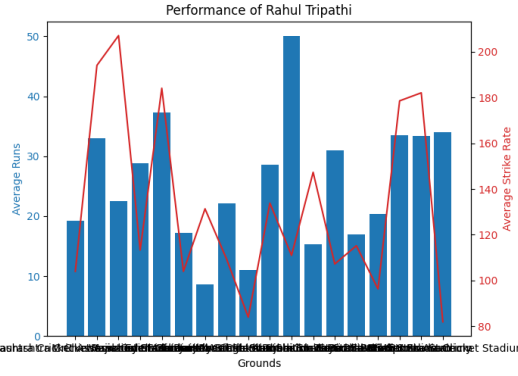
Travis Head performs worst at Maharashtra Cricket Association Stadium with a score of 13.4 (runs: 2.0, strike rate: 40.0).
Travis Head performs best at Saurashtra Cricket Association Stadium with a score of 77.1 (runs: 30.0, strike rate: 187.0).

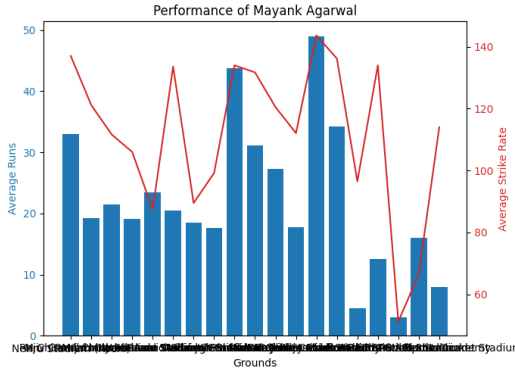
Aiden Markram performs worst at BRSABV Ekana Cricket Stadium with a score of 0.0 (runs: 0.0, strike rate: 0.0).
Aiden Markram performs best at Eden Gardens with a score of 92.6 (runs: 50.0, strike rate: 192.0).

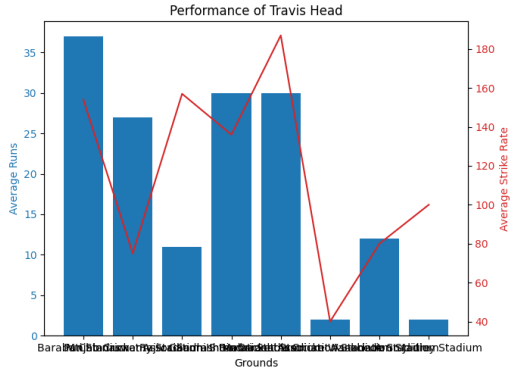

Performance of Abdul Samad
Average Runs
Average Strike Rate
Grounds

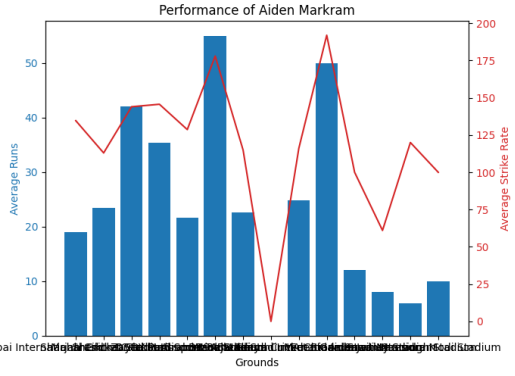

Performance of Anmolpreet Singh
Average Runs
Average Strike Rate
Grounds


Performance of Heinrich Klaasen
Average Runs
Average Strike Rate
Grounds


Performance of Rahul Tripathi
Average Runs
Average Strike Rate
Grounds


Performance of Mayank Agarwal
Average Runs
Average Strike Rate
Grounds


Performance of Travis Head
Average Runs
Average Strike Rate
Grounds


Performance of Aiden Markram
Average Runs
Average Strike Rate
Grounds

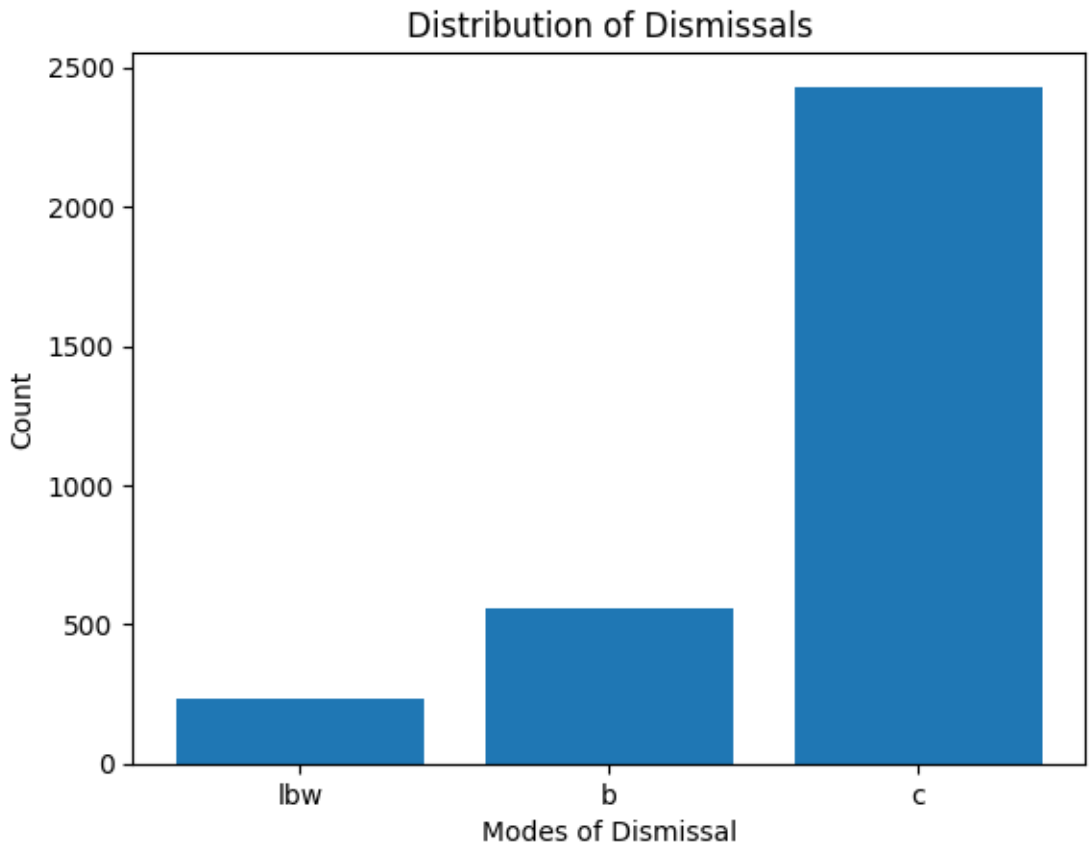
7 4. Dismissal Patterns:

What are the most common modes of dismissal among the players? Analyse the distribution of dismissals (e.g., caught, bowled, LBW).

The code analyzes and visualizes the distribution of batsmen dismissals across various cricket teams. It iterates through batsmen data in Excel files for each team, extracting information on how batsmen were dismissed. The occurrences of different dismissal modes, such as 'lbw,' 'b,' and 'c' (caught), are tallied in the `dismissals` dictionary. The resulting counts are then plotted using a bar chart to illustrate the distribution of dismissal modes, providing insights into the common ways batsmen are getting out.

[99]:
```
dismissals={'lbw':0,'b':0,'c':0}
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    dfs = {}
    for sheet in xls.sheet_names:
        dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)

    for i in dfs:
        a={}
        for index, row in dfs[i].iterrows():
            if row['How Dismissed'] .split()[0]  in ['lbw','b','c']:
                    dismissals[row['How Dismissed'] .split()[0]]+=1

print(dismissals)

plt.bar(dismissals.keys(), dismissals.values())
plt.xlabel('Modes of Dismissal')
plt.ylabel('Count')
plt.title('Distribution of Dismissals')
plt.show()
```

```
{'lbw': 232, 'b': 556, 'c': 2431}
```



8 5. Significance of each factor:

Display the significance of each factor (eg: ground, innings, versus etc …) that affect the player performance.

This code analyzes and visualizes cricket batsmen's performance based on their average scores in different contexts such as grounds, innings, and opponents ('versus'). It calculates a weighted average score considering both runs and strike rate for each context. The resulting averages are plotted using bar charts, and boxplots provide a summary of the data distribution. The code then calculates and compares the variance of average scores for each context, identifying which context has the highest variance.

Explanation:

1. The code computes weighted average scores for batsmen in different contexts (grounds, innings, versus) by considering both runs and strike rates.
2. Bar charts depict the average scores for each context, providing insights into batsmen's performance variations.
3. Boxplots visualize the distribution of average scores, offering a summary of the data spread.
4. The code calculates the variance of average scores for each context, and a comparison reveals which context has the highest variance.

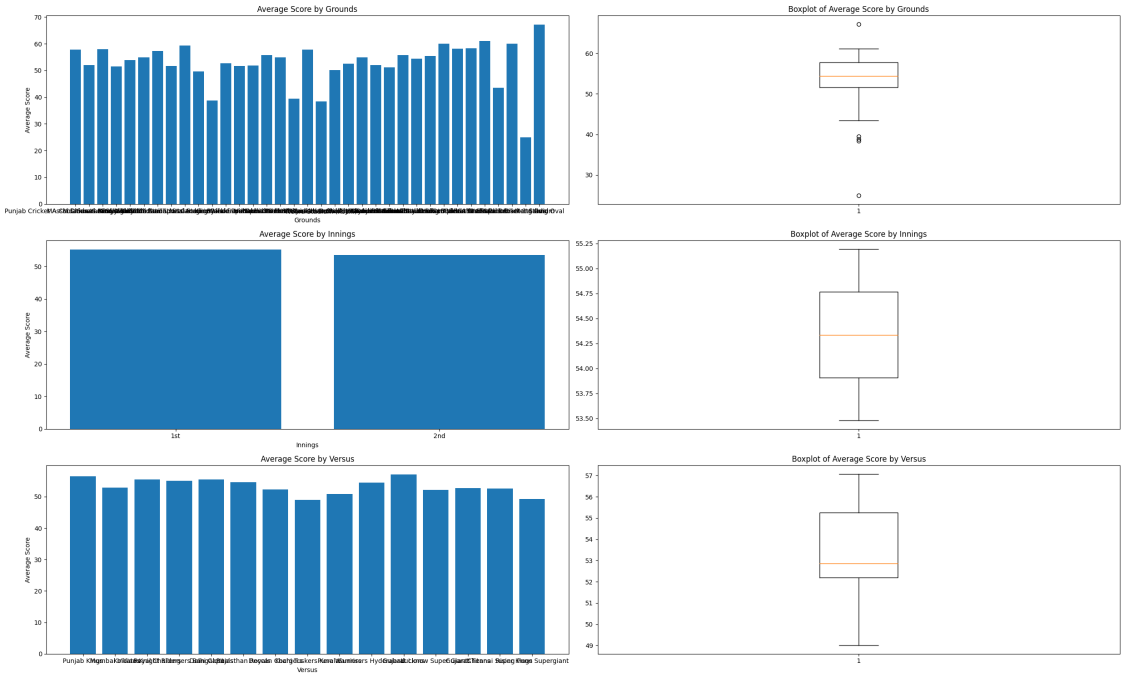
5. The result indicates whether grounds, innings, or versus context exhibits the most varied performances among batsmen.

a - Based on Ground

[61]:
```
grounds={}
innings={}
versus={}
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    dfs = {}
    worst={}
    for sheet in xls.sheet_names:
        dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)
    for i in dfs:
        a={}
        for index, row in dfs[i].iterrows():
            if row['Ground']!='-' and row['Runs'] != '-' and row['S/R'] != '-':
                row["Runs"]=str(row['Runs']).replace("*","")
                if row['Ground'] not in grounds:
                    grounds[row['Ground']]=[int(row['Runs'])*0.7+row['S/R']*0.
 ↪3,1]
                else:
                    grounds[row['Ground']]=[␣
 ↪grounds[row['Ground']][0]+int(row['Runs'])*0.7+row['S/R']*0.3,␣
 ↪grounds[row['Ground']][1]+1]
            if row['M/Inns']!='-' and row['Runs'] != '-' and row['S/R'] != '-':
                row["Runs"]=str(row["Runs"]).replace("*","")
                if row['M/Inns'] not in innings:
                    innings[row['M/Inns']]=[int(row['Runs'])*0.7+row['S/R']*0.
 ↪3,1]
                else:
                    innings[row['M/Inns']]=[ innings[row['M/
 ↪Inns']][0]+int(row['Runs'])*0.7+row['S/R']*0.3, innings[row['M/Inns']][1]+1]
            if row['Versus']!='-' and row['Runs'] != '-' and row['S/R'] != '-':
                row["Runs"]=str(row["Runs"]).replace("*","")
                if row['Versus'] not in versus:
                    versus[row['Versus']]=[int(row['Runs'])*0.7+row['S/R']*0.
 ↪3,1]
                else:
                    versus[row['Versus']]=[␣
 ↪versus[row['Versus']][0]+int(row['Runs'])*0.7+row['S/R']*0.3,␣
 ↪versus[row['Versus']][1]+1]
res=[];res1=[];res2=[]
for i in grounds:
    res.append(grounds[i][0]/grounds[i][1])
```

```
for i in innings:
    res1.append(innings[i][0]/innings[i][1])
for i in versus:
    res2.append(versus[i][0]/versus[i][1])

fig, axs = plt.subplots(3, 2, figsize=(25, 15))

axs[0, 0].bar(grounds.keys(), res)
axs[0, 0].set_xlabel('Grounds')
axs[0, 0].set_ylabel('Average Score')
axs[0, 0].set_title('Average Score by Grounds')

axs[1, 0].bar(innings.keys(), res1)
axs[1, 0].set_xlabel('Innings')
axs[1, 0].set_ylabel('Average Score')
axs[1, 0].set_title('Average Score by Innings')

axs[2, 0].bar(versus.keys(), res2)
axs[2, 0].set_xlabel('Versus')
axs[2, 0].set_ylabel('Average Score')
axs[2, 0].set_title('Average Score by Versus')

axs[0, 1].boxplot(res)
axs[0, 1].set_title('Boxplot of Average Score by Grounds')

axs[1, 1].boxplot(res1)
axs[1, 1].set_title('Boxplot of Average Score by Innings')

axs[2, 1].boxplot(res2)
axs[2, 1].set_title('Boxplot of Average Score by Versus')

plt.tight_layout()
plt.show()

var_res = np.var(res)
var_res1 = np.var(res1)
var_res2 = np.var(res2)

print(f"Variance of Grounds: {var_res}")
print(f"Variance of Innings: {var_res1}")
print(f"Variance of Versus: {var_res2}")

highest_variance = max(var_res, var_res1, var_res2)
if highest_variance == var_res:
    print("Grounds has the highest variance.")
elif highest_variance == var_res1:
    print("Innings has the highest variance.")
```

```
else:
    print("Versus has the highest variance.")
```



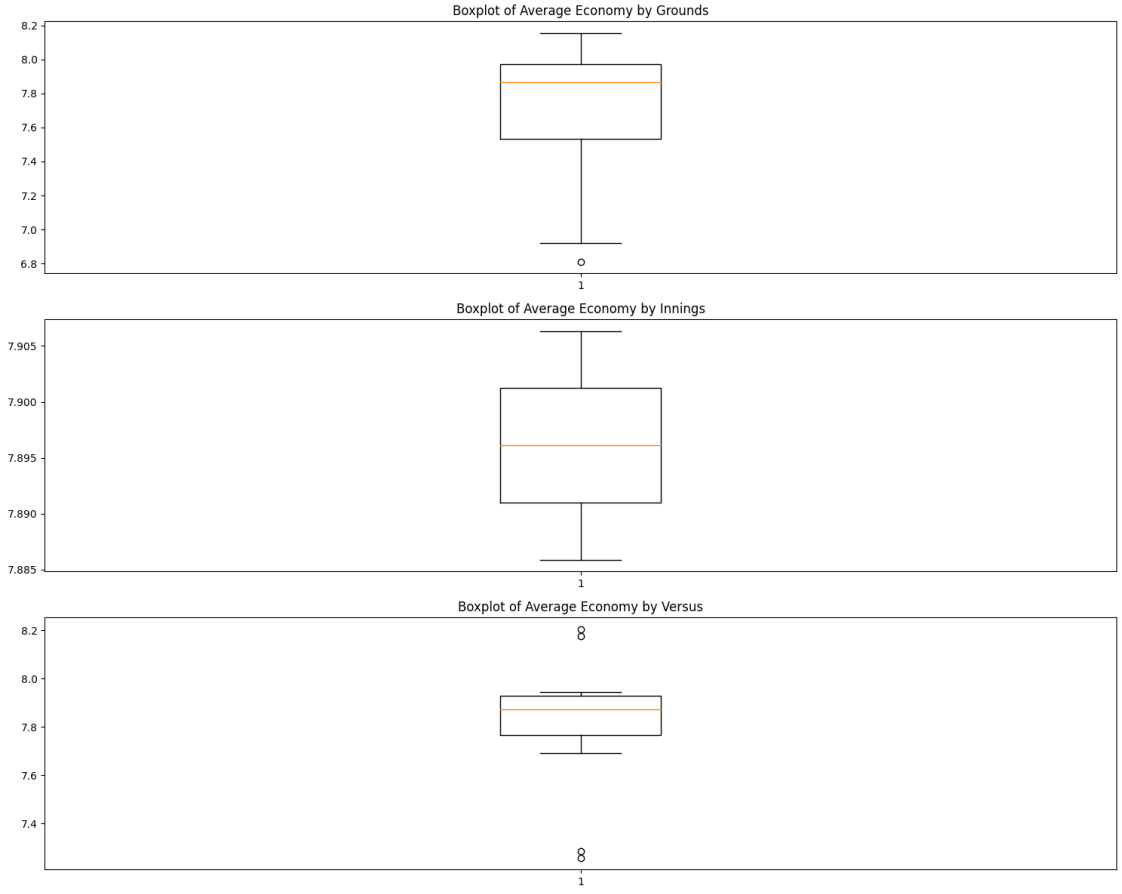

```
Variance of Grounds: 59.60939768335885
Variance of Innings: 0.7330932469244521
Variance of Versus: 5.609517190083104
Grounds has the highest variance.
```

b - Ground Innings and Verses

[62]:
```
grounds={}
innings={}
versus={}
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BOWLERS/
 ↪{i}_BOWLERS.xlsx')
    dfs = {}
    worst={}
    for sheet in xls.sheet_names:
        dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)
    for j in dfs:
        a={}
        for index, row in dfs[j].iterrows():
            try:
                if row['E/R']!='-':
```

```
                    if row['Ground'] not in grounds:
                        grounds[row['Ground']]=[float(row['E/R']),1]
                    else:
                        grounds[row['Ground']]=[␣
 ↪grounds[row['Ground']][0]+float(row['E/R']), grounds[row['Ground']][1]+1]
                if row['M/Inns']!='-':
                    if row['M/Inns'] not in innings:
                        innings[row['M/Inns']]=[float(row['E/R']),1]
                    else:
                        innings[row['M/Inns']]=[ innings[row['M/
 ↪Inns']][0]+float(row['E/R']), innings[row['M/Inns']][1]+1]
                if row['Versus']!='-':
                    if row['Versus'] not in versus:
                        versus[row['Versus']]=[float(row['E/R']),1]
                    else:
                        versus[row['Versus']]=[␣
 ↪versus[row['Versus']][0]+float(row['E/R']), versus[row['Versus']][1]+1]
            except Exception as e:
                pass
res=[];res1=[];res2=[]
for i in grounds:
    res.append(grounds[i][0]/grounds[i][1])
for i in innings:
    res1.append(innings[i][0]/innings[i][1])
for i in versus:
    res2.append(versus[i][0]/versus[i][1])

res = [value for value in res if not np.isnan(value)]
res1 = [value for value in res1 if not np.isnan(value)]
res2 = [value for value in res2 if not np.isnan(value)]

fig, axs = plt.subplots(3, figsize=(15, 12))


axs[0].boxplot(res)
axs[0].set_title('Boxplot of Average Economy by Grounds')

axs[1].boxplot(res1)
axs[1].set_title('Boxplot of Average Economy by Innings')

axs[2].boxplot(res2)
axs[2].set_title('Boxplot of Average Economy by Versus')

plt.tight_layout()
plt.show()

var_res = np.var(res)
```

```
var_res1 = np.var(res1)
var_res2 = np.var(res2)

print(f"Variance of Grounds: {var_res}")
print(f"Variance of Innings: {var_res1}")
print(f"Variance of Versus: {var_res2}")

highest_variance = max(var_res, var_res1, var_res2)
if highest_variance == var_res:
    print("Grounds has the highest variance.")
elif highest_variance == var_res1:
    print("Innings has the highest variance.")
else:
    print("Versus has the highest variance.")
```




```
Variance of Grounds: 0.1338330357068332
Variance of Innings: 0.00010495906307553092
Variance of Versus: 0.0643479585758681
Grounds has the highest variance.
```

9 6. Hypothetically swap

Hypothetically swap two players from different teams and analyse the potential impact on the winning percentage of both teams. Consider player strengths, team dynamics, and historical performances.

[77]:
```
#Insufficient data to make to predication
print("Hello World !!!")
```

```
Hello World !!!
```

10 7. Winning percentage

Compare the winning percentage of all teams against every other team ##### a) considering the ground factor ##### b) without considering the ground factor

[81]:
```
#Insufficient data to make to predication
#Number of wins and number of losts are not given

print("Hellow World !!!")
```

```
Hellow World !!!
```

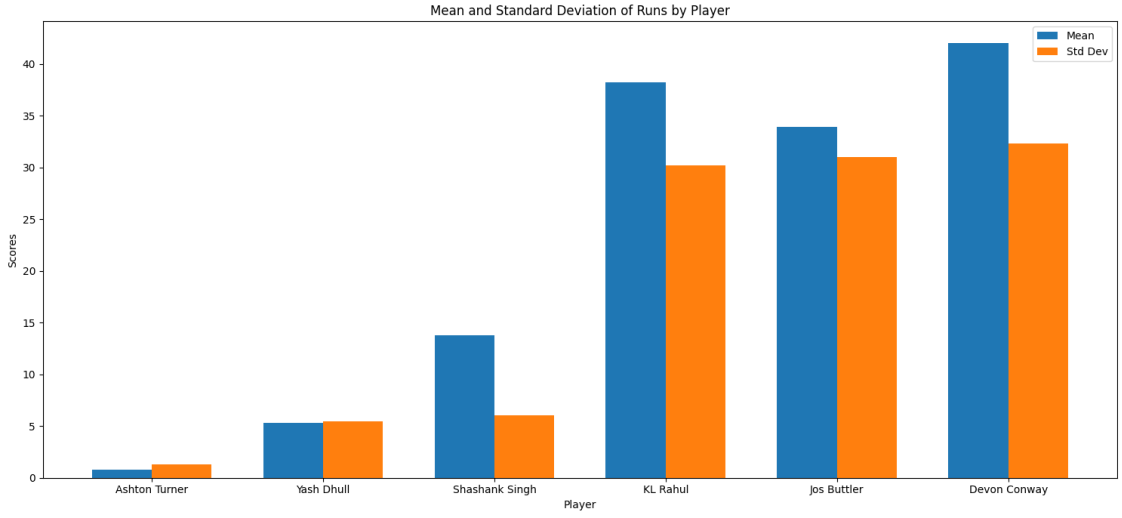
11 8. Consistency in Runs Accumulation:

Analyse the consistency of players in accumulating runs. Identify players with a consistent run-scoring pattern and those with more variable performances.

This code analyzes the mean and standard deviation of runs scored by cricket players from different teams. It iterates through batsmen data in Excel files for each team, calculating the mean and standard deviation of their runs. The resulting statistics are sorted, and the top three and bottom three performers are selected. A bar chart is then generated to visualize the mean and standard deviation of runs for these players.

[65]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players={}

for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    for sheet in xls.sheet_names:
        dfs= pd.read_excel(xls, sheet_name=sheet)
        players[sheet]=[]
        for index, row in dfs.iterrows():
            if row['Runs']!='-':
                row["Runs"]=str(row["Runs"]).replace("*","")
                players[sheet].append(int(row['Runs']))
```

```
players_stats = {player: (np.mean(runs), np.std(runs)) for player, runs in␣
 ↪players.items() if runs}

sorted_players = sorted(players_stats.items(), key=lambda x: x[1][1])

top_3 = sorted_players[:3]
bottom_3 = sorted_players[-3:]

labels = [player[0] for player in top_3 + bottom_3]
means = [player[1][0] for player in top_3 + bottom_3]
stds = [player[1][1] for player in top_3 + bottom_3]

x = np.arange(len(labels))
width = 0.35

fig, ax = plt.subplots(figsize=(15, 7))
rects1 = ax.bar(x - width/2, means, width, label='Mean')
rects2 = ax.bar(x + width/2, stds, width, label='Std Dev')

ax.set_xlabel('Player')
ax.set_ylabel('Scores')
ax.set_title('Mean and Standard Deviation of Runs by Player')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

fig.tight_layout()

plt.show()
```



12 9. Positional Impact on Performance:

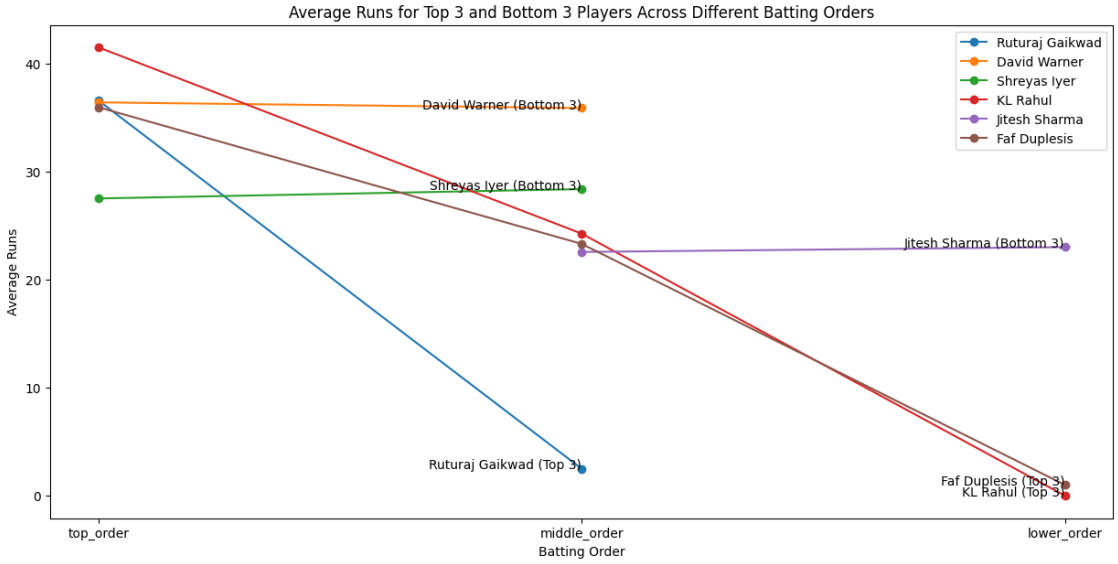
How does the batting position impact the average runs scored and strike rate? Are certain positions associated with higher or lower performance?

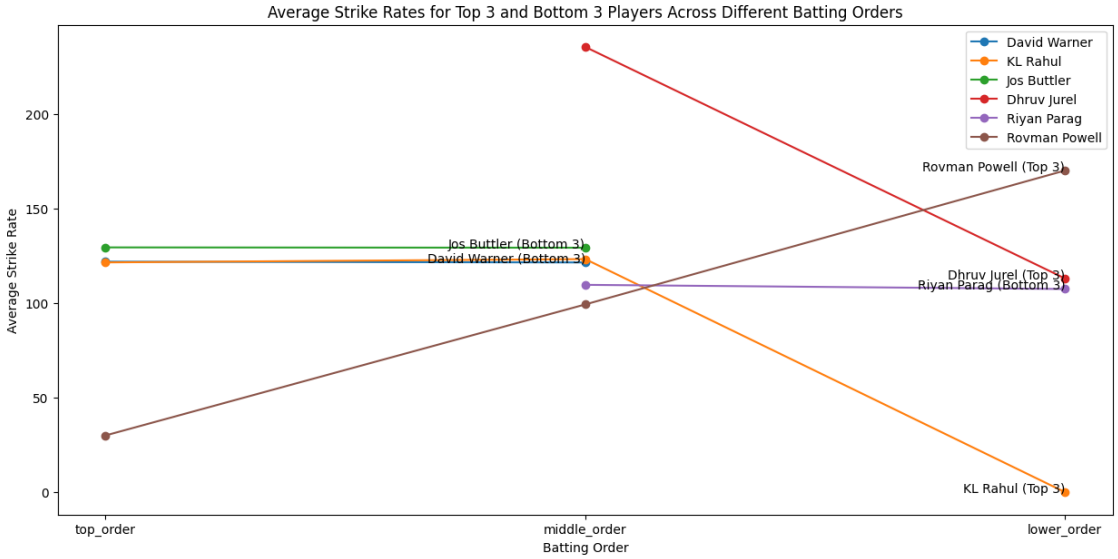
This code analyzes and visualizes the average runs and strike rates for the top and bottom three players across different batting orders (top order, middle order, lower order) in cricket teams. It computes the variance of average runs and strike rates and then identifies the top and bottom performers based on these variances. The resulting players are compared in separate line plots showcasing their average runs and strike rates across different batting orders.

[66]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players={}

for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    for sheet in xls.sheet_names:
        dfs= pd.read_excel(xls, sheet_name=sheet)
        players[sheet]={"top_order":[[],[]],"middle_order":
 ↪[[],[]],"lower_order":[[],[]]}
        for index, row in dfs.iterrows():
            if row['Runs']!='-' and row['Posn']!='-' and row['S/R']!='-':
                row["Runs"]=str(row["Runs"]).replace("*","")
                if row['Posn']<=3:
                    players[sheet]['top_order'][0].append(int(row['Runs']))
                    players[sheet]['top_order'][1].append(float(row['S/R']))
                elif row['Posn']<=6:
                    players[sheet]['middle_order'][0].append(int(row['Runs']))
                    players[sheet]['middle_order'][1].append(float(row['S/R']))
                else:
                    players[sheet]['lower_order'][0].append(int(row['Runs']))
                    players[sheet]['lower_order'][1].append(float(row['S/R']))

players_var_runs = {
    player: np.var([np.mean([float(i) for i in player_data[pos][0]]) for pos in␣
 ↪['top_order', 'middle_order', 'lower_order'] if player_data[pos][0]])
    for player, player_data in players.items() if sum(len(player_data[pos][0])␣
 ↪> 0 for pos in ['top_order', 'middle_order', 'lower_order']) >= 2
}
players_var_sr = {
    player: np.var([np.mean([float(i) for i in player_data[pos][1]]) for pos in␣
 ↪['top_order', 'middle_order', 'lower_order'] if player_data[pos][1]])
    for player, player_data in players.items() if sum(len(player_data[pos][1])␣
 ↪> 0 for pos in ['top_order', 'middle_order', 'lower_order']) >= 2
}

sorted_players_runs = sorted(players_var_runs.items(), key=lambda x: x[1])
```

```
sorted_players_sr = sorted(players_var_sr.items(), key=lambda x: x[1])

top_3_runs = sorted_players_runs[-3:]
bottom_3_runs = sorted_players_runs[:3]
top_3_sr = sorted_players_sr[-3:]
bottom_3_sr = sorted_players_sr[:3]

players_avg_runs = {
    player: {
        position: np.mean([float(i) for i in data[0]])
        for position, data in player_data.items() if data[0]
    }
    for player, player_data in players.items() if player in [p[0] for p in␣
 ↪top_3_runs + bottom_3_runs]
}
players_avg_sr = {
    player: {
        position: np.mean([float(i) for i in data[1]])
        for position, data in player_data.items() if data[1]
    }
    for player, player_data in players.items() if player in [p[0] for p in␣
 ↪top_3_sr + bottom_3_sr]
}

fig, ax = plt.subplots(figsize=(15, 7))

for player, player_data in players_avg_runs.items():
    positions = list(player_data.keys())
    avg_runs = [data for data in player_data.values()]
    ax.plot(positions, avg_runs, marker='o', label=player)
    if player in [p[0] for p in top_3_runs]:
        ax.text(positions[-1], avg_runs[-1], f'{player} (Top 3)',␣
 ↪horizontalalignment='right')
    elif player in [p[0] for p in bottom_3_runs]:
        ax.text(positions[-1], avg_runs[-1], f'{player} (Bottom 3)',␣
 ↪horizontalalignment='right')

ax.set_xlabel('Batting Order')
ax.set_ylabel('Average Runs')
ax.set_title('Average Runs for Top 3 and Bottom 3 Players Across Different␣
 ↪Batting Orders')
ax.legend()

plt.show()

fig, ax = plt.subplots(figsize=(15, 7))
```

```
for player, player_data in players_avg_sr.items():
    positions = list(player_data.keys())
    avg_sr = [data for data in player_data.values()]
    ax.plot(positions, avg_sr, marker='o', label=player)
    if player in [p[0] for p in top_3_sr]:
        ax.text(positions[-1], avg_sr[-1], f'{player} (Top 3)',␣
 ↪horizontalalignment='right')
    elif player in [p[0] for p in bottom_3_sr]:
        ax.text(positions[-1], avg_sr[-1], f'{player} (Bottom 3)',␣
 ↪horizontalalignment='right')

ax.set_xlabel('Batting Order')
ax.set_ylabel('Average Strike Rate')
ax.set_title('Average Strike Rates for Top 3 and Bottom 3 Players Across␣
 ↪Different Batting Orders')
ax.legend()

plt.show()
```





13 10. Comparison of Strike Rates:

Compare the overall strike rates of different players. Identify players with consistently high strike rates and those with lower rates.

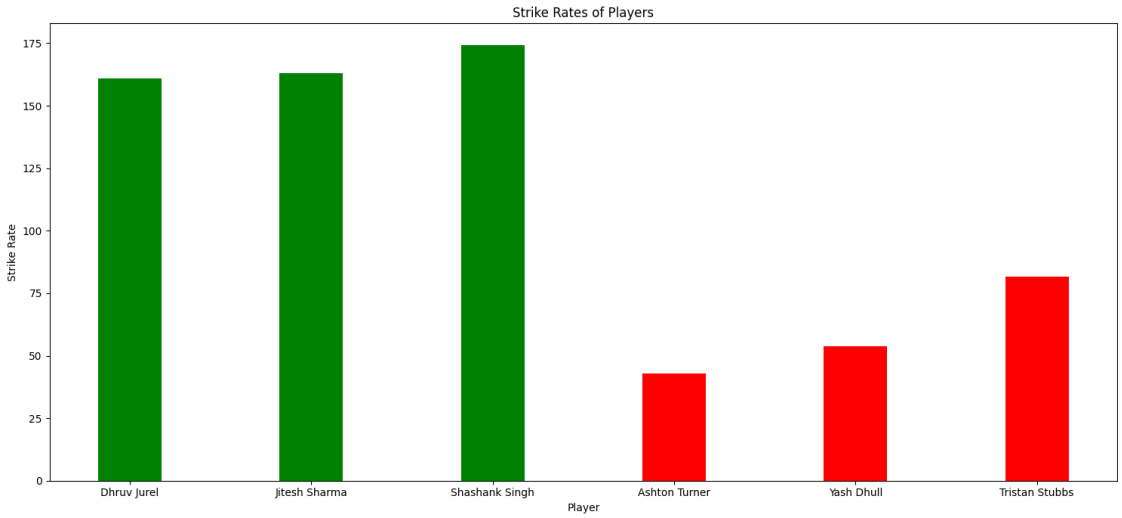
This code analyzes and visualizes the strike rates of cricket players, highlighting the top and bottom three performers. It calculates the mean, variance, and positive-mean strike rates for each player, then identifies and compares the top and bottom three players based on their positive-mean strike rates. The resulting data is plotted using a bar chart, with different colors indicating the top, bottom, and other players.

Explanation:

1. The code reads strike rate data from Excel files for each team, calculating the mean, variance, and positive-mean strike rates for each player.
2. The top and bottom three performers are identified based on their positive-mean strike rates.
3. A bar chart is created to visualize the positive-mean strike rates for these selected players, with different colors distinguishing between the top, bottom, and other players.
4. The chart provides insights into the relative performance of players based on their positive-mean strike rates, highlighting both top and bottom performers.

[67]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players={}

for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    for sheet in xls.sheet_names:
        dfs= pd.read_excel(xls, sheet_name=sheet)
```

```
        players[sheet]=[]
        for index, row in dfs.iterrows():
            if row['S/R']!='-' :
                row["S/R"]=str(row["S/R"]).replace("*","")
                players[sheet].append(row['S/R'])
players_stats = {
    player: (np.mean([float(i) for i in player_data]), np.var([float(i) for i in player_data]), np.mean([float(i) for i in player_data if float(i) > 0]))
    for player, player_data in players.items() if player_data
}
sorted_players = sorted(players_stats.items(), key=lambda x: x[1][2])

top_3 = sorted_players[-3:]
bottom_3 = sorted_players[:3]

labels = [player[0] for player in top_3 + bottom_3]
strike_rates = [player[1][2] for player in top_3 + bottom_3]

x = np.arange(len(labels))
width = 0.35

colors = ['red' if player in [p[0] for p in bottom_3] else 'green' if player in [p[0] for p in top_3] else 'blue' for player in labels]

fig, ax = plt.subplots(figsize=(15, 7))
rects = ax.bar(x, strike_rates, width, color=colors)

ax.set_xlabel('Player')
ax.set_ylabel('Strike Rate')
ax.set_title('Strike Rates of Players')
ax.set_xticks(x)
ax.set_xticklabels(labels)

fig.tight_layout()

plt.show()
```



14 11. Innings-wise Performance:

Analyse the performance of players in the 1st and 2nd innings. Are there notable differences in runs scored and strike rates?

This code analyzes and visualizes the average strike rates of cricket players in the 1st and 2nd innings across different teams. It reads strike rate data from Excel files for each team, calculates the average strike rate for each player in the 1st and 2nd innings, and then plots the results using a line chart.

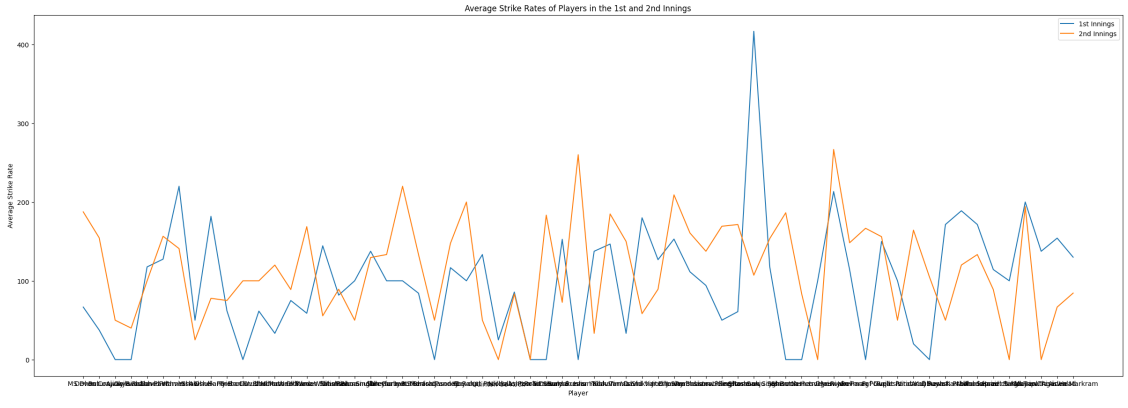
Explanation:

1. The code reads strike rate data from Excel files for each team, separating players based on their innings (1st or 2nd).
2. Average strike rates are calculated for each player in both the 1st and 2nd innings.
3. The resulting statistics are plotted using a line chart, with different lines representing the average strike rates in the 1st and 2nd innings.
4. The chart provides insights into the comparative performance of players in terms of strike rates across different innings, helping to identify any patterns or trends.

Please note that the readability of the chart might be affected if there are a large number of players. In such cases, you may consider additional adjustments to enhance visualization.

[68]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players={}

players_1st_innings = {}
players_2nd_innings = {}

for i in teams:
```

```
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    for sheet in xls.sheet_names:
        dfs= pd.read_excel(xls, sheet_name=sheet)
        players_1st_innings[sheet] = []
        players_2nd_innings[sheet] = []
        for index, row in dfs.iterrows():
            if row['S/R']!='-' :
                row["S/R"]=str(row["S/R"]).replace("*","")
                if row['Innings'] == 1:
                    players_1st_innings[sheet].append(row['S/R'])
                elif row['Innings'] == 2:
                    players_2nd_innings[sheet].append(row['S/R'])

players_stats_1st_innings = {player: np.mean([float(i) for i in player_data])␣
 ↪for player, player_data in players_1st_innings.items() if player_data}
players_stats_2nd_innings = {player: np.mean([float(i) for i in player_data])␣
 ↪for player, player_data in players_2nd_innings.items() if player_data}

fig, ax = plt.subplots(figsize=(30, 10))
ax.plot(players_stats_1st_innings.keys(), players_stats_1st_innings.values(),␣
 ↪label='1st Innings')
ax.plot(players_stats_2nd_innings.keys(), players_stats_2nd_innings.values(),␣
 ↪label='2nd Innings')

ax.set_xlabel('Player')
ax.set_ylabel('Average Strike Rate')
ax.set_title('Average Strike Rates of Players in the 1st and 2nd Innings')
ax.legend()

plt.show()
```



15 12. Create the best batting line-up;

a) For any one team against every other team

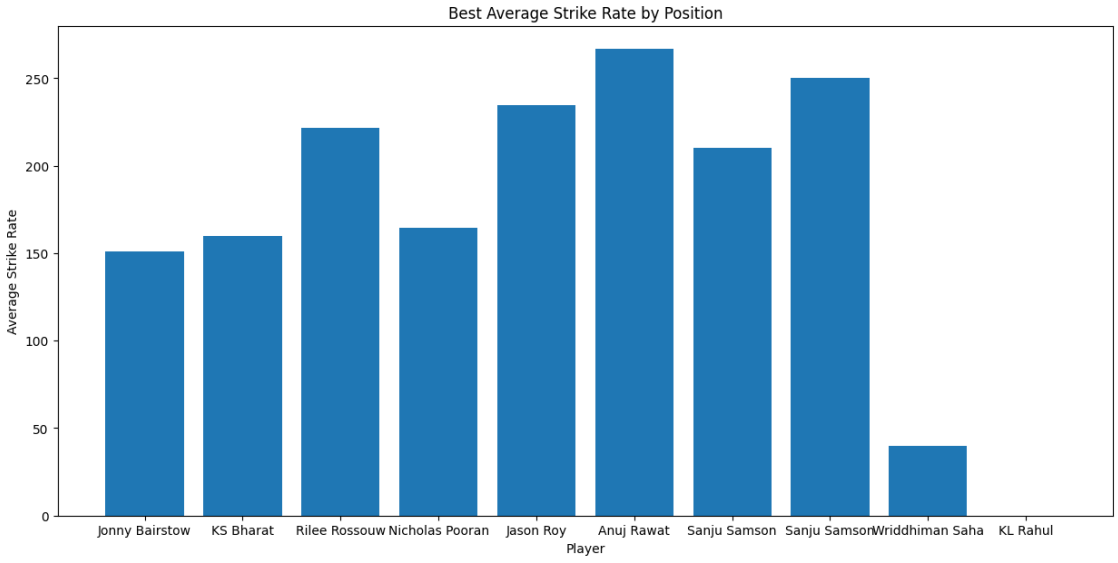
b) Of 2024 season

This code identifies and displays the player with the highest average strike rate for each batting position (1 to 11) across different cricket teams. It calculates the average strike rate for each player in each position and then selects the player with the highest average for each position.

[69]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players={}

for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    for sheet in xls.sheet_names:
        dfs= pd.read_excel(xls, sheet_name=sheet)
        for index, row in dfs.iterrows():
            if row['S/R']!='-' :
                row["S/R"]=str(row["S/R"]).replace("*","")
                position = int(row['Posn'])
                if position > 11:
                    continue
                if position not in players:
                    players[position] = {}
                if sheet not in players[position]:
                    players[position][sheet] = []
                players[position][sheet].append(float(row['S/R']))

best_players = {}
for position, player_data in players.items():
    player_avgs = {player: np.mean(strike_rates) for player, strike_rates in␣
 ↪player_data.items()}
    best_player = max(player_avgs.items(), key=lambda x: x[1])
    best_players[position] = best_player

sorted_best_players = dict(sorted(best_players.items()))

for position, player in sorted_best_players.items():
    print(f'Position: {position}, Player: {player[0]}, Average Strike Rate:␣
 ↪{player[1]}')

plt.figure(figsize=(15, 7))
plt.bar(range(len(sorted_best_players)), [player[1] for player in␣
 ↪sorted_best_players.values()], align='center')
plt.xticks(range(len(sorted_best_players)), [str(player[0]) for player in␣
 ↪sorted_best_players.values()])
```

```
plt.xlabel('Player')
plt.ylabel('Average Strike Rate')
plt.title('Best Average Strike Rate by Position')
plt.show()
```

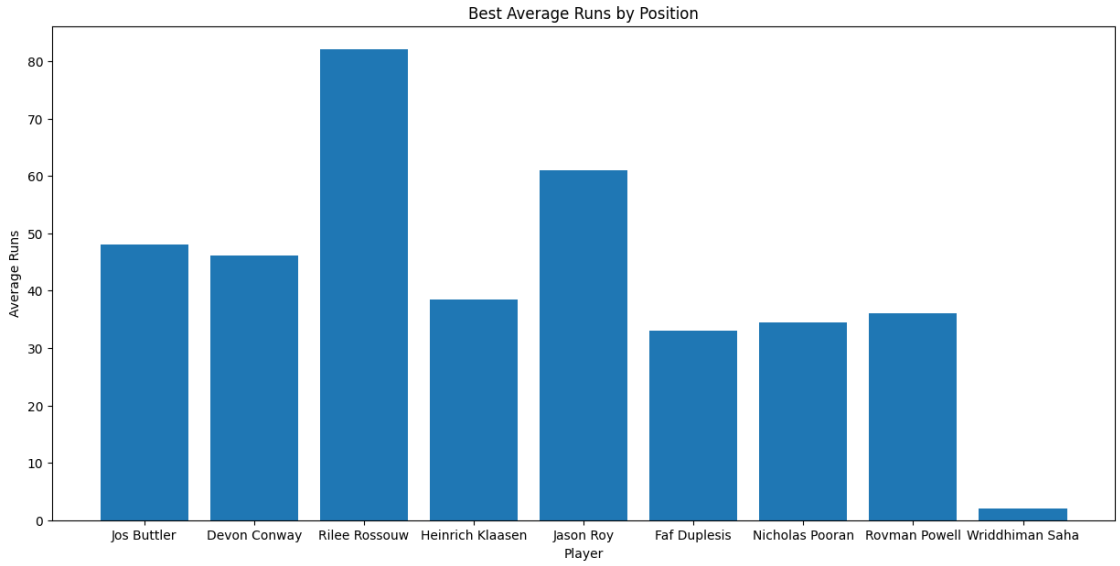
```
Position: 1, Player: Jonny Bairstow, Average Strike Rate: 150.91428571428568
Position: 2, Player: KS Bharat, Average Strike Rate: 160.0
Position: 3, Player: Rilee Rossouw, Average Strike Rate: 221.62
Position: 4, Player: Nicholas Pooran, Average Strike Rate: 164.78142857142856
Position: 5, Player: Jason Roy, Average Strike Rate: 234.62
Position: 6, Player: Anuj Rawat, Average Strike Rate: 266.67
Position: 7, Player: Sanju Samson, Average Strike Rate: 210.0
Position: 8, Player: Sanju Samson, Average Strike Rate: 250.0
Position: 9, Player: Wriddhiman Saha, Average Strike Rate: 40.0
Position: 11, Player: KL Rahul, Average Strike Rate: 0.0
```



[70]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players={}

for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    for sheet in xls.sheet_names:
        dfs= pd.read_excel(xls, sheet_name=sheet)
        for index, row in dfs.iterrows():
            if row['Runs']!='-' :
                row["Runs"]=str(row["Runs"]).replace("*","")
                position = int(row['Posn'])
```

```
                if position >= 11:
                    continue
                if position not in players:
                    players[position] = {}
                if sheet not in players[position]:
                    players[position][sheet] = []
                players[position][sheet].append(float(row['Runs']))

best_players = {}
for position, player_data in players.items():
    player_avgs = {player: np.mean(runs) for player, runs in player_data.
 ↪items()}
    best_player = max(player_avgs.items(), key=lambda x: x[1])
    best_players[position] = best_player

sorted_best_players = dict(sorted(best_players.items()))

for position, player in sorted_best_players.items():
    print(f'Position: {position}, Player: {player[0]}, Average Runs:␣
 ↪{player[1]}')

plt.figure(figsize=(15, 7))
plt.bar(range(len(sorted_best_players)), [player[1] for player in␣
 ↪sorted_best_players.values()], align='center')
plt.xticks(range(len(sorted_best_players)), [str(player[0]) for player in␣
 ↪sorted_best_players.values()])
plt.xlabel('Player')
plt.ylabel('Average Runs')
plt.title('Best Average Runs by Position')
plt.show()
```

```
Position: 1, Player: Jos Buttler, Average Runs: 48.095238095238095
Position: 2, Player: Devon Conway, Average Runs: 46.15
Position: 3, Player: Rilee Rossouw, Average Runs: 82.0
Position: 4, Player: Heinrich Klaasen, Average Runs: 38.5
Position: 5, Player: Jason Roy, Average Runs: 61.0
Position: 6, Player: Faf Duplesis, Average Runs: 33.0
Position: 7, Player: Nicholas Pooran, Average Runs: 34.5
Position: 8, Player: Rovman Powell, Average Runs: 36.0
Position: 9, Player: Wriddhiman Saha, Average Runs: 2.0
```



16 13. 2024 season

Who will get orange cap (highest run scorer) and purple cap (leading wicket taker) in 2024 season?

This code predicts the potential Orange Cap holder (the player with the highest run aggregate) for the next cricket season based on historical run data. It uses a Random Forest Regressor model to predict the average runs per match for each player and then estimates the total runs for the upcoming season.

Explanation:

1. The code reads run data from Excel files for each team, considering the batting performance of players in different sheets.
2. It organizes the runs data by player and categorizes it into periods of 5 years.
3. For each player, the code builds a Random Forest Regressor model using the period and the average runs per period as features.
4. It predicts the average runs per match for the next period (next 5 years) using the trained model.
5. The code calculates the predicted total runs for the upcoming season (considering 14 matches).
6. The player with the highest predicted total runs is identified as the potential Orange Cap holder for the next season.
7. The predicted Orange Cap holder and their estimated total runs are printed.

This analysis leverages machine learning to forecast player performance based on historical data, providing insights into potential standout performers in the upcoming cricket season.

Devon Conway : Orange

[82]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players_runs = {}
```

```
for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BATSMEN/
 ↪{i}_BATSMEN.xlsx')
    for sheet in xls.sheet_names:
        dfs = pd.read_excel(xls, sheet_name=sheet)
        for index, row in dfs.iterrows():
            if row['Runs'] != '-':
                date = pd.to_datetime(row['Date'])
                year = date.year
                runs = float(str(row['Runs']).replace('*', ''))
                if sheet not in players_runs:
                    players_runs[sheet] = {}
                period = year // 5
                if period not in players_runs[sheet]:
                    players_runs[sheet][period] = []
                players_runs[sheet][period].append(runs)

player_predictions = {}
for player, runs in players_runs.items():
    X = np.array(list(runs.keys())).reshape(-1, 1) * 5
    y = np.array([np.mean(r) for r in runs.values()]).reshape(-1, 1)
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X, y.ravel())
    next_period = max(runs.keys()) + 1
    player_predictions[player] = model.predict([[next_period * 5]])[0]

best_player = max(player_predictions, key=player_predictions.get)
print(f'\n\nThe orange cap for the next season is expected to be:␣
 ↪{best_player}')

predicted_runs_per_match = player_predictions[best_player]
predicted_total_runs = predicted_runs_per_match * 14
print(f'\nThe predicted total runs {best_player} is: {predicted_total_runs}')
```

```
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
```

```
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
```

```
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
```

```
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
```

```
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
```

```
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
<ipython-input-82-d36137531195>:10: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  date = pd.to_datetime(row['Date'])
```

```
The orange cap for the next season is expected to be: Devon Conway

The predicted total runs Devon Conway is: 588.0
```

[91]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
players_wickets = {}

for i in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{i}/BOWLERS/
 ↪{i}_BOWLERS.xlsx')
    try:
        for sheet in xls.sheet_names:
            dfs = pd.read_excel(xls, sheet_name=sheet)
            dfs['Date'] = pd.to_datetime(dfs['Date'])
            dfs = dfs.sort_values('Date')
            dfs['Year'] = dfs['Date'].dt.year
            dfs['Wkts'] = dfs['Wkts'].replace('-', '0').replace('*', '0').
 ↪astype(float)
            yearly_wickets = dfs.groupby('Year').apply(lambda x: x['Wkts'].
 ↪iloc[-1] - x['Wkts'].iloc[0])
            players_wickets[sheet] = yearly_wickets.to_dict()
    except :
        pass

current_year = pd.to_datetime('today').year
last_5_years = range(current_year - 5, current_year + 1)

player_totals = {}
for player, wkts in players_wickets.items():
    last_5_years_wkts = {year: wickets for year, wickets in wkts.items() if␣
 ↪year in last_5_years}
    if last_5_years_wkts:
        total_wickets = sum(last_5_years_wkts.values())
        player_totals[player] = total_wickets

best_player = max(player_totals, key=player_totals.get)
print(f'\n\n\nThe purple cap for the next season is expected to be:␣
 ↪{best_player}')
```

```
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
```

```
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
```

```
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
```

```
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
```

```
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
```

```
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
```

```
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
<ipython-input-91-36123c006ab7>:9: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  dfs['Date'] = pd.to_datetime(dfs['Date'])
```

```
The purple cap for the next season is expected to be: Jasprit Bumrah
```

17 14. Impact of Player Removal:

Explore the potential impact on a team if a key player is removed. Analyse changes in batting, bowling, and fielding dynamics of each team.

[79]:
```
#Insuffcient Data
#Fielding dynamics is not given in the dataset
print("Hello World !!!")
```

```
Hello World !!!
```

18 15. Likely Struggling Team in Upcoming Season:

Analyse historical data to predict which team is likely to face challenges and struggle the most in the upcoming season.

[85]:
```
#Insuffcient Data
#Number of wins and number of losts are not given in the dataset
#Also there is no data about who won a particular match
print("Hello World !!!")
```

```
Hello World !!!
```

19 16. Batsman-Specific Dismissals:

Explore which batsmen are frequently dismissed by specific bowlers. Identify the batsmen who struggle the most against particular bowlers.

The code analyzes batsmen dismissal data from multiple cricket teams. It identifies the bowler who has dismissed each batsman most frequently across various matches. The output displays the batsman and the bowler who has historically taken their wicket most often, providing insights into potential challenging matchups for batsmen.

[84]:
```
teams = ['CSK', 'DC', 'GT', 'KKR', 'LSG', 'MI', 'PBKS', 'RR', 'RCB', 'SRH']
batsmen_dismissals = {}

for team in teams:
    xls = pd.ExcelFile(f'/content/drive/MyDrive/cyberlab/Teams/{team}/BATSMEN/
 ↪{team}_BATSMEN.xlsx')
    dfs = {}
    for sheet in xls.sheet_names:
        dfs[sheet] = pd.read_excel(xls, sheet_name=sheet)

    for batsman, df in dfs.items():
        for index, row in df.iterrows():
            if isinstance(row['How Dismissed'], str):
                bowler = row['How Dismissed'].split(' b ')[-1]
                if bowler not in  ['not out','did not bat','run out']:
                    if batsman not in batsmen_dismissals:
                        batsmen_dismissals[batsman] = {}
                    if bowler not in batsmen_dismissals[batsman]:
                        batsmen_dismissals[batsman][bowler] = 0
```

```
                    batsmen_dismissals[batsman][bowler] += 1

for batsman, bowlers in batsmen_dismissals.items():
    worst_bowler = max(bowlers, key=bowlers.get)
    print(f'{batsman} is most frequently dismissed by {worst_bowler}')
```

```
MS Dhoni is most frequently dismissed by Z Khan
Devon Conway is most frequently dismissed by Mohammed Shami
Ruturaj Gaikwad is most frequently dismissed by T A Boult
Ajinkya Rahane is most frequently dismissed by B Kumar
Rishabh Pant is most frequently dismissed by J J Bumrah
David Warner is most frequently dismissed by Harbhajan Singh
Prithvi Shaw is most frequently dismissed by D L Chahar
Yash Dhull is most frequently dismissed by R P Meredith
Abishek Porel is most frequently dismissed by b Rashid Khan
Harry Brook is most frequently dismissed by b Y S Chahal
Tristan Stubbs is most frequently dismissed by Mukesh Choudhary
David Miller is most frequently dismissed by Y S Chahal
Shubman Gill is most frequently dismissed by B Kumar
Mathew Wade is most frequently dismissed by R Sharma
Wriddhiman Saha is most frequently dismissed by P P Chawla
Kane Williamson is most frequently dismissed by Kuldeep Yadav
Nitish Rana is most frequently dismissed by Y S Chahal
Rinku Singh is most frequently dismissed by J J Bumrah
Rahmanullah Gurbaz is most frequently dismissed by b N Ellis
Shreyas Iyer is most frequently dismissed by A D Russell
Sherfane Rutherford is most frequently dismissed by D S Kulkarni
KS Bharat is most frequently dismissed by A D Russell
Manish Pandey is most frequently dismissed by B Kumar
Jason Roy is most frequently dismissed by P P Chawla
KL Rahul is most frequently dismissed by D S Kulkarni
Devdutt Padikkal is most frequently dismissed by S N Thakur
Quinton de Kock is most frequently dismissed by Y S Chahal
Nicholas Pooran is most frequently dismissed by S N Thakur
Ashton Turner is most frequently dismissed by M Ashwin
Rohit Sharma is most frequently dismissed by R Vinay Kumar
Dewald Brevis is most frequently dismissed by C V Varun
Suryakumar Yadav is most frequently dismissed by Sandeep Sharma
Ishan Kishan is most frequently dismissed by Imran Tahir
Tilak Varma is most frequently dismissed by K K Ahmed
Tim David is most frequently dismissed by T U Deshpande
Vishnu Vinod is most frequently dismissed by M M Sharma
Shikhar Dhawan is most frequently dismissed by P Kumar
Jitesh Sharma is most frequently dismissed by D Pretorius
Jonny Bairstow is most frequently dismissed by Y S Chahal
Prabhsimran Singh is most frequently dismissed by Sandeep Sharma
Liam Livingstone is most frequently dismissed by Rashid Khan
```

```
Rilee Rossouw is most frequently dismissed by b A Nehra
Shashank Singh is most frequently dismissed by Mukesh Choudhary
Sanju Samson is most frequently dismissed by Shivam Mavi
Jos Buttler is most frequently dismissed by A Mishra
Shimron Hetmeyer is most frequently dismissed by J J Bumrah
Yashasvi Jaiswal is most frequently dismissed by Arshdeep Singh
Dhruv Jurel is most frequently dismissed by Akash Singh
Riyan Parag is most frequently dismissed by J O Holder
Rovman Powell is most frequently dismissed by Basil Thampi
Faf Duplesis is most frequently dismissed by Rashid Khan
Rajat Patidar is most frequently dismissed by b T A Boult
Virat Kohli is most frequently dismissed by A Nehra
Anuj Rawat is most frequently dismissed by L H Ferguson
Dinesh Karthik is most frequently dismissed by D J J Bravo
Suyash Prabhudessai is most frequently dismissed by b M M Theekshana
Abdul Samad is most frequently dismissed by Ravi Bishnoi
Anmolpreet Singh is most frequently dismissed by b D L Chahar
Heinrich Klaasen is most frequently dismissed by b S N Thakur
Rahul Tripathi is most frequently dismissed by A D Russell
Mayank Agarwal is most frequently dismissed by A D Russell
Travis Head is most frequently dismissed by Sandeep Sharma
Aiden Markram is most frequently dismissed by K H Pandya
```

20 Additionally:

Though the data about who won each match was not given, We tried to find it using cumulative score, the dates provided and the opponent.

Problems Faced:

1.Not enough historical data - For example Sachin's data is not present, only historical data of current lineup is present.

2.There Were some clashes in dates.