Picture Credit : IGN India

In [1]:

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

In [2]:

```
df = pd.read_csv('innings_deliveries.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	team	over	batter	bowler	non_striker	runs_batter	runs_extras	runs_total	player_out	wicket_kind	fielders
0	Royal Challengers Bengaluru	0	V Kohli	I Sharma	F du Plessis	0	0	0	NaN	NaN	[]
1	Royal Challengers Bengaluru	0	V Kohli	I Sharma	F du Plessis	1	0	1	NaN	NaN	[]
2	Royal Challengers Bengaluru	0	F du Plessis	I Sharma	V Kohli	1	0	1	NaN	NaN	[]
3	Royal Challengers Bengaluru	0	V Kohli	I Sharma	F du Plessis	0	0	0	NaN	NaN	[]
4	Royal Challengers Bengaluru	0	V Kohli	I Sharma	F du Plessis	2	0	2	NaN	NaN	[]

In [4]:

```
df.tail()
```

Out[4]:

	team	over	batter	bowler	non_striker	runs_batter	runs_extras	runs_total	player_out	wicket_kind	fielders
242	Delhi Capitals	18	Kuldeep Yadav	Mohammed Siraj	I Sharma	1	0	1	NaN	NaN	[]
243	Delhi Capitals	18	I Sharma	Mohammed Siraj	Kuldeep Yadav	0	1	1	NaN	NaN	[]
244	Delhi Capitals	18	I Sharma	Mohammed Siraj	Kuldeep Yadav	0	0	0	NaN	NaN	[]
245	Delhi Capitals	18	I Sharma	Mohammed Siraj	Kuldeep Yadav	0	0	0	NaN	NaN	[]
246	Delhi Capitals	19	Kuldeep Yadav	Yash Dayal	I Sharma	0	0	0	Kuldeep Yadav	bowled	[]

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
team             0
over             0
batter           0
bowler           0
non_striker      0
runs_batter      0
runs_extras      0
runs_total       0
player_out     228
wicket_kind    228
fielders         0
dtype: int64
```

In [6]:

```
df.describe
```

Out[6]:

```
<bound method NDFrame.describe of                             team  over         batter          bowler
0    Royal Challengers Bengaluru     0        V Kohli        I Sharma  \
1    Royal Challengers Bengaluru     0        V Kohli        I Sharma
2    Royal Challengers Bengaluru     0   F du Plessis        I Sharma
3    Royal Challengers Bengaluru     0        V Kohli        I Sharma
4    Royal Challengers Bengaluru     0        V Kohli        I Sharma
..                           ...   ...            ...             ...
242               Delhi Capitals    18  Kuldeep Yadav  Mohammed Siraj
243               Delhi Capitals    18       I Sharma  Mohammed Siraj
244               Delhi Capitals    18       I Sharma  Mohammed Siraj
245               Delhi Capitals    18       I Sharma  Mohammed Siraj
246               Delhi Capitals    19  Kuldeep Yadav      Yash Dayal

       non_striker  runs_batter  runs_extras  runs_total     player_out
0     F du Plessis            0            0           0            NaN  \
1     F du Plessis            1            0           1            NaN
2          V Kohli            1            0           1            NaN
3     F du Plessis            0            0           0            NaN
4     F du Plessis            2            0           2            NaN
..             ...          ...          ...         ...            ...
242       I Sharma            1            0           1            NaN
243  Kuldeep Yadav            0            1           1            NaN
244  Kuldeep Yadav            0            0           0            NaN
245  Kuldeep Yadav            0            0           0            NaN
246       I Sharma            0            0           0  Kuldeep Yadav

    wicket_kind fielders
0           NaN       []
1           NaN       []
2           NaN       []
3           NaN       []
4           NaN       []
..          ...      ...
242         NaN       []
243         NaN       []
244         NaN       []
245         NaN       []
246      bowled       []

[247 rows x 11 columns]>
```

In [8]:

```
df.dtypes
```

Out[8]:

```
team           object
over            int64
batter         object
bowler         object
non_striker    object
runs_batter     int64
runs_extras     int64
runs_total      int64
player_out     object
wicket_kind    object
fielders       object
dtype: object
```

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 247 entries, 0 to 246
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   team         247 non-null    object
 1   over         247 non-null    int64 
 2   batter       247 non-null    object
 3   bowler       247 non-null    object
 4   non_striker  247 non-null    object
 5   runs_batter  247 non-null    int64 
 6   runs_extras  247 non-null    int64 
 7   runs_total   247 non-null    int64 
 8   player_out   19 non-null     object
 9   wicket_kind  19 non-null     object
 10  fielders     247 non-null    object
dtypes: int64(4), object(7)
memory usage: 21.4+ KB
```

In [10]:

```python
import warnings
warnings.filterwarnings('ignore')
```

In [11]:

```python
df.nunique()
```

Out[11]:

```
team            2
over           20
batter         20
bowler         13
non_striker    20
runs_batter     5
runs_extras     2
runs_total      5
player_out     19
wicket_kind     4
fielders       15
dtype: int64
```

In [12]:

```python
df['team'].unique()
```

Out[12]:

```
array(['Royal Challengers Bengaluru', 'Delhi Capitals'], dtype=object)
```

In [13]:

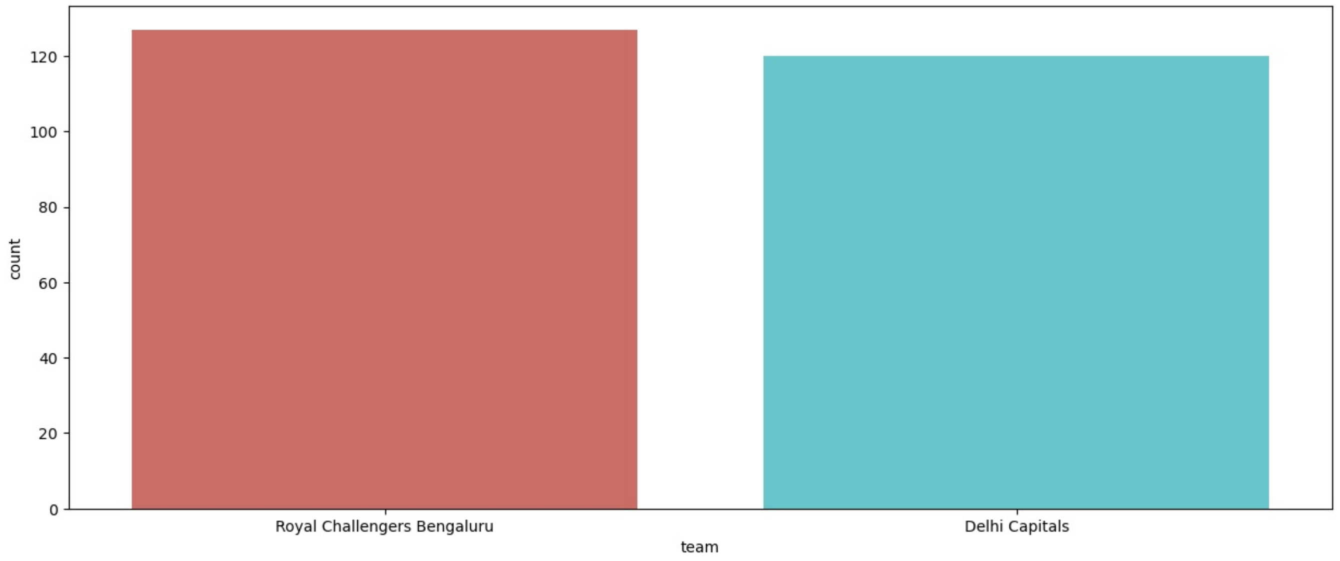
```python
df['team'].value_counts()
```

Out[13]:

```
team
Royal Challengers Bengaluru    127
Delhi Capitals                 120
Name: count, dtype: int64
```

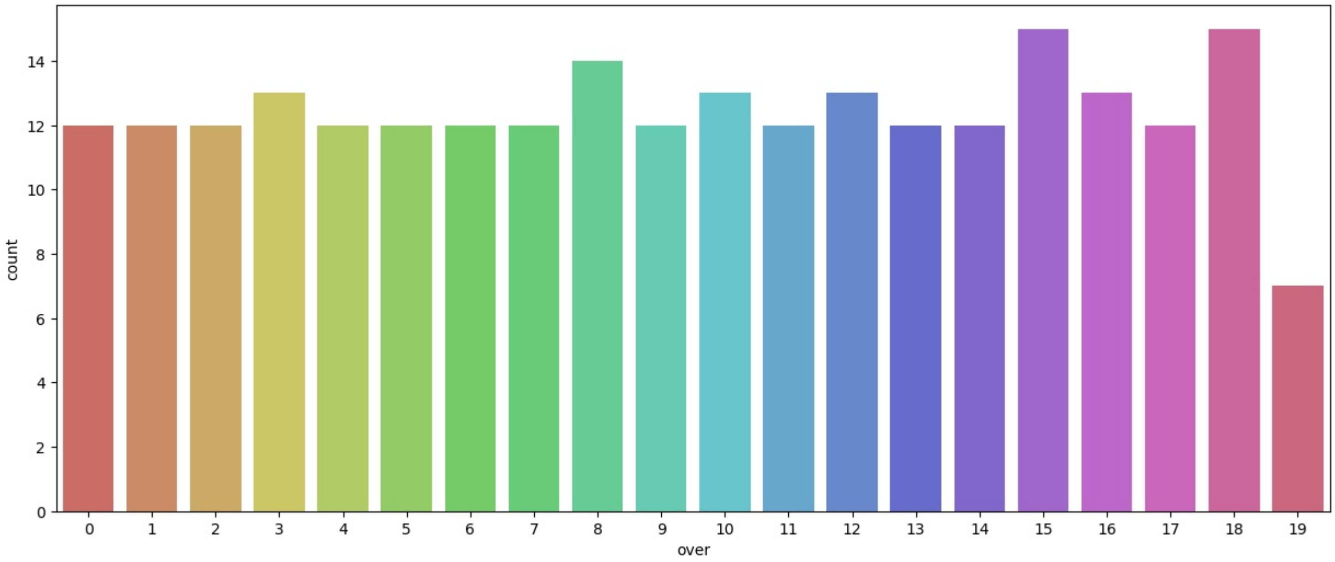
In [14]:

```python
plt.figure(figsize=(15, 6))
sns.countplot(x='team', data=df, palette='hls')
plt.show()
```



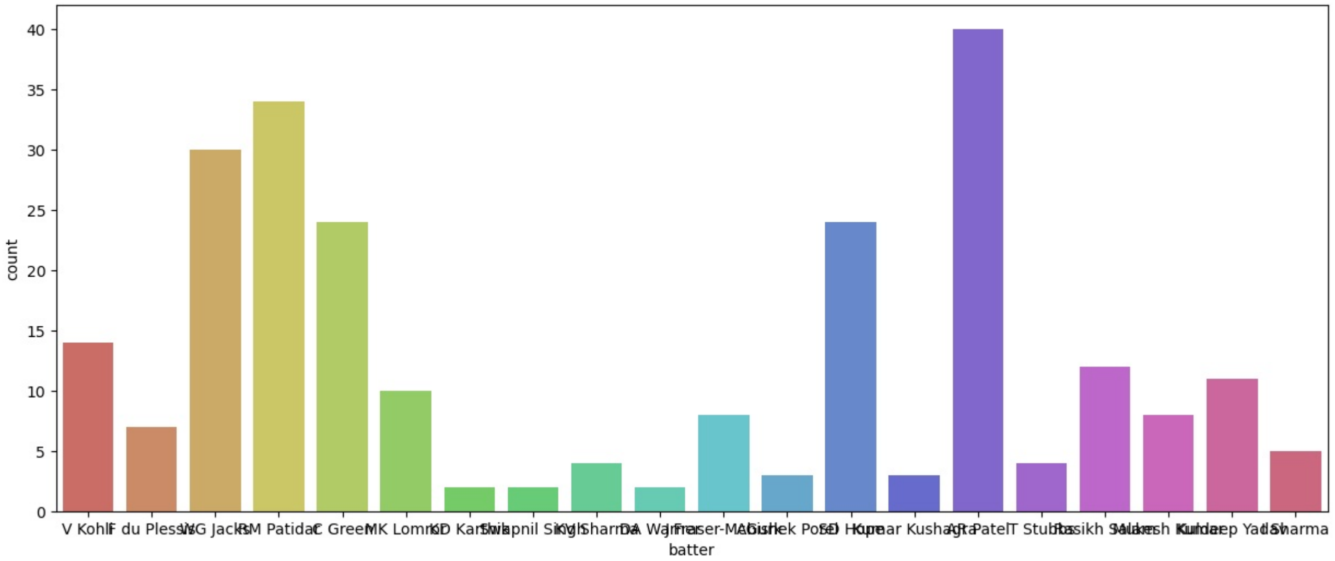
In [15]:

```python
plt.figure(figsize=(15, 6))
sns.countplot(x='over', data=df, palette='hls')
plt.show()
```



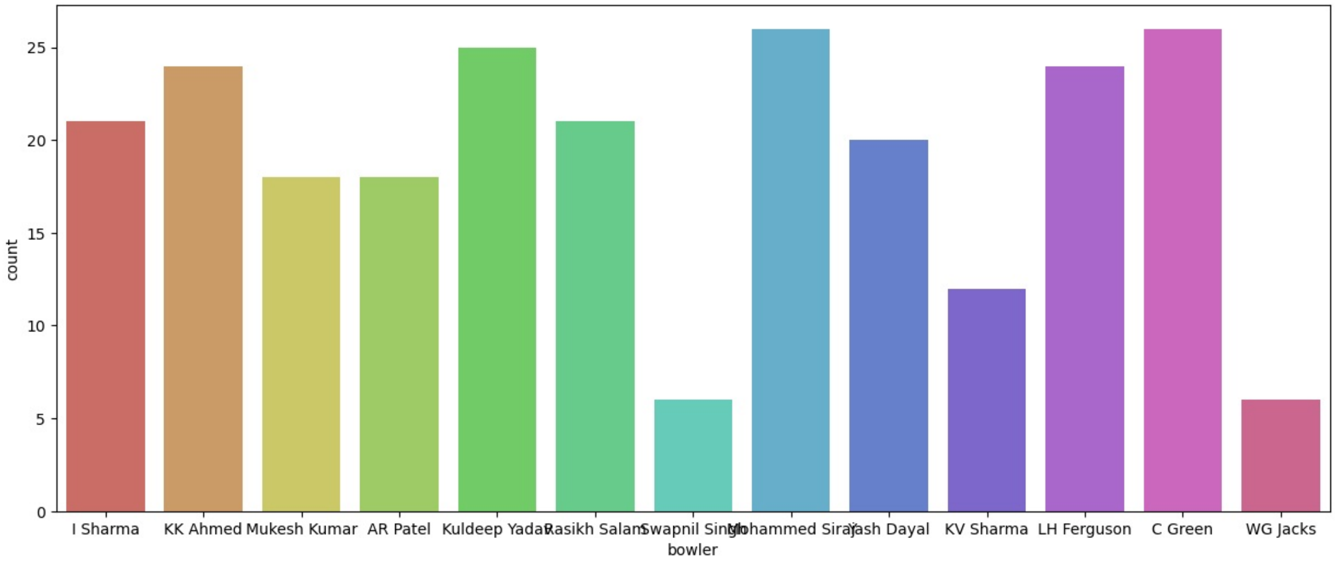
In [16]:

```
plt.figure(figsize=(15, 6))
sns.countplot(x='batter', data=df, palette='hls')
plt.show()
```



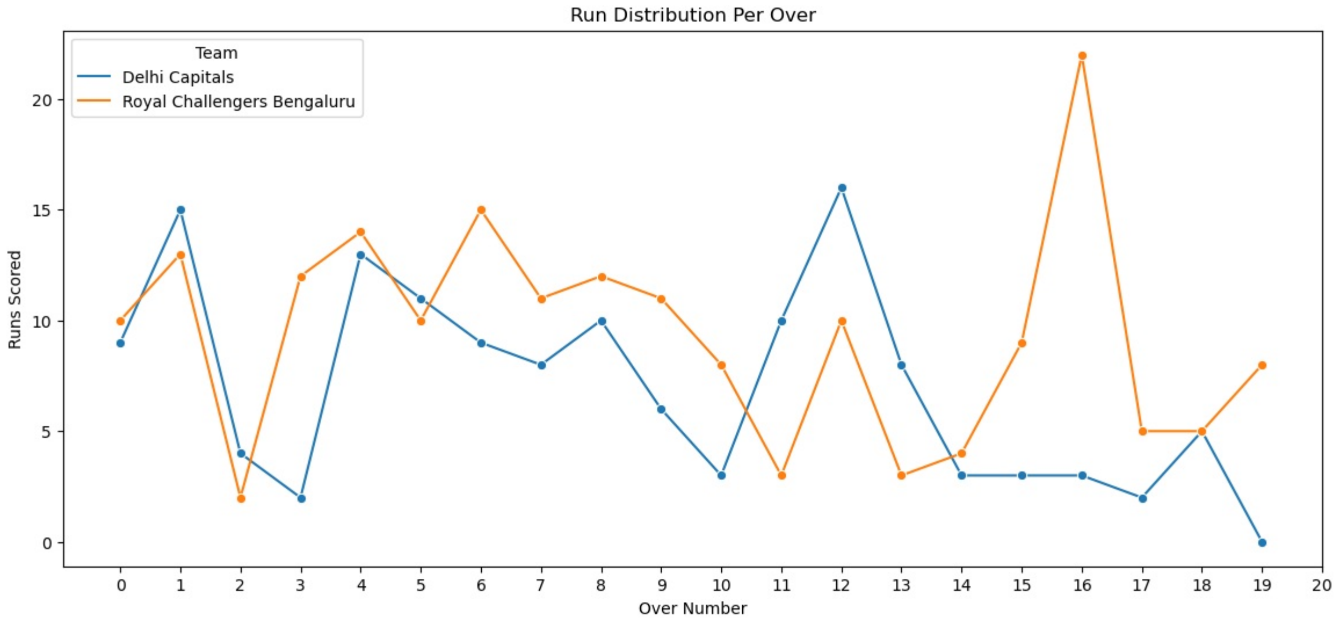
In [17]:

```
plt.figure(figsize=(15, 6))
sns.countplot(x='bowler', data=df, palette='hls')
plt.show()
```



In [19]:

```
# data preparation for run distribution per over
run_distribution =df.groupby(['team', 'over']).agg({'runs_total': 'sum'}).reset_index()

# plotting run distribution per over for both teams
plt.figure(figsize=(14, 6))
sns.lineplot(data=run_distribution, x='over', y='runs_total', hue='team', marker='o')
plt.title('Run Distribution Per Over')
plt.xlabel('Over Number')
plt.ylabel('Runs Scored')
plt.xticks(range(0, 21))  # over numbers from 0 to 20
plt.legend(title='Team')
plt.show()
```



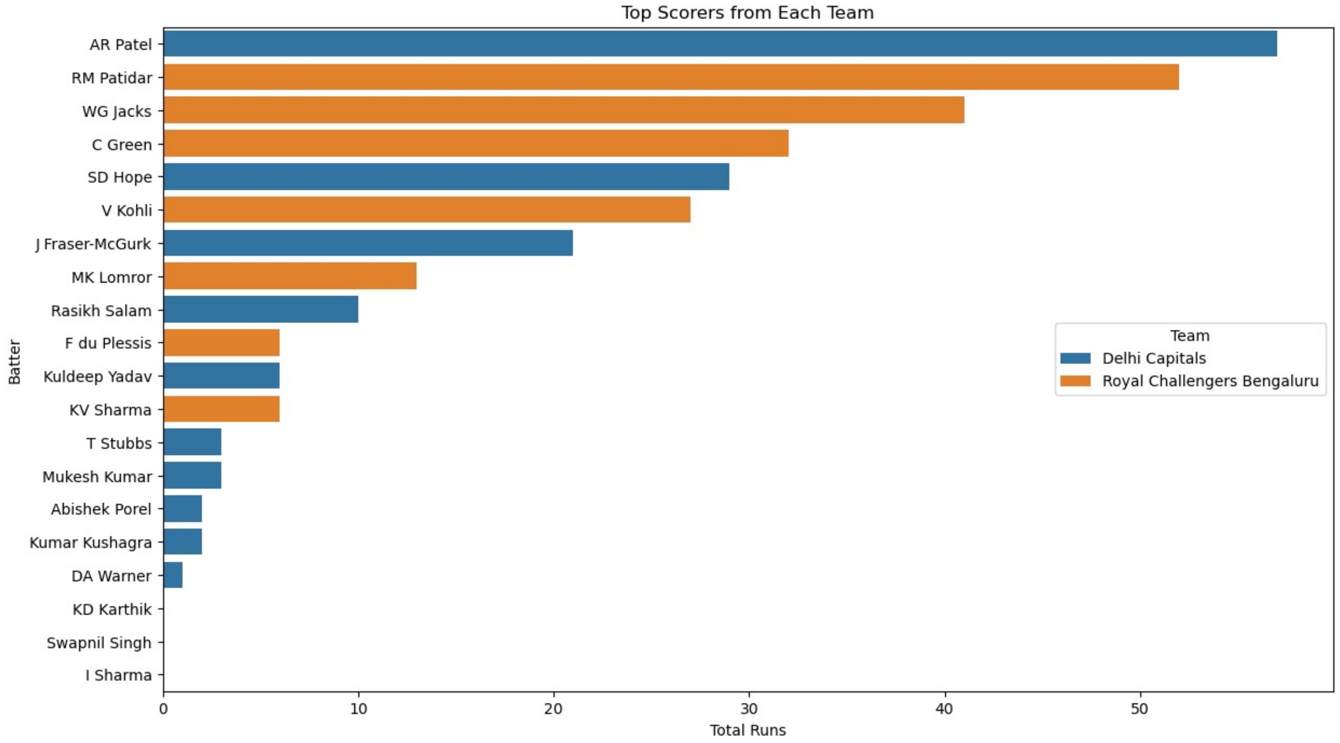
The plot above shows the run distribution per over for both teams. Here are some insights:

The scoring rate for each team shows fluctuations throughout their innings, with spikes indicating overs with high scoring, likely due to boundaries or big hits.

Royal Challengers Bangalore (RCB) appears to have a couple of overs with significantly higher runs, suggesting aggressive batting.

In [20]:

```
# calculating top scorers for each team
top_scorers = df.groupby(['team', 'batter']).agg({'runs_batter': 'sum'}).reset_index().sort_values(by='runs_bat

plt.figure(figsize=(14, 8))
sns.barplot(data=top_scorers, x='runs_batter', y='batter', hue='team', dodge=False)
plt.title('Top Scorers from Each Team')
plt.xlabel('Total Runs')
plt.ylabel('Batter')
plt.legend(title='Team', loc='center right')
plt.show()
```



Key observations from the graph include:

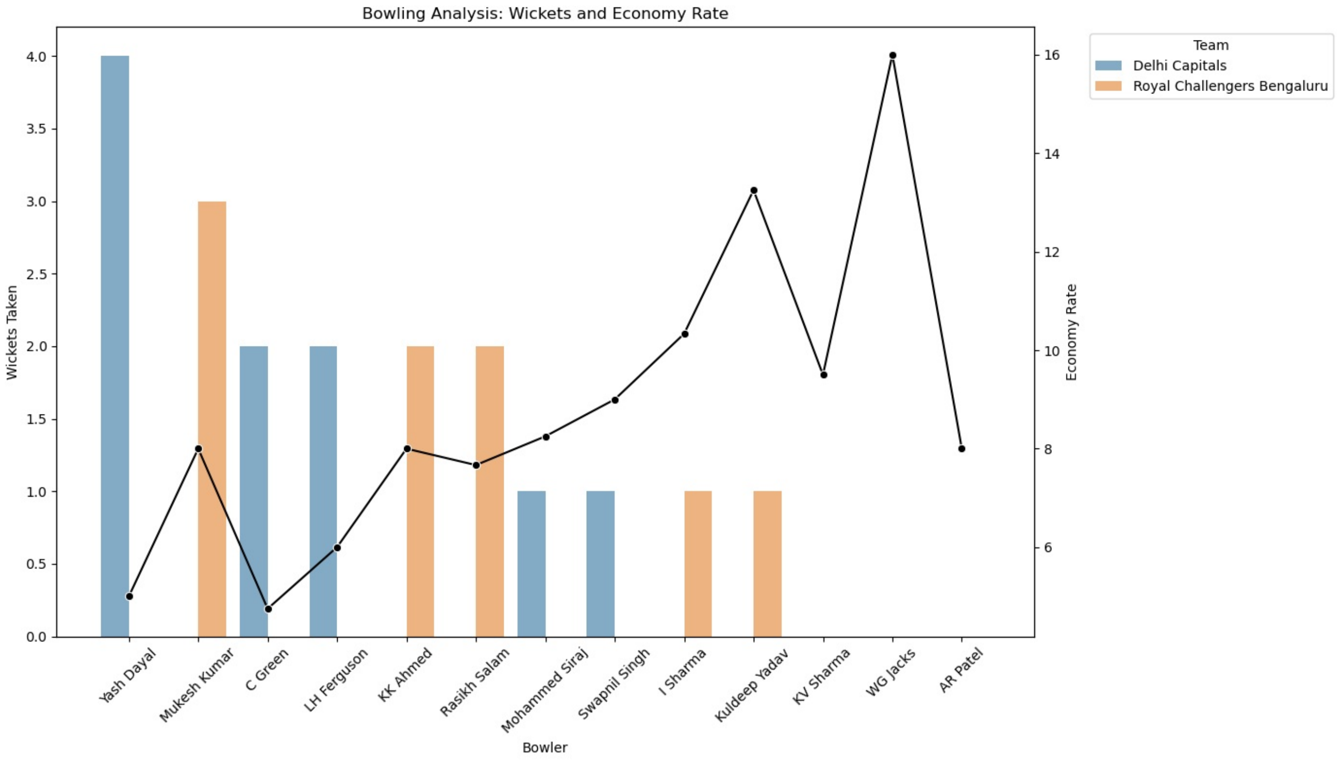
AR Patel from Delhi Capitals is the top scorer of the match, significantly outscoring others with a little over 50 runs.

RM Patidar is the top scorer for Royal Challengers Bangalore, closely approaching 50 runs.

The graph displays a diverse contribution from both teams, with several players from both sides contributing notable scores.

In [22]:

```
# preparing data for bowling analysis
df['wickets_taken'] = df['wicket_kind'].notna().astype(int)
bowling_stats = df.groupby(['team', 'bowler']).agg({'runs_total': 'sum', 'wickets_taken': 'sum', 'over': 'nuniq

# calculating economy rate (total runs conceded / number of overs bowled)
bowling_stats['economy_rate'] = bowling_stats['runs_total'] / bowling_stats['over']

# sorting the data for better visualization
bowling_stats_sorted = bowling_stats.sort_values(by='wickets_taken', ascending=False)

# prepare the DataFrame for plotting
bowling_stats_sorted['wickets_taken'] = df['wicket_kind'].notna().astype(int)
bowling_stats = df.groupby(['team', 'bowler']).agg({'runs_total': 'sum', 'wickets_taken': 'sum', 'over': 'nuniq
bowling_stats['economy_rate'] = bowling_stats['runs_total'] / bowling_stats['over']
bowling_stats_sorted = bowling_stats.sort_values(by='wickets_taken', ascending=False)

# create the plot
fig, ax1 = plt.subplots(figsize=(14, 8))

# Bar plot for wickets
sns.barplot(data=bowling_stats_sorted, x='bowler', y='wickets_taken', hue='team', ax=ax1, alpha=0.6)
ax1.set_ylabel('Wickets Taken')
ax1.set_xlabel('Bowler')
ax1.set_title('Bowling Analysis: Wickets and Economy Rate')
ax1.legend(title='Team', bbox_to_anchor=(1.05, 1), loc='upper left')

for item in ax1.get_xticklabels():
    item.set_rotation(45)

ax2 = ax1.twinx()
sns.lineplot(data=bowling_stats_sorted, x='bowler', y='economy_rate', marker='o', sort=False, ax=ax2, color='bl
ax2.set_ylabel('Economy Rate')

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



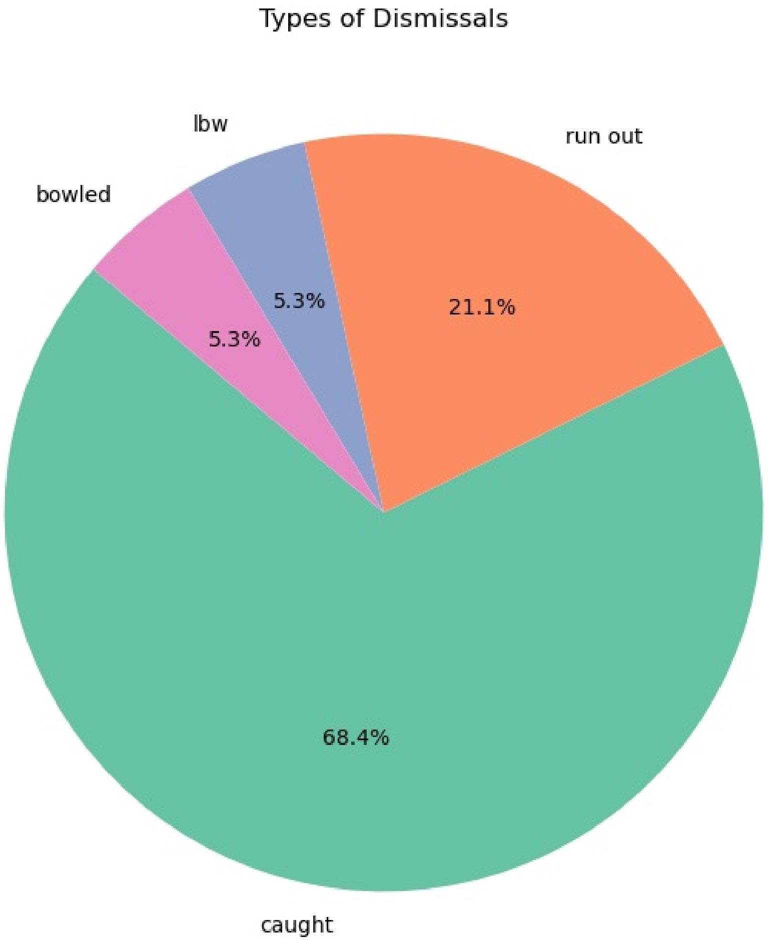
The combined bar and line plot provides a comprehensive overview of the bowling performance of each team:

Wickets Taken: The bars indicate the number of wickets each bowler took during the match. The height of the bars reflects how successful the bowlers were in terms of taking wickets. Bowlers from both teams contributed to taking wickets, with some notable performances that stand out due to higher bars.

Economy Rate: The line graph overlaid on the bar graph shows the economy rate (number of runs conceded per over) of each bowler. The economy rate is crucial as it indicates how economically a bowler has bowled in terms of runs given away.

In [23]:

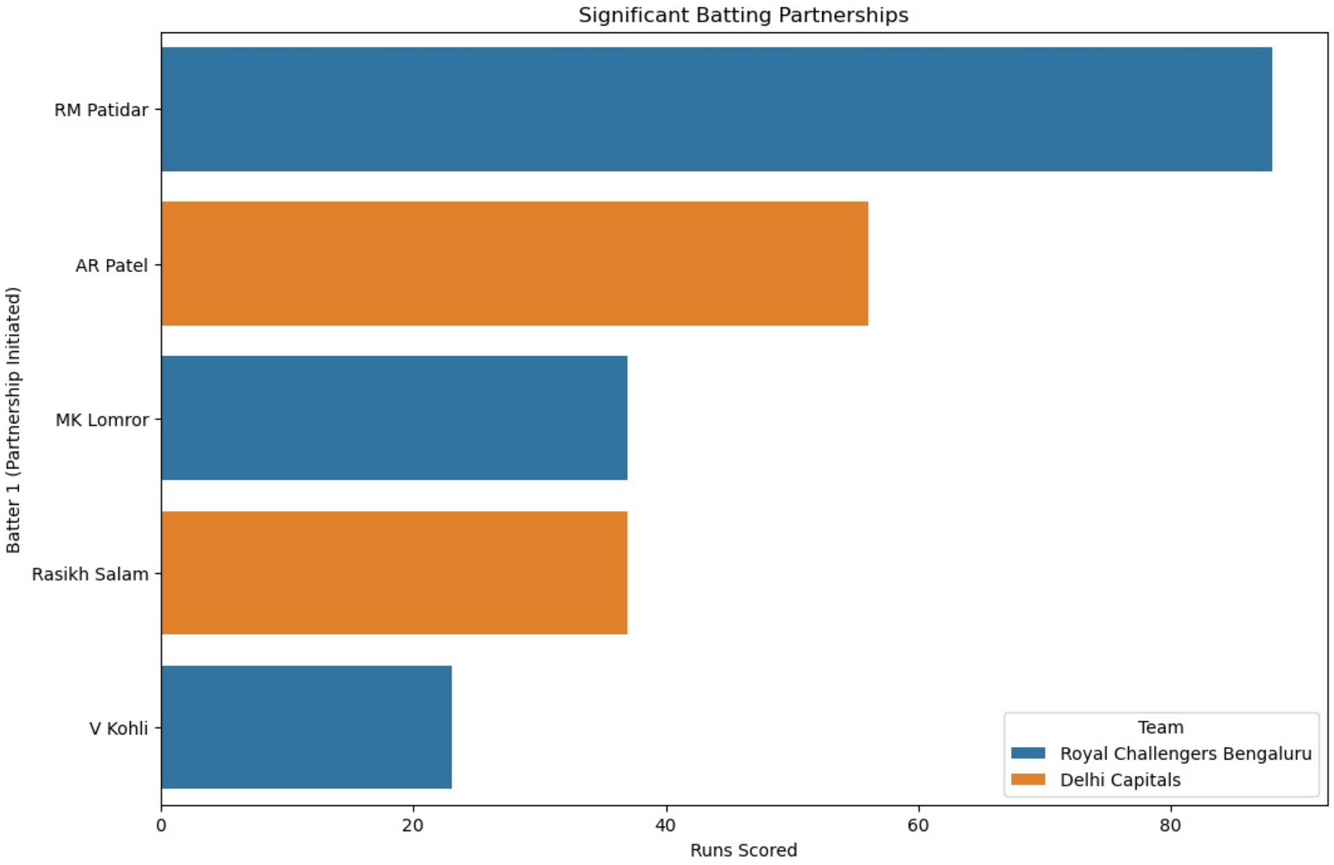
```
# counting dismissal types
dismissal_types = df['wicket_kind'].dropna().value_counts()

plt.figure(figsize=(8, 8))
plt.pie(dismissal_types, labels=dismissal_types.index, autopct='%1.1f%%', startangle=140, colors=sns.color_pale
plt.title('Types of Dismissals')
plt.show()
```



In [25]:

```
# function to calculate partnerships
def calculate_partnerships(df):
    partnerships = []
    current_partnership = {}
    for i, row in df.iterrows():
        if i == 0 or (row['batter'] not in current_partnership.values()):
            if current_partnership:
                partnerships.append(current_partnership)
            current_partnership = {
                'team': row['team'],
                'batter1': row['batter'],
                'batter2': row['non_striker'],
                'runs': 0,
                'balls': 0
            }
        current_partnership['runs'] += row['runs_total']
        current_partnership['balls'] += 1
        if 'player_out' in row and pd.notna(row['player_out']):
            if row['player_out'] == current_partnership['batter1'] or row['player_out'] == current_partnership[
                partnerships.append(current_partnership)
                current_partnership = {}
    # append the last partnership if not ended by a wicket
    if current_partnership:
        partnerships.append(current_partnership)
    return partnerships

# calculate partnerships
partnerships_data = calculate_partnerships(df)
partnerships_df = pd.DataFrame(partnerships_data)

# filter out significant partnerships (e.g., partnerships with more than 20 runs)
significant_partnerships = partnerships_df[partnerships_df['runs'] > 20]
```

```
# sort by highest runs
significant_partnerships = significant_partnerships.sort_values(by='runs', ascending=False)

plt.figure(figsize=(12, 8))
sns.barplot(data=significant_partnerships, x='runs', y='batter1', hue='team', dodge=False)
plt.title('Significant Batting Partnerships')
plt.xlabel('Runs Scored')
plt.ylabel('Batter 1 (Partnership Initiated)')
plt.legend(title='Team')
plt.show()
```



The bar chart displays significant batting partnerships from the match, highlighting partnerships that scored more than 20 runs. Here’s how these insights contribute to our analysis:

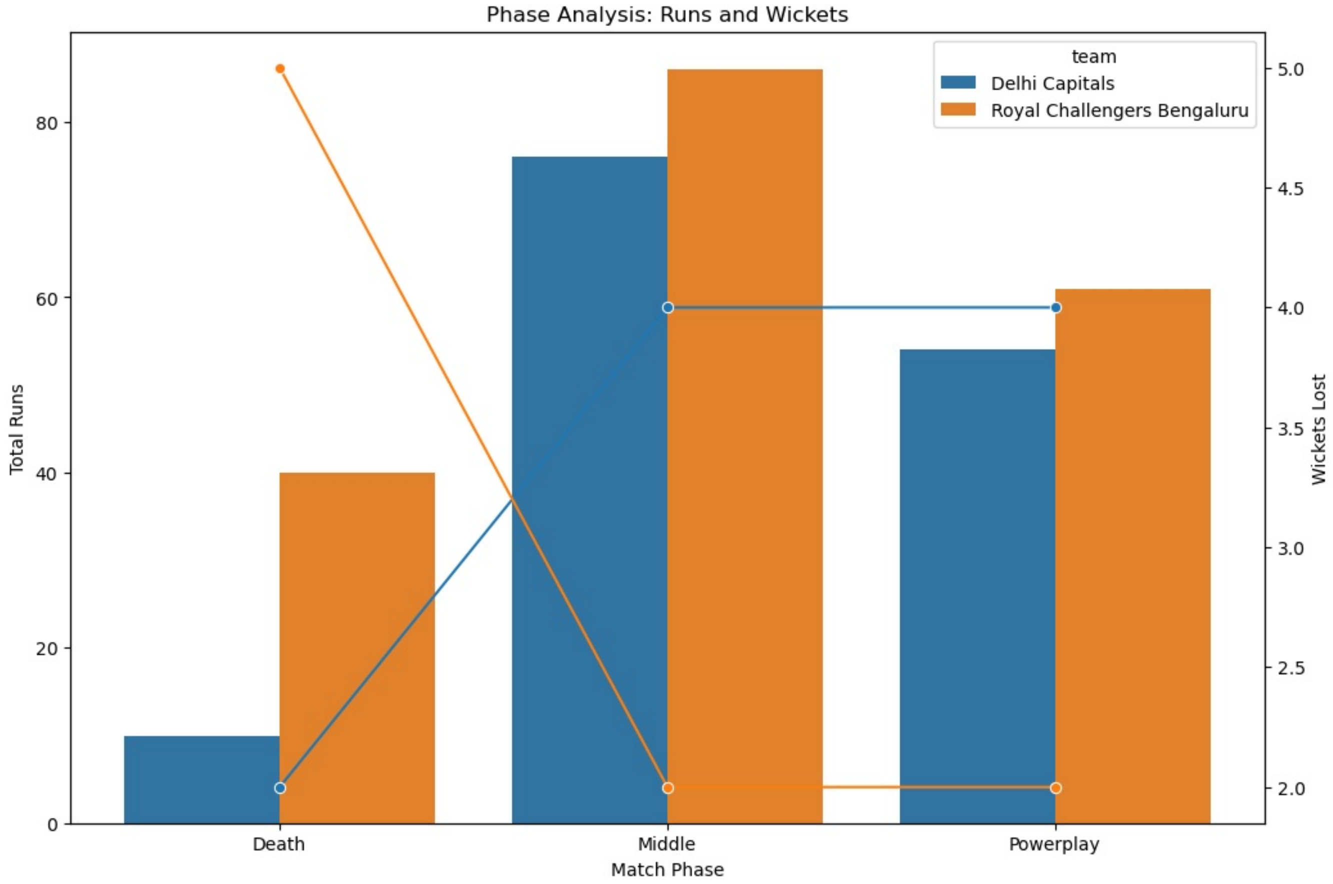
The chart identifies key partnerships that likely had a substantial impact on the match’s outcome, illustrating the effectiveness of batting pairs.

It provides insights into which players were involved in pivotal stands, which can help in assessing player form and team strategy.

In [27]:

```
# function to classify the phase of the game based on the over number
def classify_phase(over):
    if over < 6:
        return 'Powerplay'
    elif over < 16:
        return 'Middle'
    else:
        return 'Death'

# adding phase information to the dataframe
df['phase'] = df['over'].apply(classify_phase)

# grouping data by phase and team to calculate runs and wickets
phase_analysis = df.groupby(['team', 'phase']).agg({'runs_total': 'sum', 'wickets_taken': 'sum', 'over': 'count

# calculating the run rate
phase_analysis['run_rate'] = (phase_analysis['runs_total'] / phase_analysis['balls']) * 6

# plotting the phase analysis
fig, ax1 = plt.subplots(figsize=(12, 8))

# bar plot for runs scored in each phase
sns.barplot(data=phase_analysis, x='phase', y='runs_total', hue='team', ax=ax1)
ax1.set_title('Phase Analysis: Runs and Wickets')
ax1.set_ylabel('Total Runs')
ax1.set_xlabel('Match Phase')
```

```
# line plot for wickets lost
ax2 = ax1.twinx()
sns.lineplot(data=phase_analysis, x='phase', y='wickets_taken', hue='team', marker='o', ax=ax2, legend=False)
ax2.set_ylabel('Wickets Lost')

plt.show()
```



The plot above provides a clear breakdown of the match into different phases; Powerplay, Middle, and Death, and illustrates how each team performed during these segments:

Powerplay: Both teams have a relatively low total of runs, with RCB losing more wickets than DC in this phase, as indicated by the height of the orange line.

Middle: This phase shows the highest run-scoring for both teams, with DC scoring slightly more than RCB. The wickets lost remain controlled, suggesting stable innings from both teams.

Death: RCB has a sharp decrease in runs compared to the Middle phase, while DC maintains a high run rate. Wickets lost by RCB increased significantly in this phase, marked by the orange line peaking near 4.5, indicating a possible collapse or aggressive batting that did not pay off.

In [28]:

```
# calculate runs and balls faced for each batter
batter_stats = df.groupby('batter').agg({'runs_batter': 'sum', 'over': 'count'}).rename(columns={'over': 'balls_

# calculate strike rate for each batter (runs per 100 balls)
batter_stats['strike_rate'] = (batter_stats['runs_batter'] / batter_stats['balls_faced']) * 100

# sorting batters by their strike rate
batter_stats_sorted = batter_stats.sort_values(by='strike_rate', ascending=False)

# displaying calculated strike rates along with runs scored and balls faced
batter_stats_sorted.head(10)
```

Out[28]:

	batter	runs_batter	balls_faced	strike_rate
6	J Fraser-McGurk	21	8	262.500000
18	V Kohli	27	14	192.857143
13	RM Patidar	52	34	152.941176
8	KV Sharma	6	4	150.000000
0	AR Patel	57	40	142.500000
19	WG Jacks	41	30	136.666667
2	C Green	32	24	133.333333
11	MK Lomror	13	10	130.000000
15	SD Hope	29	24	120.833333
4	F du Plessis	6	7	85.714286

Here are the top performers in terms of strike rate from the match:

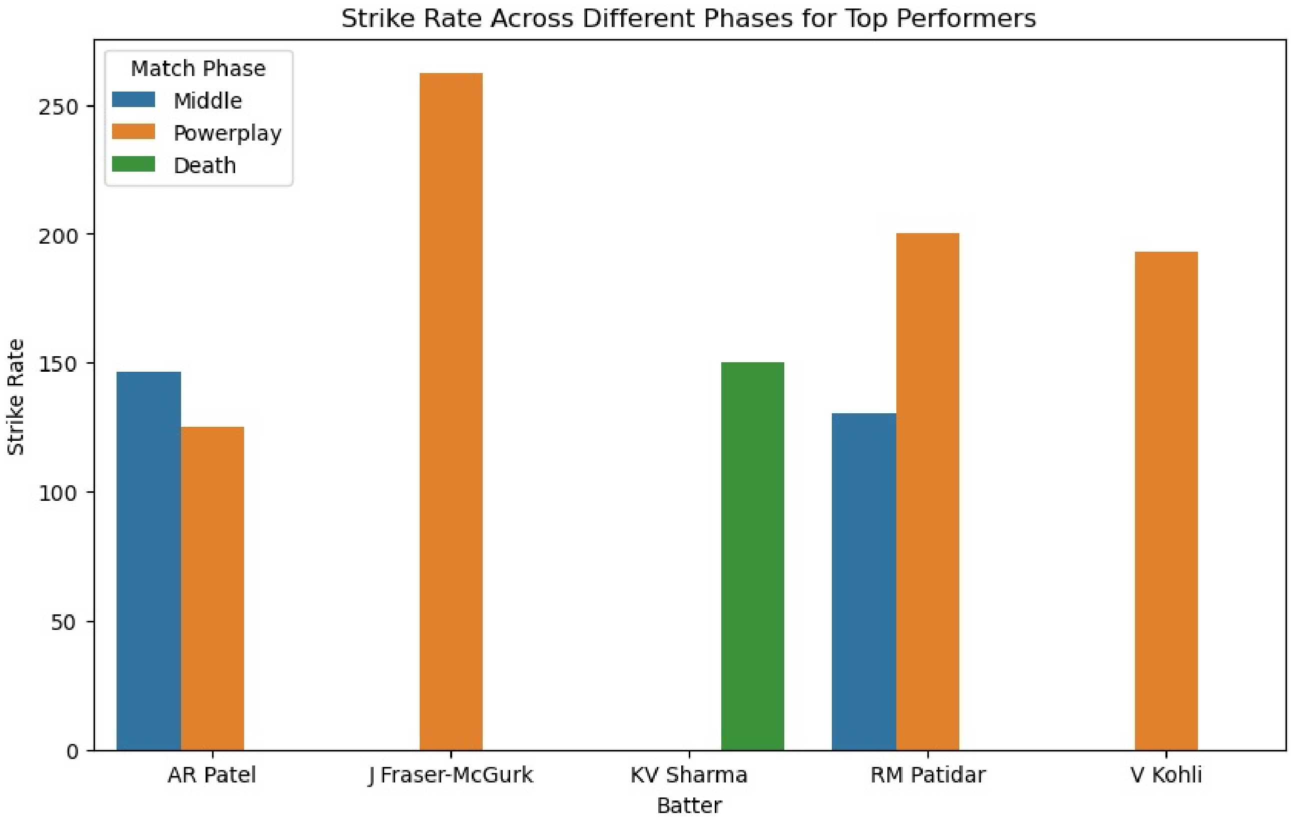
1.J Fraser-McGurk had the highest strike rate at 262.50, scoring 21 runs from just 8 balls.

2.Virat Kohli also scored efficiently, with a strike rate of 192.86, making 27 runs from 14 balls.

3. Rajat Patidar contributed significantly with a strike rate of 152.94, accumulating 52 runs from 34 balls.

In [29]:

```
# merging phase information with batter stats
batter_phase_stats = df.groupby(['batter', 'phase']).agg({'runs_batter': 'sum', 'over': 'count'}).rename(column

# calculate strike rate for each batter-phase combination
batter_phase_stats['strike_rate'] = (batter_phase_stats['runs_batter'] / batter_phase_stats['balls_faced']) * 1

# filtering for top performers based on overall strike rate
top_performers = batter_stats_sorted.head(5)['batter']
batter_phase_stats_top = batter_phase_stats[batter_phase_stats['batter'].isin(top_performers)]

# plotting strike rate across different phases for top performers
plt.figure(figsize=(10, 6))
sns.barplot(data=batter_phase_stats_top, x='batter', y='strike_rate', hue='phase')
plt.title('Strike Rate Across Different Phases for Top Performers')
plt.xlabel('Batter')
plt.ylabel('Strike Rate')
plt.legend(title='Match Phase')
plt.show()
```



The bar chart illustrates how the strike rates of the top performers varied across different phases of the match:

J Fraser-McGurk stands out with a particularly high strike rate in the Middle phase, significantly higher than any other phase or player, suggesting a highly aggressive and effective batting performance during this part of the innings.

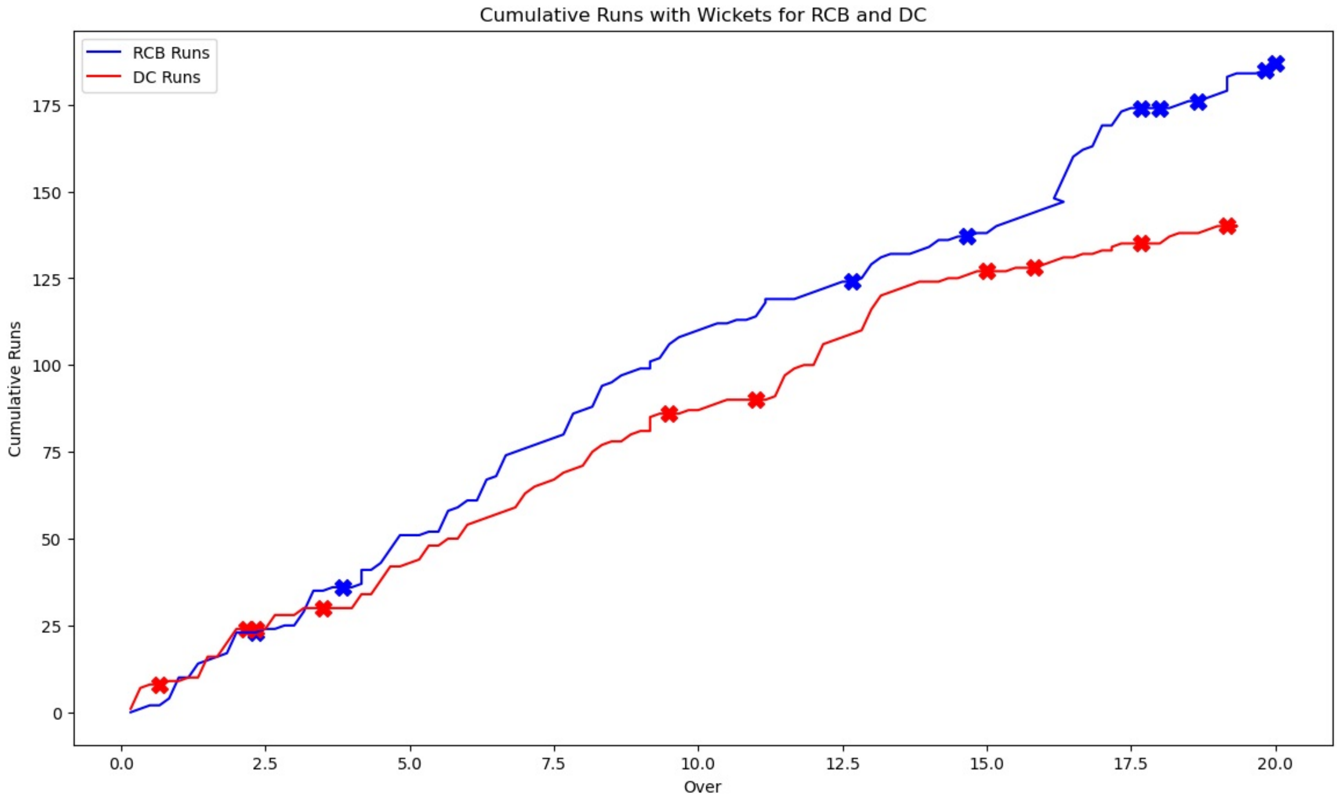
V Kohli and RM Patidar both have high strike rates in the Death phase, indicating their ability to accelerate scoring towards the end of the innings, which is crucial for setting or chasing targets.

AR Patel shows consistency in the Powerplay and Middle phases with a slightly reduced but still competitive strike rate, indicating his role as a steady opener or middle-order batter.

KV Sharma exhibits a lower strike rate in the Middle phase compared to others, suggesting a more conservative approach during this phase or difficulty in accelerating

In [33]:

```python
# calculate cumulative runs and wickets for each ball for both teams
df['cumulative_runs'] = df.groupby('team')['runs_total'].cumsum()
df['cumulative_wickets'] = df.groupby('team')['wickets_taken'].cumsum()

# separate data for both teams
rcb_deliveries = df[df['team'] == 'Royal Challengers Bengaluru']
dc_deliveries = df[df['team'] == 'Delhi Capitals']

# calculating overs for cumulative analysis
rcb_deliveries['over_ball'] = rcb_deliveries['over'] + (rcb_deliveries.groupby('over').cumcount() + 1) / 6
dc_deliveries['over_ball'] = dc_deliveries['over'] + (dc_deliveries.groupby('over').cumcount() + 1) / 6

# plotting cumulative run rates and wickets
fig, ax = plt.subplots(figsize=(14, 8))

# plot for RCB
ax.plot(rcb_deliveries['over_ball'], rcb_deliveries['cumulative_runs'], color='blue', label='RCB Runs')
ax.scatter(rcb_deliveries[rcb_deliveries['wickets_taken'] == 1]['over_ball'], rcb_deliveries[rcb_deliveries['wi

# plot for DC
ax.plot(dc_deliveries['over_ball'], dc_deliveries['cumulative_runs'], color='red', label='DC Runs')
ax.scatter(dc_deliveries[dc_deliveries['wickets_taken'] == 1]['over_ball'], dc_deliveries[dc_deliveries['wicket

ax.set_title('Cumulative Runs with Wickets for RCB and DC')
ax.set_xlabel('Over')
ax.set_ylabel('Cumulative Runs')
ax.legend()
plt.show()
```



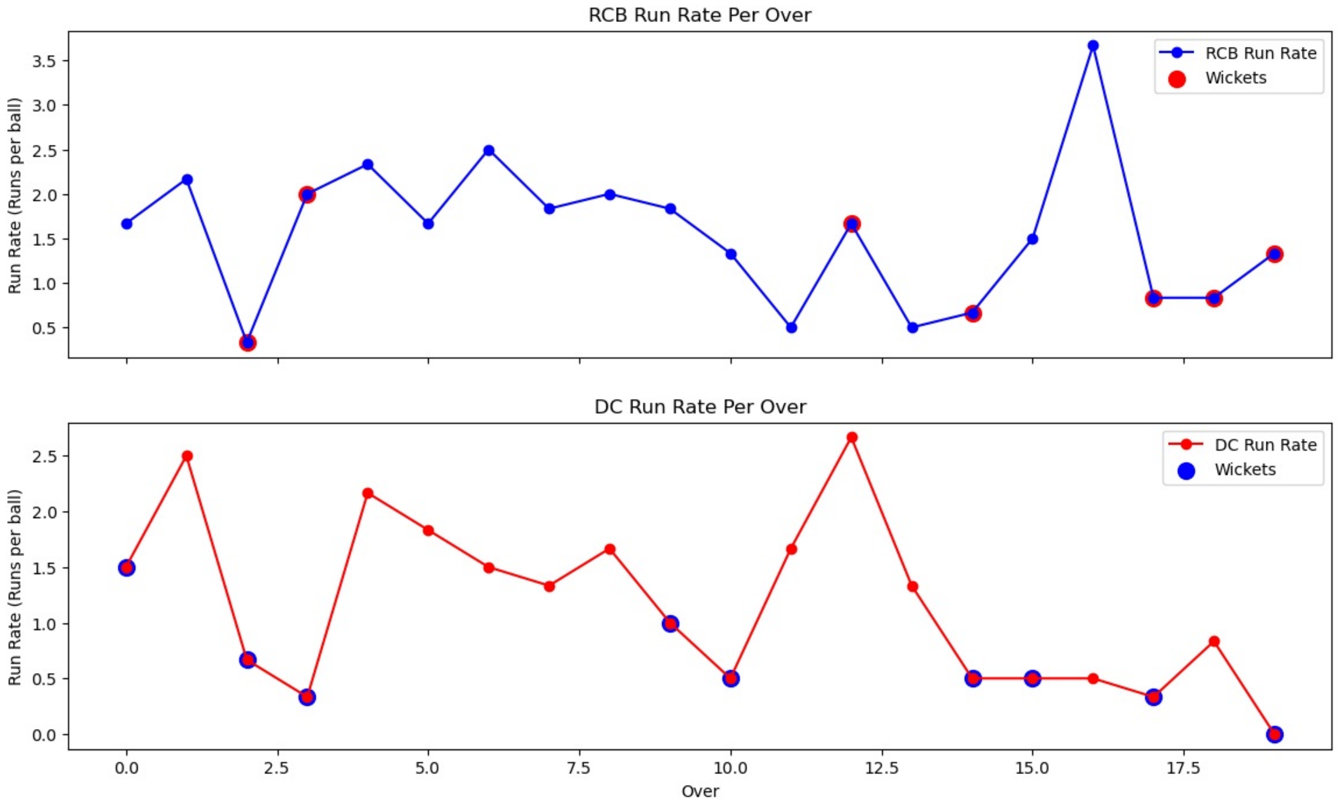
The plot shows the cumulative runs scored by each team throughout their innings, with markers indicating wickets:

Momentum Shifts: The points where wickets are lost are crucial. Despite wickets, RCB's run line does not show any drastic downturns, suggesting effective recovery by subsequent batters.

Performance Analysis: RCB's ability to keep the run rate up despite losing wickets might indicate deeper batting strength or successful innings pacing strategies. In contrast, DC, while also increasing their score, does so at a less steep rate, possibly indicating fewer big overs.

In [34]:

```
# calculate runs and wickets per over for both teams
per_over_stats = df.groupby(['team', 'over']).agg({'runs_total': 'sum', 'wickets_taken': 'sum'}).reset_index()

# calculate run rate for each over
per_over_stats['run_rate'] = (per_over_stats['runs_total'] / 6)    # Runs per over to runs per ball (standard r

# separate data for RCB and DC for plotting
rcb_per_over_stats = per_over_stats[per_over_stats['team'] == 'Royal Challengers Bengaluru']
dc_per_over_stats = per_over_stats[per_over_stats['team'] == 'Delhi Capitals']

# plotting run rates and marking wickets for each team
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

# RCB
ax1.plot(rcb_per_over_stats['over'], rcb_per_over_stats['run_rate'], marker='o', color='blue', label='RCB Run R
ax1.scatter(rcb_per_over_stats[rcb_per_over_stats['wickets_taken'] > 0]['over'], rcb_per_over_stats[rcb_per_ove
ax1.set_title('RCB Run Rate Per Over')
ax1.set_ylabel('Run Rate (Runs per ball)')
ax1.legend()

# DC
ax2.plot(dc_per_over_stats['over'], dc_per_over_stats['run_rate'], marker='o', color='red', label='DC Run Rate'
ax2.scatter(dc_per_over_stats[dc_per_over_stats['wickets_taken'] > 0]['over'], dc_per_over_stats[dc_per_over_st
ax2.set_title('DC Run Rate Per Over')
ax2.set_xlabel('Over')
ax2.set_ylabel('Run Rate (Runs per ball)')
ax2.legend()

plt.show()
```



The plotted run rates for each over, along with the moments when wickets were taken (marked with large dots), provide insights into how the match's dynamics evolved:

RCB Run Rate Fluctuations: #### RCB's run rate shows significant fluctuations, peaking at around 3.5 runs per ball towards the end of the innings. The presence of wicket markers (red circles) indicates that wickets were taken during overs where the run rate was generally lower, which is typical as wickets tend to disrupt batting flow.

DC Run Rate Patterns: DC's run rate starts strong but sees a sharp decline after the initial overs, stabilizing somewhat in the middle before another peak and subsequent fall towards the end. Wickets (blue circles) are taken in overs where the run rate drops, suggesting effective bowling from RCB during these times.