In [1]:

```
#Importing major libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
```

1. **Line Plot**:
   - **Definition**: A line plot connects data points with a line, showing how a variable changes over time or across an ordered category.
   - **When to Use**: Best for showing trends or changes in data over time.
   - **Example**: Stock prices over time, temperature changes across days.

Why do we visualize data, and how does it help in understanding information better?

We visualize data to **see patterns, trends, and insights** that might be hard to understand just by looking at raw numbers. It makes the data easier to understand and helps us make better decisions quickly.

Univariate, Bivariate, and Multivariate Analysis

1. **Univariate Analysis**:
   - **Definition**: This is the simplest form of data analysis where we analyze a single variable.
   - **Purpose**: The goal is to describe the variable’s distribution, central tendency, and spread.
   - **Statistical Methods**:
     - Mean, median, mode
     - Variance, standard deviation
     - Frequency distribution
   - **Graphs Used**:
     - Histograms
     - Box plots
     - Pie charts
     - Bar charts
2. **Bivariate Analysis**:
   - **Definition**: This type of analysis involves examining the relationship between two variables.

   - **Purpose**: To find if and how two variables are related (e.g., correlation).
   - **Statistical Methods**:
     - Correlation coefficient (e.g., Pearson, Spearman)
     - Regression analysis
   - **Graphs Used**:
     - Scatter plots
     - Line plots
     - Bar charts (grouped or stacked)
     - Heatmaps

3. **Multivariate Analysis**:

   - **Definition**: This analysis deals with more than two variables at once.
   - **Purpose**: To understand relationships between multiple variables simultaneously.
   - **Statistical Methods**:
     - Multiple regression
   - **Graphs Used**:
     - Pair plots (scatterplot matrix)
     - 3D scatter plots
     - Heatmaps
     - Parallel coordinate plots

1. **Line Plot**:
   - **Definition**: A line plot connects data points with a line, showing how a variable changes over time or across an ordered category.
   - **When to Use**: Best for showing trends or changes in data over time.
   - **Example**: Stock prices over time, temperature changes across days.

In [2]:

```
# bivariate analysis
# timeseries analysis
# trend,up,down
```

`plt.title()`, `plt.xlabel()`, and `plt.ylabel()` in Matplotlib:

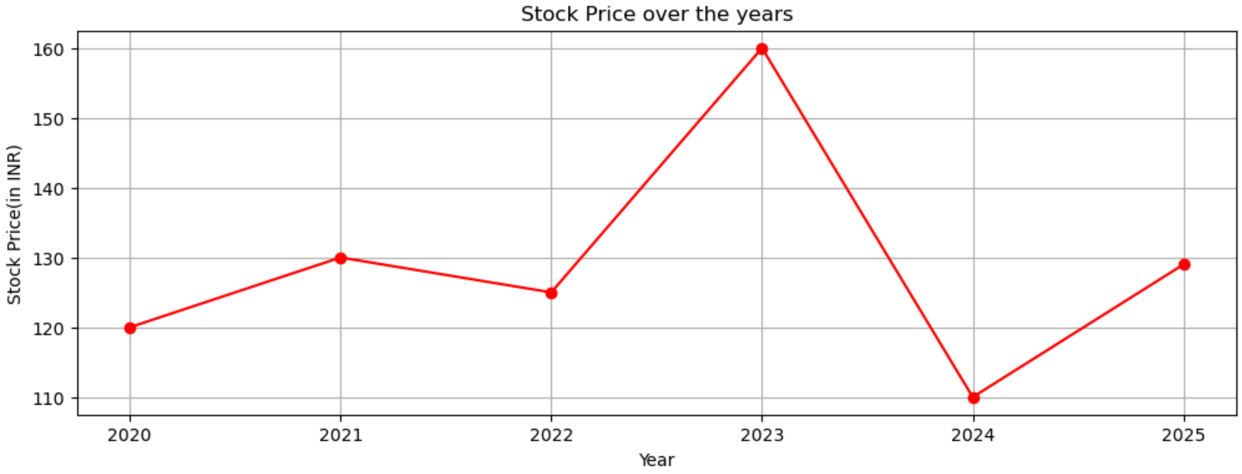
- **`plt.title()`**:

    - Adds a **title** to the plot.
    - Example: `plt.title('My Plot Title')`
- **`plt.xlabel()`** :
    - Adds a **label to the x-axis**.
    - Example: `plt.xlabel('X-Axis Label')`
- **`plt.ylabel()`** :
    - Adds a **label to the y-axis**.
    - Example: `plt.ylabel('Y-Axis Label')`

These functions are essential for improving the readability and context of your plots by providing titles and axis labels.

In [3]:
```
stock=[120,130,125,160,110,129]
year=[2020,2021,2022,2023,2024,2025]
plt.figure(figsize=(12,4))
plt.plot(year,stock,marker='o',color='Red')
plt.title('Stock Price over the years')
plt.xlabel('Year')
plt.ylabel('Stock Price(in INR)')
plt.grid()
plt.show()

#r --> red,b--> blue,g--> green,k---> black,c--> cyan  , we take hrm code from
```



Markers in Matplotlib:

- Markers are used to represent individual data points in plots such as line plots and scatter plots.

- You can specify markers using the `marker` parameter in plotting functions like `plt.plot()` or `plt.scatter()`.

- **Common Marker Symbols**:

    - `'o'` : Circle
    - `'.'` : Point
    - `','` : Pixel
    - `'x'` : X
    - `'+'` : Plus
    - `'*'` : Star
    - `'D'` : Diamond
    - `'s'` : Square
    - `'v'` : Triangle down
    - `'^'` : Triangle up
    - `'<'` : Triangle left
    - `'>'` : Triangle right
    - `'p'` : Pentagon hed line with circle markers
- **Marker Size and Color**:

    - `markersize` or `ms` : Controls the size of the marker.
        - Example: `plt.plot(x, y, marker='o', markersize=10)`
    - `markerfacecolor` or `mfc` : Sets the color inside the marker.
    - `markeredgecolor` or `mec` : Sets the edge color of the marker.

      ```
      plt.plot(x, y, marker='o', markersize=10,
      markerfacor any plot to improve visualization and
      clarity.
      ```

In [4]:

```
sharmakohli=pd.read_csv('sharma-kohli.csv')
sharmakohli
```

Out[4]:

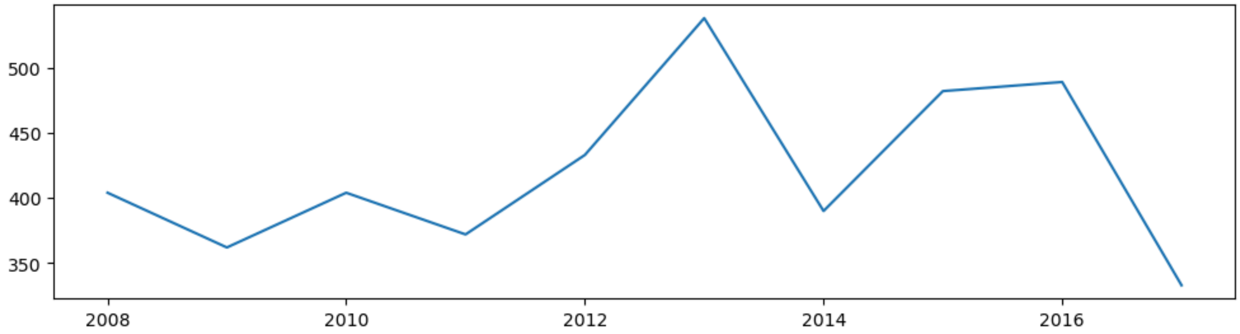
	index	RG Sharma	V Kohli
0	2008	404	165
1	2009	362	246
2	2010	404	307
3	2011	372	557
4	2012	433	364
5	2013	538	639
6	2014	390	359
7	2015	482	505
8	2016	489	973
9	2017	333	308

In [5]:

```
plt.figure(figsize=(12,3))
plt.plot(sharmakohli['index'],sharmakohli['RG Sharma'])
```

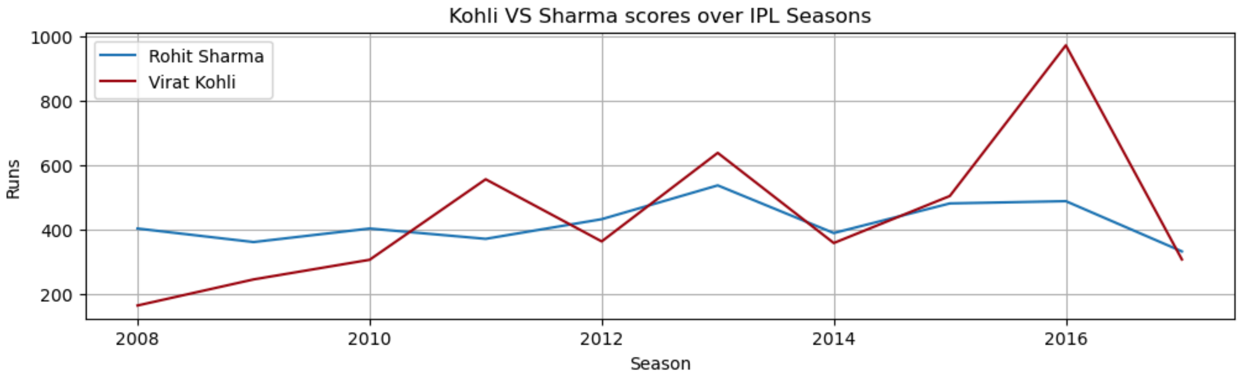
Out[5]:

```
[<matplotlib.lines.Line2D at 0x17260b88a50>]
```



In [6]:

```
plt.figure(figsize=(12,3))
plt.plot(sharmakohli['index'],sharmakohli['RG Sharma'],label='Rohit Sharma')
plt.plot(sharmakohli['index'],sharmakohli['V Kohli'],label='Virat Kohli',color
plt.xlabel('Season')
plt.ylabel('Runs')
plt.title('Kohli VS Sharma scores over IPL Seasons')
plt.grid()
plt.legend(loc='best')
plt.show()
```



`plt.legend()` and its `loc` parameter in Matplotlib:

- **`plt.legend()`**:
  - Adds a legend to the plot, which helps label different plot elements (like lines, bars, etc.).
  - The location of the legend can be specified using the `loc` parameter.
- **`loc` Parameter**:
  - Controls the position of the legend within the plot.
  - You can use:
    - **Strings**: Predefined positions like `'upper right'`, `'upper left'`, `'lower right'`, etc.
    - **Numerical Codes**: Shortcuts for the legend location.
      - Example: `0` (best), `1` (upper right), `2` (upper left), `3` (lower left), `4` (lower right), etc.
    - **Best Location (`loc=0`)**: Automatically places the legend at the optimal position.
- **Common Positions**:
  - `'best'` or `0`: Best position automatically chosen.
  - `'upper right'` or `1`: Top-right corner.
  - `'upper left'` or `2`: Top-left corner.
  - `'lower left'` or `3`: Bottom-left corner.
  - `'lower right'` or `4`: Bottom-right corner.
  - `'right'`: Right center.
  - `'center left'`: Left center.
  - `'center right'`: Right center.
  - `'center lHere are the color codes used in`

matplotlib.pyplot ` ( plt`):

- **Basic Color Codes (Single Letter)**:
  - `'b'` = blue
  - `'g'` = green
  - `'r'` = red
  - `'c'` = cyan
  - `'m'` = magenta
  - `'y'` = yellow
  - `'k'` = black
  - `'w'` = white
- **Hex Color Codes**:
  - Hexadecimal color codes are used, starting with `#`, followed by 6 digits.
  - Example: `'#FF5733'` for a shade of orange-red.
- **RGB Color Codes**:
  - You can specify RGB values as a tuple of three values ranging from 0 to 1.
  - Example: `(1.0, 0.0, 0.0)` for red.
- **Named Colors**:
  - Matplotlib also accepts named colors.
  - Example: `'skyblue'`, `'salmon'`, `'limegreen'`.
- **Grayscale**:
  - You can use a float between 0 and 1 for grayscale colors.
  - Example: `0.5` for a medium gray.

These color codes can be applied in the `color` parameter for plotting functions (e.g., `plt.plot(color='r')`).d81f-d015-4abf-b539-7bfed537121e.png)

2. **Scatter Plot**:
   - **Definition**: A scatter plot displays individual data points on a 2D plane, with each point representing the values of two variables.
   - **When to Use**: Useful for identifying relationships or correlations between two variables.
   - **Example**: Weight vs. height, age vs. income.

In [7]:
```
#USE IN Bivariate analysis
```

```
# numerical vs numerical
#trends , regression,correalations,clusters
```

In [8]:

```
batter=pd.read_csv('batter.csv').head(200)
batter
```

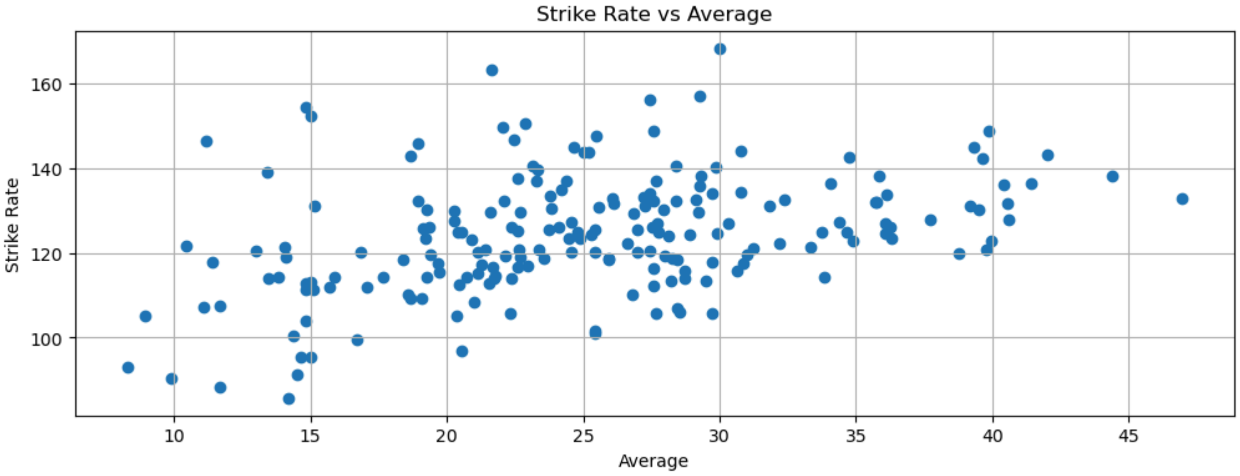
Out[8]:

	batter	runs	avg	strike_rate
0	V Kohli	6634	36.251366	125.977972
1	S Dhawan	6244	34.882682	122.840842
2	DA Warner	5883	41.429577	136.401577
3	RG Sharma	5881	30.314433	126.964594
4	SK Raina	5536	32.374269	132.535312
...	...	...	...	...
195	SP Fleming	196	21.777778	114.619883
196	JC Archer	195	15.000000	152.343750
197	AS Raut	194	21.555556	112.790698
198	BJ Rohrer	193	27.571429	132.191781
199	Salman Butt	193	27.571429	112.209302

200 rows × 4 columns

In [9]:

```
plt.figure(figsize=(12,4))
plt.scatter(batter.avg,batter.strike_rate)
plt.xlabel('Average')
plt.ylabel('Strike Rate')
plt.title('Strike Rate vs Average')
plt.grid()
plt.show()
```



In [10]:

```
batter[batter.avg>45]
```

Out[10]:

	batter	runs	avg	strike_rate
13	KL Rahul	3895	46.927711	132.799182

In [11]:

```
import plotly.express as px
px.scatter(batter,x='avg',y='strike_rate',hover_data=batter.columns).update_la
```

- **Tip**: The `s` value is proportional to the area of the points. Larger `s` means bigger points.In a `scatter` plot, the `s` parameter in `matplotlib.pyplot.scatter()` controls the **size** of the points. Here's a breakdown:

- `s` **Parameter**:

    - Represents the size of the points in the scatter plot.
    - **Default** size is 20.
    - You can specify:
        - A **single value** for uniform size for all points.
            - Example: `plt.scatter(x, y,`

`s=50)` will make all points size 50.

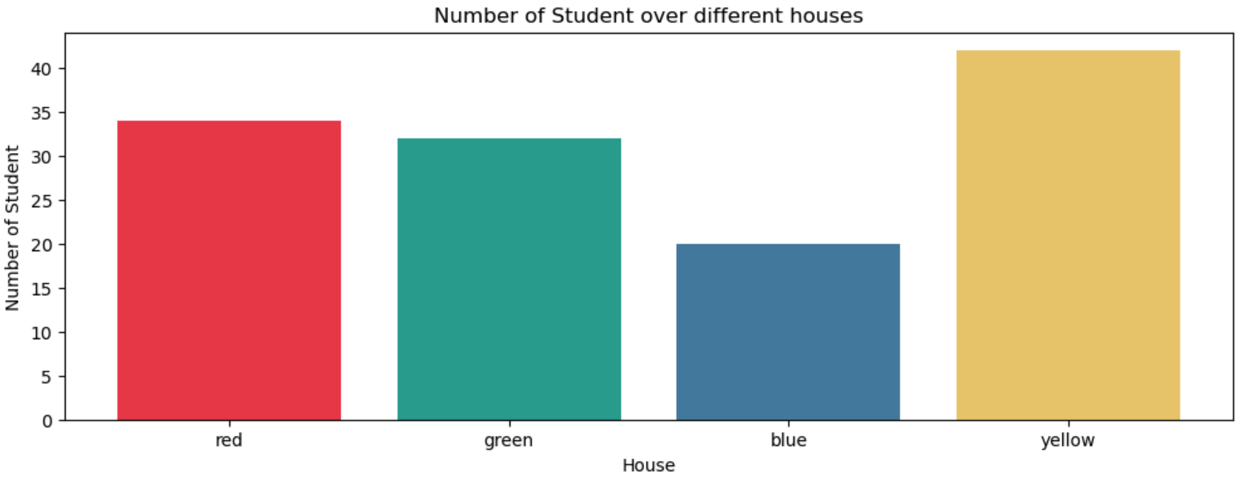
- A **list or array** of values to set different sizes for each point.
    - Example: `plt.scatter(x, y, s=sizes)` where `sizes` is a list of values for each point.

3. **Bar Chart (Vertical)**:
    - **Definition**: A vertical bar chart represents categorical data with rectangular bars. Each bar’s height corresponds to the category’s value.
    - **When to Use**: Good for comparing the values of different categories.
    - **Example**: Sales by product, population by country.

In [13]:

```
#bivarite analysis
#categorial vs numerical

house=['red','green','blue','yellow']
std=[34,32,20,42]
colors=["#E63946","#2A9D8F", "#457B9D","#E9C46A"]
plt.figure(figsize=(12,4))
plt.bar(house,std,color=colors)
plt.xlabel('House')
plt.ylabel('Number of Student')
plt.title('Number of Student over different houses')
plt.show()
```



Adds labels on top of each bar in a bar plot.

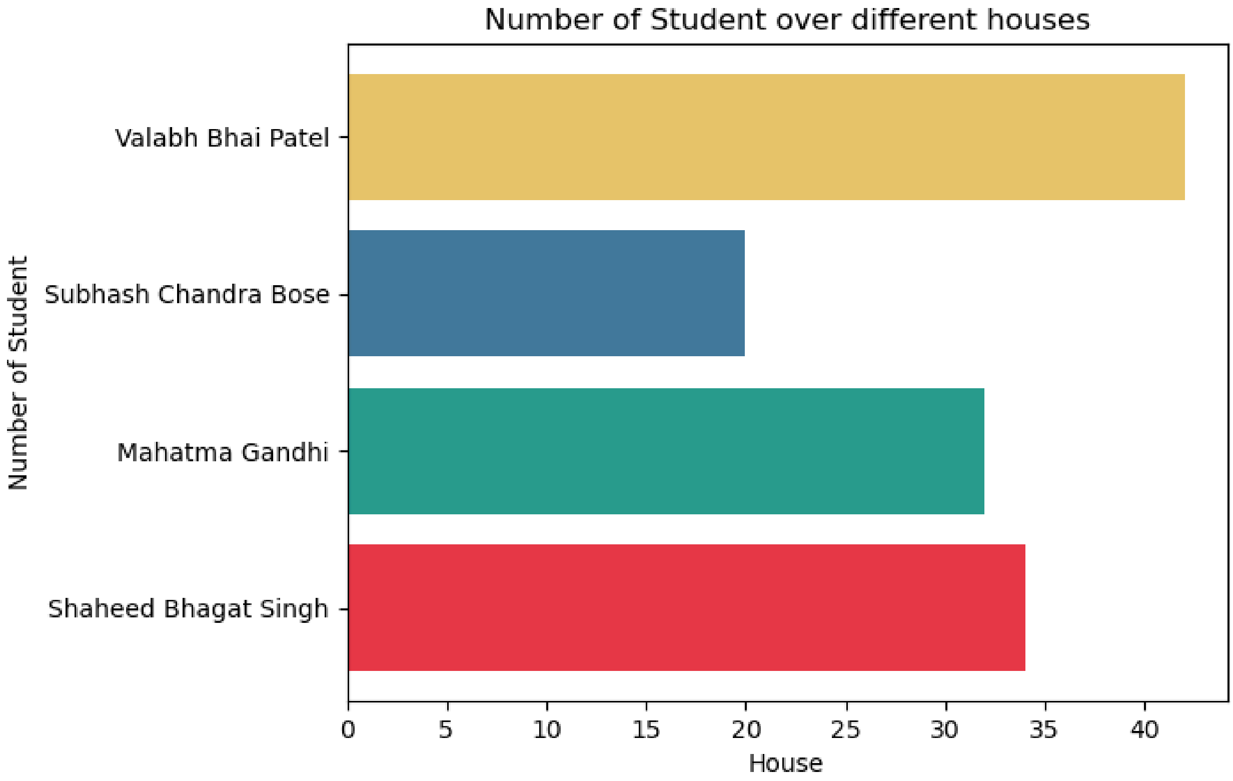
Parameters:

- bars: The bars in the bar chart (as returned by plt.bar()).
- offset: Distance above the bar to place the label (default is 1).

4. **Bar Chart (Horizontal)**:
    - **Definition**: Similar to a vertical bar chart, but the bars are horizontal. The length of the bar corresponds to the category’s value.
    - **When to Use**: Best when category names are long or you have many categories.
    - **Example**: Revenue by department, test scores by subject.

In [18]:

```python
house=['Shaheed Bhagat Singh','Mahatma Gandhi','Subhash Chandra Bose','Valabh
std=[34,32,20,42]
colors=["#E63946","#2A9D8F", "#457B9D","#E9C46A"]
#plt.figure(figsize=(12,4))
plt.barh(house,std,color=colors)
plt.xlabel('House')
plt.ylabel('Number of Student')
plt.title('Number of Student over different houses')
plt.show()
```



5. **Stacked Bar Chart**:
    - **Definition**: A bar chart where each bar is divided into sub-bars representing different categories. The total height represents the sum, and segments show the breakdown.
    - **When to Use**: Ideal for showing the composition of different categories within a total.
    - **Example**: Sales by product, broken down by region.

In [19]:

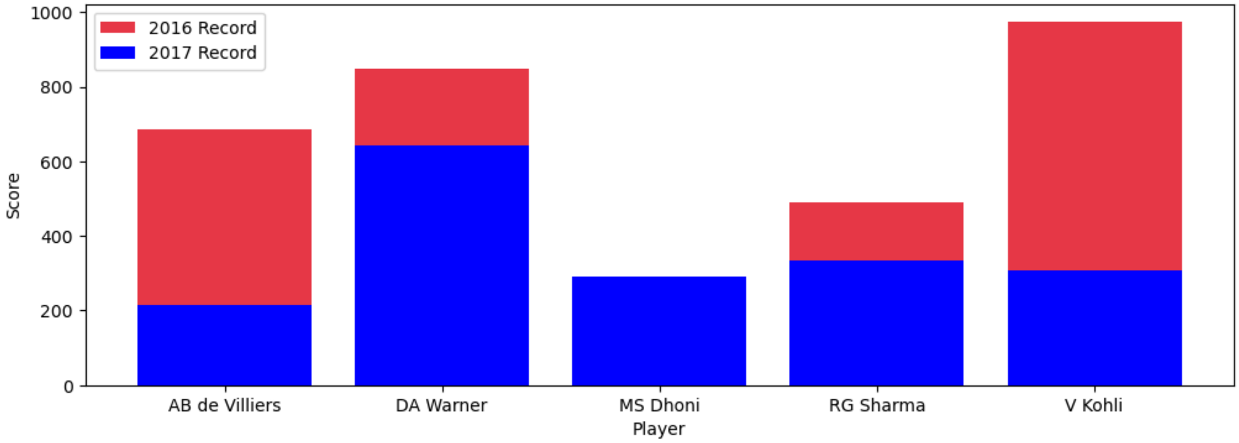
```
#batsman season record
season=pd.read_csv('batsman_season_record.csv')
season
```

Out[19]:

	batsman	2015	2016	2017
0	AB de Villiers	513	687	216
1	DA Warner	562	848	641
2	MS Dhoni	372	284	290
3	RG Sharma	482	489	333
4	V Kohli	505	973	308

In [20]:

```
plt.figure(figsize=(12,4))
plt.bar(season.batsman,season['2016'],label='2016 Record',color="#E63946")
plt.bar(season.batsman,season['2017'],label='2017 Record',color="Blue")#width=
plt.xlabel('Player')
plt.ylabel('Score')
#plt.grid()
plt.legend()
plt.show()
```



6. **Histogram**:
    - **Definition**: A histogram groups data into continuous intervals (called bins) and shows the frequency of data points within each interval.
    - **When to Use**: Great for understanding the distribution of a single variable.
    - **Example**: Distribution of test scores, age distribution in a population.

In [21]:

```
batter
```

Out[21]:

	batter	runs	avg	strike_rate
0	V Kohli	6634	36.251366	125.977972
1	S Dhawan	6244	34.882682	122.840842
2	DA Warner	5883	41.429577	136.401577
3	RG Sharma	5881	30.314433	126.964594
4	SK Raina	5536	32.374269	132.535312
...	...	...	...	...
195	SP Fleming	196	21.777778	114.619883
196	JC Archer	195	15.000000	152.343750
197	AS Raut	194	21.555556	112.790698
198	BJ Rohrer	193	27.571429	132.191781
199	Salman Butt	193	27.571429	112.209302

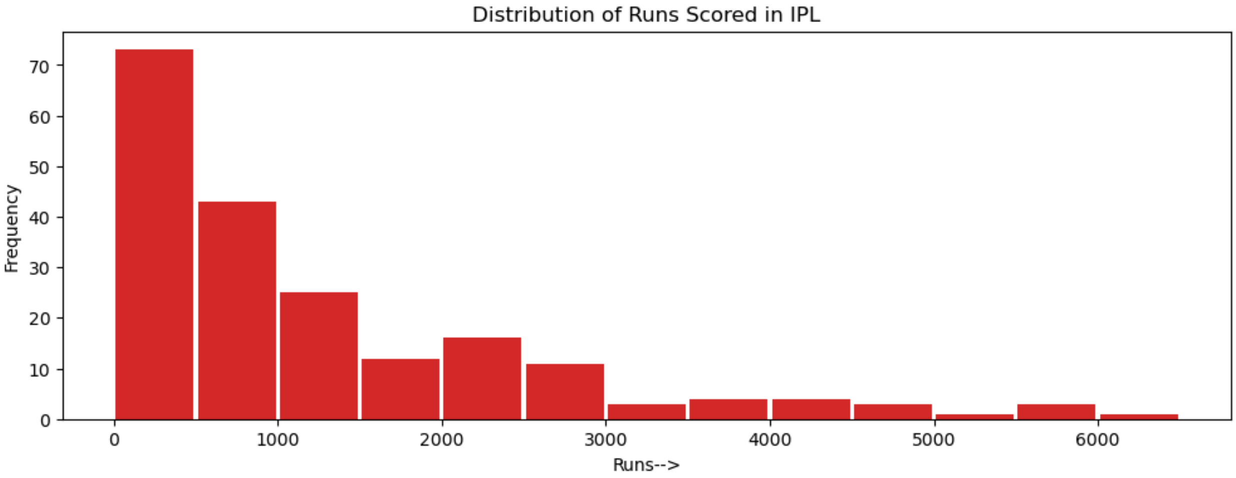
200 rows × 4 columns

In [22]:

```
#hisograms
#univariate analysis
#distribution,bimodel,gaussian curve,skewed,uniformly distibuted
#extreme number points
```

In [23]:

```
batter.runs.head(30)
plt.figure(figsize=(12,4))
plt.hist(batter.runs,bins=range(0,6701,500),rwidth=0.95,color="#D62828")
plt.xlabel('Runs-->')
plt.ylabel('Frequency')
plt.title('Distribution of Runs Scored in IPL')
plt.show()
```



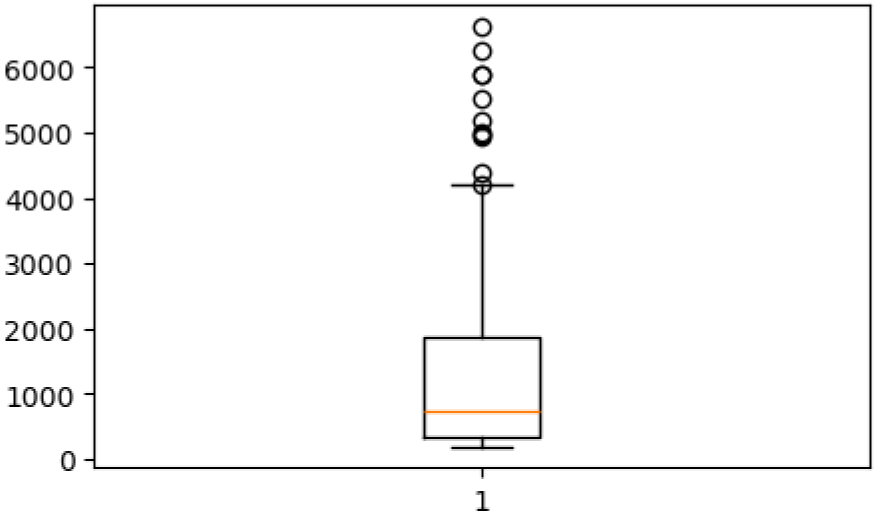
In [25]:
```
batter.runs.skew()

#-ve --> positive skewed
#~0 --> symmetric --> gaussian curve
#-ve--> negatively skewed -->
#+1--> hardly skewed
#0.05--> thoda bht skewed
```

Out[25]:
```
np.float64(1.7625305390047046)
```

7. **Box Plot**:
    - **Definition**: A box plot (or box-and-whisker plot) displays the distribution of a dataset, highlighting the median, quartiles, and outliers.
    - **When to Use**: Useful for comparing distributions between groups or showing data spread.
    - **Example**: Comparison of salaries across departments, exam scores.

In [26]:
```
# boxplot
# univarite
# outlier,distribution
plt.figure(figsize=(5,3))
plt.boxplot(batter.runs)
plt.show()
```



In [28]:
```
q3=batter.runs.quantile(0.75)
q1=batter.runs.quantile(0.25)
iqr=q3-q1
print('iqr=\t\t',iqr)
uf=q3+1.5*iqr
lf=q1-1.5*iqr
print('upper fence\t',uf)
print('lower fense\t',lf)
```

```
iqr=                     1549.0
upper fence            4201.0
lower fense            -1995.0
```

In [29]:

```
batter[batter.runs<4201.0]
```

Out[29]:

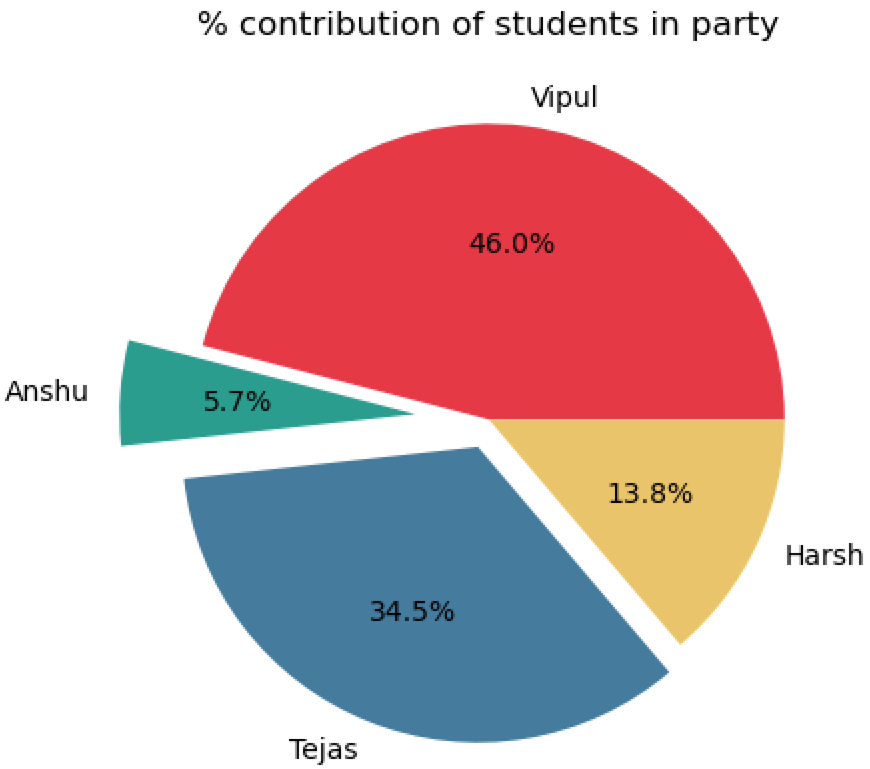
	batter	runs	avg	strike_rate
11	AT Rayudu	4190	28.896552	124.148148
12	AM Rahane	4074	30.863636	117.575758
13	KL Rahul	3895	46.927711	132.799182
14	SR Watson	3880	30.793651	134.163209
15	MK Pandey	3657	29.731707	117.739858
...	...	...	...	...
195	SP Fleming	196	21.777778	114.619883
196	JC Archer	195	15.000000	152.343750
197	AS Raut	194	21.555556	112.790698
198	BJ Rohrer	193	27.571429	132.191781
199	Salman Butt	193	27.571429	112.209302

189 rows × 4 columns

8. **Pie Chart**:
   - **Definition**: A circular chart divided into slices, with each slice representing a proportion of the whole.
   - **When to Use**: Best for showing parts of a whole as percentages.
   - **Example**: Market share by company, budget breakdown by category.

In [30]:

```
name=['Vipul','Anshu','Tejas','Harsh']
money=[400,50,300,120]
colors=["#E63946","#2A9D8F", "#457B9D","#E9C46A"]
plt.pie(money,labels=name,autopct='%1.1f%%',colors=colors,startangle=0,explode
plt.title('% contribution of students in party')
plt.show()
```



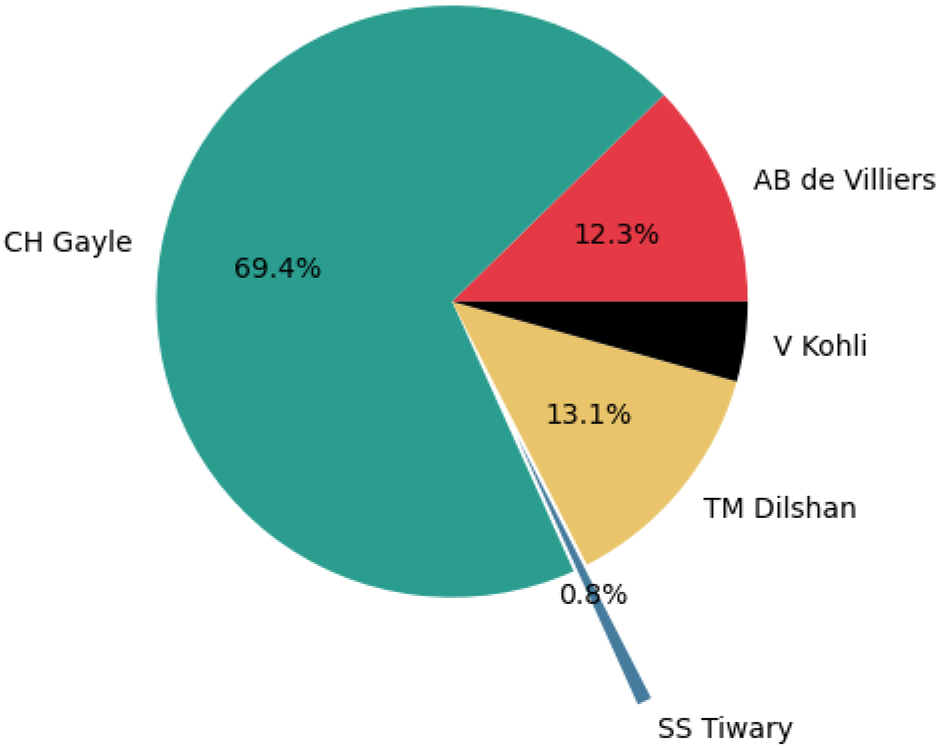
% contribution of students in party

```
In [32]: rcb=pd.read_csv('RCB.csv')
         rcb
```

Out[32]:

	batsman	batsman_runs
0	AB de Villiers	31
1	CH Gayle	175
2	R Rampaul	0
3	SS Tiwary	2
4	TM Dilshan	33
5	V Kohli	11

```
In [34]: rcb=pd.read_csv('RCB.csv').drop(2)
         colors=["#E63946","#2A9D8F", "#457B9D","#E9C46A","k"]
         plt.pie(rcb.batsman_runs,labels=rcb.batsman,autopct='%1.1f%%',
                 colors=colors,explode=[0,0,0.50,0,0])
         plt.show()
```



This Python function `func(pct, allvalues)` is designed to be used with visualizing data, such as creating pie charts. Here's how it works:

- **Inputs**:
    - `pct`: The percentage value (float) of a particular section in a pie chart or similar visualization.
    - `allvalues`: A list or array containing all the values that make up the full dataset.
- **Functionality**:
    - The function calculates the absolute value corresponding to the percentage `pct` out of the total sum of `allvalues`.
    - It computes this using the formula `int(pct / 100. * sum(allvalues))`.
    - The result is returned as a formatted string that shows both the absolute value and the percentage with one decimal precision, in the form `absolute (percentage%)`.
- **Use Case**:
    - This function is typically passed as the `autopct` argument in a plotting library like `matplotlib.pyplot.pie()`. It

helps to display both the percentage and the actual value inside each section of the pie chart.

In [ ]:

```
```