IPL BATTING ANALYSIS DATASET

SUBMITTED BY –

SAI SMARAN K.S

TL23A0047

ABSTRACT

The Indian Premier League (IPL), inaugurated in 2008, has matured into an enduring cricketing spectacle that transcends borders and exemplifies the confluence of sports, entertainment, and commerce. Over 14 seasons, the IPL has solidified its global prominence, captivating audiences worldwide and reshaping the contours of modern cricket. It is characterized by its unique franchise-based format, drawing top-tier cricketers and budding talents from across the globe, fostering an environment of fierce competition and camaraderie. The league's significance extends beyond the boundary ropes as it has played an instrumental role in talent development, propelling young cricketers to international stardom and reshaping the future of the sport.

The IPL's impact also extends to the realm of economics, where it has elevated cricket into a multi-billion-dollar industry. This abstract shed light on the league's diverse revenue streams, lucrative sponsorships, and monumental media rights deals, positioning it as one of the most financially successful sporting ventures globally. Furthermore, the IPL's global appeal and its contributions to the democratization of cricket are examined, with a focus on its pioneering innovations in broadcasting, digital media, and fan engagement, which have expanded the sport's reach beyond traditional boundaries. While facing various challenges over the years, including controversies and the balancing act between entertainment and cricketing integrity, the IPL has demonstrated resilience, reaffirming its status as a premier sporting event and an iconic chapter in the annals of cricket history.

INTRODUCTION

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.

As cricket enthusiasts and data scientists alike delve into the wealth of information made available through the IPL batting analysis dataset, it becomes evident that the IPL is more than just a cricket league; it is a reflection of the evolution of T20 cricket itself. This dataset serves as a gateway to understanding the intricacies of batting in the shortest format of the game, offering insights into player strategies, team dynamics, and the impact of changing pitch conditions over the years. In this report, we embark on a journey through this dataset, using advanced machine learning algorithms to unveil hidden patterns, dissect batting prowess, and predict future trends in IPL batting. It is a testament to how the fusion of sports and data science can enhance our understanding of the game and potentially provide valuable insights for teams, coaches, and cricket enthusiasts as they navigate the ever-evolving landscape of T20 cricket.

TASKS TO BE PERFORMED:

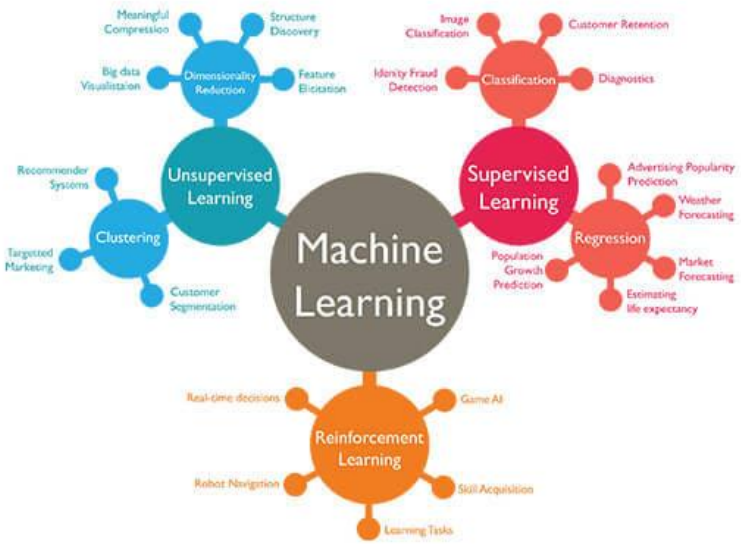
Task 1: Data Acquisition and Dataset Details

Task 2: Data Cleaning

Task 3: Data Visualization

Task 4: Data Modelling

Task 5: Testing the Model



1. DATA ACQUISITION AND DATASET DETAILS

Data acquisition, in the context of our data-centric age, represents the initial stage in the journey of transforming raw information into actionable knowledge. It involves the systematic gathering and recording of data from an array of sources, enabling organizations and individuals to extract valuable insights, make informed decisions, and drive innovation across various fields. In essence, data acquisition lays the foundation for the entire data analytics process, playing a pivotal role in our data-driven world.

The Earthquake dataset was selected from **KAGGLE WEBSITE** which offering a vast array of datasets, machine learning competitions, collaborative coding environments, and a vibrant community. It's a one-stop destination for data science, providing opportunities to hone skills, solve real-world problems, and engage in knowledge sharing.

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

[IPL - Player Performance Dataset\IPL - Player Performance Dataset\All Seasons Combined]

IMPORTING LIBRARIES

[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')
```

DATA ACQUISITION AND DATASET DETAILS

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

[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

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

[3]:

	Mat	Inns	NO	Runs	Avg	BF	SR	100	50	4s	6s
count	1986.000000	1986.000000	1986.000000	1986.000000	1986.000000	1986.000000	1986.000000	1986.000000	1986.000000	1986.000000	1986.000000
mean	8.974824	6.580060	1.527190	128.539778	18.257170	100.359013	110.863776	0.033233	0.654582	11.697885	4.798087
std	5.007739	4.841767	1.583134	155.137676	15.376013	114.014540	44.655957	0.205475	1.263126	15.458447	6.959908
min	1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	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	1.000000	0.000000
50%	9.000000	5.000000	1.000000	55.000000	16.000000	49.000000	116.270000	0.000000	0.000000	4.000000	2.000000
75%	14.000000	11.000000	2.000000	202.750000	27.345000	161.000000	135.282500	0.000000	1.000000	18.000000	7.000000
max	19.000000	19.000000	10.000000	973.000000	152.000000	640.000000	400.000000	4.000000	9.000000	88.000000	59.000000

INFERENCE: This confirms that the dataset contains 1986 rows

```
[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

From the above snapshots we come to a conclusion that the dataset contains 1986 rows and 13 columns.

DESCRIPTION OF EACH COLUMN

1. **Player:** This column contains the data of players name who have participated in IPL.
2. **Mat:** This column describes the number of matches played by the player.
3. **Inns:** This column represents the number of innings played by a player.
4. **NO:** This column stands for not out that is a number of times a player remained not out.
5. **Runs:** This column includes the total runs scored by a player.
6. **HS:** This column includes the highest score achieved by the player.

7. **Avg:** This column stands for average this means the average number of runs scored by a player before getting dismissed.
8. **BF:** This column represents the number of balls faced by the player.
9. **SR:** This column stands for strike rate this means it is a measure in cricket that indicates the pace at which a player scores runs, typically expressed as the number of runs scored per 100 balls faced.
10. **100:** This column stands for number of 100's scored by a player
11. **50:** This column stands for number of 50's scored by a player
12. **4s:** This column refers to number of 4's hit by a player.
13. **6s:** This column refers to number of 6's hit by a player.

PROBLEM STATEMENT:

In the context of IPL batting analysis, our objective is twofold: first, to predict a player's batting average based on a comprehensive set of performance metrics, and second, to forecast the total runs scored by a player given the number of balls they face. Leveraging machine learning algorithms, we seek to provide valuable insights into player consistency, proficiency, and contribution to their team's success. These predictions will serve as essential tools for cricket teams, analysts, and enthusiasts to assess and strategize around player performance in future IPL seasons, aiding in player selection and team optimization.

2. DATA CLEANING

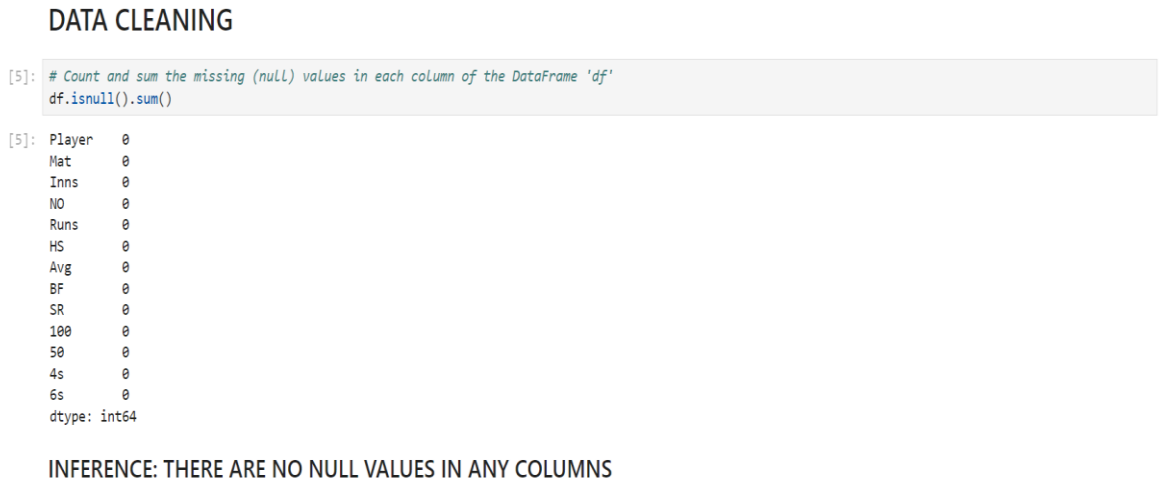
Data cleaning, often referred to as data cleansing or data scrubbing, is a crucial step in the data preprocessing pipeline. It involves the identification and correction of errors, inconsistencies, and inaccuracies in a dataset to ensure that the data is reliable, accurate, and suitable for analysis.

This process encompasses a range of tasks, including handling missing values, detecting and rectifying outliers, standardizing data formats, and resolving inconsistencies in naming conventions or encoding. Data cleaning is vital because real-world data is rarely pristine; it may contain errors introduced during data entry, inconsistencies between different data sources, or even inaccuracies due to sensor malfunction or measurement errors.

Effective data cleaning ensures that analytical models and algorithms can operate on high-quality data, leading to more accurate and robust results. It is a foundational step in the data science and machine learning workflow, enabling organizations and researchers to derive meaningful insights and make informed decisions from their data.

THREE STEPS INVOLVING IN DATA CLEANING ARE-

STEP 1: CHECKING FOR NULL VALUES


DATA CLEANING

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

[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

From the snapshoot above we can conclude that there are no null values

STEP 2: CHECKING FOR DUPLICATE ENTRIES

In the dataset while calculating the number of seasons played by each player I encountered that SHIKHAR DHAWAN has played 15 seasons while my data is only of 14 IPL seasons. We can conclude that the dataset has duplicate entries.

PLAYERS TO PLAY MOST NUMBER OF SEASONS

```
[6]: df['Player'].value_counts()
```

```
[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.

To verify what are duplicate entries we will print all the rows corresponding to SHIKHAR DHAWAN

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

[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

We can infer that the rows 218 and 219 are repeated. Now let's remove the duplicate rows. At present we know only duplicated related to Shikhar Dhawan in the dataset we don't know about others so let's print all the duplicates in the dataset

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

[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

Fron this we conclude that there are two duplicate entries in the dataset. This data may cause our model accuracy so we remove these duplicate entries.

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

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

[10]:

	Mat	Inns	NO	Runs	Avg	BF	SR	100	50	4s	6s
count	1984.000000	1984.000000	1984.000000	1984.000000	1984.000000	1984.000000	1984.000000	1984.000000	1984.000000	1984.000000	1984.000000
mean	8.975806	6.583165	1.528226	128.641129	18.266502	100.430948	110.868705	0.033266	0.655242	11.707661	4.802419
std	5.009262	4.843193	1.583516	155.182544	15.380920	114.048354	44.674902	0.205576	1.263592	15.463138	6.962059
min	1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	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	1.000000	0.000000
50%	9.000000	5.000000	1.000000	55.000000	16.000000	49.000000	116.270000	0.000000	0.000000	4.000000	2.000000
75%	14.000000	11.000000	2.000000	203.000000	27.357500	161.000000	135.297500	0.000000	1.000000	18.000000	7.000000
max	19.000000	19.000000	10.000000	973.000000	152.000000	640.000000	400.000000	4.000000	9.000000	88.000000	59.000000

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

After removing duplicate entries our dataset contains 1984 rows and 13 columns

STEP 3: REMOVING THE UNNECESSARY COLUMNS

This step is performed while we are developing the specific machine learning models.

3. DATA VISUALIZATION

Data visualization is the art and science of presenting data in visual forms such as charts, graphs, and maps to facilitate better understanding and interpretation. In an era of vast and complex datasets, data visualization plays a pivotal role in transforming raw information into meaningful insights. It enables individuals and organizations to uncover patterns, trends, and relationships, making data more accessible, actionable, and impactful. Effective data visualization not only enhances decision-making but also serves as a universal language for communicating findings, empowering data-driven narratives, and driving informed choices in various domains, from business analytics to scientific research and beyond.

1. PLAYERS TO PLAY MOST SEASONS FROM 2008-2021 [TOP 30]

STEP 1: Creating a new data frame that contains Players and number of seasons played

DATA VISUALIZATION

1.PLAYERS TO PLAY MOST SEASONS FROM 2008-2021

[11]:
```
# Count the number of seasons played by each player in the DataFrame 'df' and create a new DataFrame 'no_of_seasons'
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)
```

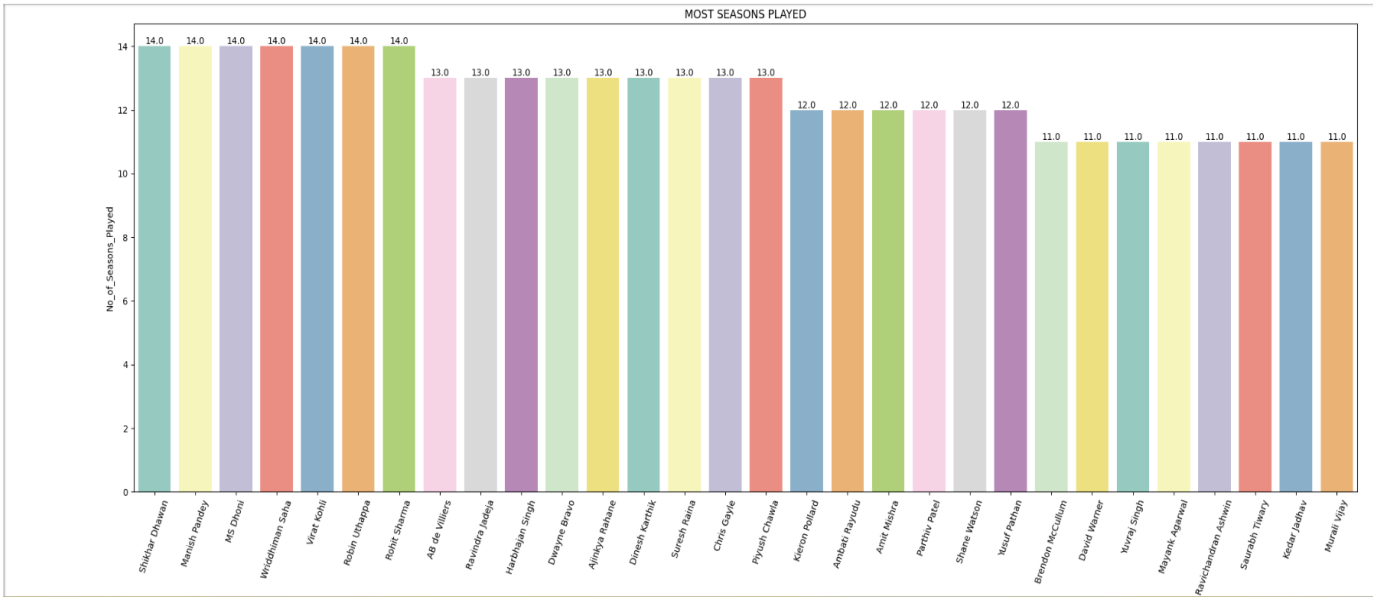
STEP 2: Sorting the data frame in descending order and selecting top 30 players

[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)
```

STEP 3: Plot the bar graph to visualize the data

[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' on the y-axis
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')
```

OUTPUT



From the graph we can infer that 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 PALYER PLAYED ATLEAST 3 SEASONS

STEP 1: Create a data frame that contains Player, mean runs per season and sort the values in descending order.

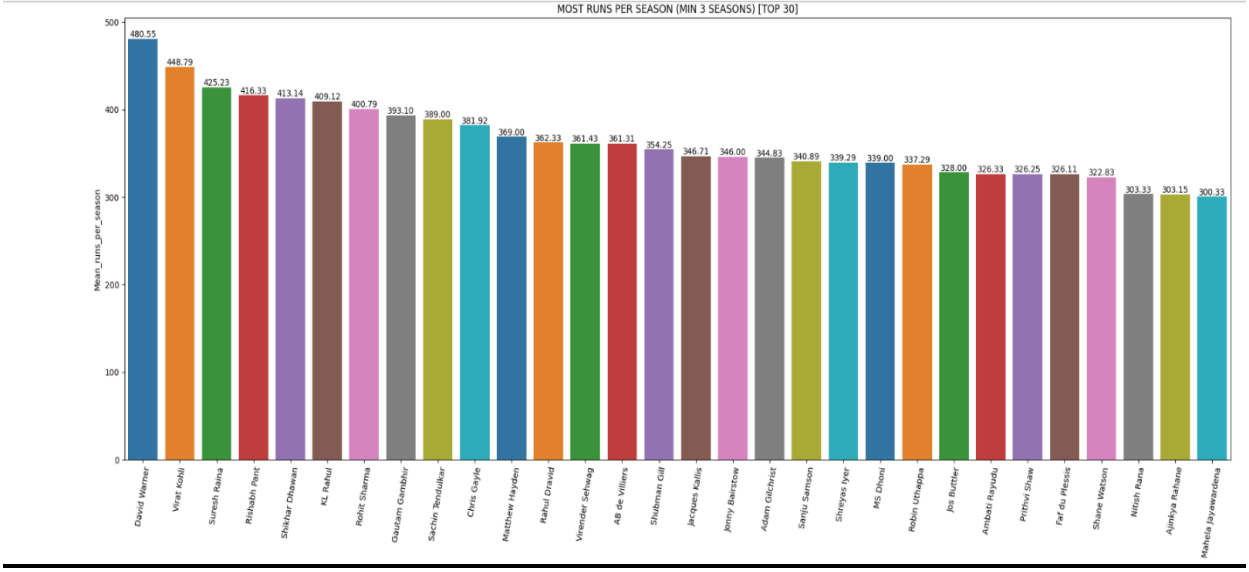
2. PLAYERS WITH MOST RUNS PER SEASON GIVEN THAT THE PLAYER HAS PLAYED ATLEAST 3 SEASONS [TOP 30]

```
[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']

# 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)
```

STEP 2: Plot the bar graph to visualize the data.

```
[15]: # 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' on the y-axis
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')
```

OUTPUT



Fron this data we can infer that DAVID WARNER has the highest runs per season that is 480.55

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

STEP 1: Create a data frame that contains Player, total runs per season and sort the values in descending order.

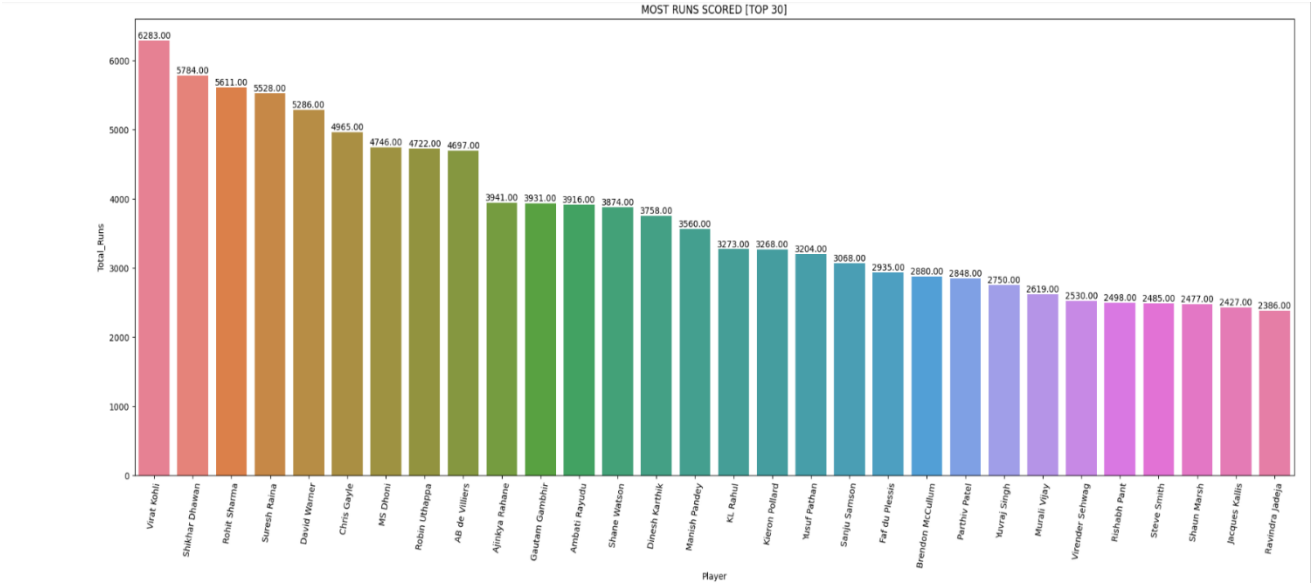
3. PLAYERS WITH MOST RUNS FROM 2008-2021 [TOP 30]

```
[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)
```

STEP 2: Plot the bar graph to visualize the data.

```
[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-axis
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)
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')
```

OUTPUT



From this we infer that VIRAT KOHLI has scored the most runs overall that is 6283.

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

STEP 1: Create a data frame that contains Player, Runs, Inns, Avg.

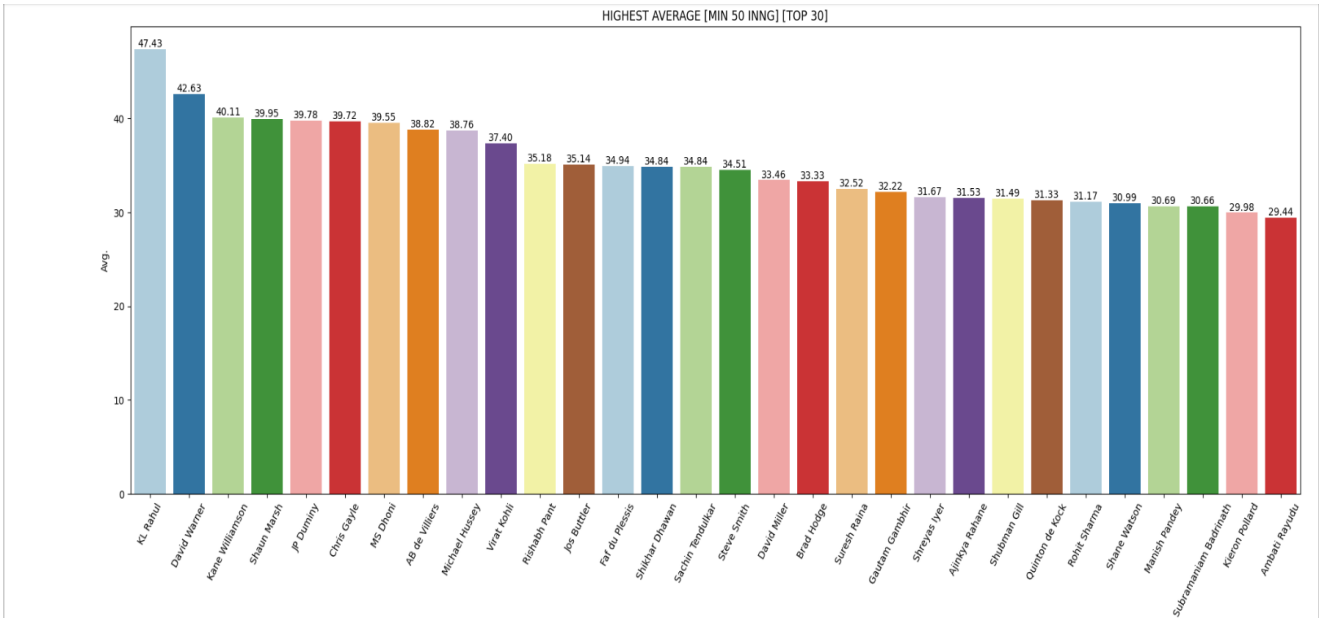
4. PLAYERS WITH HIGHEST AVERAGE [MIN 50 INNS] [TOP 30]

```
[18]: # 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)
```

STEP 2: Sort the values of Avg in descending order and plot the bar graph to visualize the data.

```
[19]: # 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=80)
# 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)
```

OUTPUT



From this we can infer that K L RAHUL has the highest Avg that is 47.43.

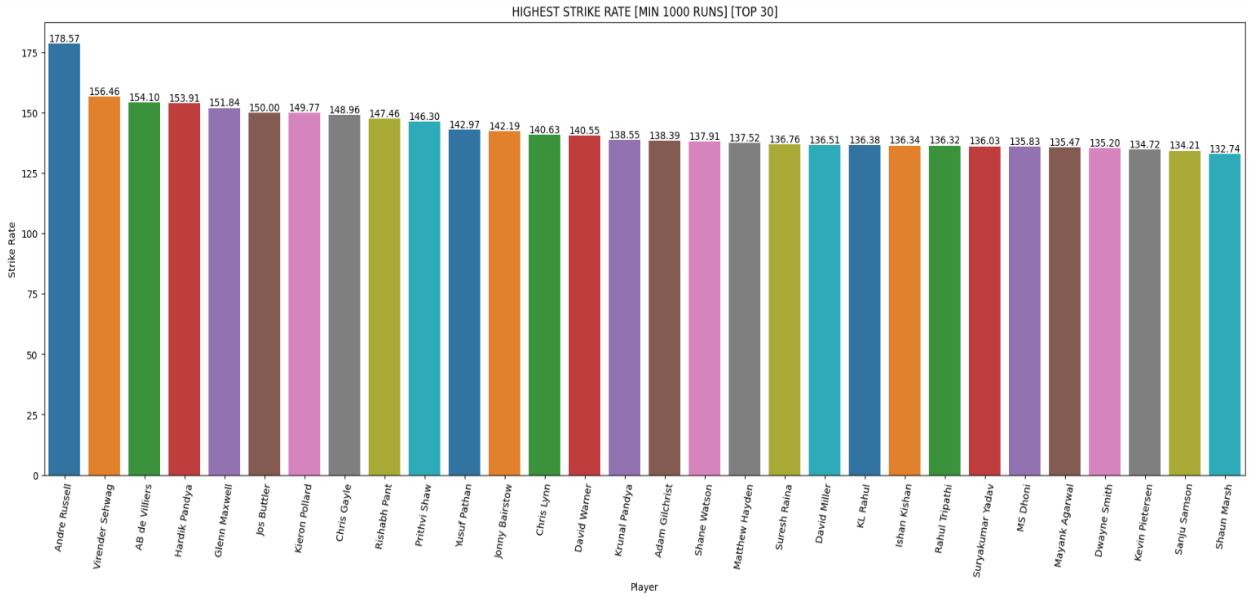
5. PLAYERS WITH HIGHEST STRIKE RATE [MIN 1000 RUNS]

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

[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-axis
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
plt.show()

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

OUTPUT



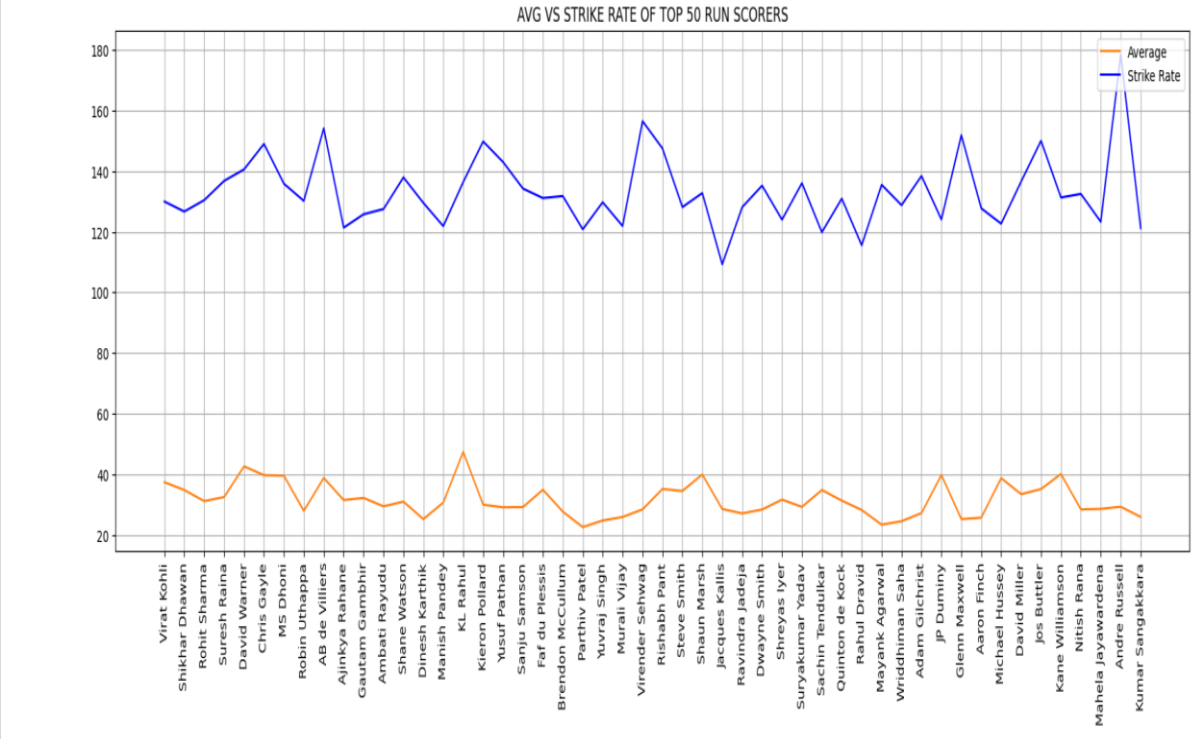
From this we can infer that ANDRE RUSSEL has the highest strike rate that is 178.57.

6. AVERAGE VS STRIKE RATE OF TOP 50 RUN SCORERS

6. AVERAGE VS STRIKE RATE OF TOP 50 RUN SCORERS

```
[21]: # 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)
```

OUTPUT



From this we can infer that we get the insights of a players average vs strike rate.

7. VIRAT KOHLI PERFORMANCE IN IPL

STEP 1: Creating a data frame that contains only the stats of VIRAT KOHLI

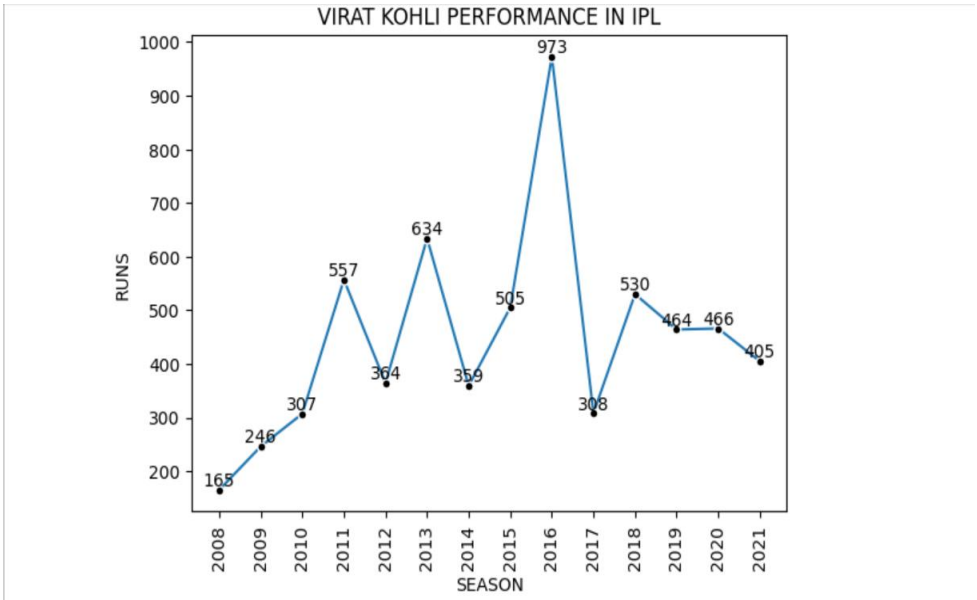
7. VIRAT KOHLI PERFORMANCE IN IPL

```
[22]: # 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', '2021']

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

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

STEP 2: Creating a line plot to visualize the runs scored by VIRAT KOHLI

```
[23]: # 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', markeredgewidth=1)

# 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()
```

OUTPUT



From this we can infer that VIRAT KOHLI lowest score is in 2008 [165 runs] and highest score is in 2016 [973 runs] and his scores are consistently above 300 runs every season.

8. AB DE VILLIERS PERFORMANCE IN IPL

STEP 1: Creating a data frame that contains only the stats of AB DE VILLIERS

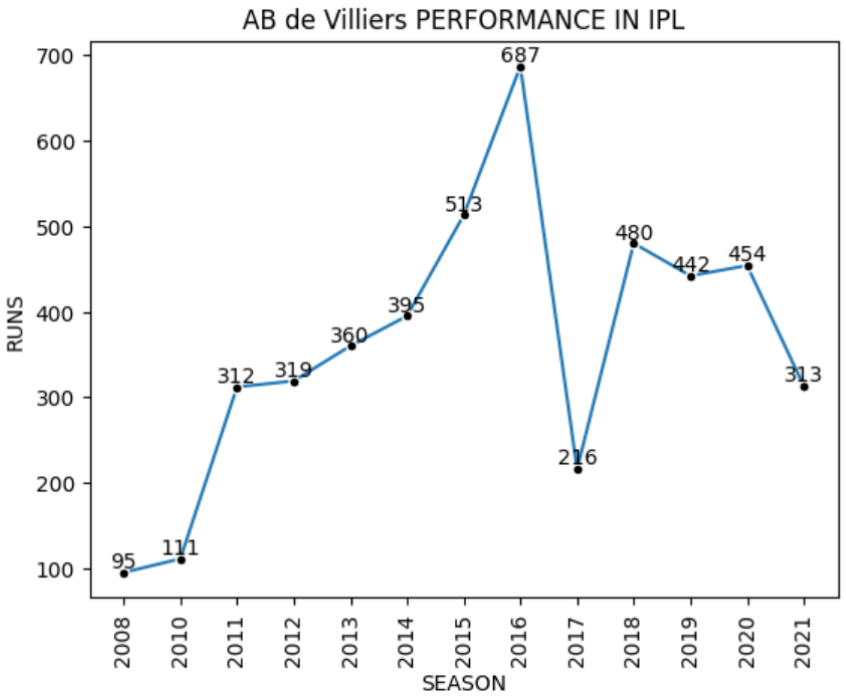
8. AB DE VILLIERS PERFORMANCE IN IPL

[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)
```

STEP 2: Creating a line plot to visualize the runs scored by AB DE VILLIERS

[25]:
```
# 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', markeredgewidth=1)

# 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()
```

OUTPUT



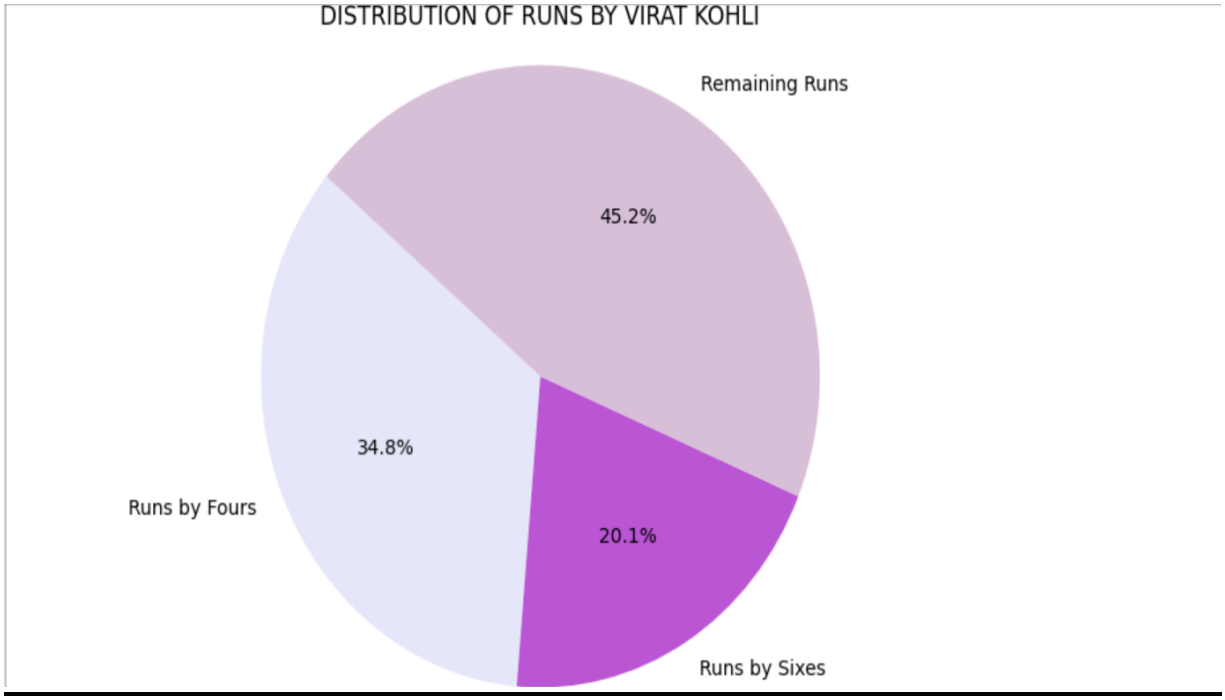
From this we can infer that AB DE VILLIERS lowest score is in 2008 [95 runs] and highest score is in 2016 [687 runs] and his scores are consistently above 300 runs every season.

9. VIRAT KOHLI DISTRIBUTION OF RUNS

9. VIRAT KOHLI DISTRIBUTION OF RUNS

[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()
```

OUTPUT



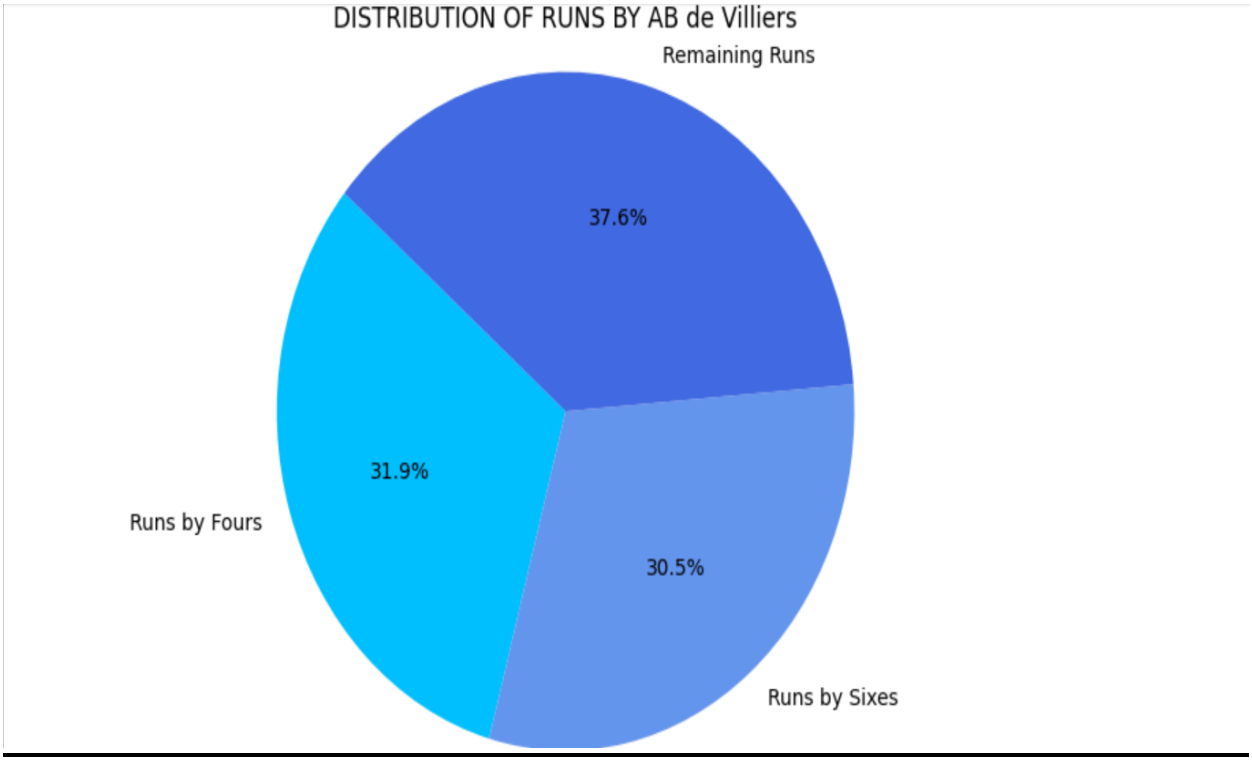
From this we can infer that VIRAT KOHLI has scored 45% of his runs by 1's, 2's, and 3's and scored 34% of his runs through 4's and 20% of his runs through 6's.

10. AB DE VILLIERS DISTRIBUTION OF RUNS

10. AB DE VILLIERS DISTRIBUTION OF RUNS

[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()
```

OUTPUT



From this we can infer that 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.

4. DATA MODELLING

Data modelling is a core component of data science and machine learning, where data is transformed into mathematical or computational representations to uncover patterns, make predictions, and gain insights. It involves various techniques, including supervised and unsupervised learning, to harness the value of data. Effective data modeling relies on data preprocessing, feature engineering, and algorithm selection, with well-constructed models providing actionable insights and automation capabilities. In an era of growing data complexity, data modelling plays a critical role in enabling organizations to harness the power of data for informed decision-making and innovation.

1. LINEAR REGRESSION

Linear regression is a statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a straight line (or hyperplane in higher dimensions) to the data. It's commonly used for predicting or estimating numeric outcomes based on input features, making it a fundamental tool in predictive modelling and statistical analysis.

STEP 1: DATA CLEANING

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

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

[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

[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()
```

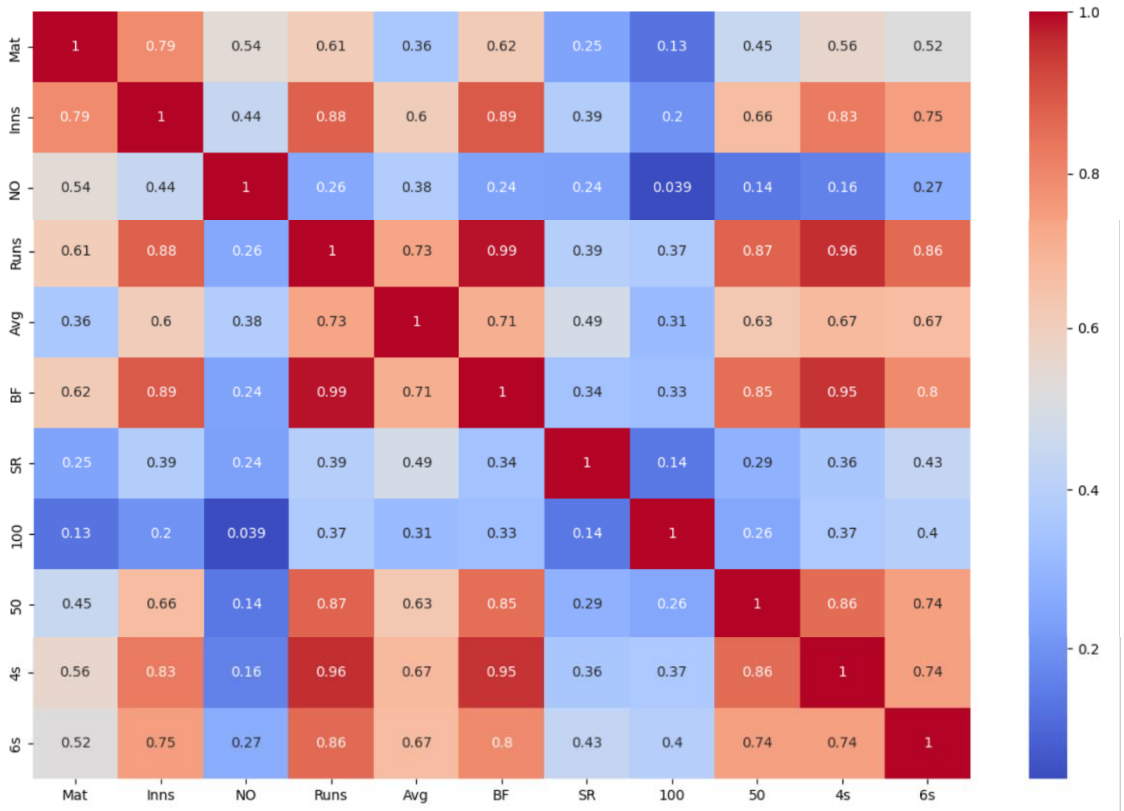
[29]:

	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

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

```
[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].

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

```
[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()
```

[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

STEP 3: SPLIT THE DATASET AS TRAINING AND TESTING DATA AND TRAIN THE MODEL

```
[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']
```

```
[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)
```

```
[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)
```

2. K-MEANS CLUSTERING

K-means clustering is a popular unsupervised machine learning technique used for data segmentation and pattern recognition. It aims to partition a dataset into distinct groups or clusters based on the similarity of data points. The algorithm iteratively assigns data points to the nearest cluster centroid and updates these centroids until convergence, effectively grouping similar data points together. K-means is versatile and finds applications in various fields, from customer segmentation in marketing to image compression in computer vision. However, it requires specifying the number of clusters (k) in advance, and its performance can be sensitive to the initial placement of centroids. Despite its simplicity and assumptions, K-means remains a fundamental and widely used clustering method for exploratory data analysis and pattern discovery.

STEP 1: CREATING THE DATAFRAME AND EXTRACTING THE REQUIRED COLUMNS

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.

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

[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

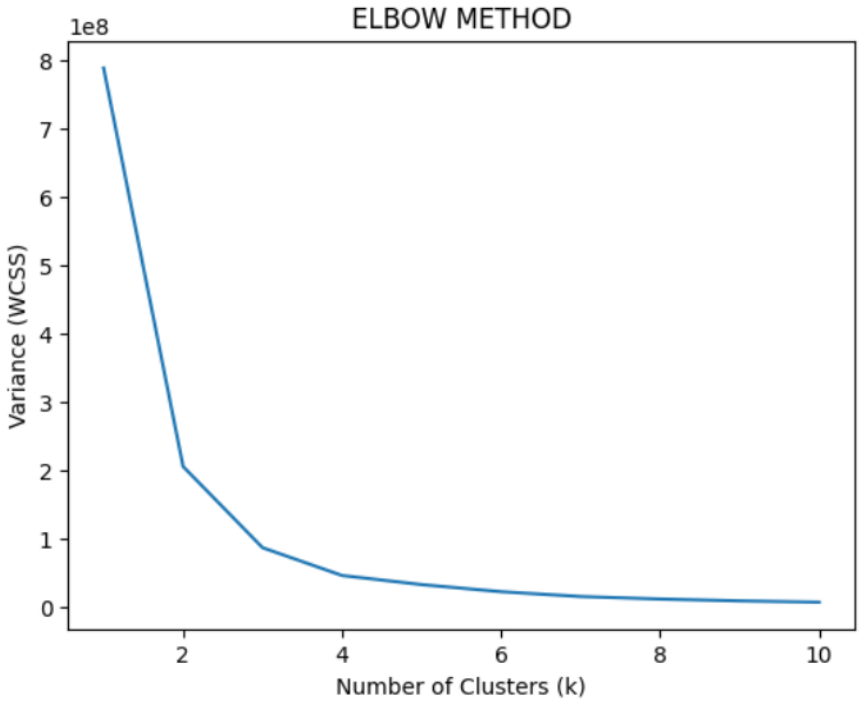
```
[37]: # Extract specific columns (columns 1 and 4) from the DataFrame 'combined_df' and convert them to a NumPy array
      X = combined_df.iloc[:, [4,1]].values
```

STEP 2: PERFORMING ELBOW METHOD TO CALCULATE K VALUE

We find out the number of clusters using ELBOW METHOD ¶

```
[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()
```

OUTPUT



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

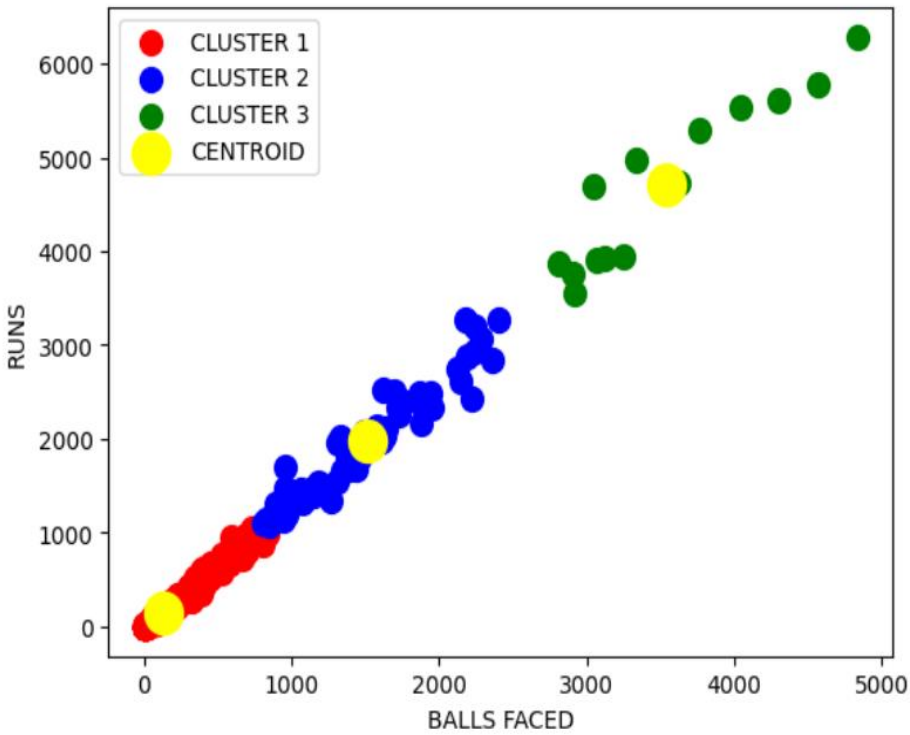
At k=3 we can observe a drastic change in graph. Hence consider Number of clusters as 3.

STEP 3: APPLYING K-MEANS ALGORITHM

[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
```

[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")

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

OUTPUT



3. RANDOM FOREST REGRESSOR

The Random Forest Regressor is a powerful machine learning algorithm that excels in predictive modelling for regression tasks. It belongs to the ensemble learning family and combines multiple decision trees to make robust predictions. Each decision tree is trained on a random subset of the data and features, resulting in a diverse set of individual models. The final prediction is an average or weighted combination of the predictions from all trees, which tends to be more accurate and less prone to overfitting than a single decision tree. Random Forest handles non-linear relationships, outliers, and high-dimensional data effectively. It's widely used in various domains, including finance, healthcare, and environmental science, to tackle regression problems such as predicting stock prices, medical diagnoses, and environmental outcomes. Its ability to provide accurate and robust predictions makes it a popular choice in the realm of regression modelling.

STEP 1: CREATING DATAFRAME AND DATA CLEANING

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. ¶

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

[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

Adding 4's, 6's, 100's, 50's to the data frame.

[42]:
```
# 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()
```

[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

Data cleaning by removing NULL values and unnecessary columns

[43]:
```
# 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()
```

[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

[44]:
```
# Drop the 'index' column from the 'combined_df' DataFrame along the 'axis=1' (columns) and update 'combined_df' in place
combined_df.drop('index', axis=1, inplace=True)

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

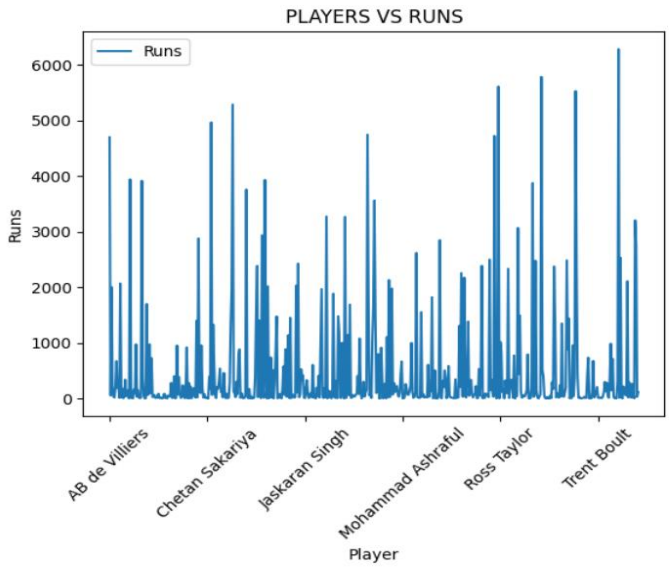
[44]:

	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

STEP 2: DISPLAYING A PLOT OF PLAYERS VS RUNS

[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
```

OUTPUT



STEP 3: APPLYING THE ALGORITHM BY SPLITTING THE DATA INTO TRAINING AND TESTING DATA

Splitting the data

[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']
```

[47]:
```
# 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)
```

Applying RandomForestRegressor Algorithm

[48]:
```
# 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
```

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

5. TESTING THE MODEL

LINEAR REGRESSION

ACTUAL VALUES VS PREDICTED VALUES

```
# 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

ACCURACY PREDICTION

```
[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%

The LINEAR REGRESSION model has accuracy of 82.5%.

RANDOM FOREST REGRESSOR

ACTUAL VALUES VS PREDICTED VALUES

```
# 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   479.120000
155       6     5.492952
132     785   815.770000
456    1417  1447.480000
90       22    18.250000
293     167   178.430000
```

ACCURACY PREDICTION

```
[50]: # 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: 0.9982367619357162
```

The accuracy of the model is 99% ¶

The Accuracy of RANDOM FOREST REGRESSOR model is 99.8%.

FUTURE ENHANCEMENTS

1. Real-Time Updates: Continuously update the dataset with data from each IPL season to keep it current and relevant. This ensures that analysts and researchers have access to the latest player statistics for ongoing and future seasons.

2. Additional Features: Incorporate new player performance metrics or additional contextual data such as match conditions (e.g., pitch type, weather), player injuries, and team dynamics. These additional features can provide a more comprehensive view of player performance.

3. Player Profiles: Develop individual player profiles that include career statistics, performance trends over multiple seasons, and historical comparisons. This can aid in assessing a player's evolution and consistency over time.

4. Advanced Analytics: Implement advanced analytics techniques, such as player clustering or player similarity analysis, to uncover hidden patterns and insights within the dataset. This can help in identifying unique player profiles or playing styles.

5. Predictive Modelling: Create machine learning models for predicting player performance metrics, such as batting average or runs scored, based on historical data. These models can assist in making data-driven predictions for upcoming seasons.

6. Visualization Tools: Build interactive data visualization tools and dashboards that allow users to explore and analyze the dataset visually. This can make it easier for cricket enthusiasts, analysts, and teams to derive insights.

7. Integration with Match Data: Integrate batting data with match-level data, including bowling statistics, team performance, and match outcomes. This comprehensive dataset can enable holistic analysis of player contributions to match results.

8. Fan Engagement: Develop features or applications that engage cricket fans, such as fantasy cricket platforms, mobile apps, or websites that provide real-time player statistics, fantasy league predictions, and in-depth player profiles.

9. Historical Analysis: Conduct historical analysis to identify trends and milestones in IPL batting performance. This can involve tracking record-breaking performances, significant partnerships, and memorable innings.

10. Collaborative Research: Encourage collaboration with cricket researchers, analysts, and enthusiasts to contribute to and enhance the dataset. Collaboration can lead to novel insights and research findings.

11. Machine Learning for Player Evaluation: Implement machine learning algorithms for player evaluation, ranking, or team composition optimization. These tools can aid IPL teams in making strategic decisions regarding player selection and team formation.

12. User Feedback: Actively seek feedback from users, analysts, and researchers to understand their specific needs and preferences for dataset enhancements. User input can guide future improvements and ensure that the dataset remains valuable to the cricket community.

By implementing these future enhancements, the IPL batting analysis dataset can continue to serve as a valuable resource for cricket analysis, research, and fan engagement, while keeping pace with the evolving dynamics of the Indian Premier League.

CONCLUSION

In conclusion, the IPL batting analysis dataset spanning 14 seasons from 2008 to 2022 serves as a comprehensive and invaluable resource for cricket enthusiasts, analysts, and researchers alike. With detailed player statistics, including runs scored, batting averages, strike rates, centuries, and more, this dataset provides deep insights into the performances of some of the most celebrated cricketers in the Indian Premier League. Its potential applications are vast, ranging from in-depth player evaluations and performance predictions to the exploration of evolving cricket trends and strategies. Furthermore, its dynamic nature allows for continuous updates and enhancements, ensuring its relevance in the ever-evolving world of T20 cricket. Whether for strategic team decisions, fan engagement, or academic research, this dataset offers a rich tapestry of cricketing data that continues to captivate and inform the cricket community, making it an indispensable asset in the analysis of IPL batting prowess.