Matplotlib

`matplotlib.pyplot` is a module in Python used to create visualizations, like charts and graphs. It makes it easy to create a wide range of plots and is part of the `matplotlib` library.

Features of `matplotlib.pyplot`:

1. **Simple Plotting**: You can create simple plots like line graphs, bar charts, histograms, and scatter plots with just a few lines of code.
2. **Customization**: You can change colors, labels, and sizes to make your plots look exactly how you want. You can also add titles, legends, and axes labels easily.
3. **Multiple Plots**: It allows you to create multiple plots on the same figure or create subplots (multiple smaller plots) in one window.
4. **Integration with Other Libraries**: It works well with other Python libraries like NumPy and pandas, which makes it easy to plot data directly from data analysis results.

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 [1]:
```
# importing major libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# additional Libraries

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

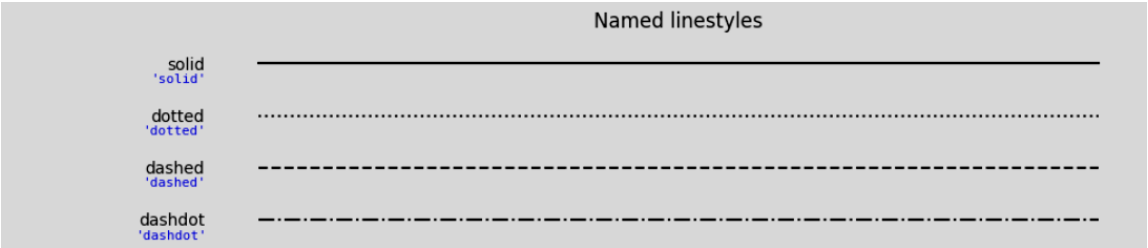
In [2]:
```
# Stock price Over the Years
year = [2021,2022,2023,2024,2025,2026,2027,2028]
price = [134,241,420,256,345,456,432,234]
```

`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]:
```
plt.plot(year,price,linestyle='solid',marker='o')
plt.title('Stock Price over the Years')
plt.xlabel('Years')
plt.ylabel('Stock Price')
plt.show()
```



Here are the notes on markers in Matplotlib:

- **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.
```

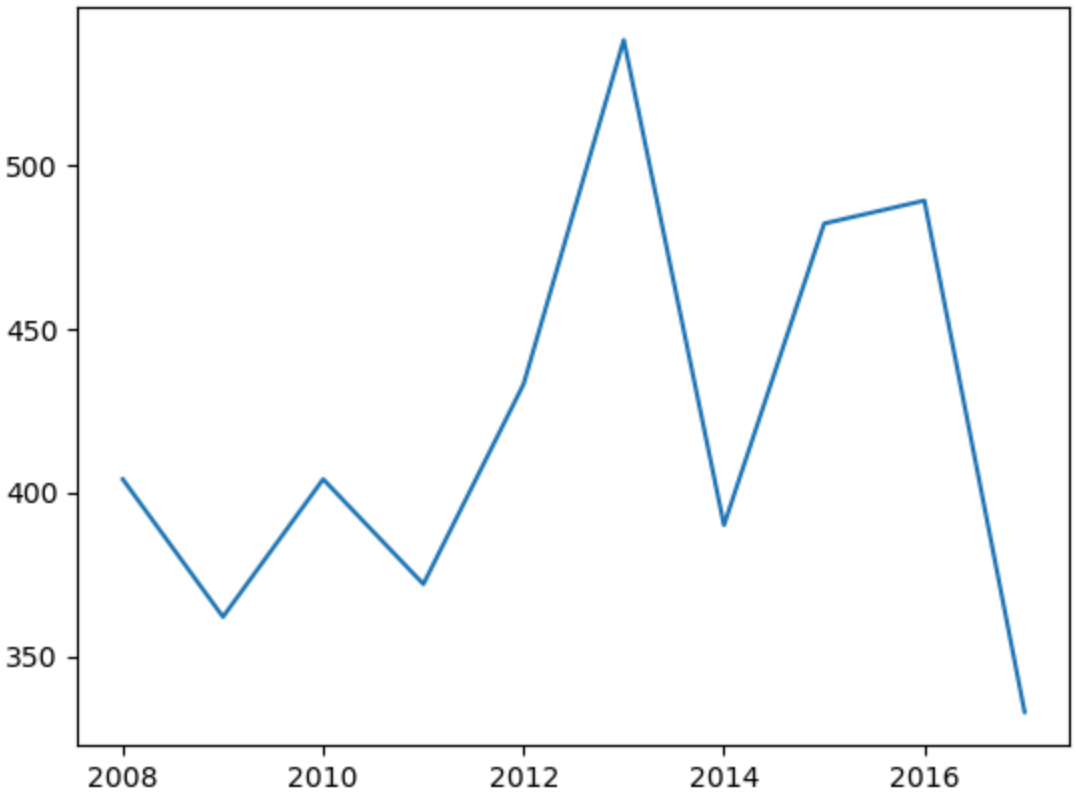
In [4]:
```
df = pd.read_csv('sharma-kohli.csv')
```

In [5]:
```
df
```

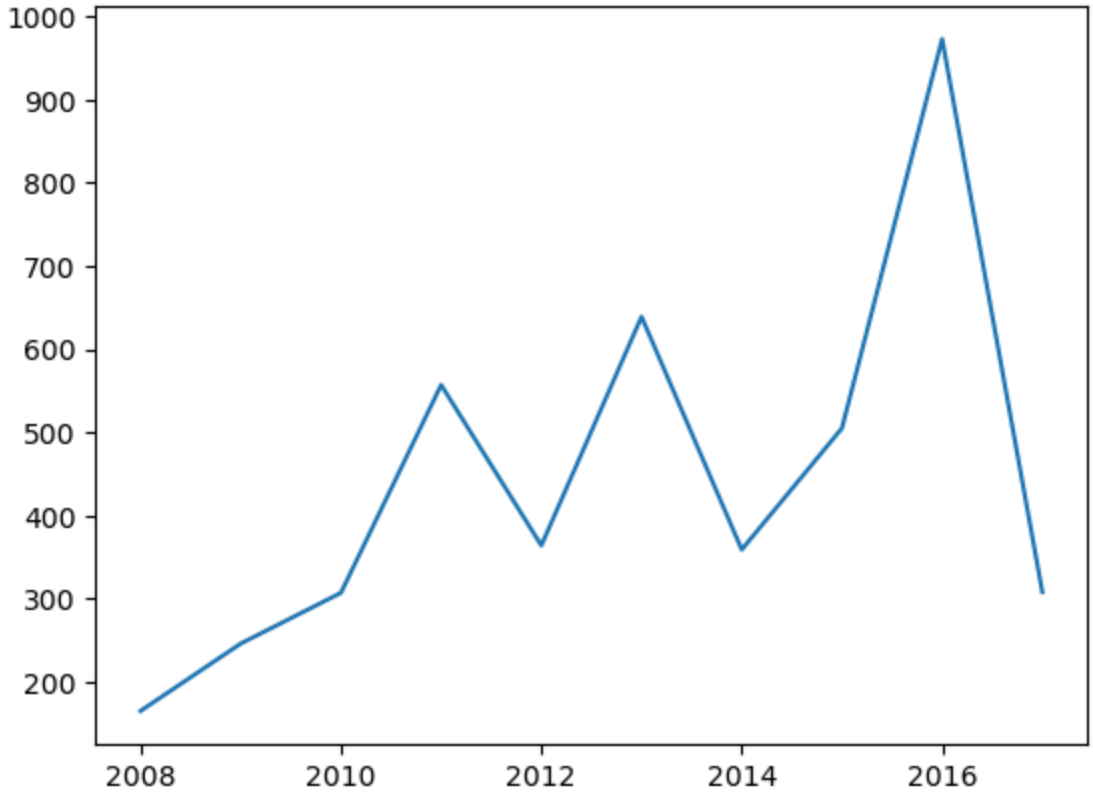
Out[5]:

	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

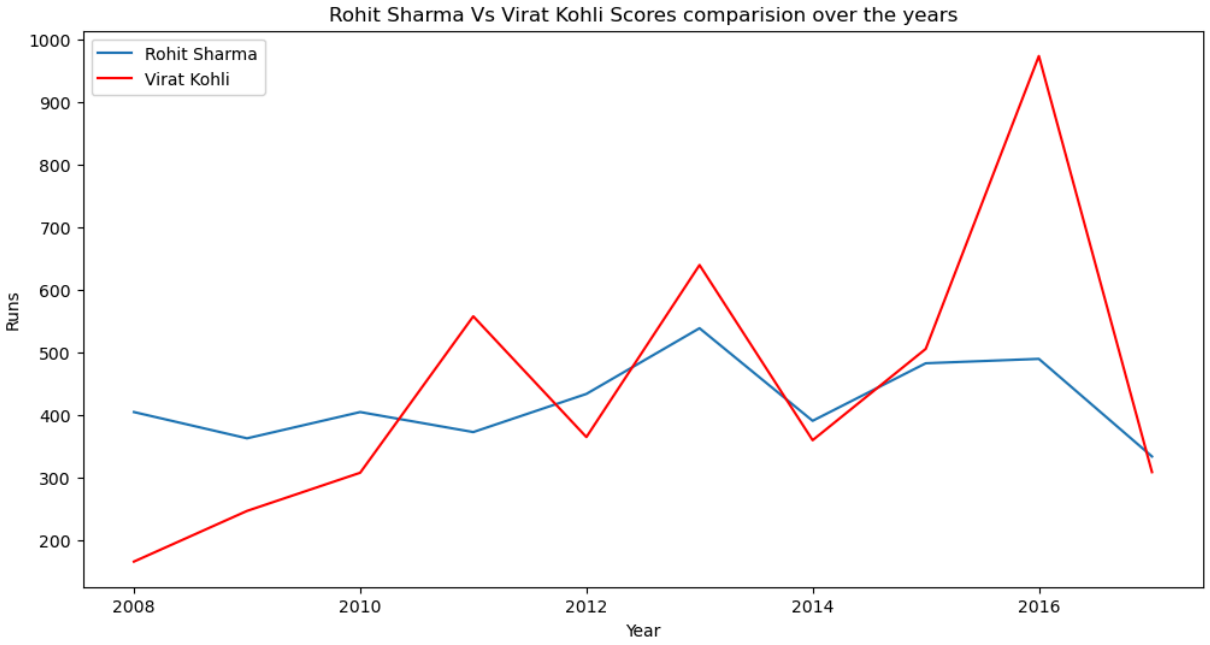
In [6]:
```
# Rohit Sharma's Career in IPL seasons over the years
plt.plot(df['index'],df['RG Sharma'])
plt.show()
```



In [7]:
```
# Virat kohli's Career in IPL seasons over the years
plt.plot(df['index'],df['V Kohli'])
plt.show()
```



In [8]:
```
plt.figure(figsize=(12,6)) # for increasing or decreasing size of graph

plt.plot(df['index'],df['RG Sharma'],label='Rohit Sharma') #(x-axis,y-axis)
plt.plot(df['index'],df['V Kohli'],label='Virat Kohli',color='r') # Color, l
plt.legend(loc=0) # legend
plt.title('Rohit Sharma Vs Virat Kohli Scores comparision over the years') #
plt.xlabel('Year') # Xlabel
plt.ylabel('Runs') # Ylabel
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 [9]:
```
df = pd.read_csv('batter.csv')
```

In [10]:
```
df
```

Out[10]:

	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
...	...	...	...	...
600	C Nanda	0	0.000000	0.000000
601	Akash Deep	0	0.000000	0.000000
602	S Ladda	0	0.000000	0.000000
603	V Pratap Singh	0	0.000000	0.000000
604	S Lamichhane	0	0.000000	0.000000

605 rows × 4 columns

In [11]:
```
# top 50 batsman
temp = df.head(50)
```

In [12]:
```
temp.shape
```

Out[12]:
```
(50, 4)
```

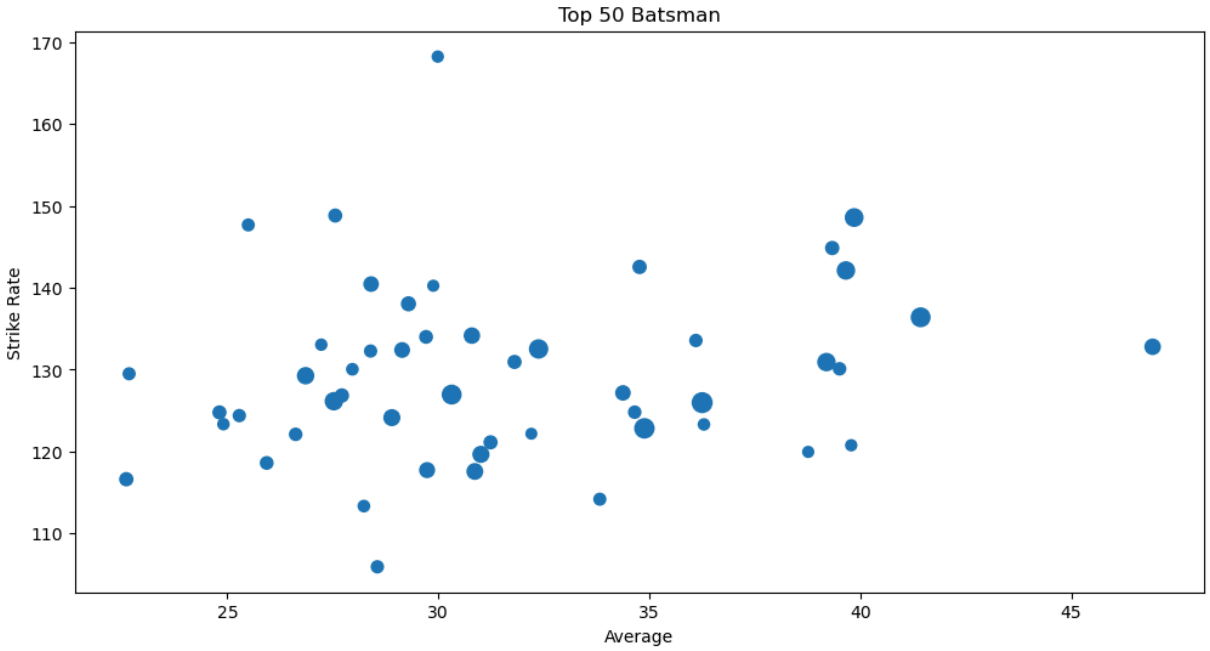
In [13]:
```
temp.head()
```

Out[13]:

	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

In [14]:
```
plt.figure(figsize=(12,6))
# plt.scatter(temp.avg,temp.strike_rate)
```

```
plt.scatter(temp.avg,temp.strike_rate,s=temp.runs*0.02) # Bubbles
plt.title('Top 50 Batsman')
plt.xlabel('Average')
plt.ylabel('Strike Rate')
plt.show()
```



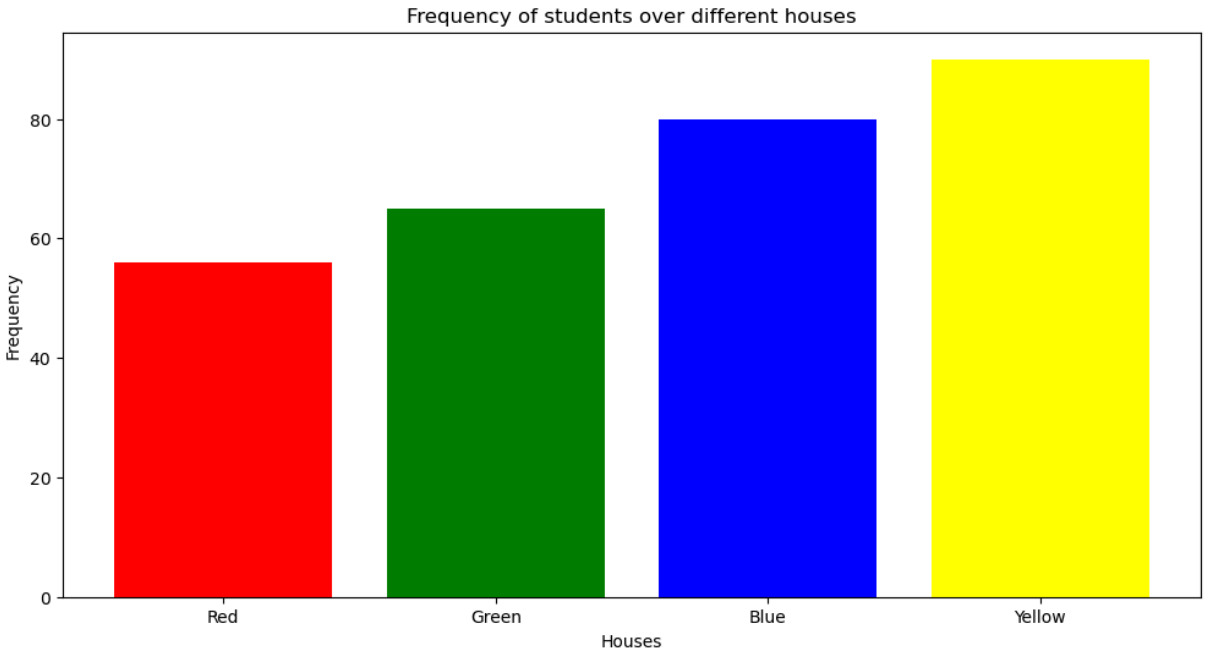
In a `scatter` plot, the `s` parameter in `matplotlib.pyplot.scatter()` controls the **size** of the points. Here's a breakdown:

- `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.
- **Tip**: The `s` value is proportional to the area of the points. Larger `s` means bigger points.

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 [15]:
```
std = [56,65,80,90]
houses = ['Red','Green','Blue','Yellow']
```

In [16]:
```
plt.figure(figsize=(12,6))
plt.bar(houses,std,color=['Red','Green','Blue','Yellow'])
plt.title('Frequency of students over different houses')
plt.xlabel('Houses')
plt.ylabel('Frequency')
plt.show()
```



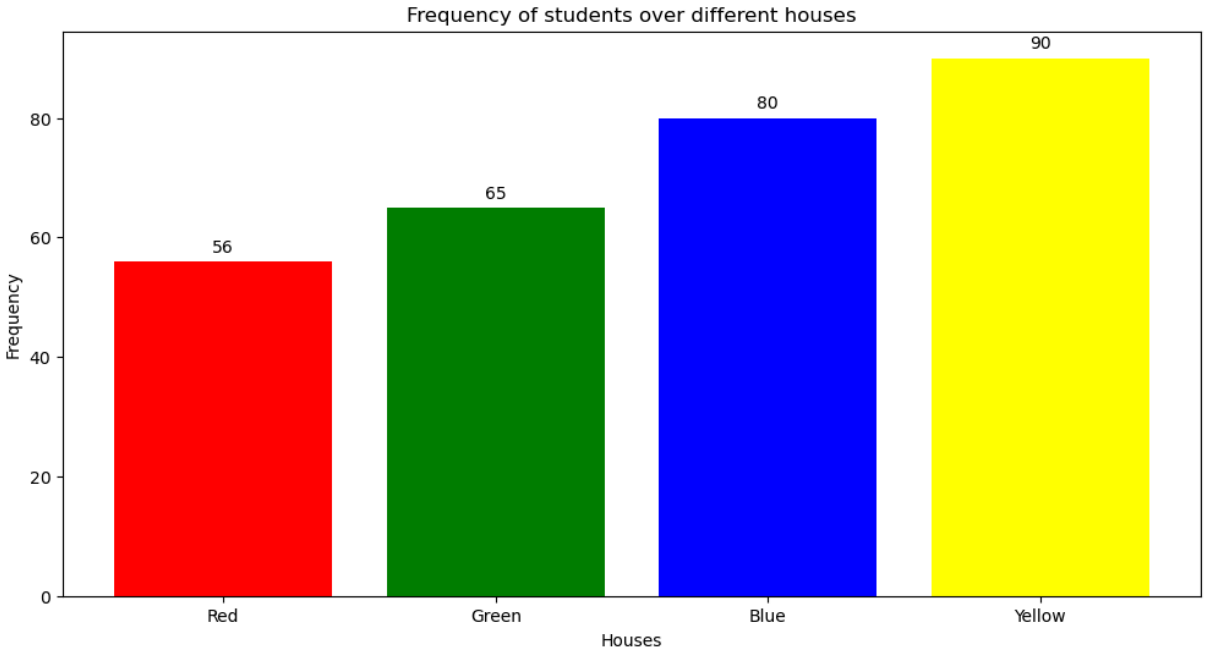
Adds labels on top of each bar in a bar plot.

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).

In [17]:
```
plt.figure(figsize=(12,6))

# Snippit
def add_bar_labels(bars, offset=1):
    for bar in bars:
        yval = bar.get_height()  # Get the height of the bar
        plt.text(bar.get_x() + bar.get_width()/2, yval + offset, int(yval),
                 ha='center', va='bottom')
plt.bar(houses,std,color=['Red','Green','Blue','Yellow'])
# Add labels to the bars
bars = plt.bar(houses, std, color=['Red', 'Green', 'Blue', 'Yellow'])

add_bar_labels(bars)
```

```
# Snippit

plt.title('Frequency of students over different houses')
plt.xlabel('Houses')
plt.ylabel('Frequency')
plt.show()
```



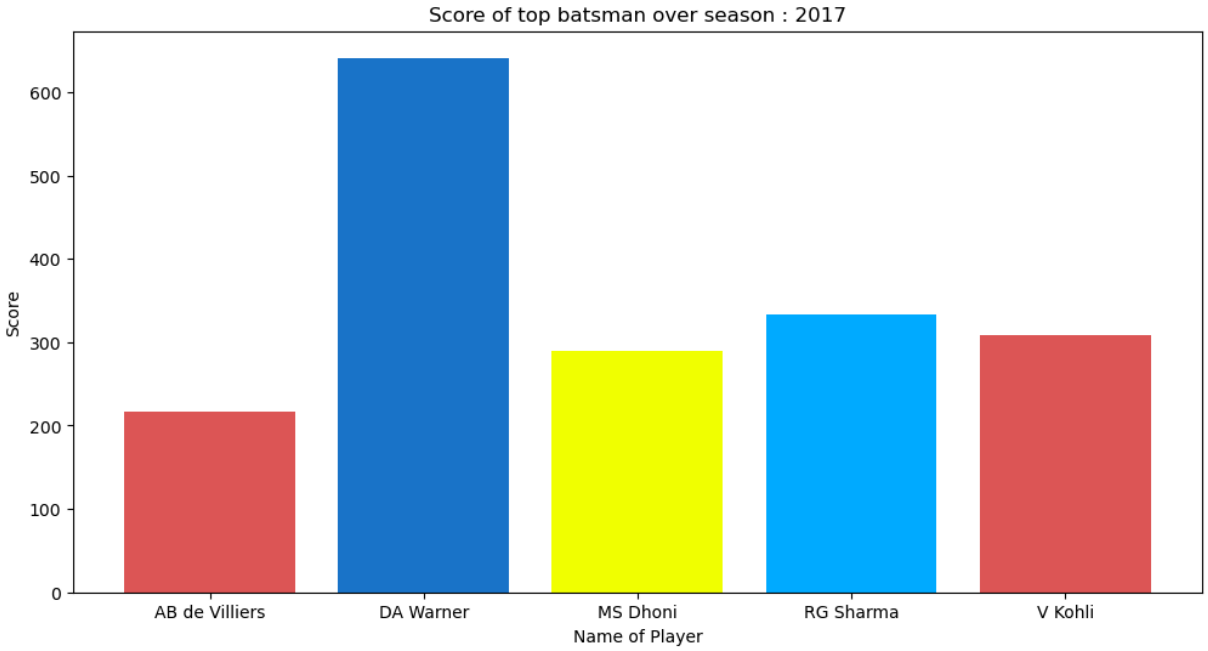
In [18]:
```
df = pd.read_csv('batsman_season_record.csv')
```

In [19]:
```
df
```

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

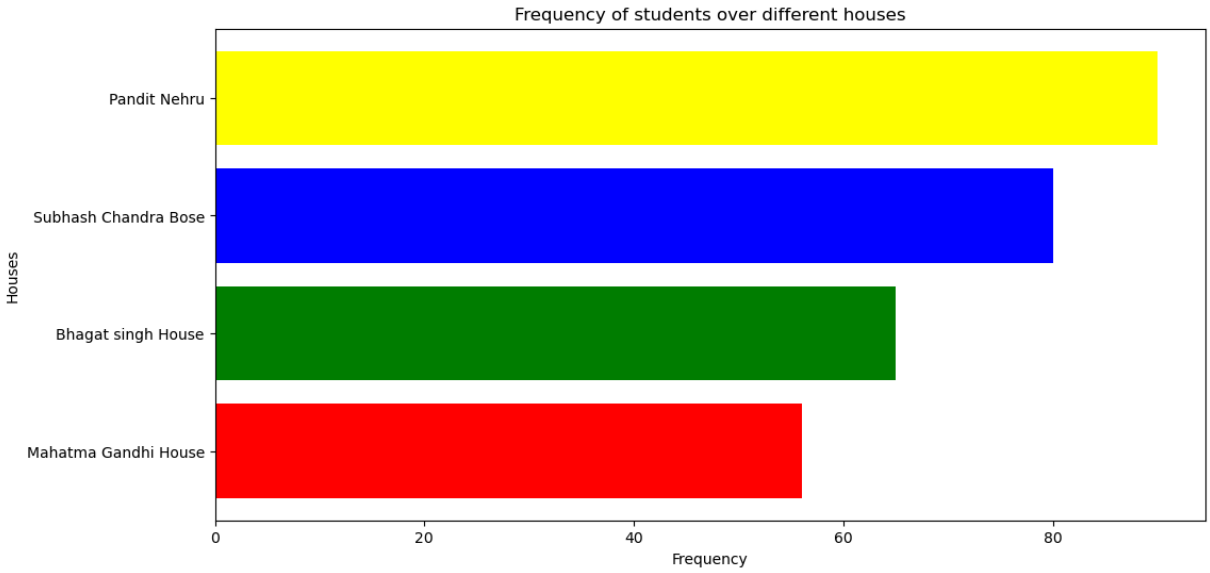
In [20]:
```
plt.figure(figsize=(12,6))
plt.bar(df.batsman,df['2017'],color=['#dd5755','#1b76cc','#f3ff00','#00aeff'
plt.title('Score of top batsman over season : 2017')
plt.xlabel('Name of Player')
plt.ylabel('Score')
plt.show()
```



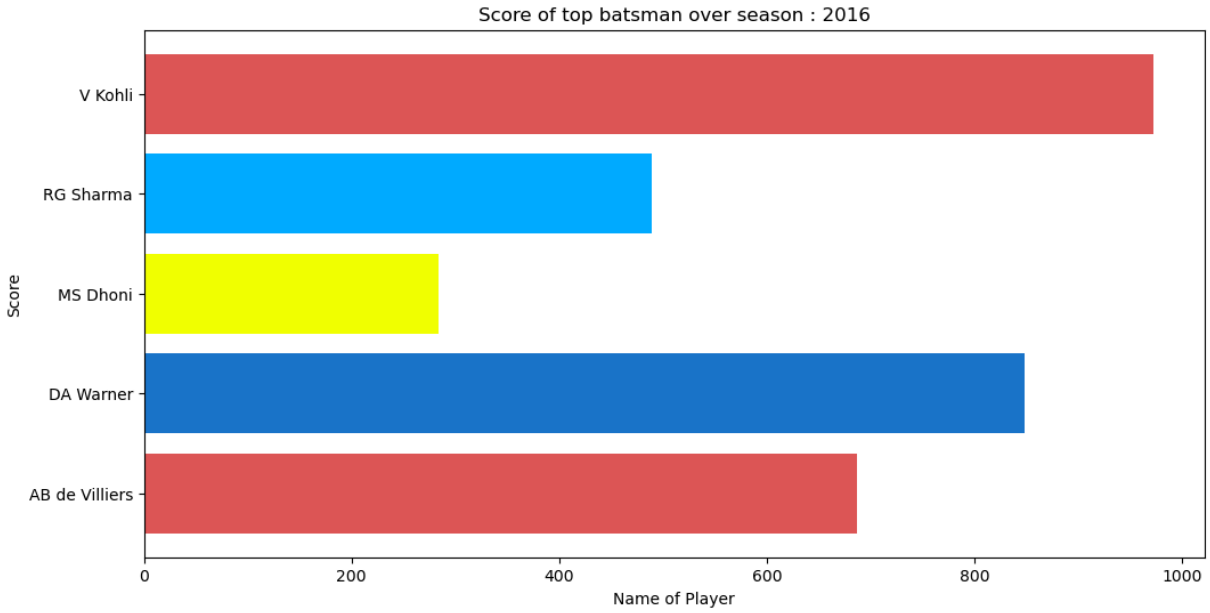
4. **Bar Chart (Horizontal)**:
   - **Definition**: Similar to a vertical bar chart, but the bars are horizontal. The length of the bar corresponds to the category's value.
   - **When to Use**: Best when category names are long or you have many categories.
   - **Example**: Revenue by department, test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

In [21]:
```
std = [56,65,80,90]
houses = ['Mahatma Gandhi House','Bhagat singh House','Subhash Chandra Bose'
```

In [22]:
```
plt.figure(figsize=(12,6))
plt.barh(houses,std,color=['Red','Green','Blue','Yellow'])
plt.title('Frequency of students over different houses')
plt.ylabel('Houses')
plt.xlabel('Frequency')
plt.show()
```

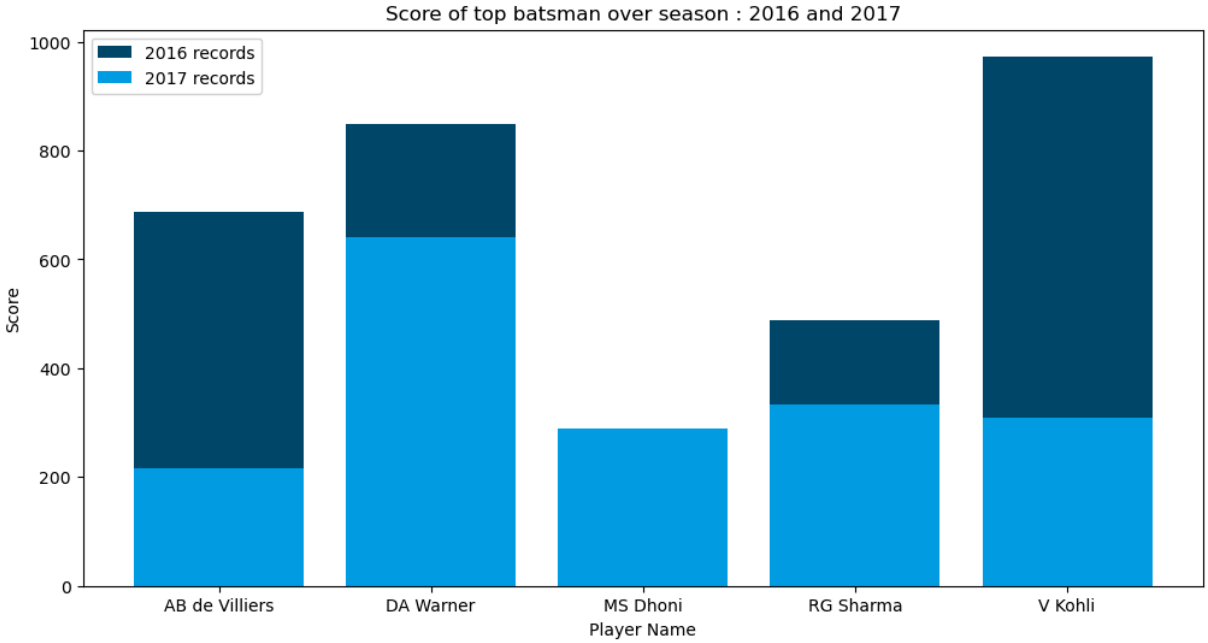


In [23]:
```
plt.figure(figsize=(12,6))
plt.barh(df.batsman,df['2016'],color=['#dd5755','#1b76cc','#f3ff00','#00aeff
plt.title('Score of top batsman over season : 2016')
plt.xlabel('Name of Player')
plt.ylabel('Score')
plt.show()
```



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 [24]:
```
plt.figure(figsize=(12,6))
plt.bar(df.batsman,df['2016'],color='#044a6a',label='2016 records')
plt.bar(df.batsman,df['2017'],color='#049ee5',label='2017 records') #width=0
plt.title('Score of top batsman over season : 2016 and 2017')
plt.xlabel('Player Name')
plt.ylabel('Score')
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.

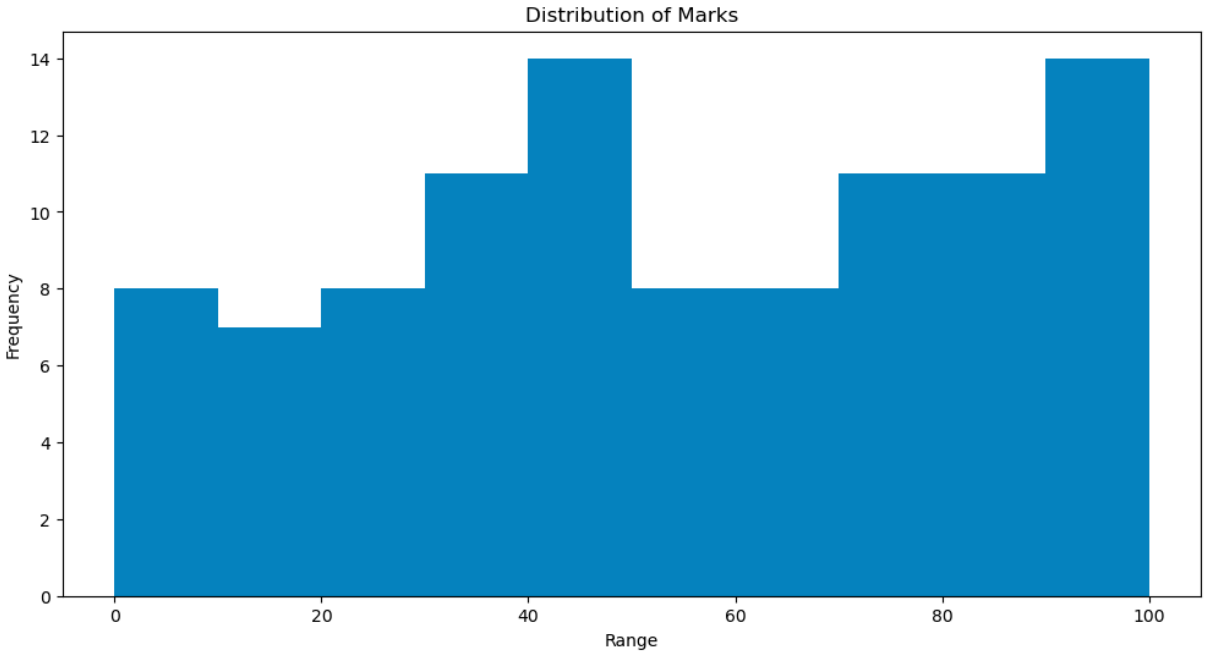
In [25]:
```
marks = np.random.randint(0,101,100)
```

In [26]:
```
marks
```

Out[26]:
```
array([ 40,  53,  63,  46,  22,  79,   7,  85,  78,  80,  35,  33,  18,
        97,   1,  99,  99,  69,  53,  21,  32,  20,  93,  34,  18, 100,
        23,  42,  36,  44,   1,  37,   5,  16, 100,  76,  27,  64, 100,
        43,  30,  33,  59,  20,  46,  56,  41,  87,  89,  94,  72,  75,
        82,  79,  65,  40,  97,  90,  61,  83,   6,   3,  49,  57,  41,
        89,  42,  10,  81,  31,  44,  99,  88,   0,  98,  76,  50,  45,
        36,  74,  47,  88,  82,  20,  60,  74,  75,  32,  57,  14,  67,
        64,  22,  95,  93,  12,   2,  57,  76,  13])
```

In [27]:
```
plt.figure(figsize=(12,6))
plt.hist(marks,bins=range(0,101,10),color='#0584be')
plt.xlabel('Range')
```

```
plt.ylabel('Frequency')
plt.title('Distribution of Marks')
plt.show()
```



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.

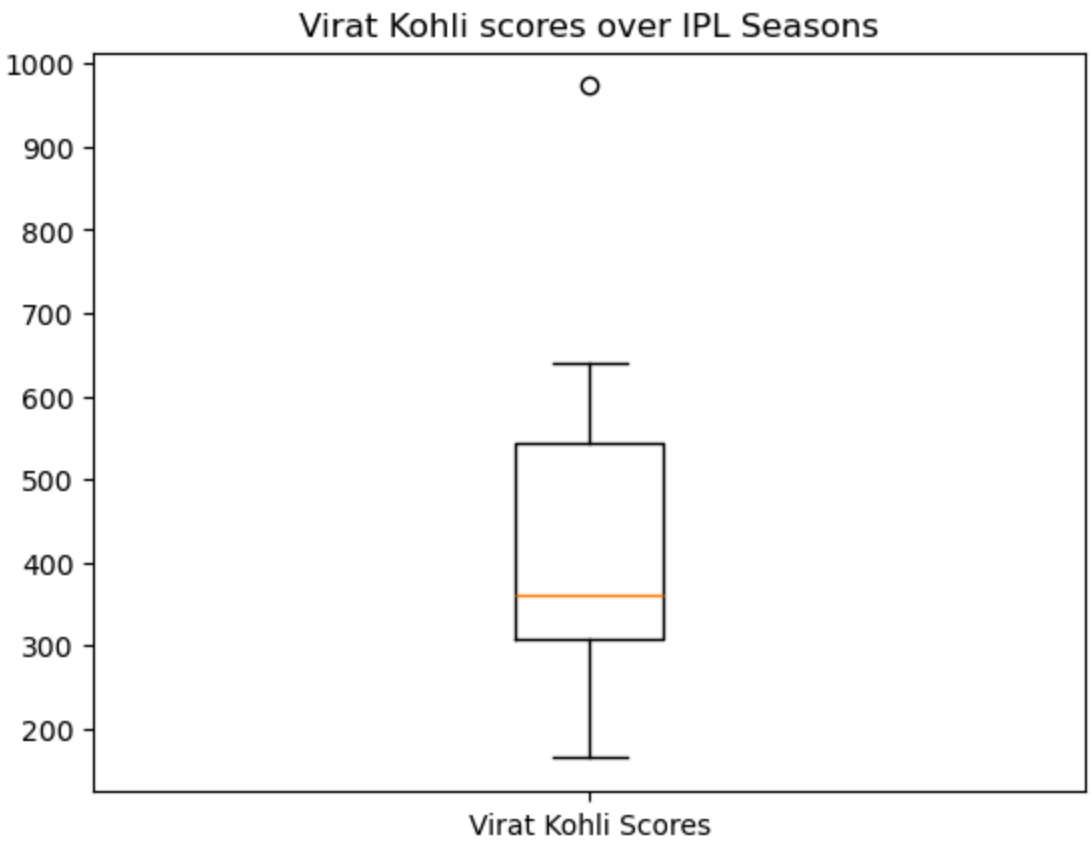
In [28]:
```
df = pd.read_csv('sharma-kohli.csv')
```

In [29]:
```
df
```

Out[29]:

	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 [30]:
```
plt.boxplot(df['V Kohli'],labels=['Virat Kohli Scores'])
plt.title('Virat Kohli scores over IPL Seasons')
plt.show()
```



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.

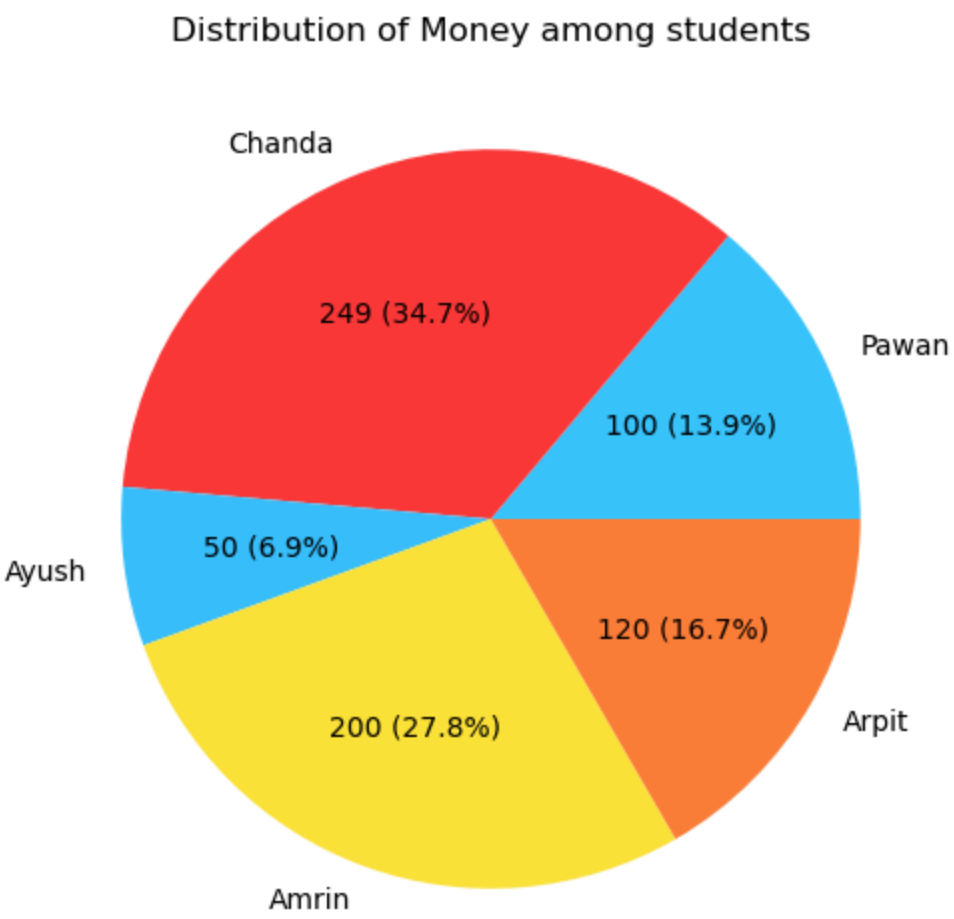
- **Example**: Market share by company, budget breakdown by category.

In [31]:
```
money = [100,250,50,200,120]
names = ['Pawan','Chanda','Ayush','Amrin','Arpit']

def func(pct,allvalues):
    absolute = int(pct/100.*sum(allvalues))
    return f'{absolute} ({pct:.1f}%)'
```

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 [32]:
```
plt.figure(figsize=(12,6))
plt.pie(money,labels=names,autopct=lambda pct: func(pct,money),colors=['#3ac
plt.title('Distribution of Money among students')
plt.show()

# autopct = '%1.1f%%'
```



The `autopct` parameter in `matplotlib.pyplot.pie()` is used to display the percentage value on the pie chart slices. It can take different types of inputs to customize how percentages are shown. Here's a breakdown:

1. `autopct` as `None` (default):

- If you don't provide `autopct`, the pie chart will not display any percentage labels on the slices.

2. `autopct` as a Format String:

- If you pass a format string (e.g., `'%1.1f%%'`), it will display the percentage values on the slices using that format.
- The format string must include a percentage symbol (`%%`), which ensures the number is formatted as a percentage.
- Example:

  ```
  plt.pie(data, autopct='%1.1f%%')
  ```
  This will display percentages with 1 decimal place (e.g., `45.7%`).

3. `autopct` as a Callable Function:

- You can pass a custom function (like the one in your original code) that formats the labels with more customization.
- The function should take two arguments:
  - `pct`: The percentage of the pie slice.
  - `allvalues`: The total values to calculate absolute numbers (if needed).
- Example:

  ```
  plt.pie(data, autopct=lambda pct: func(pct, data))
  ```
  This allows you to display both the percentage and the absolute values (e.g., `100 (33.3%)`).

4. Key Points for `autopct` Format String:

- `%1.0f%%`: Displays percentages with no decimal places (e.g., `33%`).
- `%1.1f%%`: Displays percentages with one decimal place (e.g., `33.3%`).
- `%1.2f%%`: Displays percentages with two decimal places (e.g., `33.33%`).

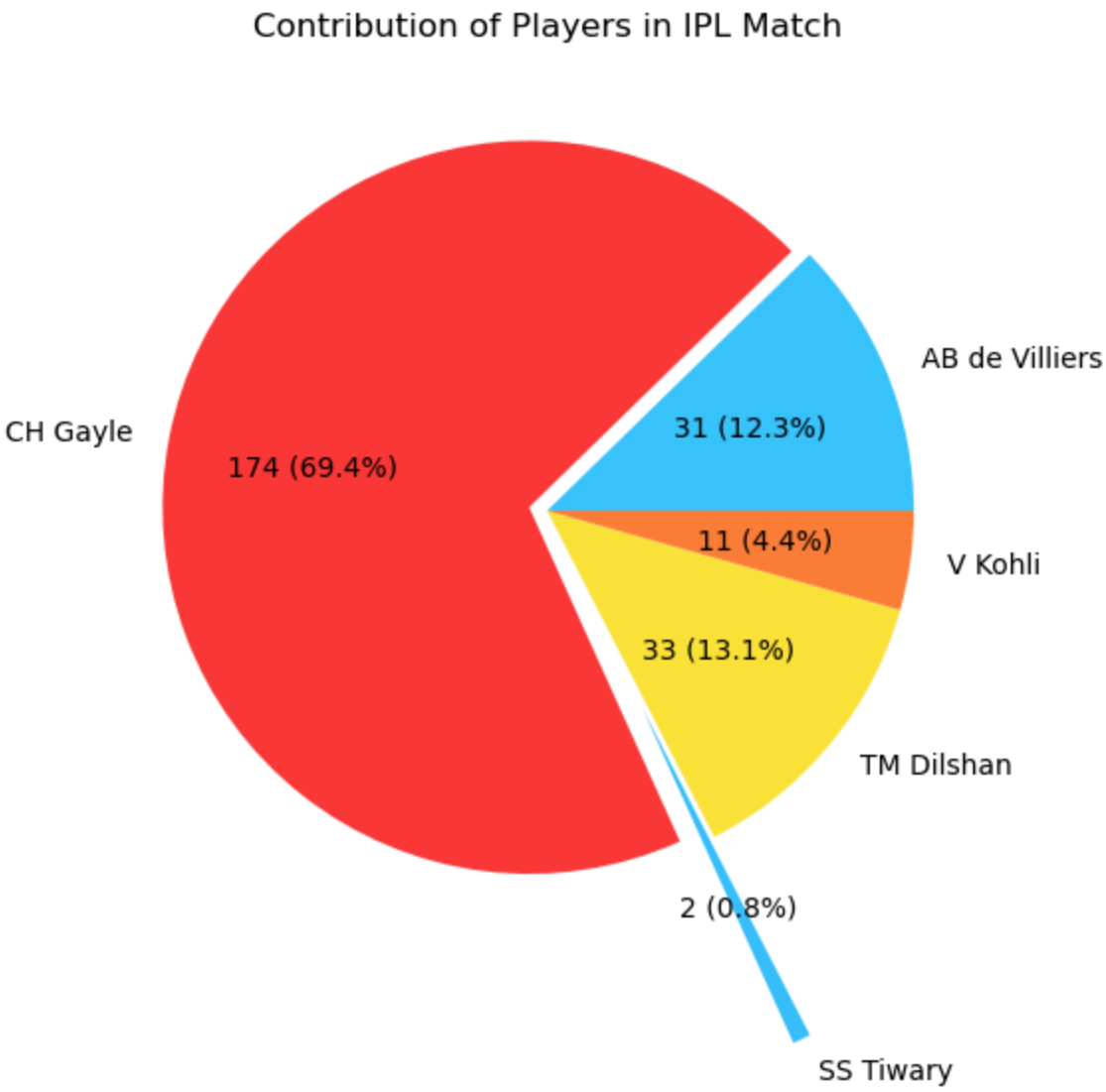
In [33]:
```
df = pd.read_csv('RCB.csv')
df
```

Out[33]:

	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]:
```
temp = df[df.batsman_runs!=0]
```

In [35]:
```
plt.figure(figsize=(12,6))
plt.pie(temp.batsman_runs,labels=temp.batsman,autopct=lambda pct: func(pct,t
        colors=['#3ac3fe','#fe3a3a','#3ac0fe','#fee33a','#fe7e3a'],startangl
        explode=[0,0.05,0.6,0,0]
       )
plt.title('Contribution of Players in IPL Match')
plt.show()
```



- **Matplotlib provides predefined themes to style your plots.**
- **Themes help customize colors, fonts, and layouts for consistent and attractive visualizations.**
- **You can switch themes easily using `plt.style.use()`.**

Predefined Themes in Matplotlib:

Matplotlib offers several built-in themes (also known as styles) to make your plots look visually appealing without manual adjustments. These themes control various aspects like colors, gridlines, fonts, and overall aesthetics.

Commonly Used Themes:

1. **`default`**: The standard Matplotlib style (since version 2.x).
2. **`ggplot`**: Inspired by the ggplot2 package in R, this theme uses a light background with gridlines.

3. `seaborn`: Comes from the Seaborn library, with improved aesthetics and color palettes.
4. `fivethirtyeight`: A style that mimics the visual style of the data journalism website FiveThirtyEight.
5. `bmh`: A minimalist theme used in Bayesian methods for hackers.
6. `dark_background`: A dark theme, useful for presentations or dark environments.
7. `grayscale`: Converts plots to a grayscale color scheme, great for print publications.
8. `Solarize_Light2`: A theme with a vintage color palette and softer tones.
9. `tableau-colorblind10`: Designed for colorblind-friendly plots, using Tableau's color scheme.

In [36]:
```
# Viewing Available Themes:
plt.style.available
```

Out[36]:
```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
 'seaborn-v0_8-dark',
 'seaborn-v0_8-dark-palette',
 'seaborn-v0_8-darkgrid',
 'seaborn-v0_8-deep',
 'seaborn-v0_8-muted',
 'seaborn-v0_8-notebook',
 'seaborn-v0_8-paper',
 'seaborn-v0_8-pastel',
 'seaborn-v0_8-poster',
 'seaborn-v0_8-talk',
 'seaborn-v0_8-ticks',
 'seaborn-v0_8-white',
 'seaborn-v0_8-whitegrid',
 'tableau-colorblind10']
```

In [37]:
```
# Changing Themes in a Plot
plt.style.use('seaborn-v0_8-darkgrid')
```

I'm **Anshum Banga**, a Data Scientist and Trainer with expertise in Python, machine learning, and data visualization. I specialize in Matplotlib, Power BI, and

Tableau, helping learners develop practical data skills. Connect with me on LinkedIn.