```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.1.31)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.1)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.10)
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.4)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-packages (from kaggle) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.3.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
! mkdir ~/.kaggle
```

```
cp  /content/drive/MyDrive/Kaggle_Api/kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
! kaggle datasets download patrickb1912/ipl-complete-dataset-20082020
```

```
Dataset URL: https://www.kaggle.com/datasets/patrickb1912/ipl-complete-dataset-20082020
License(s): DbCL-1.0
```

```
! unzip ipl-complete-dataset-20082020.zip
```

```
Archive:  ipl-complete-dataset-20082020.zip
  inflating: deliveries.csv
  inflating: matches.csv
```

Step 1: Data Loading and Initial Exploration

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

cricket_df = pd.read_csv('deliveries.csv')
# Let's examine the basic properties of the dataset

# Display the first few rows
print("First 5 rows:")
print(cricket_df.head())

# Dataset dimensions
print("\nDataset dimensions:", cricket_df.shape)

# Check data types
print("\nData types:")
print(cricket_df.dtypes)

# Check for missing values
print("\nMissing values per column:")
print(cricket_df.isnull().sum())

# Basic statistics
print("\nBasic statistics:")
print(cricket_df.describe())
```

```
batsman_runs         int64
extra_runs           int64
total_runs           int64
extras_type         object
is_wicket            int64
player_dismissed    object
dismissal_kind      object
fielder             object
dtype: object

Missing values per column:
match_id                 0
inning                   0
batting_team             0
bowling_team             0
over                     0
ball                     0
batter                   0
bowler                   0
non_striker              0
batsman_runs             0
extra_runs               0
total_runs               0
extras_type         246795
is_wicket                0
player_dismissed    247970
dismissal_kind      247970
fielder             251566
dtype: int64

Basic statistics:
           match_id         inning           over           ball  \
count  2.609200e+05  260920.000000  260920.000000  260920.000000
mean   9.070665e+05       1.483531       9.197677       3.624486
std    3.679913e+05       0.502643       5.683484       1.814920
min    3.359820e+05       1.000000       0.000000       1.000000
25%    5.483340e+05       1.000000       4.000000       2.000000
50%    9.809670e+05       1.000000       9.000000       4.000000
75%    1.254066e+06       2.000000      14.000000       5.000000
max    1.426312e+06       6.000000      19.000000      11.000000

        batsman_runs     extra_runs     total_runs      is_wicket
count  260920.000000  260920.000000  260920.000000  260920.000000
mean        1.265001       0.067806       1.332807       0.049632
std         1.639298       0.343265       1.626416       0.217184
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         1.000000       0.000000       1.000000       0.000000
75%         1.000000       0.000000       1.000000       0.000000
max         6.000000       7.000000       7.000000       1.000000
```

Step 2: Data Cleaning

```
# Convert columns to appropriate data types
cricket_df['match_id'] = cricket_df['match_id'].astype(int)
cricket_df['inning'] = cricket_df['inning'].astype(int)
cricket_df['over'] = cricket_df['over'].astype(int)
cricket_df['ball'] = cricket_df['ball'].astype(int)
cricket_df['batsman_runs'] = cricket_df['batsman_runs'].astype(int)
cricket_df['extra_runs'] = cricket_df['extra_runs'].astype(int)
cricket_df['total_runs'] = cricket_df['total_runs'].astype(int)
cricket_df['is_wicket'] = cricket_df['is_wicket'].astype(int)

# Handle missing values appropriately
# For dismissal fields, missing values are expected when no wicket falls
cricket_df['extras_type'] = cricket_df['extras_type'].fillna('none')

# Create a ball_id column for unique delivery identification
cricket_df['ball_id'] = cricket_df['match_id'].astype(str) + "_" + \
                        cricket_df['inning'].astype(str) + "_" + \
                        cricket_df['over'].astype(str) + "_" + \
                        cricket_df['ball'].astype(str)
```

Step 3: Feature Engineering

```
# Add a column for boundaries
cricket_df['is_four'] = (cricket_df['batsman_runs'] == 4).astype(int)
cricket_df['is_six'] = (cricket_df['batsman_runs'] == 6).astype(int)

# Add a column for dot balls (no runs from bat)
cricket_df['is_dot'] = ((cricket_df['batsman_runs'] == 0) &
```

```
                        (cricket_df['extras_type'].isin(['none', 'legbyes', 'byes']))).astype(int)

# Calculate over number in the match (combines innings and over)
cricket_df['match_over'] = (cricket_df['inning'] - 1) * 20 + cricket_df['over']

# Create a phase column (PowerPlay: 0-5, Middle: 6-15, Death: 16-19)
conditions = [
    cricket_df['over'] <= 5,
    (cricket_df['over'] >= 6) & (cricket_df['over'] <= 15),
    cricket_df['over'] >= 16
]
choices = ['PowerPlay', 'Middle', 'Death']
cricket_df['phase'] = np.select(conditions, choices, default='Unknown')
```

Step 4: Team Performance Analysis

```
# Calculate team-wise performance metrics
team_batting = cricket_df.groupby('batting_team').agg({
    'total_runs': 'sum',
    'is_wicket': 'sum',
    'is_four': 'sum',
    'is_six': 'sum',
    'is_dot': 'sum',
    'ball_id': 'count'
}).reset_index()

# Calculate batting statistics
team_batting['batting_avg'] = team_batting['total_runs'] / team_batting['is_wicket'].replace(0, 1)
team_batting['batting_sr'] = (team_batting['total_runs'] / team_batting['ball_id']) * 100
team_batting['boundary_percentage'] = ((team_batting['is_four'] + team_batting['is_six']) / team_batting['ball_id']) * 100
team_batting['dot_percentage'] = (team_batting['is_dot'] / team_batting['ball_id']) * 100

print("Team Batting Performance:")
print(team_batting.sort_values('batting_sr', ascending=False))

# Calculate bowling statistics
team_bowling = cricket_df.groupby('bowling_team').agg({
    'total_runs': 'sum',
    'is_wicket': 'sum',
    'ball_id': 'count'
}).reset_index()

team_bowling['economy_rate'] = (team_bowling['total_runs'] / team_bowling['ball_id']) * 6
team_bowling['bowling_avg'] = team_bowling['total_runs'] / team_bowling['is_wicket'].replace(0, 1)
team_bowling['bowling_sr'] = team_bowling['ball_id'] / team_bowling['is_wicket'].replace(0, 1)

print("\nTeam Bowling Performance:")
print(team_bowling.sort_values('economy_rate'))
```

```
9          Lucknow Super Giants        7415        264     5226      8.513203
12                 Punjab Kings        9545        335     6719      8.523590
4                 Gujarat Lions        5090        151     3545      8.614951
17  Royal Challengers Bengaluru        2820         88     1801      9.394781

    bowling_avg  bowling_sr
7     26.837838   21.810811
11    28.810924   22.928571
14    21.557522   17.061947
1     26.049327   20.266816
0     25.122890   19.295071
15    27.802632   21.250000
10    25.956003   19.802011
8     26.698087   20.261612
3     27.116228   20.531798
13    27.519186   20.698512
16    27.013514   20.169275
18    27.097765   20.220670
6     28.309480   20.894981
2     26.271331   19.139932
5     25.083893   17.788591
9     28.087121   19.795455
12    28.492537   20.056716
4     33.708609   23.476821
17    32.045455   20.465909
```

Step 5: Player Performance Analysis

```
# Batsmen analysis
batsmen = cricket_df.groupby('batter').agg({
    'batsman_runs': 'sum',
    'is_wicket': 'sum',
    'is_four': 'sum',
    'is_six': 'sum',
    'ball_id': 'count'
}).reset_index()

batsmen['batting_avg'] = batsmen['batsman_runs'] / batsmen['is_wicket'].replace(0, 1)
batsmen['strike_rate'] = (batsmen['batsman_runs'] / batsmen['ball_id']) * 100
batsmen['boundary_percentage'] = ((batsmen['is_four'] + batsmen['is_six']) / batsmen['ball_id']) * 100

print("Top Batsmen by Runs:")
print(batsmen.sort_values('batsman_runs', ascending=False).head(10))

# Bowlers analysis
bowlers = cricket_df.groupby('bowler').agg({
    'total_runs': 'sum',
    'is_wicket': 'sum',
    'ball_id': 'count'
}).reset_index()

bowlers['overs'] = bowlers['ball_id'] / 6
bowlers['economy'] = bowlers['total_runs'] / bowlers['overs']
bowlers['bowling_avg'] = bowlers['total_runs'] / bowlers['is_wicket'].replace(0, 1)
bowlers['bowling_sr'] = bowlers['ball_id'] / bowlers['is_wicket'].replace(0, 1)

print("\nTop Bowlers by Wickets:")
print(bowlers.sort_values('is_wicket', ascending=False).head(10))
```

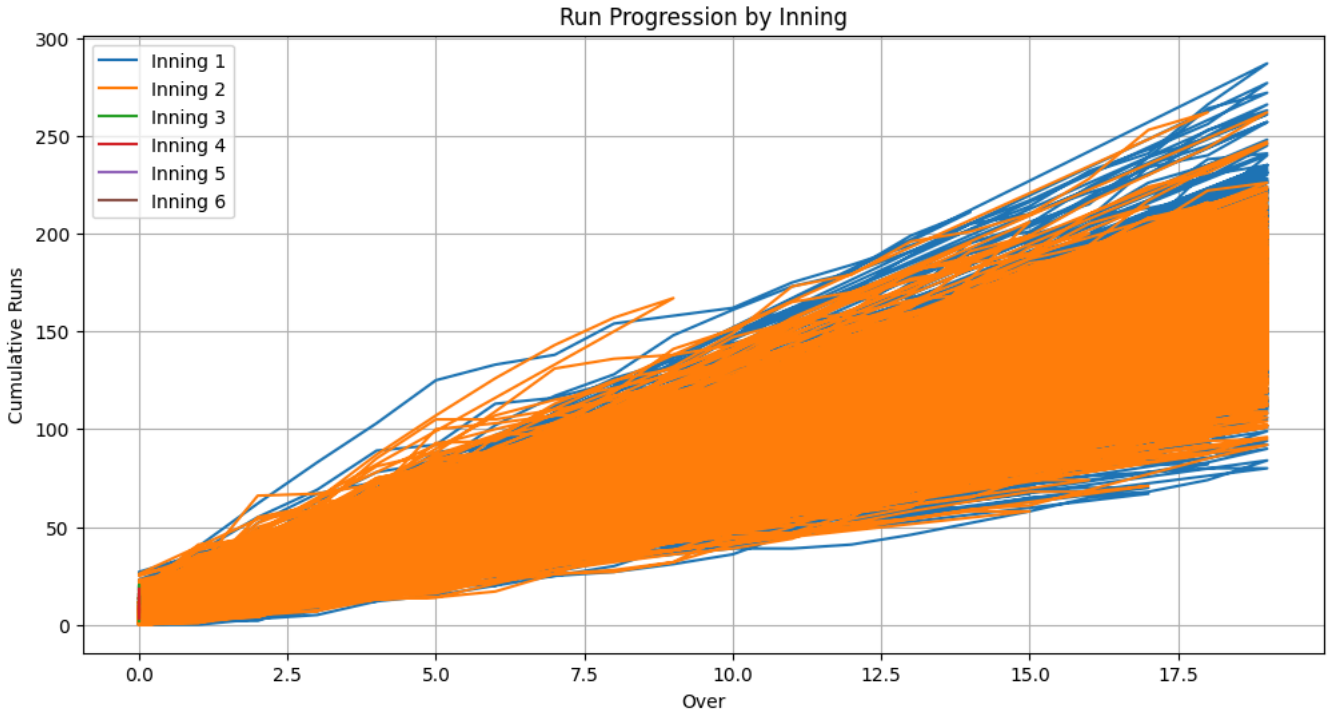
```
Top Batsmen by Runs:
              batter  batsman_runs  is_wicket  is_four  is_six  ball_id  \
631         V Kohli          8014        218      708     273     6236
512        S Dhawan          6769        194      768     153     5483
477       RG Sharma          6630        232      599     281     5183
147       DA Warner          6567        164      663     236     4849
546        SK Raina          5536        168      506     204     4177
374        MS Dhoni          5243        149      363     252     3947
30   AB de Villiers          5181        125      414     253     3487
124        CH Gayle          4997        128      408     359     3516
501      RV Uthappa          4954        184      481     182     3927
282      KD Karthik          4843        189      466     161     3687

     batting_avg  strike_rate  boundary_percentage
631    36.761468   128.511867            15.731238
512    34.891753   123.454313            16.797374
477    28.577586   127.918194            16.978584
147    40.042683   135.429986            18.539905
546    32.952381   132.535312            16.997845
374    35.187919   132.835065            15.581454
30     41.448000   148.580442            19.128190
124    39.039062   142.121729            21.814562
501    26.923913   126.152279            16.883117
282    25.624339   131.353404            17.005696

Top Bowlers by Wickets:
```

```
            bowler  total_runs  is_wicket  ball_id       overs   economy  \
524   YS Chahal        4681        213     3628  604.666667  7.741455
119    DJ Bravo        4436        207     3296  549.333333  8.075243
348   PP Chawla        5179        201     3895  649.166667  7.977920
446   SP Narine        4672        200     4146  691.000000  6.761216
355    R Ashwin        5435        198     4679  779.833333  6.969438
71      B Kumar        5051        195     4060  676.666667  7.464532
438  SL Malinga        3486        188     2974  495.666667  7.032952
8      A Mishra        4193        183     3444  574.000000  7.304878
193   JJ Bumrah        3840        182     3185  530.833333  7.233909
373   RA Jadeja        4917        169     3895  649.166667  7.574326

     bowling_avg  bowling_sr
524    21.976526   17.032864
119    21.429952   15.922705
348    25.766169   19.378109
446    23.360000   20.730000
355    27.449495   23.631313
71     25.902564   20.820513
438    18.542553   15.819149
8      22.912568   18.819672
193    21.098901   17.500000
373    29.094675   23.047337
```

Step 6: Match Progression Analysis

```
# Create run rate progression by over
over_progress = cricket_df.groupby(['match_id', 'inning', 'over']).agg({
    'total_runs': 'sum',
    'is_wicket': 'sum'
}).reset_index()

print("Over progression summary:")
print(over_progress.head())

# Group by match and inning to see match progression
# Using custom aggregation for cumulative sums
match_progress = over_progress.groupby(['match_id', 'inning']).apply(
    lambda x: pd.DataFrame({
        'over': x['over'],
        'cum_runs': x['total_runs'].cumsum(),
        'cum_wickets': x['is_wicket'].cumsum()
    })
).reset_index()

print("\nMatch progress summary:")
print(match_progress.head())

# Visualization code - for complete data
plt.figure(figsize=(12, 6))
for inning in cricket_df['inning'].unique():
    inning_data = match_progress[match_progress['inning'] == inning]
    if not inning_data.empty:
        plt.plot(inning_data['over'], inning_data['cum_runs'],
                 label=f'Inning {inning}')

plt.xlabel('Over')
plt.ylabel('Cumulative Runs')
plt.title('Run Progression by Inning')
plt.legend()
plt.grid(True)
plt.show()
```

```
Over progression summary:
   match_id  inning  over  total_runs  is_wicket
0    335982       1     0           3          0
1    335982       1     1          18          0
2    335982       1     2           6          0
3    335982       1     3          23          0
4    335982       1     4          10          0
<ipython-input-27-21937b8831db>:12: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is de
  match_progress = over_progress.groupby(['match_id', 'inning']).apply(

Match progress summary:
   match_id  inning  level_2  over  cum_runs  cum_wickets
0    335982       1        0     0         3            0
1    335982       1        1     1        21            0
2    335982       1        2     2        27            0
3    335982       1        3     3        50            0
4    335982       1        4     4        60            0
```

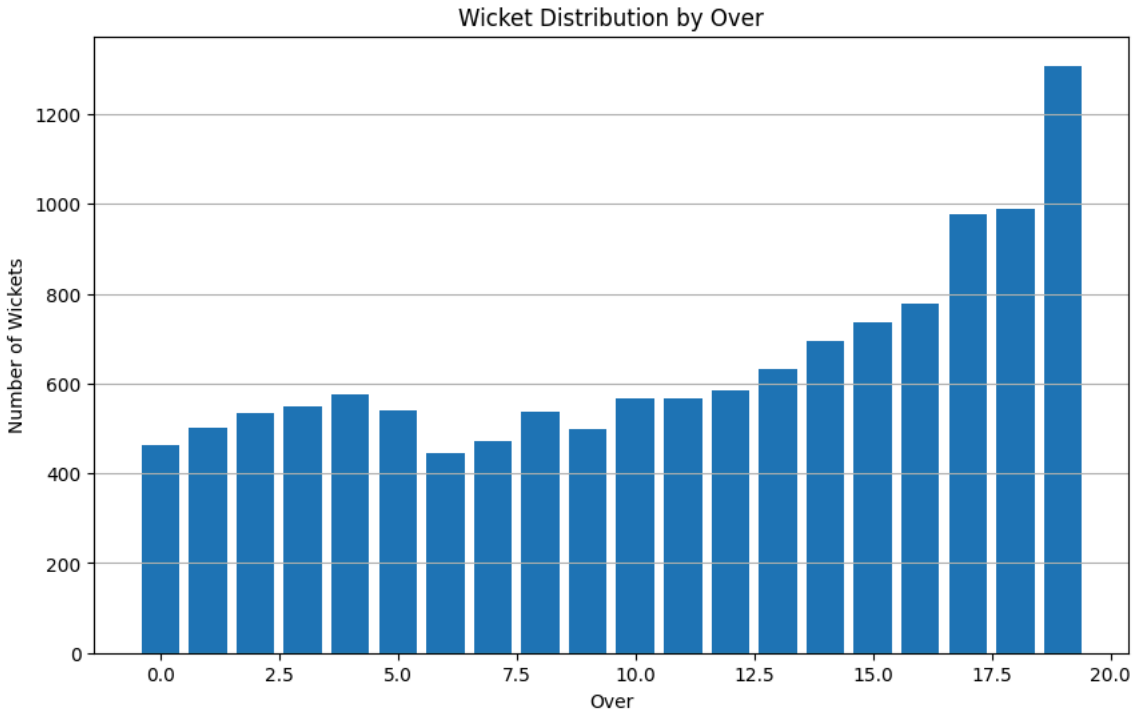


Step 7: Wicket Analysis

```
# Wicket Analysis
import pandas as pd
import matplotlib.pyplot as plt

# Count total wickets
wicket_count = cricket_df['is_wicket'].sum()
print(f"Total wickets in dataset: {wicket_count}")

# Create wicket distribution by over
wickets_by_over = cricket_df.groupby('over')['is_wicket'].sum().reset_index()
print("\nWicket distribution by over:")
print(wickets_by_over)

# For datasets with wickets, analyze dismissal types
if wicket_count > 0:
    wicket_data = cricket_df[cricket_df['is_wicket'] == 1]

    # Wicket distribution by dismissal type
    wicket_types = wicket_data['dismissal_kind'].value_counts()
    print("\nWicket types distribution:")
    print(wicket_types)

    # Visualize wicket distribution
    plt.figure(figsize=(10, 6))
    plt.bar(wickets_by_over['over'], wickets_by_over['is_wicket'])
    plt.xlabel('Over')
    plt.ylabel('Number of Wickets')
    plt.title('Wicket Distribution by Over')
    plt.grid(axis='y')
    plt.show()
```

```
else:
    print("\nNo wickets in the sample data to analyze dismissal types.")
```

```
Total wickets in dataset: 12950

Wicket distribution by over:
    over  is_wicket
0      0        463
1      1        503
2      2        533
3      3        548
4      4        576
5      5        540
6      6        444
7      7        472
8      8        537
9      9        498
10    10        568
11    11        567
12    12        585
13    13        633
14    14        695
15    15        736
16    16        779
17    17        976
18    18        989
19    19       1308

Wicket types distribution:
dismissal_kind
caught                   8063
bowled                   2212
run out                  1114
lbw                       800
caught and bowled         367
stumped                   358
retired hurt               15
hit wicket                 15
obstructing the field       3
retired out                 3
Name: count, dtype: int64
```



Step 8: Partnership Analysis

```
# Partnership Analysis
import pandas as pd

def analyze_partnerships(df):
    """
    Analyze batting partnerships in cricket match data

    Args:
        df: DataFrame with ball-by-ball cricket data
```

```
        Returns:
            DataFrame with partnership details
        """
        # Check if we have enough data
        if len(df) <= 1:
            print("Not enough data to identify partnerships")
            return pd.DataFrame()

        # Sort data by match, inning, over and ball
        df = df.sort_values(['match_id', 'inning', 'over', 'ball'])

        partnerships = []
        matches = df['match_id'].unique()

        for match in matches:
            match_data = df[df['match_id'] == match]
            innings = match_data['inning'].unique()

            for inning in innings:
                inning_data = match_data[match_data['inning'] == inning]

                # Track current batters and partnership runs
                partnership_start_idx = 0
                current_batters = [inning_data.iloc[0]['batter'], inning_data.iloc[0]['non_striker']]
                partnership_runs = 0

                for i, ball in inning_data.iterrows():
                    # Add runs to current partnership
                    partnership_runs += ball['total_runs']

                    # Check if wicket fell
                    if ball['is_wicket'] == 1 and ball['player_dismissed'] in current_batters:
                        # Record partnership
                        partnership_info = {
                            'match_id': match,
                            'inning': inning,
                            'batters': sorted(current_batters),
                            'partnership_runs': partnership_runs,
                            'balls': i - partnership_start_idx + 1
                        }
                        partnerships.append(partnership_info)

                        # Update batters for next partnership
                        dismissed = ball['player_dismissed']
                        current_batters.remove(dismissed)

                        # Try to find next batter
                        next_ball_idx = i + 1
                        if next_ball_idx < len(inning_data):
                            next_ball = inning_data.iloc[next_ball_idx]
                            new_batters = [next_ball['batter'], next_ball['non_striker']]
                            new_batter = [b for b in new_batters if b not in current_batters]
                            if new_batter:
                                current_batters.append(new_batter[0])

                        # Reset for next partnership
                        partnership_start_idx = i + 1
                        partnership_runs = 0

                # Record final partnership if it has runs
                if partnership_runs > 0:
                    partnership_info = {
                        'match_id': match,
                        'inning': inning,
                        'batters': sorted(current_batters),
                        'partnership_runs': partnership_runs,
                        'balls': len(inning_data) - partnership_start_idx
                    }
                    partnerships.append(partnership_info)

        # Create DataFrame from partnerships list
        if partnerships:
            partnerships_df = pd.DataFrame(partnerships)
            partnerships_df['run_rate'] = (partnerships_df['partnership_runs'] /
                                           partnerships_df['balls']) * 6
            return partnerships_df
        else:
            return pd.DataFrame()

    # Apply partnership analysis
    partnerships_df = analyze_partnerships(cricket_df)
```

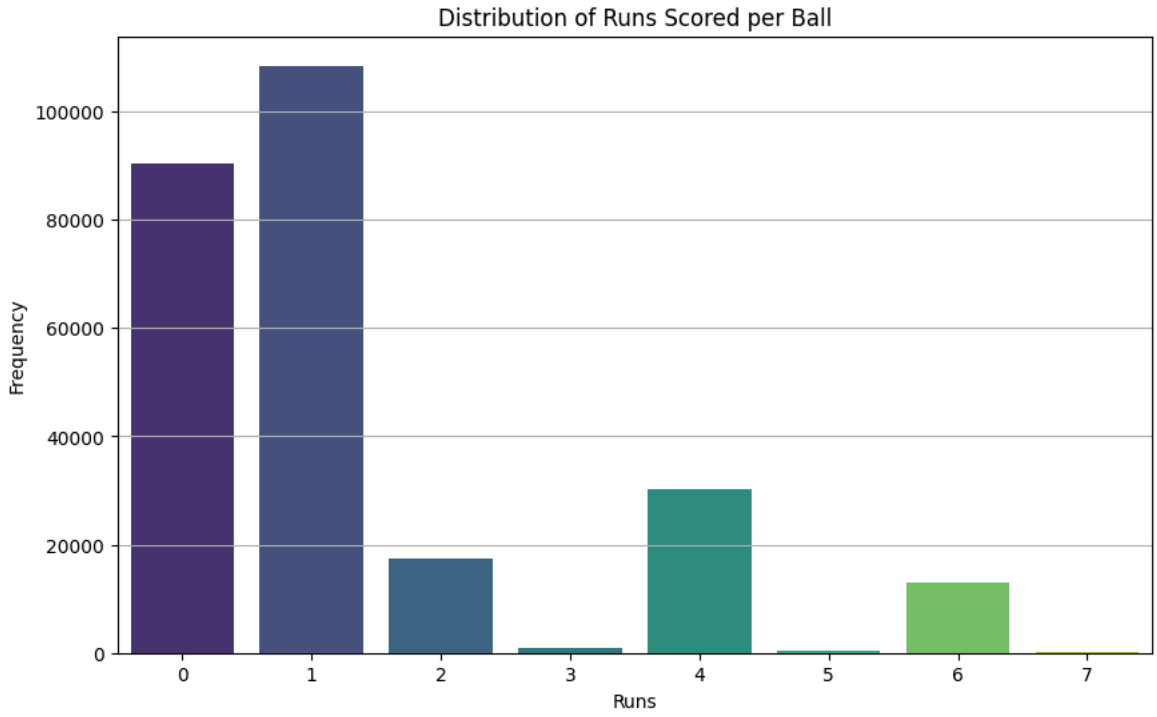
```
if not partnerships_df.empty:
    print("Top Partnerships:")
    print(partnerships_df.sort_values('partnership_runs', ascending=False).head(10))
else:
    print("No partnership data available from the sample.")
```

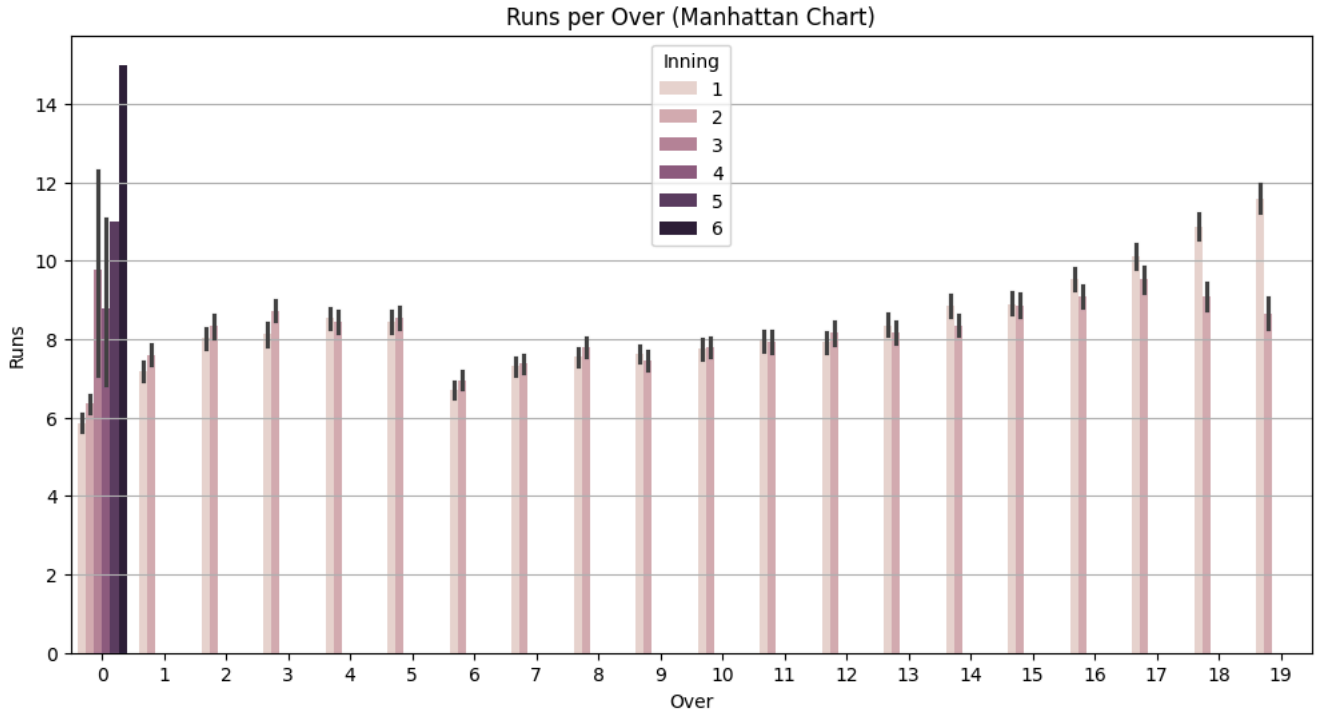
```
Top Partnerships:
      match_id  inning                         batters  partnership_runs  \
3220    980987       1                      [V Kohli]               229
2899    829795       1                      [V Kohli]               215
6237   1426297       1  [B Sai Sudharsan, Shubman Gill]               210
5425   1304112       1           [KL Rahul, Q de Kock]               210
1899    598006       1                             []               208
5335   1304096       1                    [DA Warner]               207
1360    501260       1                 [AC Gilchrist]               206
6123   1426277       1                   [RD Gaikwad]               206
1793    548372       1                     [CH Gayle]               204
6080   1426269       2                   [JC Buttler]               202

       balls   run_rate
3220      98  14.020408
2899 -119133  -0.010828
6237  257960   0.004884
5425     122  10.327869
1899  -78182  -0.015963
5335 -219821  -0.005650
1360     100  12.360000
6123 -252959  -0.004886
1793  -73821  -0.016581
6080 -251179  -0.004825
```

Step 9: Visualization of Key Insights

```
# Run Distribution Analysis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Analyze runs per ball
runs_distribution = cricket_df['total_runs'].value_counts().sort_index()
print("Runs distribution:")
print(runs_distribution)

# Visualize runs distribution
plt.figure(figsize=(10, 6))
sns.countplot(x='total_runs', data=cricket_df, palette='viridis')
plt.title('Distribution of Runs Scored per Ball')
plt.xlabel('Runs')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()

# Analyze runs by over
runs_by_over = cricket_df.groupby(['match_id', 'inning', 'over'])['total_runs'].sum().reset_index()

# Create Manhattan chart for run rate by over
plt.figure(figsize=(12, 6))
sns.barplot(x='over', y='total_runs', hue='inning', data=runs_by_over)
plt.title('Runs per Over (Manhattan Chart)')
plt.xlabel('Over')
plt.ylabel('Runs')
plt.legend(title='Inning')
plt.grid(axis='y')
plt.show()

# Calculate dot ball percentage
total_balls = len(cricket_df)
dot_balls = len(cricket_df[cricket_df['total_runs'] == 0])
dot_percentage = (dot_balls / total_balls) * 100
print(f"\nDot ball percentage: {dot_percentage:.2f}%")

# Calculate boundary percentage
boundaries = len(cricket_df[cricket_df['batsman_runs'].isin([4, 6])])
boundary_percentage = (boundaries / total_balls) * 100
print(f"Boundary percentage: {boundary_percentage:.2f}%")
```

```
Runs distribution:
total_runs
0     90438
1    108440
2     17323
3       922
4     30221
5       524
6     12964
7        88
Name: count, dtype: int64
<ipython-input-30-38a37dbaec1b>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(x='total_runs', data=cricket_df, palette='viridis')
```






```
Dot ball percentage: 34.66%
Boundary percentage: 16.44%
```

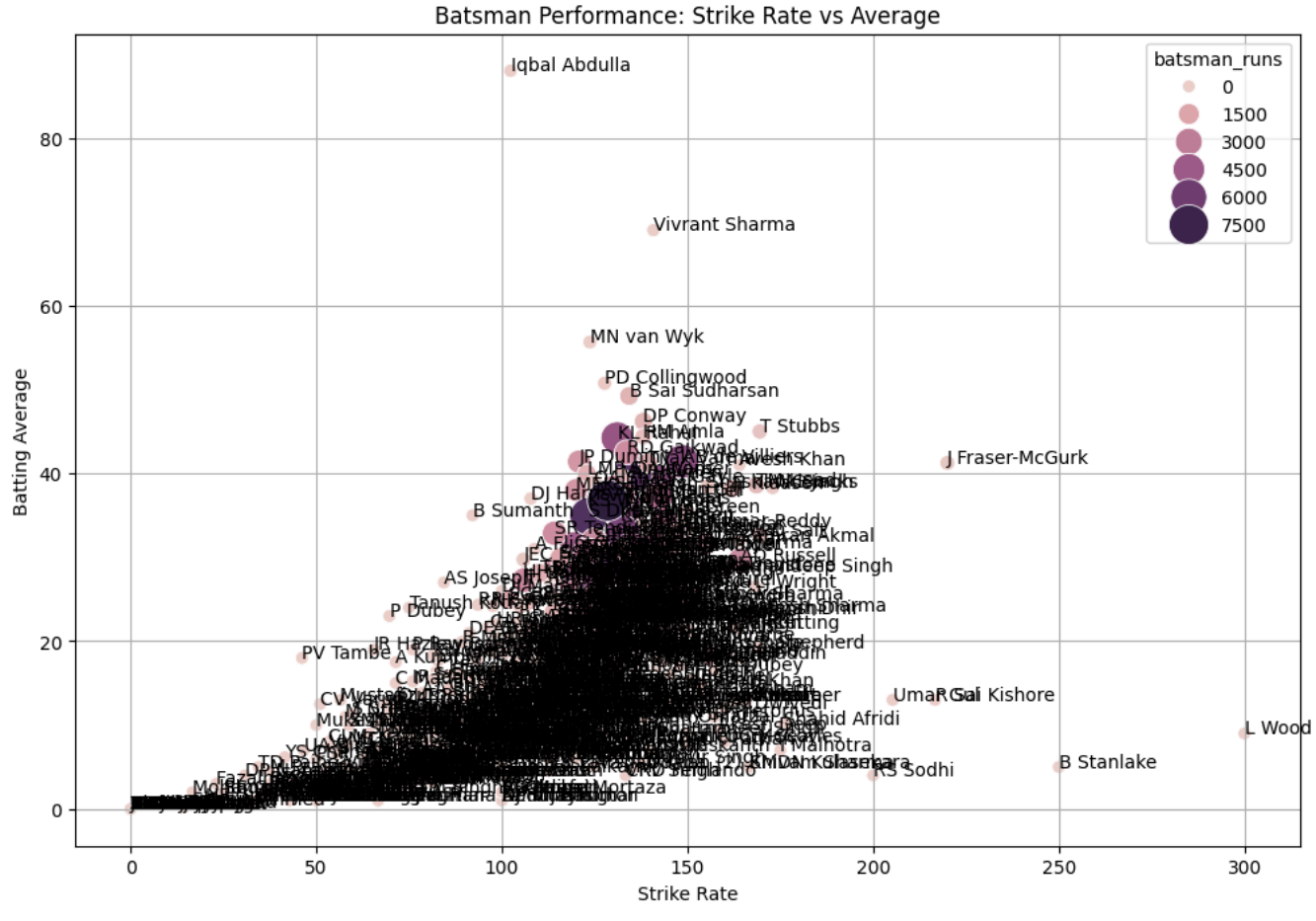
Step 10: Advanced Analytics & Insights

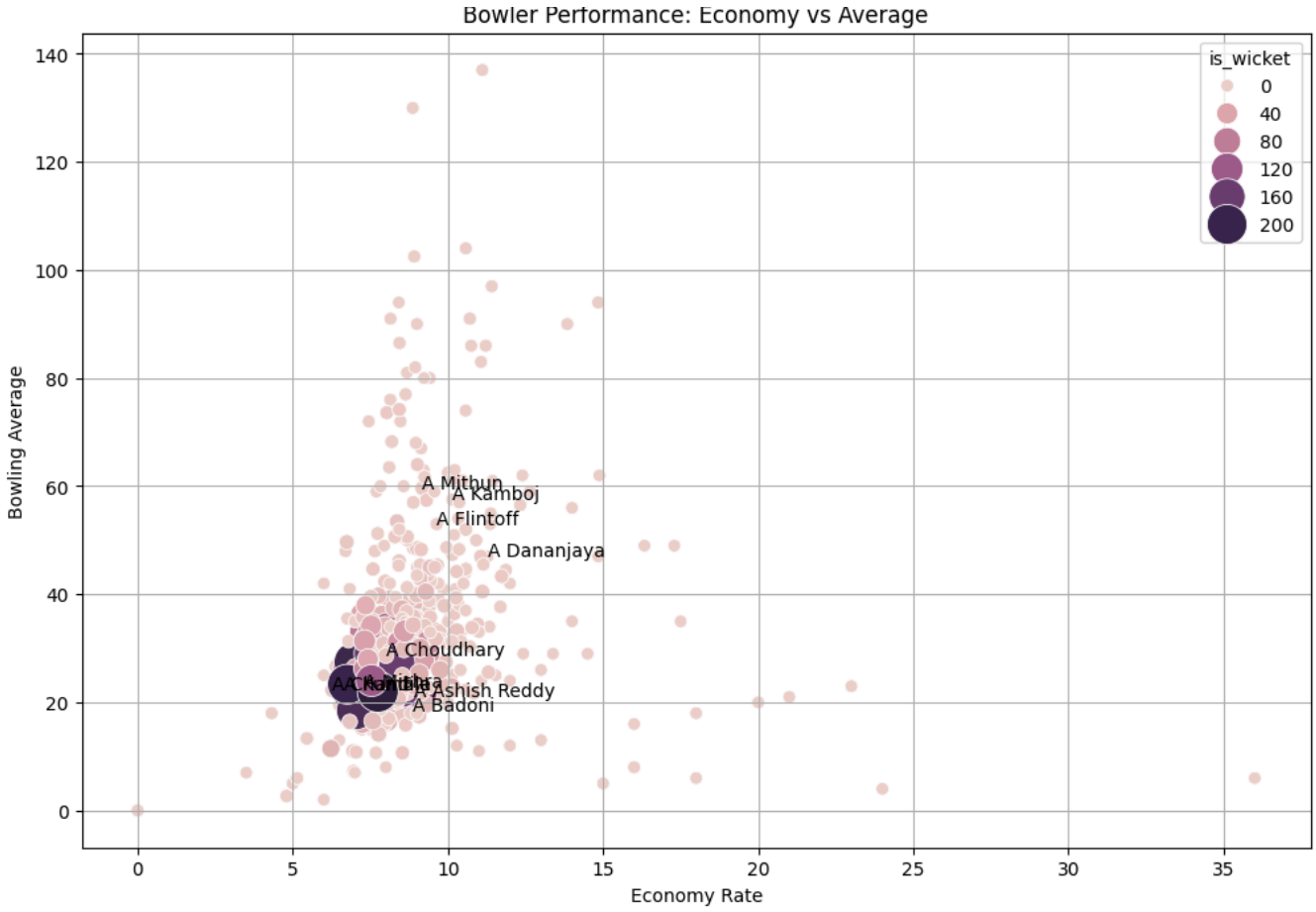
Player Performance Analysis

```
# Player Performance Analysis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Batsman analysis
batsmen = cricket_df.groupby('batter').agg({
    'batsman_runs': 'sum',
    'is_wicket': 'sum',
    'match_id': 'nunique',  # count unique matches
}).reset_index()

# Calculate balls faced per batsman
balls_faced = cricket_df.groupby('batter').size().reset_index(name='balls_faced')
batsmen = pd.merge(batsmen, balls_faced, on='batter')

# Calculate batting statistics
batsmen['batting_avg'] = batsmen['batsman_runs'] / batsmen['is_wicket'].replace(0, 1)
batsmen['strike_rate'] = (batsmen['batsman_runs'] / batsmen['balls_faced']) * 100
batsmen['matches'] = batsmen['match_id']

# Count boundaries
fours = cricket_df[cricket_df['batsman_runs'] == 4].groupby('batter').size().reset_index(name='fours')
sixes = cricket_df[cricket_df['batsman_runs'] == 6].groupby('batter').size().reset_index(name='sixes')

# Merge boundary counts
batsmen = pd.merge(batsmen, fours, on='batter', how='left')
batsmen = pd.merge(batsmen, sixes, on='batter', how='left')
batsmen['fours'] = batsmen['fours'].fillna(0)
batsmen['sixes'] = batsmen['sixes'].fillna(0)

print("Top Batsmen by Runs:")
print(batsmen.sort_values('batsman_runs', ascending=False).head(10))

# Bowler analysis
bowlers = cricket_df.groupby('bowler').agg({
    'total_runs': 'sum',
    'is_wicket': 'sum',
    'match_id': 'nunique',  # count unique matches
}).reset_index()

# Calculate balls bowled per bowler
balls_bowled = cricket_df.groupby('bowler').size().reset_index(name='balls_bowled')
bowlers = pd.merge(bowlers, balls_bowled, on='bowler')

# Calculate bowling statistics
bowlers['overs'] = bowlers['balls_bowled'] / 6
bowlers['economy'] = bowlers['total_runs'] / bowlers['overs']
bowlers['bowling_avg'] = bowlers['total_runs'] / bowlers['is_wicket'].replace(0, 1)
bowlers['bowling_sr'] = bowlers['balls_bowled'] / bowlers['is_wicket'].replace(0, 1)
bowlers['matches'] = bowlers['match_id']

print("\nTop Bowlers by Wickets:")
print(bowlers.sort_values('is_wicket', ascending=False).head(10))

# Visualize player performance
if len(batsmen) > 1:
    plt.figure(figsize=(12, 8))
    sns.scatterplot(x='strike_rate', y='batting_avg', size='batsman_runs',
                    hue='batsman_runs', data=batsmen, sizes=(50, 500))
    plt.title('Batsman Performance: Strike Rate vs Average')
    plt.xlabel('Strike Rate')
    plt.ylabel('Batting Average')
    for i, row in batsmen.iterrows():
        plt.annotate(row['batter'], (row['strike_rate'], row['batting_avg']))
    plt.grid(True)
    plt.show()

if len(bowlers) > 1:
    plt.figure(figsize=(12, 8))
    sns.scatterplot(x='economy', y='bowling_avg', size='is_wicket',
                    hue='is_wicket', data=bowlers, sizes=(50, 500))
    plt.title('Bowler Performance: Economy vs Average')
    plt.xlabel('Economy Rate')
    plt.ylabel('Bowling Average')
    for i, row in bowlers.head(10).iterrows():
        plt.annotate(row['bowler'], (row['economy'], row['bowling_avg']))
```

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

```
Top Batsmen by Runs:
            batter  batsman_runs  is_wicket  match_id  balls_faced  \
631         V Kohli          8014        218       244         6236
512        S Dhawan          6769        194       221         5483
477       RG Sharma          6630        232       251         5183
147       DA Warner          6567        164       184         4849
546        SK Raina          5536        168       200         4177
374        MS Dhoni          5243        149       228         3947
30   AB de Villiers          5181        125       170         3487
124        CH Gayle          4997        128       141         3516
501      RV Uthappa          4954        184       197         3927
282      KD Karthik          4843        189       233         3687

     batting_avg  strike_rate  matches  fours  sixes
631    36.761468   128.511867      244  708.0  273.0
512    34.891753   123.454313      221  768.0  153.0
477    28.577586   127.918194      251  599.0  281.0
147    40.042683   135.429986      184  663.0  236.0
546    32.952381   132.535312      200  506.0  204.0
374    35.187919   132.835065      228  363.0  252.0
30     41.448000   148.580442      170  414.0  253.0
124    39.039062   142.121729      141  408.0  359.0
501    26.923913   126.152279      197  481.0  182.0
282    25.624339   131.353404      233  466.0  161.0

Top Bowlers by Wickets:
          bowler  total_runs  is_wicket  match_id  balls_bowled       overs  \
524   YS Chahal        4681        213       159          3628  604.666667
119    DJ Bravo        4436        207       158          3296  549.333333
348   PP Chawla        5179        201       191          3895  649.166667
446   SP Narine        4672        200       175          4146  691.000000
355    R Ashwin        5435        198       208          4679  779.833333
71      B Kumar        5051        195       176          4060  676.666667
438  SL Malinga        3486        188       122          2974  495.666667
8      A Mishra        4193        183       162          3444  574.000000
193   JJ Bumrah        3840        182       133          3185  530.833333
373   RA Jadeja        4917        169       211          3895  649.166667

      economy  bowling_avg  bowling_sr  matches
524  7.741455    21.976526   17.032864      159
119  8.075243    21.429952   15.922705      158
348  7.977920    25.766169   19.378109      191
446  6.761216    23.360000   20.730000      175
355  6.969438    27.449495   23.631313      208
71   7.464532    25.902564   20.820513      176
438  7.032952    18.542553   15.819149      122
8    7.304878    22.912568   18.819672      162
193  7.233909    21.098901   17.500000      133
373  7.574326    29.094675   23.047337      211
```





Step 11: Match-up Analysis

```
# Match-up Analysis
import pandas as pd

# Analyze batsman vs bowler match-ups
matchups = cricket_df.groupby(['batter', 'bowler']).agg({
    'batsman_runs': 'sum',
    'is_wicket': 'sum',
    'match_id': 'nunique',  # count unique matches
}).reset_index()

# Calculate balls bowled in each match-up
balls_in_matchup = cricket_df.groupby(['batter', 'bowler']).size().reset_index(name='balls')
matchups = pd.merge(matchups, balls_in_matchup, on=['batter', 'bowler'])

# Calculate match-up statistics
matchups['strike_rate'] = (matchups['batsman_runs'] / matchups['balls']) * 100
matchups['dismissal_rate'] = matchups['is_wicket'] / matchups['balls']
matchups['dominance_ratio'] = matchups['batsman_runs'] / (matchups['is_wicket'] * 10 + 1)  # Higher value means batsman dominates

# Filter for significant match-ups (minimum balls faced)
significant_matchups = matchups[matchups['balls'] >= 6]  # At least 1 over

print("Significant Batsman-Bowler Match-ups:")
print(significant_matchups.sort_values('balls', ascending=False).head(10))

print("\nMatch-ups where Bowler Dominates:")
print(significant_matchups.sort_values('dominance_ratio').head(5))

print("\nMatch-ups where Batsman Dominates:")
print(significant_matchups.sort_values('dominance_ratio', ascending=False).head(5))
```

```
Significant Batsman-Bowler Match-ups:
          batter          bowler  batsman_runs  is_wicket  match_id  balls  \
26199    V Kohli       R Ashwin           179          1        22    153
26208    V Kohli      RA Jadeja           157          5        19    148
19788  RG Sharma      SP Narine           143          9        21    136
5481   DA Warner      SP Narine           195          2        16    127
11655   KL Rahul      JJ Bumrah           150          3        13    126
22427   SK Raina  Harbhajan Singh           132          5        19    125
21179   S Dhawan  Harbhajan Singh           147          4        15    123
19809  RG Sharma        UT Yadav           170          5        17    121
```

```
5438   DA Warner         R Ashwin           122          4        14    118
26240    V Kohli        SP Narine           127          4        16    118

       strike_rate  dismissal_rate  dominance_ratio
26199   116.993464        0.006536        16.272727
26208   106.081081        0.033784         3.078431
19788   105.147059        0.066176         1.571429
5481    153.543307        0.015748         9.285714
11655   119.047619        0.023810         4.838710
22427   105.600000        0.040000         2.588235
21179   119.512195        0.032520         3.585366
19809   140.495868        0.041322         3.333333
5438    103.389831        0.033898         2.975610
26240   107.627119        0.033898         3.097561

Match-ups where Bowler Dominates:
            batter      bowler  batsman_runs  is_wicket  match_id  balls  \
18669    R Rampaul    DJ Bravo             0          1         1      6
6629      DR Smith  M de Lange             0          0         1     13
3911       BR Dunk     B Kumar             0          0         1      7
15974  Mohsin Khan   CJ Jordan             0          0         1      6
22021   SC Ganguly  FH Edwards             0          0         1      8

       strike_rate  dismissal_rate  dominance_ratio
18669          0.0        0.166667              0.0
6629           0.0        0.000000              0.0
3911           0.0        0.000000              0.0
15974          0.0        0.000000              0.0
22021          0.0        0.000000              0.0

Match-ups where Batsman Dominates:
           batter      bowler  batsman_runs  is_wicket  match_id  balls  \
11627    KL Rahul   DL Chahar           158          0        11    103
11143  KD Karthik  JD Unadkat           124          0         9     64
5448    DA Warner   RA Jadeja           116          0         7     68
21878    SA Yadav    R Ashwin           115          0        13     82
16266      N Rana    R Ashwin           108          0         8     60

       strike_rate  dismissal_rate  dominance_ratio
11627   153.398058             0.0            158.0
11143   193.750000             0.0            124.0
5448    170.588235             0.0            116.0
21878   140.243902             0.0            115.0
16266   180.000000             0.0            108.0
```

Step 12: Team Performance Analysis

```
# Team Performance Analysis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Team batting performance
team_batting = cricket_df.groupby('batting_team').agg({
    'total_runs': 'sum',
    'is_wicket': 'sum',
    'match_id': 'nunique',  # count unique matches
}).reset_index()

# Calculate team batting statistics
team_balls = cricket_df.groupby('batting_team').size().reset_index(name='balls_faced')
team_batting = pd.merge(team_batting, team_balls, on='batting_team')

team_batting['run_rate'] = (team_batting['total_runs'] / team_balls['balls_faced']) * 6
team_batting['batting_avg'] = team_batting['total_runs'] / team_batting['is_wicket'].replace(0, 1)

# Count boundaries by team
team_fours = cricket_df[cricket_df['batsman_runs'] == 4].groupby('batting_team').size().reset_index(name='fours')
team_sixes = cricket_df[cricket_df['batsman_runs'] == 6].groupby('batting_team').size().reset_index(name='sixes')

# Merge boundary counts
team_batting = pd.merge(team_batting, team_fours, on='batting_team', how='left')
team_batting = pd.merge(team_batting, team_sixes, on='batting_team', how='left')
team_batting['fours'] = team_batting['fours'].fillna(0)
team_batting['sixes'] = team_batting['sixes'].fillna(0)
team_batting['boundary_percentage'] = ((team_batting['fours'] + team_batting['sixes']) / team_batting['balls_faced']) * 100

print("Team Batting Performance:")
print(team_batting.sort_values('run_rate', ascending=False))

# Team bowling performance
team_bowling = cricket_df.groupby('bowling_team').agg({
    'total_runs': 'sum',
    'is_wicket': 'sum'
```

```
        'is_wicket': 'sum',
        'match_id': 'nunique',  # count unique matches
    }).reset_index()

    # Calculate team bowling statistics
    team_balls_bowled = cricket_df.groupby('bowling_team').size().reset_index(name='balls_bowled')
    team_bowling = pd.merge(team_bowling, team_balls_bowled, on='bowling_team')

    team_bowling['economy_rate'] = (team_bowling['total_runs'] / team_bowling['balls_bowled']) * 6
    team_bowling['bowling_avg'] = team_bowling['total_runs'] / team_bowling['is_wicket'].replace(0, 1)
    team_bowling['bowling_sr'] = team_bowling['balls_bowled'] / team_bowling['is_wicket'].replace(0, 1)

    print("\nTeam Bowling Performance:")
    print(team_bowling.sort_values('economy_rate'))

    # Visualize team comparison
    if len(team_batting) > 1:
        plt.figure(figsize=(12, 6))
        sns.barplot(x='batting_team', y='run_rate', data=team_batting.sort_values('run_rate', ascending=False))
        plt.title('Team Run Rates')
        plt.xlabel('Team')
        plt.ylabel('Run Rate')
        plt.xticks(rotation=45, ha='right')
        plt.grid(axis='y')
        plt.tight_layout()
        plt.show()
```

```
Team Batting Performance:
                    batting_team  total_runs  is_wicket  match_id  balls_faced  \
17  Royal Challengers Bengaluru        2930         99        15         1818
5                Gujarat Titans        7757        247        45         5494
12                 Punjab Kings        9536        371        56         6833
9          Lucknow Super Giants        7510        276        44         5400
4                Gujarat Lions        4862        188        30         3566
2                Delhi Capitals       14900        570        91        10946
0           Chennai Super Kings       38629       1245       237        28651
10               Mumbai Indians       42176       1573       261        31437
16  Royal Challengers Bangalore       37692       1384       240        28205
8          Kolkata Knight Riders       39331       1491       251        29514
18           Sunrisers Hyderabad       29071       1058       182        21843
6                Kings XI Punjab       30064       1158       190        22646
13             Rajasthan Royals       34747       1312       220        26242
15      Rising Pune Supergiants        2063         68        14         1580
14       Rising Pune Supergiant        2470         90        16         1900
3               Delhi Daredevils       24296        952       161        18786
1                Deccan Chargers       11463        484        75         9034
7           Kochi Tuskers Kerala        1901         86        14         1582
11                Pune Warriors        6358        298        45         5443
```