In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.1.0 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.

Traceback (most recent call last):  File "<frozen runpy>", line 198, in _run_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "C:\Users\VISHNUVARADHAN M\pms_for_ml\pms_env\Lib\site-packages\ipykernel_launcher.py", line 17, in <module>
    app.launch_new_instance()
  File "C:\Users\VISHNUVARADHAN M\pms_for_ml\pms_env\Lib\site-packages\traitlets\config\application.py", line 992, i
n launch_instance
    app.start()
  File "C:\Users\VISHNUVARADHAN M\pms_for_ml\pms_env\Lib\site-packages\ipykernel\kernelapp.py", line 711, in start
    self.io_loop.start()
```

In [3]:
```
df=pd.read_csv('odi_bbb-25.csv')
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2828551154.py:1: DtypeWarning: Columns (41,55,56) have mixed
types. Specify dtype option on import or set low_memory=False.
  df=pd.read_csv('odi_bbb-25.csv')
```

In [4]:
```
df.drop(['Unnamed: 0'],axis=1,inplace=True)
```

In [5]:
```
after_world_cup=df[df['year']>2018]
after_world_cup.head()
```

Out[5]:

	p_match	inns	bat	p_bat	team_bat	bowl	p_bowl	team_bowl	ball	ball_id	...	gmt_offset	wagonX	wagonY	wagonZone	
964614	1187023	1	Evin Lewis	431901	West Indies	Shardul Thakur	475281	India	1	0.01	...	0.0	242.0	195.0	3.0	OUTSIDE_
964615	1187023	1	Evin Lewis	431901	West Indies	Shardul Thakur	475281	India	2	0.02	...	0.0	136.0	108.0	8.0	ON_TI
964616	1187023	1	Evin Lewis	431901	West Indies	Shardul Thakur	475281	India	1	4.01	...	0.0	220.0	229.0	4.0	OUTSIDE_
964617	1187023	1	Evin Lewis	431901	West Indies	Shardul Thakur	475281	India	2	4.02	...	0.0	242.0	189.0	3.0	ON_TI
964618	1187023	1	Evin Lewis	431901	West Indies	Shardul Thakur	475281	India	3	4.03	...	0.0	0.0	0.0	0.0	OUTSIDE_

5 rows × 60 columns

In [6]:
```
after_world_cup.groupby(['year'])['p_match'].nunique()
```

Out[6]:
```
year
2019    149
2020     44
2021     71
2022    161
2023    218
2024    104
2025     13
Name: p_match, dtype: int64
```

In [7]:
```
#Venue Analysis
```

In [8]:
```
a=df[df['country']=='Pakistan']
a.head()
pak_venues=a[a['ground']!='Multan Cricket Stadium']
print(pak_venues['ground'].unique())
pak_venues.head()
```

```
['Rawalpindi Cricket Stadium' 'National Stadium, Karachi'
 'Gaddafi Stadium, Lahore' 'Iqbal Stadium, Faisalabad'
 'Arbab Niaz Stadium, Peshawar' 'Sheikhupura Stadium'
 'Niaz Stadium, Hyderabad, Sind']
```

Out[8]:

	p_match	inns	bat	p_bat	team_bat	bowl	p_bowl	team_bowl	ball	ball_id	...	gmt_offset	wagonX	wagonY	wagonZone	line	len
517	226356	1	Marcus Trescothick	21585	England	Naved-ul-Hasan	42272	Pakistan	1	0.01	...	0.0	112.0	63.0	8.0	NaN	N
518	226356	1	Marcus Trescothick	21585	England	Naved-ul-Hasan	42272	Pakistan	5	0.05	...	0.0	115.0	212.0	6.0	NaN	N
519	226356	1	Marcus Trescothick	21585	England	Naved-ul-Hasan	42272	Pakistan	6	0.06	...	0.0	110.0	203.0	6.0	NaN	N
520	226356	1	Matt Prior	18675	England	Naved-ul-Hasan	42272	Pakistan	2	0.02	...	0.0	218.0	205.0	4.0	NaN	N
521	226356	1	Matt Prior	18675	England	Naved-ul-Hasan	42272	Pakistan	3	0.03	...	0.0	0.0	0.0	0.0	NaN	N

5 rows × 60 columns

In [9]:
```
pak_venues['p_match'].nunique()
Karachi=pak_venues[pak_venues['ground']=='National Stadium, Karachi']
Rawalpindi=pak_venues[pak_venues['ground']=='Rawalpindi Cricket Stadium']
Lahore=pak_venues[pak_venues['ground']=='Gaddafi Stadium, Lahore']
Dubai=df[df['ground']=='Dubai International Cricket Stadium']
```

In [10]:
```
Lahore['daynight']=Lahore['daynight'].apply(lambda x:'Day' if x=='day match' else 'Day/Night')
Karachi['daynight']=Karachi['daynight'].apply(lambda x:'Day' if x=='day match' else 'Day/Night')
Rawalpindi['daynight']=Rawalpindi['daynight'].apply(lambda x:'Day' if x=='day match' else 'Day/Night')
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2597313951.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Lahore['daynight']=Lahore['daynight'].apply(lambda x:'Day' if x=='day match' else 'Day/Night')
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2597313951.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Karachi['daynight']=Karachi['daynight'].apply(lambda x:'Day' if x=='day match' else 'Day/Night')
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2597313951.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Rawalpindi['daynight']=Rawalpindi['daynight'].apply(lambda x:'Day' if x=='day match' else 'Day/Night')
```

In [11]:
```
def win(row):
  if row['inns']==1:
    if row['winner']==row['team_bat']:
      return 'Bat First'
    elif row['winner']==np.nan:
      return 'NR'
    else:
      return 'Bat Second'
Karachi['win']=Karachi.apply(win,axis=1)
Lahore['win']=Lahore.apply(win,axis=1)
Rawalpindi['win']=Rawalpindi.apply(win,axis=1)
Dubai['win']=Dubai.apply(win,axis=1)
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\1315307450.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Karachi['win']=Karachi.apply(win,axis=1)
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\1315307450.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Lahore['win']=Lahore.apply(win,axis=1)
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\1315307450.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Rawalpindi['win']=Rawalpindi.apply(win,axis=1)
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\1315307450.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  Dubai['win']=Dubai.apply(win,axis=1)
```

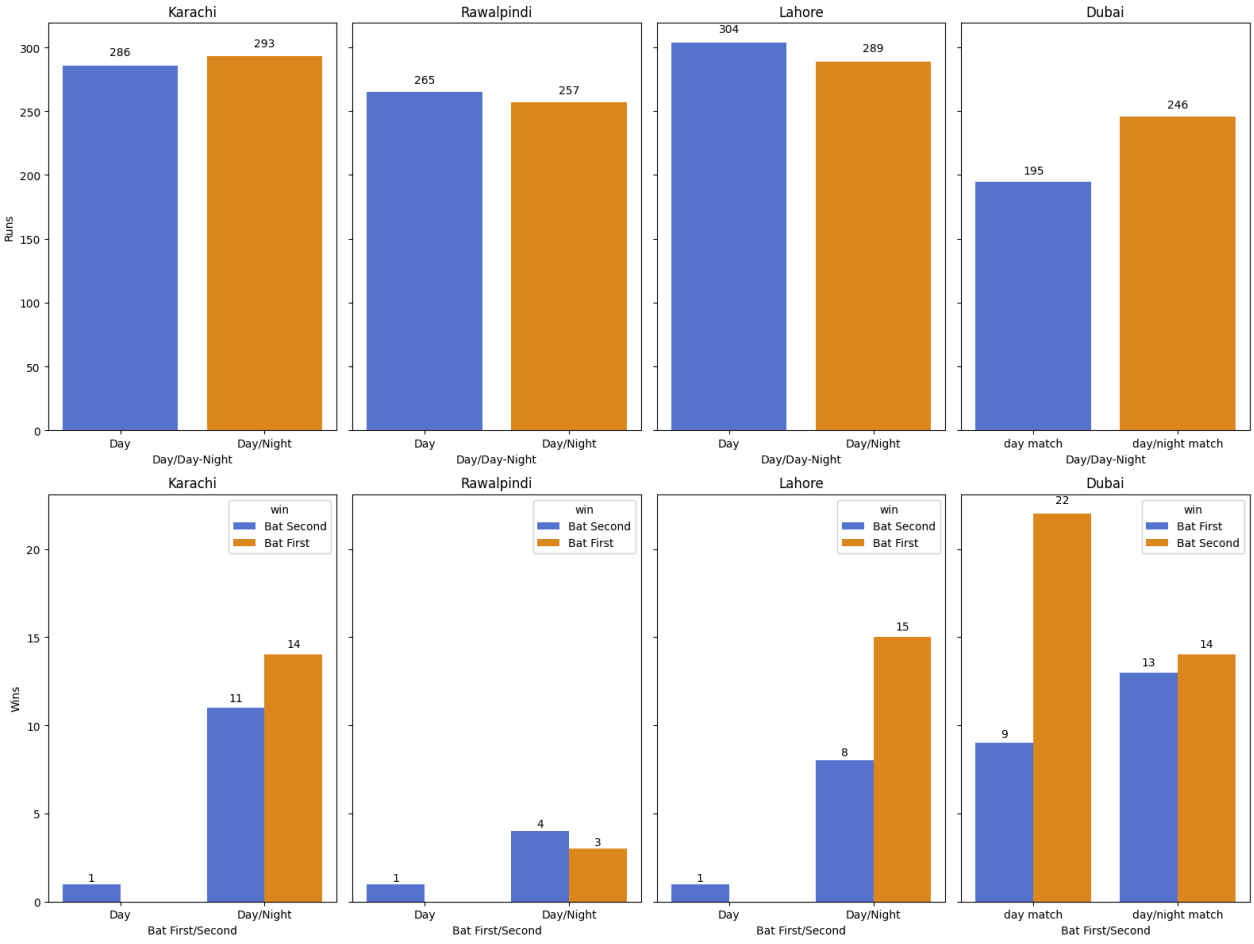
In [12]:
```
l_count=Lahore.groupby(['daynight','win'])['p_match'].nunique().reset_index(name='count')
r_count=Rawalpindi.groupby(['daynight','win'])['p_match'].nunique().reset_index(name='count')
k_count=Karachi.groupby(['daynight','win'])['p_match'].nunique().reset_index(name='count')
```

In [13]:
```
Karachi=a[a['ground']=='National Stadium, Karachi']
Rawalpindi=a[a['ground']=='Rawalpindi Cricket Stadium']
Lahore=a[a['ground']=='Gaddafi Stadium, Lahore']

avg_karachi=Karachi.groupby(['p_match','daynight','target']).size().reset_index()
avg_karachi['target']=avg_karachi['target']-1
avg_karachi['daynight']=avg_karachi['daynight'].apply(lambda x :'Day' if x=='day match' else 'Day/Night')
avg_k=avg_karachi.groupby(['daynight'])['target'].mean().astype(int).reset_index()


avg_rawalpindi=Rawalpindi.groupby(['p_match','daynight','target']).size().reset_index()
avg_rawalpindi['target']=avg_rawalpindi['target']-1
avg_rawalpindi['daynight']=avg_rawalpindi['daynight'].apply(lambda x :'Day' if x=='day match' else 'Day/Night')
avg_r=avg_rawalpindi.groupby(['daynight'])['target'].mean().astype(int).reset_index()

avg_lahore=Lahore.groupby(['p_match','daynight','target']).size().reset_index()
avg_lahore['target']=avg_lahore['target']-1
avg_lahore['daynight']=avg_lahore['daynight'].apply(lambda x :'Day' if x=='day match' else 'Day/Night')
avg_l=avg_lahore.groupby(['daynight'])['target'].mean().astype(int).reset_index()

d_count=Dubai.groupby(['daynight','win'])['p_match'].nunique().reset_index(name='count')
avg_Dubai=Dubai.groupby(['p_match','daynight','target']).size().reset_index()
avg_Dubai['target']=avg_Dubai['target']-1
avg_Dubai['daynight']=avg_Dubai['daynight']
avg_d=avg_Dubai.groupby(['daynight'])['target'].mean().astype(int).reset_index()
```

In [14]:

```
fig,ax=plt.subplots(2,4,figsize=(16,12),sharey='row')
sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_k,ax=ax[0,0])
sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_r,ax=ax[0,1])
sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_l,ax=ax[0,2])
sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_d,ax=ax[0,3])

ax[0,0].set_title('Karachi')
ax[0,1].set_title('Rawalpindi')
ax[0,2].set_title('Lahore')
ax[0,3].set_title('Dubai')
ax[0,0].set_xlabel('Day/Day-Night')
ax[0,1].set_xlabel('Day/Day-Night')
ax[0,2].set_xlabel('Day/Day-Night')
ax[0,3].set_xlabel('Day/Day-Night')
ax[0,0].set_ylabel('Runs')
ax[0,1].set_ylabel('Runs')
ax[0,2].set_ylabel('Runs')
ax[0,3].set_ylabel('Runs')
plt.title('Average 1st innings score')


sns.barplot(data=k_count,x='daynight',y='count',hue='win', palette=["royalblue", "darkorange"],ax=ax[1,0])
sns.barplot(data=r_count,x='daynight',y='count',hue='win', palette=["royalblue", "darkorange"],ax=ax[1,1])
sns.barplot(data=l_count,x='daynight',y='count',hue='win', palette=["royalblue", "darkorange"],ax=ax[1,2])
sns.barplot(data=d_count,x='daynight',y='count',hue='win', palette=["royalblue", "darkorange"],ax=ax[1,3])

ax[1,0].set_title('Karachi')
ax[1,1].set_title('Rawalpindi')
ax[1,2].set_title('Lahore')
ax[1,3].set_title('Dubai')
ax[1,0].set_xlabel('Bat First/Second')
ax[1,1].set_xlabel('Bat First/Second')
ax[1,2].set_xlabel('Bat First/Second')
ax[1,3].set_xlabel('Bat First/Second')
ax[1,0].set_ylabel('Wins')
ax[1,1].set_ylabel('Wins')
ax[1,2].set_ylabel('Wins')
ax[1,3].set_ylabel('Wins')
plt.tight_layout()

for i in range(2):
  for j in range(4):
    for bars in ax[i,j].containers:
      for bar in bars:
        height=bar.get_height()
        width=bar.get_width()
        x=bar.get_x()
        y=bar.get_y()
        ax[i,j].annotate(f'{int(height)}',(x+width/2,y+height*1.02),ha='center',va='bottom')


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

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:2: FutureWarning:

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

  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_k,ax=ax[0,0])
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:2: UserWarning: The palette list has more valu
es (3) than needed (2), which may not be intended.
  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_k,ax=ax[0,0])
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:3: FutureWarning:

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

  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_r,ax=ax[0,1])
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:3: UserWarning: The palette list has more valu
es (3) than needed (2), which may not be intended.
  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_r,ax=ax[0,1])
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:4: FutureWarning:

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

  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_l,ax=ax[0,2])
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:4: UserWarning: The palette list has more valu
es (3) than needed (2), which may not be intended.
  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_l,ax=ax[0,2])
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:5: FutureWarning:

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

  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_d,ax=ax[0,3])
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3871099337.py:5: UserWarning: The palette list has more valu
es (3) than needed (2), which may not be intended.
  sns.barplot(x='daynight',y='target',palette=["royalblue", "darkorange",'green'],data=avg_d,ax=ax[0,3])
```



-->Among the four stadiums analyzed, Dubai has the lowest average first-innings score. Additionally, there is a noticeable difference in first-innings scores between day and day-night matches. This suggests that batting conditions improve as the day progresses.

--> In the second row of charts, it is evident that Dubai has generally favored teams batting second. However, in day-night matches, the outcomes have not been strongly influenced by the toss. This indicates that matches in Dubai are not dictated by a coin flip, which is an encouraging aspect for cricket fans.

-->For day-night matches, focusing on adaptability rather than toss dependency is key, as both batting first and second have similar chances

In [15]:
```
avg_lahore['daynight']=avg_lahore['daynight'].apply(lambda x :'Day' if x=='day match' else 'Day/Night')
```

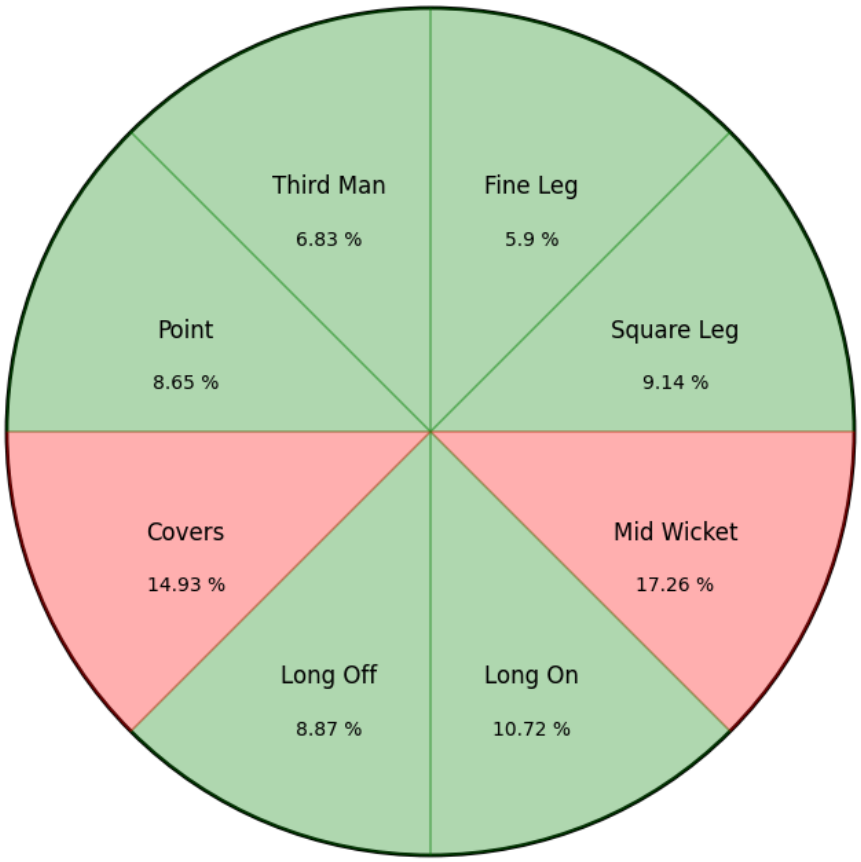
In [16]:
```
def per(run,z):
    return (run/z)*100
mapping={1:'Fine Leg',2:'Square Leg',3:'Mid Wicket',4:'Long on',5:'Long off',6:'Covers',7:'Point',8:'Third Man'}
```

In [17]:
```
rhb_lahore=Lahore[Lahore['bat_hand']=='RHB']
s1=rhb_lahore['batruns'].sum()
rhb_group1=rhb_lahore.groupby('wagonZone')['batruns'].sum().reset_index()
rhb_group1=rhb_group1[1:]
rhb_group1['wagonZone']=rhb_group1['wagonZone'].map(mapping)
rhb_group1['Propotion']=rhb_group1['batruns'].apply(lambda x:per(x,s1)).round(2)

#karachi
rhb_karachi=Karachi[Karachi['bat_hand']=='RHB']
s2=rhb_karachi['batruns'].sum()
rhb_group2=rhb_karachi.groupby('wagonZone')['batruns'].sum().reset_index()
mapping={1:'Fine Leg',2:'Square Leg',3:'Mid Wicket',4:'Long on',5:'Long off',6:'Covers',7:'Point',8:'Third Man'}
rhb_group2=rhb_group2[1:]
rhb_group2['wagonZone']=rhb_group2['wagonZone'].map(mapping)
rhb_group2['Propotion']=rhb_group2['batruns'].apply(lambda x:per(x,s2)).round(2)

#Rawalpindi
rhb_rawalpindi=Rawalpindi[Rawalpindi['bat_hand']=='RHB']
sn=rhb_rawalpindi['batruns'].sum()
rhb_group3=rhb_rawalpindi.groupby('wagonZone')['batruns'].sum().reset_index()
mapping={1:'Fine Leg',2:'Square Leg',3:'Mid Wicket',4:'Long on',5:'Long off',6:'Covers',7:'Point',8:'Third Man'}
rhb_group3=rhb_group3[1:]
rhb_group3['wagonZone']=rhb_group3['wagonZone'].map(mapping)
rhb_group3['Propotion']=rhb_group3['batruns'].apply(lambda x:per(x,sn)).round(2)

#Dubai
rhb_dubai=Dubai[Dubai['bat_hand']=='RHB']
s3=rhb_dubai['batruns'].sum()
rhb_group4=rhb_dubai.groupby('wagonZone')['batruns'].sum().reset_index()
mapping={1:'Fine Leg',2:'Square Leg',3:'Mid Wicket',4:'Long on',5:'Long off',6:'Covers',7:'Point',8:'Third Man'}
rhb_group4=rhb_group4[1:]
rhb_group4['wagonZone']=rhb_group4['wagonZone'].map(mapping)
rhb_group4['Propotion']=rhb_group4['batruns'].apply(lambda x:per(x,s3)).round(2)
```

In [18]:
```
field_positions = [
    ("Fine Leg", 45, 90),
    ("Square Leg", 0, 45),
    ("Mid Wicket", 315, 360),
    ("Long On", 270, 315),
    ("Long Off", 225, 270),
    ("Covers", 180, 225),
    ("Point", 135, 180),
    ("Third Man", 90, 135)
]
```

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Wedge

fig, ax = plt.subplots(figsize=(10,10))
ax.set_xlim(-50, 50)
ax.set_ylim(-50, 50)
ax.set_aspect('equal')


circle = plt.Circle((0, 0), 40, color='black', fill=False, linewidth=2)
ax.add_patch(circle)

p=np.array(rhb_group4['Propotion'])
m=np.sort(p)
for label, start, end, runs in zip([x[0] for x in field_positions], [x[1] for x in field_positions], [x[2] for x in fiel
    if runs==m[-1] or runs==m[-2]:
        wedge = Wedge((0, 0), 40, start, end, color='red', alpha=0.3, edgecolor='black')
        ax.add_patch(wedge)
    else:
         wedge = Wedge((0, 0), 40, start, end, color='green', alpha=0.3, edgecolor='black')
         ax.add_patch(wedge)



    angle = np.radians((start + end) / 2)
    x_text = 25 * np.cos(angle)
    y_text = 25 * np.sin(angle)

    ax.text(x_text, y_text, label, fontsize=12, ha='center', va='center', color='black')
    ax.text(x_text, y_text - 5, f'{runs} %', fontsize=10, ha='center', va='center', color='black')


ax.set_title('Distribution of Runs in Dubai[RHB]')
ax.set_xticks([])
ax.set_yticks([])
ax.set_frame_on(False)

plt.show()
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2948428524.py:21: UserWarning: Setting the 'color' property
will override the edgecolor or facecolor properties.
  wedge = Wedge((0, 0), 40, start, end, color='green', alpha=0.3, edgecolor='black')
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2948428524.py:18: UserWarning: Setting the 'color' property
will override the edgecolor or facecolor properties.
  wedge = Wedge((0, 0), 40, start, end, color='red', alpha=0.3, edgecolor='black')
```



--> As expected, mid-wicket has been a dominant scoring region, indicating that players frequently target this area. Additionally, a significant number of runs have come from cover drives, showing a preference for front-foot strokes.

--> The Long On (10.72%) and Mid Wicket (17.26%) regions have seen a high share of runs, reinforcing that batters prefer playing straighter or towards the on-side rather than relying heavily on square shots. This could indicate that bowlers often bowl fuller lengths, allowing drives and wristy on-side strokes.

--> More than 50% of the total runs have been scored in front of the wicket, while only around 30% have been scored square of the wicket. This suggests that the pitch is not particularly quick, meaning that extra pace may not be as effective. Instead, slower deliveries and cutters are likely to be more impactful in Dubai conditions.

In [20]:
```
spin_pace_Dubai = Dubai[(Dubai["out"] == True)&(Dubai['year']>2019)].groupby(["ground",'daynight', "bowl_kind"])["out"].
spin_pace_Dubai['Total_Wickets']=(spin_pace_Dubai['mixture/unknown']+spin_pace_Dubai['pace bowler']+spin_pace_Dubai['spi
spin_pace_Dubai['Spin bowler Wickets %']=(100*spin_pace_Dubai['spin bowler']/spin_pace_Dubai['Total_Wickets']).round(2)
spin_pace_Dubai['Pace bowler Wickets %']=(100*spin_pace_Dubai['pace bowler']/spin_pace_Dubai['Total_Wickets']).round(2)
spin_pace_Dubai['Mixture/Unknown Wickets %']=(100*spin_pace_Dubai['mixture/unknown']/spin_pace_Dubai['Total_Wickets']).r
spin_pace_Dubai
```

Out[20]:

ground	daynight	bowl_kind	mixture/unknown	pace bowler	spin bowler	Total_Wickets	Spin bowler Wickets %	Pace bowler Wickets %	Mixture/Unknown Wickets %
Dubai International Cricket Stadium	day match	12	169	96	277	34.66	61.01	4.33	
	day/night match	1	57	35	93	37.63	61.29	1.08	

In [21]:
```
pak_venues = pak_venues[pak_venues["ground"].isin(["Gaddafi Stadium, Lahore",  "National Stadium, Karachi", "Rawalpindi
v=pak_venues['ground'].unique()
```

In [22]:
```
spin_pace_pak_venue = pak_venues[pak_venues["out"] == True].groupby(["ground",'daynight', "bowl_kind"])["out"].count().u
spin_pace_pak_venue['Total_Wickets']=(spin_pace_pak_venue['mixture/unknown']+spin_pace_pak_venue['pace bowler']+spin_pac
spin_pace_pak_venue['Spin bowler Wickets %']=(100*spin_pace_pak_venue['spin bowler']/spin_pace_pak_venue['Total_Wickets'
spin_pace_pak_venue['Pace bowler Wickets %']=(100*spin_pace_pak_venue['pace bowler']/spin_pace_pak_venue['Total_Wickets'
spin_pace_pak_venue['Mixture/Unknown Wickets %']=(100*spin_pace_pak_venue['mixture/unknown']/spin_pace_pak_venue['Total_
spin_pace_pak_venue
```

Out[22]:

ground	daynight	bowl_kind	mixture/unknown	pace bowler	spin bowler	Total_Wickets	Spin bowler Wickets %	Pace bowler Wickets %	Mixture/Unknown Wickets %
Gaddafi Stadium, Lahore	day match	0.0	6.0	4.0	10.0	40.00	60.00	0.00	
	day/night match	0.0	188.0	90.0	278.0	32.37	67.63	0.00	
	night match	1.0	23.0	29.0	53.0	54.72	43.40	1.89	
National Stadium, Karachi	day match	0.0	9.0	1.0	10.0	10.00	90.00	0.00	
	day/night match	0.0	144.0	85.0	229.0	37.12	62.88	0.00	
	night match	1.0	68.0	62.0	131.0	47.33	51.91	0.76	
Rawalpindi Cricket Stadium	day match	0.0	11.0	2.0	13.0	15.38	84.62	0.00	
	day/night match	4.0	75.0	26.0	105.0	24.76	71.43	3.81	

In [23]:
```
ind=df[(df['team_bat']=='India')]
ind=ind[ind['year']>2019]
```

In [24]:
```
champions_trophy_players = ["Rohit Sharma","Shubman Gill","Virat Kohli","Shreyas Iyer","KL Rahul","Hardik Pandya","Ravin
]
ind_ct=ind[ind['bat'].isin(champions_trophy_players)]
```

In [25]:
```
bat_stats=ind_ct.groupby('bat').aggregate({'p_match':'nunique','batruns':'sum','out':'sum','ball':'count','cur_bat_runs'
bat_stats['Average']=(bat_stats['batruns']/bat_stats['out']).round(2)
bat_stats['Strike Rate']=((bat_stats['batruns']/bat_stats['ball'])*100).round(2)
bat_stats=bat_stats.rename(columns={'p_match':'Matches','batruns':'Runs','bat':'Batsman','cur_bat_runs':'Highest Score'}
bat_stats=bat_stats.sort_values(by='Runs',ascending=False)
bat_stats.head()
```

Out[25]:

	Batsman	Matches	Runs	Highest Score	Average	Strike Rate
11	Shubman Gill	48	2564	208	59.63	99.84
12	Virat Kohli	52	2340	166	49.79	92.75
10	Shreyas Iyer	50	2127	128	49.47	99.11
9	Rohit Sharma	46	2036	131	49.66	110.83
4	KL Rahul	50	2014	112	51.64	89.67

In [26]:
```
import matplotlib.pyplot as plt
import pandas as pd
bat_stats = pd.DataFrame(bat_stats)
fig, ax = plt.subplots(figsize=(8, 4))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=bat_stats.values,
                 colLabels=bat_stats.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)
plt.title("Stats of India Players Since 2020", fontsize=14, weight='bold')
plt.show()
```

Stats of India Players Since 2020

Batsman	Matches	Runs	Highest Score	Average	Strike Rate
Shubman Gill	48	2564	208	59.63	99.84
Virat Kohli	52	2340	166	49.79	92.75
Shreyas Iyer	50	2127	128	49.47	99.11
Rohit Sharma	46	2036	131	49.66	110.83
KL Rahul	50	2014	112	51.64	89.67
Hardik Pandya	26	848	92	40.38	104.05
Ravindra Jadeja	28	591	66	34.76	79.76
Rishabh Pant	14	525	125	43.75	104.37
Axar Patel	22	492	64	27.33	90.77
Washington Sunda	15	329	51	23.5	82.04
Kuldeep Yadav	21	105	17	8.08	42.34
Mohammed Sham	14	84	23	7.64	82.35
Arshdeep Singh	6	39	18	9.75	108.33
Harshit Rana	1	13	13	13.0	130.0

--> Shubman Gill has been outstanding since 2020, boasting an impressive average of 59.63, which significantly strengthens India's batting lineup. His consistency has been a key asset for the team.

-->Ravindra Jadeja's strike rate (79.76) is a bit concerning, especially considering his role as a finisher. This suggests he may need to accelerate more effectively in the later stages of an innings.

--> Despite discussions about Virat Kohli's form, his numbers tell a different story—he has accumulated a significant number of runs with an excellent average (49.79) and a strong strike rate (92.75), reaffirming his value in the batting order.

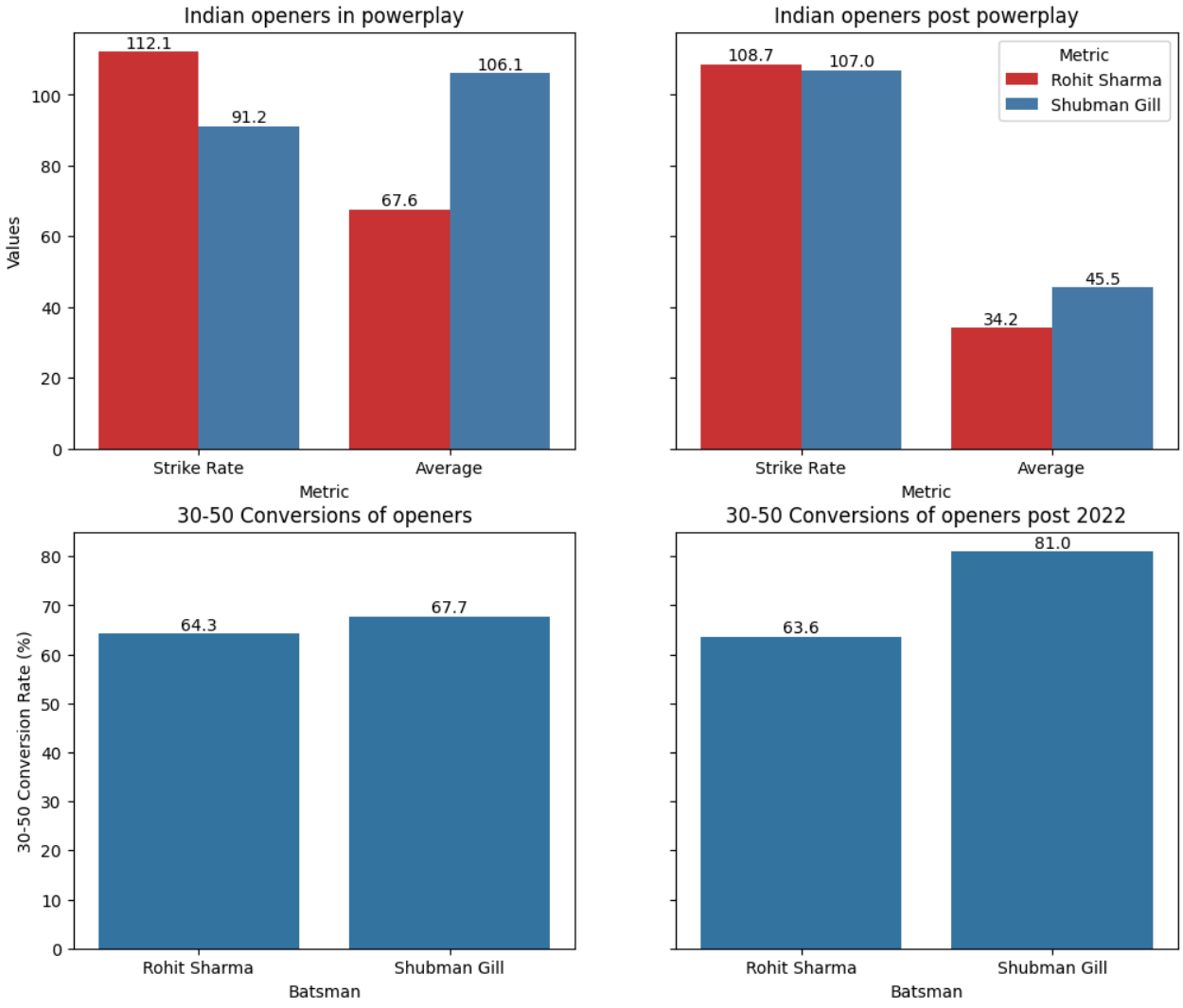
..>With a strike rate of 110.83, Rohit Sharma has been the most aggressive top-order batter in the lineup. His ability to score quickly while maintaining a strong average (49.66) is crucial in setting the tone for India.

--> KL Rahul has quietly maintained an average of 51.64, which is the second-best after Gill. Despite fluctuations in his role (opener vs. middle-order batter), he has delivered consistent performances.

In [27]:
```
#openers
powerplay=ind_ct[ind_ct['over']<11]
openers_in_powerplay=powerplay[powerplay['bat'].isin(champions_trophy_players[:2])]
opener_pp_stats=openers_in_powerplay.groupby('bat').aggregate({'batruns':'sum','ball':'count','cur_bat_runs':'max','p_ma
opener_pp_stats['Strike Rate']=((opener_pp_stats['batruns']/opener_pp_stats['ball'])*100).round(2)
opener_pp_stats['Average']=(opener_pp_stats['batruns']/opener_pp_stats['out']).round(2)

opener_pp_stats=opener_pp_stats.rename(columns={'p_match':'Matches','batruns':'Runs','bat':'Batsman','cur_bat_runs':'Hig
melted=opener_pp_stats.drop(['Matches','Highest Score','Runs'],axis=1).melt(id_vars='Batsman',var_name='Metric',value_na

post_pp=ind_ct[ind_ct['over']>10]
openers_post_pp=post_pp[post_pp['bat'].isin(champions_trophy_players[:2])]
opener_post_pp_stats=openers_post_pp.groupby('bat').aggregate({'batruns':'sum','ball':'count','cur_bat_runs':'max','p_ma
#opener_post_pp_stats['Average Duration']=(opener_post_pp_stats['ball']/opener_post_pp_stats['p_match']).round()
opener_post_pp_stats['Strike Rate']=((opener_post_pp_stats['batruns']/opener_post_pp_stats['ball'])*100).round(2)
opener_post_pp_stats['Average']=(opener_post_pp_stats['batruns']/opener_post_pp_stats['out']).round(2)

opener_post_pp_stats=opener_post_pp_stats.rename(columns={'p_match':'Matches','batruns':'Runs','bat':'Batsman','cur_bat_
melted1=opener_post_pp_stats.drop(['Matches','Highest Score','Runs'],axis=1).melt(id_vars='Batsman',var_name='Metric',va
```

In [28]:
```
openers=ind_ct[ind_ct['bat'].isin(champions_trophy_players[:2])]
openers_2022=openers[openers['year']>2022]
openers_stats=openers.groupby(['bat','p_match']).aggregate({'cur_bat_runs':'max'}).reset_index()
openers_stats_2022=openers_2022.groupby(['bat','p_match']).aggregate({'cur_bat_runs':'max'}).reset_index()
openers_stats['50 Conversion']=openers_stats['cur_bat_runs'].apply(lambda x: 1 if (x>30 and x>50)  else (0 if (x>30 and
openers_stats['100 Conversion']=openers_stats['cur_bat_runs'].apply(lambda x: 1 if (x>50 and x>100)  else (0 if (x>50 an
openers_stats['30 Conversion']=openers_stats['cur_bat_runs'].apply(lambda x: 1 if (x>30 and x>0)  else (0 if (x<30 and x
openers_stats_2022['50 Conversion']=openers_stats_2022['cur_bat_runs'].apply(lambda x: 1 if (x>30 and x>50)  else (0 if
openers_stats_2022['100 Conversion']=openers_stats_2022['cur_bat_runs'].apply(lambda x: 1 if (x>50 and x>100)  else (0 i
openers_stats_2022['30 Conversion']=openers_stats_2022['cur_bat_runs'].apply(lambda x: 1 if (x>30 and x>0)  else (0 if (
conversion_stats = openers_stats.groupby("bat").agg(
    thirty_plus_scores=("30 Conversion", lambda x: (x == 1).sum()),
    fifty_plus_scores=("50 Conversion", lambda x: (x == 1).sum()),
    hundreds=("100 Conversion", lambda x: (x == 1).sum())
).reset_index()
conversion_stats_2022 = openers_stats_2022.groupby("bat").agg(
    thirty_plus_scores=("30 Conversion", lambda x: (x == 1).sum()),
    fifty_plus_scores=("50 Conversion", lambda x: (x == 1).sum()),
    hundreds=("100 Conversion", lambda x: (x == 1).sum())
).reset_index()
conversion_stats["50-100 Conversion Rate (%)"] = (
    (conversion_stats["hundreds"] / conversion_stats["fifty_plus_scores"]) * 100
).round(2).fillna(0)
conversion_stats["30-50 Conversion Rate (%)"] = (
    (conversion_stats["fifty_plus_scores"] / conversion_stats["thirty_plus_scores"]) * 100
).round(2).fillna(0)
conversion_stats_2022["50-100 Conversion Rate (%)"] = (
    (conversion_stats_2022["hundreds"] / conversion_stats_2022["fifty_plus_scores"]) * 100
).round(2).fillna(0)
conversion_stats_2022["30-50 Conversion Rate (%)"] = (
    (conversion_stats_2022["fifty_plus_scores"] / conversion_stats_2022["thirty_plus_scores"]) * 100
).round(2).fillna(0)
```

In [29]:
```
#post powerplay average (long innings)
fig,ax=plt.subplots(2,2,figsize=(12,10),sharey='row')
sns.barplot(data=melted,x='Metric',y='Values',hue='Batsman',palette='Set1',ax=ax[0,0],legend=False)
ax[0,0].set_title('Indian openers in powerplay')
for p in ax[0,0].patches:
    ax[0,0].annotate(format(p.get_height(), '.1f'),
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)


sns.barplot(data=melted1,x='Metric',y='Values',hue='Batsman',palette='Set1',ax=ax[0,1])
ax[0,1].set_title('Indian openers post powerplay')
ax[0,1].legend(title="Metric")
for p in ax[0,1].patches:
    height=p.get_height()
    if height>0:
        ax[0,1].annotate(format(p.get_height(), '.1f'),
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)

sns.barplot(x='bat',y='30-50 Conversion Rate (%)',data=conversion_stats,ax=ax[1,0],legend=False)
ax[1,0].set_title('30-50 Conversions of openers')
ax[1,0].set_xlabel('Batsman')
for p in ax[1,0].patches:
    ax[1,0].annotate(format(p.get_height(), '.1f'),
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)



sns.barplot(x='bat',y='30-50 Conversion Rate (%)',data=conversion_stats_2022,ax=ax[1,1],legend=False)
ax[1,1].set_title('30-50 Conversions of openers post 2022')
ax[1,1].set_xlabel('Batsman')
for p in ax[1,1].patches:
    ax[1,1].annotate(format(p.get_height(), '.1f'),
                     (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='bottom', fontsize=10)
plt.show()
```



--> Rohit Sharma’s aggressive intent (SR: 112.1) and Shubman Gill’s stability (Avg: 106.1) goes hand in hand at the top.

-->Rohit’s attacking approach helps put pressure on the bowlers, giving Gill the time and space to settle in before accelerating.

--> Gill significantly improves his strike rate post-powerplay (107.0, close to Rohit’s 108.7), indicating his ability to shift gears. However, both openers see a drop in average (Rohit: 34.2, Gill: 45.5), suggesting they are more likely to get dismissed once fielding restrictions ease.

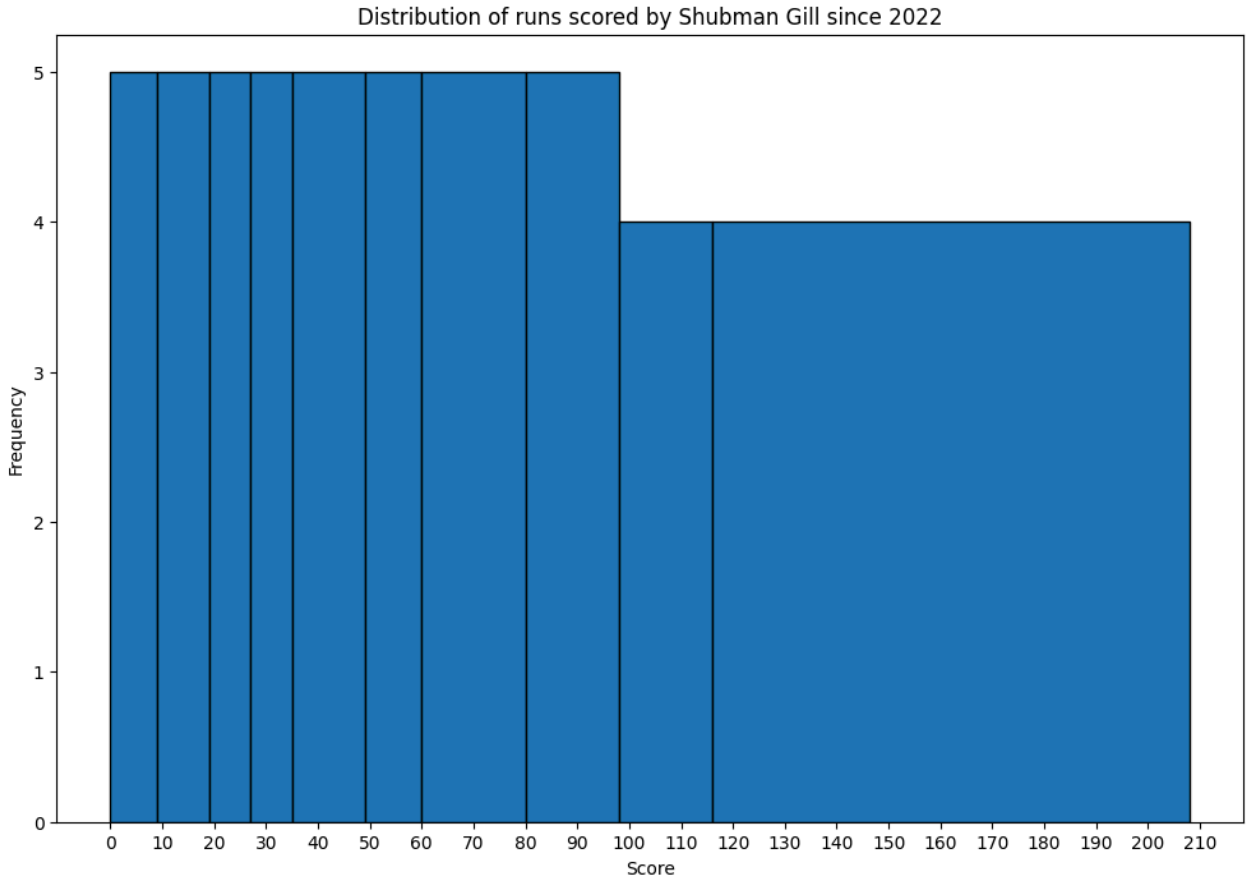
--> Post-powerplay dismissals need to be minimized for at least one opener to bat deep.

-->The first plot in the second row shows the 30-50 conversion rate of both openers, with both Rohit Sharma and Shubman Gill having similar conversion rates (~64-68%), indicating that they consistently turn their starts into impactful scores.

--> But post-2022, Shubman Gill's conversion rate has surged to 81%, an incredible 14% jump, which clearly shows how he has improved his temperament as an opener and his ability to build big innings.

In [30]:
```
gill=openers[openers['bat']=='Shubman Gill']
gill_scores=gill.groupby('p_match')['cur_bat_runs'].max().reset_index()
val=np.sort(gill_scores['cur_bat_runs'])
bins=np.array_split(val,10)
bin_edges=[bin[0] for bin in bins]+[bins[-1][-1]]
plt.figure(figsize=(12,8))
plt.hist(val,bins=bin_edges,edgecolor='black')
plt.xticks(np.arange(0,220,10))
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.title('Distribution of runs scored by Shubman Gill since 2022')
```

Out[30]:
```
Text(0.5, 1.0, 'Distribution of runs scored by Shubman Gill since 2022')
```



Smaller bins mean more bars where Gill scores often, showing his most common scores clearly.

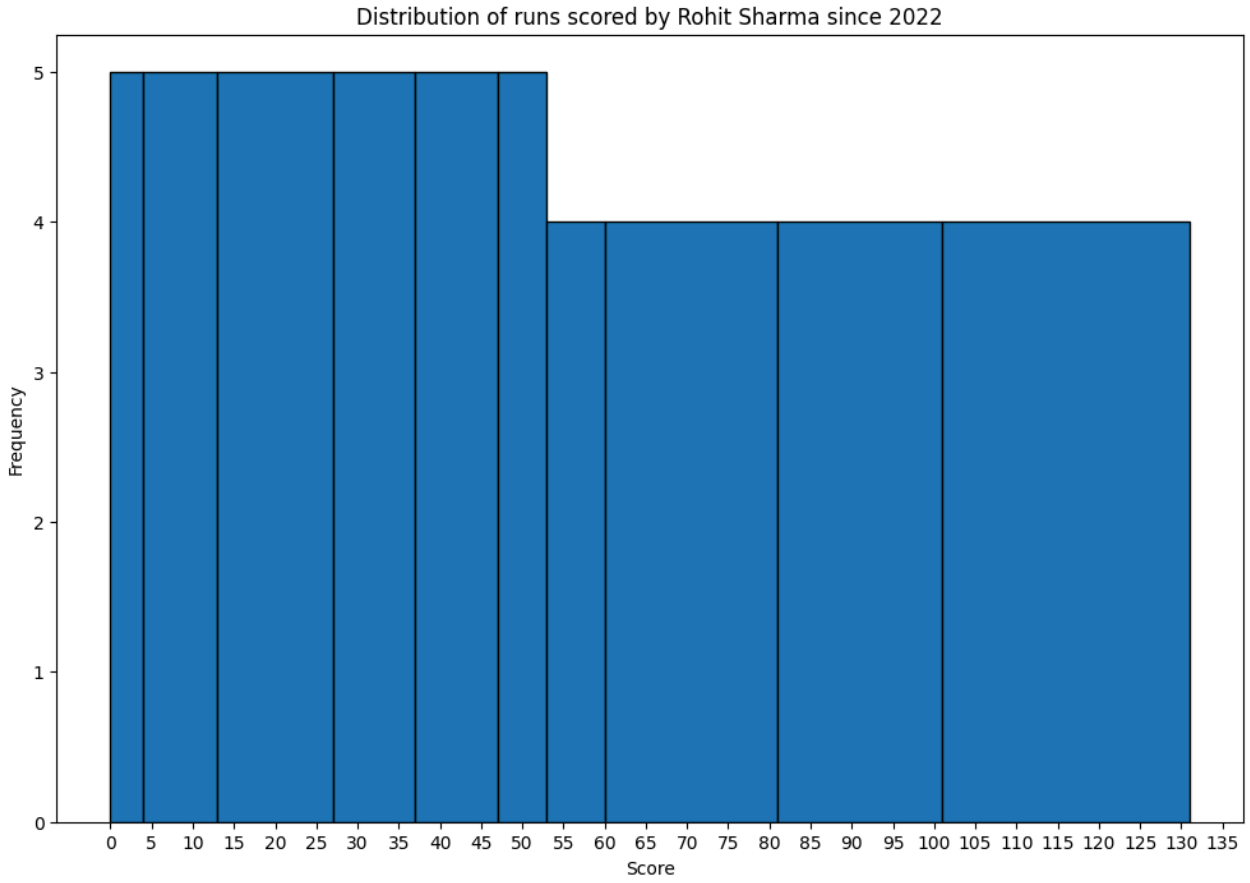
The high frequency of scores between 10-30 indicates that while Gill often gets starts, he tends to throw his wicket away in the powerplay. Addressing this could significantly boost his consistency.

here's a noticeable drop-off in conversions between 50-60, suggesting that he has been dismissed multiple times shortly after reaching a half-century. This might indicate a lapse in concentration or a shift in approach during this phase.

Once he crosses 60, there's a higher likelihood of him converting into a century, showing that when he settles in, he goes big. This highlights his ability to capitalize on good starts, but he needs to work on getting through the 50-60 phase more consistently.

In [31]:
```
gill=openers[openers['bat']=='Rohit Sharma']
gill_scores=gill.groupby('p_match')['cur_bat_runs'].max().reset_index()
val=np.sort(gill_scores['cur_bat_runs'])
bins=np.array_split(val,10)
bin_edges=[bin[0] for bin in bins]+[bins[-1][-1]]
plt.figure(figsize=(12,8))
plt.hist(val,bins=bin_edges,edgecolor='black')
plt.xticks(np.arange(0,140,5))
plt.xlabel('Score')
plt.ylabel('Frequency')
plt.title('Distribution of runs scored by Rohit Sharma since 2022')
```

Out[31]: Text(0.5, 1.0, 'Distribution of runs scored by Rohit Sharma since 2022')



Both Rohit Sharma and Shubman Gill have been dismissed early quite a few times, but Rohit's early dismissals are more frequent. This often puts India under pressure in the early overs.

Gill shows a strong tendency to convert 60+ scores into centuries.

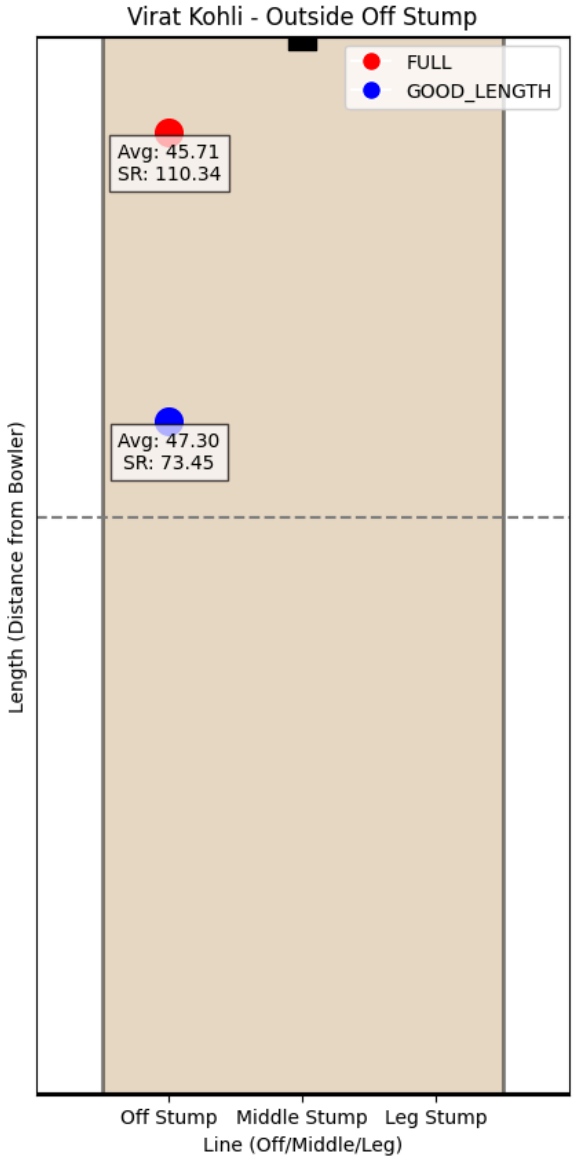
Rohit, on the other hand, has often fallen between 50-60 and has rarely converted to big scores in recent times.

In [32]:
```
kholi=df[(df['year']>2021) & (df['bat']=='Virat Kohli')]
```

In [33]:
```
oos=kholi.groupby(['line','length']).aggregate({'batruns':'sum','ballfaced':'sum','out':'sum','dismissal':'count'}).rese
oos=oos[(oos['line']=='OUTSIDE_OFFSTUMP')]
oos=oos[(oos['length']=='FULL')|(oos['length']=='GOOD_LENGTH')]
oos['average']=(oos['batruns']/oos['out']).round(2)
oos['strike rate']=((oos['batruns']/oos['ballfaced'])*100).round(2)
```

In [34]:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
ball_data = [
    {"line": "OUTSIDE_OFFSTUMP", "length": "FULL", "average": 45.71, "strike_rate": 110.34},
    {"line": "OUTSIDE_OFFSTUMP", "length": "GOOD_LENGTH", "average": 47.30, "strike_rate": 73.45},
]

line_map = {"OUTSIDE_OFFSTUMP": -1}
length_map = {"FULL": 2, "GOOD_LENGTH": 8}
length_colors = {"FULL": "red", "GOOD_LENGTH": "blue"}

fig, ax = plt.subplots(figsize=(5, 10))
pitch = patches.Rectangle((-1.5, 0), 3, 22, linewidth=2, edgecolor='black', facecolor='tan', alpha=0.5)
ax.add_patch(pitch)
plt.axhline(y=0, color="black", linewidth=3)
plt.axhline(y=22, color="black", linewidth=3)
plt.axhline(y=10, color="gray", linestyle="--", label="Good Length Zone")
plt.scatter([0], [0], color="black", s=200, label="Stumps", marker='s')
for ball in ball_data:
    x = line_map[ball["line"]]
    y = length_map[ball["length"]]
    color = length_colors[ball["length"]]
    plt.scatter(x, y, color=color, s=200, label=ball["length"])
    plt.text(x, y + 1, f"Avg: {ball['average']:.2f}\nSR: {ball['strike_rate']:.2f}",
             fontsize=10, ha='center', bbox=dict(facecolor='white', alpha=0.7))


plt.xlim(-2, 2)
plt.ylim(0, 22)
plt.xlabel("Line (Off/Middle/Leg)")
plt.ylabel("Length (Distance from Bowler)")
plt.title("Virat Kohli - Outside Off Stump")

plt.gca().invert_yaxis()
plt.xticks([-1, 0, 1], ["Off Stump", "Middle Stump", "Leg Stump"])
plt.yticks([])
handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color, markersize=10, label=length)
           for length, color in length_colors.items()]
plt.legend(handles=handles, loc="upper right")

plt.show()
```



Virat Kohli in ODIs has a different story for outside off-stump, as his weakness in this area has been widely discussed in Test matches. However, his ODI performance paints a contrasting picture:

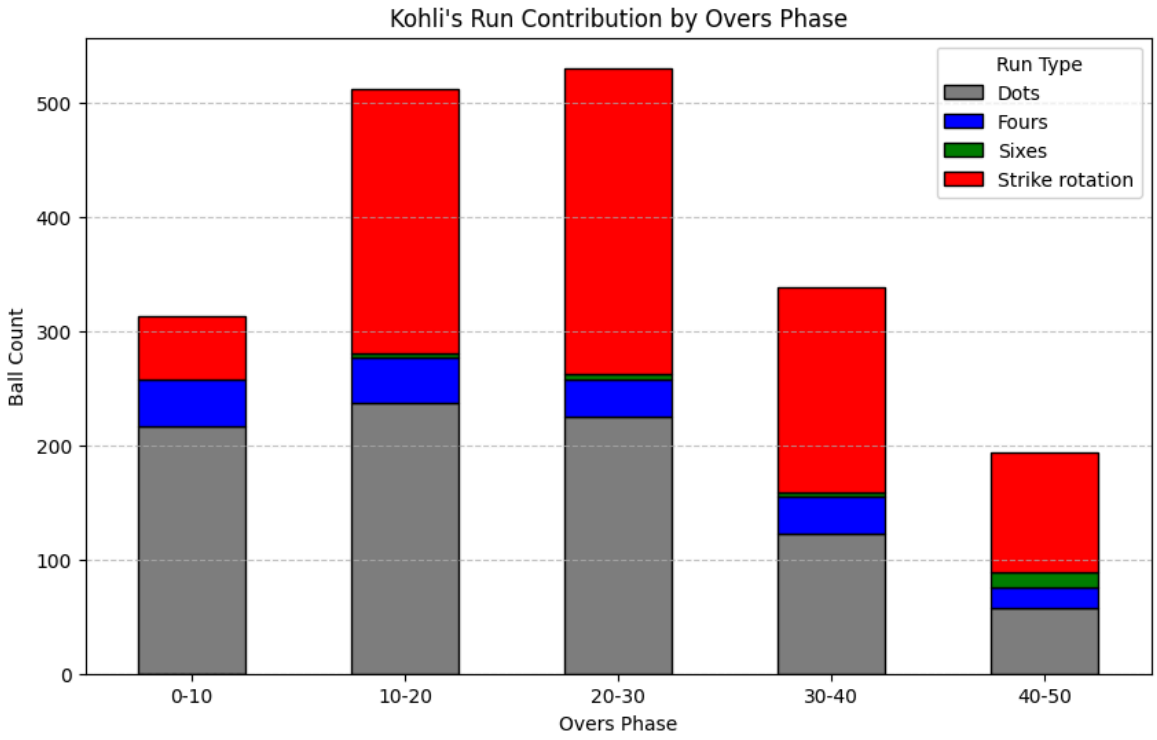
--> He has maintained a strong average against both fuller and good-length deliveries, showing his ability to handle this line effectively in limited-overs cricket.

--> His strike rate of 73.45 on good-length balls indicates a controlled and patient approach, often opting to leave or rotate the strike instead of taking unnecessary risks.

-->The higher strike rate on fuller balls suggests he is more aggressive when the ball is in his hitting zone, capitalizing on scoring opportunities.

In [35]:
```
bins=[0,10,20,30,40,50]
labels=['0-10','10-20','20-30','30-40','40-50']
kholi['Overs']=pd.cut(kholi['over'],bins=bins,labels=labels,right=False)
over_wise=kholi.groupby(['Overs','batruns']).size().reset_index(name='count')
over_wise['count'] = pd.to_numeric(over_wise['count'], errors='coerce')
over_wise['category'] = over_wise['batruns'].replace({1: 'Strike rotation', 2: 'Strike rotation', 3: 'Strike rotation',
                                        4: 'Fours', 6: 'Sixes', 0: 'Dots'})
grouped = over_wise.groupby(['Overs', 'category'])['count'].sum().unstack(fill_value=0)
grouped.plot(kind='bar', stacked=True, figsize=(10, 6),
             color=['gray', 'blue', 'green', 'red'], edgecolor='black')

plt.xlabel("Overs Phase")
plt.ylabel("Ball Count")
plt.title("Kohli's Run Contribution by Overs Phase")
plt.legend(title="Run Type")
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3346443526.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  kholi['Overs']=pd.cut(kholi['over'],bins=bins,labels=labels,right=False)
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3346443526.py:4: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  over_wise=kholi.groupby(['Overs','batruns']).size().reset_index(name='count')
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3346443526.py:8: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  grouped = over_wise.groupby(['Overs', 'category'])['count'].sum().unstack(fill_value=0)
```



How Kohli Builds His Innings in ODIs

Early Overs (0-10): Playing it Safe

Kohli starts cautiously, respecting the new ball. He faces a lot of dot balls, showing he isn't in a rush. Instead of taking unnecessary risks, he focuses on settling in.

Middle Overs (10-30): Master of Strike Rotation

This is where Kohli really shines! He keeps the scoreboard moving with quick singles and twos. You don't see too many big shots, but his ability to rotate strike ensures there's no pressure. Occasional boundaries show he's in control but not forcing anything.

Approaching the Death (30-40): Shifting Gears

Kohli starts to pick up pace but still keeps it smart. Fewer dot balls now, and he's more willing to take risks. He begins to find boundaries more often, setting up for a strong finish.

Death Overs (40-50): Finishing Strong

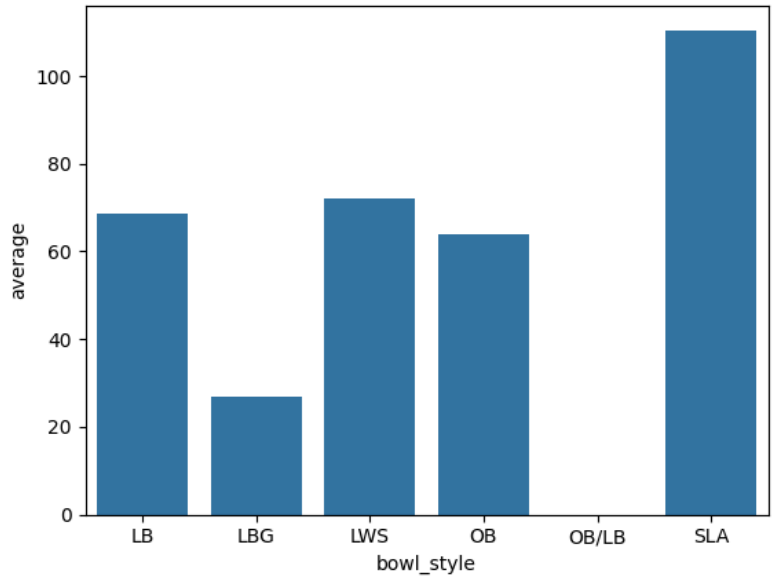
By now, Kohli has done his job – he either stays to finish or has laid the perfect platform. The dot balls are almost gone, replaced by quick runs and boundaries. You see some sixes, but he still prefers controlled aggression over reckless hitting.

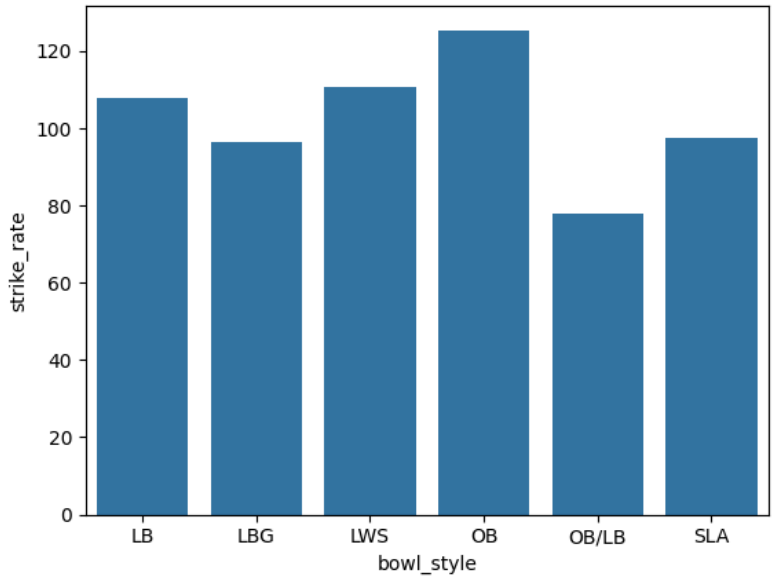
--> One important thing to note is, number of dot balls decreases as the game goes on , which shows how kholi keeps himself busy in the game.

In [36]:
```
batsman='Shreyas Iyer'
t=df[df['bat']==batsman]
t.head()

t_stats=t.groupby(['bowl_kind','bowl_style']).aggregate({'batruns':'sum','out':'sum','ballfaced':'sum','p_match':'nuniqu
t_stats_spin=t_stats[(t_stats['bowl_kind']=='spin bowler')]
t_stats_spin
t_stats_spin['average']=t_stats_spin['batruns']/t_stats_spin['out']
t_stats_spin['strike_rate']=(t_stats_spin['batruns']/t_stats_spin['ballfaced'])*100
display(t_stats_spin)

sns.barplot(data=t_stats_spin,x='bowl_style',y='average')
plt.show()
sns.barplot(data=t_stats_spin,x='bowl_style',y='strike_rate')
plt.show()
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3484837439.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  t_stats_spin['average']=t_stats_spin['batruns']/t_stats_spin['out']
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3484837439.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  t_stats_spin['strike_rate']=(t_stats_spin['batruns']/t_stats_spin['ballfaced'])*100
```

	bowl_kind	bowl_style	batruns	out	ballfaced	p_match	average	strike_rate
7	spin bowler	LB	137	2	127	13	68.50	107.874016
8	spin bowler	LBG	107	4	111	12	26.75	96.396396
9	spin bowler	LWS	72	1	65	6	72.00	110.769231
10	spin bowler	OB	256	4	204	21	64.00	125.490196
11	spin bowler	OB/LB	57	0	73	6	inf	78.082192
12	spin bowler	SLA	442	4	454	30	110.50	97.356828



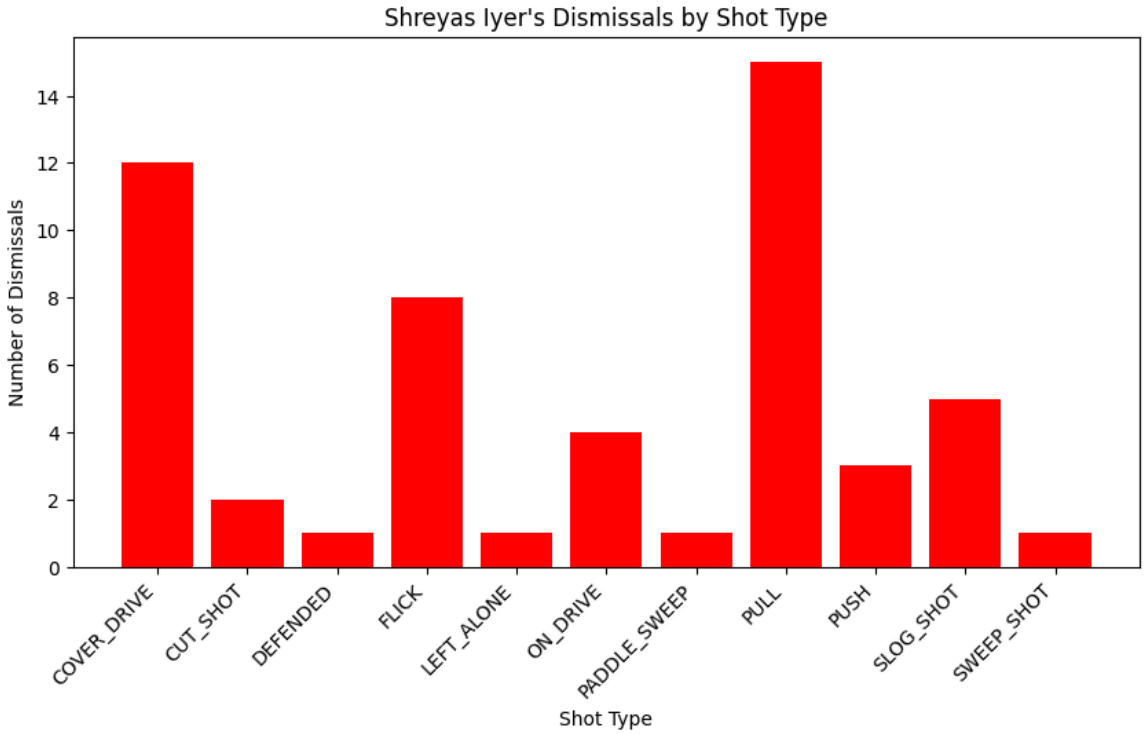


Shreyas Iyer has played a crucial role in strengthening India's middle order, solving the No. 4 issue.

His ability to handle spin effectively made him a key asset in the middle overs

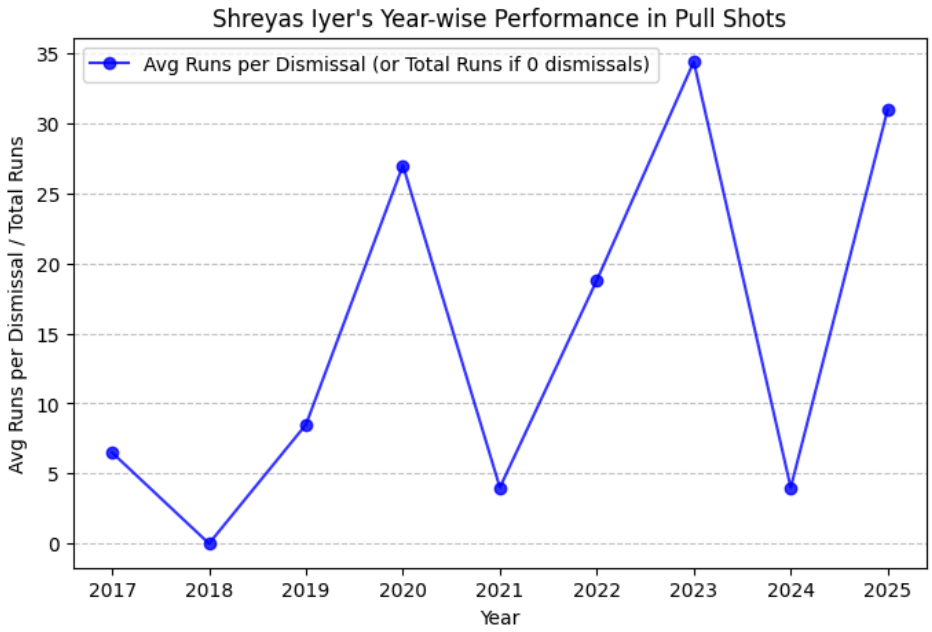
In [37]:
```
df_iyer_dismissals = df[(df['bat'] == "Shreyas Iyer") & (df['out'] == 1)]
dismissals_by_shot = df_iyer_dismissals.groupby('shot').size().reset_index(name='dismissal_count')
print("Shreyas Iyer's Dismissals by Shot Type:")
print(dismissals_by_shot)
plt.figure(figsize=(10, 5))
plt.bar(dismissals_by_shot['shot'], dismissals_by_shot['dismissal_count'], color='red')
plt.xlabel("Shot Type")
plt.ylabel("Number of Dismissals")
plt.title("Shreyas Iyer's Dismissals by Shot Type")
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
Shreyas Iyer's Dismissals by Shot Type:
            shot  dismissal_count
0    COVER_DRIVE               12
1       CUT_SHOT                2
2       DEFENDED                1
3          FLICK                8
4     LEFT_ALONE                1
5       ON_DRIVE                4
6   PADDLE_SWEEP                1
7           PULL               15
8           PUSH                3
9      SLOG_SHOT                5
10    SWEEP_SHOT                1
```



In [38]:
```
import pandas as pd
import matplotlib.pyplot as plt
df_iyer = df[df['bat'] == "Shreyas Iyer"]
pull_shots = ["PULL_HOOK_ON_BACK_FOOT", "PULL_HOOK_ON_FRONT_FOOT", "PULL"]
df_pull_shots = df_iyer[df_iyer['shot'].isin(pull_shots)]
yearwise_pull_stats = df_pull_shots.groupby('year').agg(
    total_runs=('batruns', 'sum'),
    total_dismissals=('out', 'sum')
).reset_index()
yearwise_pull_stats['avg_or_total_runs'] = yearwise_pull_stats.apply(
    lambda row: row['total_runs'] if row['total_dismissals'] == 0 else row['total_runs'] / row['total_dismissals'],
    axis=1
)
print("Year-wise Average Runs Scored by Shreyas Iyer While Playing Pull Shots:")
print(yearwise_pull_stats[['year', 'avg_or_total_runs']])

plt.figure(figsize=(8, 5))
plt.plot(yearwise_pull_stats['year'].astype(str), yearwise_pull_stats['avg_or_total_runs'],
         marker='o', linestyle='-', color='blue', alpha=0.8, label="Avg Runs per Dismissal (or Total Runs if 0 dismissal


plt.xlabel("Year")
plt.ylabel("Avg Runs per Dismissal / Total Runs")
plt.title("Shreyas Iyer's Year-wise Performance in Pull Shots")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend()
plt.show()
```

```
Year-wise Average Runs Scored by Shreyas Iyer While Playing Pull Shots:
   year  avg_or_total_runs
0  2017                6.5
1  2018                0.0
2  2019                8.5
3  2020               27.0
4  2021                4.0
5  2022               18.8
6  2023               34.4
7  2024                4.0
8  2025               31.0
```



Initially, he struggled against short-pitched deliveries, leading to poor returns on pull shots.

After 2022, he significantly improved his short-ball game, scoring more effectively off short-length deliveries.

This adjustment has enhanced his overall performance against pace attacks, making him a more reliable middle-order batsman.

Note: In 2024, India played only three ODI matches.

In [39]:
```
ind=df[(df['year']>2019)& ((df['team_bowl']=='India')) ]
ind=ind[ind['bowl'].isin(['Kuldeep Yadav','Ravindra Jadeja', 'Mohammed Shami','Hardik Pandya','Axar Patel','Washington S
       'Varun Chakravarthy'])]
ind.sort_values(by='bowl', ascending=True, inplace=True)
```

In [40]:
```
#pd.set_option('display.max_column',None)
import plotly.graph_objects as go
pre_final=ind.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wickets=
final=pre_final.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wickets','
final['Economy']=(final['Runs']/final['Overs']).round(2)
final['Bowling_Average']=(final['Runs']/final['Wickets']).round(2)


def bowl_avg(Bowling_Average):
    return 'NIL' if np.isinf(Bowling_Average) else Bowling_Average
final['Bowling_Average']=final['Bowling_Average'].apply(bowl_avg)

final=final.sort_values(by='Wickets',ascending=False)

ind=ind[ind['bowl_kind']!='mixture/unknown']
int_part = final['Overs'].astype(int)
dec_part = (final['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
final['Overs']=int_part+rem_ball
final['Strike_Rate']=(balls/final['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 'NIL' if np.isinf(Strike_Rate) else Strike_Rate
final['Strike_Rate']=final['Strike_Rate'].apply(str_rate)

final.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled':'
final
```

Out[40]:

	Bowler	Innings Bowled	Runs	Wickets	MWI	Overs	Economy	Bowling Average	Strike Rate
4	Kuldeep Yadav	51	2108	77	5	419.5	5.03	27.38	32.71
5	Mohammed Shami	30	1322	68	7	227.5	5.81	19.44	20.10
6	Ravindra Jadeja	36	1445	48	5	305.1	4.74	30.10	38.15
2	Hardik Pandya	30	893	34	4	157.3	5.69	26.26	27.79
8	Washington Sundar	19	604	23	3	128.0	4.72	26.26	33.39
1	Axar Patel	23	767	23	3	164.5	4.66	33.35	43.00
0	Arshdeep Singh	8	322	16	5	62.1	5.19	20.12	23.31
3	Harshit Rana	3	146	7	3	21.0	6.95	20.86	18.00
7	Varun Chakravarthy	1	54	1	1	10.0	5.40	54.00	60.00

In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
plt.figure(figsize=(15,10))
fig, axes = plt.subplots(1, 4, figsize=(18, 5), sharey=True)
sns.barplot(data=final, x='Economy', y='Bowler', ax=axes[1], palette='Blues_r')
axes[1].set_title("Economy Rate")
axes[1].set_xlabel("Economy")
sns.barplot(data=final, x='Bowling Average', y='Bowler', ax=axes[2], palette='Greens_r')
axes[2].set_title("Bowling Average")
axes[2].set_xlabel("Bowling Average")
sns.barplot(data=final, x='Strike Rate', y='Bowler', ax=axes[3], palette='Reds_r')
axes[3].set_title("Strike Rate")
axes[3].set_xlabel("Strike Rate")
sns.barplot(data=final, x='Wickets', y='Bowler', ax=axes[0], palette='Purples_r')
axes[0].set_title("Wickets Taken")
axes[0].set_xlabel("Wickets")
plt.tight_layout()
plt.show()
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2379886890.py:6: FutureWarning:

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

  sns.barplot(data=final, x='Economy', y='Bowler', ax=axes[1], palette='Blues_r')
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2379886890.py:9: FutureWarning:

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

  sns.barplot(data=final, x='Bowling Average', y='Bowler', ax=axes[2], palette='Greens_r')
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2379886890.py:12: FutureWarning:

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

  sns.barplot(data=final, x='Strike Rate', y='Bowler', ax=axes[3], palette='Reds_r')
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2379886890.py:15: FutureWarning:
```

Insights:

Since 2020, Kuldeep Yadav and Mohammed Shami have been India's top wicket-takers.
Among the bowlers, Axar Patel, Ravindra Jadeja, and Washington Sundar have maintained a low economy rate, keeping things tight in the middle overs.
When it comes to bowling average, Shami and Washington Sundar stand out, meaning they concede fewer runs per wicket.
Meanwhile, Shami and Hardik Pandya have had the best strike rates, indicating they pick up wickets more frequently.

Advantages:

Indian spinners are highly economical compared to the pacers, which is a great sign, especially in Dubai, where pitches are generally on the slower side.
Pacers have a lower bowling average and strike rate compared to spinners, which is another positive sign for Dubai conditions. Despite the slow pitch, pacers have still been successful in taking wickets, which could be crucial in big games.

Disadvantages:

While the spinners have kept things tight with their economy rate, they haven't been able to pick up wickets consistently. This could become a concern if the fast bowlers fail to strike early in the powerplay.
Some pacers, like Hardik Pandya (5.93) and Harshit Rana (7.46), have been expensive. Conceding crucial runs, especially against top teams like Australia and New Zealand, could put India under pressure in high-stakes matches.

Conclusion:

Overall, India's bowling attack looks solid and well-suited for Dubai conditions. The fast bowlers are taking wickets, and the spinners are controlling the run flow, which is an ideal combination. However, there are a few concerns:

- Mohammed Shami is coming off a long injury break, and his form will be a key factor.
- Some pacers have high economy rates, which could prove costly in close games.
- The spinners' lack of wickets might hurt India, especially if the pacers struggle to provide early breakthroughs.

India has the resources, but they need to ensure a balance between containing runs and taking wickets to dominate in Dubai.

In [42]:
```
a=ind[ind['out']==True].groupby(['bowl','bowl_kind']).agg(Total_Wickets=('out','sum'),Matches_Played=('p_match','nunique
```

In [43]:
```
c=a.pop('Matches_Played')
a.insert(1,'Matches_Played',c)
a
```

Out[43]:

	bowl	Matches_Played	bowl_kind	Total_Wickets
0	Arshdeep Singh	6	pace bowler	16
1	Axar Patel	15	spin bowler	23
2	Hardik Pandya	22	pace bowler	34
3	Harshit Rana	3	pace bowler	7
4	Kuldeep Yadav	41	spin bowler	77
5	Mohammed Shami	24	pace bowler	68
6	Ravindra Jadeja	23	spin bowler	48
7	Varun Chakravarthy	1	spin bowler	1
8	Washington Sundar	13	spin bowler	23

In [44]:
```
pd.set_option('display.max_rows',None)
ind.head()
```

Out[44]:

	p_match	inns	bat	p_bat	team_bat	bowl	p_bowl	team_bowl	ball	ball_id	...	gmt_offset	wagonX	wagonY	wagonZone
1195681	1387600	1	Andile Phehlukwayo	540316	South Africa	Arshdeep Singh	1125976	India	5	13.05	...	0.0	0.0	0.0	0.0
1194277	1387602	2	Heinrich Klaasen	436757	South Africa	Arshdeep Singh	1125976	India	6	31.06	...	0.0	312.0	275.0	3.0
1194276	1387602	2	Heinrich Klaasen	436757	South Africa	Arshdeep Singh	1125976	India	5	31.05	...	0.0	199.0	195.0	4.0
1194275	1387602	2	Heinrich Klaasen	436757	South Africa	Arshdeep Singh	1125976	India	4	31.04	...	0.0	0.0	0.0	0.0
1111998	1322280	2	Finn Allen	959759	New Zealand	Arshdeep Singh	1125976	India	6	7.06	...	0.0	0.0	0.0	0.0

5 rows × 60 columns

In [45]:
```
#The Ovrrall Strike (i is defined here)
ovr=ind[(ind['out']!='mixture/unknown')&(ind['year']>2019)]


pre_a=ovr.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wickets=('ou
a=pre_a.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wickets','sum'),Ov
a['Economy']=(a['Runs']/a['Overs']).round(2)
a['Bowling_Average']=(a['Runs']/a['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
a['Bowling_Average']=a['Bowling_Average'].apply(bowl_avg)

a=a.sort_values(by='Wickets',ascending=False)


int_part = a['Overs'].astype(int)
dec_part = (a['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
a['Overs']=int_part+rem_ball
a['Strike_Rate']=(balls/a['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
a['Strike_Rate']=a['Strike_Rate'].apply(str_rate)

a.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled':'Inni

a
```

Out[45]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
4	Kuldeep Yadav	51	2108	77	419.5	5.03	27.38	32.71
5	Mohammed Shami	30	1322	68	227.5	5.81	19.44	20.10
6	Ravindra Jadeja	36	1445	48	305.1	4.74	30.10	38.15
2	Hardik Pandya	30	893	34	157.3	5.69	26.26	27.79
8	Washington Sundar	19	604	23	128.0	4.72	26.26	33.39
1	Axar Patel	23	767	23	164.5	4.66	33.35	43.00
0	Arshdeep Singh	8	322	16	62.1	5.19	20.12	23.31
3	Harshit Rana	3	146	7	21.0	6.95	20.86	18.00
7	Varun Chakravarthy	1	54	1	10.0	5.40	54.00	60.00

In [46]:
```
i=a[['Bowler','Wickets']]
i
```

Out[46]:

	Bowler	Wickets
4	Kuldeep Yadav	77
5	Mohammed Shami	68
6	Ravindra Jadeja	48
2	Hardik Pandya	34
8	Washington Sundar	23
1	Axar Patel	23
0	Arshdeep Singh	16
3	Harshit Rana	7
7	Varun Chakravarthy	1

In [47]:
```
#Middle Overs
mid=ind[(ind['over']>10)&(ind['over']<41)&(ind['out']!='mixture/unknown')&(ind['year']>2019)]


pre_final_2=mid.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wicket
final_2=pre_final_2.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wicket
final_2['Economy']=(final_2['Runs']/final_2['Overs']).round(2)
final_2['Bowling_Average']=(final_2['Runs']/final_2['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
final_2['Bowling_Average']=final_2['Bowling_Average'].apply(bowl_avg)

final_2=final_2.sort_values(by='Wickets',ascending=False)

mid=mid[mid['bowl_kind']!='mixture/unknown']
int_part = final_2['Overs'].astype(int)
dec_part = (final_2['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
final_2['Overs']=int_part+rem_ball
final_2['Strike_Rate']=(balls/final_2['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
final_2['Strike_Rate']=final_2['Strike_Rate'].apply(str_rate)
final_2.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled'
final_2['Proportion_mid_overs']=(final_2['Wickets']*100/i['Wickets']).round(2)
def prop(Proportion_mid_overs):
    return 0 if np.isinf(Proportion_mid_overs) or np.isnan(Proportion_mid_overs) else Proportion_mid_overs
final_2['Proportion_mid_overs']=final_2['Proportion_mid_overs'].apply(prop)
Wash_total_wickets = i.loc[i['Bowler'] == 'Washington Sundar', 'Wickets']
washington_wickets=pd.DataFrame()
washington_wickets['Wickets'] = final_2.loc[final_2['Bowler'] == 'Washington Sundar', 'Wickets']
val = (100 * washington_wickets['Wickets'] / Wash_total_wickets).round(2)
final_2.loc[final_2['Bowler'] == 'Washington Sundar', 'Proportion_mid_overs'] = val
final_2['Proportion_mid_overs'] = final_2['Proportion_mid_overs'].apply(lambda x: 0 if np.isnan(x) else x)
final_2
```

Out[47]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate	Proportion_mid_overs
4	Kuldeep Yadav	50	1903	65	385.2	4.95	29.28	35.57	84.42
6	Ravindra Jadeja	36	1355	40	285.0	4.76	33.88	42.75	83.33
5	Mohammed Shami	29	996	25	188.4	5.29	39.84	45.28	36.76
2	Hardik Pandya	26	699	22	127.0	5.52	31.77	34.64	64.71
8	Washington Sundar	19	572	20	121.0	4.73	28.60	36.30	86.96
1	Axar Patel	23	669	20	147.2	4.54	33.45	44.20	86.96
3	Harshit Rana	3	125	4	19.0	6.58	31.25	28.50	57.14
0	Arshdeep Singh	6	196	3	43.0	4.56	65.33	86.00	18.75
7	Varun Chakravarthy	1	39	1	8.0	4.88	39.00	48.00	100.00

In [48]:
```
g=mid
g=g[(g['outcome']!='wide')&(g['outcome']!='no ball')]
g.groupby('bowl')['over'].count()
```

Out[48]:
```
bowl
Arshdeep Singh         114
Axar Patel             795
Hardik Pandya          594
Harshit Rana            66
Kuldeep Yadav         2313
Mohammed Shami         514
Ravindra Jadeja       1704
Varun Chakravarthy      43
Washington Sundar      649
Name: over, dtype: int64
```

In [49]:
```
#The Powerplay Strike
powe=ind[(ind['over']<11)&(ind['out']!='mixture/unknown')&(ind['year']>2019)]


pre_final_1=powe.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wicke
final_1=pre_final_1.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wicket
final_1['Economy']=(final_1['Runs']/final_1['Overs']).round(2)
final_1['Bowling_Average']=(final_1['Runs']/final_1['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
final_1['Bowling_Average']=final_1['Bowling_Average'].apply(bowl_avg)

final_1=final_1.sort_values(by='Wickets',ascending=False)


int_part = final_1['Overs'].astype(int)
dec_part = (final_1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
final_1['Overs']=int_part+rem_ball
final_1['Strike_Rate']=(balls/final_1['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
final_1['Strike_Rate']=final_1['Strike_Rate'].apply(str_rate)

final_1.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled'
final_1['Proportion_pow_overs']=100*(final_1['Wickets']/i['Wickets']).round(2)
def prop(Proportion_pow_overs):
    return 0 if np.isinf(Proportion_pow_overs) or np.isnan(Proportion_pow_overs) else Proportion_pow_overs
final_1['Proportion_pow_overs']=final_1['Proportion_pow_overs'].apply(prop)
final_1
washington_wickets = final_1.loc[final_1['Bowler'] == 'Washington Sundar', 'Wickets'].values[0]
val = (100 * (washington_wickets/i.loc[i['Bowler'] == 'Washington Sundar', 'Wickets']))
final_1.loc[final_1['Bowler'] == 'Washington Sundar', 'Proportion_pow_overs'] = val
Wash_total_wickets = i.loc[i['Bowler'] == 'Washington Sundar', 'Wickets']
washington_wickets=pd.DataFrame()
washington_wickets['Wickets'] = final_1.loc[final_1['Bowler'] == 'Washington Sundar', 'Wickets']
val = (100 * washington_wickets['Wickets'] / Wash_total_wickets).round(2)
final_1.loc[final_1['Bowler'] == 'Washington Sundar', 'Proportion_pow_overs'] = val
final_1['Proportion_pow_overs'] = final_1['Proportion_pow_overs'].apply(lambda x: 0 if np.isnan(x) else x)
final_1
```

Out[49]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate	Proportion_pow_overs
5	Mohammed Shami	30	559	18	106.0	5.27	31.06	35.33	26.00
0	Arshdeep Singh	8	160	7	33.0	4.85	22.86	28.29	44.00
2	Hardik Pandya	22	205	6	36.3	5.65	34.17	36.50	18.00
3	Harshit Rana	3	75	2	8.0	9.38	37.50	24.00	29.00
1	Axar Patel	9	74	1	15.0	4.93	74.00	90.00	4.00
8	Washington Sundar	7	55	1	13.0	4.23	55.00	78.00	4.35
6	Ravindra Jadeja	2	12	1	2.0	6.00	12.00	12.00	2.00
4	Kuldeep Yadav	1	12	0	1.0	12.00	0.00	0.00	0.00
7	Varun Chakravarthy	1	4	0	1.0	4.00	0.00	0.00	0.00

In [50]:
```
#Deather Overs
deathe=ind[(ind['over']>40)&(ind['out']!='mixture/unknown')&(ind['year']>2019)]
pre_final_3=deathe.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wic
final_3=pre_final_3.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wicket
final_3['Economy']=(final_3['Runs']/final_3['Overs']).round(2)
final_3['Bowling_Average']=(final_3['Runs']/final_3['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
final_3['Bowling_Average']=final_3['Bowling_Average'].apply(bowl_avg)

final_3=final_3.sort_values(by='Wickets',ascending=False)


int_part = final_3['Overs'].astype(int)
dec_part = (final_3['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
final_3['Overs']=int_part+rem_ball
final_3['Strike_Rate']=(balls/final_3['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
final_3['Strike_Rate']=final_3['Strike_Rate'].apply(str_rate)

final_3.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled'
final_3['Proportion_Death_overs']=100*(final_3['Wickets']/i['Wickets']).round(2)
def prop(Proportion_Death_overs):
    return 0 if np.isinf(Proportion_Death_overs) or np.isnan(Proportion_Death_overs) else Proportion_Death_overs
final_3['Proportion_Death_overs']=final_3['Proportion_Death_overs'].apply(prop)
Wash_total_wickets = i.loc[i['Bowler'] == 'Washington Sundar', 'Wickets']
washington_wickets=pd.DataFrame()
washington_wickets['Wickets'] = final_3.loc[final_3['Bowler'] == 'Washington Sundar', 'Wickets']
val = (100 * washington_wickets['Wickets'] / Wash_total_wickets).round(2)
final_3.loc[final_3['Bowler'] == 'Washington Sundar', 'Proportion_Death_overs'] = val
final_3['Proportion_Death_overs'] = final_3['Proportion_Death_overs'].apply(lambda x: 0 if np.isnan(x) else x)
final_3
```

Out[50]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate	Proportion_Death_overs
5	Mohammed Shami	17	924	25	149.1	6.20	36.96	35.80	37.0
4	Kuldeep Yadav	21	1095	12	202.1	5.43	91.25	101.08	16.0
6	Ravindra Jadeja	12	498	7	109.5	4.55	71.14	94.14	15.0
0	Arshdeep Singh	4	203	6	34.1	5.95	33.83	34.17	38.0
2	Hardik Pandya	12	448	6	75.0	5.97	74.67	75.00	18.0
8	Washington Sundar	4	163	2	31.0	5.26	81.50	93.00	8.7
1	Axar Patel	13	522	2	110.3	4.73	261.00	331.50	9.0
3	Harshit Rana	1	62	1	9.0	6.89	62.00	54.00	14.0
7	Varun Chakravarthy	1	54	0	10.0	5.40	0.00	0.00	0.0

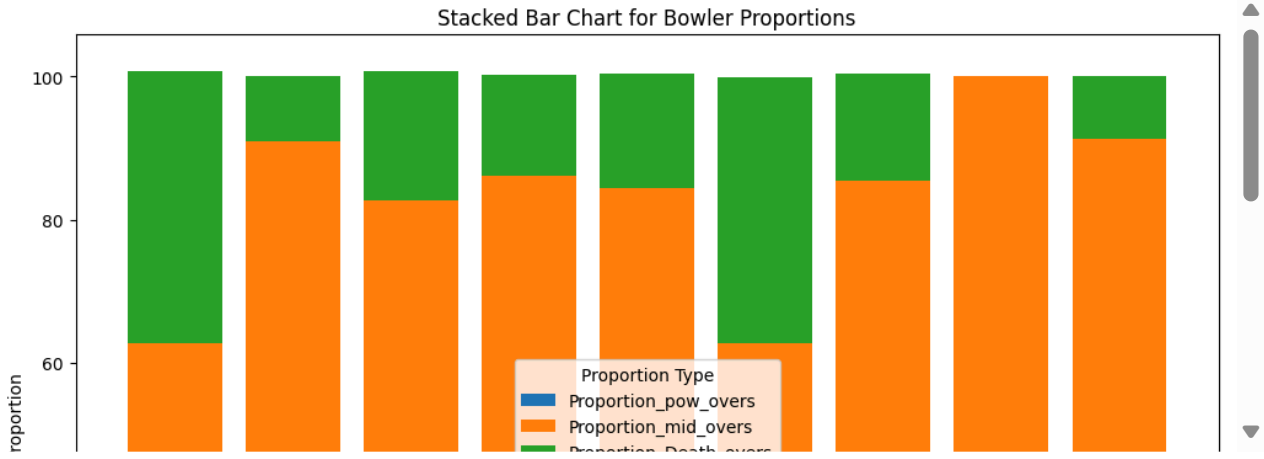
In [51]:
```
merged = pd.DataFrame()
merged_df = (
    final_1[['Bowler', 'Proportion_pow_overs']]
    .merge(final_2[['Bowler', 'Proportion_mid_overs']], on='Bowler', how='outer').merge(final_3[['Bowler', 'Proportion_D
merged_df.fillna(0, inplace=True)
merged_df.sort_values(by='Bowler', inplace=True)
merged_df.head()

if merged_df.shape[1] > 12:
    merged_df = merged_df.iloc[:, 12:]

print(merged_df.columns)
merged_df['Total']=merged_df.iloc[:,1]+merged_df.iloc[:,2]+merged_df.iloc[:,3]
merged_df
```

```
Index(['Bowler', 'Proportion_pow_overs', 'Proportion_mid_overs',
       'Proportion_Death_overs'],
      dtype='object')
```

Out[51]:

	Bowler	Proportion_pow_overs	Proportion_mid_overs	Proportion_Death_overs	Total
0	Arshdeep Singh	44.00	18.75	38.0	100.75
1	Axar Patel	4.00	86.96	9.0	99.96
2	Hardik Pandya	18.00	64.71	18.0	100.71
3	Harshit Rana	29.00	57.14	14.0	100.14
4	Kuldeep Yadav	0.00	84.42	16.0	100.42
5	Mohammed Shami	26.00	36.76	37.0	99.76
6	Ravindra Jadeja	2.00	83.33	15.0	100.33
7	Varun Chakravarthy	0.00	100.00	0.0	100.00
8	Washington Sundar	4.35	86.96	8.7	100.01

In [52]:
```
grp = merged_df.iloc[:, 1:4]
grp['Bowlers']=merged_df['Bowler']
plt.figure(figsize=(12, 8))
bottom = np.zeros(len(merged_df))
for i in grp.columns:
    if grp[i].dtype!='O':
        plt.bar(merged_df['Bowler'], grp[i], label=i, bottom=bottom)
        bottom += grp[i].values
plt.xlabel("Bowler")
plt.ylabel("Proportion")
plt.title("Stacked Bar Chart for Bowler Proportions")
plt.xticks(rotation=90)
plt.legend(title="Proportion Type",loc='center')

plt.show()
```



Insights:

Mohammed Shami has Almost taken equal wickets in all 3 phases. Arshdeep is so good in Power play and Death Overs.but not effective in Middle Overs. Indian Spinners have taken most of their wickets in Middle overs.

Conclusion:

As the Average 1st Inning in Dubai is 246 (Day-Night Matches),which is not high scoring when compared to pakistan wickets,the team which bowls better in 15 to 40 have high win chances.As indian spinners were so effective in middle overs,India have an edge

In [53]:
```
#Mid_1 Overs
mid_1=ind[(ind['over']>10)&(ind['over']<18)&(ind['out']!='mixture/unknown')&(ind['year']>2019)]


pre_s1=mid_1.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wickets=(
s1=pre_s1.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wickets','sum'),
s1['Economy']=(s1['Runs']/s1['Overs']).round(2)
s1['Bowling_Average']=(s1['Runs']/s1['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
s1['Bowling_Average']=s1['Bowling_Average'].apply(bowl_avg)

s1=s1.sort_values(by='Wickets',ascending=False)


int_part = s1['Overs'].astype(int)
dec_part = (s1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
s1['Overs']=int_part+rem_ball
s1['Strike_Rate']=(balls/s1['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
s1['Strike_Rate']=s1['Strike_Rate'].apply(str_rate)
s1.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled':'Inn
s1['Bowler']=s1['Bowler'].sort_values(ascending=False)
s1
```

Out[53]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
4	Kuldeep Yadav	36	414	14	88.0	4.70	29.57	37.71
2	Hardik Pandya	22	329	11	65.2	5.05	29.91	35.64
6	Ravindra Jadeja	26	271	7	56.0	4.84	38.71	48.00
5	Mohammed Shami	11	205	6	51.0	4.02	34.17	51.00
1	Axar Patel	15	211	5	46.0	4.59	42.20	55.20
8	Washington Sundar	9	120	4	31.0	3.87	30.00	46.50
7	Varun Chakravarthy	1	18	1	5.0	3.60	18.00	30.00
0	Arshdeep Singh	2	46	0	12.0	3.83	0.00	0.00
3	Harshit Rana	2	68	0	9.0	7.56	0.00	0.00

In [54]:
```
#Mid_2 Overs
mid_2=ind[(ind['over']>17)&(ind['over']<26)&(ind['out']!='mixture/unknown')&(ind['year']>2019)]


pre_s2=mid_2.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wickets=(
s2=pre_s2.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wickets','sum'),
s2['Economy']=(s2['Runs']/s2['Overs']).round(2)
s2['Bowling_Average']=(s2['Runs']/s2['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
s2['Bowling_Average']=s2['Bowling_Average'].apply(bowl_avg)

s2=s2.sort_values(by='Wickets',ascending=False)


int_part = s2['Overs'].astype(int)
dec_part = (s2['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
s2['Overs']=int_part+rem_ball
s2['Strike_Rate']=(balls/s2['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
s2['Strike_Rate']=s2['Strike_Rate'].apply(str_rate)
s2.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled':'Inn
s2['Bowler']=s2['Bowler'].sort_values(ascending=False)

s2
```

Out[54]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
4	Kuldeep Yadav	49	1084	23	233.0	4.65	47.13	60.78
6	Ravindra Jadeja	34	773	12	166.4	4.65	64.42	83.33
5	Mohammed Shami	14	392	8	82.0	4.78	49.00	61.50
8	Washington Sundar	13	227	6	56.0	4.05	37.83	56.00
1	Axar Patel	18	350	6	86.0	4.07	58.33	86.00
3	Harshit Rana	1	22	1	3.0	7.33	22.00	18.00
0	Arshdeep Singh	3	83	0	18.0	4.61	0.00	0.00
2	Hardik Pandya	2	39	0	9.0	4.33	0.00	0.00
7	Varun Chakravarthy	1	26	0	6.0	4.33	0.00	0.00

In [55]:
```
#Mid_3 Overs
mid_3=ind[(ind['over']>25)&(ind['over']<33)&(ind['out']!='mixture/unknown')&(ind['year']>2019)]


pre_s3=mid_3.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wickets=(
s3=pre_s3.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wickets','sum'),
s3['Economy']=(s3['Runs']/s3['Overs']).round(2)
s3['Bowling_Average']=(s3['Runs']/s3['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
s3['Bowling_Average']=s3['Bowling_Average'].apply(bowl_avg)

s3=s3.sort_values(by='Wickets',ascending=False)


int_part = s3['Overs'].astype(int)
dec_part = (s3['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
s3['Overs']=int_part+rem_ball
s3['Strike_Rate']=(balls/s3['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
s3['Strike_Rate']=s3['Strike_Rate'].apply(str_rate)
s3.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled':'Inn
s3.sort_values(by='Bowler', ascending=True, inplace=True)
s3
```

Out[55]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	Arshdeep Singh	5	157	3	36.0	4.36	52.33	72.00
1	Axar Patel	14	387	4	83.0	4.66	96.75	124.50
2	Hardik Pandya	8	206	4	39.0	5.28	51.50	58.50
3	Harshit Rana	2	72	2	12.0	6.00	36.00	36.00
4	Kuldeep Yadav	38	1278	14	256.5	4.98	91.29	110.07
5	Mohammed Shami	11	341	5	68.0	5.01	68.20	81.60
6	Ravindra Jadeja	32	1042	16	226.2	4.61	65.12	84.88
7	Varun Chakravarthy	1	33	0	7.0	4.71	0.00	0.00
8	Washington Sundar	16	410	5	84.0	4.88	82.00	100.80

In [56]:
```
#Mid_4 Overs
mid_4=ind[(ind['over']>32)&(ind['over']<41)&(ind['out']!='mixture/unknown')&(ind['year']>2019)]


pre_s4=mid_4.groupby(['bowl','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'),Wickets=(
s4=pre_s4.groupby(['bowl']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets=('Wickets','sum'),
s4['Economy']=(s4['Runs']/s4['Overs']).round(2)
s4['Bowling_Average']=(s4['Runs']/s4['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
s4['Bowling_Average']=s4['Bowling_Average'].apply(bowl_avg)

s4=s4.sort_values(by='Wickets',ascending=False)


int_part = s4['Overs'].astype(int)
dec_part = (s4['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
s4['Overs']=int_part+rem_ball
s4['Strike_Rate']=(balls/s4['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
s4['Strike_Rate']=s4['Strike_Rate'].apply(str_rate)
s4.rename(columns={'bowl':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_Bowled':'Inn
s4
```

Out[56]:

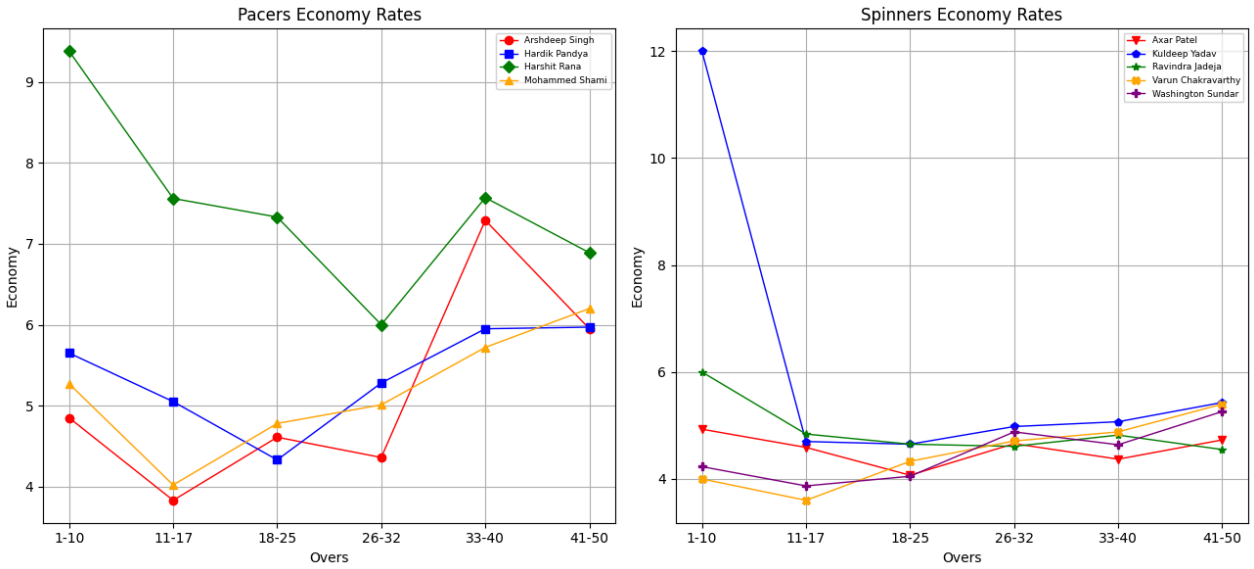
	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
4	Kuldeep Yadav	34	1496	14	295.3	5.07	106.86	126.64
2	Hardik Pandya	17	532	7	89.4	5.95	76.00	76.86
5	Mohammed Shami	16	632	6	110.4	5.72	105.33	110.67
8	Washington Sundar	12	427	5	92.0	4.64	85.40	110.40
1	Axar Patel	13	416	5	95.2	4.37	83.20	114.40
6	Ravindra Jadeja	20	877	5	182.0	4.82	175.40	218.40
3	Harshit Rana	1	53	1	7.0	7.57	53.00	42.00
0	Arshdeep Singh	1	51	0	7.0	7.29	0.00	0.00
7	Varun Chakravarthy	1	39	0	8.0	4.88	0.00	0.00

In [57]:
```
economy = (
    final_1[['Bowler', 'Economy']].rename(columns={'Economy': 'Economy_1-10'})
    .merge(s1[['Bowler', 'Economy']].rename(columns={'Economy': 'Economy_11-17'}), on='Bowler', how='outer')
    .merge(s2[['Bowler', 'Economy']].rename(columns={'Economy': 'Economy_18-25'}), on='Bowler', how='outer')
    .merge(s3[['Bowler', 'Economy']].rename(columns={'Economy': 'Economy_26-32'}), on='Bowler', how='outer')
    .merge(s4[['Bowler', 'Economy']].rename(columns={'Economy': 'Economy_33-40'}), on='Bowler', how='outer')
    .merge(final_3[['Bowler', 'Economy']].rename(columns={'Economy': 'Economy_41-50'}), on='Bowler', how='outer')
)

economy.fillna(0, inplace=True)
economy.sort_values(by='Bowler', inplace=True)

economy
```

Out[57]:

	Bowler	Economy_1-10	Economy_11-17	Economy_18-25	Economy_26-32	Economy_33-40	Economy_41-50
0	Arshdeep Singh	4.85	3.83	4.61	4.36	7.29	5.95
1	Axar Patel	4.93	4.59	4.07	4.66	4.37	4.73
2	Hardik Pandya	5.65	5.05	4.33	5.28	5.95	5.97
3	Harshit Rana	9.38	7.56	7.33	6.00	7.57	6.89
4	Kuldeep Yadav	12.00	4.70	4.65	4.98	5.07	5.43
5	Mohammed Shami	5.27	4.02	4.78	5.01	5.72	6.20
6	Ravindra Jadeja	6.00	4.84	4.65	4.61	4.82	4.55
7	Varun Chakravarthy	4.00	3.60	4.33	4.71	4.88	5.40
8	Washington Sundar	4.23	3.87	4.05	4.88	4.64	5.26

In [58]:
```
import matplotlib.pyplot as plt
pa = economy.set_index('Bowler').T
pacers = ["Arshdeep Singh", "Hardik Pandya", "Harshit Rana", "Mohammed Shami"]
spinners = ["Axar Patel", "Kuldeep Yadav", "Ravindra Jadeja", "Varun Chakravarthy", "Washington Sundar"]
pa_pacers = pa.loc[:, pa.columns.isin(pacers)]
pa_spinners = pa.loc[:, pa.columns.isin(spinners)]
colors = ["Red", "Blue", "Green", "Orange", "Purple", "Brown", "Pink", "Cyan", "Magenta", "Yellow"]
markers1 = ["o", "s", "D", "^"]
markers2 = ["v", "p", "*", "X", "P", "h"]
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 6))
for i, bowler in enumerate(pa_pacers.columns):
    ax1.plot(['1-10','11-17','18-25','26-32','33-40','41-50'], pa_pacers[bowler], color=colors[i % len(colors)], marker=

ax1.set_xlabel('Overs')
ax1.set_ylabel('Economy')
ax1.set_title('Pacers Economy Rates')
ax1.legend(loc='upper right', fontsize=6.5)
ax1.grid(True)
for i, bowler in enumerate(pa_spinners.columns):
    ax2.plot(['1-10','11-17','18-25','26-32','33-40','41-50'], pa_spinners[bowler], color=colors[i % len(colors)], marke
ax2.set_xlabel('Overs')
ax2.set_ylabel('Economy')
ax2.set_title('Spinners Economy Rates')
ax2.legend(loc='upper right', fontsize=6.5)
ax2.grid(True)

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



Insights:

Pacers struggled early (Harshit Rana 9+ economy) but recovered in middle overs. Spinners were stable, except Kuldeep 12 economy in powerplay. (this won't be a concern as kuldeep mostly bowls in middle overs) Washington Sundar was ultra-economical in 26-32 overs. Death overs (41-50) were controlled for both pacers & spinners.

Conclusion:

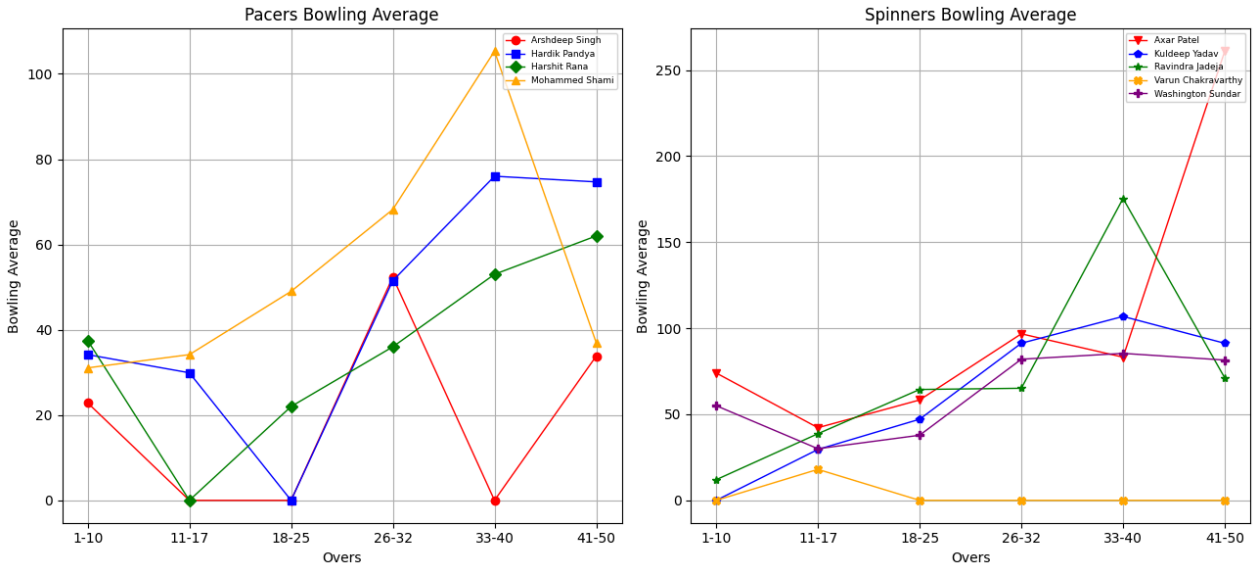
Considering the slow surface in dubai ,Pacers need better powerplay control, spinners dominate middle overs.from the previous insights both hardik and shami have proven to be best options in both powerplay and death,as they tend to concede less runs and take wickets compared to other fast bowlers

In [59]:
```
ba = (
    final_1[['Bowler', 'Bowling Average']].rename(columns={'Bowling Average': '1-10'})
    .merge(s1[['Bowler','Bowling Average']].rename(columns={'Bowling Average': '11-17'}), on='Bowler', how='outer')
    .merge(s2[['Bowler', 'Bowling Average']].rename(columns={'Bowling Average': '18-25'}), on='Bowler', how='outer')
    .merge(s3[['Bowler', 'Bowling Average']].rename(columns={'Bowling Average': '26-32'}), on='Bowler', how='outer')
    .merge(s4[['Bowler', 'Bowling Average']].rename(columns={'Bowling Average': '33-40'}), on='Bowler', how='outer')
    .merge(final_3[['Bowler', 'Bowling Average']].rename(columns={'Bowling Average': '41-50'}), on='Bowler', how='outer'
)

ba.fillna(0, inplace=True)
ba.sort_values(by='Bowler', inplace=True)

ba
```

Out[59]:

	Bowler	1-10	11-17	18-25	26-32	33-40	41-50
0	Arshdeep Singh	22.86	0.00	0.00	52.33	0.00	33.83
1	Axar Patel	74.00	42.20	58.33	96.75	83.20	261.00
2	Hardik Pandya	34.17	29.91	0.00	51.50	76.00	74.67
3	Harshit Rana	37.50	0.00	22.00	36.00	53.00	62.00
4	Kuldeep Yadav	0.00	29.57	47.13	91.29	106.86	91.25
5	Mohammed Shami	31.06	34.17	49.00	68.20	105.33	36.96
6	Ravindra Jadeja	12.00	38.71	64.42	65.12	175.40	71.14
7	Varun Chakravarthy	0.00	18.00	0.00	0.00	0.00	0.00
8	Washington Sundar	55.00	30.00	37.83	82.00	85.40	81.50

In [60]:
```
import matplotlib.pyplot as plt
pa = ba.set_index('Bowler').T
pacers = ["Arshdeep Singh", "Hardik Pandya", "Harshit Rana", "Mohammed Shami"]
spinners = ["Axar Patel", "Kuldeep Yadav", "Ravindra Jadeja", "Varun Chakravarthy", "Washington Sundar"]
pa_pacers = pa.loc[:, pa.columns.isin(pacers)]
pa_spinners = pa.loc[:, pa.columns.isin(spinners)]
colors = ["Red", "Blue", "Green", "Orange", "Purple", "Brown", "Pink", "Cyan", "Magenta", "Yellow"]
markers1 = ["o", "s", "D", "^"]
markers2 = ["v", "p", "*", "X", "P", "h"]
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 6))
for i, bowler in enumerate(pa_pacers.columns):
    ax1.plot(['1-10','11-17','18-25','26-32','33-40','41-50'], pa_pacers[bowler], color=colors[i % len(colors)], marker=

ax1.set_xlabel('Overs')
ax1.set_ylabel('Bowling Average')
ax1.set_title('Pacers Bowling Average')
ax1.legend(loc='upper right', fontsize=6.5)
ax1.grid(True)
for i, bowler in enumerate(pa_spinners.columns):
    ax2.plot(['1-10','11-17','18-25','26-32','33-40','41-50'], pa_spinners[bowler], color=colors[i % len(colors)], marke
ax2.set_xlabel('Overs')
ax2.set_ylabel('Bowling Average')
ax2.set_title('Spinners Bowling Average')
ax2.legend(loc='upper right', fontsize=6.5)
ax2.grid(True)

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



Insights:

Shami shines in death overs , Hardik steady, Harshit seems inconsistent. Axar collapses late , Jadeja & Kuldeep solid in middle. Sundar ultra-consistent .

Conclusion:

Pacers need better death-over control, spinners must tighten late-game! Spinners Advantage: Dubai's slow pitches favor spinners like Kuldeep, Jadeja, and Sundar, who rely on turn & control. Expect them to be more effective. Pacers Challenge: Hit-the-deck pacers like Harshit & Hardik may struggle. Shami's variations will be key in death overs. Slower balls & cutters will be crucial. Game Plan: Spinners should attack early, pacers must use variations smartly. Dubai will test their adaptability!

In [61]:
```
Dubai['bowl_style'].head()
```

Out[61]:
```
327849    LFM
327850    LFM
327851    LFM
327852    LFM
327853    LFM
Name: bowl_style, dtype: object
```

In [62]:
```
#The Powerplay Strike
d_powe=Dubai[(Dubai['over']>0)&(Dubai['over']<11)&(Dubai['out']!='mixture/unknown')&(Dubai['year']>2019)]


d_pre_final_1=powe.groupby(['bowl_kind','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max'
d_final_1=d_pre_final_1.groupby(['bowl_kind']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wickets
d_final_1['Economy']=(d_final_1['Runs']/d_final_1['Overs']).round(2)
d_final_1['Bowling_Average']=(d_final_1['Runs']/d_final_1['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
d_final_1['Bowling_Average']=d_final_1['Bowling_Average'].apply(bowl_avg)

d_final_1=d_final_1.sort_values(by='Wickets',ascending=False)


int_part = d_final_1['Overs'].astype(int)
dec_part = (d_final_1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
d_final_1['Overs']=int_part+rem_ball
d_final_1['Strike_Rate']=(balls/final_1['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
d_final_1['Strike_Rate']=d_final_1['Strike_Rate'].apply(str_rate)

d_final_1.rename(columns={'bowl_kind':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings_
d_final_1
```

Out[62]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	pace bowler	48	864	33	160.3	5.39	26.18	137.57
1	spin bowler	18	145	3	30.0	4.83	48.33	180.00

In [63]:
```
d2_powe=Dubai[(Dubai['over']>10)&(Dubai['over']<50)&(Dubai['out']!='mixture/unknown')&(Dubai['year']>2019)]


d2_pre_final_1=powe.groupby(['bowl_kind','p_match']).agg(Matches_Played=('p_match','nunique'),Runs=('cur_bowl_runs','max
d2_final_1=d2_pre_final_1.groupby(['bowl_kind']).agg(Innings_Bowled=('Matches_Played','count'),Runs=('Runs','sum'),Wicke
d2_final_1['Economy']=(d2_final_1['Runs']/d2_final_1['Overs']).round(2)
d2_final_1['Bowling_Average']=(d2_final_1['Runs']/d2_final_1['Wickets']).round(2)

def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average
d2_final_1['Bowling_Average']=d2_final_1['Bowling_Average'].apply(bowl_avg)

d2_final_1=d2_final_1.sort_values(by='Wickets',ascending=False)


int_part = d2_final_1['Overs'].astype(int)
dec_part = (d2_final_1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
Over = int_part + rem_ball
d2_final_1['Overs']=int_part+rem_ball
d2_final_1['Strike_Rate']=(balls/final_1['Wickets']).round(2)

def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate
d2_final_1['Strike_Rate']=d2_final_1['Strike_Rate'].apply(str_rate)

d2_final_1.rename(columns={'bowl_kind':'Bowler','Bowling_Average':'Bowling Average','Strike_Rate':'Strike Rate','Innings
d2_final_1
```

Out[63]:

	Bowler	Innings Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	pace bowler	48	864	33	160.3	5.39	26.18	137.57
1	spin bowler	18	145	3	30.0	4.83	48.33	180.00

In [64]:
```
pd.set_option("display.max_columns",None)
Dubai.head()
```

Out[64]:

	p_match	inns	bat	p_bat	team_bat	bowl	p_bowl	team_bowl	ball	ball_id	outcome	score	out	dismissal	p_out	over	noball
327849	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	1	0.01	no run	0	False	NaN	42683	1	0
327850	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	2	0.02	no run	0	False	NaN	42683	1	0
327851	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	3	0.03	no run	0	False	NaN	42683	1	0
327852	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	4	0.04	no run	0	False	NaN	42683	1	0
327853	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	5	0.05	no run	0	False	NaN	42683	1	0

In [65]:
```
Dubai['bowl_style'].value_counts()
```

Out[65]:
```
bowl_style
OB        5649
RMF       4855
SLA       4165
RM        3165
RFM       3107
LBG       2084
RF        1947
LF        1572
LB        1458
LFM       1019
LMF       1003
LM         533
RM/OB      409
LWS        350
RM/LBG     120
-           90
OB/LB       50
RMF/OB      46
RFM/LB      19
RM/LB       12
OB/LBG       6
Name: count, dtype: int64
```

In [73]:

```
bowling_categories = {
    "Pace": ["RF", "RFM", "RMF", "RM", "LF", "LFM", "LMF", "LM"],
    "Spin": ["LB", "LBG", "OB", "SLA", "LWS"],
    "Mixed": ["RFM/LB", "RM/LB", "RM/LBG", "RM/OB", "RMF/OB"]
}

def categorize_bowler(bowler):
    for category, types in bowling_categories.items():
        if bowler in types:
            return category
    return "Unknown"
final_1['Category'] = final_1['Bowler'].apply(categorize_bowler)
final_1
Dubai.head()
```

Out[73]:

	p_match	inns	bat	p_bat	team_bat	bowl	p_bowl	team_bowl	ball	ball_id	outcome	score	out	dismissal	p_out	over	noball
327849	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	1	0.01	no run	0	False	NaN	42683	1	0
327850	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	2	0.02	no run	0	False	NaN	42683	1	0
327851	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	3	0.03	no run	0	False	NaN	42683	1	0
327852	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	4	0.04	no run	0	False	NaN	42683	1	0
327853	392617	1	Salman Butt	42683	Pakistan	Nathan Bracken	4185	Australia	5	0.05	no run	0	False	NaN	42683	1	0

In [84]:
```
# Powerplay (1-10 overs)
d_powe = Dubai[(Dubai['over'] > 0) & (Dubai['over'] < 11) & (Dubai['out'] != 'mixture/unknown') & (Dubai['year'] > 2017)
bowling_categories = {
    "Left_Pace": ["LF", "LFM", "LMF", "LM"],
    "Right_Pace": ["RF", "RFM", "RMF", "RM"],
    "Off_Spin": ['OB', 'OB/LB', 'SLA', 'RM/OB', 'RMF/OB'],
    "Leg_Spin": ["LB", "LBG", "LWS", 'OB/LBG']
}

def categorize_bowler(bowl_style):
    for category, types in bowling_categories.items():
        if bowl_style in types:
            return category
    return "Unknown"
d_powe['Category'] = d_powe['bowl_style'].apply(categorize_bowler)
d_pre_final_1 = d_powe.groupby(['Category','bowl_kind', 'bowl_style', 'p_match']).agg(
    Matches_Played=('p_match', 'nunique'),
    Runs=('cur_bowl_runs', 'max'),
    Wickets=('out', 'sum'),
    Overs=('cur_bowl_ovr', 'max')
).reset_index()

d_final_1 = d_pre_final_1.groupby(['Category', 'bowl_kind', 'bowl_style']).agg(
    Innings_Bowled=('Matches_Played', 'count'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()
d_final_1['Economy'] = (d_final_1['Runs'] / d_final_1['Overs']).round(2)
d_final_1['Bowling_Average'] = (d_final_1['Runs'] / d_final_1['Wickets']).round(2)
def bowl_avg(Bowling_Average):
    return 0 if np.isinf(Bowling_Average) else Bowling_Average

d_final_1['Bowling_Average'] = d_final_1['Bowling_Average'].apply(bowl_avg)
d_final_1 = d_final_1.sort_values(by='Wickets', ascending=False)
int_part = d_final_1['Overs'].astype(int)
dec_part = (d_final_1['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d_final_1['Overs'] = int_part + rem_ball
d_final_1['Strike_Rate'] = (balls / d_final_1['Wickets']).round(2)
def str_rate(Strike_Rate):
    return 0 if np.isinf(Strike_Rate) else Strike_Rate

d_final_1['Strike_Rate'] = d_final_1['Strike_Rate'].apply(str_rate)
d_final_1.rename(columns={'bowl_kind': 'Bowler', 'Bowling_Average': 'Bowling Average', 'Strike_Rate': 'Strike Rate', 'In

d_final_1.drop(columns=['bowl_style','Bowler'], inplace=True, axis=1)
d_final_1_grouped = d_final_1.groupby('Category').agg(
    Innings_Bowled=('Innings Bowled', 'sum'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()

d_final_1_grouped['Economy'] = (d_final_1_grouped['Runs'] / d_final_1_grouped['Overs']).round(2)
d_final_1_grouped['Bowling Average'] = (d_final_1_grouped['Runs'] / d_final_1_grouped['Wickets']).round(2)
d_final_1_grouped['Bowling Average'] = d_final_1_grouped['Bowling Average'].replace([np.inf, -np.inf], 0)
int_part = d_final_1_grouped['Overs'].astype(int)
dec_part = (d_final_1_grouped['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d_final_1_grouped['Overs'] = int_part + rem_ball
d_final_1_grouped['Strike Rate'] = (balls / d_final_1_grouped['Wickets']).round(2)
d_final_1_grouped['Strike Rate'] = d_final_1_grouped['Strike Rate'].replace([np.inf, -np.inf], 0)
d_final_1_grouped
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3900507178.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  d_powe['Category'] = d_powe['bowl_style'].apply(categorize_bowler)
```

Out[84]:

	Category	Innings_Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	Left_Pace	49	741	22	179.5	4.13	33.68	49.05
1	Leg_Spin	4	44	1	7.0	6.29	44.00	42.00
2	Off_Spin	33	335	13	75.0	4.47	25.77	34.62
3	Right_Pace	90	1394	64	325.0	4.29	21.78	30.47
4	Unknown	1	4	0	1.0	4.00	0.00	0.00

In [85]:
```
# Middle Overs (11-17 overs)
d2_powe = Dubai[(Dubai['over'] > 10) & (Dubai['over'] < 18) & (Dubai['out'] != 'mixture/unknown') & (Dubai['year'] > 201
bowling_categories = {
    "Left_Pace": ["LF", "LFM", "LMF", "LM"],
    "Right_Pace": ["RF", "RFM", "RMF", "RM"],
    "Off_Spin": ['OB', 'OB/LB', 'SLA', 'RM/OB', 'RMF/OB'],
    "Leg_Spin": ["LB", "LBG", "LWS", 'OB/LBG']
}

def categorize_bowler(bowl_style):
    for category, types in bowling_categories.items():
        if bowl_style in types:
            return category
    return "Unknown"
d2_powe['Category'] = d2_powe['bowl_style'].apply(categorize_bowler)
d2_pre_final_1 = d2_powe.groupby(['Category','bowl_kind', 'bowl_style', 'p_match']).agg(
    Matches_Played=('p_match', 'nunique'),
    Runs=('cur_bowl_runs', 'max'),
    Wickets=('out', 'sum'),
    Overs=('cur_bowl_ovr', 'max')
).reset_index()

d2_final_1 = d2_pre_final_1.groupby(['Category', 'bowl_kind', 'bowl_style']).agg(
    Innings_Bowled=('Matches_Played', 'count'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()
d2_final_1['Economy'] = (d2_final_1['Runs'] / d2_final_1['Overs']).round(2)
d2_final_1['Bowling_Average'] = (d2_final_1['Runs'] / d2_final_1['Wickets']).round(2)

d2_final_1['Bowling_Average'] = d2_final_1['Bowling_Average'].apply(bowl_avg)

d2_final_1 = d2_final_1.sort_values(by='Wickets', ascending=False)

int_part = d2_final_1['Overs'].astype(int)
dec_part = (d2_final_1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
d2_final_1['Overs'] = int_part + rem_ball

d2_final_1['Strike_Rate'] = (balls / d2_final_1['Wickets']).round(2)
d2_final_1['Strike_Rate'] = d2_final_1['Strike_Rate'].apply(str_rate)

d2_final_1.rename(columns={'bowl_kind': 'Bowler', 'Bowling_Average': 'Bowling Average', 'Strike_Rate': 'Strike Rate', 'I
d2_final_1.drop(columns=['bowl_style','Bowler'], inplace=True, axis=1)
d_final_2_grouped = d2_final_1.groupby('Category').agg(
    Innings_Bowled=('Innings Bowled', 'sum'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()

d_final_2_grouped['Economy'] = (d_final_2_grouped['Runs'] / d_final_2_grouped['Overs']).round(2)
d_final_2_grouped['Bowling Average'] = (d_final_2_grouped['Runs'] / d_final_2_grouped['Wickets']).round(2)
d_final_2_grouped['Bowling Average'] = d_final_2_grouped['Bowling Average'].replace([np.inf, -np.inf], 0)
int_part = d_final_2_grouped['Overs'].astype(int)
dec_part = (d_final_2_grouped['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d_final_2_grouped['Overs'] = int_part + rem_ball
d_final_2_grouped['Strike Rate'] = (balls / d_final_2_grouped['Wickets']).round(2)
d_final_2_grouped['Strike Rate'] = d_final_2_grouped['Strike Rate'].replace([np.inf, -np.inf], 0)
d_final_2_grouped
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2158660569.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  d2_powe['Category'] = d2_powe['bowl_style'].apply(categorize_bowler)
```

Out[85]:

	Category	Innings_Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	Left_Pace	17	263	7	81.0	3.25	37.57	69.43
1	Leg_Spin	30	299	10	70.3	4.25	29.90	42.30
2	Off_Spin	54	691	25	181.3	3.81	27.64	43.56
3	Right_Pace	62	1061	22	270.0	3.93	48.23	73.64
4	Unknown	1	9	0	2.0	4.50	0.00	0.00

In [86]:
```
# Middle Overs (18-25 overs)
d3_powe = Dubai[(Dubai['over'] > 17) & (Dubai['over'] < 26) & (Dubai['out'] != 'mixture/unknown') & (Dubai['year'] > 201
bowling_categories = {
    "Left_Pace": ["LF", "LFM", "LMF", "LM"],
    "Right_Pace": ["RF", "RFM", "RMF", "RM"],
    "Off_Spin": ['OB', 'OB/LB', 'SLA', 'RM/OB', 'RMF/OB'],
    "Leg_Spin": ["LB", "LBG", "LWS", 'OB/LBG']
}

def categorize_bowler(bowl_style):
    for category, types in bowling_categories.items():
        if bowl_style in types:
            return category
    return "Unknown"
d3_powe['Category'] = d3_powe['bowl_style'].apply(categorize_bowler)
d3_pre_final_1 = d3_powe.groupby(['Category','bowl_kind', 'bowl_style', 'p_match']).agg(
    Matches_Played=('p_match', 'nunique'),
    Runs=('cur_bowl_runs', 'max'),
    Wickets=('out', 'sum'),
    Overs=('cur_bowl_ovr', 'max')
).reset_index()

d3_final_1 = d3_pre_final_1.groupby(['Category', 'bowl_kind', 'bowl_style']).agg(
    Innings_Bowled=('Matches_Played', 'count'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()
d3_final_1['Economy'] = (d3_final_1['Runs'] / d3_final_1['Overs']).round(2)
d3_final_1['Bowling_Average'] = (d3_final_1['Runs'] / d3_final_1['Wickets']).round(2)

d3_final_1['Bowling_Average'] = d3_final_1['Bowling_Average'].apply(bowl_avg)

d3_final_1 = d3_final_1.sort_values(by='Wickets', ascending=False)

int_part = d3_final_1['Overs'].astype(int)
dec_part = (d3_final_1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
d3_final_1['Overs'] = int_part + rem_ball

d3_final_1['Strike_Rate'] = (balls / d3_final_1['Wickets']).round(2)
d3_final_1['Strike_Rate'] = d3_final_1['Strike_Rate'].apply(str_rate)

d3_final_1.rename(columns={'bowl_kind': 'Bowler', 'Bowling_Average': 'Bowling Average', 'Strike_Rate': 'Strike Rate', 'I
d3_final_1.drop(columns=['bowl_style','Bowler'], inplace=True, axis=1)
d3_final_1_grouped = d3_final_1.groupby('Category').agg(
    Innings_Bowled=('Innings Bowled', 'sum'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()

d3_final_1_grouped['Economy'] = (d3_final_1_grouped['Runs'] / d3_final_1_grouped['Overs']).round(2)
d3_final_1_grouped['Bowling Average'] = (d3_final_1_grouped['Runs'] / d3_final_1_grouped['Wickets']).round(2)
d3_final_1_grouped['Bowling Average'] = d3_final_1_grouped['Bowling Average'].replace([np.inf, -np.inf], 0)
int_part = d3_final_1_grouped['Overs'].astype(int)
dec_part = (d3_final_1_grouped['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d3_final_1_grouped['Overs'] = int_part + rem_ball
d3_final_1_grouped['Strike Rate'] = (balls / d3_final_1_grouped['Wickets']).round(2)
d3_final_1_grouped['Strike Rate'] = d3_final_1_grouped['Strike Rate'].replace([np.inf, -np.inf], 0)
d3_final_1_grouped
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\793530704.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  d3_powe['Category'] = d3_powe['bowl_style'].apply(categorize_bowler)
```

Out[86]:

	Category	Innings_Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	Left_Pace	17	371	2	94.0	3.95	185.50	282.00
1	Leg_Spin	40	683	14	167.0	4.09	48.79	71.57
2	Off_Spin	62	1295	40	323.3	4.01	32.38	48.52
3	Right_Pace	52	1105	19	266.5	4.15	58.16	84.26
4	Unknown	2	10	1	3.0	3.33	10.00	18.00

In [87]:
```
# Middle Overs (26-32 overs)
d4_powe = Dubai[(Dubai['over'] > 25) & (Dubai['over'] < 33) & (Dubai['out'] != 'mixture/unknown') & (Dubai['year'] > 201
bowling_categories = {
    "Left_Pace": ["LF", "LFM", "LMF", "LM"],
    "Right_Pace": ["RF", "RFM", "RMF", "RM"],
    "Off_Spin": ['OB', 'OB/LB', 'SLA', 'RM/OB', 'RMF/OB'],
    "Leg_Spin": ["LB", "LBG", "LWS", 'OB/LBG']
}

def categorize_bowler(bowl_style):
    for category, types in bowling_categories.items():
        if bowl_style in types:
            return category
    return "Unknown"
d4_powe['Category'] = d4_powe['bowl_style'].apply(categorize_bowler)
d4_pre_final_1 = d4_powe.groupby(['Category','bowl_kind', 'bowl_style', 'p_match']).agg(
    Matches_Played=('p_match', 'nunique'),
    Runs=('cur_bowl_runs', 'max'),
    Wickets=('out', 'sum'),
    Overs=('cur_bowl_ovr', 'max')
).reset_index()

d4_final_1 = d4_pre_final_1.groupby(['Category', 'bowl_kind', 'bowl_style']).agg(
    Innings_Bowled=('Matches_Played', 'count'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()
d4_final_1['Economy'] = (d4_final_1['Runs'] / d4_final_1['Overs']).round(2)
d4_final_1['Bowling_Average'] = (d4_final_1['Runs'] / d4_final_1['Wickets']).round(2)

d4_final_1['Bowling_Average'] = d4_final_1['Bowling_Average'].apply(bowl_avg)

d4_final_1 = d4_final_1.sort_values(by='Wickets', ascending=False)

int_part = d4_final_1['Overs'].astype(int)
dec_part = (d4_final_1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
d4_final_1['Overs'] = int_part + rem_ball

d4_final_1['Strike_Rate'] = (balls / d4_final_1['Wickets']).round(2)
d4_final_1['Strike_Rate'] = d4_final_1['Strike_Rate'].apply(str_rate)

d4_final_1.rename(columns={'bowl_kind': 'Bowler', 'Bowling_Average': 'Bowling Average', 'Strike_Rate': 'Strike Rate', 'I
d4_final_1.drop(columns=['bowl_style','Bowler'], inplace=True, axis=1)
d4_final_1_grouped = d4_final_1.groupby('Category').agg(
    Innings_Bowled=('Innings Bowled', 'sum'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()

d4_final_1_grouped['Economy'] = (d4_final_1_grouped['Runs'] / d4_final_1_grouped['Overs']).round(2)
d4_final_1_grouped['Bowling Average'] = (d4_final_1_grouped['Runs'] / d4_final_1_grouped['Wickets']).round(2)
d4_final_1_grouped['Bowling Average'] = d4_final_1_grouped['Bowling Average'].replace([np.inf, -np.inf], 0)
int_part = d4_final_1_grouped['Overs'].astype(int)
dec_part = (d4_final_1_grouped['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d4_final_1_grouped['Overs'] = int_part + rem_ball
d4_final_1_grouped['Strike Rate'] = (balls / d4_final_1_grouped['Wickets']).round(2)
d4_final_1_grouped['Strike Rate'] = d4_final_1_grouped['Strike Rate'].replace([np.inf, -np.inf], 0)
d4_final_1_grouped
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\3726838209.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  d4_powe['Category'] = d4_powe['bowl_style'].apply(categorize_bowler)
```

Out[87]:

	Category	Innings_Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	Left_Pace	20	509	7	126.0	4.04	72.71	108.00
1	Leg_Spin	32	822	7	184.0	4.47	117.43	157.71
2	Off_Spin	65	1504	21	394.4	3.81	71.62	112.76
3	Right_Pace	58	1505	22	348.2	4.32	68.41	95.00
4	Unknown	2	43	1	10.0	4.30	43.00	60.00

In [88]:
```
# Middle Overs (33-40 overs)
d5_powe = Dubai[(Dubai['over'] > 32) & (Dubai['over'] < 41) & (Dubai['out'] != 'mixture/unknown') & (Dubai['year'] > 201

bowling_categories = {
    "Left_Pace": ["LF", "LFM", "LMF", "LM"],
    "Right_Pace": ["RF", "RFM", "RMF", "RM"],
    "Off_Spin": ['OB', 'OB/LB', 'SLA', 'RM/OB', 'RMF/OB'],
    "Leg_Spin": ["LB", "LBG", "LWS", 'OB/LBG']
}

def categorize_bowler(bowl_style):
    for category, types in bowling_categories.items():
        if bowl_style in types:
            return category
    return "Unknown"
d5_powe['Category'] = d5_powe['bowl_style'].apply(categorize_bowler)
d5_pre_final_1 = d5_powe.groupby(['Category','bowl_kind', 'bowl_style', 'p_match']).agg(
    Matches_Played=('p_match', 'nunique'),
    Runs=('cur_bowl_runs', 'max'),
    Wickets=('out', 'sum'),
    Overs=('cur_bowl_ovr', 'max')
).reset_index()

d5_final_1 = d5_pre_final_1.groupby(['Category', 'bowl_kind', 'bowl_style']).agg(
    Innings_Bowled=('Matches_Played', 'count'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()

d5_final_1['Economy'] = (d5_final_1['Runs'] / d5_final_1['Overs']).round(2)
d5_final_1['Bowling_Average'] = (d5_final_1['Runs'] / d5_final_1['Wickets']).round(2)

d5_final_1['Bowling_Average'] = d5_final_1['Bowling_Average'].apply(bowl_avg)

d5_final_1 = d5_final_1.sort_values(by='Wickets', ascending=False)

int_part = d5_final_1['Overs'].astype(int)
dec_part = (d5_final_1['Overs'] - int_part) * 10

balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10

rem_ball = rem_ball + (dec_part > 5) * 1
d5_final_1['Overs'] = int_part + rem_ball

d5_final_1['Strike_Rate'] = (balls / d5_final_1['Wickets']).round(2)
d5_final_1['Strike_Rate'] = d5_final_1['Strike_Rate'].apply(str_rate)

d5_final_1.rename(columns={'bowl_kind': 'Bowler', 'Bowling_Average': 'Bowling Average', 'Strike_Rate': 'Strike Rate', 'I
d5_final_1.drop(columns=['bowl_style','Bowler'], inplace=True, axis=1)

d5_final_1_grouped = d5_final_1.groupby('Category').agg(
    Innings_Bowled=('Innings Bowled', 'sum'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()

d5_final_1_grouped['Economy'] = (d5_final_1_grouped['Runs'] / d5_final_1_grouped['Overs']).round(2)
d5_final_1_grouped['Bowling Average'] = (d5_final_1_grouped['Runs'] / d5_final_1_grouped['Wickets']).round(2)
d5_final_1_grouped['Bowling Average'] = d5_final_1_grouped['Bowling Average'].replace([np.inf, -np.inf], 0)
int_part = d5_final_1_grouped['Overs'].astype(int)
dec_part = (d5_final_1_grouped['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d5_final_1_grouped['Overs'] = int_part + rem_ball
d5_final_1_grouped['Strike Rate'] = (balls / d5_final_1_grouped['Wickets']).round(2)
d5_final_1_grouped['Strike Rate'] = d5_final_1_grouped['Strike Rate'].replace([np.inf, -np.inf], 0)
d5_final_1_grouped
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\1124120842.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  d5_powe['Category'] = d5_powe['bowl_style'].apply(categorize_bowler)
```

Out[88]:

	Category	Innings_Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	Left_Pace	22	695	11	158.2	4.40	63.18	86.36
1	Leg_Spin	31	925	8	225.0	4.11	115.62	168.75
2	Off_Spin	56	1611	24	417.3	3.86	67.12	104.38
3	Right_Pace	65	1957	37	465.4	4.20	52.89	75.51
4	Unknown	2	36	1	10.0	3.60	36.00	60.00

In [89]:
```
d6_powe = Dubai[(Dubai['over'] > 40) & (Dubai['over'] < 51) & (Dubai['out'] != 'mixture/unknown') & (Dubai['year'] > 201

bowling_categories = {
    "Left_Pace": ["LF", "LFM", "LMF", "LM"],
    "Right_Pace": ["RF", "RFM", "RMF", "RM"],
    "Off_Spin": ['OB', 'OB/LB', 'SLA', 'RM/OB', 'RMF/OB'],
    "Leg_Spin": ["LB", "LBG", "LWS", 'OB/LBG']
}

def categorize_bowler(bowl_style):
    for category, types in bowling_categories.items():
        if bowl_style in types:
            return category
    return "Unknown"
d6_powe['Category'] = d6_powe['bowl_style'].apply(categorize_bowler)
d6_pre_final_1 = d6_powe.groupby(['Category','bowl_kind', 'bowl_style', 'p_match']).agg(
    Matches_Played=('p_match', 'nunique'),
    Runs=('cur_bowl_runs', 'max'),
    Wickets=('out', 'sum'),
    Overs=('cur_bowl_ovr', 'max')
).reset_index()

d6_final_1 = d6_pre_final_1.groupby(['Category', 'bowl_kind', 'bowl_style']).agg(
    Innings_Bowled=('Matches_Played', 'count'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()
d6_final_1['Economy'] = (d6_final_1['Runs'] / d6_final_1['Overs']).round(2)
d6_final_1['Bowling_Average'] = (d6_final_1['Runs'] / d6_final_1['Wickets']).round(2)
d6_final_1['Bowling_Average'] = d6_final_1['Bowling_Average'].apply(bowl_avg)
d6_final_1 = d6_final_1.sort_values(by='Wickets', ascending=False)
int_part = d6_final_1['Overs'].astype(int)
dec_part = (d6_final_1['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d6_final_1['Overs'] = int_part + rem_ball
d6_final_1['Strike_Rate'] = (balls / d6_final_1['Wickets']).round(2)
d6_final_1['Strike_Rate'] = d6_final_1['Strike_Rate'].apply(str_rate)
d6_final_1.rename(columns={'bowl_kind': 'Bowler', 'Bowling_Average': 'Bowling Average', 'Strike_Rate': 'Strike Rate', 'I

d6_final_1.drop(columns=['bowl_style','Bowler'], inplace=True, axis=1)

d6_final_1_grouped = d_final_1.groupby('Category').agg(
    Innings_Bowled=('Innings Bowled', 'sum'),
    Runs=('Runs', 'sum'),
    Wickets=('Wickets', 'sum'),
    Overs=('Overs', 'sum')
).reset_index()

d6_final_1_grouped['Economy'] = (d6_final_1_grouped['Runs'] / d6_final_1_grouped['Overs']).round(2)
d6_final_1_grouped['Bowling Average'] = (d6_final_1_grouped['Runs'] / d6_final_1_grouped['Wickets']).round(2)
d6_final_1_grouped['Bowling Average'] = d6_final_1_grouped['Bowling Average'].replace([np.inf, -np.inf], 0)
int_part = d6_final_1_grouped['Overs'].astype(int)
dec_part = (d6_final_1_grouped['Overs'] - int_part) * 10
balls = int_part * 6 + dec_part
rem_ball = (balls % 6) / 10
rem_ball = rem_ball + (dec_part > 5) * 1
d6_final_1_grouped['Overs'] = int_part + rem_ball
d6_final_1_grouped['Strike Rate'] = (balls / d6_final_1_grouped['Wickets']).round(2)
d6_final_1_grouped['Strike Rate'] = d6_final_1_grouped['Strike Rate'].replace([np.inf, -np.inf], 0)
d6_final_1_grouped
```

```
C:\Users\VISHNUVARADHAN M\AppData\Local\Temp\ipykernel_552\2144473079.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  d6_powe['Category'] = d6_powe['bowl_style'].apply(categorize_bowler)
```

Out[89]:

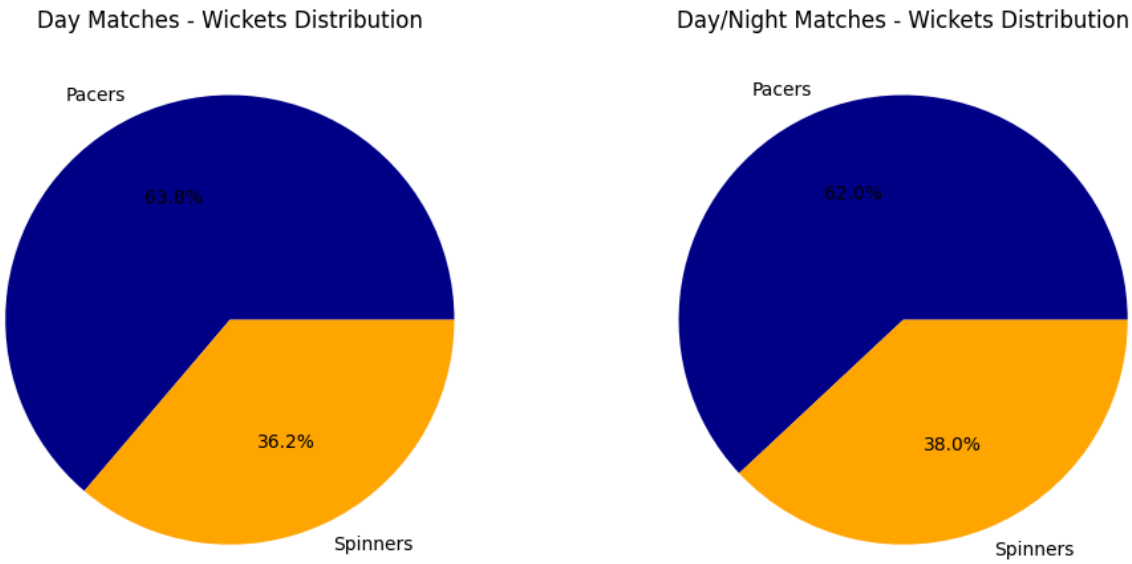
	Category	Innings_Bowled	Runs	Wickets	Overs	Economy	Bowling Average	Strike Rate
0	Left_Pace	49	741	22	179.5	4.13	33.68	49.05
1	Leg_Spin	4	44	1	7.0	6.29	44.00	42.00
2	Off_Spin	33	335	13	75.0	4.47	25.77	34.62
3	Right_Pace	90	1394	64	325.0	4.29	21.78	30.47
4	Unknown	1	4	0	1.0	4.00	0.00	0.00

In [91]:
```
d_ba = (
    d_final_1_grouped[['Category', 'Bowling Average']].rename(columns={'Bowling Average': '1-10'})
    .merge(d_final_1_grouped[['Category','Bowling Average']].rename(columns={'Bowling Average': '11-17'}), on='Category'
    .merge(d3_final_1_grouped[['Category', 'Bowling Average']].rename(columns={'Bowling Average': '18-25'}), on='Categor
    .merge(d4_final_1_grouped[['Category', 'Bowling Average']].rename(columns={'Bowling Average': '26-32'}), on='Categor
    .merge(d5_final_1_grouped[['Category', 'Bowling Average']].rename(columns={'Bowling Average': '33-40'}), on='Categor
    .merge(d6_final_1_grouped[['Category', 'Bowling Average']].rename(columns={'Bowling Average': '41-50'}), on='Categor
)
d_ba = d_ba[d_ba['Category'] != 'Unknown']
d_ba.fillna(0, inplace=True)
d_ba.sort_values(by='Category', inplace=True)
```

In [92]:
```
d_ec = (
    d_final_1_grouped[['Category', 'Economy']].rename(columns={'Economy': '1-10'})
    .merge(d_final_1_grouped[['Category','Economy']].rename(columns={'Economy': '11-17'}), on='Category', how='outer')
    .merge(d3_final_1_grouped[['Category', 'Economy']].rename(columns={'Economy': '18-25'}), on='Category', how='outer')
    .merge(d4_final_1_grouped[['Category', 'Economy']].rename(columns={'Economy': '26-32'}), on='Category', how='outer')
    .merge(d5_final_1_grouped[['Category', 'Economy']].rename(columns={'Economy': '33-40'}), on='Category', how='outer')
    .merge(d6_final_1_grouped[['Category', 'Economy']].rename(columns={'Economy': '41-50'}), on='Category', how='outer')
)
d_ec = d_ec[d_ec['Category'] != 'Unknown']
d_ec.fillna(0, inplace=True)
d_ec.sort_values(by='Category', inplace=True)
```

In [93]:
```
d_sr = (
    d_final_1_grouped[['Category', 'Strike Rate']].rename(columns={'Strike Rate': '1-10'})
    .merge(d_final_1_grouped[['Category','Strike Rate']].rename(columns={'Strike Rate': '11-17'}), on='Category', how='o
    .merge(d3_final_1_grouped[['Category', 'Strike Rate']].rename(columns={'Strike Rate': '18-25'}), on='Category', how=
    .merge(d4_final_1_grouped[['Category', 'Strike Rate']].rename(columns={'Strike Rate': '26-32'}), on='Category', how=
    .merge(d5_final_1_grouped[['Category', 'Strike Rate']].rename(columns={'Strike Rate': '33-40'}), on='Category', how=
    .merge(d6_final_1_grouped[['Category', 'Strike Rate']].rename(columns={'Strike Rate': '41-50'}), on='Category', how=
)
d_sr = d_sr[d_sr['Category'] != 'Unknown']
d_sr.fillna(0, inplace=True)
d_sr.sort_values(by='Category', inplace=True)
```

In [94]:
```
import matplotlib.pyplot as plt
size_day = [169, 96]
size_night = [57, 35]
labels = ['Pacers', 'Spinners']
fig, ax = plt.subplots(1, 2, figsize=(12, 6))
ax[0].pie(size_day, labels=labels, autopct='%1.1f%%', colors=['darkblue', 'orange'])
ax[0].set_title("Day Matches - Wickets Distribution")
ax[1].pie(size_night, labels=labels, autopct='%1.1f%%', colors=['darkblue', 'orange'])
ax[1].set_title("Day/Night Matches - Wickets Distribution")
plt.show()
spin_pace_Dubai
```



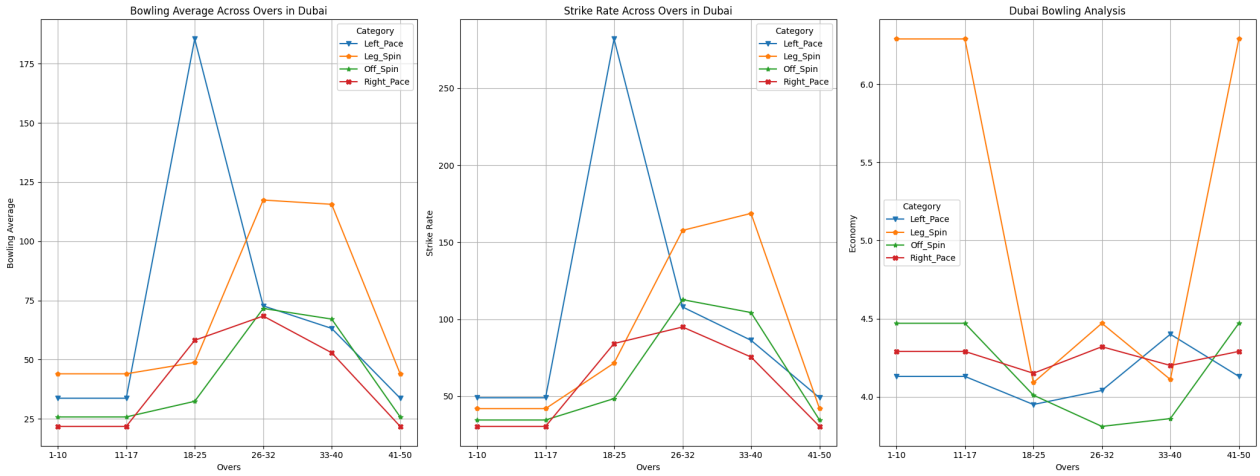
Out[94]:

ground	bowl_kind	mixture/unknown	pace bowler	spin bowler	Total_Wickets	Spin bowler Wickets %	Pace bowler Wickets %	Mixture/Unknown Wickets %
Dubai International Cricket Stadium	day match	12	169	96	277	34.66	61.01	4.33
	day/night match	1	57	35	93	37.63	61.29	1.08

Insights:

The pacers had taken more wickets than spinners in both Day and Night games of Dubai ODI's The proportion of wickets by both spinners and pacers are almost same(pacers-61% ,spinners-35%) So Team with great quality of pacers taking wickets has an edge in playing Dubai A team with a strong pace attack that consistently takes wickets has an advantage when playing in Dubai.

In [96]:
```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 3, figsize=(21, 8), sharex=True)
markers = ["v", "p", "*", "X", "P", "h","o", "s", "D", "^"]
axes[0].set_title('Bowling Average Across Overs in Dubai')
for i, category in enumerate(d_ba['Category'].unique()):
    subset = d_ba[d_ba['Category'] == category]
    axes[0].plot(['1-10', '11-17', '18-25', '26-32', '33-40', '41-50'],
                 subset.iloc[0, 1:], label=category, marker=markers[i % len(markers)])
axes[0].set_xlabel('Overs')
axes[0].set_ylabel('Bowling Average')
axes[0].legend(title='Category')
axes[0].grid(True)

axes[1].set_title('Strike Rate Across Overs in Dubai')
for i, category in enumerate(d_sr['Category'].unique()):
    subset = d_sr[d_sr['Category'] == category]
    axes[1].plot(['1-10', '11-17', '18-25', '26-32', '33-40', '41-50'],
                 subset.iloc[0, 1:], label=category, marker=markers[i % len(markers)])
axes[1].set_xlabel('Overs')
axes[1].set_ylabel('Strike Rate')
axes[1].legend(title='Category')
axes[1].grid(True)

axes[2].set_title('Economy Across Overs in Dubai')
for i, category in enumerate(d_ec['Category'].unique()):
    subset = d_ec[d_ec['Category'] == category]
    axes[2].plot(['1-10', '11-17', '18-25', '26-32', '33-40', '41-50'],
                 subset.iloc[0, 1:], label=category, marker=markers[i % len(markers)])
axes[2].set_xlabel('Overs')
axes[2].set_ylabel('Economy')
axes[2].legend(title='Category')
axes[2].grid(True)
plt.title("Dubai Bowling Analysis")
plt.tight_layout()
plt.show()
```



Dubai Venue Bowling Analysis - Champions Trophy 2025

Key Insights:

- **Powerplay Dominance (1-10 Overs):**
  - **Right-arm and Left-arm pacers** have been highly effective in the powerplay, maintaining a **low economy rate** and consistently picking up wickets. This has significantly reduced the chances of a strong start for the batting team, allowing the bowling side to dominate the initial phase.
  - Teams that use **swing bowlers** in the early overs tend to make significant inroads.
- **Middle Overs Control (11-40 Overs):**
  - **Leg spinners** have shown limited wicket-taking abilities despite maintaining a reasonable economy rate. The **high bowling average and strike rate** in this phase indicate that well-set batters tend to dominate against leg spinners.
  - **Off-spinners** have been outstanding in controlling the run flow, maintaining a **consistent economy rate** across all overs. However, their lower strike rate suggests they have not been as effective in securing crucial breakthroughs.
- **Death Overs Challenge (41-50 Overs):**
  - **Pacers** have adopted a **high-risk, high-reward strategy** in the death overs. While they have been successful in **picking wickets**, their **economy rate spikes significantly**, resulting in expensive finishes.
  - **Leg spinners** appear to struggle the most in the death overs, frequently conceding runs without taking wickets. This emphasizes the need for better bowling tactics during the death phase.

Strengths of Bowling in Dubai:

- **Powerplay Control:** Fast bowlers have shown **exceptional effectiveness** in the first 10 overs by consistently taking early wickets and maintaining low economy rates, preventing flying starts for batting teams.

- **Middle Overs Containment:**
  - Off-spinners have played a crucial role in **controlling the run flow** during the middle overs (11-40).
  - Leg spinners, although less effective in taking wickets, have still contributed by restricting quick runs in the middle phase.
- **Death Overs Wickets:** Despite the surge in economy rates, **fast bowlers** have consistently picked up crucial wickets during the death overs, minimizing the damage to a certain extent.

Challenges Faced in Bowling:

- **Leg Spinners' Struggles:**
  - Leg spinners have shown **higher bowling averages** and **longer strike rates**, indicating they struggle to get breakthroughs in crucial moments. This often allows well-set batters to accelerate.
- **Expensive Death Overs:**
  - Fast bowlers have faced a **run leakage problem** in the death overs, where their economy rate has spiked drastically.
  - This signifies a lack of effective **death-over execution** (wide yorkers, slower balls, cutters).
- **Over-Reliance on Pacers:**
  - Teams have shown a **higher dependency on pacers** for wickets, underutilizing their spin resources in the middle overs. This has resulted in missed opportunities for breaking partnerships early.

Tactical Recommendations for Teams in Dubai:

1. **Maximize Powerplay Wickets:**
   - Use **Right-arm and Left-arm pacers** aggressively in the first 10 overs to capitalize on early wickets.
   - Deploy attacking field sets during powerplay to build pressure.
2. **Effective Middle Over Strategy:**
   - Introduce **off-spinners** with attacking field sets to slow down the run rate and break partnerships.
   - Avoid allowing settled batters to dominate leg-spinners; consider tactical bowling changes in the middle overs.
3. **Optimized Death Over Execution:**
   - Focus on executing **wide yorkers, slower deliveries, and off-pace balls** in the death overs to minimize run leakage.
   - Place boundary riders strategically to prevent easy boundary runs during death overs.
4. **Balanced Bowling Approach:**
   - Avoid **over-burdening pacers** for wickets; instead, utilize spinners more effectively during the middle overs to create breakthroughs.
   - Combine **variations in pace** with aggressive field settings to restrict high-scoring finishes.

Conclusion:

The bowling performance in **Dubai conditions** reveals a clear pattern:

- **Pacers dominate the powerplay** with wickets and low economy but struggle to contain runs in death overs.
- **Spinners effectively control the middle overs** but lack wicket-taking potential, allowing set batters to build partnerships.
- **Teams that balance spin and pace usage** have a significant advantage in Dubai conditions.

To improve bowling performance in **Champions Trophy 2025**, teams should:

- Enhance **death-over execution** with slower balls, wide yorkers, and tight lines.
- Deploy **spinners aggressively** in the middle overs to break partnerships instead of just controlling runs.
- Optimize **powerplay strategies** to exploit favorable pitch conditions and reduce early momentum for batting sides.

By executing these strategies effectively, teams can significantly boost their **win probability** in Dubai's **low-scoring, spin-friendly conditions**.

Overall Conclusion

- India's batting lineup appears well-balanced, with experienced players like Virat Kohli and Rohit Sharma showcasing remarkable consistency. The rising form of Shubman Gill is a major boost, offering stability in the top order. However, batting concerns after the 2-down position may pose a challenge if early wickets fall. Addressing the lower-middle order's finishing ability will be critical.
- The bowling attack shows immense potential, particularly with Mohammed Shami excelling in wicket-taking during powerplay and death overs. Spinners have been effective in controlling the run flow, but their wicket-taking capabilities remain a concern, especially in spin-friendly Dubai conditions. Increasing their strike rate will be crucial for India's success.
- Venue analysis highlights that Dubai favors teams batting second, primarily due to dew factors and pitch behavior in night matches. However, day-night games neutralize the toss advantage, meaning India's adaptability to varying conditions will play a major role. Ensuring a solid game plan irrespective of the toss will strengthen their chances.
- India's key challenges include:
  - Tackling spin-friendly conditions by ensuring batters rotate the strike frequently in the middle overs.
  - Preventing collapses after the top order falls, as seen in previous ICC tournaments.
  - Enhancing their death-over execution, especially for pacers, to minimize run leakage in the final overs.
  - Adopting smarter bowling changes in middle overs to break set partnerships without leaking runs.
- Strategic Recommendations:
  - Deploy specialist spinners aggressively during middle overs to break partnerships.
  - Improve post-powerplay acceleration by encouraging batters to target weak bowlers.
  - Strengthen death-over planning with clear yorker plans, slower deliveries, and better field placements to reduce boundary leakage.
  - Focus on rotating strike in middle overs, as playing dot balls could significantly damage India's chances in slow Dubai pitches.

Final Outlook:

India possesses a power-packed squad with a perfect blend of experience and youth. However, their key to success in the Champions Trophy will revolve around:

- Maximizing powerplay wickets through aggressive bowling setups.
- Tightening middle-over control using spinners while also creating breakthroughs.
- Refining death-over strategies to minimize late-innings scoring.
- Boosting lower-order batting performance to avoid collapses in crunch moments.

If India successfully implements these strategic improvements, they will significantly boost their winning probability in the 2025 Champions Trophy and stand a strong chance of lifting the trophy.

In [ ]: