# IPL BATTING ANALYSIS

## SOURCE: www.kaggle.com/datasets/iamsouravbanerjee/ipl-player-performance-dataset/data

## SUB FOLDER- ( [IPL - Player Performance Dataset\IPL - Player Performance Dataset\All Seasons Combine] )

ABSTRACT

The Indian Premier League (IPL) has emerged as a cricketing phenomenon since its inception in 2008, captivating audiences around the world with its blend of high-octane cricket, entertainment, and commercial success. Over the course of 14 seasons spanning from 2008 to 2022, the IPL has not only redefined the way cricket is played but has also evolved into a data-rich treasure trove that encapsulates the journey of every player, every boundary, and every run scored.

### TASKS TO BE DONE

TASK 1: Data Acquisition and Dataset Details

TASK 2: Data Cleaning

TASK 3: Data Visualization [10 results]

```
 1. PLAYERS TO PLAY MOST SEASONS FROM 2008-2021 [TOP 30]
 2. PLAYERS WITH MOST RUNS PER SEASON GIVEN THAT THE PLAYER PLAYED ATLEAST 3 SEASONS
 3. PLAYERS WITH MOST RUNS FROM 2008-2021 [TOP 30]
 4. PLAYERS WITH HIGHEST AVERAGE [MIN 50 INNS] [TOP 30]
 5. PLAYERS WITH HIGHEST STRIKE RATE [MIN 1000 RUNS]
 6. AVERAGE VS STRIKE RATE OF TOP 50 RUN SCORERS
 7. VIRAT KOHLI PERFORMANCE IN IPL
 8. AB DE VILLIERS PERFORMANCE IN IPL
 9. VIRAT KOHLI DISTRIBUTION OF RUNS
10. AB DE VILLIERS DISTRIBUTION OF RUNS
```

TASK 4: Data Modelling [3 MODELS]

```
1. Linear Regression
2. Kmeans Clustering
3. Random forest regressor
```

TASK 5: Testing the Model

## IMPORTING LIBRARIES

In [1]:
```
#import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

## TASK 1: DATA ACQUISITION AND DATASET DETAILS

In [2]:
```
#data acquisition
df=pd.read_csv('Most_Runs_All_Seasons_Combine.csv',index_col=[0])
df.head()
```

Out[2]:

| | Player | Mat | Inns | NO | Runs | HS | Avg | BF | SR | 100 | 50 | 4s | 6s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Shaun Marsh | 11 | 11 | 2 | 616 | 115 | 68.44 | 441 | 139.68 | 1 | 5 | 59 | 26 |
| **1** | Gautam Gambhir | 14 | 14 | 1 | 534 | 86 | 41.07 | 379 | 140.89 | 0 | 5 | 68 | 8 |
| **2** | Sanath Jayasuriya | 14 | 14 | 2 | 518 | 114* | 43.16 | 309 | 167.63 | 1 | 2 | 58 | 31 |
| **3** | Shane Watson | 15 | 15 | 5 | 472 | 76* | 47.20 | 311 | 151.76 | 0 | 4 | 47 | 19 |
| **4** | Graeme Smith | 11 | 11 | 2 | 441 | 91 | 49.00 | 362 | 121.82 | 0 | 3 | 54 | 8 |

In [3]:
```
# Generate descriptive statistics for the DataFrame 'df'
df.describe()
```

Out[3]:

| | Mat | Inns | NO | Runs | Avg | BF | SR | 100 | 50 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 1986.000000 | 1986.000000 | 1986.000000 | 1986.000000 | 1986.000000 | 1986.000000 | 1986.000000 | 1986.000000 | 1986.000000 | 198 |
| **mean** | 8.974824 | 6.580060 | 1.527190 | 128.539778 | 18.257170 | 100.359013 | 110.863776 | 0.033233 | 0.654582 | 1 |
| **std** | 5.007739 | 4.841767 | 1.583134 | 155.137676 | 15.376013 | 114.014540 | 44.655957 | 0.205475 | 1.263126 | 1 |
| **min** | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 4.000000 | 2.000000 | 0.000000 | 12.000000 | 6.000000 | 13.000000 | 88.920000 | 0.000000 | 0.000000 | |
| **50%** | 9.000000 | 5.000000 | 1.000000 | 55.000000 | 16.000000 | 49.000000 | 116.270000 | 0.000000 | 0.000000 | |
| **75%** | 14.000000 | 11.000000 | 2.000000 | 202.750000 | 27.345000 | 161.000000 | 135.282500 | 0.000000 | 1.000000 | 1 |
| **max** | 19.000000 | 19.000000 | 10.000000 | 973.000000 | 152.000000 | 640.000000 | 400.000000 | 4.000000 | 9.000000 | 8 |

INFERENCE: This confirms that the dataset contains 1986 rows

In [4]:
```
# Display information about the DataFrame 'df'
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1986 entries, 0 to 1985
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Player  1986 non-null   object
 1   Mat     1986 non-null   int64
 2   Inns    1986 non-null   int64
 3   NO      1986 non-null   int64
 4   Runs    1986 non-null   int64
 5   HS      1986 non-null   object
 6   Avg     1986 non-null   float64
 7   BF      1986 non-null   int64
 8   SR      1986 non-null   float64
 9   100     1986 non-null   int64
 10  50      1986 non-null   int64
 11  4s      1986 non-null   int64
 12  6s      1986 non-null   int64
dtypes: float64(2), int64(9), object(2)
memory usage: 217.2+ KB
```

INFERENCE: This says us that the dataset contains 13 columns

## TASK 2: DATA CLEANING

In [5]:
```
# Count and sum the missing (null) values in each column of the DataFrame 'df'
df.isnull().sum()
```

Out[5]:
```
Player    0
Mat       0
Inns      0
NO        0
Runs      0
HS        0
Avg       0
BF        0
SR        0
100       0
50        0
4s        0
6s        0
dtype: int64
```

INFERENCE: THERE ARE NO NULL VALUES IN ANY COLUMNS

### PLAYERS TO PLAY MOST NUMBER OF SEASONS

In [6]:
```
# Count the number of occurrences of each player's name in the 'Player' column of the DataFrame 'df'
df['Player'].value_counts()
```

Out[6]:
```
Player
Shikhar Dhawan       15
Virat Kohli          14
Wriddhiman Saha      14
Manish Pandey        14
MS Dhoni             14
                     ..
Lee Carseldine        1
Rob Quiney            1
Marchant de Lange     1
Abdur Razzak          1
Anuj Rawat            1
Name: count, Length: 545, dtype: int64
```

## INFERENCE: SHIKHAR DHAWAN has played 15 seasons while my data is only of 14 IPL seasons. We can conclude that the dataset has duplicate entries.

In [7]:
```
# Filter the DataFrame 'df' to select rows where the 'Player' column is 'Shikhar Dhawan'
df.loc[df['Player']=='Shikhar Dhawan']
```

Out[7]:

| | Player | Mat | Inns | NO | Runs | HS | Avg | BF | SR | 100 | 50 | 4s | 6s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **13** | Shikhar Dhawan | 14 | 14 | 5 | 340 | 68* | 37.77 | 295 | 115.25 | 0 | 4 | 35 | 8 |
| **218** | Shikhar Dhawan | 5 | 4 | 0 | 40 | 22 | 10.00 | 45 | 88.88 | 0 | 0 | 3 | 0 |
| **219** | Shikhar Dhawan | 5 | 4 | 0 | 40 | 22 | 10.00 | 45 | 88.88 | 0 | 0 | 3 | 0 |
| **321** | Shikhar Dhawan | 10 | 10 | 0 | 191 | 56 | 19.10 | 170 | 112.35 | 0 | 2 | 23 | 3 |
| **442** | Shikhar Dhawan | 14 | 14 | 2 | 400 | 95* | 33.33 | 310 | 129.03 | 0 | 2 | 47 | 7 |
| **584** | Shikhar Dhawan | 15 | 15 | 1 | 569 | 84 | 40.64 | 439 | 129.61 | 0 | 5 | 58 | 18 |
| **754** | Shikhar Dhawan | 10 | 10 | 2 | 311 | 73* | 38.87 | 253 | 122.92 | 0 | 3 | 37 | 5 |
| **895** | Shikhar Dhawan | 14 | 14 | 1 | 377 | 64* | 29.00 | 319 | 118.18 | 0 | 2 | 49 | 7 |
| **1031** | Shikhar Dhawan | 14 | 14 | 1 | 353 | 54 | 27.15 | 286 | 123.42 | 0 | 3 | 45 | 6 |
| **1146** | Shikhar Dhawan | 17 | 17 | 4 | 501 | 82* | 38.53 | 429 | 116.78 | 0 | 4 | 51 | 8 |
| **1281** | Shikhar Dhawan | 14 | 14 | 1 | 479 | 77 | 36.84 | 376 | 127.39 | 0 | 3 | 53 | 9 |
| **1431** | Shikhar Dhawan | 16 | 16 | 3 | 497 | 92* | 38.23 | 363 | 136.91 | 0 | 4 | 59 | 14 |
| **1563** | Shikhar Dhawan | 16 | 16 | 1 | 521 | 97* | 34.73 | 384 | 135.67 | 0 | 5 | 64 | 11 |
| **1705** | Shikhar Dhawan | 17 | 17 | 3 | 618 | 106* | 44.14 | 427 | 144.73 | 2 | 4 | 67 | 12 |
| **1840** | Shikhar Dhawan | 16 | 16 | 1 | 587 | 92 | 39.13 | 471 | 124.62 | 0 | 3 | 63 | 16 |

## INFERENCE: WE CAN SEE THAT 218 AND 219 ARE REPEATED

In [8]:
```
# Select and display rows in the DataFrame 'df' that are duplicates based on all columns
df[df.duplicated()]
```

Out[8]:

| | Player | Mat | Inns | NO | Runs | HS | Avg | BF | SR | 100 | 50 | 4s | 6s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **219** | Shikhar Dhawan | 5 | 4 | 0 | 40 | 22 | 10.0 | 45 | 88.88 | 0 | 0 | 3 | 0 |
| **240** | Ishant Sharma | 11 | 3 | 1 | 16 | 9 | 8.0 | 13 | 123.07 | 0 | 0 | 1 | 1 |

## INFERENCE- THERE ARE TWO DUPLICATE DATA IN THE DATASET

In [9]:
```
# Remove duplicate rows from the DataFrame 'df' and apply the changes in-place
df.drop_duplicates(inplace=True)
```

In [10]:
```
# Generate summary statistics for the DataFrame 'df'
df.describe()
```

Out[10]:

| | Mat | Inns | NO | Runs | Avg | BF | SR | 100 | 50 | |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 1984.000000 | 1984.000000 | 1984.000000 | 1984.000000 | 1984.000000 | 1984.000000 | 1984.000000 | 1984.000000 | 1984.000000 | 198 |
| mean | 8.975806 | 6.583165 | 1.528226 | 128.641129 | 18.266502 | 100.430948 | 110.868705 | 0.033266 | 0.655242 | 1 |
| std | 5.009262 | 4.843193 | 1.583516 | 155.182544 | 15.380920 | 114.048354 | 44.674902 | 0.205576 | 1.263592 | 1 |
| min | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 25% | 4.000000 | 2.000000 | 0.000000 | 12.000000 | 6.000000 | 13.000000 | 89.000000 | 0.000000 | 0.000000 | |
| 50% | 9.000000 | 5.000000 | 1.000000 | 55.000000 | 16.000000 | 49.000000 | 116.270000 | 0.000000 | 0.000000 | |
| 75% | 14.000000 | 11.000000 | 2.000000 | 203.000000 | 27.357500 | 161.000000 | 135.297500 | 0.000000 | 1.000000 | 1 |
| max | 19.000000 | 19.000000 | 10.000000 | 973.000000 | 152.000000 | 640.000000 | 400.000000 | 4.000000 | 9.000000 | 8 |

### INFERENCE: AFTER REMOVING THE DUPLICATE ENTRIES OUR DATASET CONTAINS 1984 ROWS AND 13 COLUMNS

## TASK 3: DATA VISUALIZATION

### 1.PLAYERS TO PLAY MOST SEASONS FROM 2008-2021

In [11]:

```
# Count the number of seasons played by each player in the DataFrame 'df' and create a new DataFrame 'no_of_sea
no_of_seasons = df['Player'].value_counts().reset_index()

# Rename the columns in the new DataFrame for clarity
no_of_seasons.columns = ['Player', 'No_of_Seasons_Played']

# Sort the 'no_of_seasons' DataFrame based on player names
no_of_seasons.sort_values('Player', inplace=True)

# Reset the index to maintain a clean structure
no_of_seasons.reset_index(inplace=True)

# Print the resulting DataFrame 'no_of_seasons'
print(no_of_seasons)
```

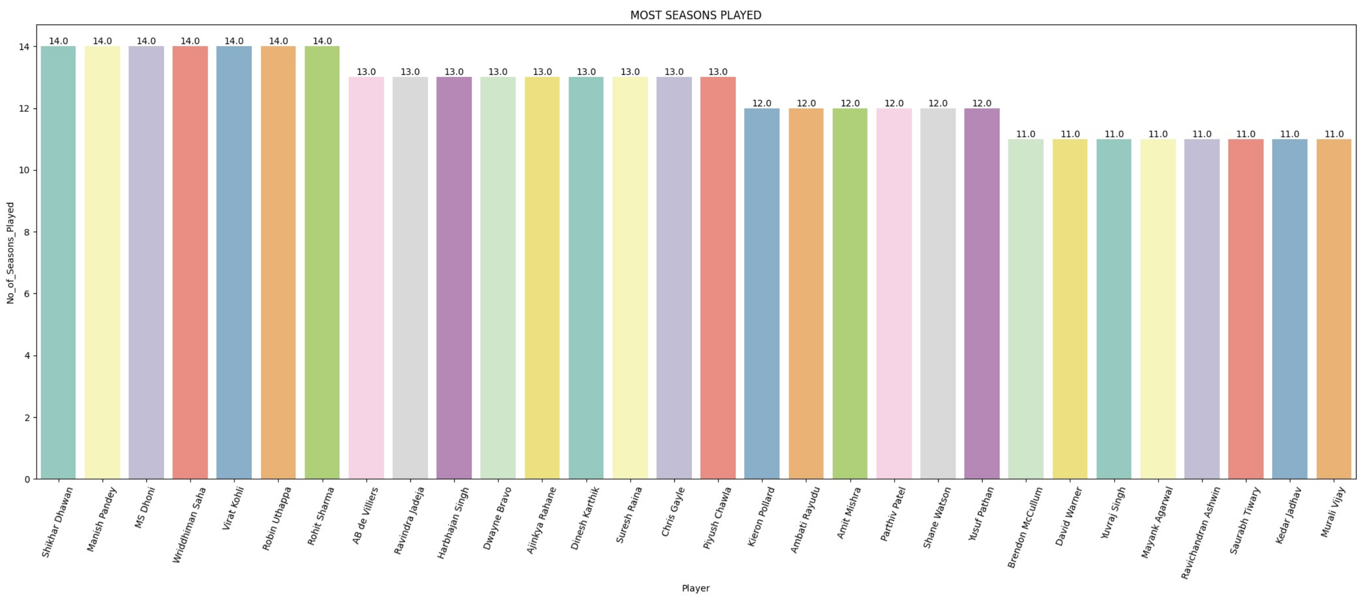
```
     index             Player  No_of_Seasons_Played
0        8     AB de Villiers                    13
1      337      Aakash Chopra                     2
2       31        Aaron Finch                    10
3      284        Abdul Samad                     2
4      543      Abdur Razzak                      1
..     ...                ...                   ...
540    495        Younis Khan                     1
541     20       Yusuf Pathan                    12
542     25       Yuvraj Singh                    11
543     71  Yuzvendra Chahal                     7
544     59        Zaheer Khan                     8

[545 rows x 3 columns]
```

In [12]:

```
# Sort the 'no_of_seasons' DataFrame by the number of seasons played in descending order,
# select the top 30 players, and reset the index to maintain a clean structure
temp = no_of_seasons.sort_values('No_of_Seasons_Played', ascending=False)[:30].reset_index()

# Print the resulting DataFrame 'temp'
print(temp)
```

| | level_0 | index | Player | No_of_Seasons_Played |
|---|---|---|---|---|
| 0 | 445 | 0 | Shikhar Dhawan | 14 |
| 1 | 273 | 3 | Manish Pandey | 14 |
| 2 | 266 | 4 | MS Dhoni | 14 |
| 3 | 533 | 2 | Wriddhiman Saha | 14 |
| 4 | 524 | 1 | Virat Kohli | 14 |
| 5 | 397 | 6 | Robin Uthappa | 14 |
| 6 | 401 | 5 | Rohit Sharma | 14 |
| 7 | 0 | 8 | AB de Villiers | 13 |
| 8 | 384 | 9 | Ravindra Jadeja | 13 |
| 9 | 171 | 10 | Harbhajan Singh | 13 |
| 10 | 151 | 12 | Dwayne Bravo | 13 |
| 11 | 22 | 15 | Ajinkya Rahane | 13 |
| 12 | 141 | 7 | Dinesh Karthik | 13 |
| 13 | 480 | 14 | Suresh Raina | 13 |
| 14 | 105 | 13 | Chris Gayle | 13 |
| 15 | 349 | 11 | Piyush Chawla | 13 |
| 16 | 242 | 16 | Kieron Pollard | 12 |
| 17 | 34 | 18 | Ambati Rayudu | 12 |
| 18 | 35 | 19 | Amit Mishra | 12 |
| 19 | 340 | 21 | Parthiv Patel | 12 |
| 20 | 436 | 17 | Shane Watson | 12 |
| 21 | 541 | 20 | Yusuf Pathan | 12 |
| 22 | 92 | 28 | Brendon McCullum | 11 |
| 23 | 127 | 27 | David Warner | 11 |
| 24 | 542 | 25 | Yuvraj Singh | 11 |
| 25 | 288 | 24 | Mayank Agarwal | 11 |
| 26 | 383 | 22 | Ravichandran Ashwin | 11 |
| 27 | 423 | 23 | Saurabh Tiwary | 11 |
| 28 | 236 | 29 | Kedar Jadhav | 11 |
| 29 | 316 | 26 | Murali Vijay | 11 |

In [13]:
```
# Create a bar plot to visualize the top 30 players with the most seasons played
plt.figure(figsize=(26, 9))
plt.title("MOST SEASONS PLAYED")

# Use seaborn to create the bar plot, specifying the data, 'Player' on the x-axis, and 'No_of_Seasons_Played' o
sns.barplot(data=temp, x='Player', y='No_of_Seasons_Played', palette='Set3')

# Rotate x-axis labels for better readability
plt.xticks(rotation=70)

# Access the current axis
ax = plt.gca()

# Annotate each bar with the corresponding number of seasons played
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='bottom'
```




INFERENCE: There are 6 players who played all the 14 seasons they are SHIKHAR DHAWAN, MANISH PANDEY, M S DHONI, WRIDDHIMAN SAHA, VIRAT KOHLI, ROBIN UTHAPPA.

## 2. PLAYERS WITH MOST RUNS PER SEASON GIVEN THAT THE PLAYER HAS PLAYED ATLEAST 3 SEASONS [TOP 30]

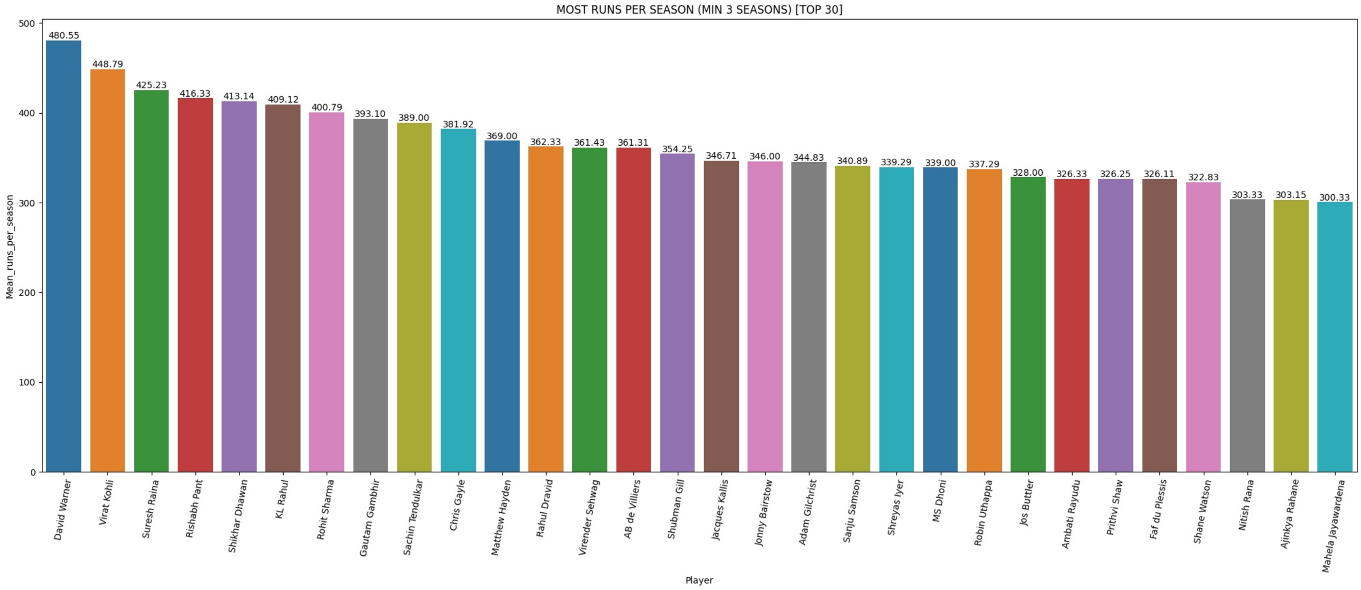
In [14]:
```
# Calculate the mean runs scored per season for each player and create a new DataFrame 'mean_runs'
mean_runs = df.groupby('Player')['Runs'].mean().reset_index()
mean_runs.columns = ['Player', 'Mean_runs_per_season']
```

```python
# Filter players who have played at least 3 seasons as 'contendors'
temp = mean_runs.loc[no_of_seasons['No_of_Seasons_Played'] >= 3]

# Count the number of players who satisfy the criteria
contendors = temp.shape[0]
print("Total number of players satisfying this criteria =", contendors)

# Sort the 'temp' DataFrame by mean runs per season in descending order, selecting the top 30 players
temp = temp.sort_values('Mean_runs_per_season', ascending=False)[:30].reset_index()
print(temp)
```

```
Total number of players satisfying this criteria = 273
    index               Player  Mean_runs_per_season
0     127         David Warner            480.545455
1     524          Virat Kohli            448.785714
2     480         Suresh Raina            425.230769
3     392         Rishabh Pant            416.333333
4     445       Shikhar Dhawan            413.142857
5     223             KL Rahul            409.125000
6     401         Rohit Sharma            400.785714
7     160       Gautam Gambhir            393.100000
8     411     Sachin Tendulkar            389.000000
9     105          Chris Gayle            381.923077
10    286       Matthew Hayden            369.000000
11    366         Rahul Dravid            362.333333
12    526      Virender Sehwag            361.428571
13      0       AB de Villiers            361.307692
14    459         Shubman Gill            354.250000
15    194       Jacques Kallis            346.714286
16    217       Jonny Bairstow            346.000000
17     12       Adam Gilchrist            344.833333
18    421         Sanju Samson            340.888889
19    458         Shreyas Iyer            339.285714
20    266             MS Dhoni            339.000000
21    397        Robin Uthappa            337.285714
22    218          Jos Buttler            328.000000
23     34        Ambati Rayudu            326.333333
24    361         Prithvi Shaw            326.250000
25    157        Faf du Plessis           326.111111
26    436         Shane Watson            322.833333
27    332          Nitish Rana            303.333333
28     22       Ajinkya Rahane            303.153846
29    267   Mahela Jayawardena            300.333333
```

In [15]:

```python
# Create a bar plot to visualize the top 30 players with the highest mean runs per season (min 3 seasons)
plt.figure(figsize=(26, 9))
plt.title("MOST RUNS PER SEASON (MIN 3 SEASONS) [TOP 30]")

# Use seaborn to create the bar plot, specifying the data, 'Player' on the x-axis, and 'Mean_runs_per_season' o
sns.barplot(data=temp, x='Player', y='Mean_runs_per_season', palette='tab10', width=0.8)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Access the current axis
ax = plt.gca()

# Annotate each bar with the corresponding mean runs per season (formatted to two decimal places)
for p in ax.patches:
    label = f'{p.get_height():.2f}'  # Format the label to two decimal places
    ax.annotate(label, (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='bottom')
```




INFERENCE: DAVID WARNER has the highest runs per season that is 480.55

## 3. PLAYERS WITH MOST RUNS FROM 2008-2021 [TOP 30]

In [16]:

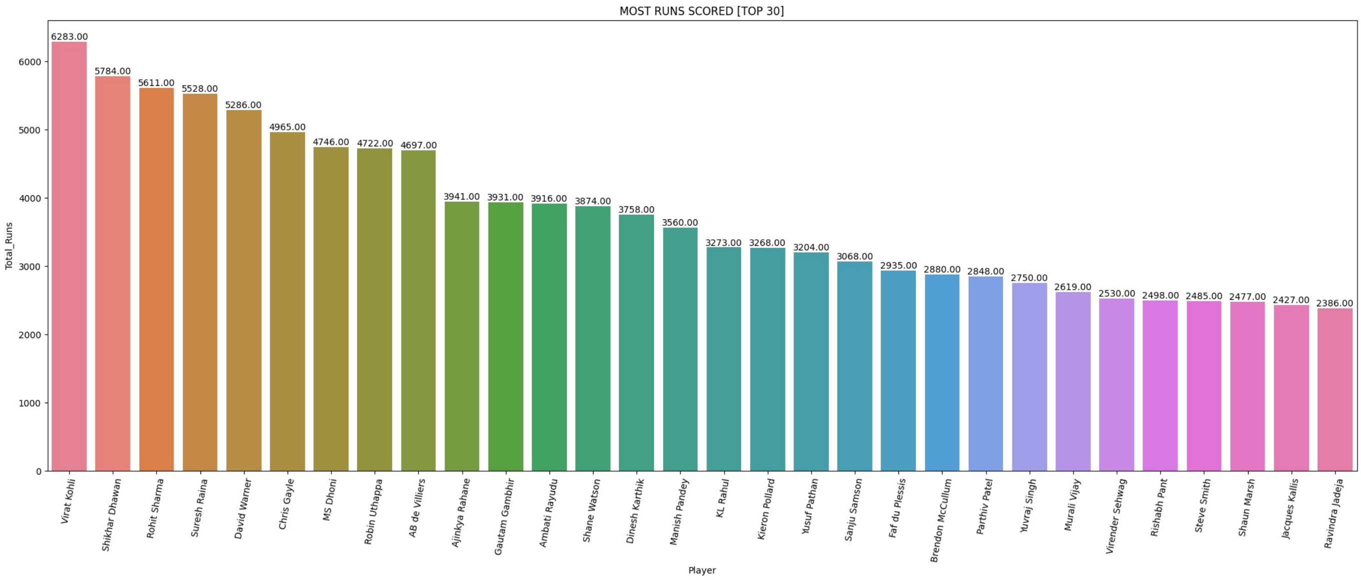
```
# Calculate the total runs scored by each player and create a new DataFrame 'total_runs'
total_runs = df.groupby('Player')['Runs'].sum().reset_index()
total_runs.columns = ['Player', 'Total_Runs']

# Sort the 'total_runs' DataFrame by total runs scored in descending order, selecting the top 30 players
temp = total_runs.sort_values('Total_Runs', ascending=False)[:30].reset_index()
print(temp)
```

```
    index            Player  Total_Runs
0     524       Virat Kohli        6283
1     445    Shikhar Dhawan        5784
2     401      Rohit Sharma        5611
3     480      Suresh Raina        5528
4     127      David Warner        5286
5     105       Chris Gayle        4965
6     266          MS Dhoni        4746
7     397     Robin Uthappa        4722
8       0    AB de Villiers        4697
9      22    Ajinkya Rahane        3941
10    160    Gautam Gambhir        3931
11     34     Ambati Rayudu        3916
12    436      Shane Watson        3874
13    141    Dinesh Karthik        3758
14    273     Manish Pandey        3560
15    223          KL Rahul        3273
16    242    Kieron Pollard        3268
17    541     Yusuf Pathan        3204
18    421      Sanju Samson        3068
19    157    Faf du Plessis        2935
20     92  Brendon McCullum        2880
21    340     Parthiv Patel        2848
22    542      Yuvraj Singh        2750
23    316      Murali Vijay        2619
24    526   Virender Sehwag        2530
25    392      Rishabh Pant        2498
26    471       Steve Smith        2485
27    439       Shaun Marsh        2477
28    194    Jacques Kallis        2427
29    384   Ravindra Jadeja        2386
```

In [17]:

```
# Create a bar plot to visualize the top 30 players with the most total runs scored
plt.figure(figsize=(26, 9))
plt.title("MOST RUNS SCORED [TOP 30]")

# Use seaborn to create the bar plot, specifying the data, 'Player' on the x-axis, and 'Total_Runs' on the y-ax
sns.barplot(data=temp, x='Player', y='Total_Runs', palette='husl', width=0.8)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Access the current axis
ax = plt.gca()

# Annotate each bar with the corresponding total runs scored (formatted to two decimal places)
```

```python
for p in ax.patches:
    label = f'{p.get_height():.2f}'  # Format the label to two decimal places
    ax.annotate(label, (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='bottom')
```




INFERENCE: VIRAT KOHLI has scored the most runs overall that is 6283

## 4. PLAYERS WITH HIGHEST AVERAGE [MIN 50 INNS] [TOP 30]

In [18]:

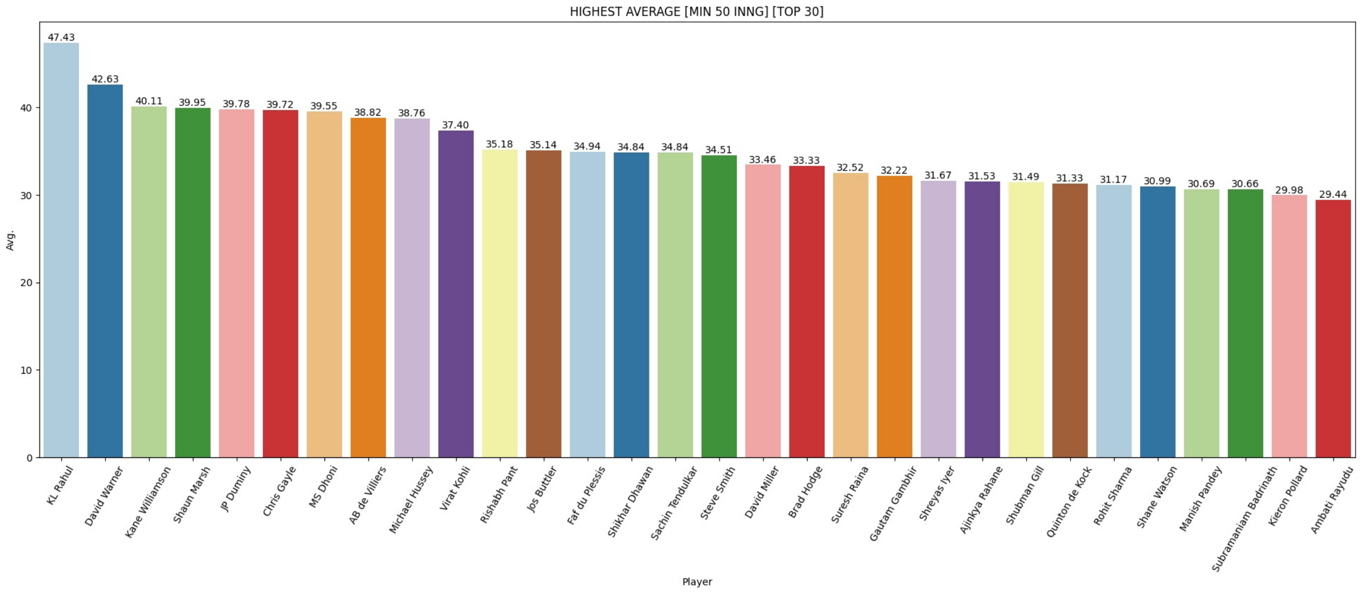
```python
# Calculate the total number of not outs (NO) for each player and create a new DataFrame 'not_outs'
not_outs = df.groupby('Player')['NO'].sum().reset_index()
not_outs.columns = ['Player', 'No_of_not_outs']

# Create a new DataFrame 'overall_avg' to calculate the overall batting average for each player
overall_avg = pd.DataFrame()
overall_avg['Player'] = total_runs['Player']
overall_avg['Runs'] = total_runs['Total_Runs']

# Calculate the total number of innings played by each player
inns = df.groupby('Player')['Inns'].sum().reset_index()
inns.columns = ['Player', 'Total_Innings_Played']

# Add 'Inns' and calculate the overall batting average (Avg.) for each player
overall_avg['Inns'] = inns['Total_Innings_Played']
overall_avg['Avg.'] = overall_avg['Runs'] / (overall_avg['Inns'] - not_outs['No_of_not_outs'])

# Print the resulting DataFrame 'overall_avg'
print(overall_avg)
```

```
               Player  Runs  Inns       Avg.
0      AB de Villiers  4697   157  38.818182
1       Aakash Chopra    53     6   8.833333
2         Aaron Finch  2005    85  25.705128
3         Abdul Samad   222    18  15.857143
4       Abdur Razzak     0     1        NaN
..                ...   ...   ...        ...
540       Younis Khan     3     1   3.000000
541      Yusuf Pathan  3204   154  29.127273
542      Yuvraj Singh  2750   126  24.774775
543  Yuzvendra Chahal    32    18   5.333333
544        Zaheer Khan   117    32   8.357143

[545 rows x 4 columns]
```

In [19]:

```python
# Filter players who have played a minimum of 50 innings as 'contendors'
temp = overall_avg.loc[inns['Total_Innings_Played'] >= 50]
contenders = temp.shape[0]

# Sort the 'temp' DataFrame by batting average (Avg.) in descending order, selecting the top 30 players
temp = temp.sort_values('Avg.', ascending=False)[:30]

# Create a bar plot to visualize the top 30 players with the highest batting averages (min 50 innings)
plt.figure(figsize=(24, 8))
plt.title("HIGHEST AVERAGE [MIN 50 INNG] [TOP 30]")

# Use seaborn to create the bar plot, specifying the data, 'Player' on the x-axis, and 'Avg.' on the y-axis
sns.barplot(data=temp, x='Player', y='Avg.', palette='Paired')

# Rotate x-axis labels for better readability
plt.xticks(rotation=60)
```

```
# Access the current axis
ax = plt.gca()

# Annotate each bar with the corresponding batting average (formatted to two decimal places)
for p in ax.patches:
    label = f'{p.get_height():.2f}'  # Format the label to two decimal places
    ax.annotate(label, (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='bottom')

# Display the plot
plt.show()

# Print the number of contenders satisfying the criteria
print(contenders)
```




86

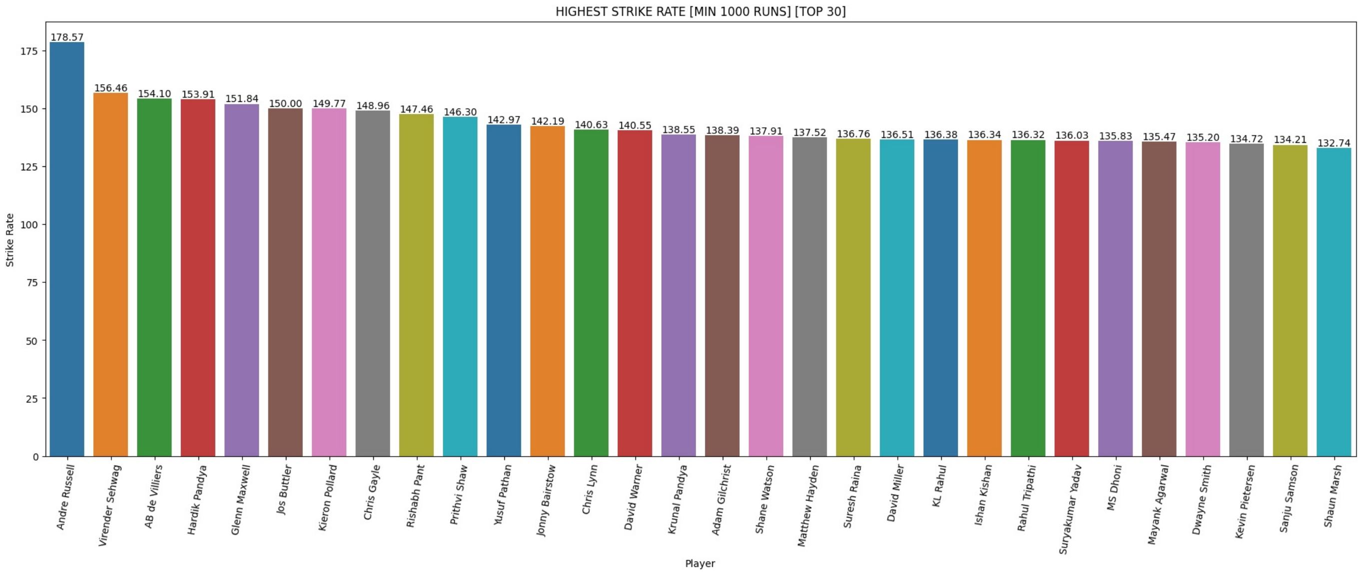
INFERENCE: NUMBER OF PLAYERS COMING UNDER THIS CATEGORY IS 86 AND K L RAHUL has the highest Avg that is 47.43.

## 5. PLAYERS WITH HIGHEST STRIKE RATE [MIN 1000 RUNS]

In [20]:

```
# Calculate the total number of balls faced by each player and create a new DataFrame 'total_balls_faced'
total_balls_faced = df.groupby('Player')['BF'].sum().reset_index()
total_balls_faced.columns = ['Player', 'No_of_balls_faced']

# Create a new DataFrame 'overall_sr' to calculate the overall strike rate for each player
overall_sr = pd.DataFrame()
overall_sr['Player'] = total_runs['Player']

# Calculate the strike rate for each player (min 1000 runs)
overall_sr['Strike Rate'] = (total_runs['Total_Runs'] / total_balls_faced['No_of_balls_faced']) * 100

# Filter players who have scored a minimum of 1000 runs as 'contendors'
temp = overall_sr.loc[total_runs['Total_Runs'] >= 1000]
contenders = temp.shape[0]

# Sort the 'temp' DataFrame by strike rate in descending order, selecting the top 30 players
temp = temp.sort_values('Strike Rate', ascending=False)[:30]

# Create a bar plot to visualize the top 30 players with the highest strike rates (min 1000 runs)
plt.figure(figsize=(24, 8))
plt.title("HIGHEST STRIKE RATE [MIN 1000 RUNS] [TOP 30]")

# Use seaborn to create the bar plot, specifying the data, 'Player' on the x-axis, and 'Strike Rate' on the y-a
sns.barplot(data=temp, x='Player', y='Strike Rate', palette='tab10')

# Rotate x-axis labels for better readability
plt.xticks(rotation=80);

# Access the current axis
ax = plt.gca()

# Annotate each bar with the corresponding strike rate (formatted to two decimal places)
for p in ax.patches:
    label = f'{p.get_height():.2f}'  # Format the label to two decimal places
    ax.annotate(label, (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='bottom')

# Display the plot
```

```python
plt.show()

# Print the number of contenders satisfying the criteria
print(contenders)
```




```
77
```

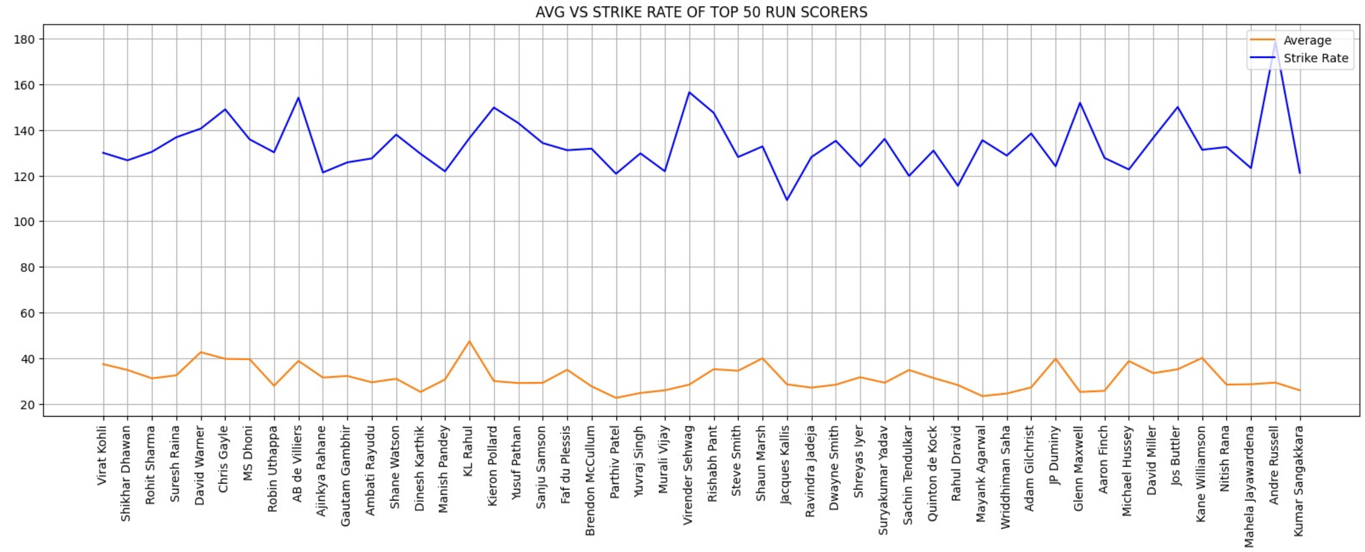
## INFERENCE: NUMBER OF PLAYERS THAT FALL UNDER THIS CATEGORY IS 77 AND ANDRE RUSSEL HAS THE HIGHEST STRIKE RATE THAT IS 178.57

## 6. AVERAGE VS STRIKE RATE OF TOP 50 RUN SCORERS

In [21]:

```python
# Create a new DataFrame 'combined_df' to combine relevant player statistics
combined_df = pd.DataFrame()
combined_df['Player'] = total_runs['Player']
combined_df['Runs'] = total_runs['Total_Runs']
combined_df['Avg'] = overall_avg['Avg.']
combined_df['Strike_Rate'] = overall_sr['Strike Rate']
combined_df['Balls_Faced'] = total_balls_faced['No_of_balls_faced']

# Sort the 'combined_df' DataFrame by runs scored in descending order
temp = combined_df.sort_values('Runs', ascending=False).reset_index()

# Create a line plot to visualize the average and strike rate of the top 50 run scorers
plt.figure(figsize=(20, 6))
plt.plot(temp["Player"][:50], temp["Avg"][:50], color='tab:orange')
plt.plot(temp["Player"][:50], temp["Strike_Rate"][:50], color='b')

# Add a legend and title to the plot
plt.legend(["Average", "Strike Rate"], loc="upper right")
plt.title("AVG VS STRIKE RATE OF TOP 50 RUN SCORERS")
plt.grid()
plt.xticks(rotation=90)

# Print the resulting DataFrame 'temp'
print(temp)
```

```
     index             Player  Runs        Avg  Strike_Rate  Balls_Faced
0      524        Virat Kohli  6283  37.398810   129.948294         4835
1      445      Shikhar Dhawan  5784  34.843373   126.647690         4567
2      401       Rohit Sharma  5611  31.172222   130.397397         4303
3      480       Suresh Raina  5528  32.517647   136.763978         4042
4      127       David Warner  5286  42.629032   140.547727         3761
..     ...                ...   ...        ...          ...          ...
540    279      Marco Jansen     0   0.000000     0.000000            3
541    212         Joe Denly     0   0.000000     0.000000            1
542    450    Shivil Kaushik     0   0.000000     0.000000            1
543     21    Ajantha Mendis     0   0.000000     0.000000            1
544    360  Prithvi Raj Yarra     0        NaN     0.000000            1

[545 rows x 6 columns]
```




## INFERENCE: FROM THE ABOVE GRAPH WE CAN GET THE INSIGHTS OF A PLAYERS AVG VS STRIKE RATE

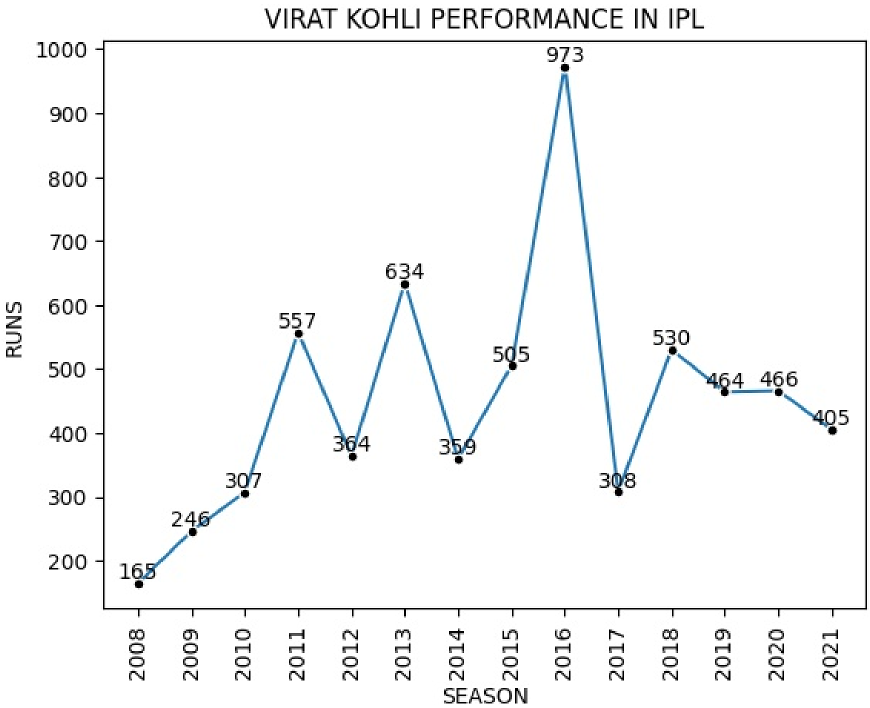
## 7. VIRAT KOHLI PERFORMANCE IN IPL

In [22]:
```python
# Filter the DataFrame 'df' to select data for the player 'Virat Kohli' and reset the index
virat_df = df.loc[df['Player'] == 'Virat Kohli'].reset_index()

# Create a list of seasons
season = ['2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020

# Add the 'Season' column to the 'virat_df' DataFrame
virat_df['Season'] = season

# Print the resulting DataFrame 'virat_df'
print(virat_df)
```

```
    index       Player  Mat  Inns  NO  Runs   HS    Avg   BF      SR  100  50  \
0      37  Virat Kohli   13    12   1   165   38  15.00  157  105.09    0   0
1     167  Virat Kohli   16    13   2   246   50  22.36  219  112.32    0   1
2     300  Virat Kohli   16    13   2   307   58  27.90  212  144.81    0   1
3     433  Virat Kohli   16    16   4   557   71  46.41  460  121.08    0   4
4     597  Virat Kohli   16    15   2   364  73*  28.00  326  111.65    0   2
5     734  Virat Kohli   16    16   2   634   99  45.28  457  138.73    0   6
6     900  Virat Kohli   14    14   1   359   73  27.61  294  122.10    0   2
7    1018  Virat Kohli   16    16   5   505  82*  45.90  386  130.82    0   3
8    1143  Virat Kohli   16    16   4   973  113  81.08  640  152.03    4   7
9    1301  Virat Kohli   10    10   0   308   64  30.80  252  122.22    0   4
10   1428  Virat Kohli   14    14   3   530  92*  48.18  381  139.10    0   4
11   1567  Virat Kohli   14    14   0   464  100  33.14  328  141.46    1   2
12   1712  Virat Kohli   15    15   4   466  90*  42.36  384  121.35    0   3
13   1848  Virat Kohli   15    15   1   405  72*  28.92  339  119.46    0   3

    4s  6s Season
0   18   4   2008
1   22   8   2009
2   26  12   2010
3   55  16   2011
4   33   9   2012
5   64  22   2013
6   23  16   2014
7   35  23   2015
8   83  38   2016
9   23  11   2017
10  52  18   2018
11  46  13   2019
12  23  11   2020
13  43   9   2021
```

In [23]:
```python
# Create a line plot using Seaborn to visualize Virat Kohli's performance in IPL over seasons
```

```
sns.lineplot(data=virat_df, x='Season', y='Runs', marker='o', markersize=5, markerfacecolor='black', markeredgew

# Label the x and y axes
plt.xlabel('SEASON')
plt.ylabel('RUNS')

# Set the plot title
plt.title('VIRAT KOHLI PERFORMANCE IN IPL')

# Rotate x-axis labels for better readability
plt.xticks(rotation=90)

# Annotate data points with the corresponding runs scored
for i, row in virat_df.iterrows():
    plt.text(i, row['Runs'], str(row['Runs']), ha='center', va='bottom')

# Show the plot
plt.show()
```




INFERENCE: VIRAT KOHLI LOWEST SCORE IS IN 2008 [168 RUNS] AND HIGHEST SCORE IS IN 2016 [973 RUNS] AND VIRAT CONSISTENTLY IS SCORING ABOVE 300 RUNS EVERY SEASON.

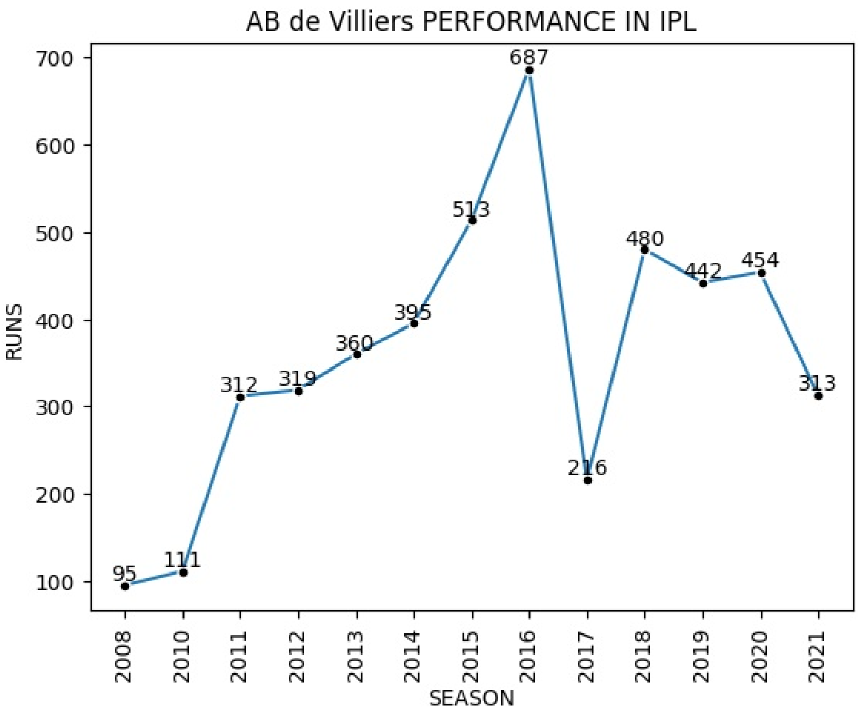
## 8. AB DE VILLIERS PERFORMANCE IN IPL

In [24]:
```
# Filter the DataFrame 'df' to select data for the player 'AB de Villiers' and reset the index
abd_df = df.loc[df['Player'] == 'AB de Villiers'].reset_index()

# Create a list of seasons
season = ['2008', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020', '2021

# Add the 'Season' column to the 'abd_df' DataFrame
abd_df['Season'] = season

# Print the resulting DataFrame 'abd_df'
print(abd_df)
```

| | index | Player | Mat | Inns | NO | Runs | HS | Avg | BF | SR | 100 | 50 | 4s | 6s | Season |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 62 | AB de Villiers | 6 | 6 | 1 | 95 | 26* | 19.00 | 98 | 96.93 | 0 | 0 | 5 | 1 | 2008 |
| 1 | 334 | AB de Villiers | 7 | 7 | 0 | 111 | 45 | 15.85 | 119 | 93.27 | 0 | 0 | 7 | 0 | 2010 |
| 2 | 457 | AB de Villiers | 16 | 13 | 4 | 312 | 65 | 34.66 | 243 | 128.39 | 0 | 2 | 21 | 14 | 2011 |
| 3 | 606 | AB de Villiers | 16 | 13 | 5 | 319 | 64* | 39.87 | 198 | 161.11 | 0 | 3 | 26 | 15 | 2012 |
| 4 | 749 | AB de Villiers | 14 | 14 | 4 | 360 | 64 | 36.00 | 219 | 164.38 | 0 | 2 | 34 | 15 | 2013 |
| 5 | 892 | AB de Villiers | 14 | 13 | 2 | 395 | 89* | 35.90 | 249 | 158.63 | 0 | 3 | 26 | 24 | 2014 |
| 6 | 1017 | AB de Villiers | 16 | 14 | 3 | 513 | 133* | 46.63 | 293 | 175.08 | 1 | 2 | 60 | 22 | 2015 |
| 7 | 1145 | AB de Villiers | 16 | 16 | 3 | 687 | 129* | 52.84 | 407 | 168.79 | 1 | 6 | 57 | 37 | 2016 |
| 8 | 1320 | AB de Villiers | 9 | 9 | 1 | 216 | 89* | 27.00 | 163 | 132.51 | 0 | 1 | 12 | 16 | 2017 |
| 9 | 1433 | AB de Villiers | 12 | 11 | 2 | 480 | 90* | 53.33 | 275 | 174.54 | 0 | 6 | 39 | 30 | 2018 |
| 10 | 1570 | AB de Villiers | 13 | 13 | 3 | 442 | 82* | 44.20 | 287 | 154.00 | 0 | 5 | 31 | 26 | 2019 |
| 11 | 1713 | AB de Villiers | 15 | 14 | 4 | 454 | 73* | 45.40 | 286 | 158.74 | 0 | 5 | 33 | 23 | 2020 |
| 12 | 1855 | AB de Villiers | 15 | 14 | 4 | 313 | 76* | 31.30 | 211 | 148.34 | 0 | 2 | 23 | 16 | 2021 |

In [25]:

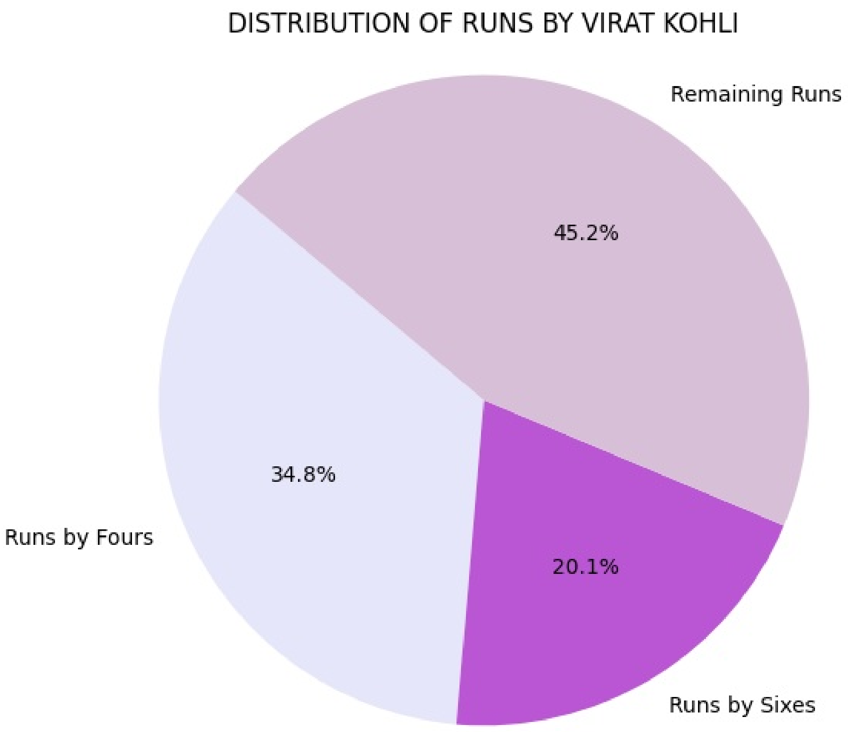
```python
# Create a line plot using Seaborn to visualize AB de Villiers' performance in IPL over seasons
sns.lineplot(data=abd_df, x='Season', y='Runs', marker='o', markersize=5, markerfacecolor='black', markeredgewi

# Label the x and y axes
plt.xlabel('SEASON')
plt.ylabel('RUNS')

# Set the plot title
plt.title('AB de Villiers PERFORMANCE IN IPL')

# Rotate x-axis labels for better readability
plt.xticks(rotation=90)

# Annotate data points with the corresponding runs scored
for i, row in abd_df.iterrows():
    plt.text(i, row['Runs'], str(row['Runs']), ha='center', va='bottom')

# Show the plot
plt.show()
```




INFERENCE: AB DE VILLIERS LOWEST SCORE IS IN 2008 [95 RUNS] AND HIGHEST SCORE IS IN 2016 [687 RUNS] AND AB CONSISTENTLY SCORED ABOVE 300 RUNS IN MOST OF THE SEASONS

## 9. VIRAT KOHLI DISTRIBUTION OF RUNS

In [26]:
```
# Calculate the total runs scored by Virat Kohli
total_runs = virat_df['Runs'].sum()

# Calculate the runs scored by fours and sixes
runs_by_fours = 4 * virat_df['4s'].sum()
runs_by_sixes = 6 * virat_df['6s'].sum()

# Calculate the remaining runs not scored by fours or sixes
remaining_runs = total_runs - runs_by_fours - runs_by_sixes

# Define labels, values, and colors for the pie chart
labels = ['Runs by Fours', 'Runs by Sixes', 'Remaining Runs']
sizes = [runs_by_fours, runs_by_sixes, remaining_runs]
colors = ['lavender', 'mediumorchid', 'thistle']

# Create a pie chart to visualize the distribution of runs by Virat Kohli
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)

# Set the aspect ratio to be equal to draw the pie chart as a circle
plt.axis('equal')

# Add a title to the pie chart
plt.title('DISTRIBUTION OF RUNS BY VIRAT KOHLI')

# Show the pie chart
plt.show()
```

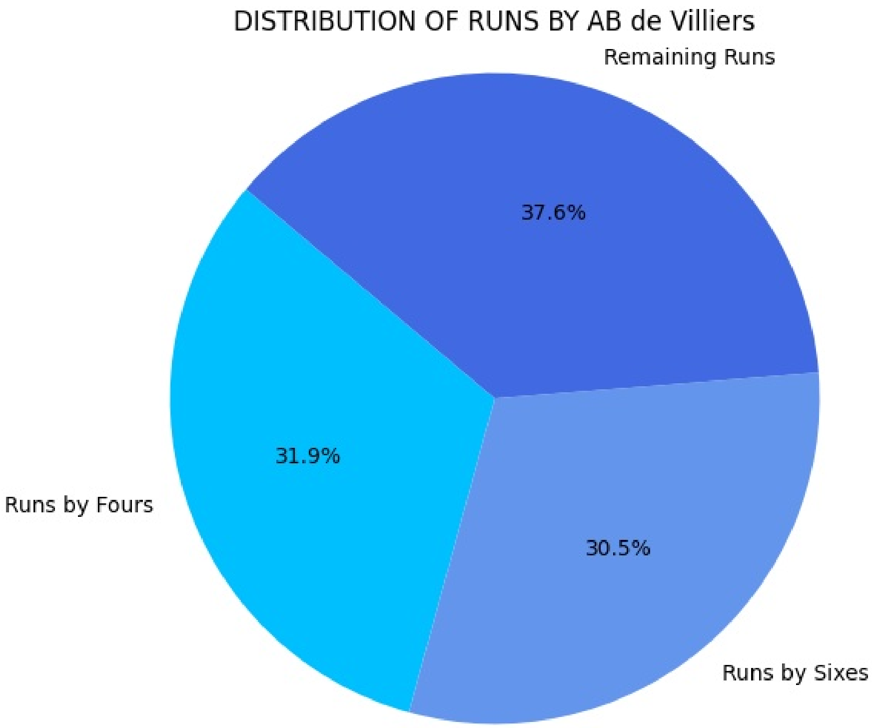



## INFERENCE: VIRAT KOHLI HAS SCORED 45% OF HIS RUNS BY 1'S, 2'S AND 3'S AND SCORED 34% OF HIS RUNS THROUGH 4'S AND 20% OF HIS RUNS THROUGH 6'S

## 10. AB DE VILLIERS DISTRIBUTION OF RUNS

In [27]:
```
# Calculate the total runs scored by AB de Villiers
total_runs = abd_df['Runs'].sum()

# Calculate the runs scored by fours and sixes
runs_by_fours = 4 * abd_df['4s'].sum()
runs_by_sixes = 6 * abd_df['6s'].sum()

# Calculate the remaining runs not scored by fours or sixes
remaining_runs = total_runs - runs_by_fours - runs_by_sixes

# Define labels, values, and colors for the pie chart
labels = ['Runs by Fours', 'Runs by Sixes', 'Remaining Runs']
sizes = [runs_by_fours, runs_by_sixes, remaining_runs]
colors = ['deepskyblue', 'cornflowerblue', 'royalblue']
```

```
# Create a pie chart to visualize the distribution of runs by AB de Villiers
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)

# Set the aspect ratio to be equal to draw the pie chart as a circle
plt.axis('equal')

# Add a title to the pie chart
plt.title('DISTRIBUTION OF RUNS BY AB de Villiers')

# Show the pie chart
plt.show()
```




INFERENCE: AB DE VILLIERS HAS SCORED 37% OF HIS RUNS BY 1'S, 2'S AND 3'S AND SCORED 31% OF HIS RUNS THROUGH 4'S AND 30% OF HIS RUNS THROUGH 6'S

# TASK 4: MACHINE LEARNING MODELS

## 1. LINEAR REGRESSION

In Linear regression we develop a model to predict average based on the stats of a player. Therefore, our target variable is Avg

In [28]:
```
# Display the first few rows of the DataFrame 'df' to provide an overview of its structure and content
df.head()
```

Out[28]:

| | Player | Mat | Inns | NO | Runs | HS | Avg | BF | SR | 100 | 50 | 4s | 6s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Shaun Marsh | 11 | 11 | 2 | 616 | 115 | 68.44 | 441 | 139.68 | 1 | 5 | 59 | 26 |
| **1** | Gautam Gambhir | 14 | 14 | 1 | 534 | 86 | 41.07 | 379 | 140.89 | 0 | 5 | 68 | 8 |
| **2** | Sanath Jayasuriya | 14 | 14 | 2 | 518 | 114* | 43.16 | 309 | 167.63 | 1 | 2 | 58 | 31 |
| **3** | Shane Watson | 15 | 15 | 5 | 472 | 76* | 47.20 | 311 | 151.76 | 0 | 4 | 47 | 19 |
| **4** | Graeme Smith | 11 | 11 | 2 | 441 | 91 | 49.00 | 362 | 121.82 | 0 | 3 | 54 | 8 |

## DATA CLEANING

Since the columns Player and HS are Strings and can't be usefull for linear regression model hence we drop those columns

In [29]:
```
# Drop the 'Player' and 'HS' (highest score) columns from the DataFrame 'df'
# and assign the resulting DataFrame to 'e_df'
e_df = df.drop(columns=['Player', 'HS'])

# Display the first few rows of the DataFrame 'e_df'
e_df.head()
```

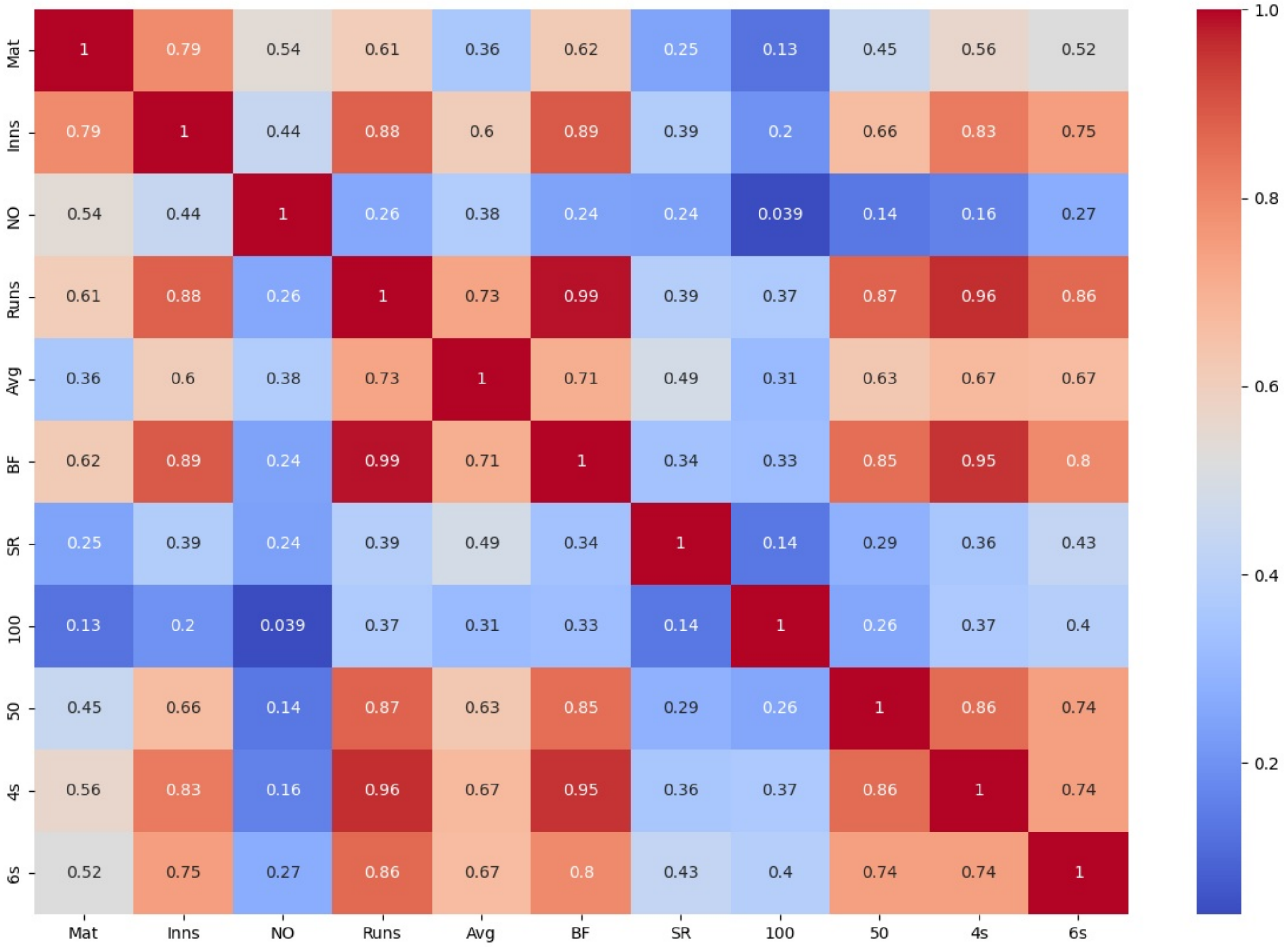
Out[29]:

| | Mat | Inns | NO | Runs | Avg | BF | SR | 100 | 50 | 4s | 6s |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 11 | 11 | 2 | 616 | 68.44 | 441 | 139.68 | 1 | 5 | 59 | 26 |
| 1 | 14 | 14 | 1 | 534 | 41.07 | 379 | 140.89 | 0 | 5 | 68 | 8 |
| 2 | 14 | 14 | 2 | 518 | 43.16 | 309 | 167.63 | 1 | 2 | 58 | 31 |
| 3 | 15 | 15 | 5 | 472 | 47.20 | 311 | 151.76 | 0 | 4 | 47 | 19 |
| 4 | 11 | 11 | 2 | 441 | 49.00 | 362 | 121.82 | 0 | 3 | 54 | 8 |

## CORRELATION MATRIX

In [30]:
```
corr = e_df.corr()
plt.figure(figsize=(15,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[30]: <Axes: >




Target variable is Avg. In correlation matrix those columns that have value above 0.5 for the target variable Avg is choosen and rest of the columns are dropped. Therefore the dropped columns [100,SR,NO,Mat].

In [31]:
```
# Drop the columns 'Mat' (matches played), '100' (centuries), 'NO' (not outs), and 'SR' (strike rate)
# from the DataFrame 'e_df'
e_df = e_df.drop(columns=['Mat', '100', 'NO', 'SR'])

# Display the first few rows of the modified DataFrame 'e_df'
e_df.head()
```

Out[31]:

| | Inns | Runs | Avg | BF | 50 | 4s | 6s |
|---|---|---|---|---|---|---|---|
| 0 | 11 | 616 | 68.44 | 441 | 5 | 59 | 26 |
| 1 | 14 | 534 | 41.07 | 379 | 5 | 68 | 8 |
| 2 | 14 | 518 | 43.16 | 309 | 2 | 58 | 31 |
| 3 | 15 | 472 | 47.20 | 311 | 4 | 47 | 19 |
| 4 | 11 | 441 | 49.00 | 362 | 3 | 54 | 8 |

In [32]:
```
# Create the feature matrix 'X' containing selected columns from the DataFrame 'e_df'
X = e_df[['Inns', 'Runs', 'BF', '50', '4s', '6s']]
```

```
# Create the target variable 'Y' representing batting averages from the DataFrame 'e_df'
Y = e_df['Avg']
```

In [33]:
```
# Import the train_test_split function from the scikit-learn library
from sklearn.model_selection import train_test_split

# Split the feature matrix 'X' and target variable 'Y' into training and testing sets
# with a test size of 1% and a specified random seed for reproducibility
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.01, random_state=0)
```

In [34]:
```
# Import the LinearRegression model from scikit-learn
from sklearn.linear_model import LinearRegression

# Create an instance of the LinearRegression model
regressor = LinearRegression()

# Fit the model to the training data
regressor.fit(x_train, y_train)

# Use the trained model to make predictions on the test data
y_pred = regressor.predict(x_test)

# Create a DataFrame 'CrossCheckData' to compare actual and predicted values
CrossCheckData = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Print the DataFrame 'CrossCheckData' to cross-check actual and predicted values
print(CrossCheckData)
```

```
      Actual  Predicted
1323   39.50  23.040902
76     20.00  12.772000
532    13.00  12.978881
631    23.87  25.948939
1505    9.66  10.298553
963    10.33  10.122503
889    51.25  48.740748
1810    7.00   8.284302
135     1.00   9.458163
18     27.45  18.705858
1016   49.09  53.166853
1341   15.40  13.423231
161    30.18  31.595864
617    30.62  32.265232
1412    1.00   8.751623
668    23.50  15.549498
242     7.50  10.433285
1540    1.00   7.295920
388     0.00  11.291395
1311   42.66  33.699820
```

# TASK 5: TESTING THE MODEL

In [35]:
```
# Print the accuracy score of the Linear Regression model on the test data
print('Accuracy:', regressor.score(x_test, y_test))
```

```
Accuracy: 0.825583008697394
```

The accuracy of the model is 82%

# 2. KMeans-Clustering

In k-means clustering we predict the Runs scored by the player based on the balls faced. To achieve this we form clusters and consider the df where each players total Runs is calculated [this is obtained from the "combined_df" that we have used before.

In [36]:
```
# Display the first few rows of the DataFrame 'combined_df' to provide an overview of its structure and content
combined_df.head()
```

Out[36]:

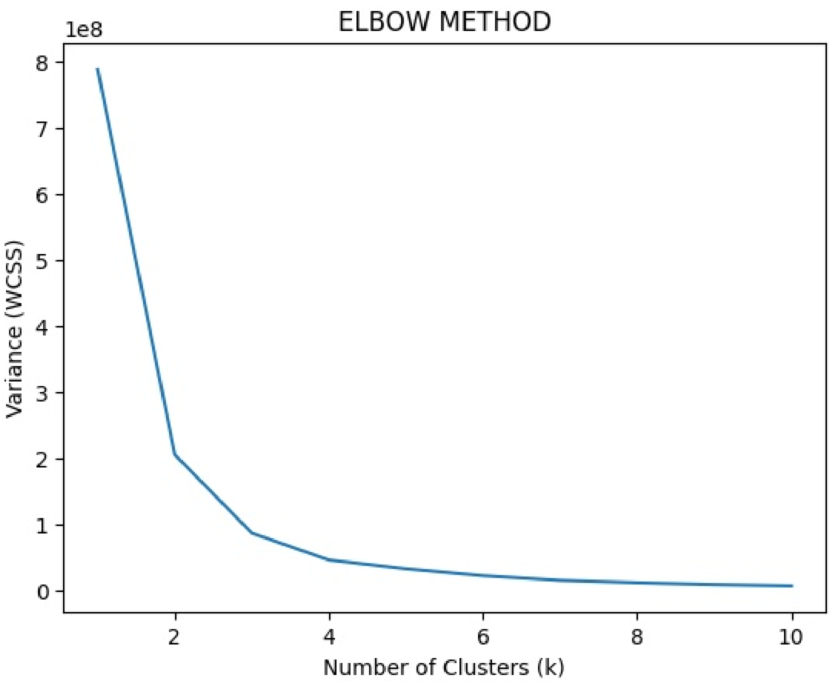
| | Player | Runs | Avg | Strike_Rate | Balls_Faced |
|---|---|---|---|---|---|
| **0** | AB de Villiers | 4697 | 38.818182 | 154.101050 | 3048 |
| **1** | Aakash Chopra | 53 | 8.833333 | 74.647887 | 71 |
| **2** | Aaron Finch | 2005 | 25.705128 | 127.707006 | 1570 |
| **3** | Abdul Samad | 222 | 15.857143 | 146.052632 | 152 |
| **4** | Abdur Razzak | 0 | NaN | 0.000000 | 2 |

Here we take Balls faced as x-axis and Runs as y-axis hence those two columns are selected

In [37]:
```
# Extract specific columns (columns 1 and 4) from the DataFrame 'combined_df' and convert them to a NumPy array
x = combined_df.iloc[:, [4,1]].values
```

We find out the number of clusters using ELBOW METHOD

In [38]:
```
# Import the KMeans clustering algorithm from scikit-learn
from sklearn.cluster import KMeans

wcss = []  # List to store the within-cluster sum of squares (WCSS)

# Iterate through a range of k-values to determine the optimal number of clusters
for i in range(1, 11):
    # Create a KMeans model with 'i' clusters using k-means++ initialization and other parameters
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)

    # Fit the KMeans model to the dataset 'x'
    kmeans.fit(x)  # You need to specify your dataset here

    # Calculate and store the WCSS for the current number of clusters
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method graph to visualize the optimal number of clusters
plt.plot(range(1, 11), wcss)
plt.title("ELBOW METHOD")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Variance (WCSS)")
plt.show()
```

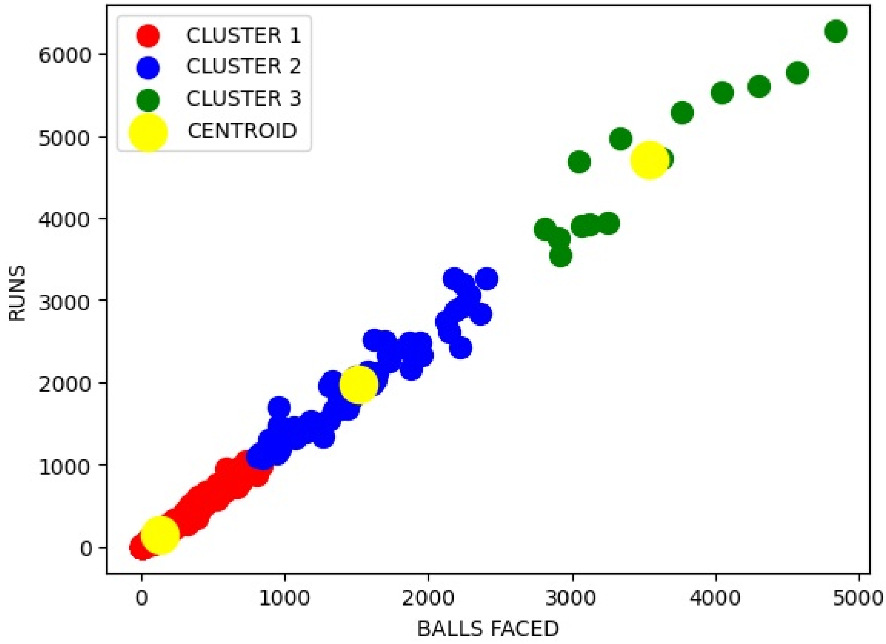



At k=3 there is a drastic change in graph. Hence the value of k is 3

In [39]:
```
# Create a KMeans clustering model with 3 clusters using k-means++ initialization and other parameters
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)

# Fit the KMeans model to the dataset 'x' and obtain cluster assignments for each data point
y_kmeans = kmeans.fit_predict(x)

# The variable 'y_kmeans' now contains the cluster labels for each data point
```

In [40]:
```
# Create a scatter plot to visualize the clustered data points and centroids
# Data points belonging to Cluster 1 are plotted in red, Cluster 2 in blue, and Cluster 3 in green
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s=100, c='red', label='CLUSTER 1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s=100, c='blue', label='CLUSTER 2')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s=100, c='green', label='CLUSTER 3')

# Plot the cluster centroids in yellow
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='yellow', label='CENTROID')

# Add a legend and labels to the plot
plt.legend()
plt.xlabel("BALLS FACED")
plt.ylabel("RUNS")
```

```python
# Show the scatter plot
plt.show()
```




## 3. RANDOM FOREST REGRESSOR

By using this machine model we predict the Total number of Runs scored based on the various parameters of the player. To achieve this we consider the "combined_df" that contains total runs of the player which was used earlier and we add certain parameters to the dataframe that are required for this model.

In [41]:
```python
# Display the first few rows of the DataFrame 'combined_df' to provide an overview of its structure and content
combined_df.head()
```

Out[41]:

| | Player | Runs | Avg | Strike_Rate | Balls_Faced |
|---|---|---|---|---|---|
| 0 | AB de Villiers | 4697 | 38.818182 | 154.101050 | 3048 |
| 1 | Aakash Chopra | 53 | 8.833333 | 74.647887 | 71 |
| 2 | Aaron Finch | 2005 | 25.705128 | 127.707006 | 1570 |
| 3 | Abdul Samad | 222 | 15.857143 | 146.052632 | 152 |
| 4 | Abdur Razzak | 0 | NaN | 0.000000 | 2 |

In [42]:
```python
# Calculate the total number of 4s, 6s, centuries (100s), and half-centuries (50s) for each player
total_4s = df.groupby('Player')['4s'].sum().reset_index()
total_6s = df.groupby('Player')['6s'].sum().reset_index()
total_100 = df.groupby('Player')['100'].sum().reset_index()
total_50 = df.groupby('Player')['50'].sum().reset_index()

# Update the 'combined_df' DataFrame with the calculated totals for 4s, 6s, 100s, and 50s
combined_df['4s'] = total_4s['4s']
combined_df['6s'] = total_6s['6s']
combined_df['100'] = total_100['100']
combined_df['50'] = total_50['50']

# Display the first few rows of the updated 'combined_df' DataFrame
combined_df.head()
```

Out[42]:

| | Player | Runs | Avg | Strike_Rate | Balls_Faced | 4s | 6s | 100 | 50 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | AB de Villiers | 4697 | 38.818182 | 154.101050 | 3048 | 374 | 239 | 2 | 37 |
| 1 | Aakash Chopra | 53 | 8.833333 | 74.647887 | 71 | 7 | 0 | 0 | 0 |
| 2 | Aaron Finch | 2005 | 25.705128 | 127.707006 | 1570 | 204 | 75 | 0 | 14 |
| 3 | Abdul Samad | 222 | 15.857143 | 146.052632 | 152 | 12 | 14 | 0 | 0 |
| 4 | Abdur Razzak | 0 | NaN | 0.000000 | 2 | 0 | 0 | 0 | 0 |

## Data cleaning by removing NULL values and unnecessary columns

In [43]:
```python
# Drop rows with missing values (NaN) from the 'combined_df' DataFrame and reset the index
combined_df = combined_df.dropna().reset_index()
```

```
# Display the first few rows of the updated 'combined_df' DataFrame
combined_df.head()
```

Out[43]:

| | index | Player | Runs | Avg | Strike_Rate | Balls_Faced | 4s | 6s | 100 | 50 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | AB de Villiers | 4697 | 38.818182 | 154.101050 | 3048 | 374 | 239 | 2 | 37 |
| **1** | 1 | Aakash Chopra | 53 | 8.833333 | 74.647887 | 71 | 7 | 0 | 0 | 0 |
| **2** | 2 | Aaron Finch | 2005 | 25.705128 | 127.707006 | 1570 | 204 | 75 | 0 | 14 |
| **3** | 3 | Abdul Samad | 222 | 15.857143 | 146.052632 | 152 | 12 | 14 | 0 | 0 |
| **4** | 5 | Abhimanyu Mithun | 32 | 8.000000 | 133.333333 | 24 | 4 | 1 | 0 | 0 |

In [44]:

```
# Drop the 'index' column from the 'combined_df' DataFrame along the 'axis=1' (columns) and update 'combined_df
combined_df.drop('index', axis=1, inplace=True)

# Display the first few rows of the updated 'combined_df' DataFrame
combined_df.head()
```

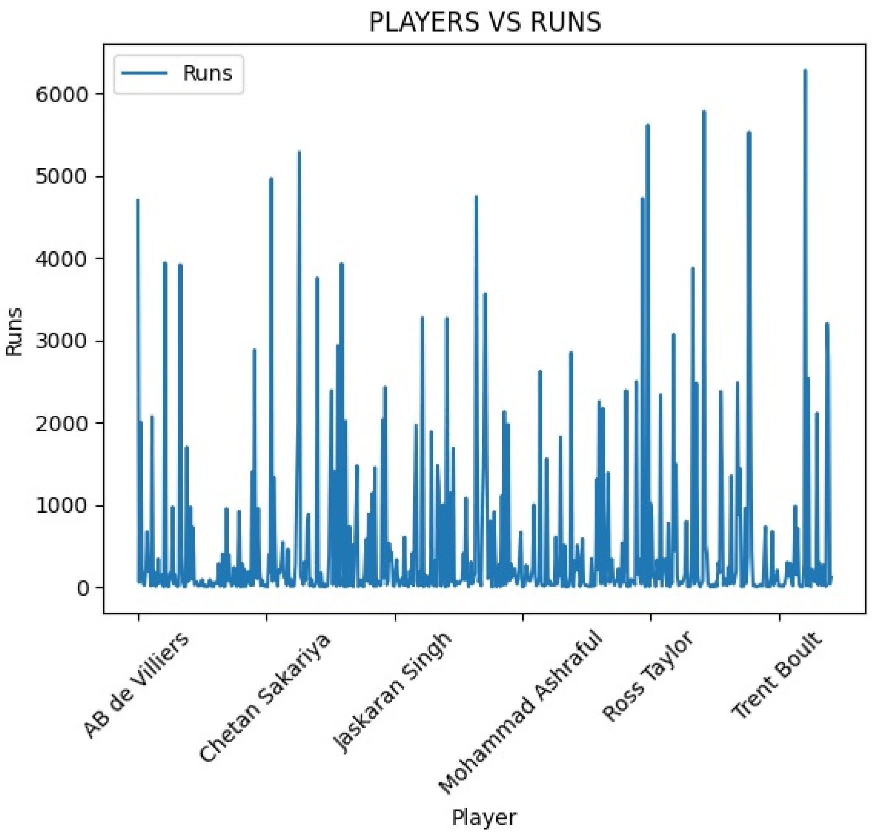
Out[44]:

| | Player | Runs | Avg | Strike_Rate | Balls_Faced | 4s | 6s | 100 | 50 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | AB de Villiers | 4697 | 38.818182 | 154.101050 | 3048 | 374 | 239 | 2 | 37 |
| **1** | Aakash Chopra | 53 | 8.833333 | 74.647887 | 71 | 7 | 0 | 0 | 0 |
| **2** | Aaron Finch | 2005 | 25.705128 | 127.707006 | 1570 | 204 | 75 | 0 | 14 |
| **3** | Abdul Samad | 222 | 15.857143 | 146.052632 | 152 | 12 | 14 | 0 | 0 |
| **4** | Abhimanyu Mithun | 32 | 8.000000 | 133.333333 | 24 | 4 | 1 | 0 | 0 |

In [45]:

```
# Create a plot of runs scored ('Runs') for each player, with player names on the x-axis
combined_df.plot(x='Player', y='Runs')

# Rotate the x-axis labels by 45 degrees for better readability
plt.xticks(rotation=45)
plt.title("PLAYERS VS RUNS")
plt.xlabel("Player")
plt.ylabel("Runs")

# Display the plot
```

Out[45]: Text(0, 0.5, 'Runs')




In [46]:

```
# Create the feature matrix 'X' containing selected columns from the 'combined_df' DataFrame
X = combined_df[['Balls_Faced', '4s', '6s', '50', '100']]

# Create the target variable 'Y' representing runs scored from the 'combined_df' DataFrame
Y = combined_df['Runs']
```

In [47]:
```python
# Import the train_test_split function from scikit-learn
from sklearn.model_selection import train_test_split

# Split the feature matrix 'X' and target variable 'Y' into training and testing sets
# with a test size of 1% and a specified random seed for reproducibility
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.01, random_state=0)
```

In [48]:
```python
# Import the RandomForestRegressor model from scikit-learn's ensemble module
from sklearn.ensemble import RandomForestRegressor

# Create an instance of the RandomForestRegressor model
regressor = RandomForestRegressor()

# The 'regressor' object now represents an instance of the RandomForestRegressor model
```

In [49]:
```python
# Train the RandomForestRegressor model on the training data (x_train and y_train)
regressor.fit(x_train, y_train)

# Use the trained model to make predictions on the test data (x_test)
y_pred = regressor.predict(x_test)

# Create a DataFrame 'CrossCheckData' to compare actual and predicted values
CrossCheckData = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Print the 'CrossCheckData' DataFrame to assess the model's performance
print(CrossCheckData)
```

```
     Actual    Predicted
380     454   469.500000
155       6     5.290857
132     785   826.550000
456    1417  1438.500000
90       22    18.595000
293     167   184.960000
```

# TASK 5: TESTING THE MODEL

In [50]:
```python
# Print the R-squared (coefficient of determination) as a measure of model accuracy on the test data
print('Accuracy:', regressor.score(x_test, y_test))
```

```
Accuracy: 0.9981624511932665
```

## The accuracy of the model is 99%