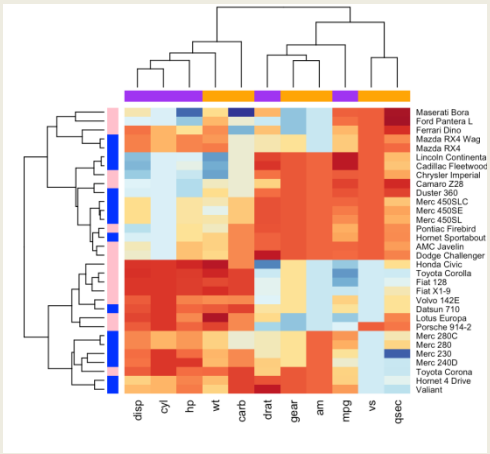
Machine Learning Part 03

1. 2D Line Plot in Matplotlib
2. ScatterPlot and Bar Chart in Matplotlib
3. Histogram and Pie chart in Matplotlib
4. Advanced Matplotlib (part-1)
5. Advanced Matplotlib (part-2)
6. Heatmap
7. Visualization with Pandas Plot Function
8. Seaborn
9. Distribution Plots in Seaborn
10. Matrix Plot

seaborn



matplotlib

2D Line Plot in Matplotlib

Firstly,

What is Matplotlib?

Matplotlib is a popular data visualization library in Python. It provides a wide range of tools for creating various types of plots and charts, making it a valuable tool for data analysis, scientific research, and data presentation. Matplotlib allows you to create high-quality, customizable plots and figures for a variety of purposes, including line plots, bar charts, scatter plots, histograms, and more.

Matplotlib is highly customizable and can be used to control almost every aspect of your plots, from the colors and styles to labels and legends. It provides both a functional and an object-oriented interface for creating plots, making it suitable for a wide range of users, from beginners to advanced data scientists and researchers.

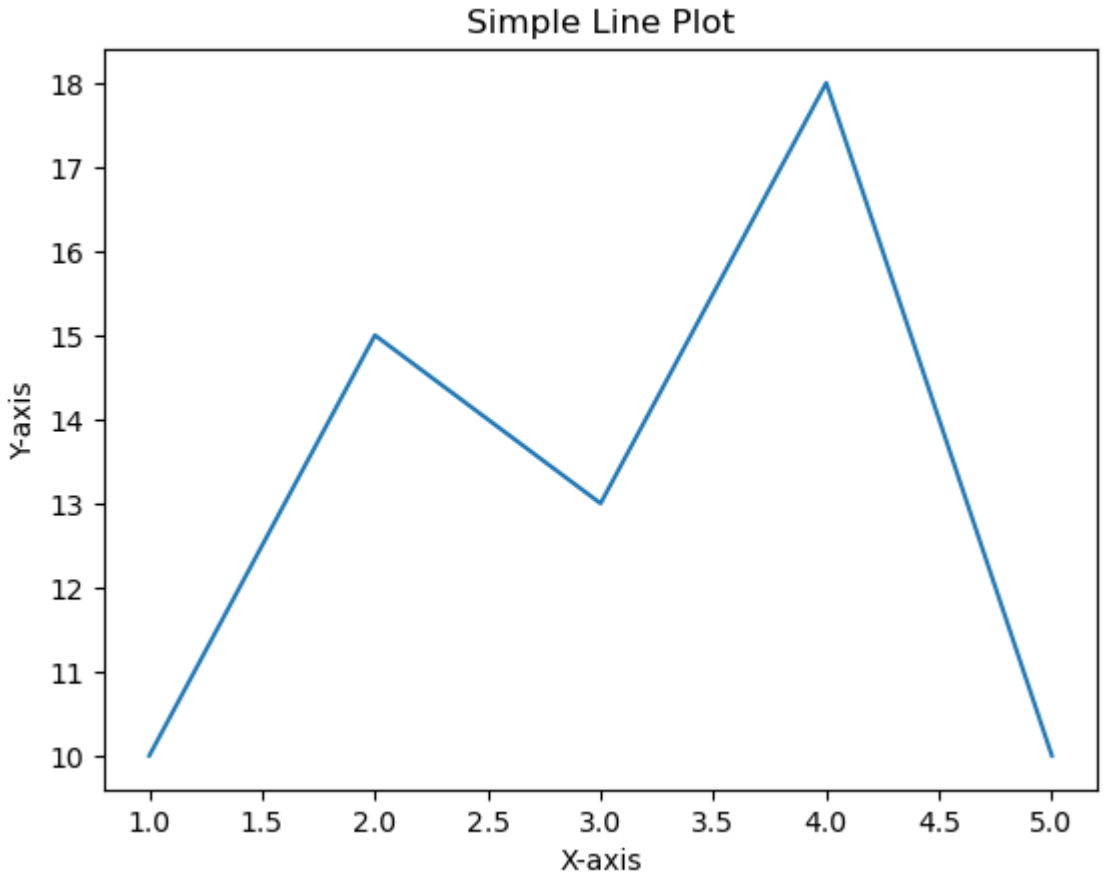
Matplotlib can be used in various contexts, including Jupyter notebooks, standalone Python scripts, and integration with web applications and GUI frameworks. It also works well with other Python libraries commonly used in data analysis and scientific computing, such as NumPy and Pandas.

To use Matplotlib, you typically need to import the library in your Python code, create the desired plot or chart, and then display or save it as needed. Here's a simple example of creating a basic line plot using Matplotlib:

In [ ]:
```
import numpy as np
import pandas as pd
```

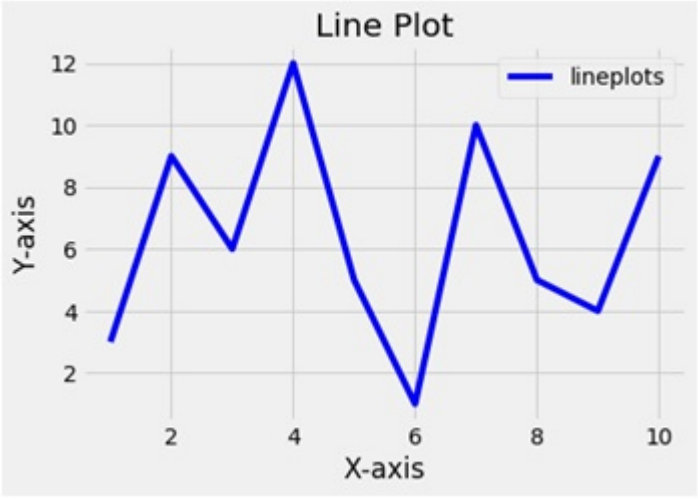
In [ ]:
```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [10, 15, 13, 18, 10]
```

```
# Create a line plot
plt.plot(x, y)

# Add labels and a title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')

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



This is just a basic introduction to Matplotlib. The library is quite versatile, and you can explore its documentation and tutorials to learn more about its capabilities and how to create various types of visualizations for your data.

Line Plot

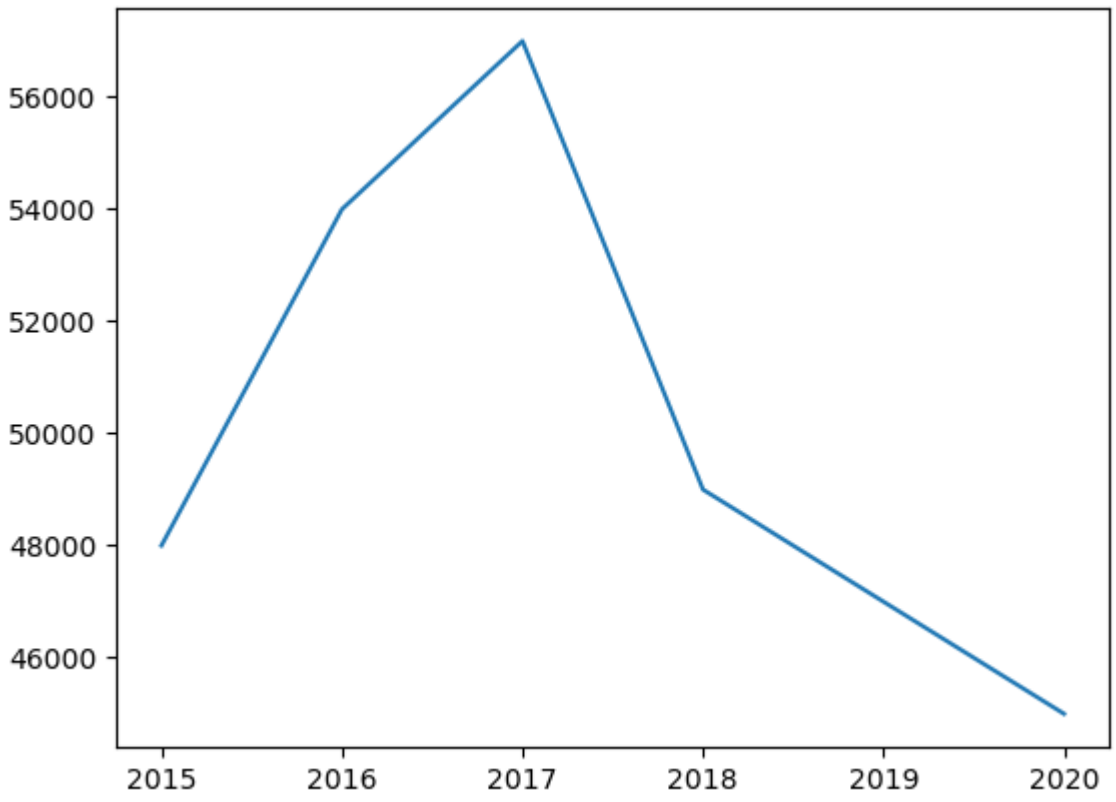


A 2D line plot is one of the most common types of plots in Matplotlib. It's used to visualize data with two continuous variables, typically representing one variable on the x-axis and another on the y-axis, and connecting the data points with lines. This type of plot is useful for showing trends, relationships, or patterns in data over a continuous range.

- Bivariate Analysis
- categorical -> numerical and numerical -> numerical
- Use case - Time series data

In [ ]:
```
# plotting simple graphs

price = [48000,54000,57000,49000,47000,45000]
year = [2015,2016,2017,2018,2019,2020]

plt.plot(year,price)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x28b7c3742e0>]
```



Real-world Dataset

In [ ]:
```
batsman = pd.read_csv('Data\Day45\sharma-kohli.csv')
```

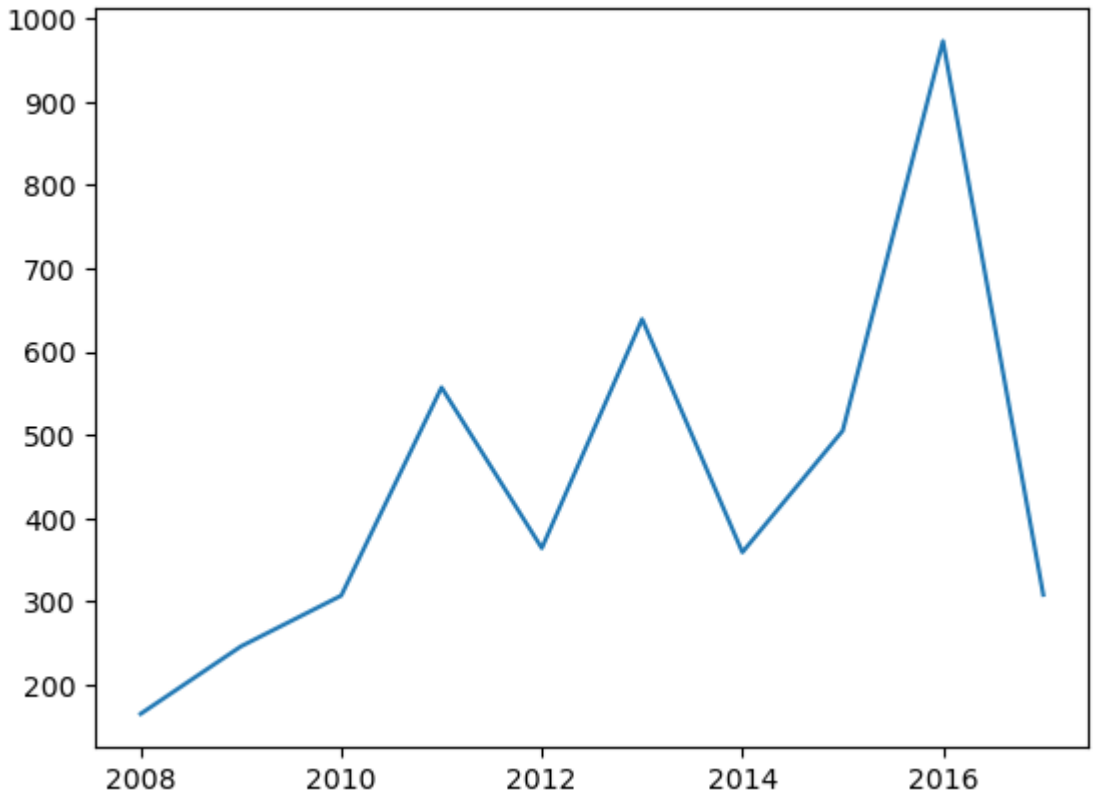
In [ ]:
```
batsman.head()
```

Out[ ]:

	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

In [ ]:
```
# plot the graph
plt.plot(batsman['index'],batsman['V Kohli'])
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x28b7c40ca60>]
```

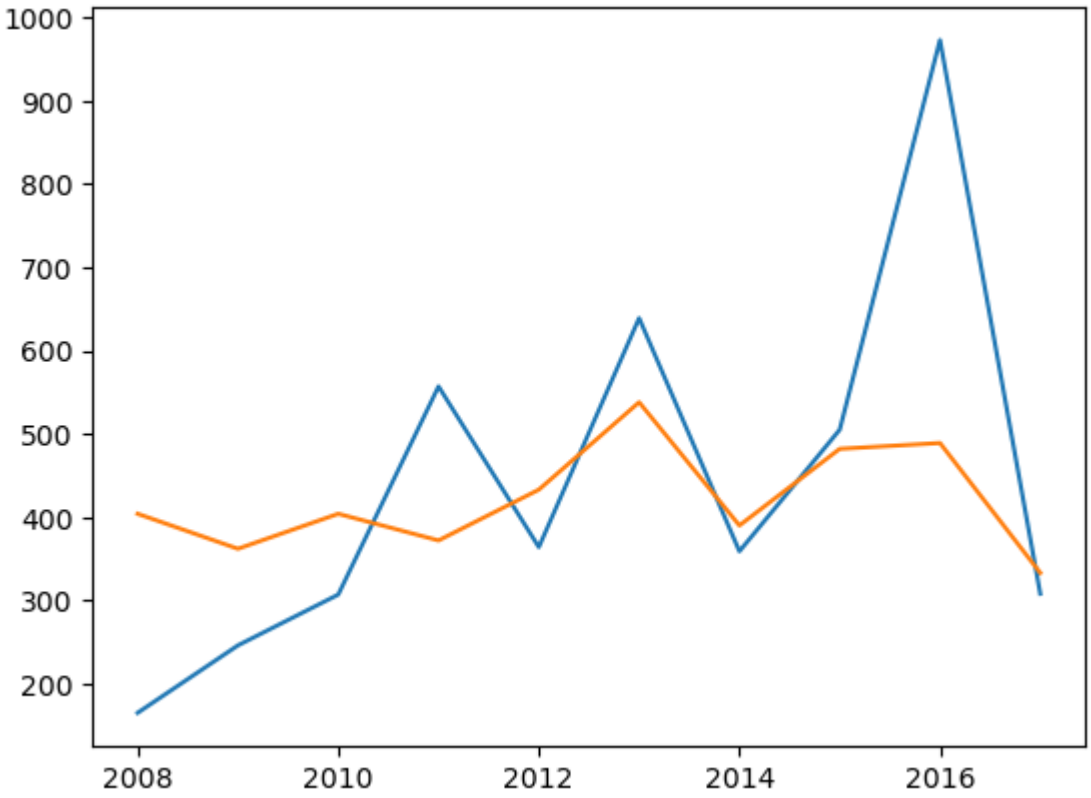


Multiple Plots:

It's possible to create multiple lines on a single plot, making it easy to compare multiple datasets or variables. In the example, both Rohit Sharma's and Virat Kohli's career runs are plotted on the same graph.

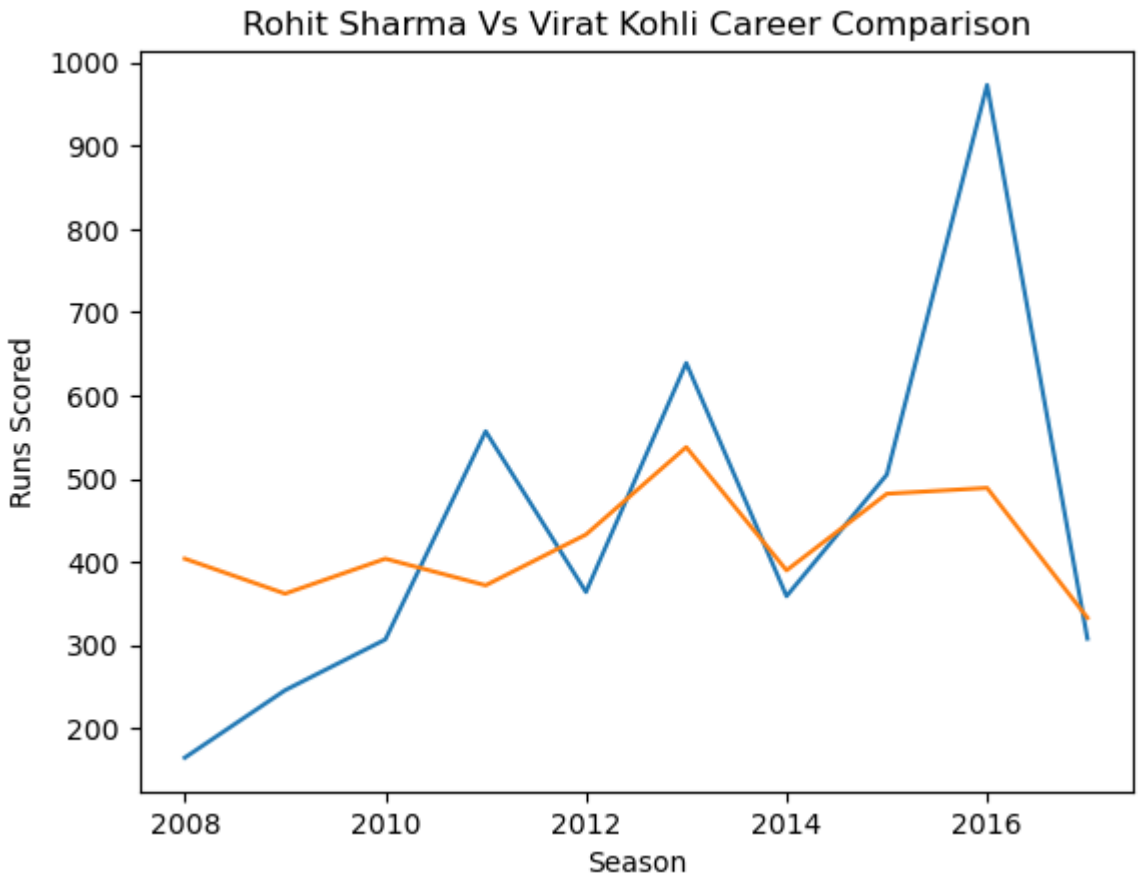
In [ ]:
```
# plotting multiple plots
plt.plot(batsman['index'],batsman['V Kohli'])
plt.plot(batsman['index'],batsman['RG Sharma'])
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x28b7d4e2610>]
```



In [ ]:
```
# Labels title
plt.plot(batsman['index'],batsman['V Kohli'])
plt.plot(batsman['index'],batsman['RG Sharma'])

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

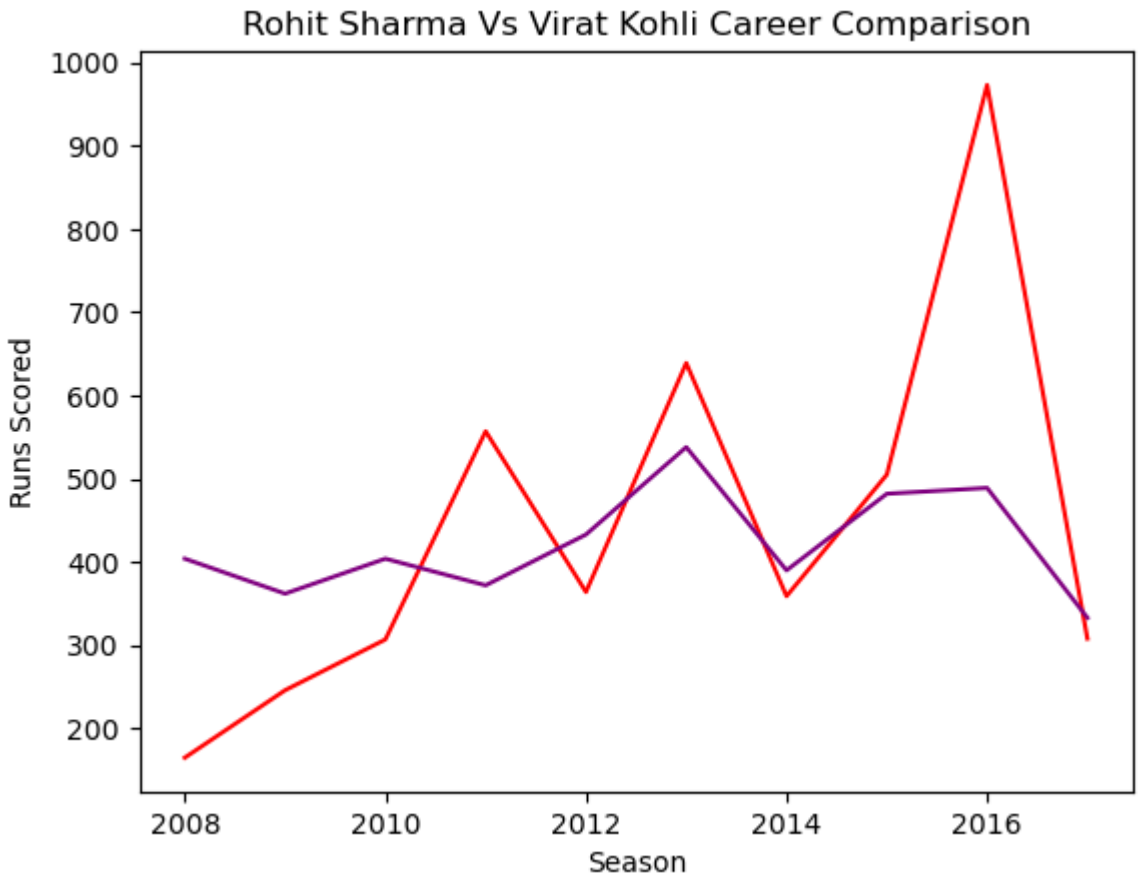
Out[ ]:
```
Text(0, 0.5, 'Runs Scored')
```



colors(hex) and line(width and style) and marker(size)

In [ ]:
```
#colors
plt.plot(batsman['index'],batsman['V Kohli'],color='Red')
plt.plot(batsman['index'],batsman['RG Sharma'],color='Purple')

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

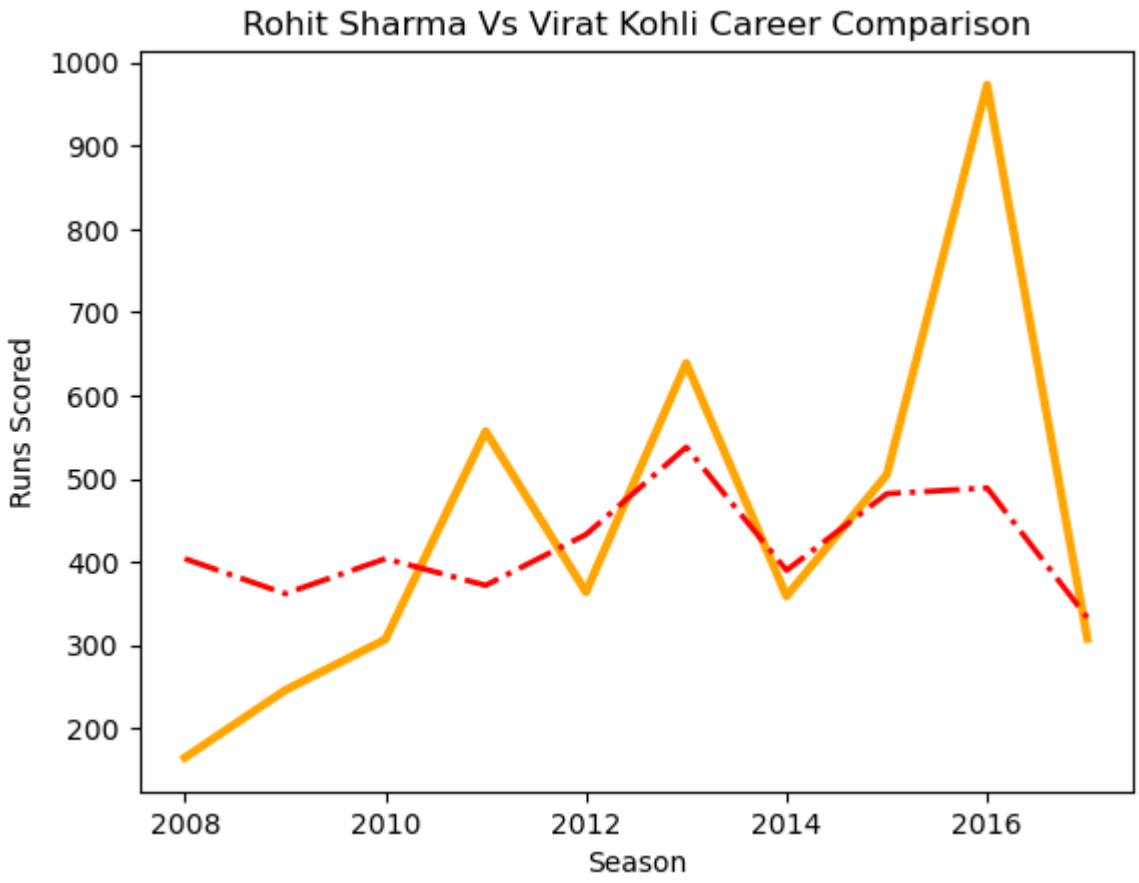
Out[ ]:
```
Text(0, 0.5, 'Runs Scored')
```



You can specify different colors for each line in the plot. In the example, colors like 'Red' and 'Purple' are used to differentiate the lines.

In [ ]:
```
# linestyle and linewidth
plt.plot(batsman['index'],batsman['V Kohli'],color='Orange',linestyle='solid',linew
plt.plot(batsman['index'],batsman['RG Sharma'],color='Red',linestyle='dashdot',line

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

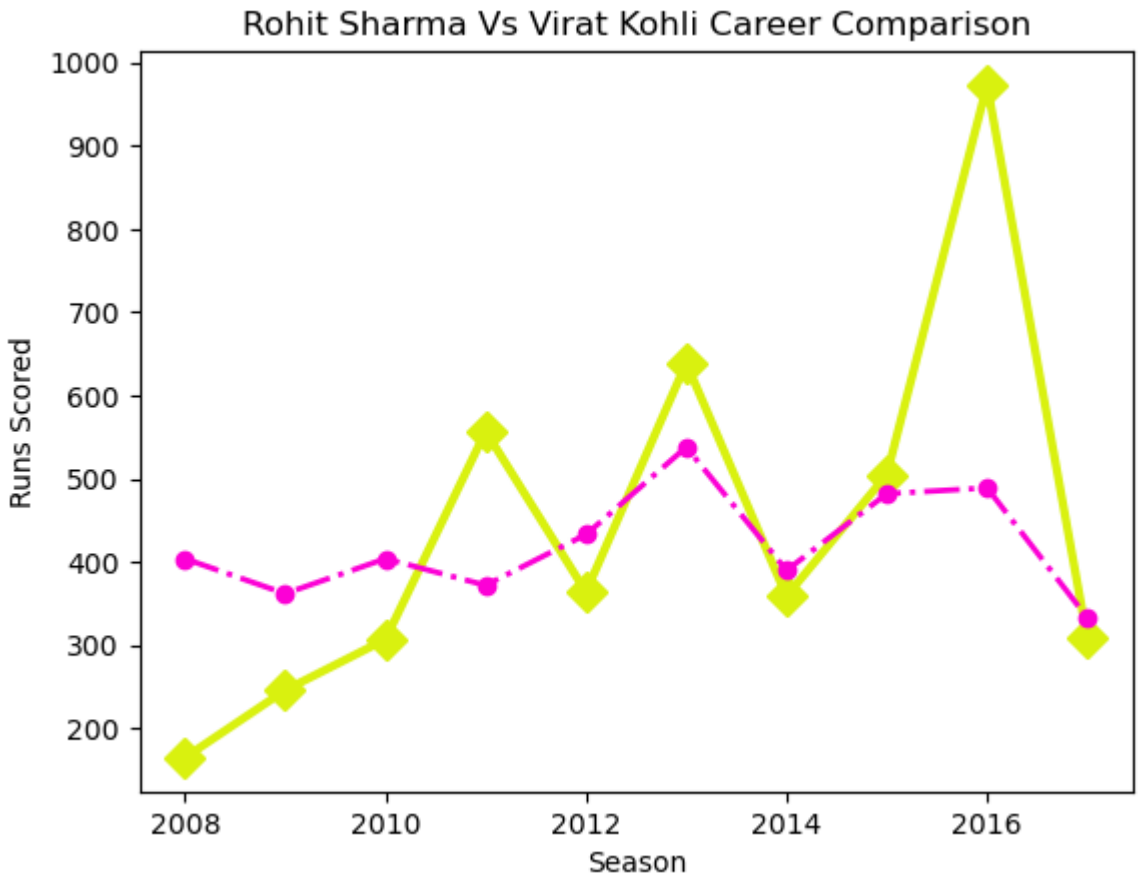
Out[ ]:
```
Text(0, 0.5, 'Runs Scored')
```



You can change the style and width of the lines. Common line styles include 'solid,' 'dotted,' 'dashed,' etc. In the example, 'solid' and 'dashdot' line styles are used.

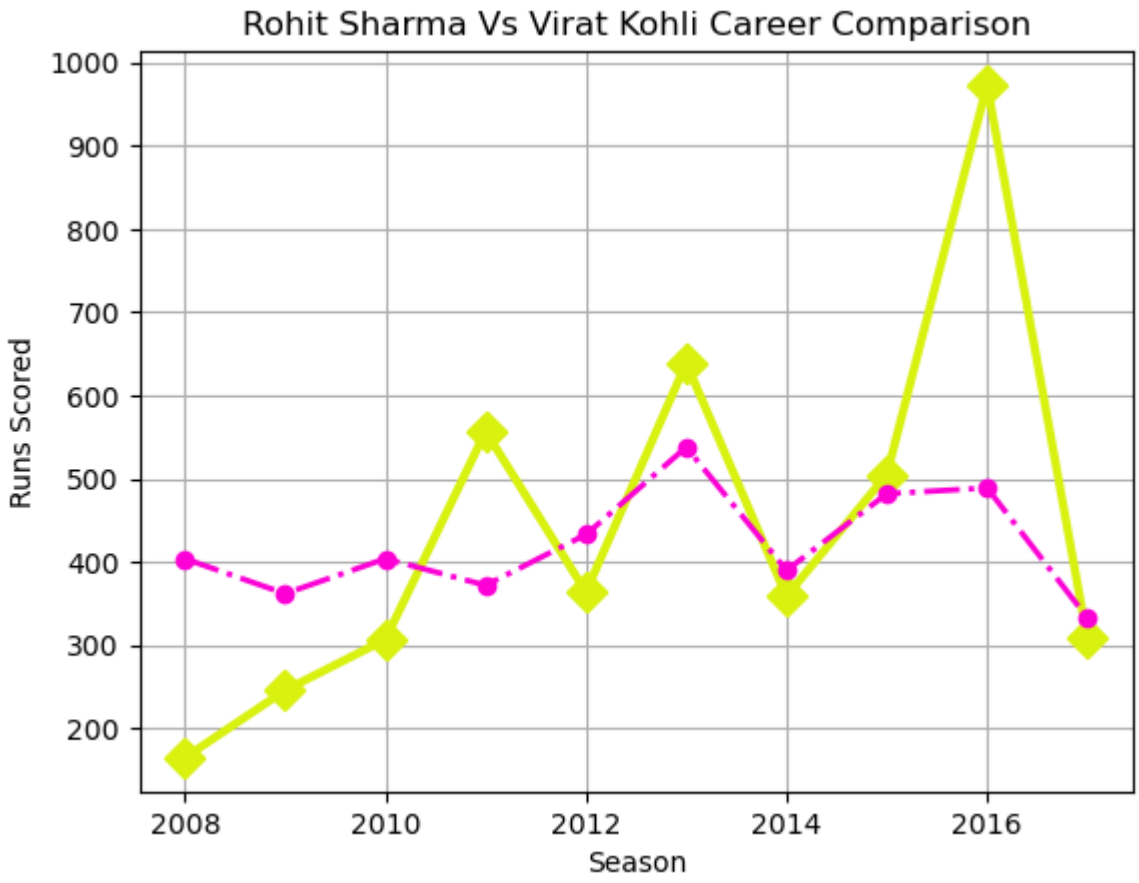
In [ ]:
```
# Marker
plt.plot(batsman['index'],batsman['V Kohli'],color='#D9F10F',linestyle='solid',line
plt.plot(batsman['index'],batsman['RG Sharma'],color='#FC00D6',linestyle='dashdot',

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')
```

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



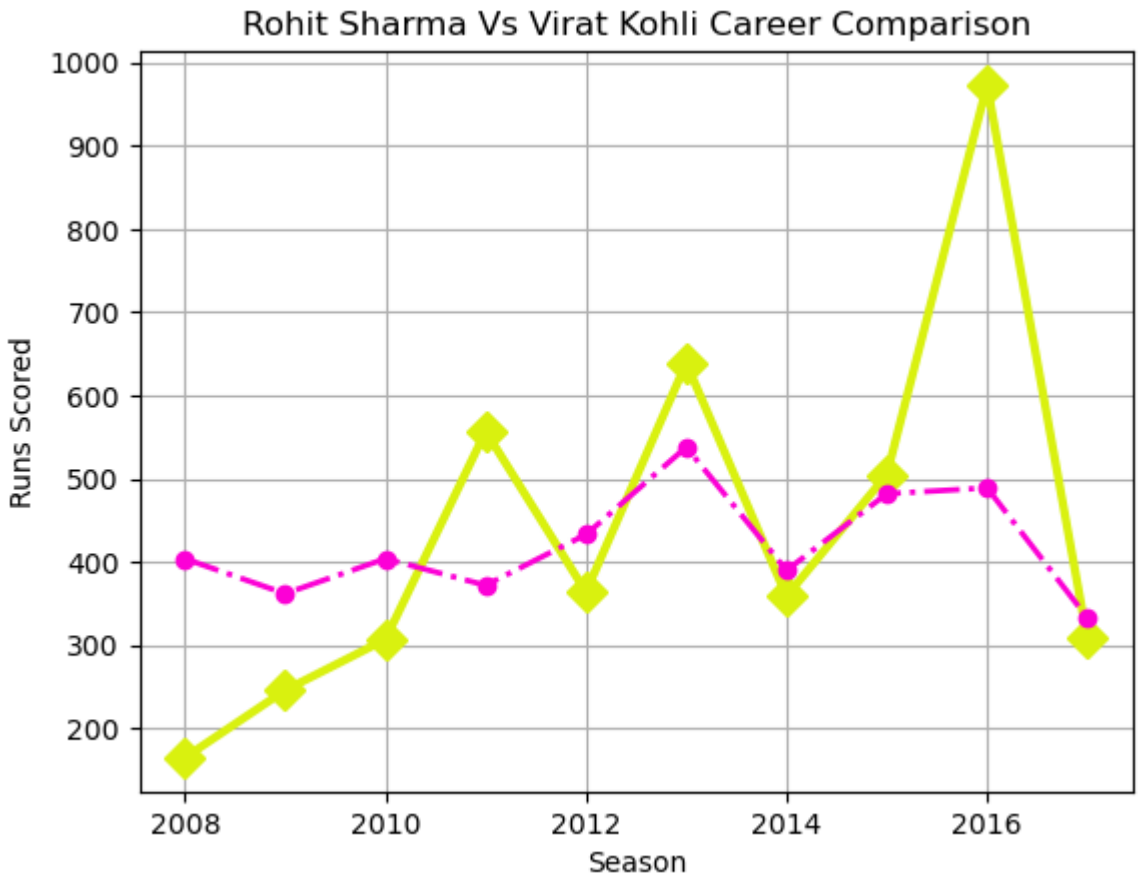
Markers are used to highlight data points on the line plot. You can customize markers' style and size. In the example, markers like 'D' and 'o' are used with different colors.

In [ ]:
```
# grid
plt.plot(batsman['index'],batsman['V Kohli'],color='#D9F10F',linestyle='solid',line
plt.plot(batsman['index'],batsman['RG Sharma'],color='#FC00D6',linestyle='dashdot',

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')

plt.grid()
```



Adding a grid to the plot can make it easier to read and interpret the data. The grid helps in aligning the data points with the tick marks on the axes.

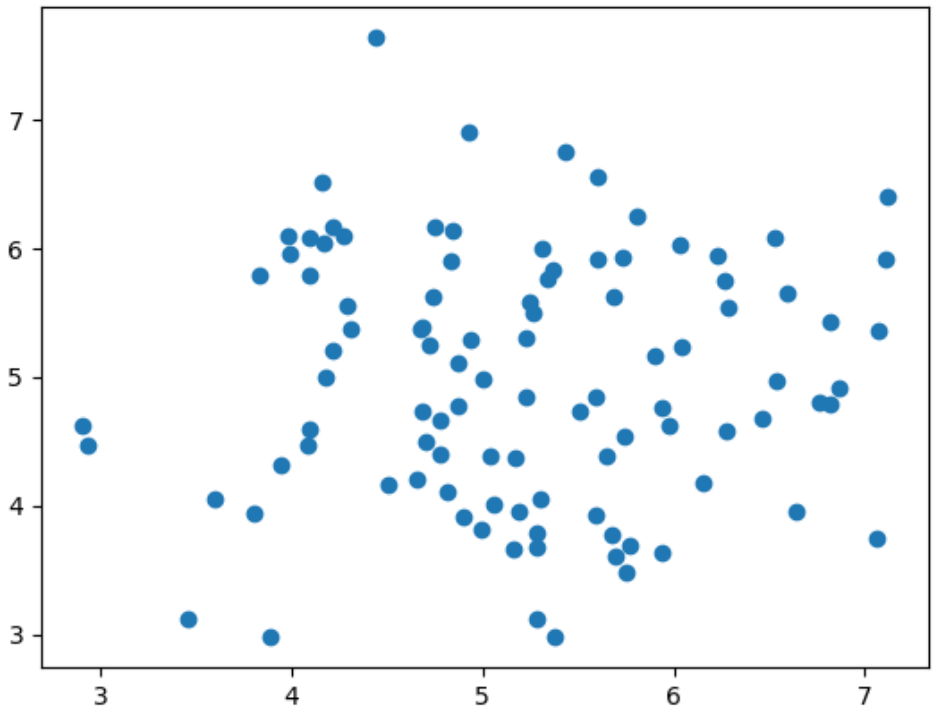
In [ ]:
```
# show
plt.plot(batsman['index'],batsman['V Kohli'],color='#D9F10F',linestyle='solid',line
plt.plot(batsman['index'],batsman['RG Sharma'],color='#FC00D6',linestyle='dashdot',

plt.title('Rohit Sharma Vs Virat Kohli Career Comparison')
plt.xlabel('Season')
plt.ylabel('Runs Scored')

plt.grid()

plt.show()
```



After customizing your plot, you can use plt.show() to display it. This command is often used in Jupyter notebooks or standalone Python scripts.

2D line plots are valuable for visualizing time series data, comparing trends in multiple datasets, and exploring the relationship between two continuous variables. Customization options in Matplotlib allow you to create visually appealing and informative plots for data analysis and presentation.

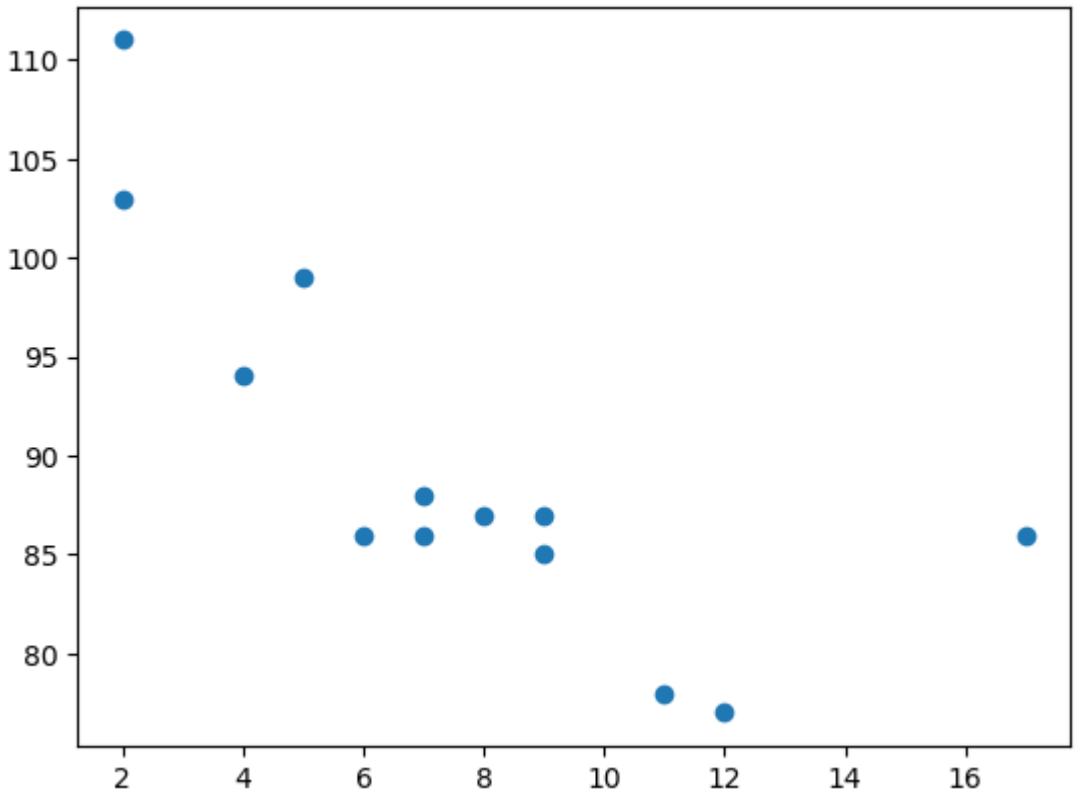
ScatterPlot and Bar Chart in Matplotlib



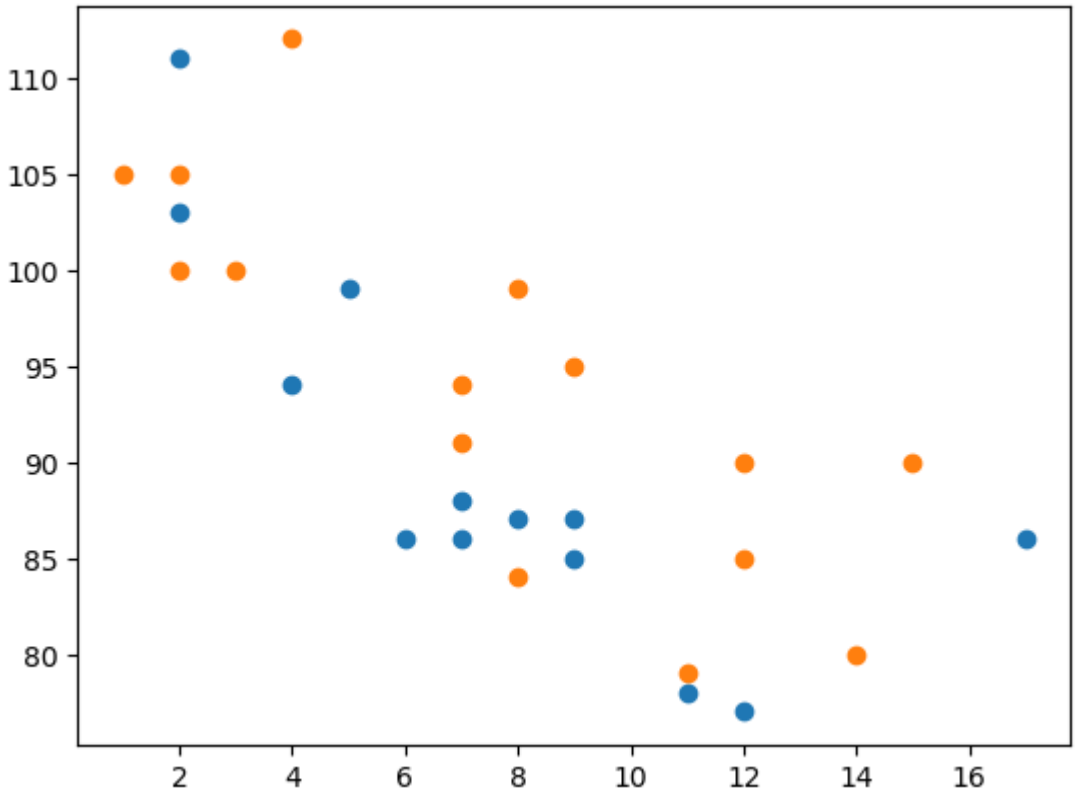
A scatter plot, also known as a scatterplot or scatter chart, is a type of data visualization used in statistics and data analysis. It's used to display the relationship between two variables by representing individual data points as points on a two-dimensional graph. Each point on the plot corresponds to a single data entry with values for both variables, making it a useful tool for identifying patterns, trends, clusters, or outliers in data.

- Bivariate Analysis
- numerical vs numerical
- Use case - Finding correlation

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



In [ ]:
```
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```

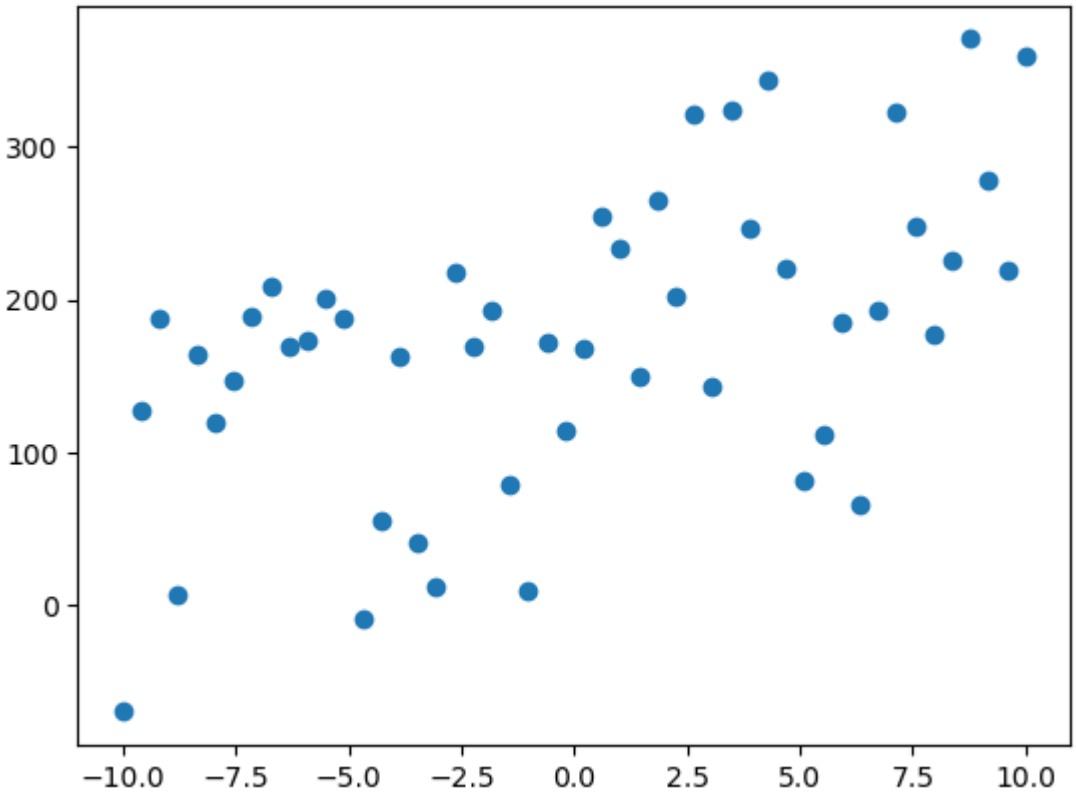


In [ ]:
```
# plt.scatter simple function
x = np.linspace(-10,10,50)

y = 10*x + 3 + np.random.randint(0,300,50)
y
```

Out[ ]:
```
array([-70.        , 127.08163265, 187.16326531,   6.24489796,
       164.32653061, 119.40816327, 146.48979592, 189.57142857,
       208.65306122, 169.73469388, 173.81632653, 200.89795918,
       187.97959184,  -8.93877551,  55.14285714, 162.2244898 ,
        40.30612245,  12.3877551 , 218.46938776, 169.55102041,
       193.63265306,  78.71428571,   9.79591837, 171.87755102,
       113.95918367, 168.04081633, 255.12244898, 233.20408163,
       150.28571429, 265.36734694, 202.44897959, 321.53061224,
       142.6122449 , 324.69387755, 246.7755102 , 343.85714286,
       220.93877551,  81.02040816, 111.10204082, 185.18367347,
        66.26530612, 193.34693878, 323.42857143, 248.51020408,
       177.59183673, 225.67346939, 370.75510204, 277.83673469,
       218.91836735, 360.        ])
```

In [ ]:
```
plt.scatter(x,y)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x264627ccc70>
```



In [ ]:
```
import numpy as np
import pandas as pd
```

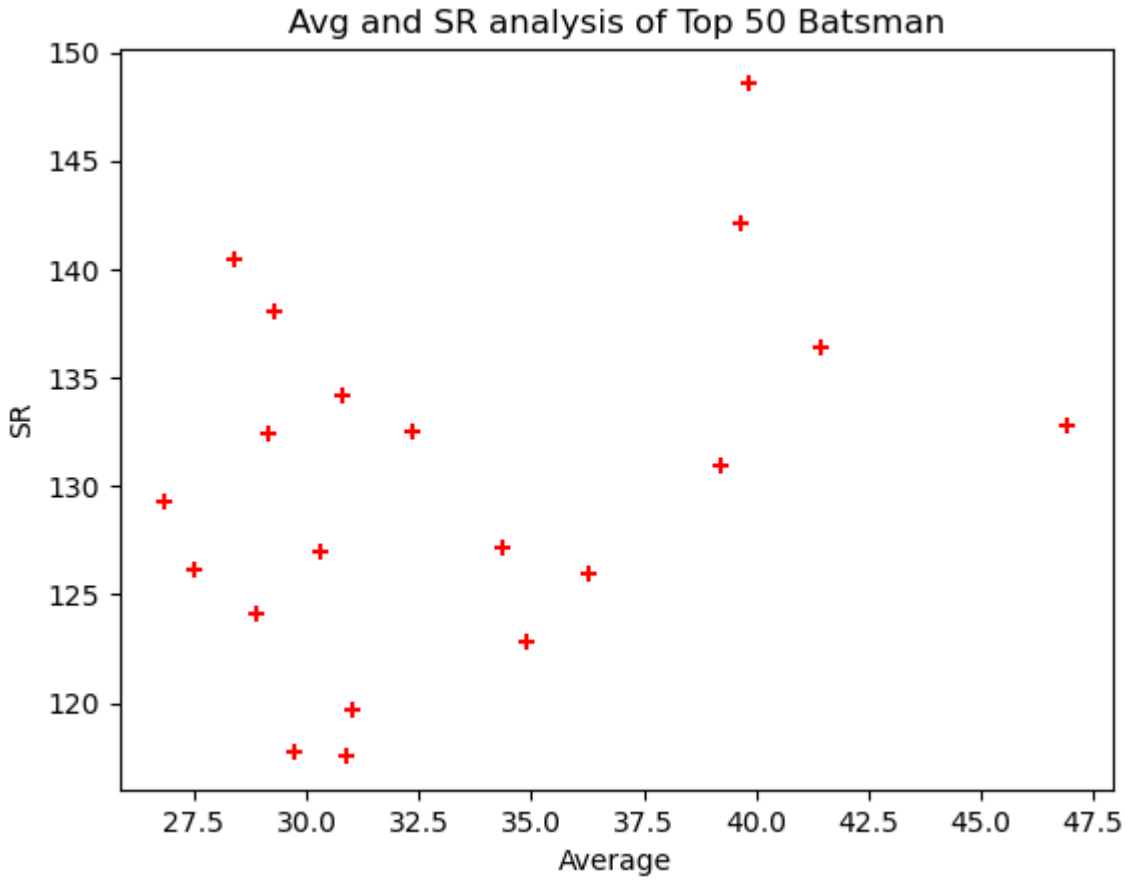
In [ ]:
```
# plt.scatter on pandas data
df = pd.read_csv('Data\Day47\Batter.csv')
df = df.head(20)
df
```

Out[ ]:

	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
5	AB de Villiers	5181	39.853846	148.580442
6	CH Gayle	4997	39.658730	142.121729
7	MS Dhoni	4978	39.196850	130.931089
8	RV Uthappa	4954	27.522222	126.152279
9	KD Karthik	4377	26.852761	129.267572
10	G Gambhir	4217	31.007353	119.665153
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
16	SV Samson	3526	29.140496	132.407060
17	KA Pollard	3437	28.404959	140.457703
18	F du Plessis	3403	34.373737	127.167414
19	YK Pathan	3222	29.290909	138.046272

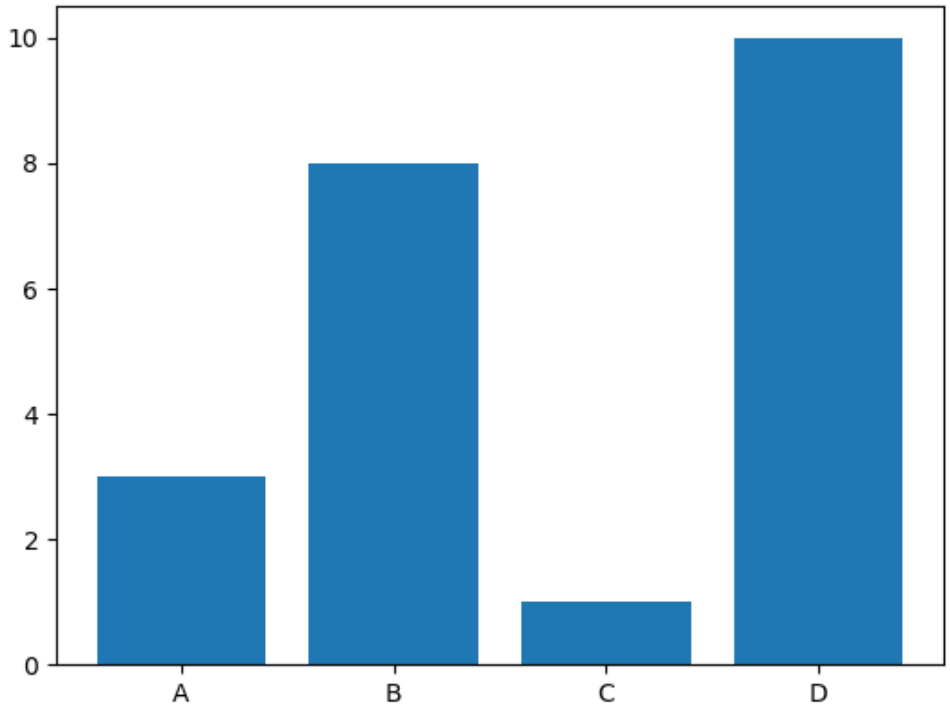
In [ ]:
```
# marker
plt.scatter(df['avg'],df['strike_rate'],color='red',marker='+')
plt.title('Avg and SR analysis of Top 50 Batsman')
plt.xlabel('Average')
plt.ylabel('SR')
```

Out[ ]:
```
Text(0, 0.5, 'SR')
```



Scatter plots are particularly useful for visualizing the distribution of data, identifying correlations or relationships between variables, and spotting outliers. You can adjust the appearance and characteristics of the scatter plot to suit your needs, including marker size, color, and transparency. This makes scatter plots a versatile tool for data exploration and analysis.

Bar plot

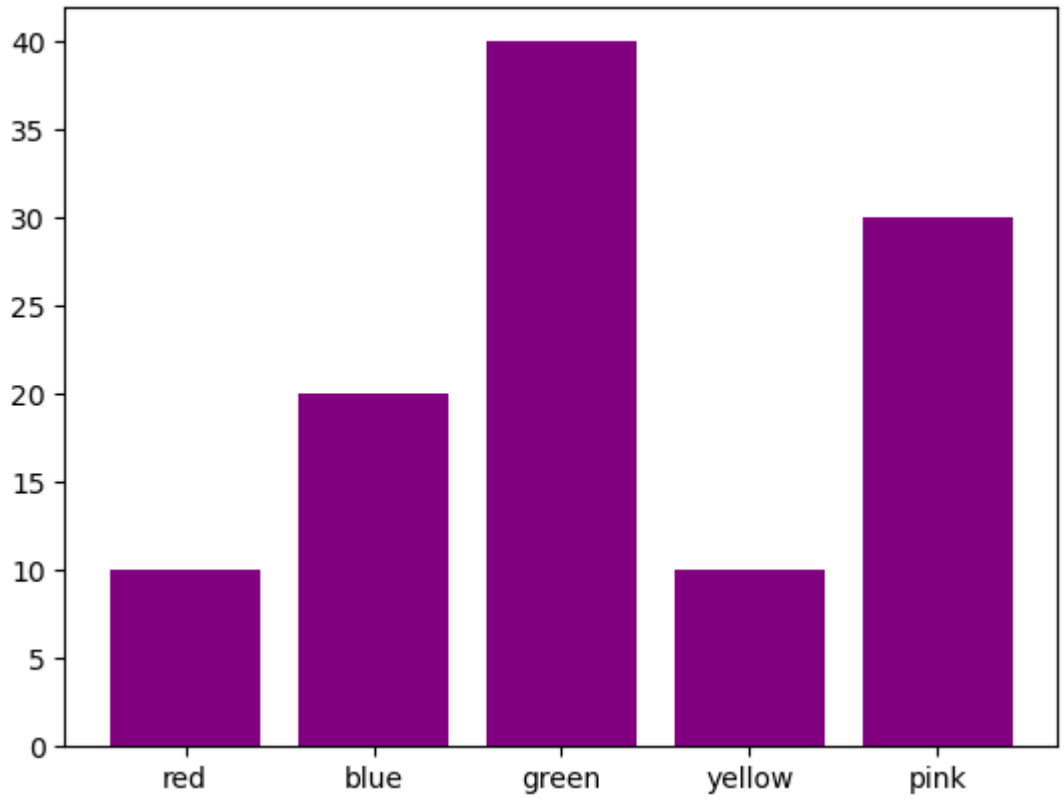


A bar plot, also known as a bar chart or bar graph, is a type of data visualization that is used to represent categorical data with rectangular bars. Each bar's length or height is proportional to the value it represents. Bar plots are typically used to compare and display the relative sizes or quantities of different categories or groups.

- Bivariate Analysis
- Numerical vs Categorical
- Use case - Aggregate analysis of groups

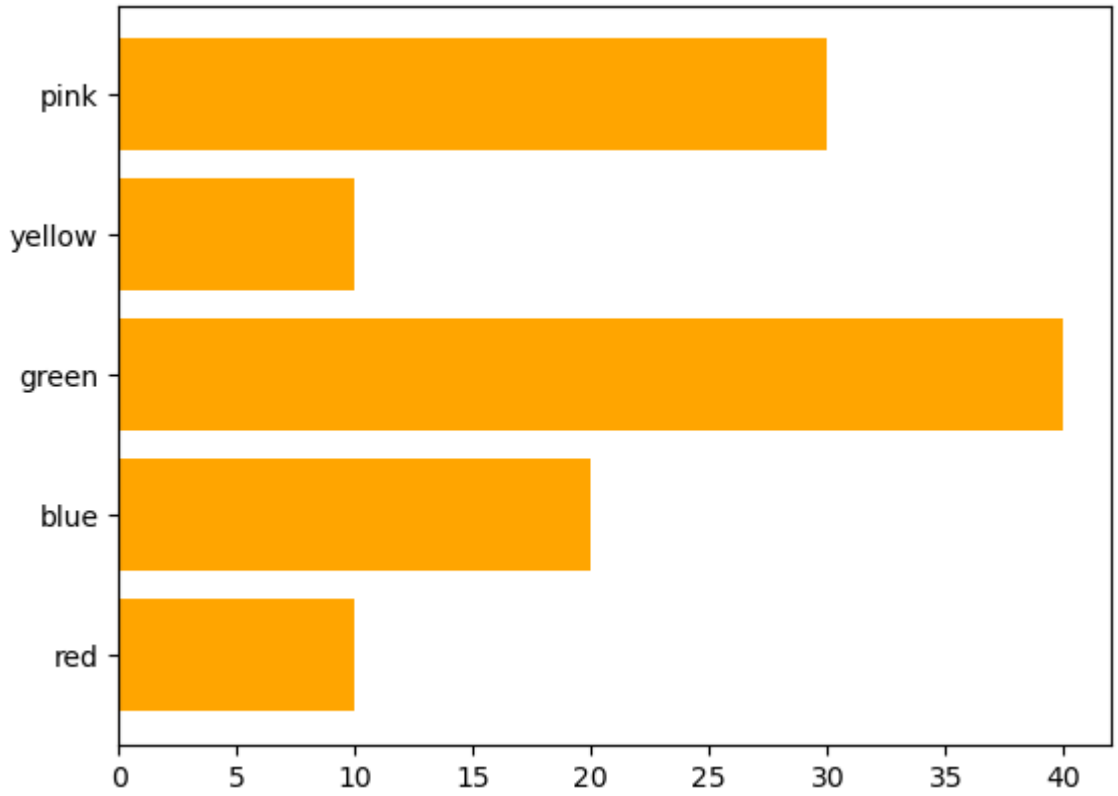
In [ ]:
```
# simple bar chart
children = [10,20,40,10,30]
colors = ['red','blue','green','yellow','pink']

plt.bar(colors,children,color='Purple')
```

Out[ ]:
```
<BarContainer object of 5 artists>
```



In [ ]:
```
# horizontal bar chart
plt.barh(colors,children,color='Orange')
```

Out[ ]:
```
<BarContainer object of 5 artists>
```

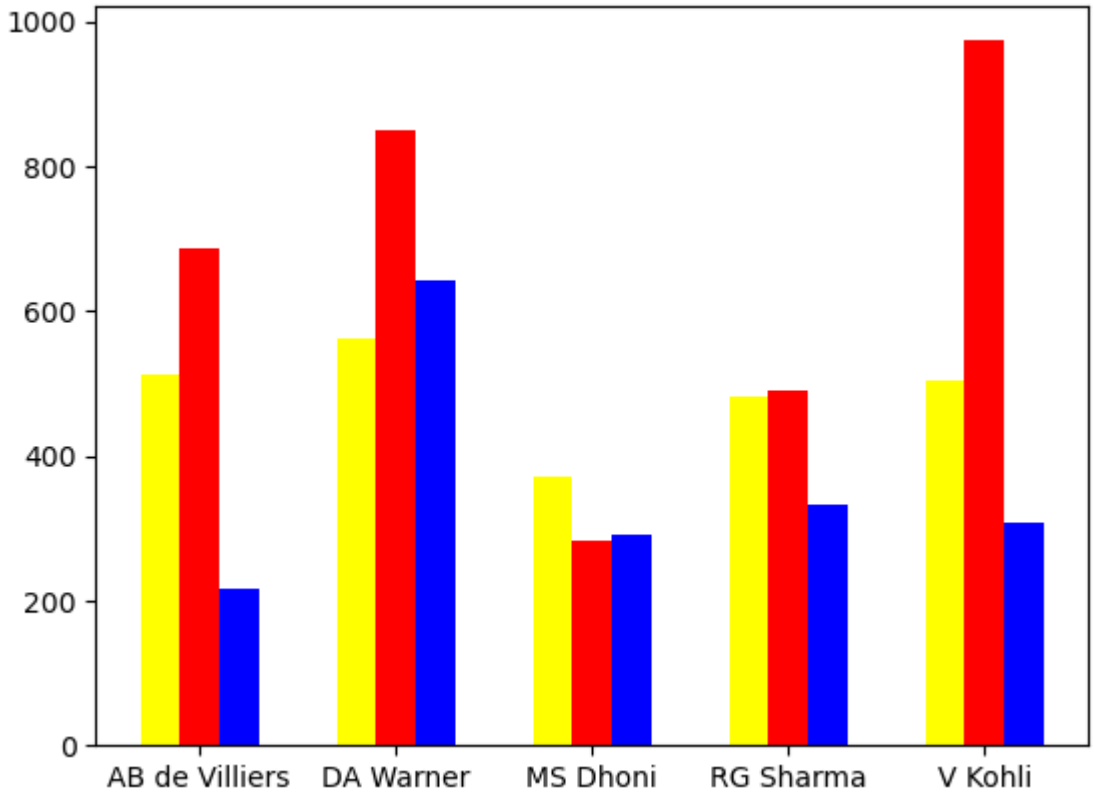


In [ ]:
```
# color and label
df = pd.read_csv('Data\Day47\Batsman_season.csv')
df
```

Out[ ]:

	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

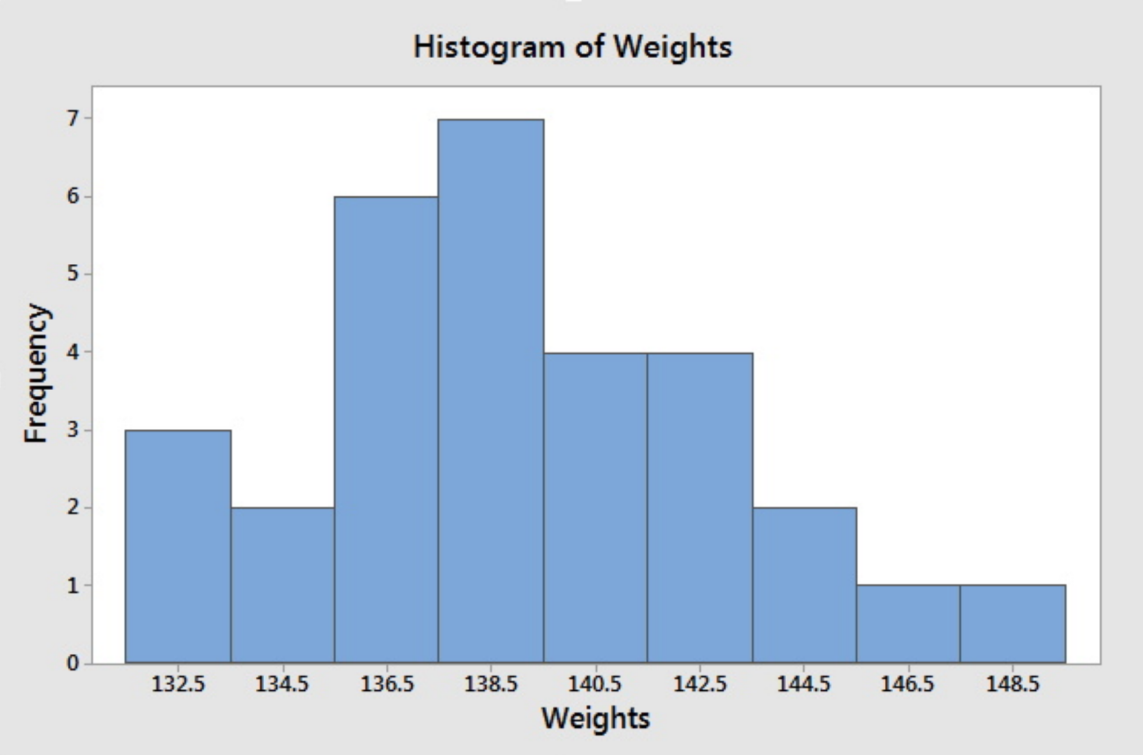
In [ ]:
```
plt.bar(np.arange(df.shape[0]) - 0.2,df['2015'],width=0.2,color='yellow')
plt.bar(np.arange(df.shape[0]),df['2016'],width=0.2,color='red')
plt.bar(np.arange(df.shape[0]) + 0.2,df['2017'],width=0.2,color='blue')

plt.xticks(np.arange(df.shape[0]), df['batsman'])

plt.show()
```



Bar plots are useful for comparing the values of different categories and for showing the distribution of data within each category. They are commonly used in various fields, including business, economics, and data analysis, to make comparisons and convey information about categorical data. You can customize bar plots to make them more visually appealing and informative.

Histogram and Pie chart in Matplotlib

Histogram



A histogram is a type of chart that shows the distribution of numerical data. It's a graphical representation of data where data is grouped into continuous number ranges and each range corresponds to a vertical bar. The horizontal axis displays the number range, and the vertical axis (frequency) represents the amount of data that is present in each range.

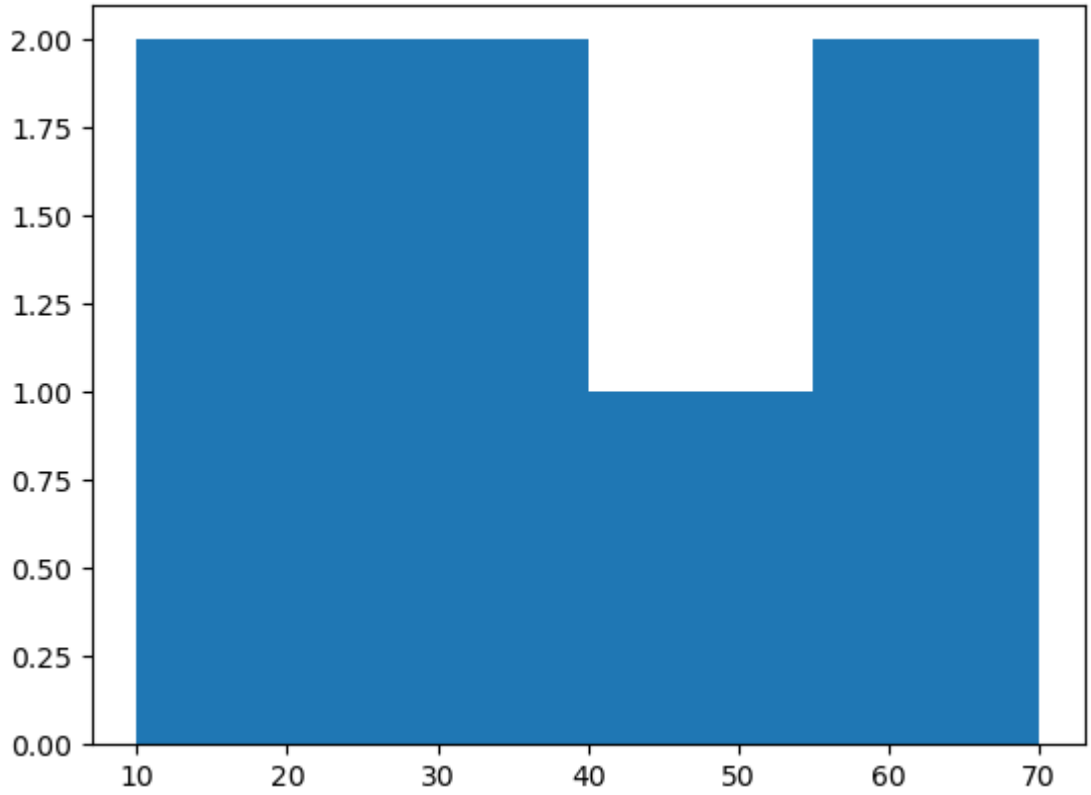
A histogram is a set of rectangles with bases along with the intervals between class boundaries and with areas proportional to frequencies in the corresponding classes. The x-axis of the graph represents the class interval, and the y-axis shows the various frequencies corresponding to different class intervals. A histogram is a type of data visualization used to represent the distribution of a dataset, especially when dealing with continuous or numeric data. It displays the frequency or count of data points falling into specific intervals or "bins" along a continuous range. Histograms provide insights into the shape, central tendency, and spread of a dataset.

- Univariate Analysis
- Numerical col
- Use case - Frequency Count

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

In [ ]:
```
# simple data

data = [32,45,56,10,15,27,61]

plt.hist(data,bins=[10,25,40,55,70])
```

Out[ ]:
```
(array([2., 2., 1., 2.]),
 array([10., 25., 40., 55., 70.]),
 <BarContainer object of 4 artists>)
```



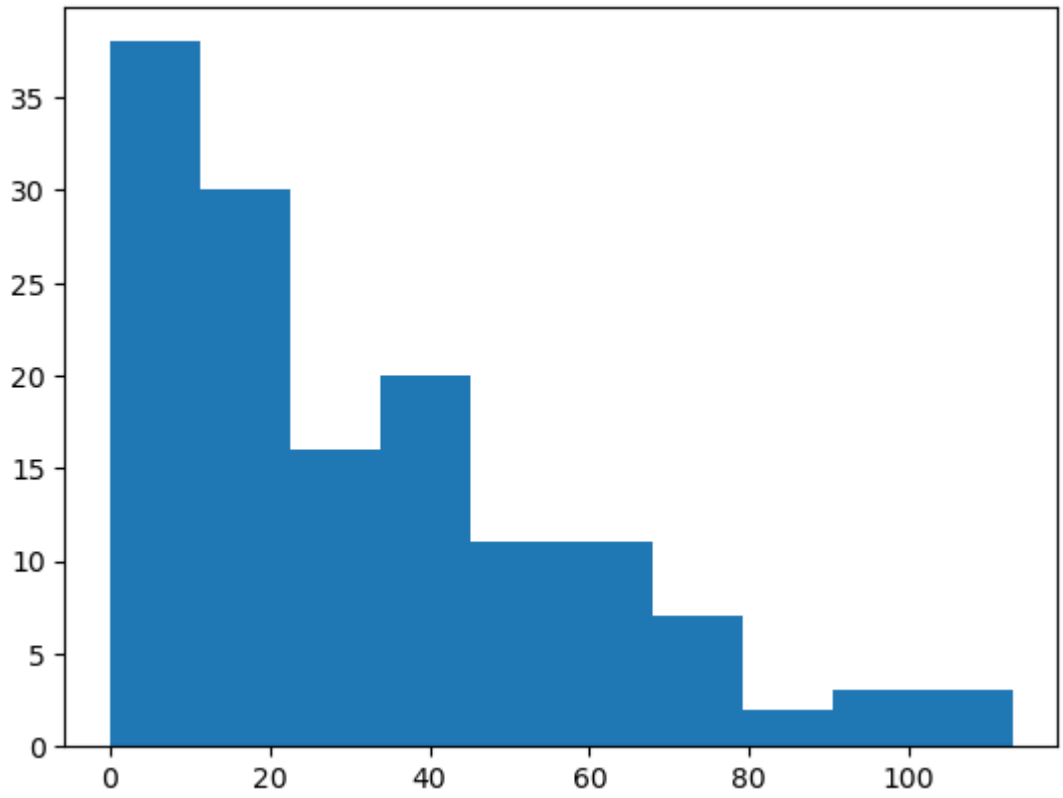
In [ ]:
```
# on some data
df = pd.read_csv('Data\Day48\Vk.csv')
df
```

Out[ ]:

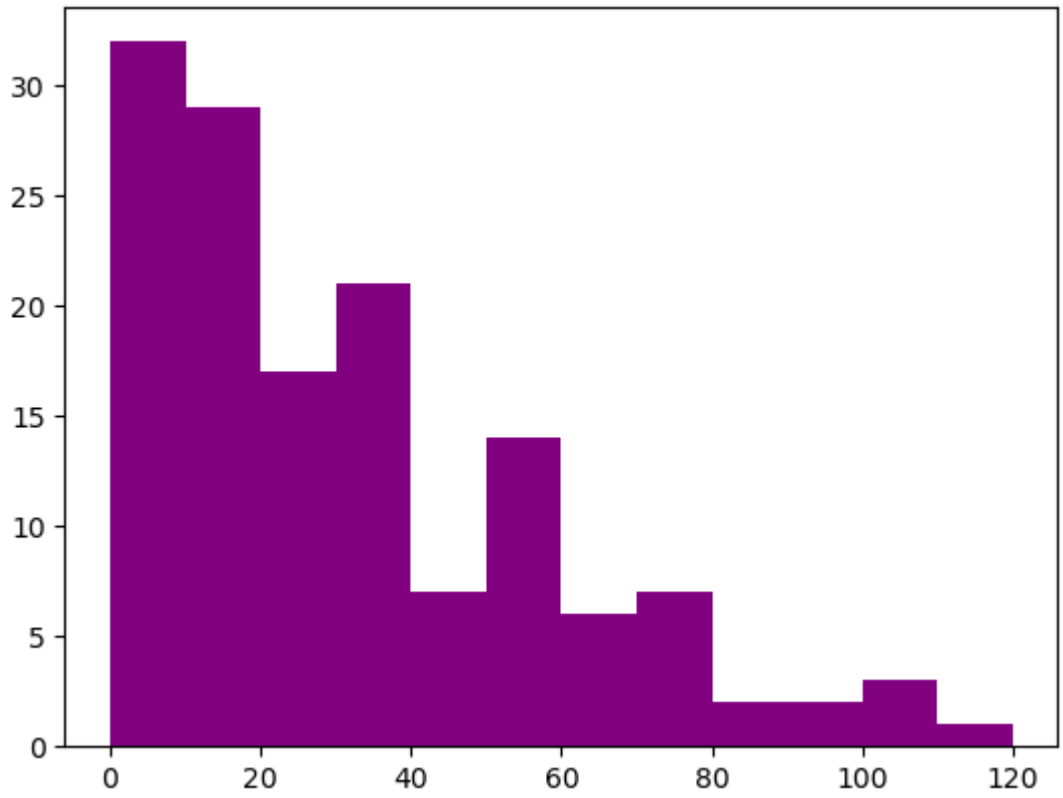
	match_id	batsman_runs
0	12	62
1	17	28
2	20	64
3	27	0
4	30	10
...	...	...
136	624	75
137	626	113
138	632	54
139	633	0
140	636	54

141 rows × 2 columns

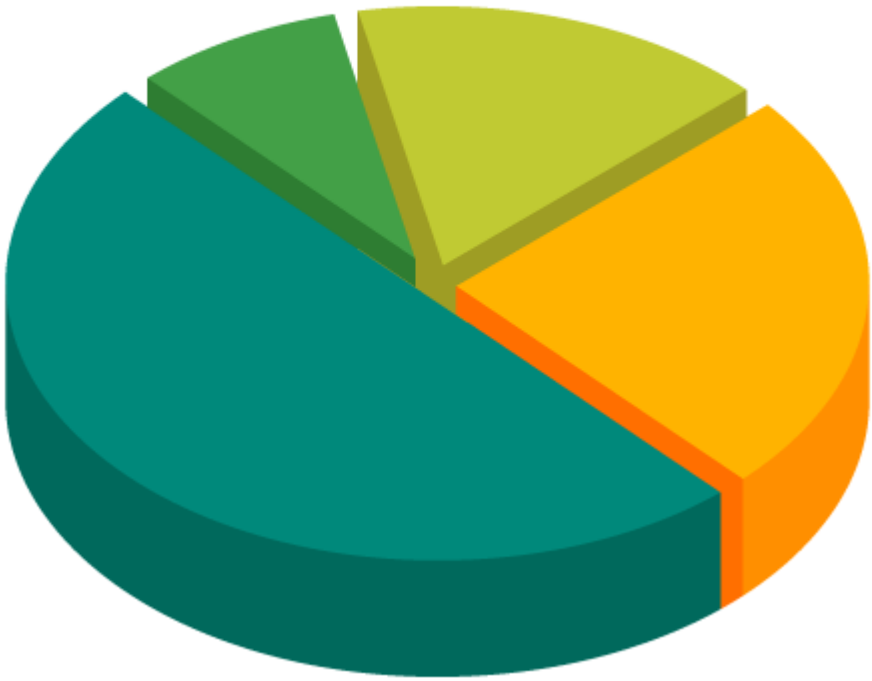
In [ ]:
```
plt.hist(df['batsman_runs'])
plt.show()
```



In [ ]:
```
# handling bins
plt.hist(df['batsman_runs'],bins=[0,10,20,30,40,50,60,70,80,90,100,110,120],color='
plt.show()
```



Pie Chart

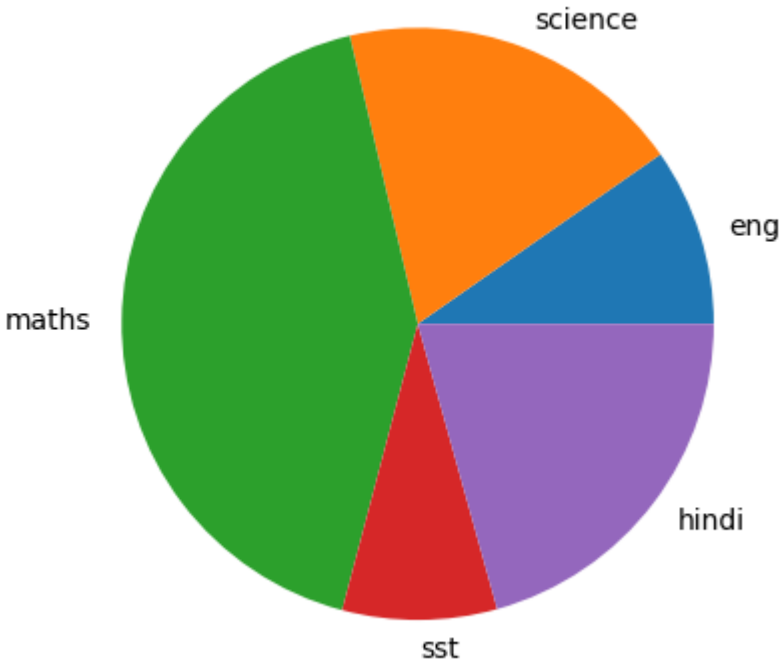


A pie chart is a circular graph that's divided into slices to illustrate numerical proportion. The slices of the pie show the relative size of the data. The arc length of each slice, and consequently its central angle and area, is proportional to the quantity it represents.

All slices of the pie add up to make the whole equaling 100 percent and 360 degrees. Pie charts are often used to represent sample data. Each of these categories is represented as a “slice of the pie”. The size of each slice is directly proportional to the number of data points that belong to a particular category.

- Univariate/Bivariate Analysis
- Categorical vs numerical
- Use case - To find contibution on a standard scale

In [ ]:
```
# simple data
data = [23,45,100,20,49]
subjects = ['eng','science','maths','sst','hindi']
plt.pie(data,labels=subjects)

plt.show()
```

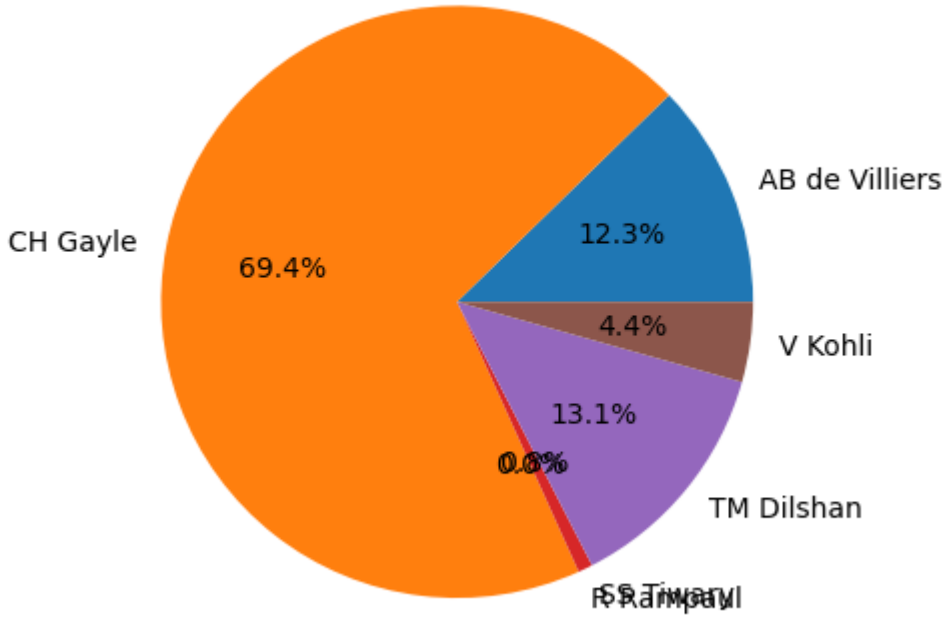


In [ ]:
```
# dataset
df = pd.read_csv('Data\Day48\Gayle-175.csv')
df
```

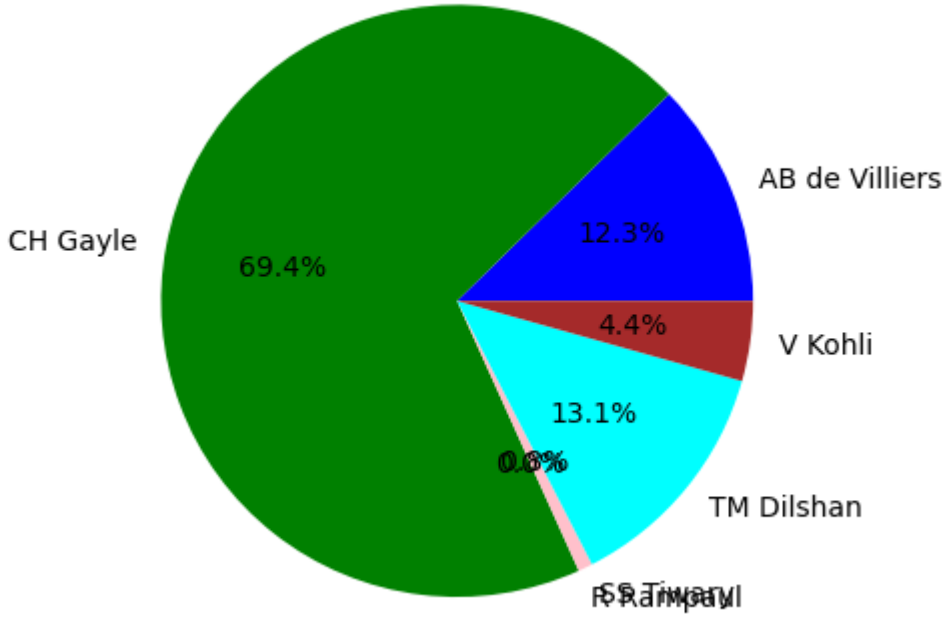
Out[ ]:

	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

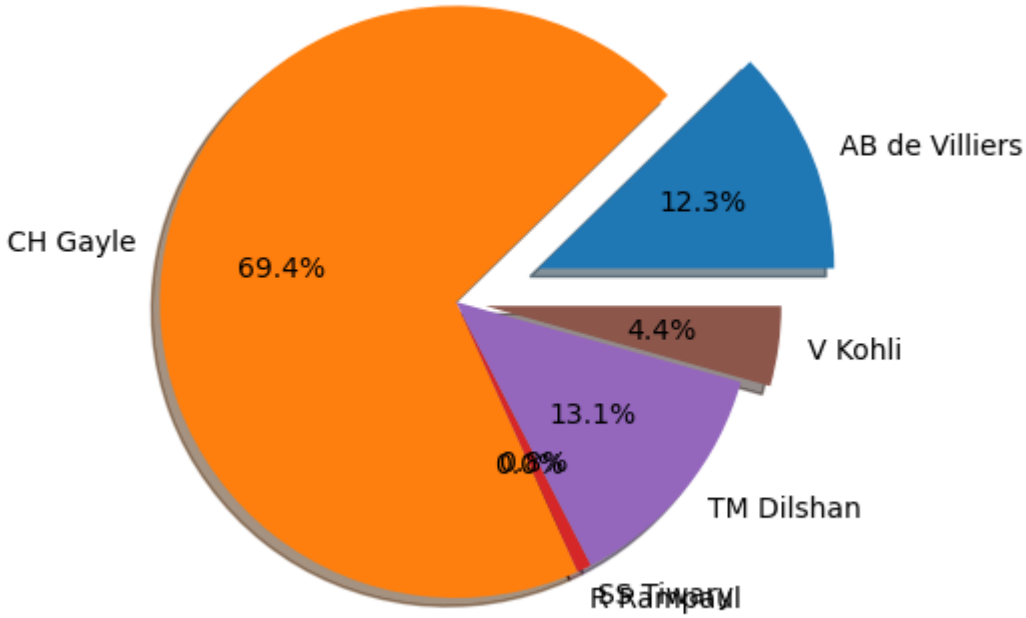
In [ ]:
```
plt.pie(df['batsman_runs'],labels=df['batsman'],autopct='%0.1f%%')
plt.show()
```



In [ ]:
```
# percentage and colors
plt.pie(df['batsman_runs'],labels=df['batsman'],autopct='%0.1f%%',colors=['blue','g
plt.show()
```



In [ ]:
```
# explode shadow
plt.pie(df['batsman_runs'],labels=df['batsman'],autopct='%0.1f%%',explode=[0.3,0,0,
plt.show()
```



Matplotlib Doc Website :

https://matplotlib.org/stable/

Advanced Matplotlib(part-1)

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

Colored Scatterplots

In [ ]:
```
iris = pd.read_csv('Data\Day49\iris.csv')
iris.sample(5)
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
146	147	6.3	2.5	5.0	1.9	Iris-virginica
123	124	6.3	2.7	4.9	1.8	Iris-virginica
78	79	6.0	2.9	4.5	1.5	Iris-versicolor
127	128	6.1	3.0	4.9	1.8	Iris-virginica
143	144	6.8	3.2	5.9	2.3	Iris-virginica

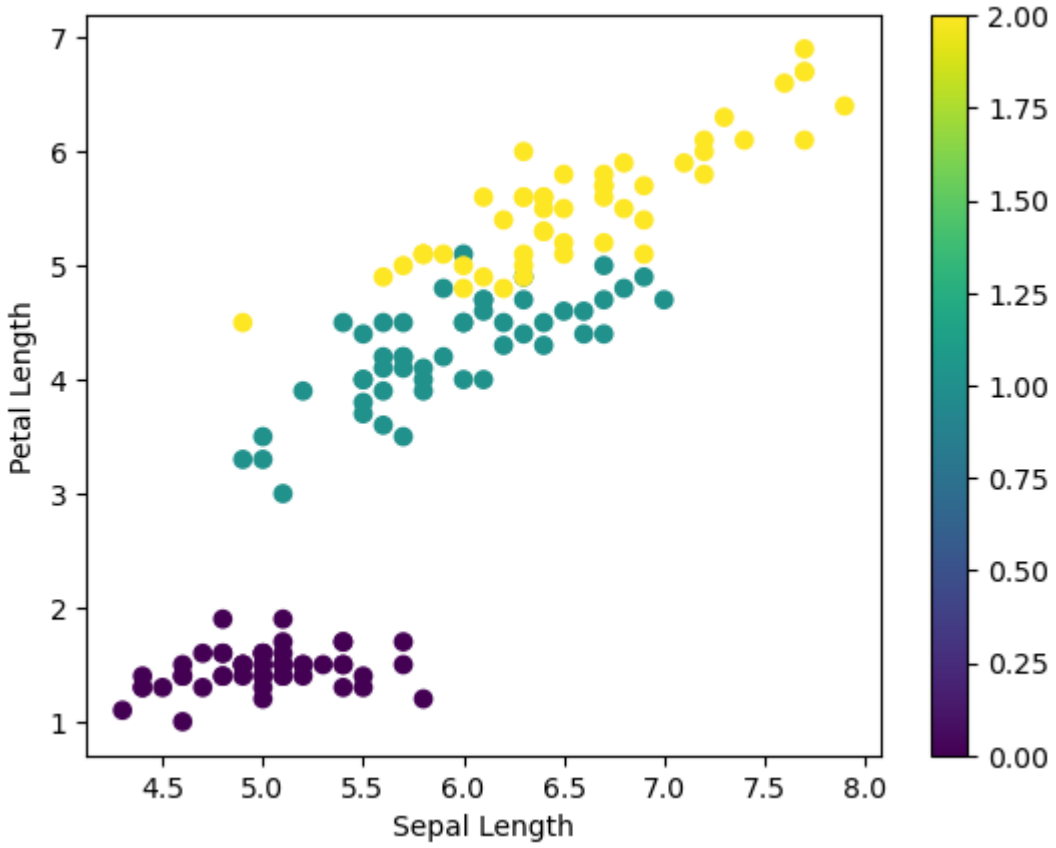
In [ ]:
```
iris['Species'] = iris['Species'].replace({'Iris-setosa':0,'Iris-versicolor':1,'Iri
iris.sample(5)
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
90	91	5.5	2.6	4.4	1.2	1
25	26	5.0	3.0	1.6	0.2	0
89	90	5.5	2.5	4.0	1.3	1
95	96	5.7	3.0	4.2	1.2	1
148	149	6.2	3.4	5.4	2.3	2

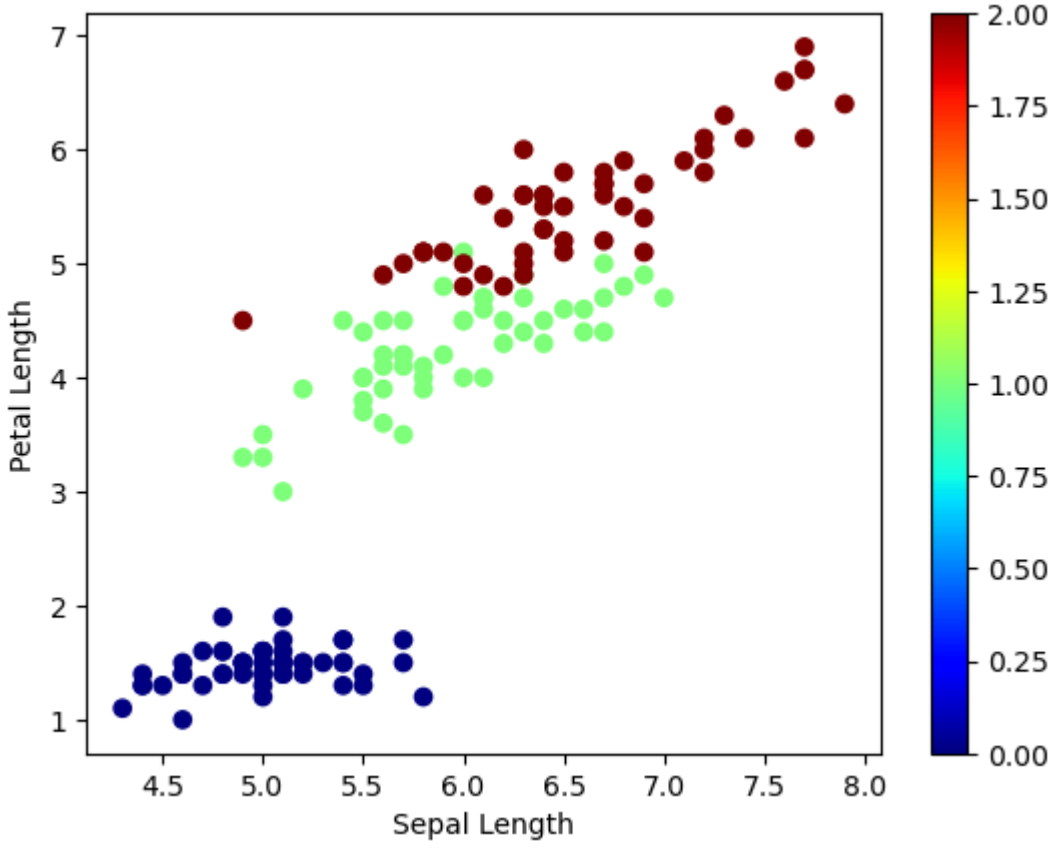
In [ ]:
```
plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'])
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec76de6880>
```



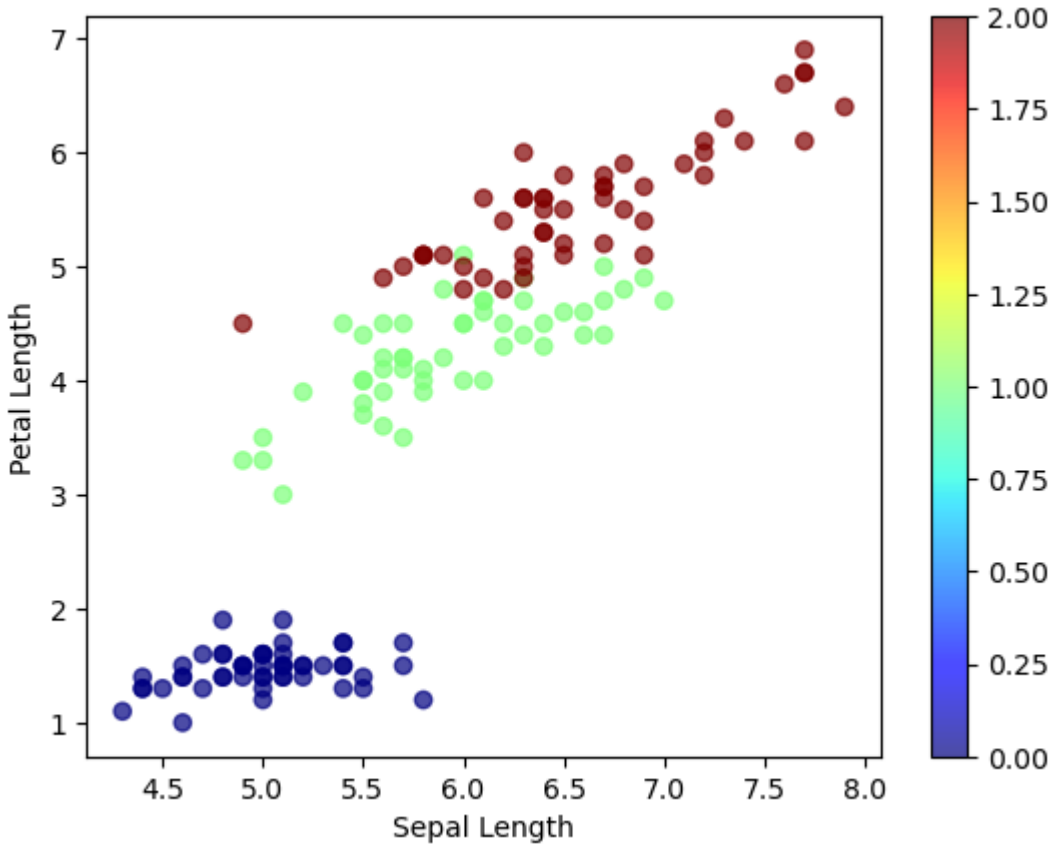
In [ ]:
```
# cmap
plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'],cmap='jet
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec75d12f10>
```



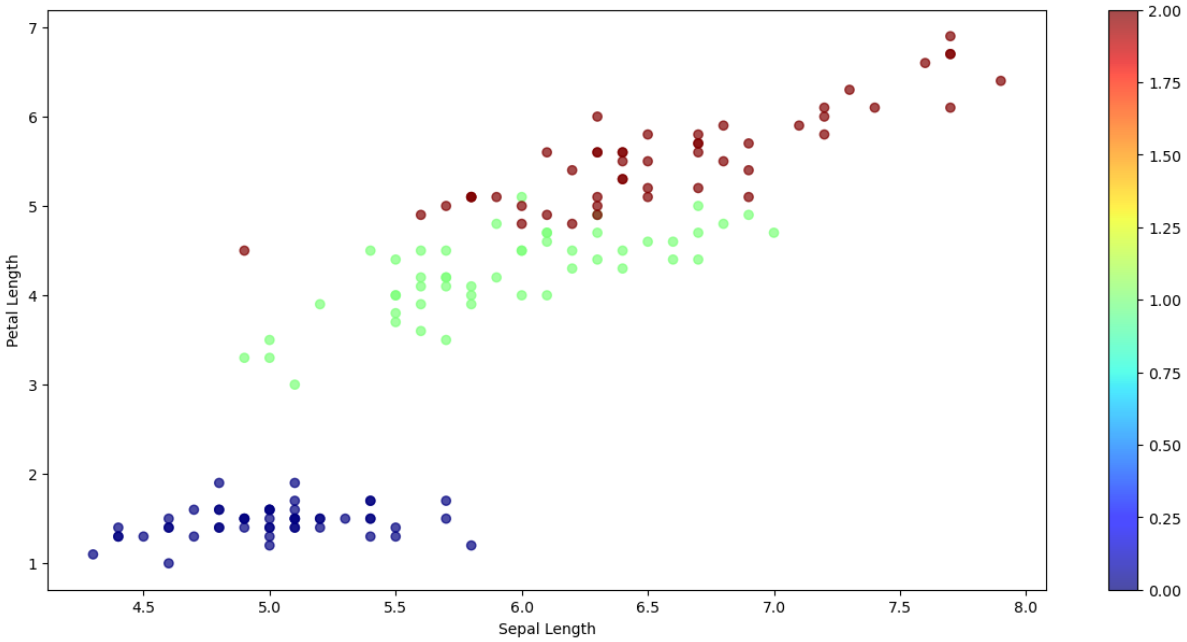
In [ ]:
```
# alpha
plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'],cmap='jet
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec76e54790>
```



In [ ]:
```
# plot size
plt.figure(figsize=(15,7))

plt.scatter(iris['SepalLengthCm'],iris['PetalLengthCm'],c=iris['Species'],cmap='jet
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1ec76f3bf40>
```



Annotations

In [ ]:
```
batters = pd.read_csv('Data\Day49\Batter.csv')
```

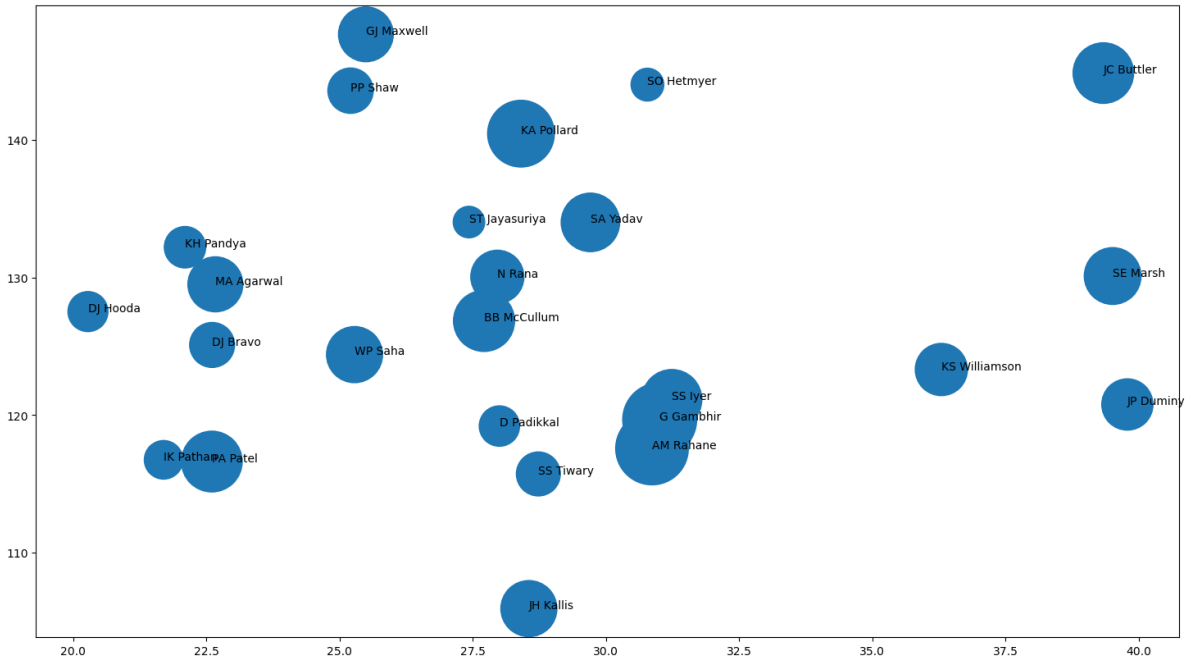
In [ ]:
```
sample_df = batters.head(100).sample(25,random_state=5)
```

In [ ]:
```
sample_df
```

Out[ ]:

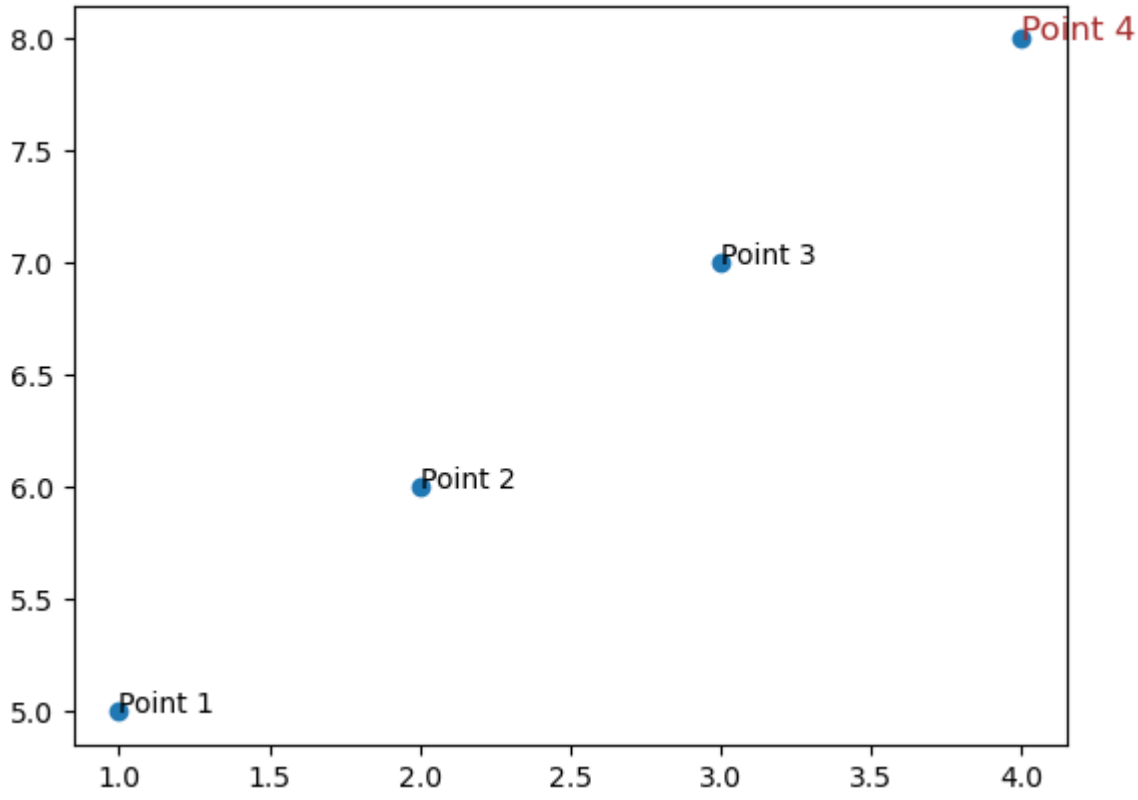
	batter	runs	avg	strike_rate
66	KH Pandya	1326	22.100000	132.203390
32	SE Marsh	2489	39.507937	130.109775
46	JP Duminy	2029	39.784314	120.773810
28	SA Yadav	2644	29.707865	134.009123
74	IK Pathan	1150	21.698113	116.751269
23	JC Buttler	2832	39.333333	144.859335
10	G Gambhir	4217	31.007353	119.665153
20	BB McCullum	2882	27.711538	126.848592
17	KA Pollard	3437	28.404959	140.457703
35	WP Saha	2427	25.281250	124.397745
97	ST Jayasuriya	768	27.428571	134.031414
37	MA Agarwal	2335	22.669903	129.506378
70	DJ Hooda	1237	20.278689	127.525773
40	N Rana	2181	27.961538	130.053667
60	SS Tiwary	1494	28.730769	115.724245
34	JH Kallis	2427	28.552941	105.936272
42	KS Williamson	2105	36.293103	123.315759
57	DJ Bravo	1560	22.608696	125.100241
12	AM Rahane	4074	30.863636	117.575758
69	D Padikkal	1260	28.000000	119.205298
94	SO Hetmyer	831	30.777778	144.020797
56	PP Shaw	1588	25.206349	143.580470
22	PA Patel	2848	22.603175	116.625717
39	GJ Maxwell	2320	25.494505	147.676639
24	SS Iyer	2780	31.235955	121.132898

In [ ]:
```
plt.figure(figsize=(18,10))
plt.scatter(sample_df['avg'],sample_df['strike_rate'],s=sample_df['runs'])

for i in range(sample_df.shape[0]):
  plt.text(sample_df['avg'].values[i],sample_df['strike_rate'].values[i],sample_df[
```



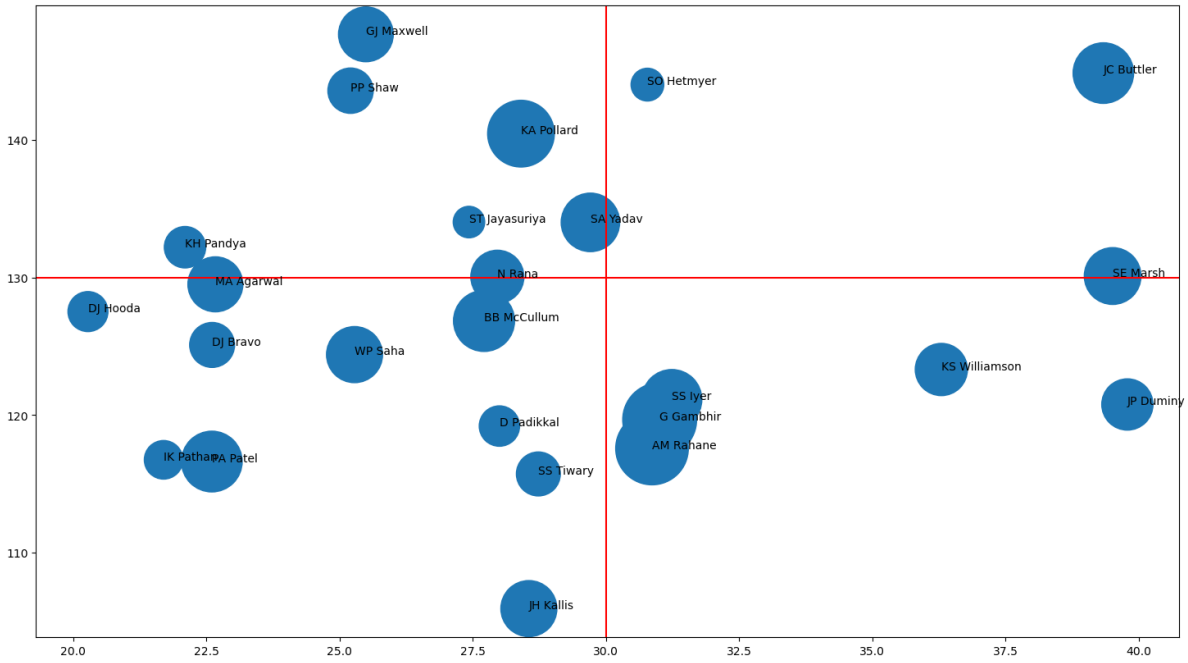
In [ ]:
```
x = [1,2,3,4]
y = [5,6,7,8]

plt.scatter(x,y)
plt.text(1,5,'Point 1')
plt.text(2,6,'Point 2')
plt.text(3,7,'Point 3')
plt.text(4,8,'Point 4',fontdict={'size':12,'color':'brown'})
```

Out[ ]:
```
Text(4, 8, 'Point 4')
```



In [ ]:
```
### Horizontal and Vertical lines

plt.figure(figsize=(18,10))
plt.scatter(sample_df['avg'],sample_df['strike_rate'],s=sample_df['runs'])
```

```
plt.axhline(130,color='red')

plt.axvline(30,color='red')

for i in range(sample_df.shape[0]):
  plt.text(sample_df['avg'].values[i],sample_df['strike_rate'].values[i],sample_df[
```



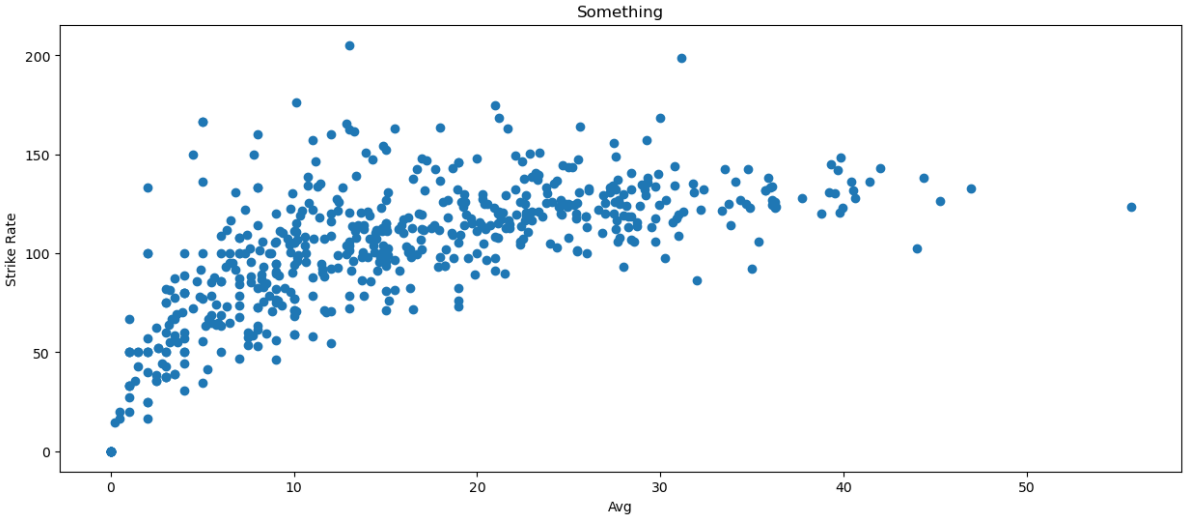
Subplots

In [ ]:
```
# A diff way to plot graphs
batters.head()
```

Out[ ]:

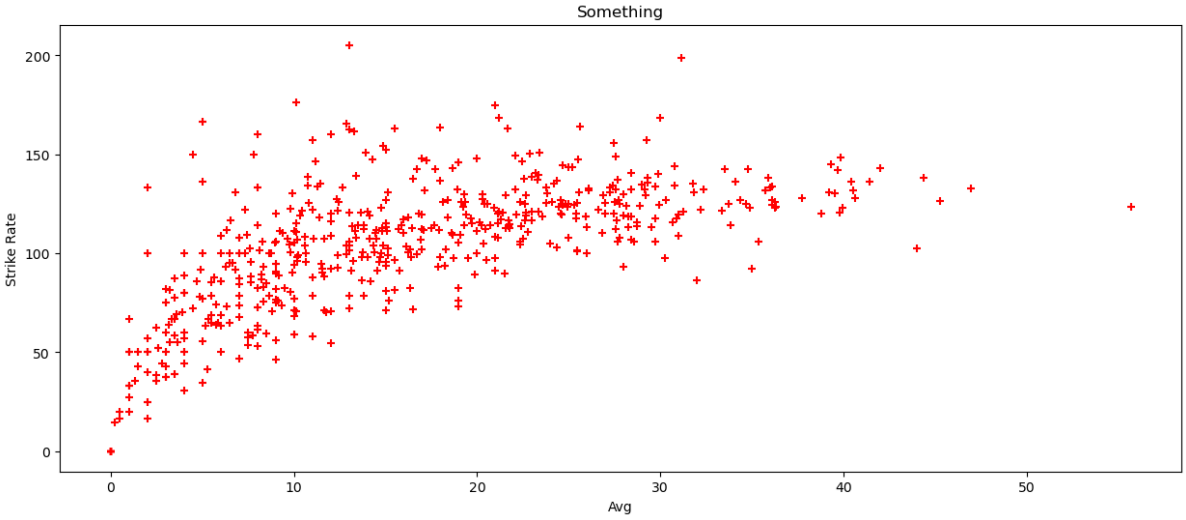
	batter	runs	avg	strike_rate
0	V Kohli	6634	36.251366	125.977972
1	S Dhawan	6244	34.882682	122.840842
2	DA Warner	5883	41.429577	136.401577
3	RG Sharma	5881	30.314433	126.964594
4	SK Raina	5536	32.374269	132.535312

In [ ]:
```
plt.figure(figsize=(15,6))
plt.scatter(batters['avg'],batters['strike_rate'])
plt.title('Something')
plt.xlabel('Avg')
plt.ylabel('Strike Rate')

plt.show()
```



In [ ]:
```
fig,ax = plt.subplots(figsize=(15,6))

ax.scatter(batters['avg'],batters['strike_rate'],color='red',marker='+')
ax.set_title('Something')
ax.set_xlabel('Avg')
ax.set_ylabel('Strike Rate')

fig.show()
```

```
C:\Users\disha\AppData\Local\Temp\ipykernel_4684\3179312453.py:8: UserWarning: Mat
plotlib is currently using module://matplotlib_inline.backend_inline, which is a n
on-GUI backend, so cannot show the figure.
  fig.show()
```

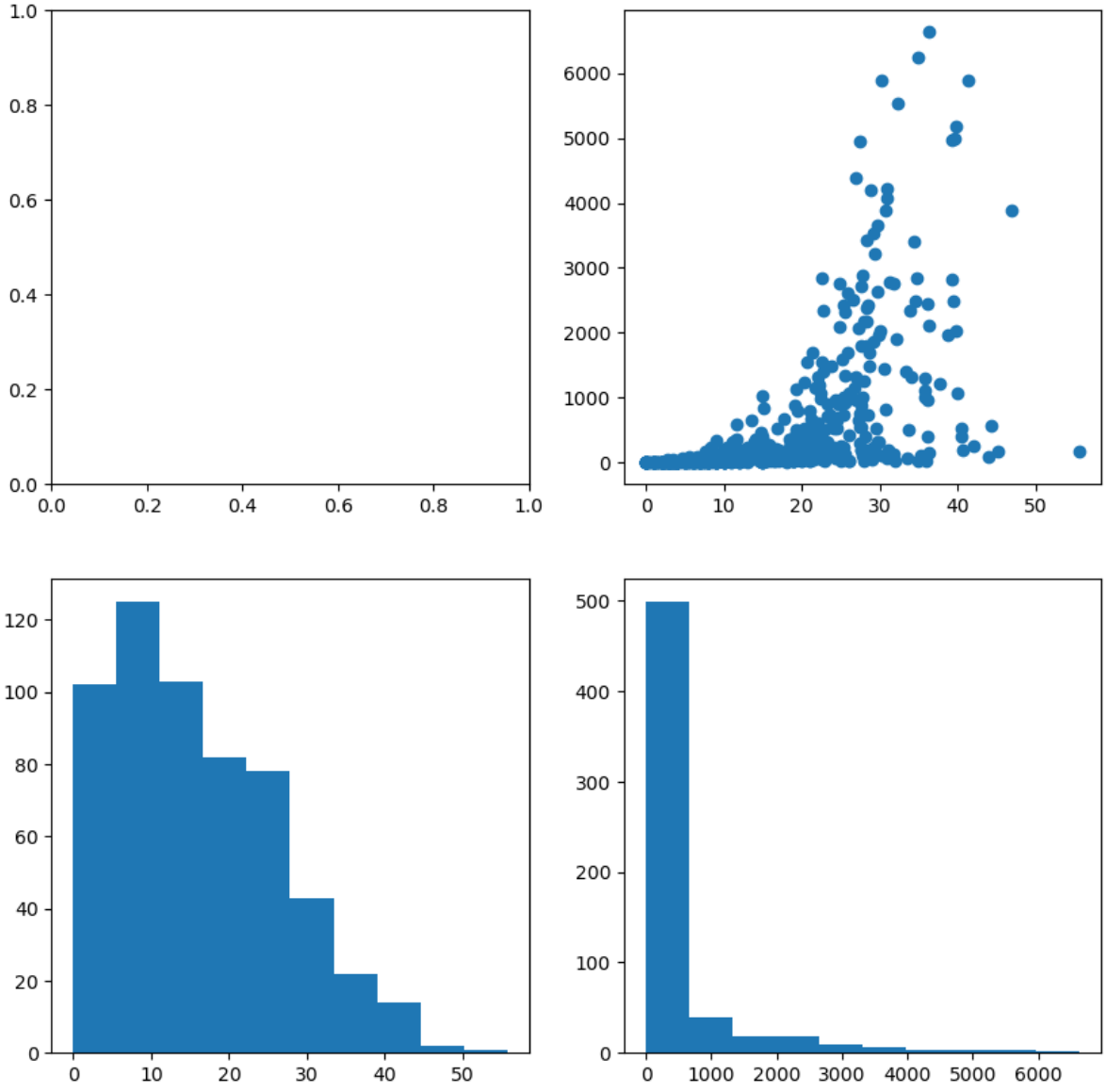


In [ ]:
```
ig, ax = plt.subplots(nrows=2,ncols=1,sharex=True,figsize=(10,6))

ax[0].scatter(batters['avg'],batters['strike_rate'],color='red')
ax[1].scatter(batters['avg'],batters['runs'])

ax[0].set_title('Avg Vs Strike Rate')
ax[0].set_ylabel('Strike Rate')


ax[1].set_title('Avg Vs Runs')
ax[1].set_ylabel('Runs')
ax[1].set_xlabel('Avg')
```

Out[ ]:
```
Text(0.5, 0, 'Avg')
```

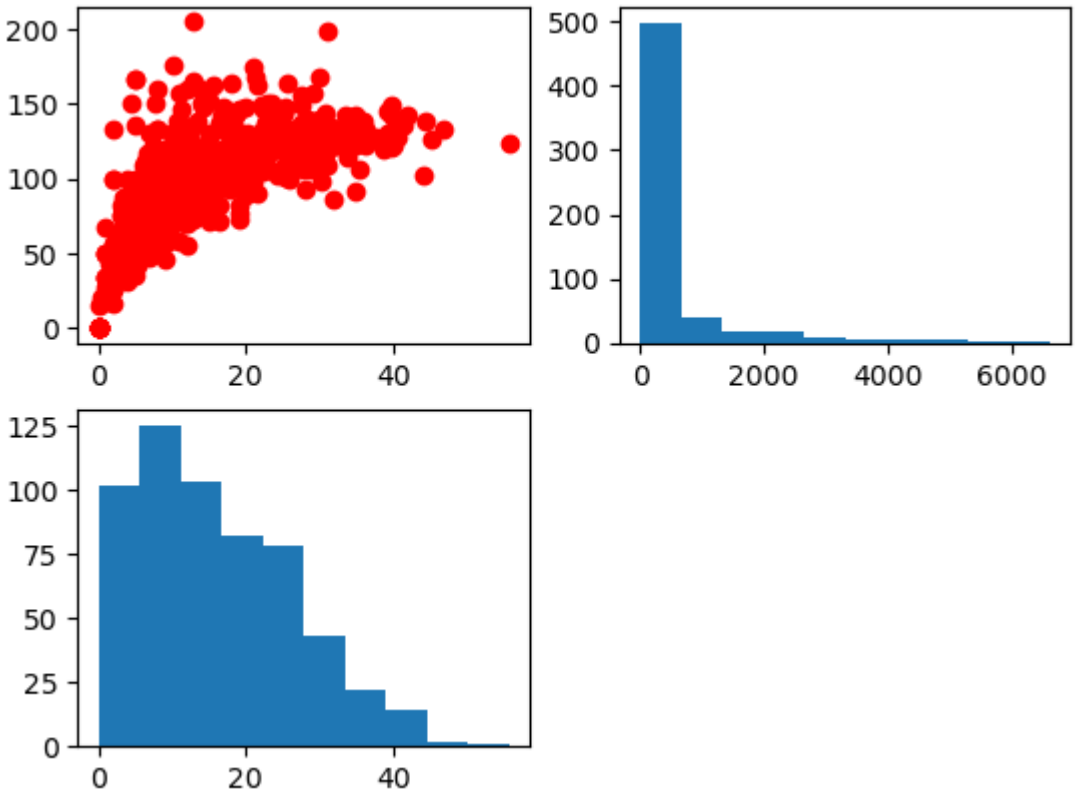
In [ ]:
```
fig, ax = plt.subplots(nrows=2,ncols=2,figsize=(10,10))

ax[0,0]
ax[0,1].scatter(batters['avg'],batters['runs'])
ax[1,0].hist(batters['avg'])
ax[1,1].hist(batters['runs'])
```

Out[ ]:
```
(array([499.,  40.,  19.,  19.,   9.,   6.,   4.,   4.,   3.,   2.]),
 array([   0. ,  663.4, 1326.8, 1990.2, 2653.6, 3317. , 3980.4, 4643.8,
        5307.2, 5970.6, 6634. ]),
 <BarContainer object of 10 artists>)
```



In [ ]:
```
fig = plt.figure()

ax1 = fig.add_subplot(2,2,1)
ax1.scatter(batters['avg'],batters['strike_rate'],color='red')

ax2 = fig.add_subplot(2,2,2)
ax2.hist(batters['runs'])

ax3 = fig.add_subplot(2,2,3)
ax3.hist(batters['avg'])
```

Out[ ]:
```
(array([102., 125., 103.,  82.,  78.,  43.,  22.,  14.,   2.,   1.]),
 array([ 0.        ,  5.56666667, 11.13333333, 16.7       , 22.26666667,
        27.83333333, 33.4       , 38.96666667, 44.53333333, 50.1       ,
        55.66666667]),
 <BarContainer object of 10 artists>)
```


200
150
100
50
0
0
20
40
500
400
300
200
100
0
0
2000
4000
6000
125
100
75
50
25
0
0
20
40

Advanced Matplotlib(part-2)

3D scatter Plot

- A 3D scatter plot is used to represent data points in a three-dimensional space.

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

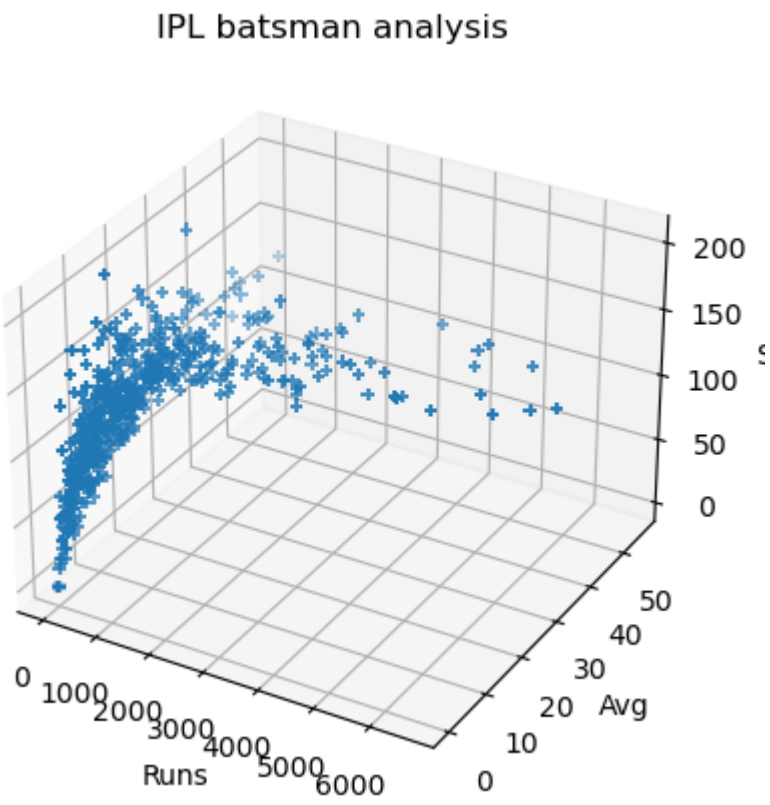
In [ ]:
```
batters = pd.read_csv('Data\Day49\Batter.csv')
batters.head()
```

Out[ ]:

	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 [ ]:
```
fig = plt.figure()

ax = plt.subplot(projection='3d')

ax.scatter3D(batters['runs'],batters['avg'],batters['strike_rate'],marker='+')
ax.set_title('IPL batsman analysis')

ax.set_xlabel('Runs')
ax.set_ylabel('Avg')
ax.set_zlabel('SR')
```

Out[ ]:
```
Text(0.5, 0, 'SR')
```



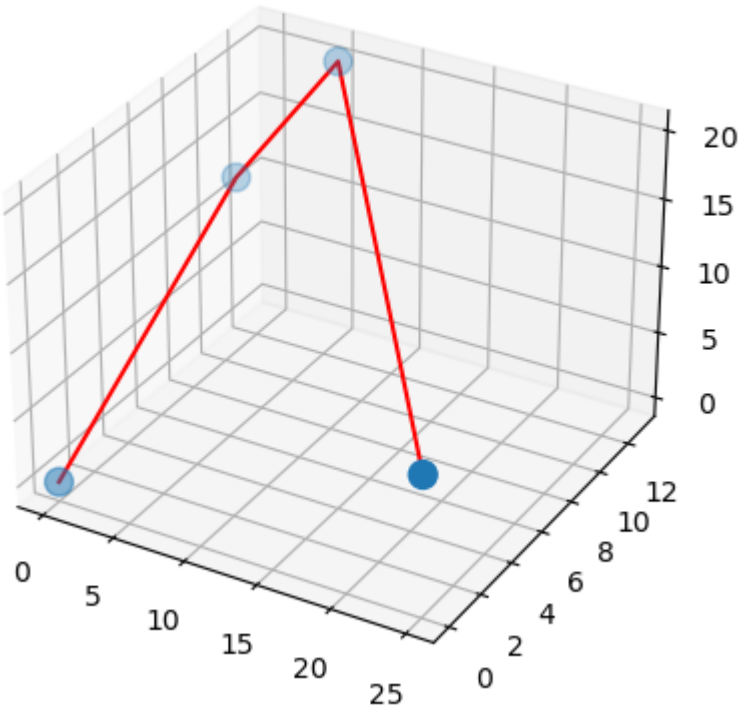
- In the example, you created a 3D scatter plot to analyze IPL batsmen based on runs, average (avg), and strike rate (SR).
- The ax.scatter3D function was used to create the plot, where the three variables were mapped to the x, y, and z axes.

3D Line Plot

- A 3D line plot represents data as a line in three-dimensional space.

In [ ]:
```
x = [0,1,5,25]
y = [0,10,13,0]
z = [0,13,20,9]

fig = plt.figure()

ax = plt.subplot(projection='3d')

ax.scatter3D(x,y,z,s=[100,100,100,100])
ax.plot3D(x,y,z,color='red')
```

Out[ ]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x23ec4988340>]
```



- In the given example, you created a 3D line plot with three sets of data points represented by lists x, y, and z.
- The ax.plot3D function was used to create the line plot.

3D Surface Plots

- 3D surface plots are used to visualize functions of two variables as surfaces in three-dimensional space.

In [ ]:
```
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)
```

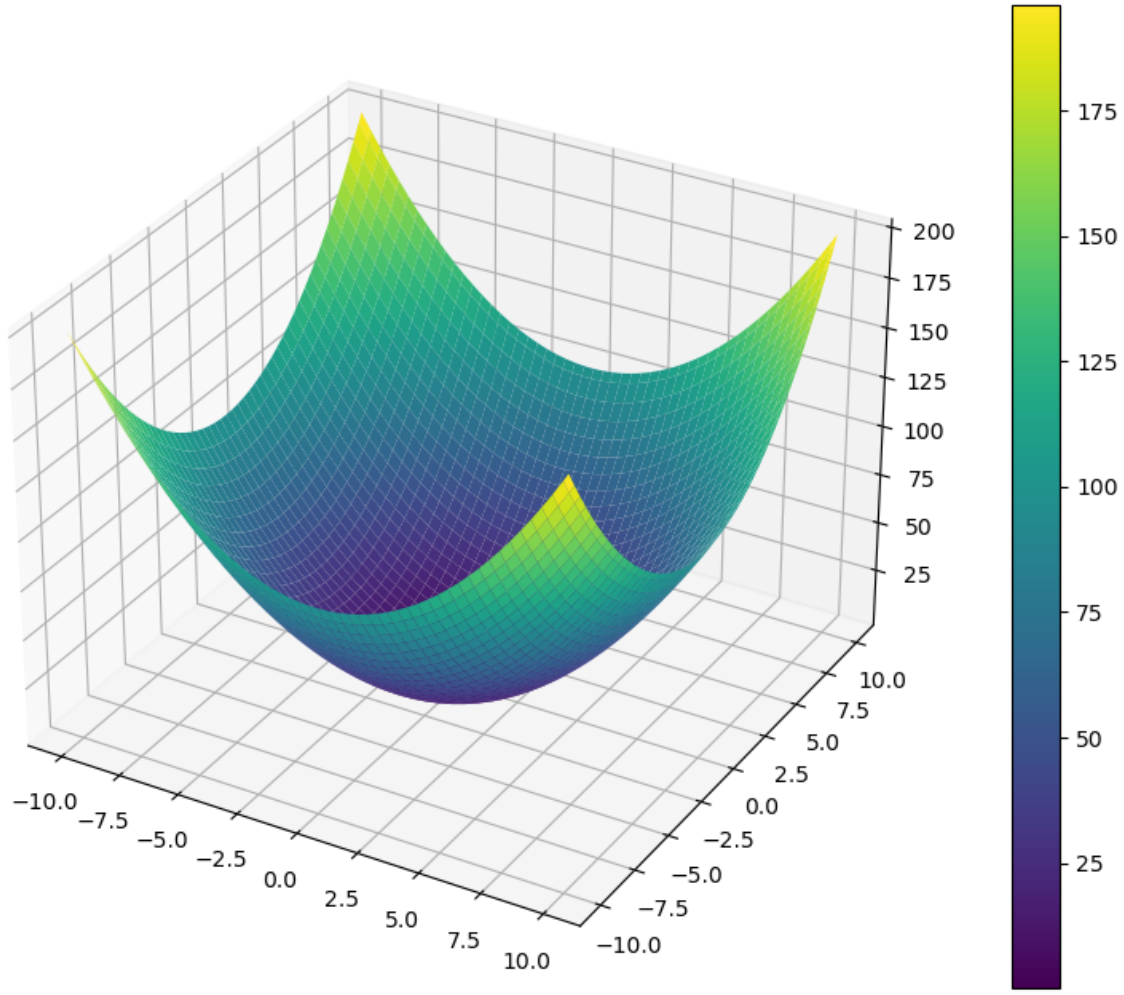
In [ ]:
```
xx, yy = np.meshgrid(x,y)
```

In [ ]:
```
z = xx**2 + yy**2
z.shape
```

Out[ ]:
```
(100, 100)
```

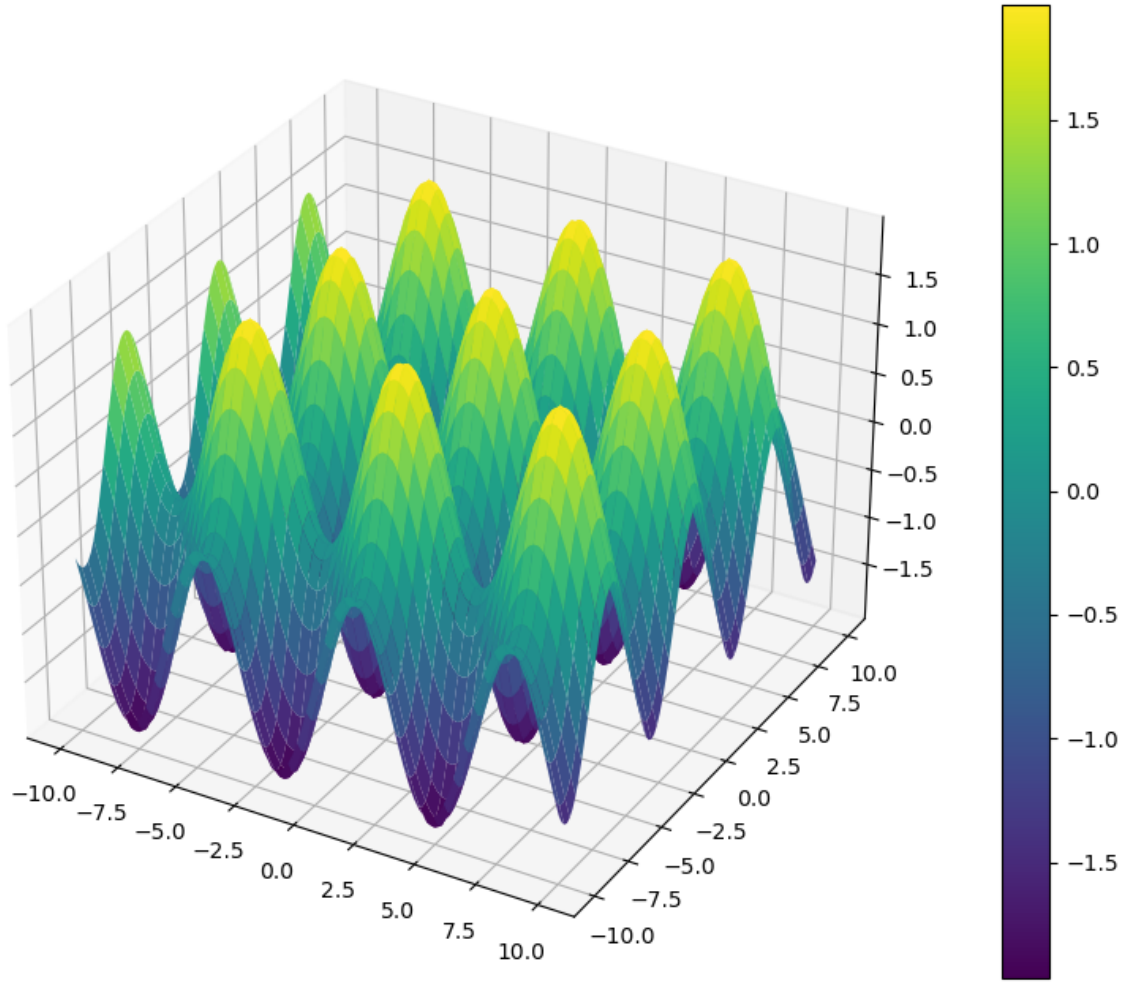
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec66ca9a0>
```



In [ ]:
```
z = np.sin(xx) + np.cos(yy)

fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec33aa520>
```



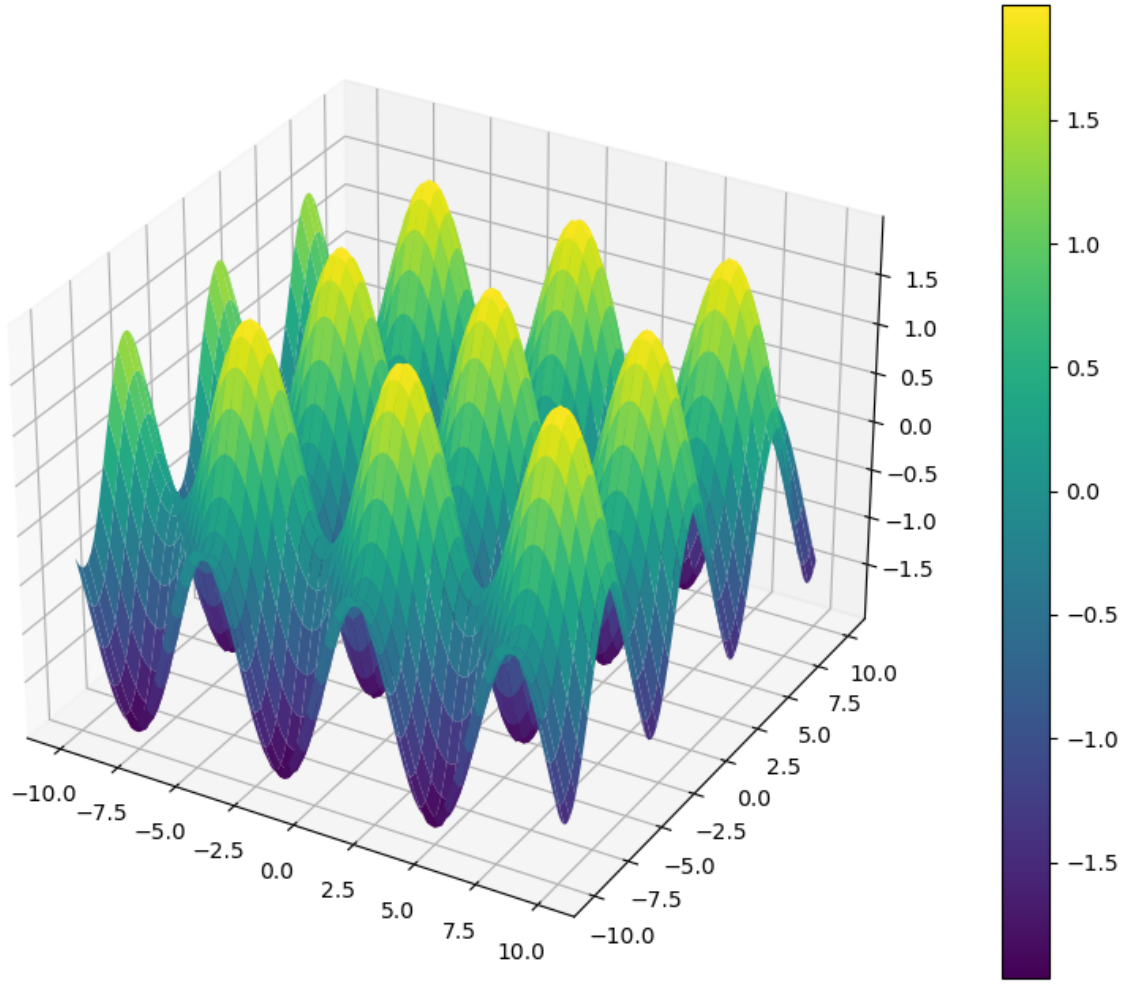
surface plot using the ax.plot_surface function. In First example, you plotted a parabolic surface, and in Seound, you plotted a surface with sine and cosine functions.

Contour Plots

- Contour plots are used to visualize 3D data in 2D, representing data as contours on a 2D plane.

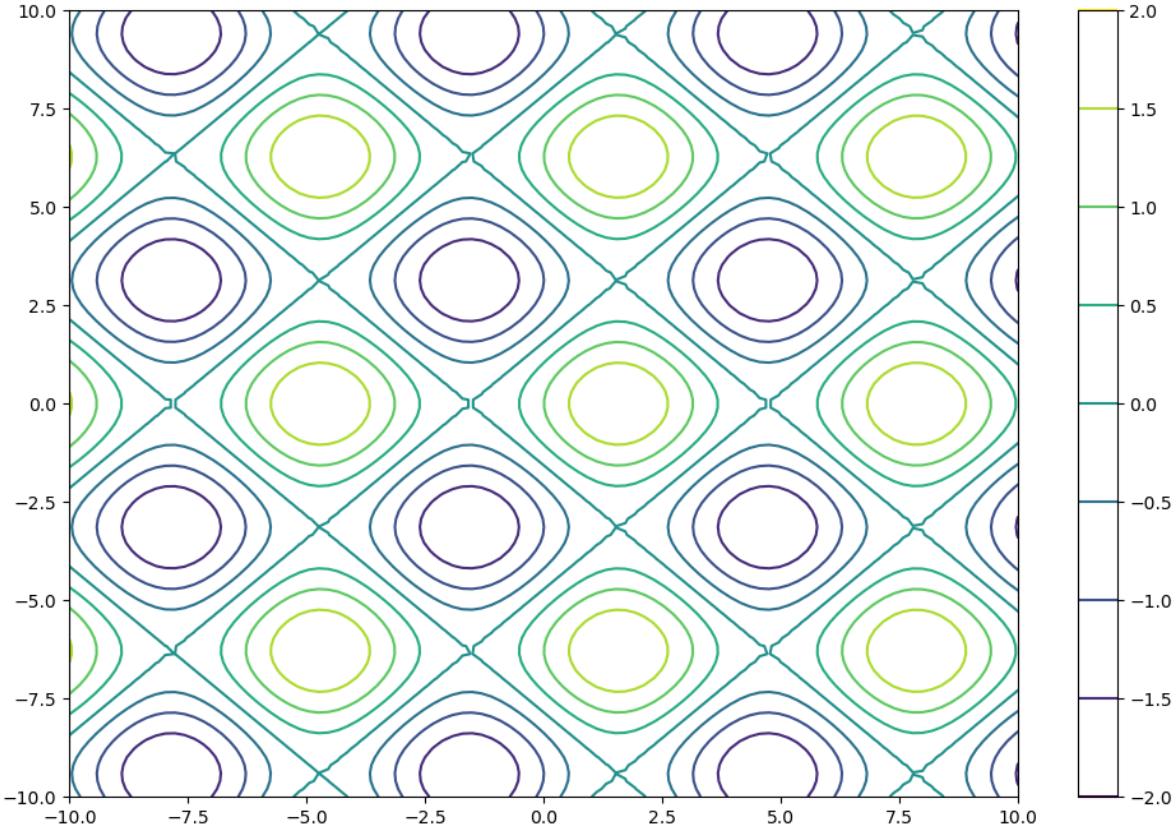
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec616e0d0>
```



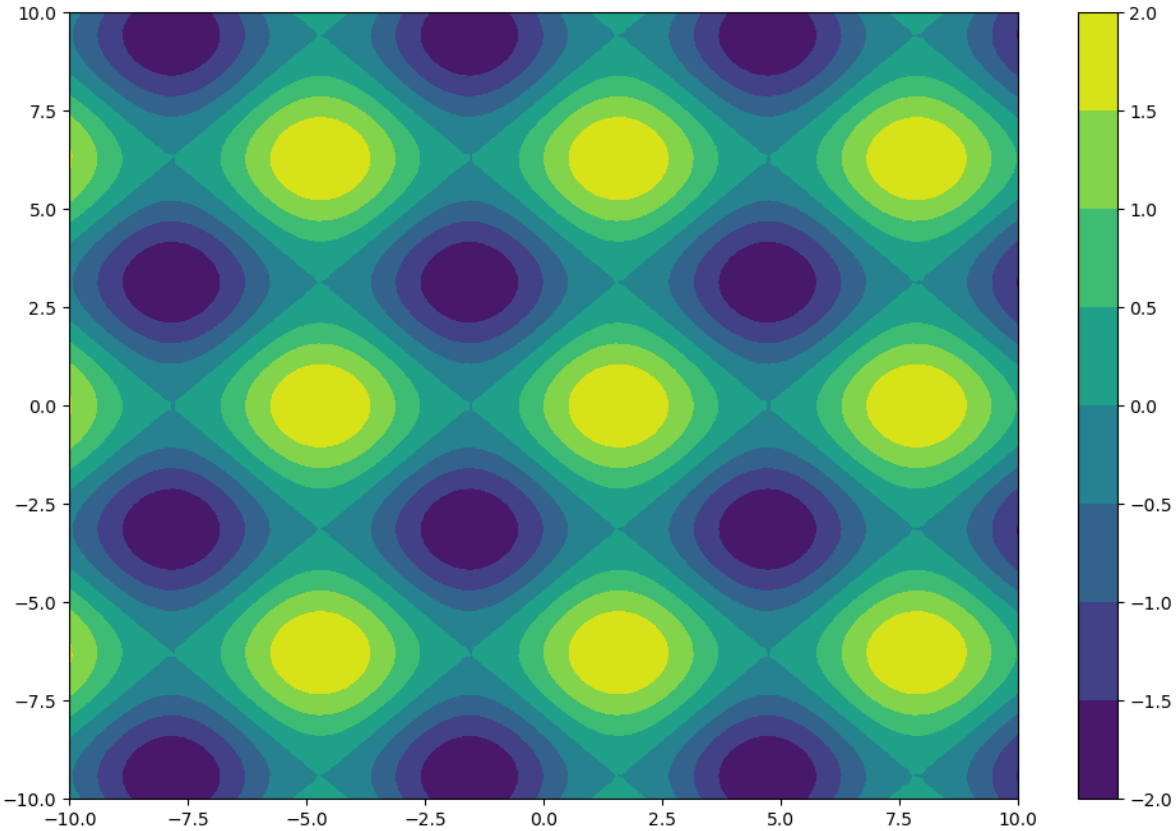
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contour(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec56e4be0>
```



In [ ]:
```
z = np.sin(xx) + np.cos(yy)

fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contourf(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

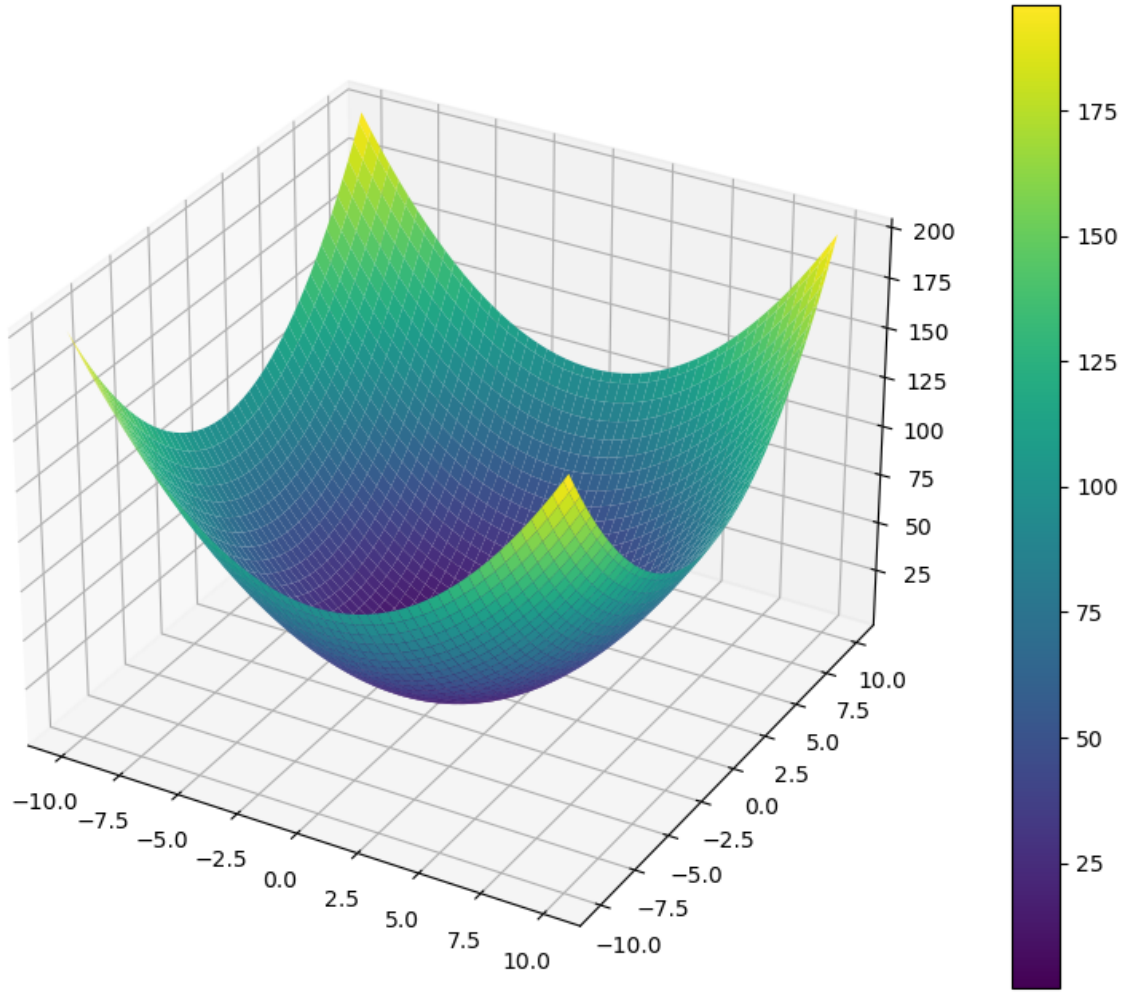
Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec8865f40>
```



You created both filled contour plots (ax.contourf) and contour line plots (ax.contour) in 2D space. These plots are useful for representing functions over a grid.

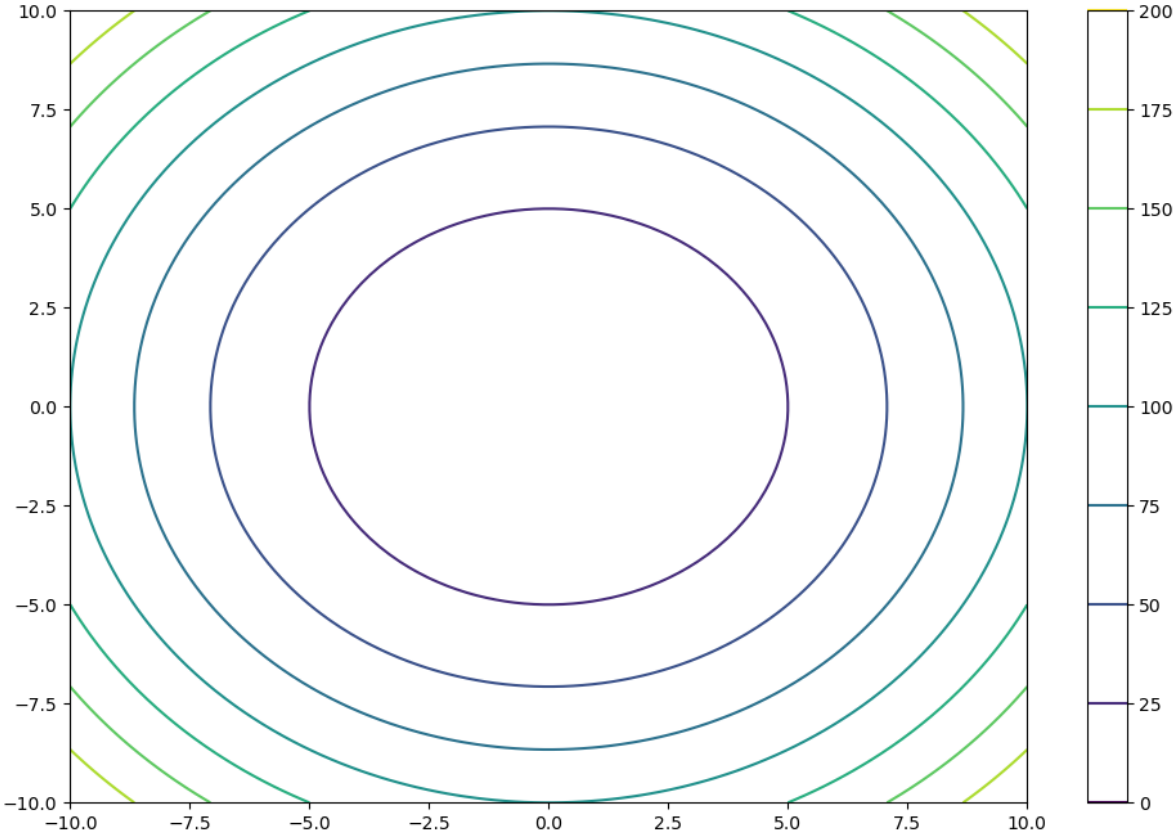
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot(projection='3d')

p = ax.plot_surface(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec7b7ca00>
```



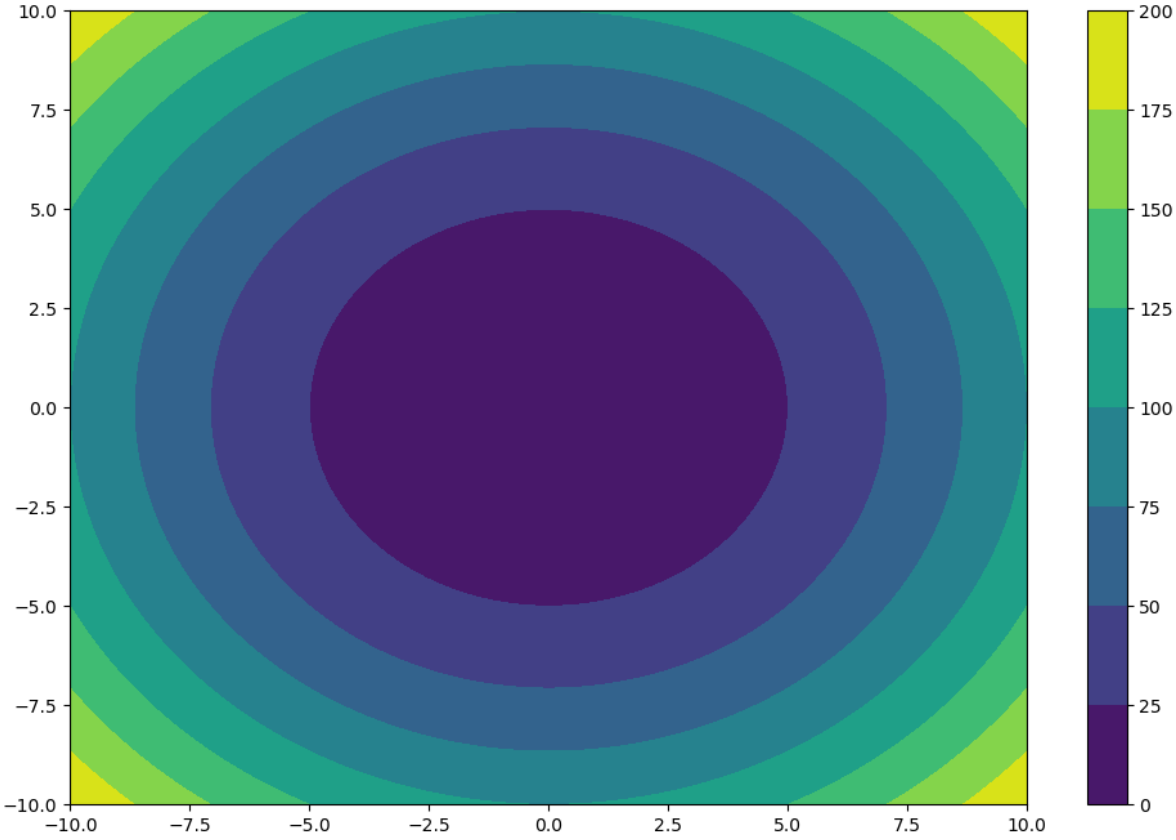
In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contour(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec7c698e0>
```



In [ ]:
```
fig = plt.figure(figsize=(12,8))

ax = plt.subplot()

p = ax.contourf(xx,yy,z,cmap='viridis')
fig.colorbar(p)
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec7ed9700>
```



Heatmap

A heatmap is a graphical representation of data in a 2D grid, where individual values are represented as colors.

In [ ]:
```
delivery = pd.read_csv('Data\Day50\IPL_Ball_by_Ball_2008_2022.csv')
delivery.head()
```

Out[ ]:

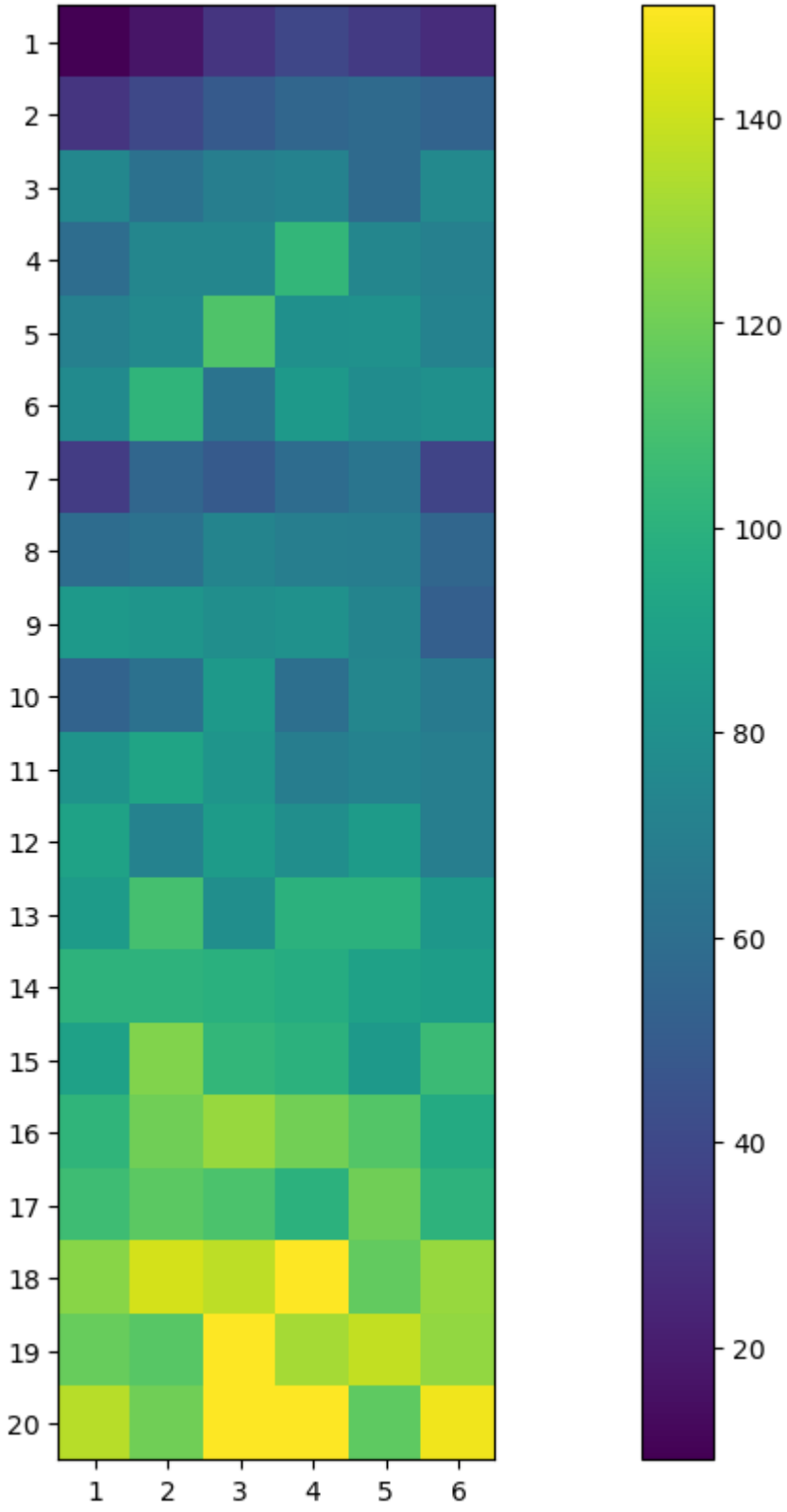
	ID	innings	overs	ballnumber	batter	bowler	non-striker	extra_type	batsman_run	ex
0	1312200	1	0	1	YBK Jaiswal	Mohammed Shami	JC Buttler	NaN	0	
1	1312200	1	0	2	YBK Jaiswal	Mohammed Shami	JC Buttler	legbyes	0	
2	1312200	1	0	3	JC Buttler	Mohammed Shami	YBK Jaiswal	NaN	1	
3	1312200	1	0	4	YBK Jaiswal	Mohammed Shami	JC Buttler	NaN	0	
4	1312200	1	0	5	YBK Jaiswal	Mohammed Shami	JC Buttler	NaN	0	

In [ ]:
```
temp_df = delivery[(delivery['ballnumber'].isin([1,2,3,4,5,6])) & (delivery['batsma
```

In [ ]:
```
grid = temp_df.pivot_table(index='overs',columns='ballnumber',values='batsman_run',
```

In [ ]:
```
plt.figure(figsize=(20,10))
plt.imshow(grid)
plt.yticks(delivery['overs'].unique(), list(range(1,21)))
plt.xticks(np.arange(0,6), list(range(1,7)))
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x23ec9384820>
```



- In the given example, we used the `imshow` function to create a heatmap of IPL deliveries.
- The grid represented ball-by-ball data with the number of sixes (batsman_run=6) in each over and ball number.
- Heatmaps are effective for visualizing patterns and trends in large datasets.

These techniques provide powerful tools for visualizing complex data in three dimensions and for representing large datasets effectively. Each type of plot is suitable for different types of data and can help in gaining insights from the data.

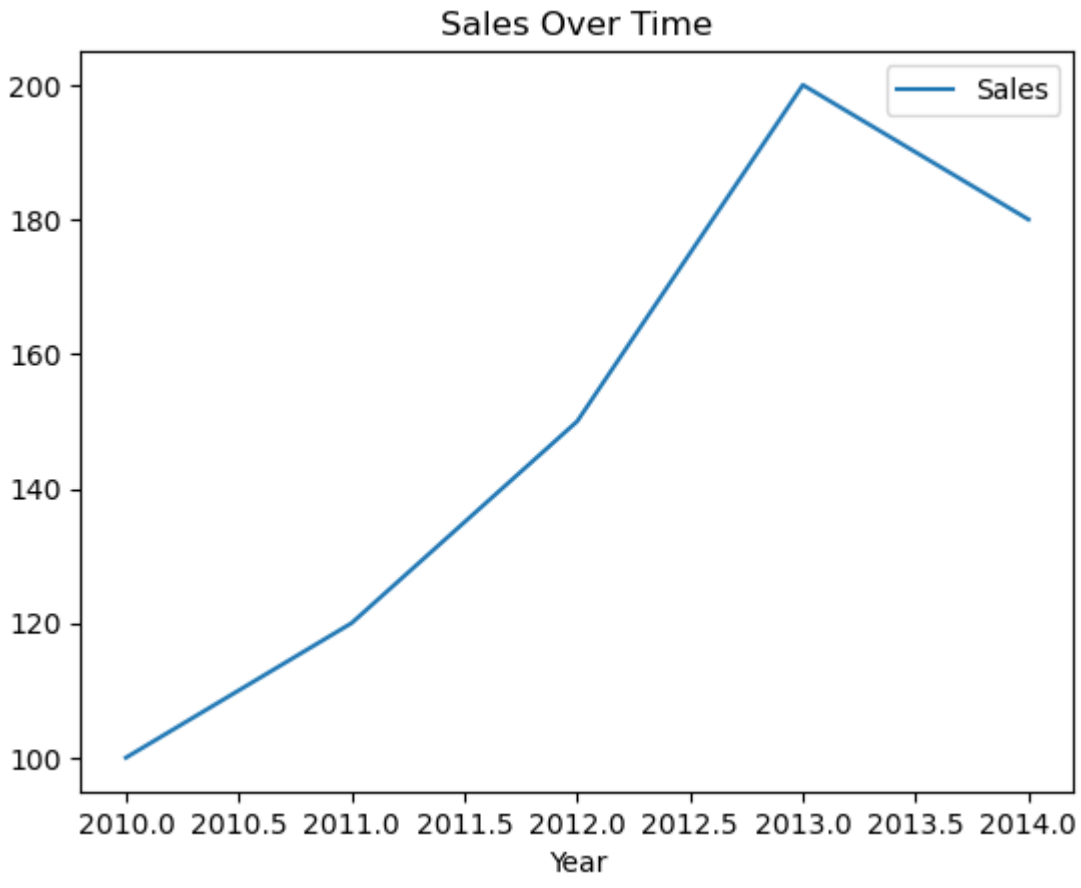
Visualization with the help of `Pandas Plot` Function

Pandas is a popular Python library for data manipulation and analysis, and it provides a convenient way to create basic visualizations using its built-in `plot` function. The `plot` function is primarily used for creating simple plots and charts directly from Pandas DataFrame and Series objects, without the need for importing additional libraries like Matplotlib. It's a quick and easy way to explore your data visually. Here's how you can use the `plot` function to create various types of visualizations:

1. Line Plot

In [ ]:
```
import pandas as pd

# Create a DataFrame
data = {'Year': [2010, 2011, 2012, 2013, 2014],
        'Sales': [100, 120, 150, 200, 180]}
df = pd.DataFrame(data)

# Create a line plot
df.plot(x='Year', y='Sales', title='Sales Over Time')
```

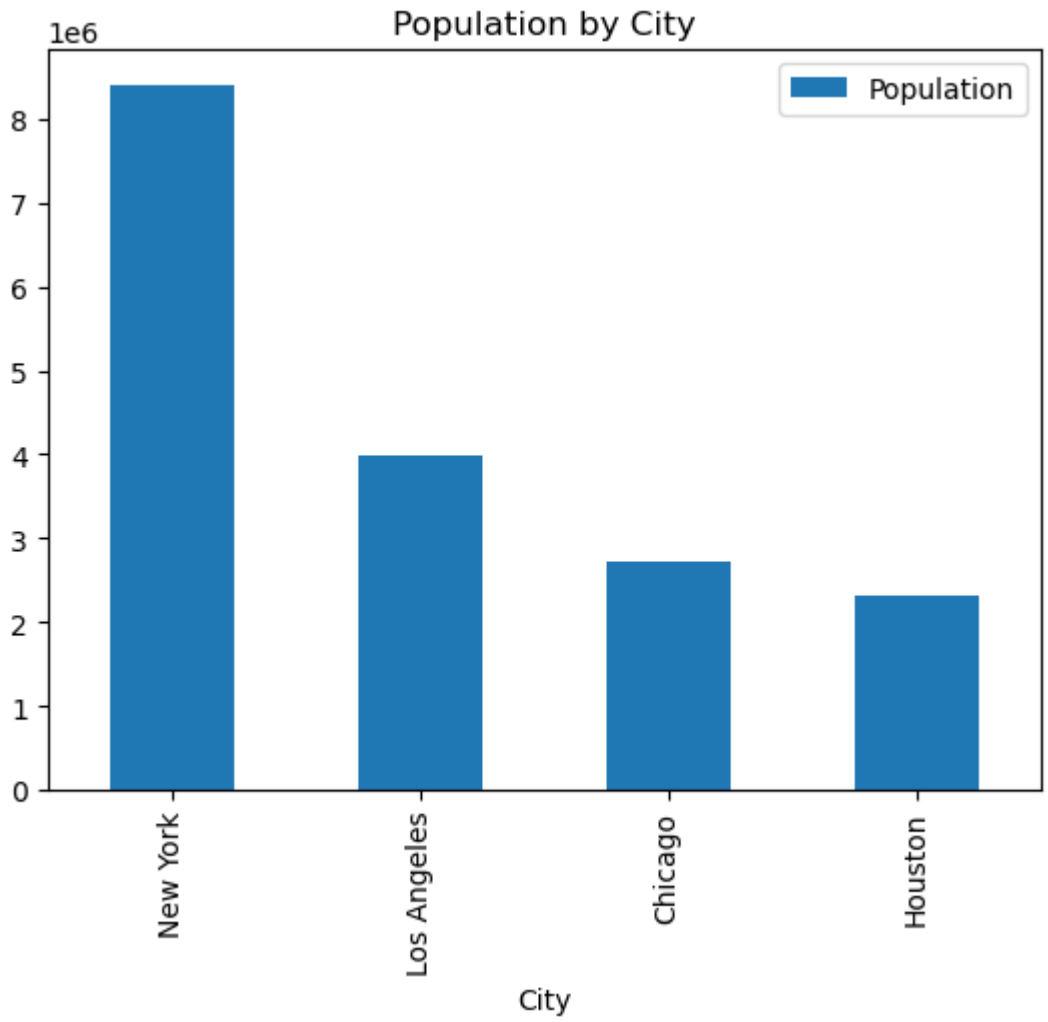
Out[ ]:
```
<Axes: title={'center': 'Sales Over Time'}, xlabel='Year'>
```



2. Bar Plot:

In [ ]:
```
import pandas as pd

# Create a DataFrame
data = {'City': ['New York', 'Los Angeles', 'Chicago', 'Houston'],
        'Population': [8398748, 3980408, 2716000, 2326006]}
df = pd.DataFrame(data)

# Create a bar plot
df.plot(x='City', y='Population', kind='bar', title='Population by City')
```

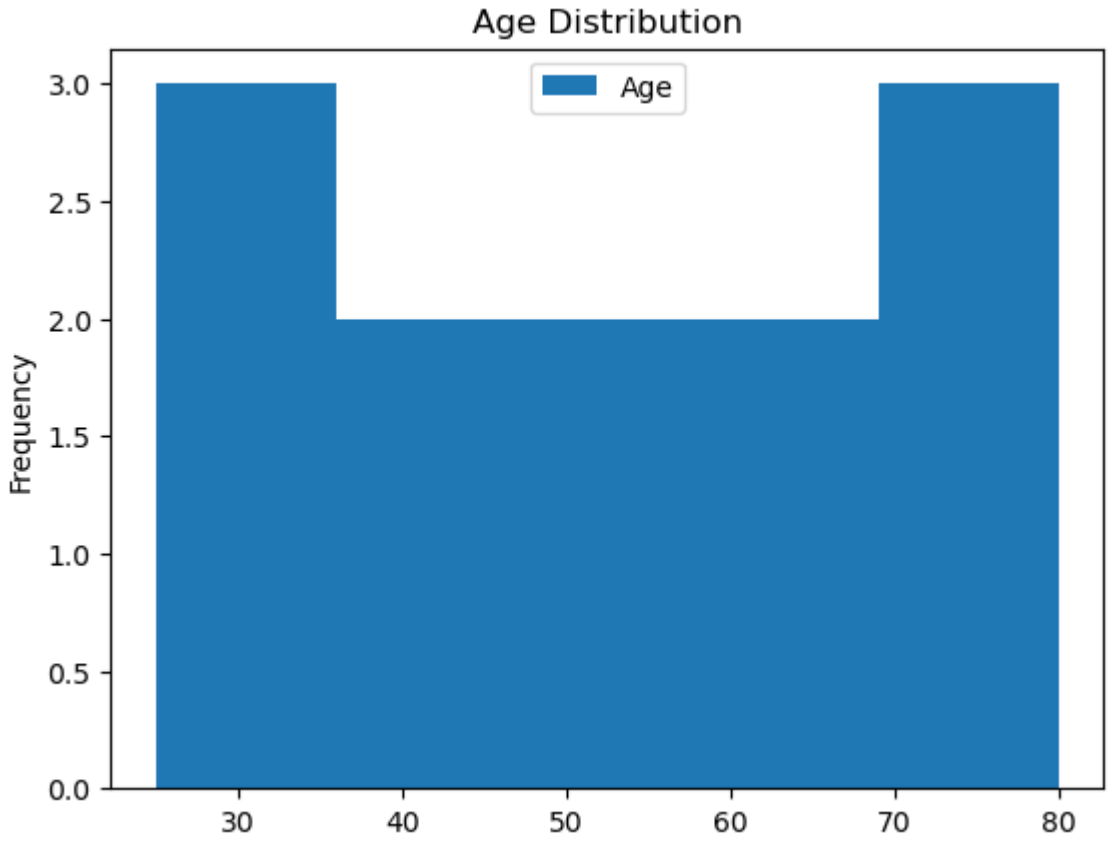
Out[ ]:
```
<Axes: title={'center': 'Population by City'}, xlabel='City'>
```



3. Histogram:

In [ ]:
```
import pandas as pd

# Create a DataFrame
data = {'Age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80]}
df = pd.DataFrame(data)

# Create a histogram
df.plot(y='Age', kind='hist', bins=5, title='Age Distribution')
```

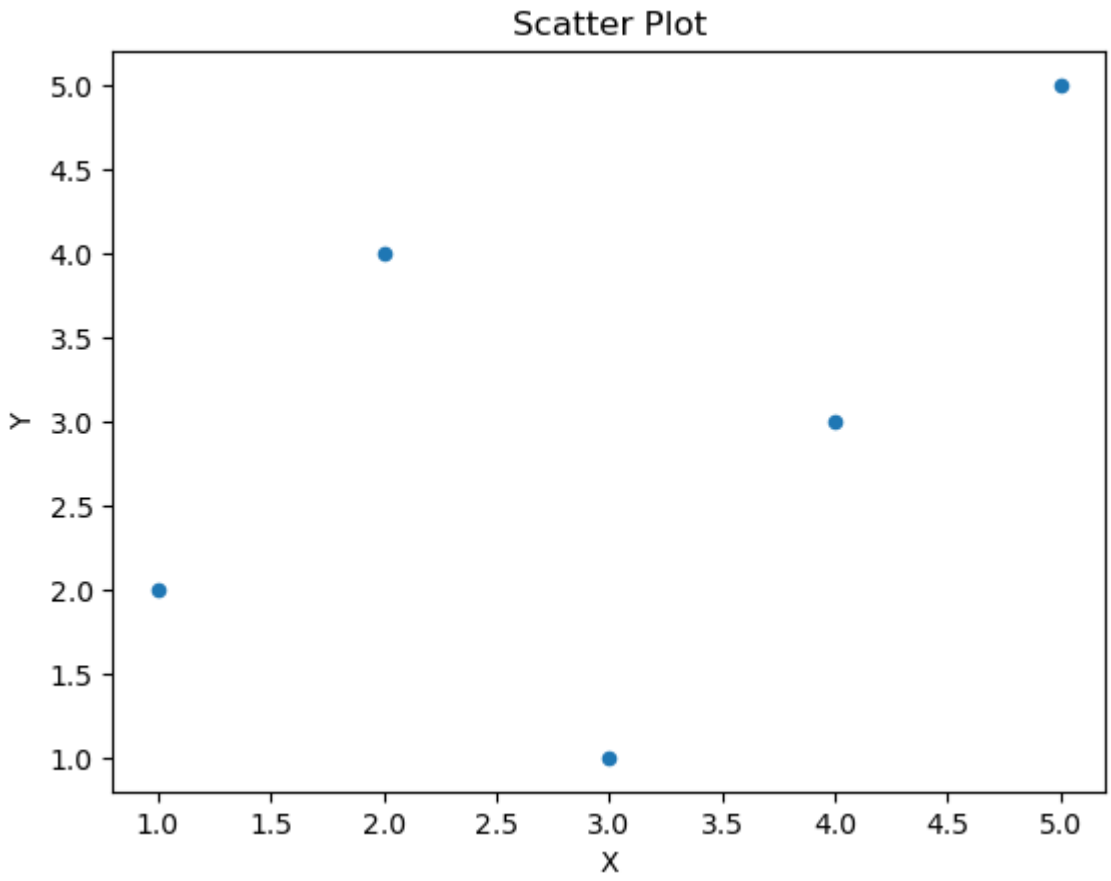
Out[ ]:
```
<Axes: title={'center': 'Age Distribution'}, ylabel='Frequency'>
```



4. Scatter Plot:

In [ ]:
```
import pandas as pd

# Create a DataFrame
data = {'X': [1, 2, 3, 4, 5],
        'Y': [2, 4, 1, 3, 5]}
df = pd.DataFrame(data)

# Create a scatter plot
df.plot(x='X', y='Y', kind='scatter', title='Scatter Plot')
```

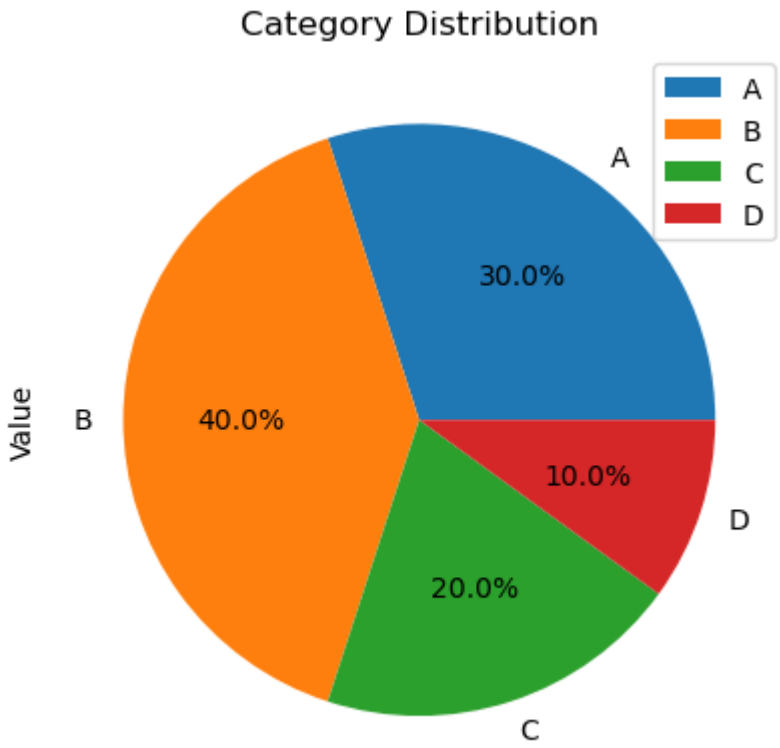
Out[ ]:
```
<Axes: title={'center': 'Scatter Plot'}, xlabel='X', ylabel='Y'>
```



5. Pie Chart:

In [ ]:
```
import pandas as pd

# Create a DataFrame
data = {'Category': ['A', 'B', 'C', 'D'],
        'Value': [30, 40, 20, 10]}
df = pd.DataFrame(data)

# Create a pie chart
df.plot(y='Value', kind='pie', labels=df['Category'], autopct='%1.1f%%', title='Cat
```

Out[ ]:
```
<Axes: title={'center': 'Category Distribution'}, ylabel='Value'>
```



The plot function provides various customization options for labels, titles, colors, and more, and it can handle different plot types by specifying the kind parameter. While Pandas' plot function is useful for quick and simple visualizations, more complex and customized visualizations may require using libraries like Matplotlib or Seaborn in combination with Pandas.

Seaborn is a data visualization library in Python that is based on Matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn comes with several built-in themes and color palettes to make it easy to create aesthetically pleasing visualizations.

Documentation link : https://seaborn.pydata.org/

Some key features of Seaborn include:

1. **Statistical Plots:** Seaborn simplifies the process of creating statistical plots by providing functions for common statistical visualizations such as scatter plots, line plots, bar plots, box plots, violin plots, and more.
2. **Themes and Color Palettes:** Seaborn allows you to easily customize the look of your visualizations by providing different themes and color palettes. This makes it simple to create professional-looking plots without having to manually tweak every detail.
3. **Categorical Plots:** Seaborn excels at visualizing relationships in categorical data. It has functions specifically designed for working with categorical variables, making it easy to create plots that show the distribution of data across different categories.
4. **Matrix Plots:** Seaborn provides functions for visualizing matrices of data, such as heatmaps. These can be useful for exploring relationships and patterns in large datasets.
5. **Time Series Plots:** Seaborn supports the visualization of time series data, allowing you to create informative plots for temporal analysis.

While Seaborn is built on top of Matplotlib, it abstracts away much of the complexity and provides a more concise and visually appealing syntax for creating statistical visualizations. It is a popular choice among data scientists and analysts for quickly generating exploratory data visualizations.

Seaborn offers several other plot types for various visualization needs:

1. **Categorical Plots:**
   - `barplot` : Displays the central tendency and confidence interval of a numeric variable for different levels of a categorical variable.
   - `countplot` : Shows the count of observations in each category of a categorical variable using bars.
   - `boxplot` : Illustrates the distribution of a continuous variable across different categories, emphasizing quartiles and potential outliers.

- `violinplot` : Combines aspects of a box plot and a kernel density plot, providing insights into the distribution of a variable for different categories.
- `stripplot` **and** `swarmplot` : Visualize individual data points along with a categorical distribution. `stripplot` places points on a single axis, while `swarmplot` adjusts the points to avoid overlap.

2. **Distribution Plots:**
   - `histplot` : Creates a histogram to display the distribution of a single variable.
   - `kdeplot` : Plots the Kernel Density Estimation of a univariate or bivariate dataset, providing a smoothed representation of the distribution.
   - `rugplot` : Adds small vertical lines (rug) to a plot to indicate the distribution of data points along the x or y-axis.
3. **Regression Plots:**
   - `lmplot` : Combines regression plots with FacetGrid to allow visualization of relationships between variables across multiple subplots.
   - `regplot` : Creates a scatter plot with a linear regression line fitted to the data.
4. **Time Series Plots:**
   - `tsplot` : Formerly used for time series data, it has been replaced by the more flexible `lineplot` . However, it's still available for backward compatibility.
5. **Facet Grids:**
   - `FacetGrid` : Although not a specific plot type, `FacetGrid` is a powerful tool that allows you to create a grid of subplots based on the values of one or more categorical variables. It can be used with various plot types to create a matrix of visualizations.
6. **Relational Plots:**
   - `relplot` : Combines aspects of scatter plots and line plots to visualize the relationship between two variables across different levels of a third variable. It's a flexible function that can create scatter plots, line plots, or other types of relational plots.
   - `scatterplot` : Creates a simple scatter plot to show the relationship between two variables.
   - `lineplot` : Generates line plots to depict the trend between two variables.
   - `jointplot` : Combines scatter plots for two variables along with histograms for each variable.
7. **Matrix Plots:**
   - `heatmap` : Displays a matrix where the values are represented by colors. Heatmaps are often used to visualize correlation matrices or other two-dimensional datasets.

- `clustermap` : Hierarchically clusters and orders rows and columns in a heatmap, which can be useful for exploring patterns in the data.
- `pairplot` : Creates a matrix of scatterplots for multiple pairs of variables. Diagonal elements show univariate distributions, while off-diagonal elements show bivariate relationships.
- `jointplot` **with** `kind='hex'` **or** `kind='kde'` : While commonly used for scatter plots, `jointplot` can display hexbin plots or kernel density estimation plots, providing a different perspective on the distribution of data points.
- `corrplot` : Visualizes a correlation matrix with colored cells, making it easy to identify strong and weak correlations between variables.
- `catplot` **with** `kind='point'` **or** `kind='bar'` : Allows you to create categorical plots with a matrix structure, where the rows and columns represent different categorical variables.

These functions provide a range of tools for exploring relationships in data, whether you're interested in visualizing the distribution of individual variables, the relationship between two variables, or patterns within a matrix of data.

These are just a selection of Seaborn's capabilities. The library is designed to make it easy to create a wide range of statistical graphics for data exploration and presentation. The choice of which plot to use depends on the nature of your data and the specific insights you want to extract.

In Seaborn, the concepts of "axis-level" and "figure-level" functions refer to the level at which the functions operate and the structure of the resulting plots.

1. **Axis-Level Functions:**
   - **Operate at the level of a single subplot or axis.**
   - **Produce a single plot by default.**
   - **Examples include functions like** `sns.scatterplot()` , `sns.lineplot()` , `sns.boxplot()` **, etc.**
   - **Accept the** `ax` **parameter to specify the Axes where the plot will be drawn. If not specified, a new Axes is created.**

1. **Figure-Level Functions:**
   - **Operate at the level of the entire figure, potentially creating multiple subplots.**
   - **Produce a FacetGrid or a similar object that can be used to create a grid of subplots.**
   - **Examples include functions like** `sns.relplot()` , `sns.catplot()` , `sns.pairplot()` **, etc.**
   - **Return a FacetGrid object, allowing for easy creation of subplots based on additional categorical variables.**

The choice between axis-level and figure-level functions depends on your specific needs. Axis-level functions are often more straightforward for simple plots, while figure-level functions are powerful for creating complex visualizations with multiple subplots or facets based on categorical variables.

Keep in mind that figure-level functions return objects like FacetGrid, and you can customize the resulting plots further using the methods and attributes of these objects. The documentation for each function provides details on how to use and customize the output.

1. Relational Plots

- to see the statistical relation between 2 or more variables.
- Bivariate Analysis

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

In [ ]:
```
import pandas as pd
```

In [ ]:
```
tips = sns.load_dataset('tips')
tips
```

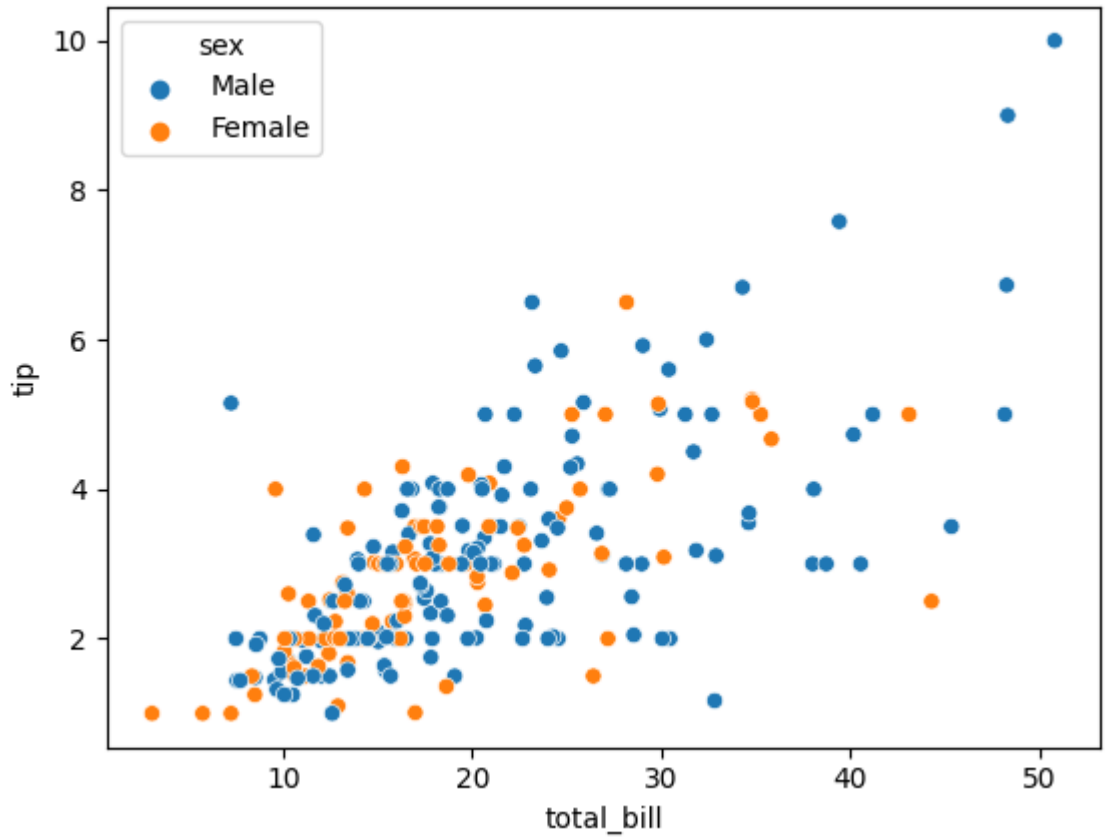
Out[ ]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

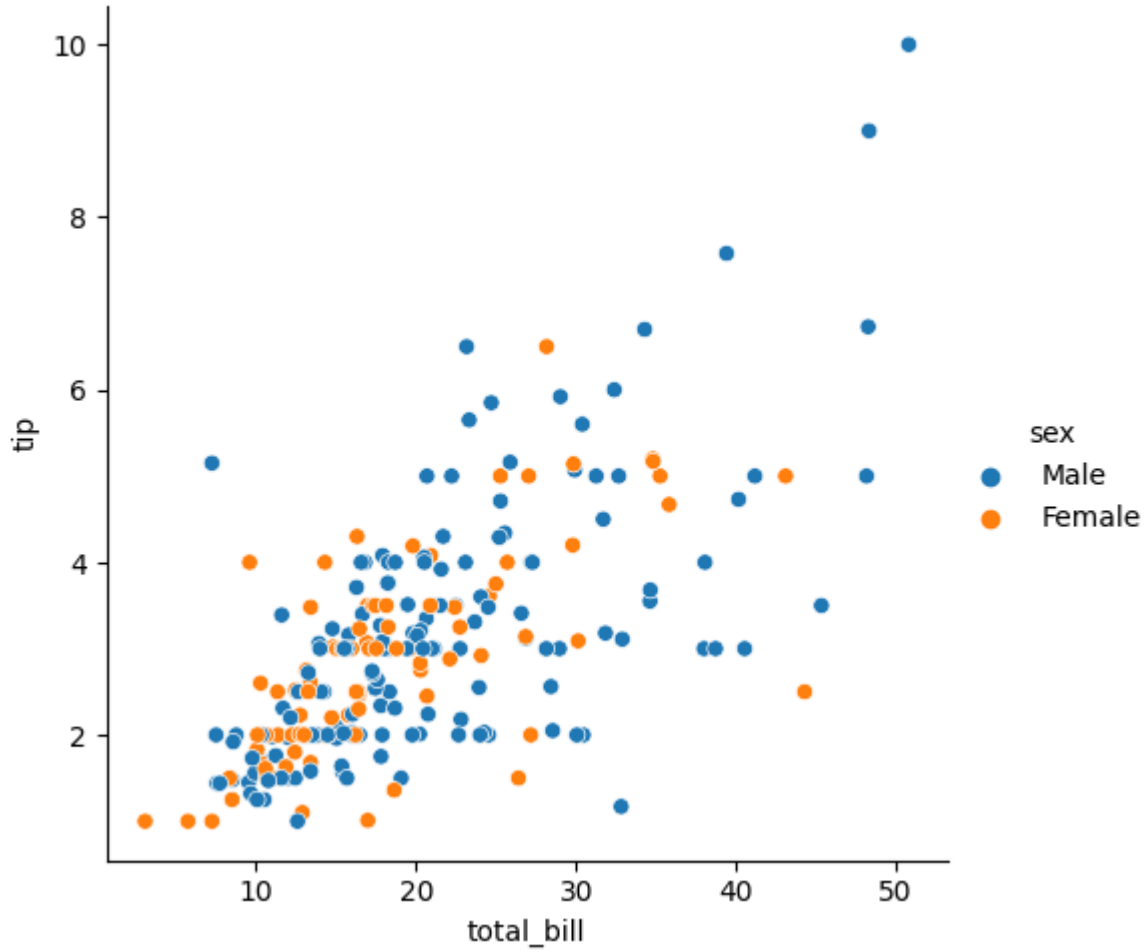
In [ ]:
```
# scatter plot -> axes level function
sns.scatterplot(data=tips, x='total_bill', y='tip',hue='sex')
```

Out[ ]:
```
<Axes: xlabel='total_bill', ylabel='tip'>
```



In [ ]:
```
# relplot -> figure level -> square shape
sns.relplot(data=tips, x='total_bill', y='tip', kind='scatter',hue='sex')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f686180b340>
```



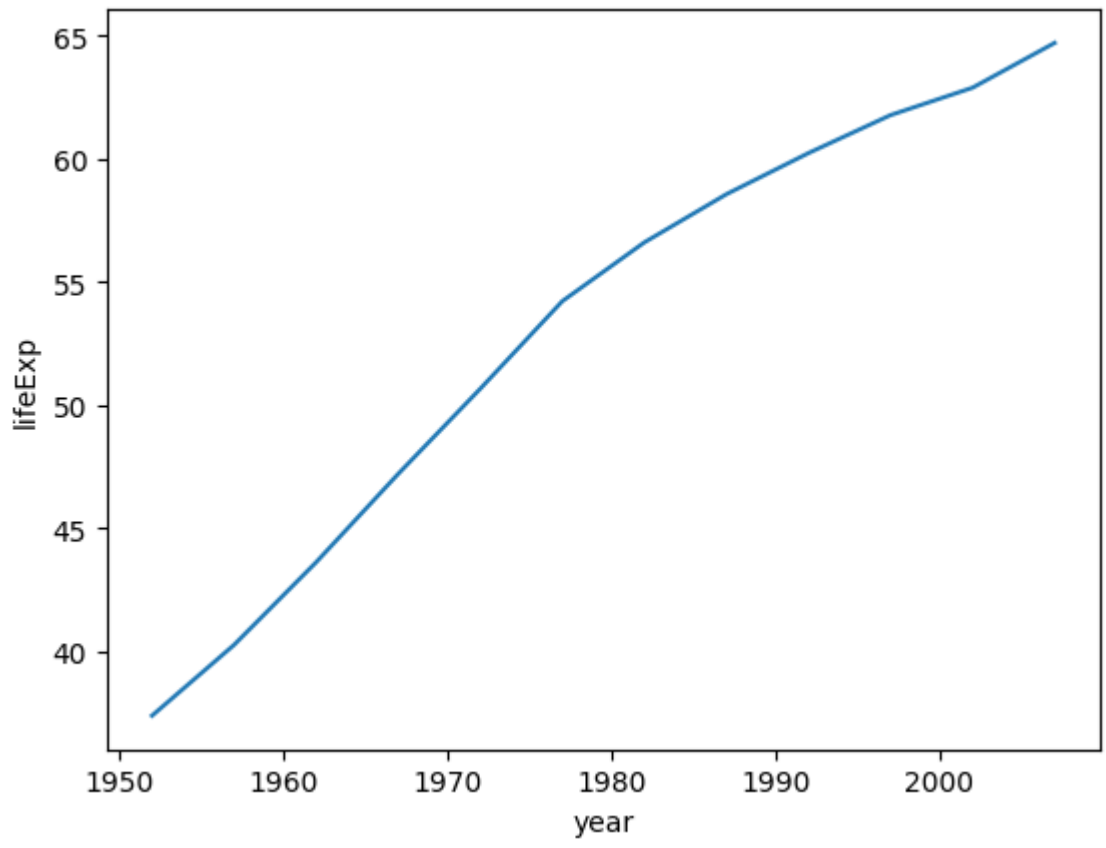
In [ ]:
```
# line plot
gap = px.data.gapminder()
temp_df = gap[gap['country'] == 'India']
temp_df
```

Out[ ]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
696	India	Asia	1952	37.373	372000000	546.565749	IND	356
697	India	Asia	1957	40.249	409000000	590.061996	IND	356
698	India	Asia	1962	43.605	454000000	658.347151	IND	356
699	India	Asia	1967	47.193	506000000	700.770611	IND	356
700	India	Asia	1972	50.651	567000000	724.032527	IND	356
701	India	Asia	1977	54.208	634000000	813.337323	IND	356
702	India	Asia	1982	56.596	708000000	855.723538	IND	356
703	India	Asia	1987	58.553	788000000	976.512676	IND	356
704	India	Asia	1992	60.223	872000000	1164.406809	IND	356
705	India	Asia	1997	61.765	959000000	1458.817442	IND	356
706	India	Asia	2002	62.879	1034172547	1746.769454	IND	356
707	India	Asia	2007	64.698	1110396331	2452.210407	IND	356

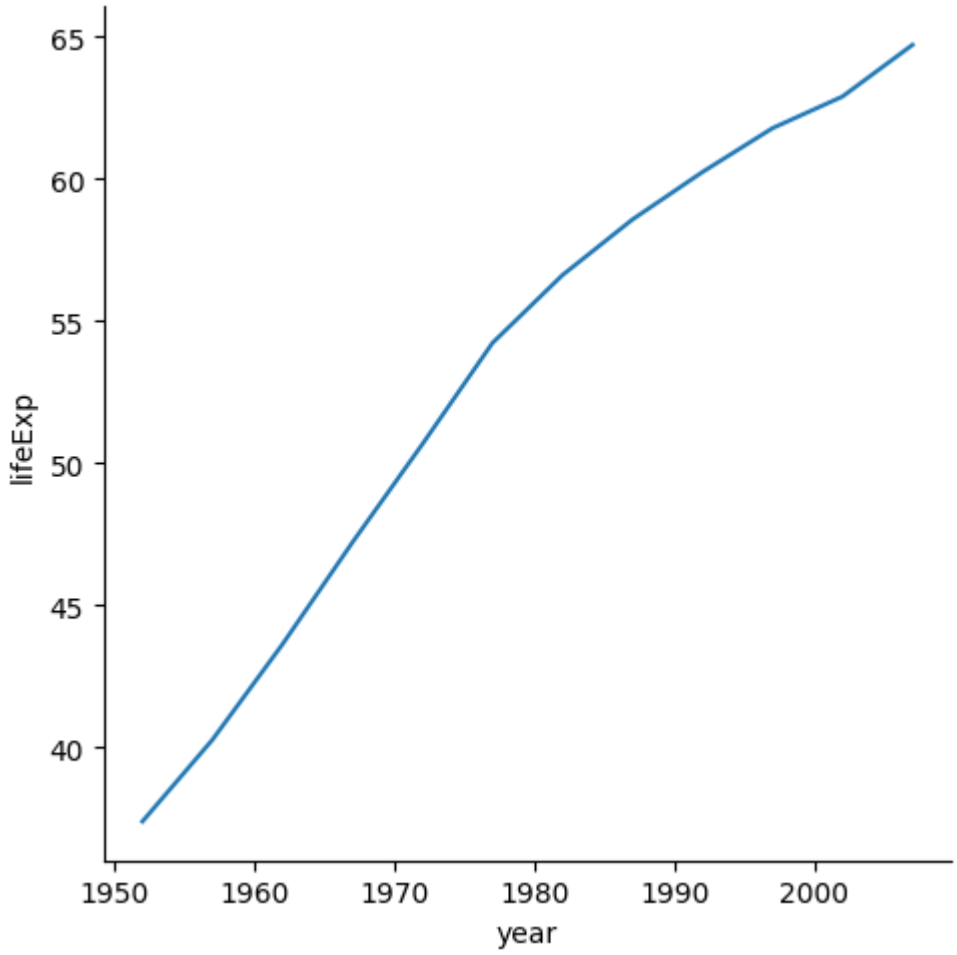
In [ ]:
```
# axes level function
sns.lineplot(data=temp_df, x='year', y='lifeExp')
```

Out[ ]:
```
<Axes: xlabel='year', ylabel='lifeExp'>
```



In [ ]:
```
# using relpplot
sns.relplot(data=temp_df, x='year', y='lifeExp', kind='line')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f686180bf70>
```



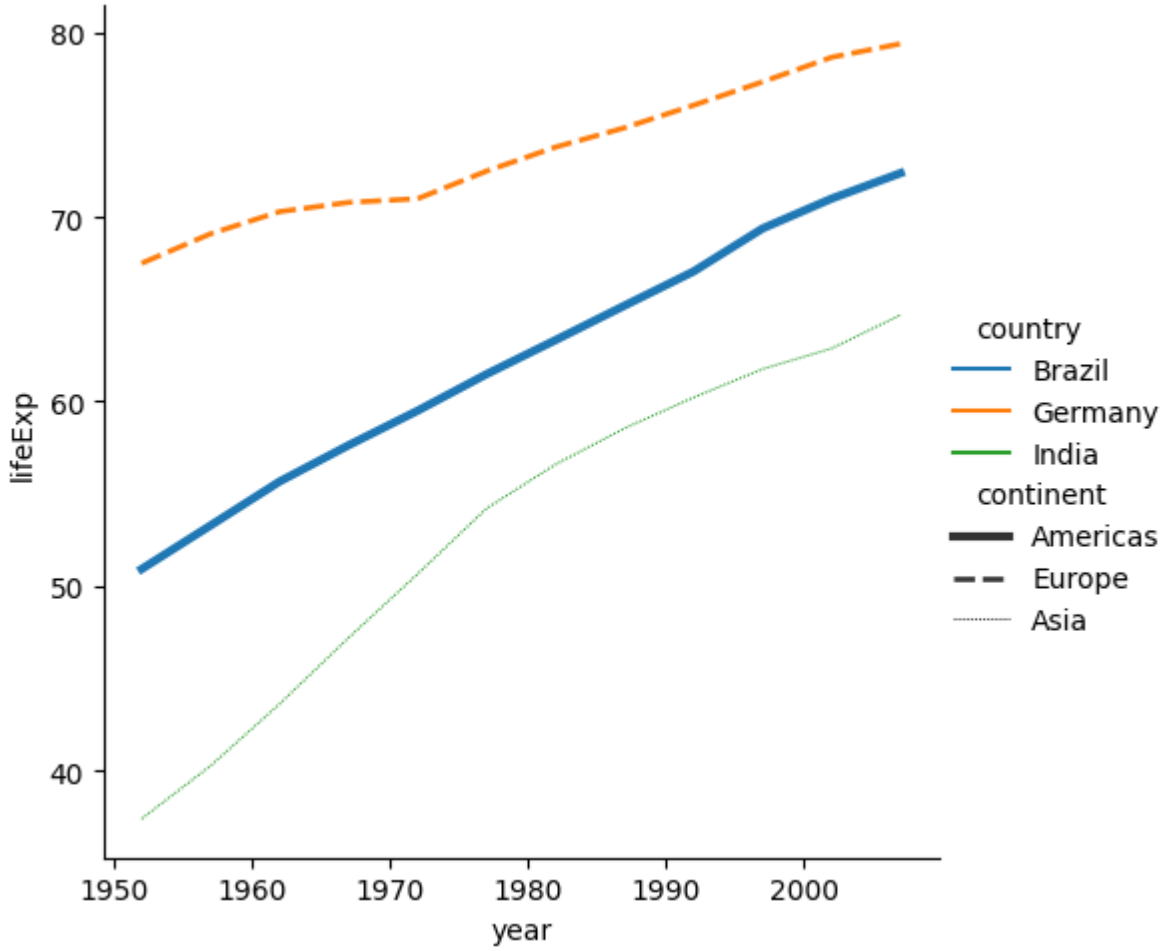
In [ ]:
```
temp_df = gap[gap['country'].isin(['India','Brazil','Germany'])]
temp_df
```

Out[ ]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
168	Brazil	Americas	1952	50.917	56602560	2108.944355	BRA	76
169	Brazil	Americas	1957	53.285	65551171	2487.365989	BRA	76
170	Brazil	Americas	1962	55.665	76039390	3336.585802	BRA	76
171	Brazil	Americas	1967	57.632	88049823	3429.864357	BRA	76
172	Brazil	Americas	1972	59.504	100840058	4985.711467	BRA	76
173	Brazil	Americas	1977	61.489	114313951	6660.118654	BRA	76
174	Brazil	Americas	1982	63.336	128962939	7030.835878	BRA	76
175	Brazil	Americas	1987	65.205	142938076	7807.095818	BRA	76
176	Brazil	Americas	1992	67.057	155975974	6950.283021	BRA	76
177	Brazil	Americas	1997	69.388	168546719	7957.980824	BRA	76
178	Brazil	Americas	2002	71.006	179914212	8131.212843	BRA	76
179	Brazil	Americas	2007	72.390	190010647	9065.800825	BRA	76
564	Germany	Europe	1952	67.500	69145952	7144.114393	DEU	276
565	Germany	Europe	1957	69.100	71019069	10187.826650	DEU	276
566	Germany	Europe	1962	70.300	73739117	12902.462910	DEU	276
567	Germany	Europe	1967	70.800	76368453	14745.625610	DEU	276
568	Germany	Europe	1972	71.000	78717088	18016.180270	DEU	276
569	Germany	Europe	1977	72.500	78160773	20512.921230	DEU	276
570	Germany	Europe	1982	73.800	78335266	22031.532740	DEU	276
571	Germany	Europe	1987	74.847	77718298	24639.185660	DEU	276
572	Germany	Europe	1992	76.070	80597764	26505.303170	DEU	276
573	Germany	Europe	1997	77.340	82011073	27788.884160	DEU	276
574	Germany	Europe	2002	78.670	82350671	30035.801980	DEU	276
575	Germany	Europe	2007	79.406	82400996	32170.374420	DEU	276
696	India	Asia	1952	37.373	372000000	546.565749	IND	356
697	India	Asia	1957	40.249	409000000	590.061996	IND	356
698	India	Asia	1962	43.605	454000000	658.347151	IND	356
699	India	Asia	1967	47.193	506000000	700.770611	IND	356
700	India	Asia	1972	50.651	567000000	724.032527	IND	356
701	India	Asia	1977	54.208	634000000	813.337323	IND	356
702	India	Asia	1982	56.596	708000000	855.723538	IND	356
703	India	Asia	1987	58.553	788000000	976.512676	IND	356
704	India	Asia	1992	60.223	872000000	1164.406809	IND	356
705	India	Asia	1997	61.765	959000000	1458.817442	IND	356
706	India	Asia	2002	62.879	1034172547	1746.769454	IND	356
707	India	Asia	2007	64.698	1110396331	2452.210407	IND	356

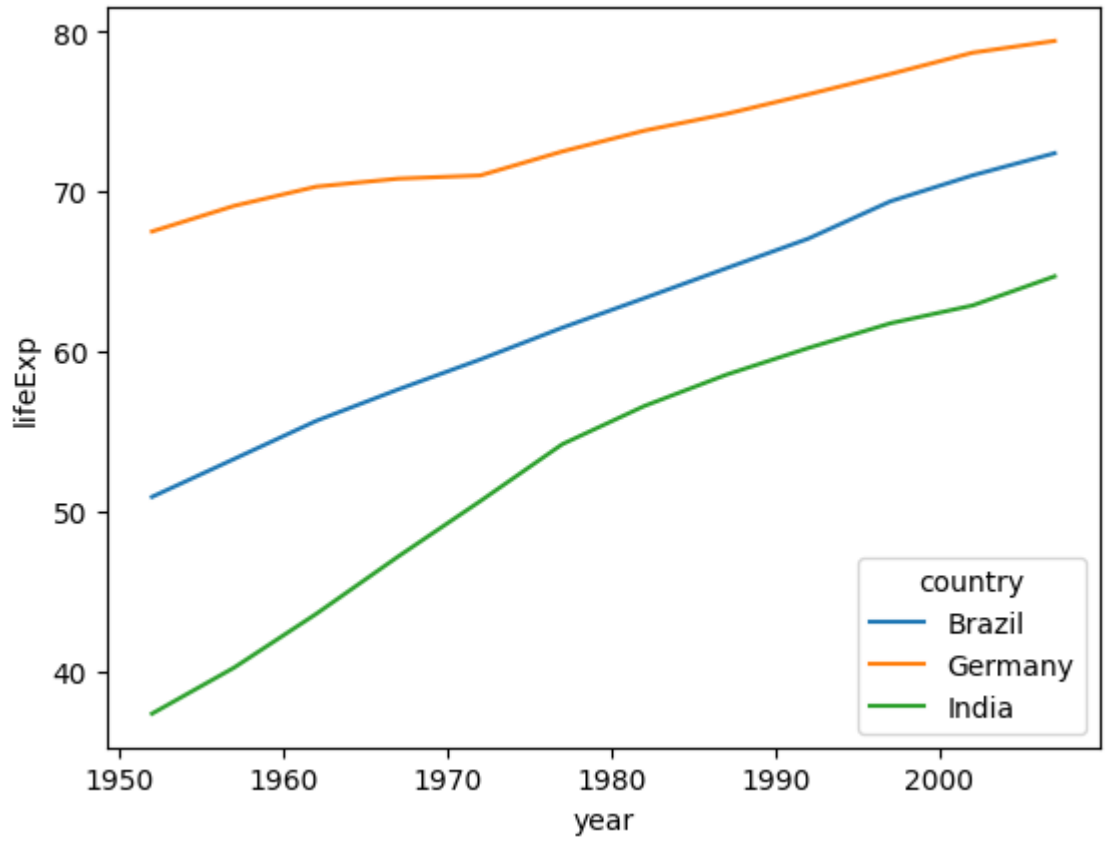
In [ ]:
```
sns.relplot(kind='line', data=temp_df, x='year', y='lifeExp', hue='country', style=
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f68616088e0>
```



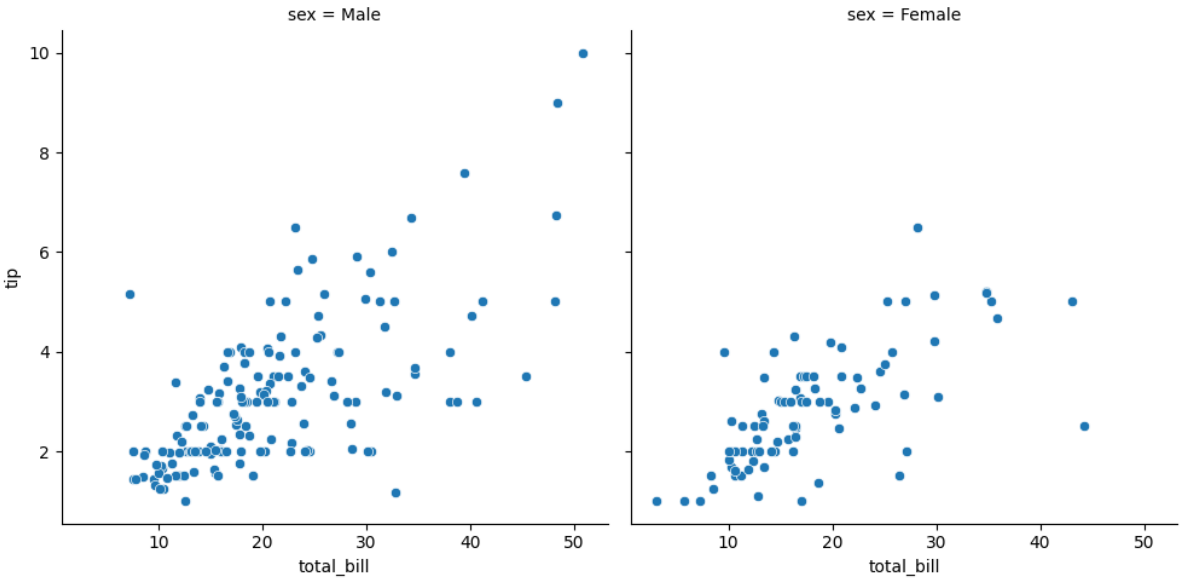
In [ ]:
```
sns.lineplot(data=temp_df, x='year', y='lifeExp', hue='country')
```

Out[ ]:
```
<Axes: xlabel='year', ylabel='lifeExp'>
```



In [ ]:
```
# facet plot -> figure level function -> work with relplot
# it will not work with scatterplot and lineplot
sns.relplot(data=tips, x='total_bill', y='tip', kind='scatter', col='sex')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f68630464d0>
```



In [ ]:

Distribution Plots In Seaborn

In seaborn, a distribution plot, or displot, is used to visualize the distribution of data in a dataset. This can be done for both univariate and bivariate data.

For univariate data, the displot will show a histogram of the data, along with a line representing the kernel density estimate (KDE). The KDE is a smoother version of the histogram that can be used to better visualize the underlying distribution of the data.

For bivariate data, the displot will show a scatter plot of the data, along with a contour plot representing the joint distribution of the data. The contour plot is a way of visualizing the relationship between two variables, and can be used to identify clusters of data points or to see how the variables are related.

- used for univariate analysis
- used to find out the distribution
- Range of the observation
- Central Tendency
- is the data bimodal?
- Are there outliers?

Plots under distribution plot

- histplot
- kdeplot
- rugplot

Figure level and axes level

- figure level -> displot
- axes level -> histplot -> kdeplot -> rugplot

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

In [ ]:
```
tips = sns.load_dataset('tips')
tips
```

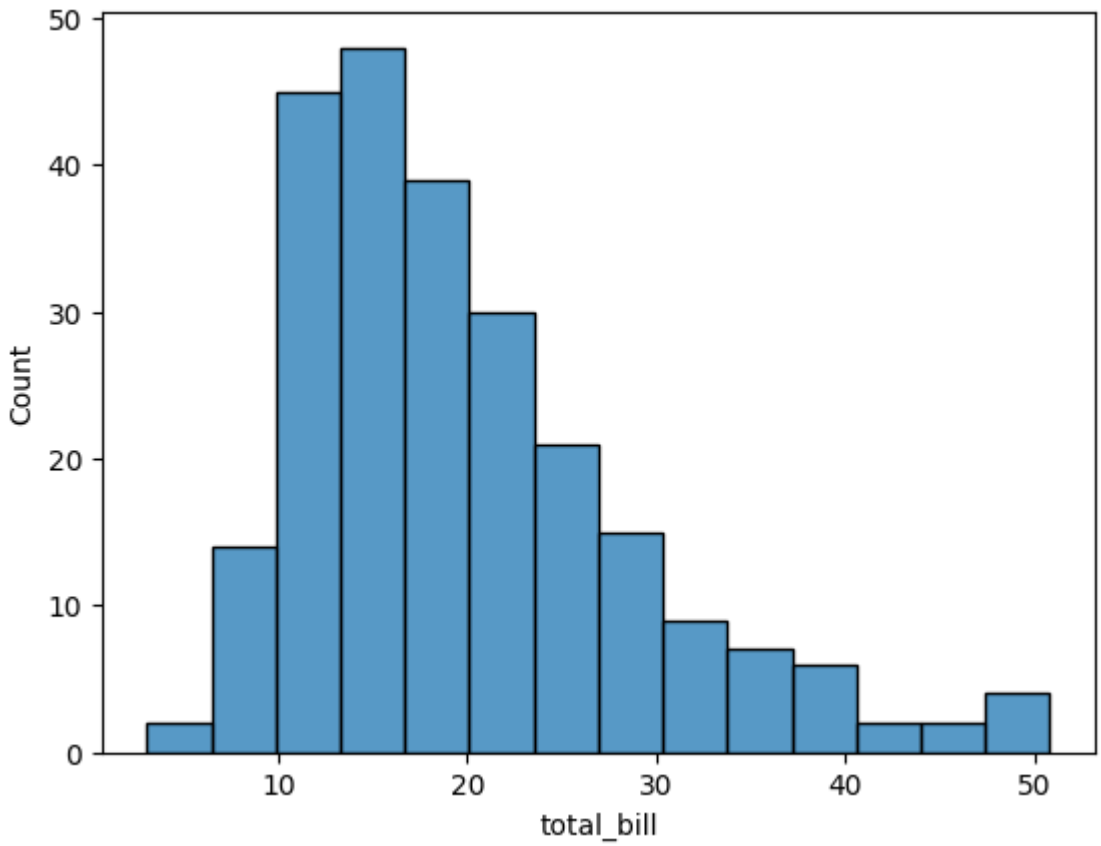
Out[ ]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

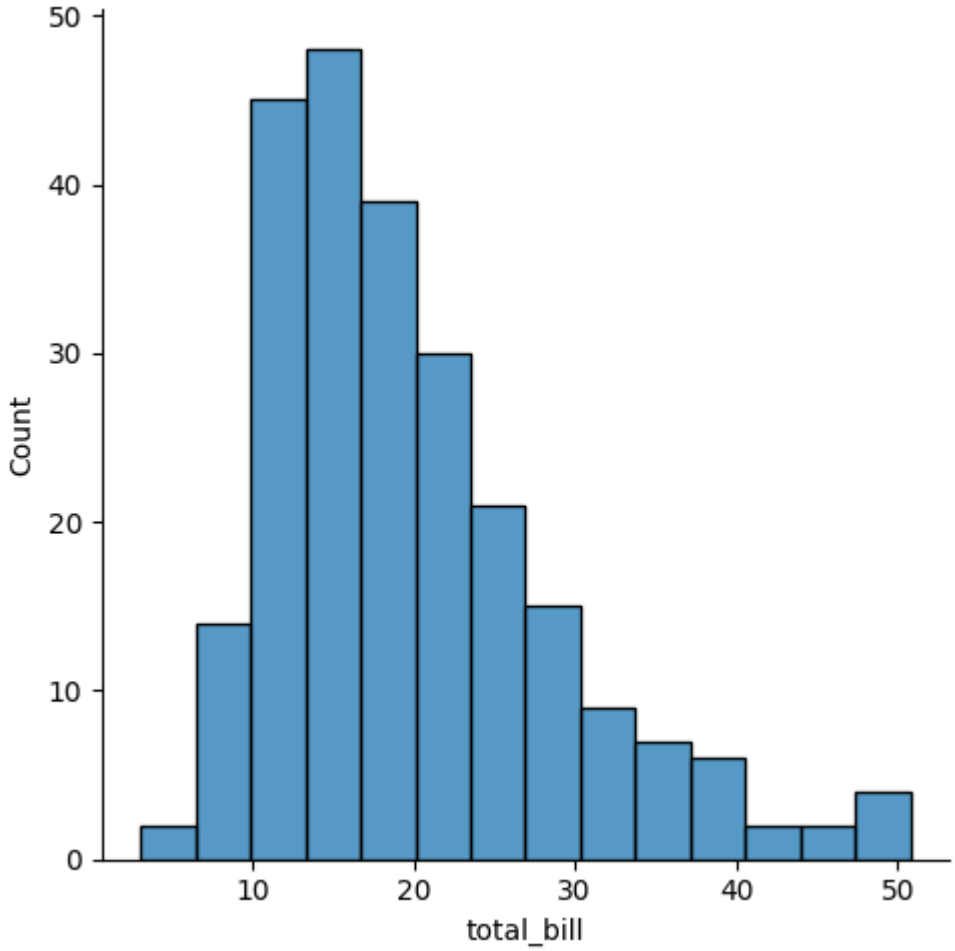
In [ ]:
```
# plotting univariate histogram
sns.histplot(data=tips, x='total_bill')
```

Out[ ]:
```
<Axes: xlabel='total_bill', ylabel='Count'>
```



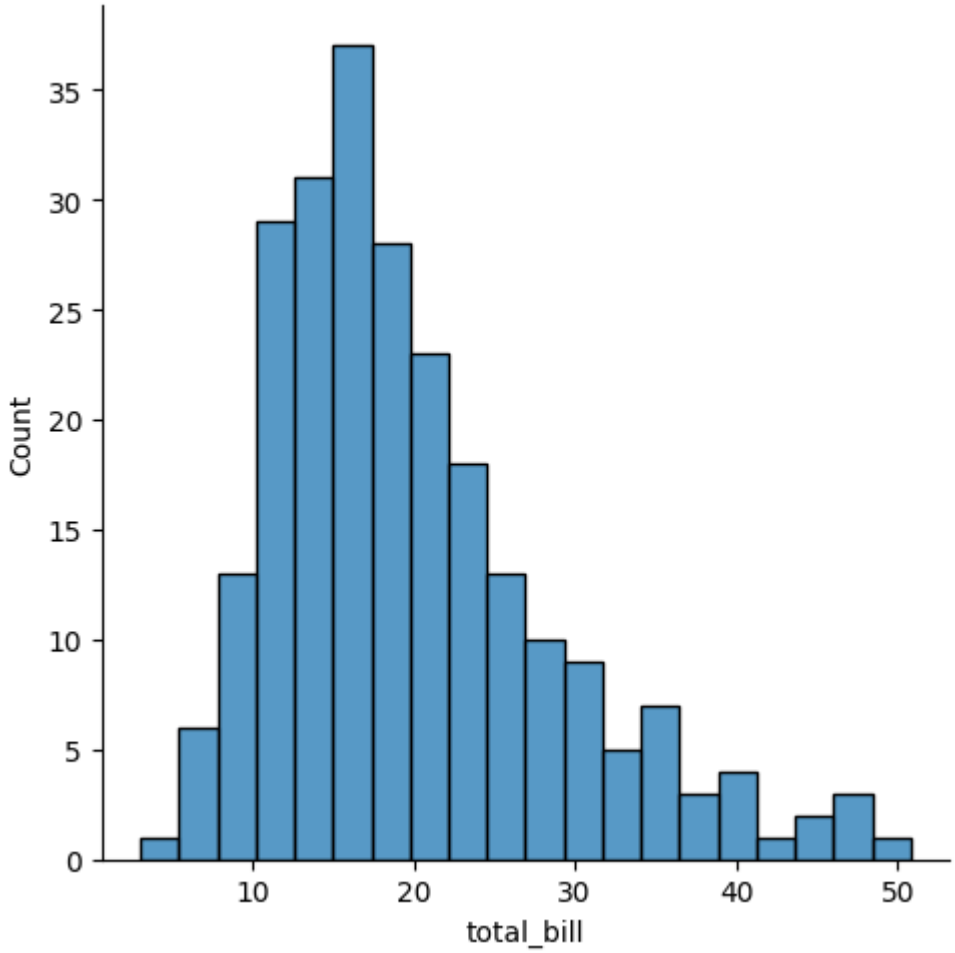
In [ ]:
```
# displot
sns.displot(data=tips, x='total_bill', kind='hist')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b831cbeaef0>
```



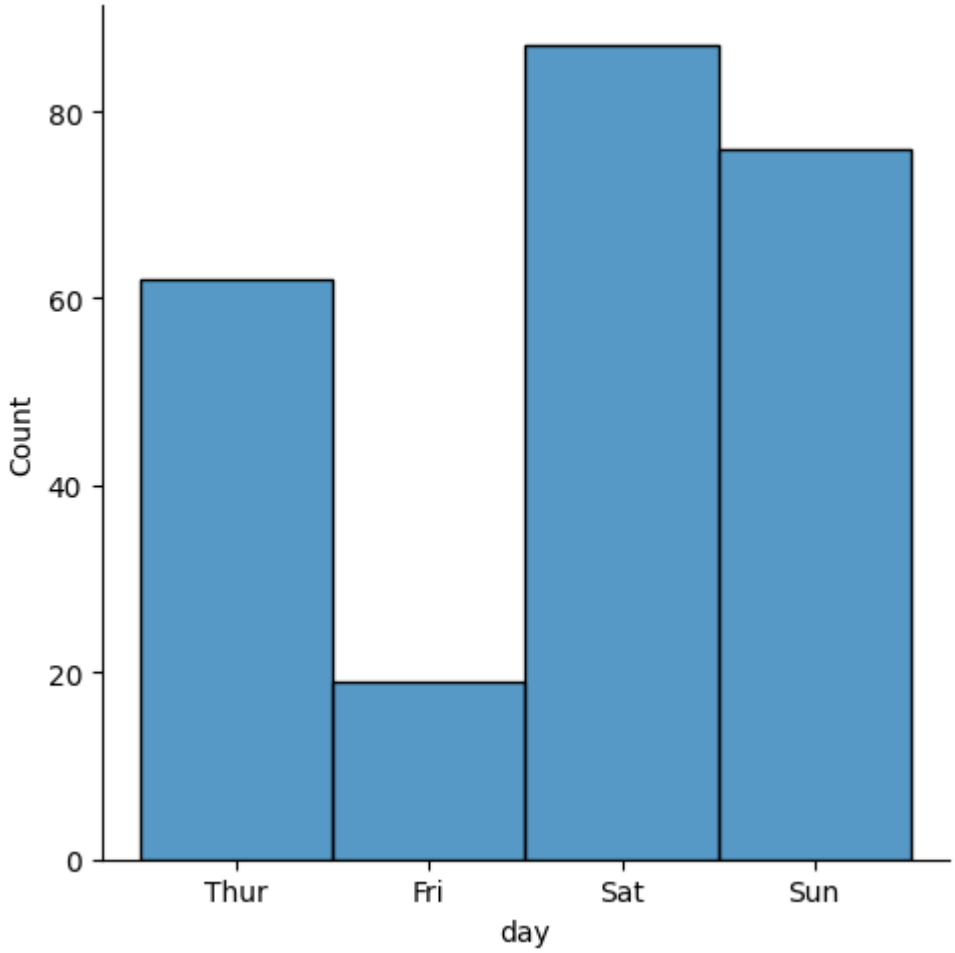
In [ ]:
```
sns.displot(data=tips, x='total_bill', kind='hist', bins=20)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b831a9590f0>
```



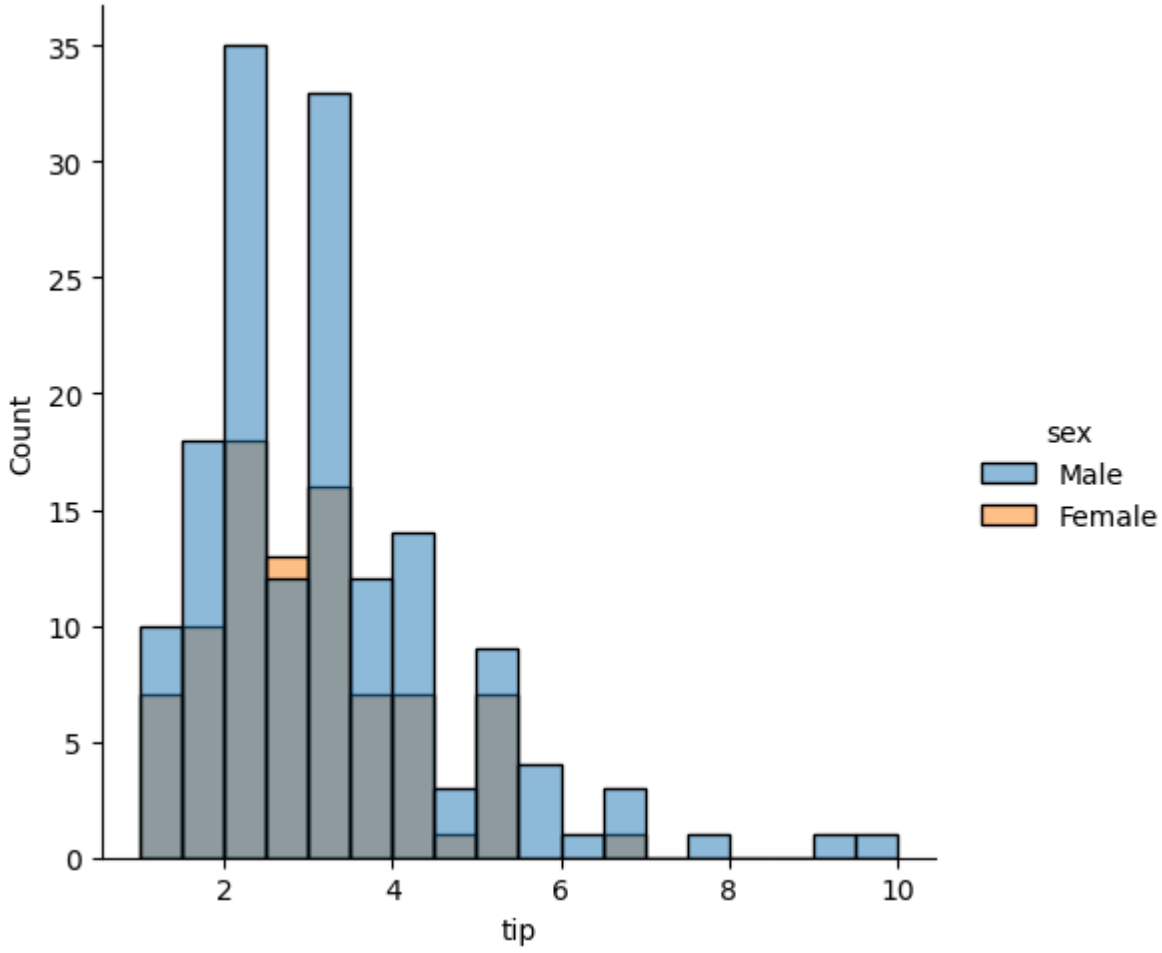
In [ ]:
```
# It's also possible to visualize the distribution of a categorical variable using
# Discrete bins are automatically set for categorical variables

# countplot
sns.displot(data=tips, x='day', kind='hist')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b83171a4a90>
```



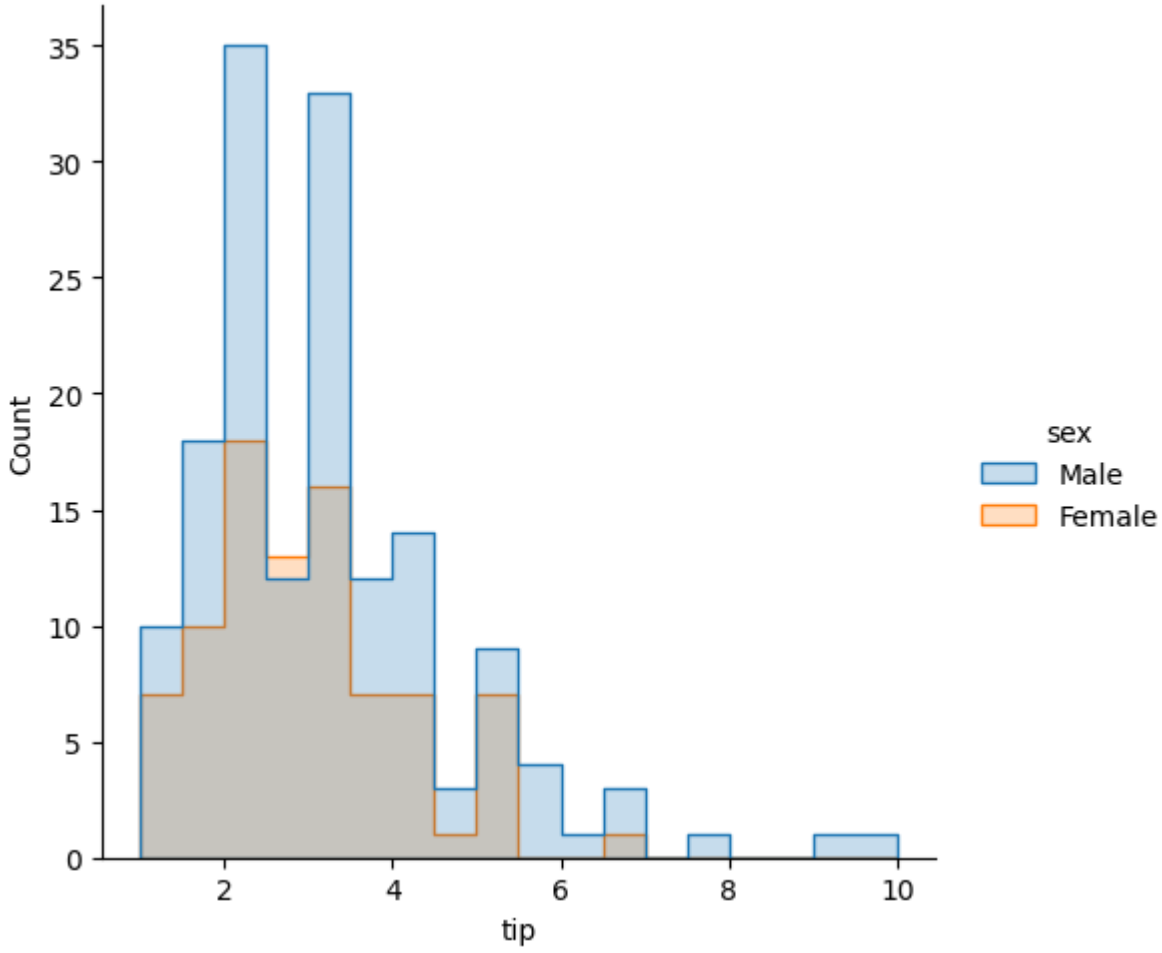
In [ ]:
```
# hue parameter
sns.displot(data=tips, x='tip', kind='hist',hue='sex')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b8317081f90>
```



In [ ]:
```
# element -> step
sns.displot(data=tips, x='tip', kind='hist', hue='sex', element='step')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b8316eddc30>
```



In [ ]:
```
titanic = sns.load_dataset('titanic')
```

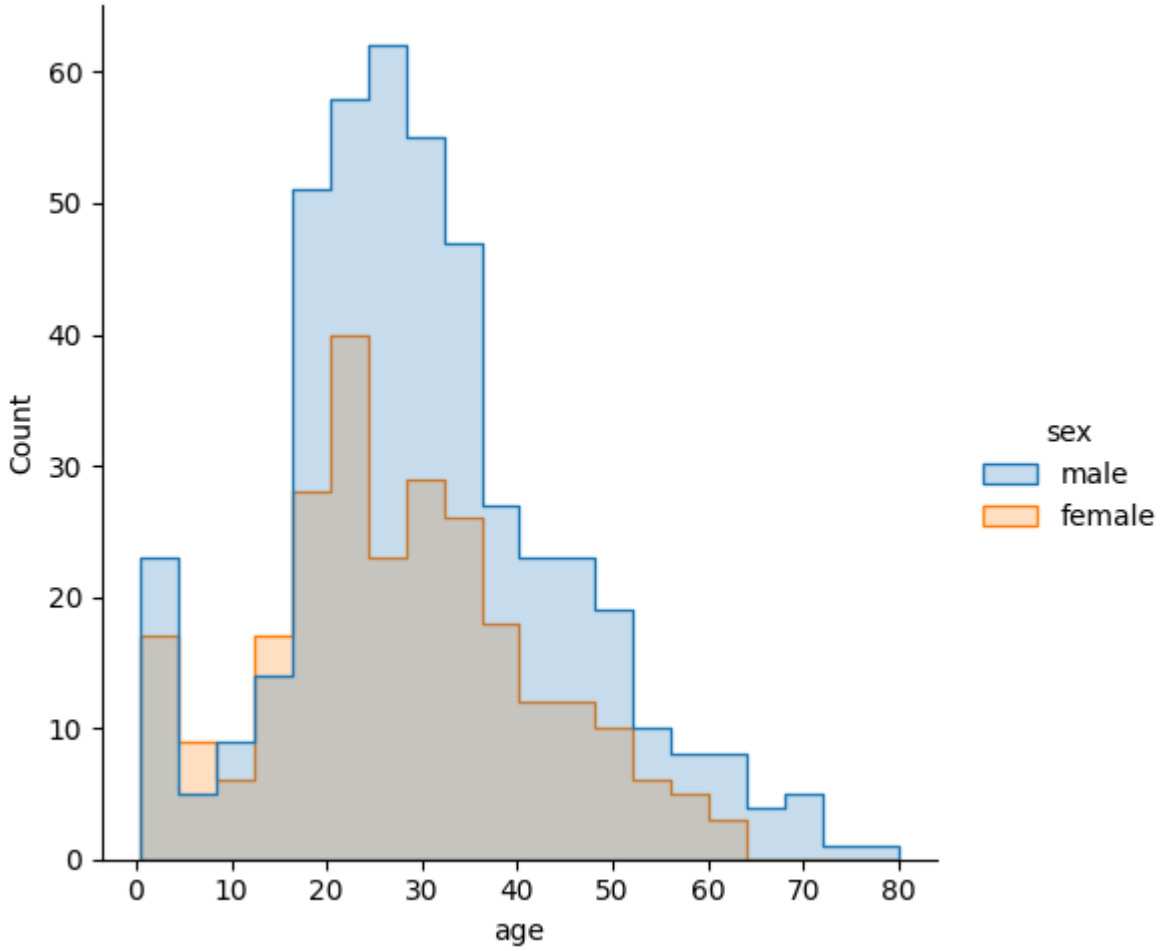
In [ ]:
```
titanic.head()
```

Out[ ]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	de
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Na
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Na
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Na

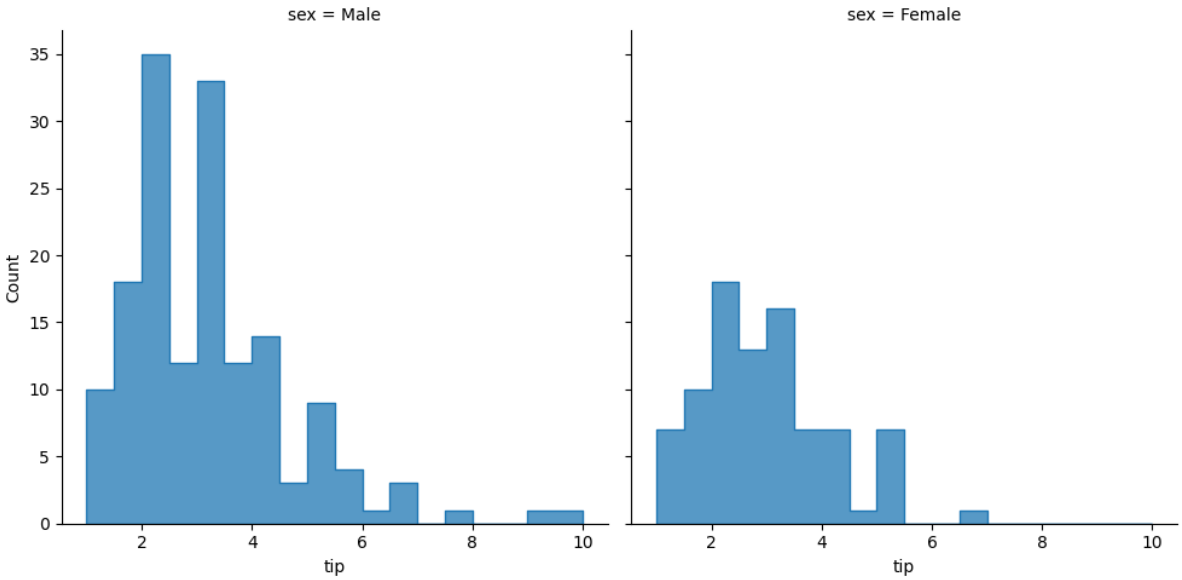
In [ ]:
```
sns.displot(data=titanic, x='age', kind='hist',element='step',hue='sex')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b831a9e2440>
```



In [ ]:
```
# faceting uusin col and rows -> and it not work on histplot function

sns.displot(data=tips, x='tip', kind='hist', col='sex', element='step')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b8316eddf30>
```

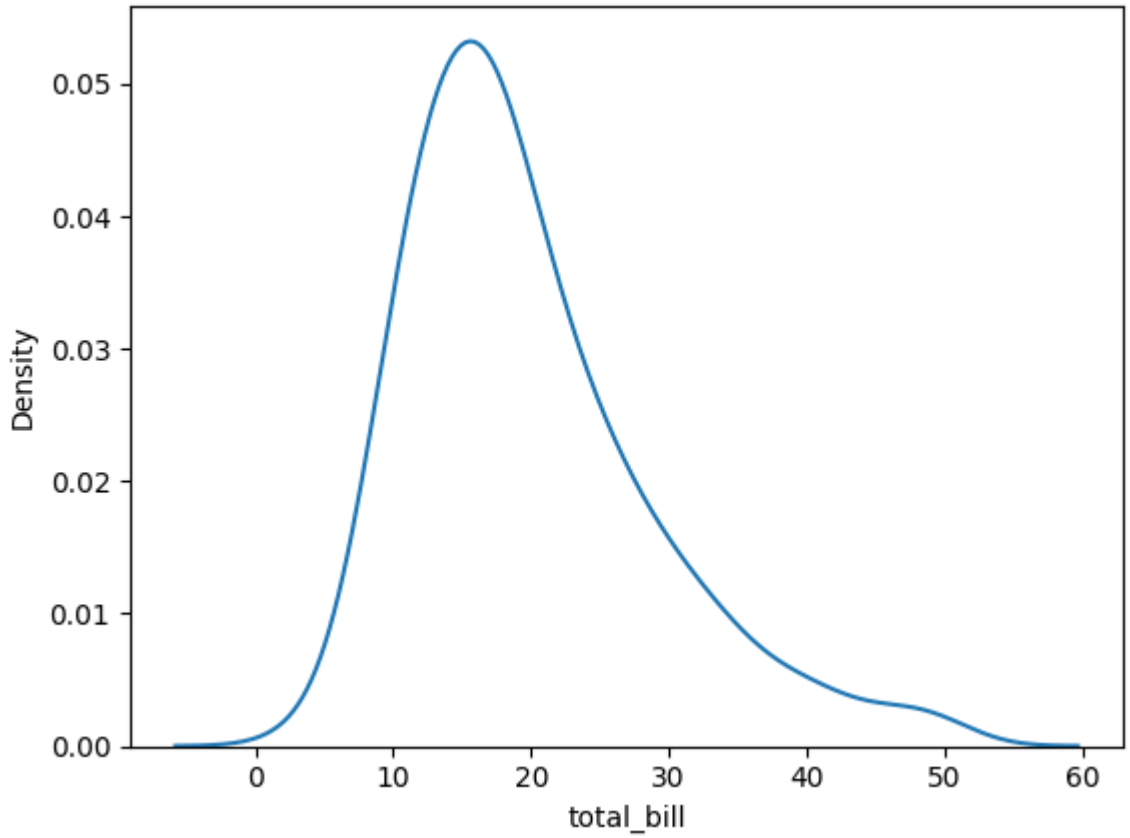


kdeplot

- Rather than using discrete bins, a KDE plot smooths the observations with a Gaussian kernel, producing a continuous density estimate

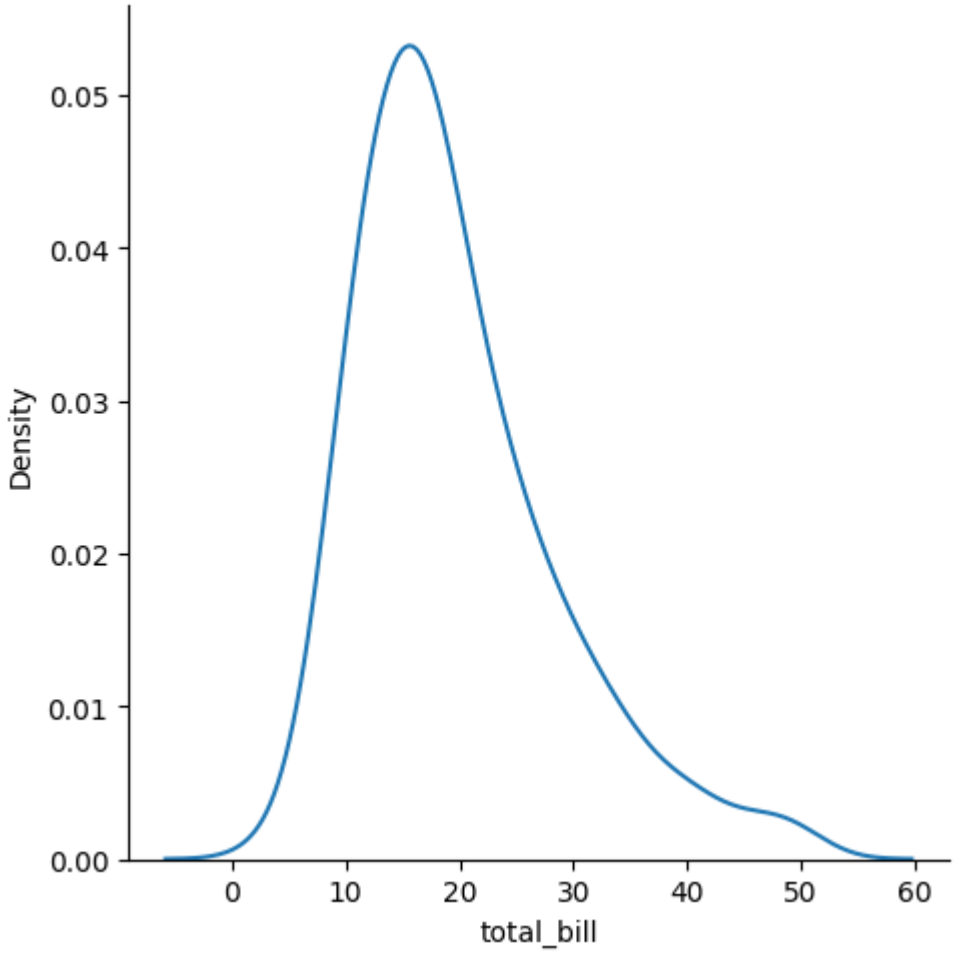
In [ ]:
```
sns.kdeplot(data=tips,x='total_bill')
```

Out[ ]:
```
<Axes: xlabel='total_bill', ylabel='Density'>
```



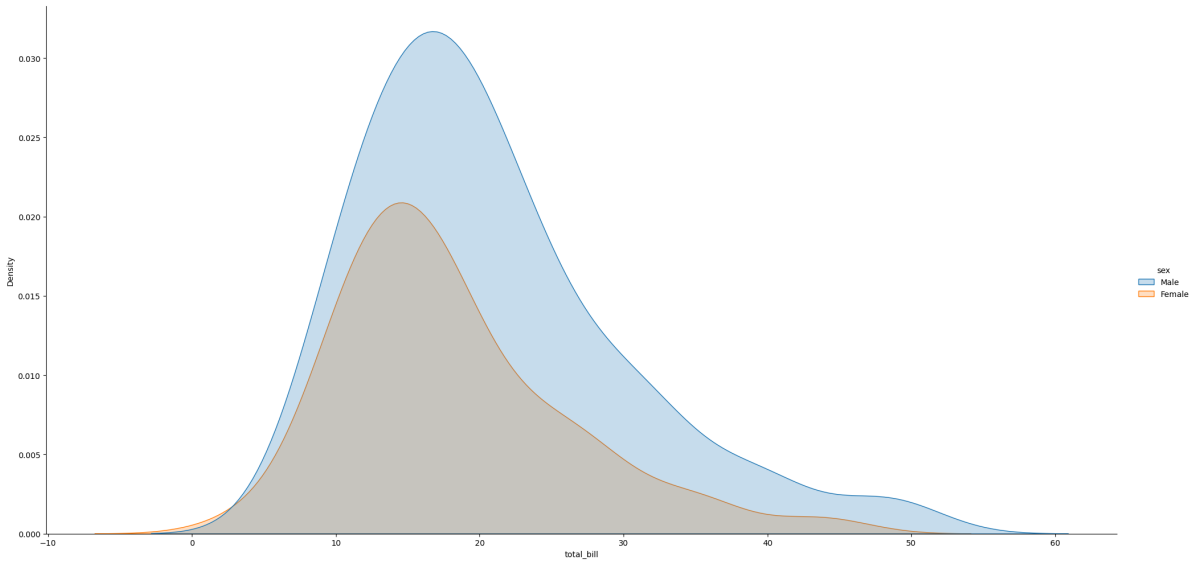
In [ ]:
```
sns.displot(data=tips,x='total_bill',kind='kde')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b8316ba3250>
```



In [ ]:
```
# hue -> fill
sns.displot(data=tips,x='total_bill',kind='kde',hue='sex',fill=True,height=10,aspec
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7b831691bf40>
```



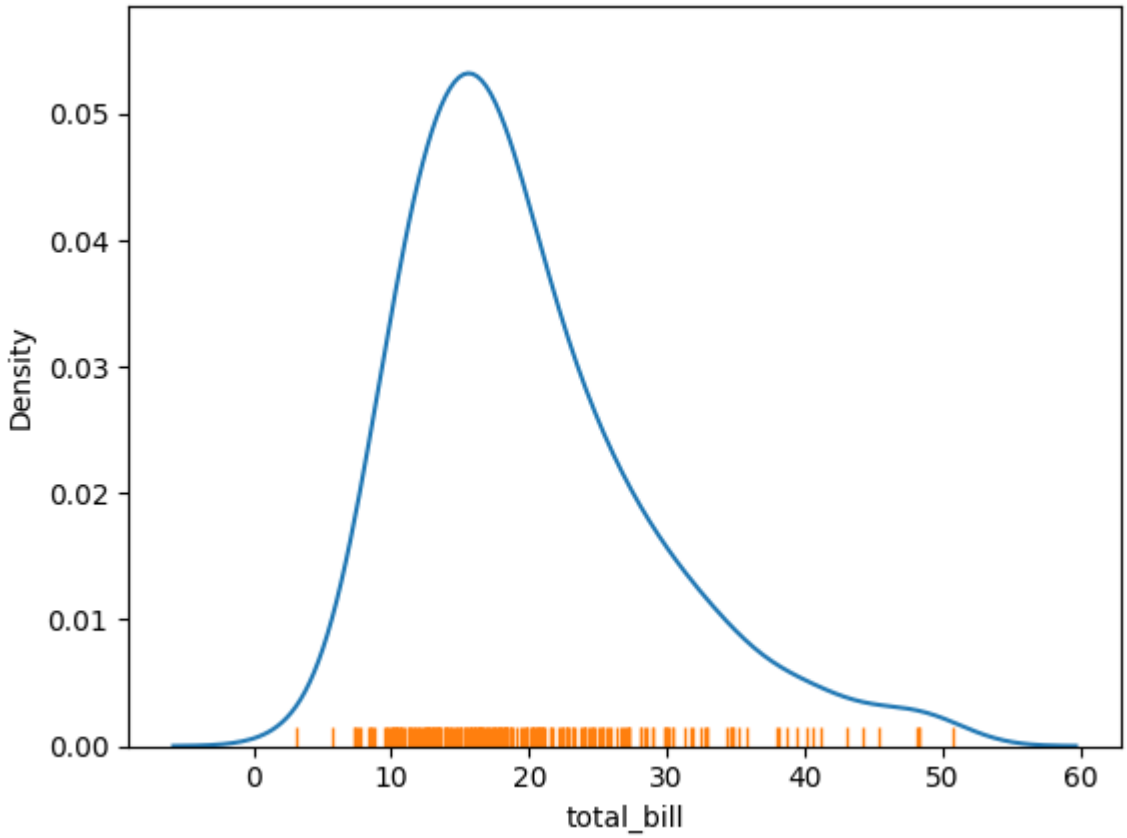
Rugplot

Plot marginal distributions by drawing ticks along the x and y axes.

This function is intended to complement other plots by showing the location of individual observations in an unobtrusive way.

In [ ]:
```
sns.kdeplot(data=tips,x='total_bill')
sns.rugplot(data=tips,x='total_bill')
```

Out[ ]:
```
<Axes: xlabel='total_bill', ylabel='Density'>
```

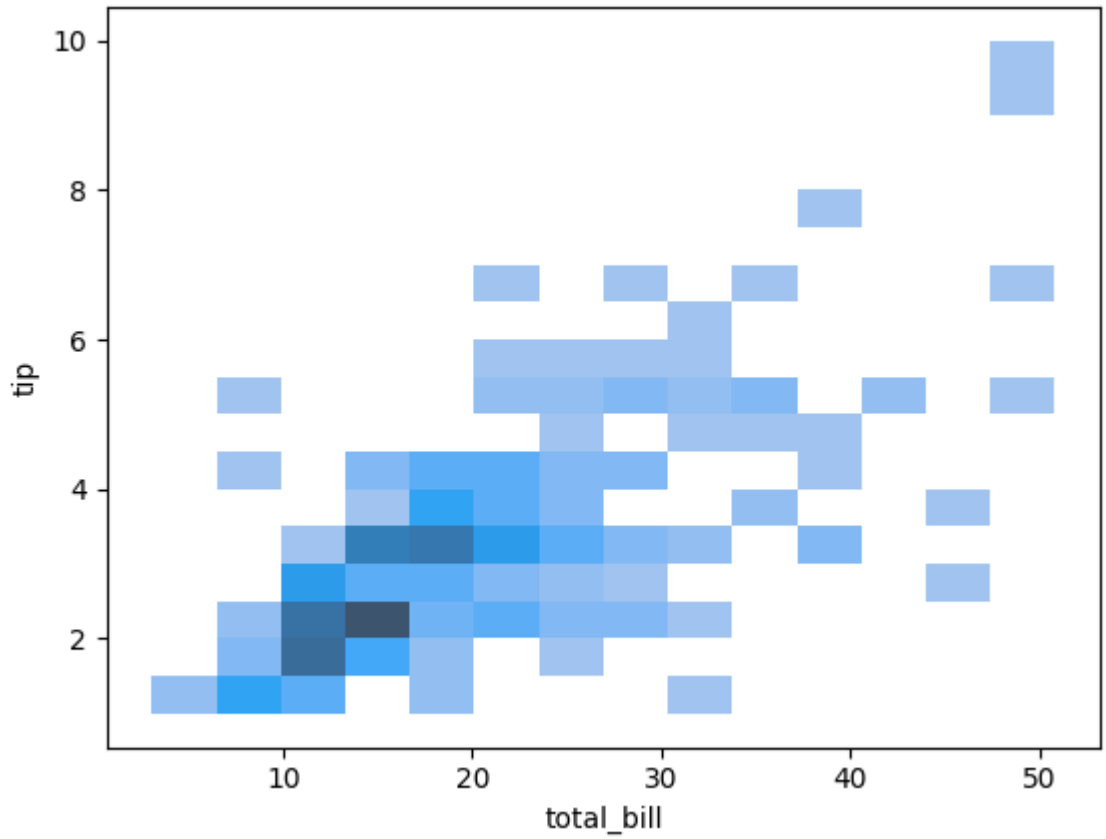


Bivariate histogram

A bivariate histogram bins the data within rectangles that tile the plot and then shows the count of observations within each rectangle with the fill color

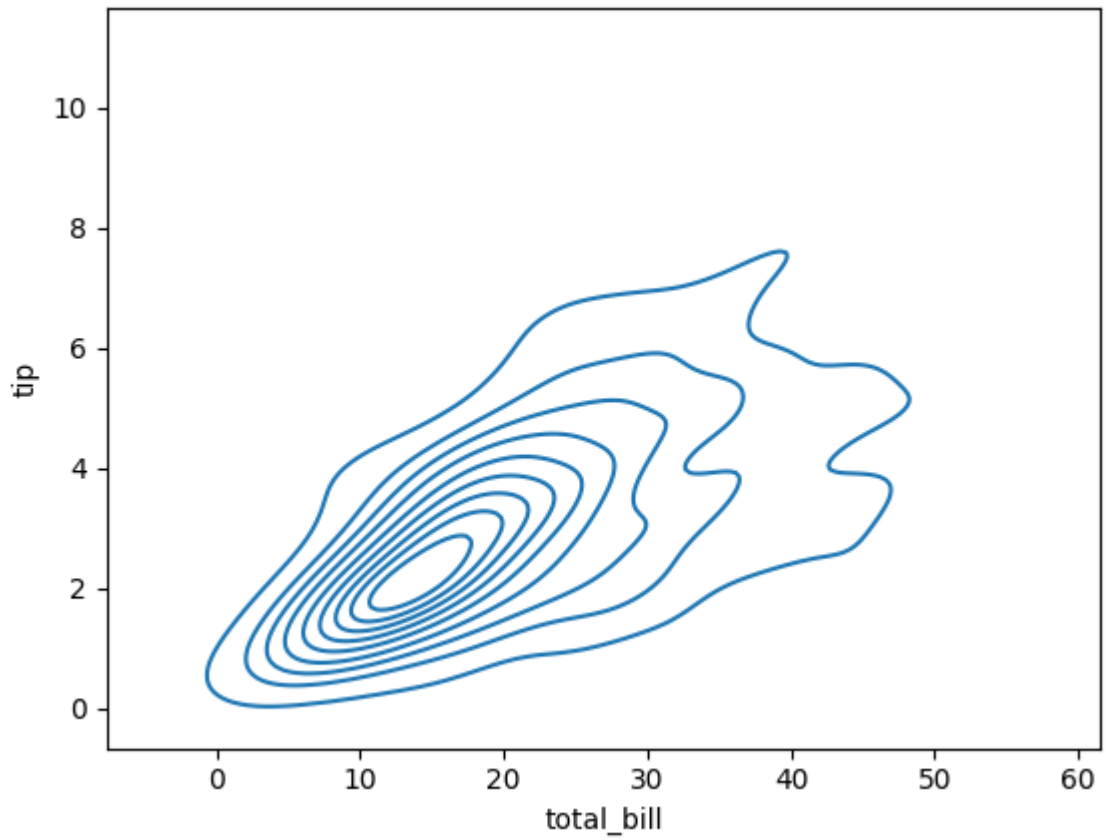
In [ ]:
```
sns.histplot(data=tips, x='total_bill', y='tip')
```

Out[ ]:
```
<Axes: xlabel='total_bill', ylabel='tip'>
```



In [ ]:
```
# Bivariate Kdeplot
# a bivariate KDE plot smoothes the (x, y) observations with a 2D Gaussian
sns.kdeplot(data=tips, x='total_bill', y='tip')
```

Out[ ]:
```
<Axes: xlabel='total_bill', ylabel='tip'>
```



Matrix Plot

In Seaborn, both `heatmap` and `clustermap` functions are used for visualizing matrices, but they serve slightly different purposes.

1. **Heatmap:**
   - The `heatmap` function is used to plot rectangular data as a color-encoded matrix. It is essentially a 2D representation of the data where each cell is colored based on its value.
   - Heatmaps are useful for visualizing relationships and patterns in the data, making them suitable for tasks such as correlation matrices or any other situation where you want to visualize the magnitude of a phenomenon.

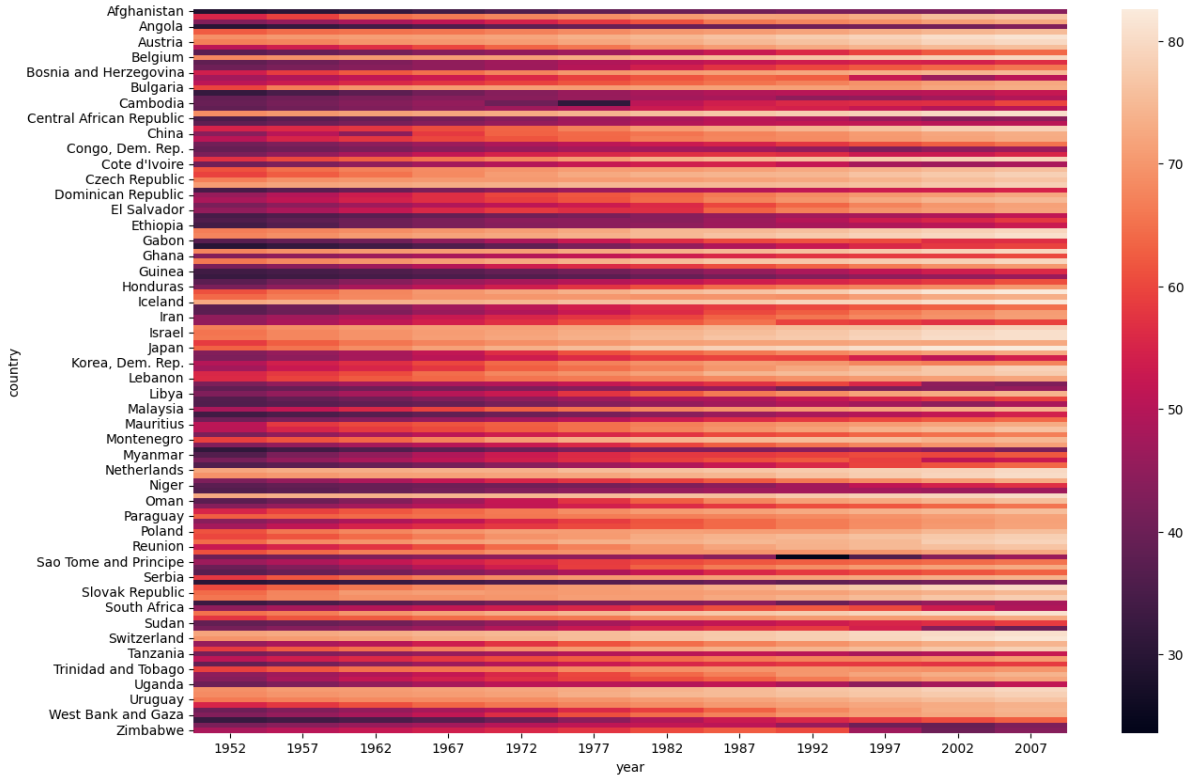
1. **Clustermap:**
   - The `clustermap` function, on the other hand, not only visualizes the matrix but also performs hierarchical clustering on both rows and columns to reorder them based on similarity.
   - It is useful when you want to identify patterns not only in the individual values of the matrix but also in the relationships between rows and columns.

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

In [ ]:
```
gap = px.data.gapminder()
```

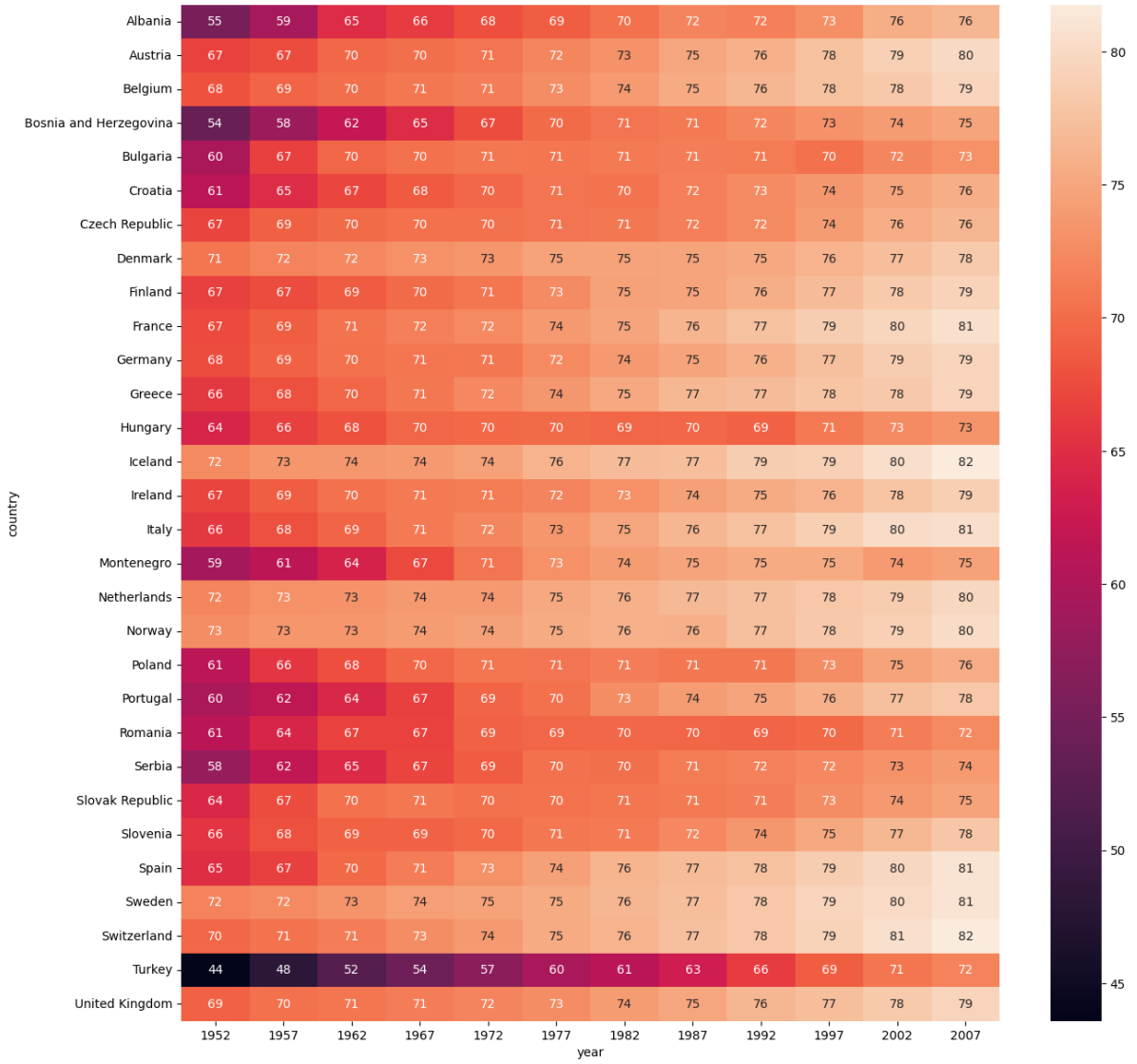
In [ ]:
```
# Heatmap

# Plot rectangular data as a color-encoded matrix
temp_df = gap.pivot(index='country',columns='year',values='lifeExp')

# axes level function
plt.figure(figsize=(15,10))
sns.heatmap(temp_df)
```

Out[ ]:
```
<Axes: xlabel='year', ylabel='country'>
```



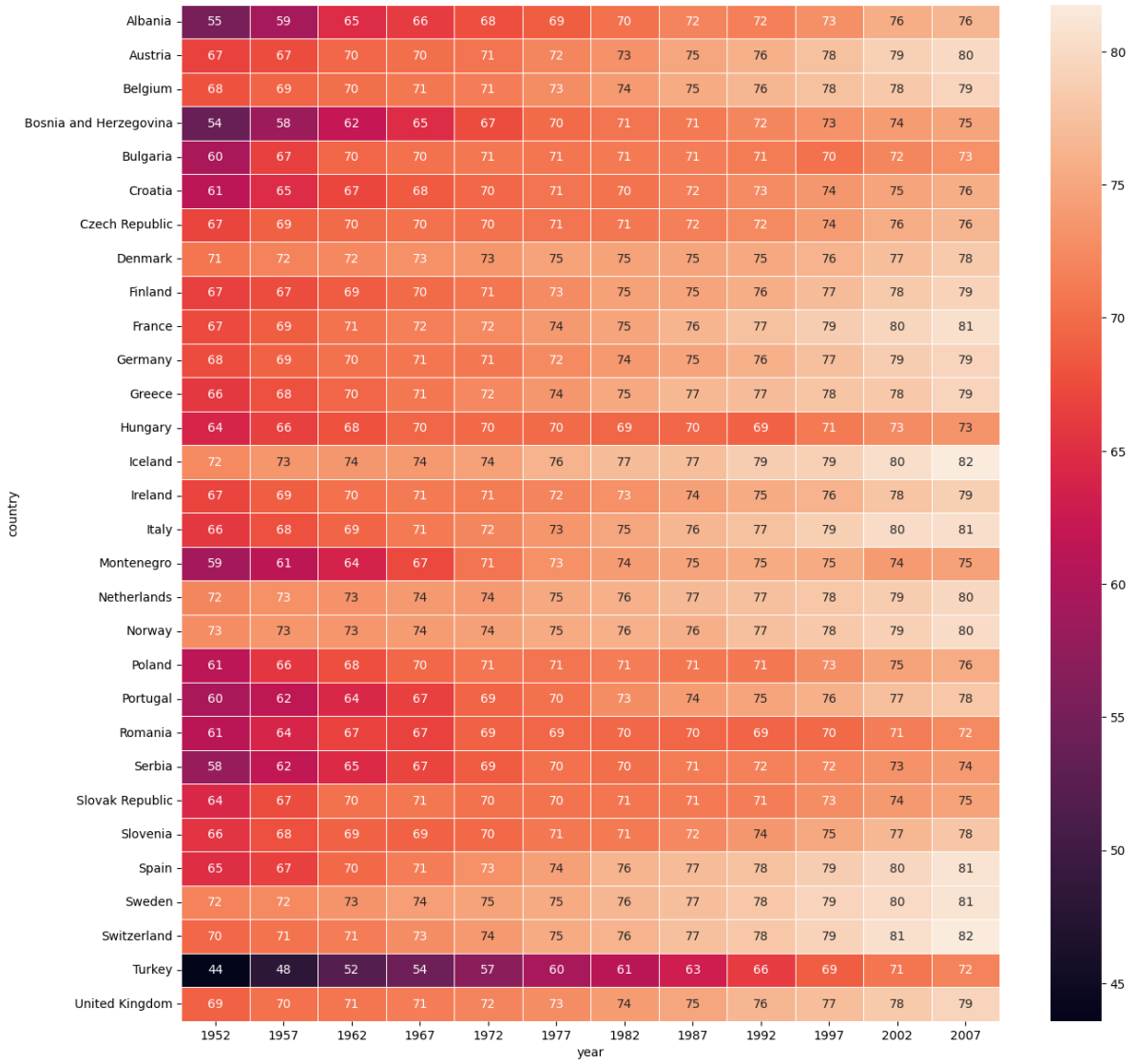
In [ ]:
```
# annot
temp_df = gap[gap['continent'] == 'Europe'].pivot(index='country',columns='year',va

plt.figure(figsize=(15,15))
sns.heatmap(temp_df,annot=True)
```

Out[ ]:
```
<Axes: xlabel='year', ylabel='country'>
```



In [ ]:
```
# linewidth
# annot
temp_df = gap[gap['continent'] == 'Europe'].pivot(index='country',columns='year',va

plt.figure(figsize=(15,15))
sns.heatmap(temp_df,annot=True,linewidth=0.5)
```

Out[ ]:
```
<Axes: xlabel='year', ylabel='country'>
```



In [ ]:
```
# cmap
# annot
temp_df = gap[gap['continent'] == 'Europe'].pivot(index='country',columns='year',va

plt.figure(figsize=(15,15))
sns.heatmap(temp_df,annot=True,linewidth=0.5, cmap='summer')
```

Out[ ]:
```
<Axes: xlabel='year', ylabel='country'>
```