

MATPLOTLIB

DATA SCIENCE

▼ Data Visualization : Matplotlib

```
import numpy as np
from matplotlib import pyplot as plt
```

▼ Line plot

- Displays data points connected by straight lines, suitable for showing trends or continuous data over time.

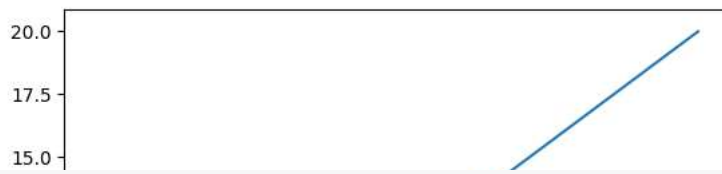
```
x = np.arange(1, 11)
x

array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

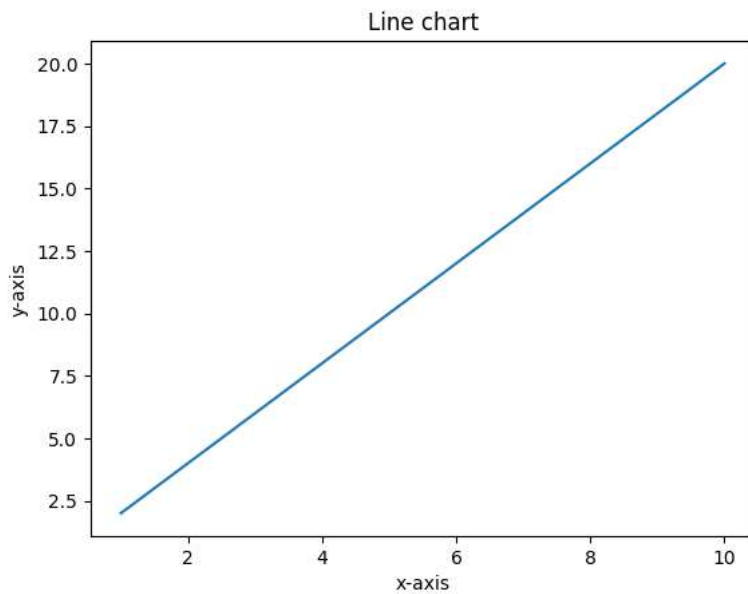
```
y = 2 * x
y

array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

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

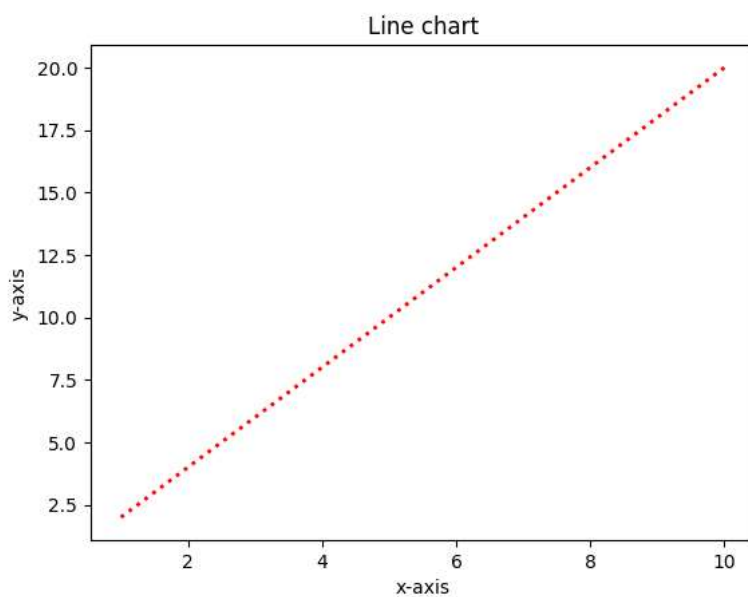


```
plt.plot(x,y)
plt.title('Line chart')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```



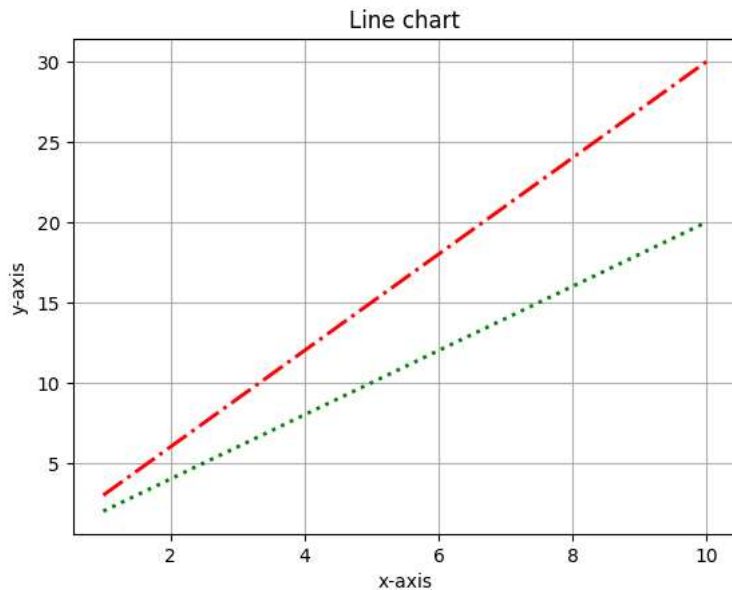
▼ Color, line width, linestyle

```
plt.plot(x,y, linestyle = ':', linewidth = 2, color = 'r')
plt.title('Line chart')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```



▾ Adding two line to same plot

```
y1 = 2*x
y2 = 3*x
plt.plot(x,y1, color = 'g', linewidth = 2, linestyle = ':')
plt.plot(x,y2, color = 'r', linewidth = 2, linestyle = '-.')
plt.title('Line chart')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.grid(True)
plt.show()
```



▾ Sub-plot

`plt.subplot(row, number of col, plot no)`

```
x = np.arange(1, 25)

y1 = x*2
y2 = y1*3

# Sub plot
plt.subplot(1, 3, 1)
plt.plot(x, y1, color = 'r', linewidth = 2, linestyle = '-.')

plt.title('Sub- plot -1')
plt.grid(True)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
# plt.show()

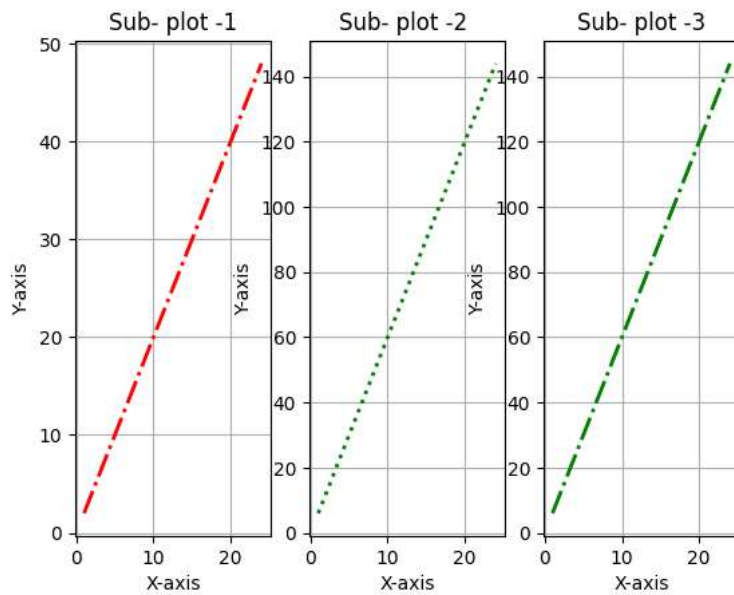
plt.subplot(1, 3, 2)
plt.plot(x, y2, color = 'g', linewidth = 2, linestyle = ':')

plt.title('Sub- plot -2 ')
plt.grid(True)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

plt.subplot(1, 3, 3)
plt.plot(x, y2, color = 'g', linewidth = 2, linestyle = '-.')

plt.title('Sub- plot -3 ')
plt.grid(True)
plt.xlabel('X-axis')
```

```
plt.ylabel('Y-axis')
plt.show()
```



```
x = np.arange(1, 25)

y1 = x*2
y2 = y1*3

# Sub plot
plt.subplot(2, 2, 1)
plt.plot(x, y1, color = 'r', linewidth = 2, linestyle = '-.')

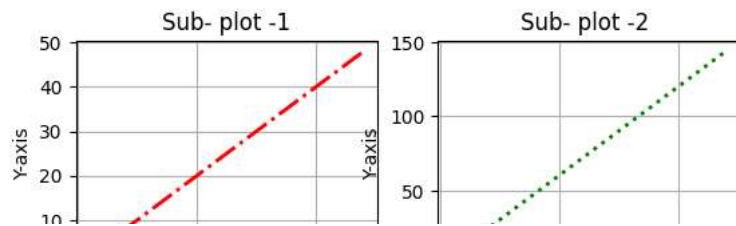
plt.title('Sub- plot -1')
plt.grid(True)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
# plt.show()

plt.subplot(2, 2, 2)
plt.plot(x, y2, color = 'g', linewidth = 2, linestyle = ':')

plt.title('Sub- plot -2 ')
plt.grid(True)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

plt.subplot(2, 2, 3)
plt.plot(x, y2, color = 'g', linewidth = 2, linestyle = '-.')

plt.title('Sub- plot -3 ')
plt.grid(True)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



▼ Bar Plot

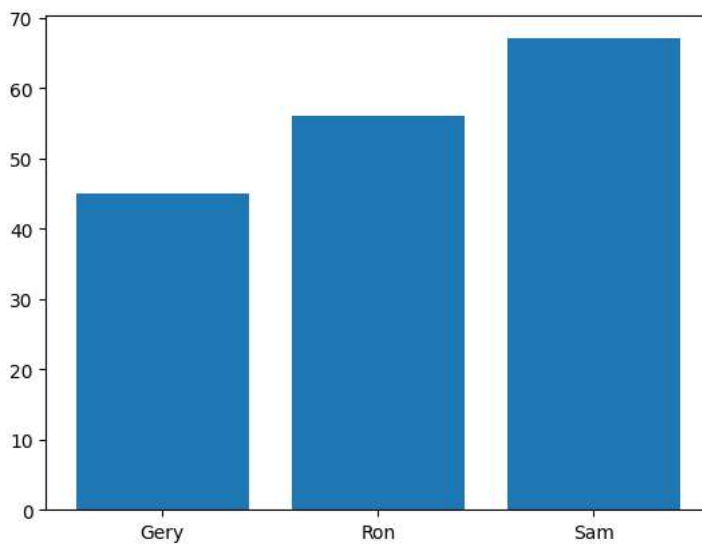
- Represents categorical data with rectangular bars, commonly used for comparing different categories.

```
student = { 'Gery': 45, 'Ron': 56, 'Sam': 67 }
```

▼ Bar plot takes 2 parameter categorical name and numerical values

```
names = list(student.keys())
values = list(student.values())

plt.bar(names, values)
plt.show()
```



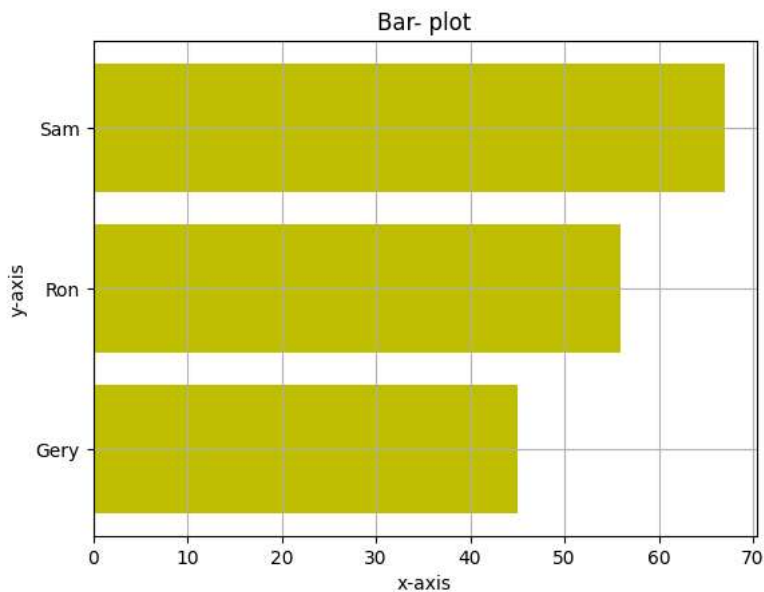
```
plt.bar(names, values, color = 'g')
plt.title('Bar- plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.grid(True)
plt.show()
```



▼ Horizontal bar - plot

Bar-plot

```
plt.barh(names, values, color = 'y')
plt.title('Bar- plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.grid(True)
plt.show()
```



Double-click (or enter) to edit

```
rng = np.random.default_rng()
y = rng.random(10)
y
array([0.57651115, 0.83646096, 0.92390669, 0.92223764, 0.43793735,
       0.43026936, 0.85292748, 0.96921211, 0.91769442, 0.89557301])
```

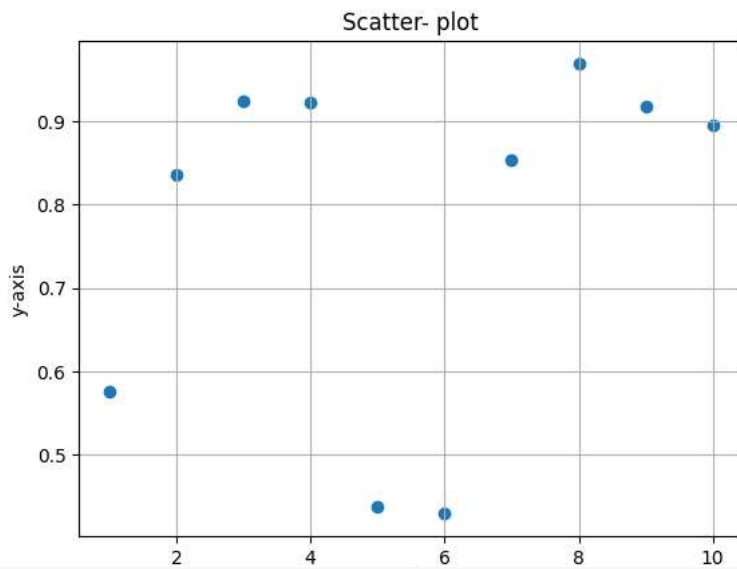
▼ Scatter plot

- Plot: Shows the relationship between two variables by displaying individual data points as dots on a two-dimensional plane.

```
# same in number
x = [1,2,3,4,5,6,7,8,9,10]

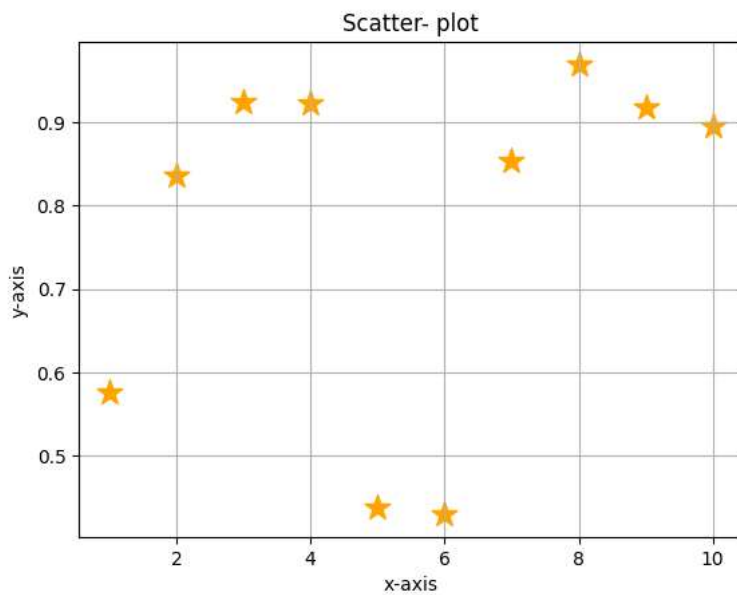
# Scatter plt
plt.scatter(x, y)
plt.title('Scatter- plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.grid(True)

plt.show()
```



```
plt.scatter(x, y, marker = '*', color = 'orange', s = 200) # s : size of start
plt.title('Scatter- plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.grid(True)

plt.show()
```

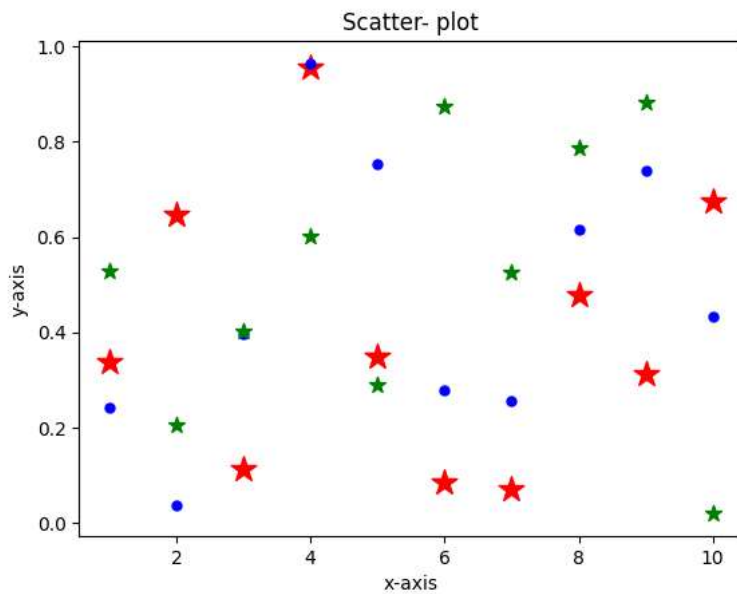


▾ Adding two marker in same scatter plt

```
y1 = rng.random(10)
y2 = rng.random(10)
y3 = rng.random(10)

plt.scatter(x, y1, marker = '*', color = 'red', s = 200)
plt.scatter(x, y2, marker = '.', color = 'blue', s = 100)
plt.scatter(x, y3, marker = '*', color = 'green', s = 80)
plt.title('Scatter- plot')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
# plt.grid(True)

plt.show()
```



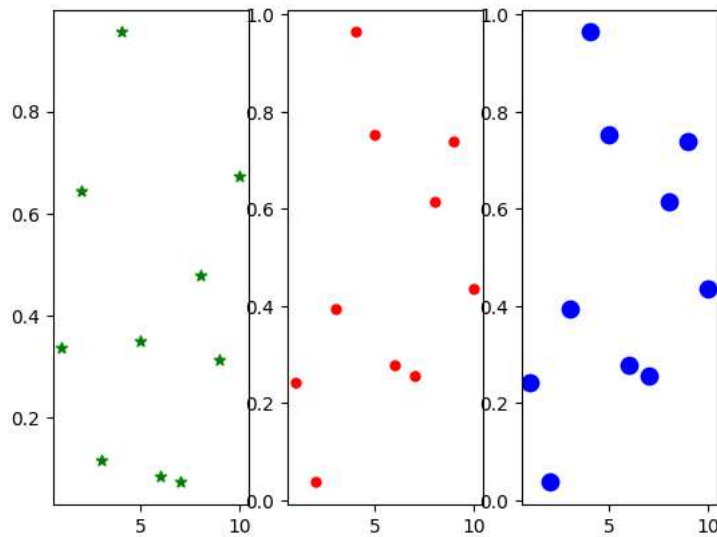
Instead of create two or three in one plot we can create subplot

```
plt.subplot(1, 3, 1)
plt.scatter(x, y1, marker = '*', color = 'g')

plt.subplot(1, 3, 2)
plt.scatter(x, y2, marker = '.', color = 'r', s = 100)

plt.subplot(1, 3, 3)
plt.scatter(x, y2, color = 'b', s = 80)

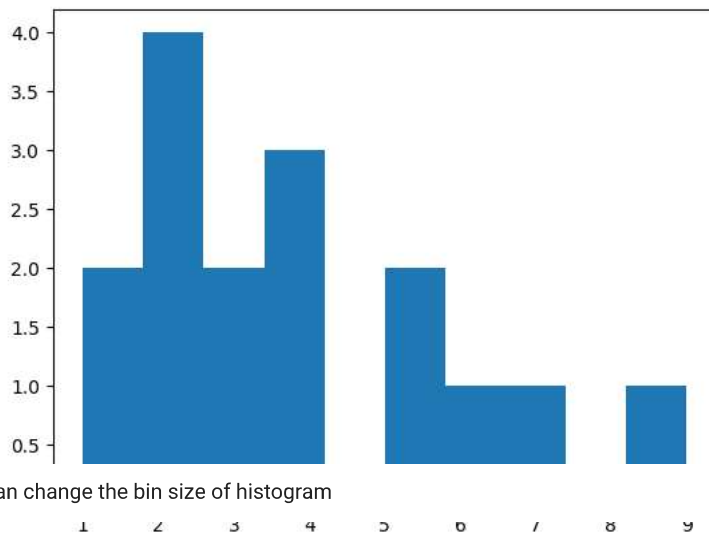
plt.show()
```



▼ Histogram

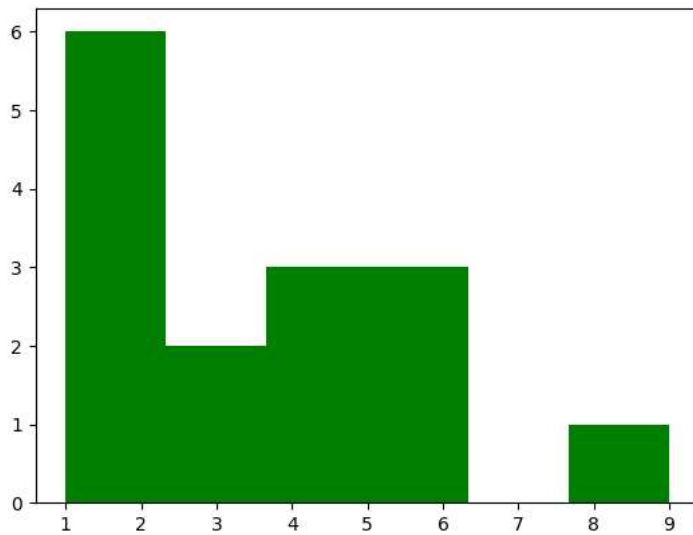
- Visualizes the distribution of continuous or discrete data by dividing it into bins and displaying the frequency or density of values within each bin.

```
data = [1, 1, 2, 3, 2, 4, 2, 2, 5, 4, 7, 9, 3, 4, 5, 6]
plt.hist(data)
plt.show()
```

We can change the bin size of histogram

```
plt.hist(data, color = 'g', bins = 6)
plt.show()
```




```
import pandas as pd
df= pd.read_csv('iris.csv')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
df.columns
```

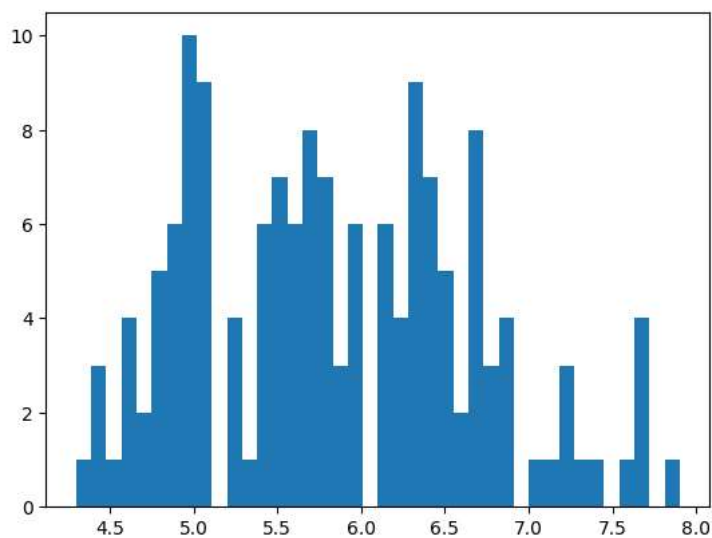
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
      'species'],
      dtype='object')
```

```
df.describe()
```

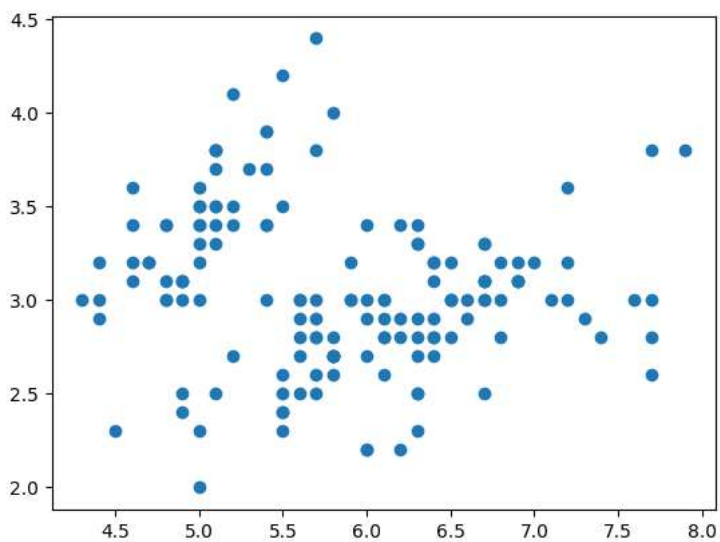
	sepal_length	sepal_width	petal_length	petal_width	
count	150.000000	150.000000	150.000000	150.000000	
mean	5.843333	3.054000	3.758667	1.198667	
std	0.828066	0.433594	1.764420	0.763161	
min	4.300000	2.000000	1.000000	0.100000	
25%	5.100000	2.800000	1.600000	0.300000	
50%	5.800000	3.000000	4.350000	1.300000	

▾ lets plot histogram for "sepal_length"

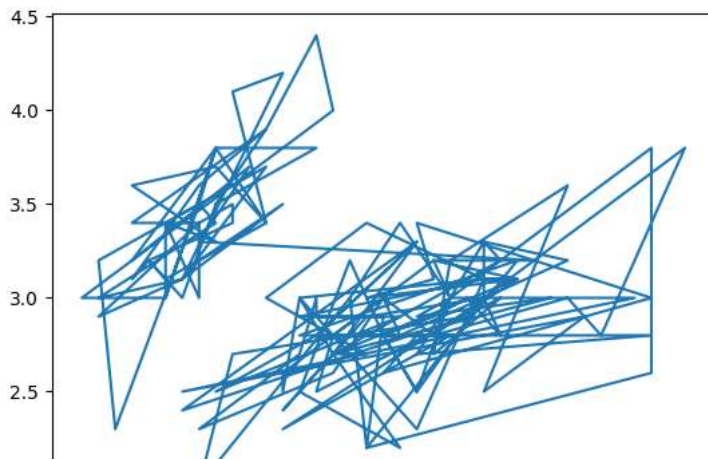
```
plt.hist(df['sepal_length'], bins = 40)
plt.show()
```



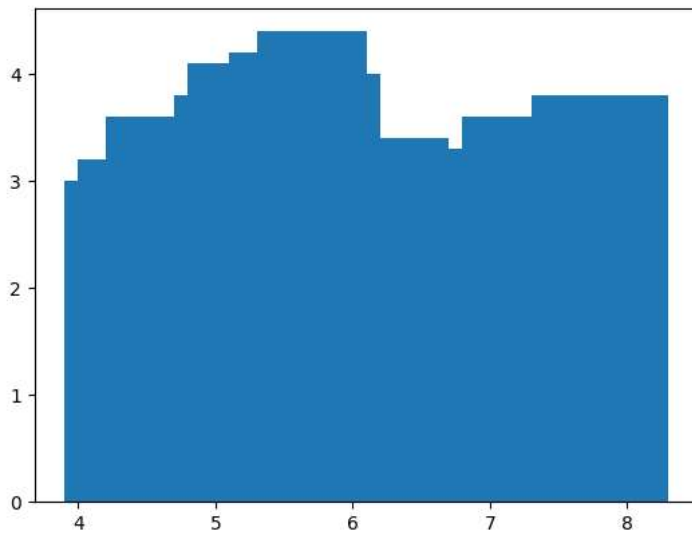
```
plt.scatter(df['sepal_length'], df['sepal_width'])
plt.show()
```



```
plt.plot(df['sepal_length'], df['sepal_width'])
plt.show()
```



```
plt.bar(df['sepal_length'], df['sepal_width'])
plt.show()
```



▼ Box plot : 5 number summary

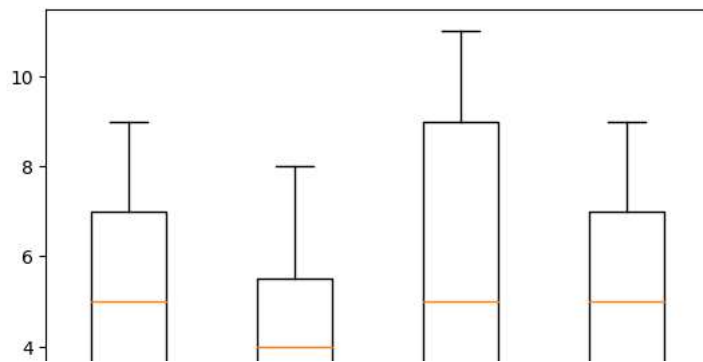
- Illustrates the distribution of a dataset by displaying the quartiles, median, and any outliers or extreme values.

1. max
2. 75%
3. 50% [median]
4. 25%
5. min

```
l1 = [1,2,3,4,5,6,7,8,9]
l2 = [3,4,1,5,6,3,8]
l3 = [1,2,8,9, 10, 11,2,3,5]

data = list([l1, l2, l3])
# plt.boxplot(data)
plt.boxplot([l1, l2, l3, l1])

plt.show()
```

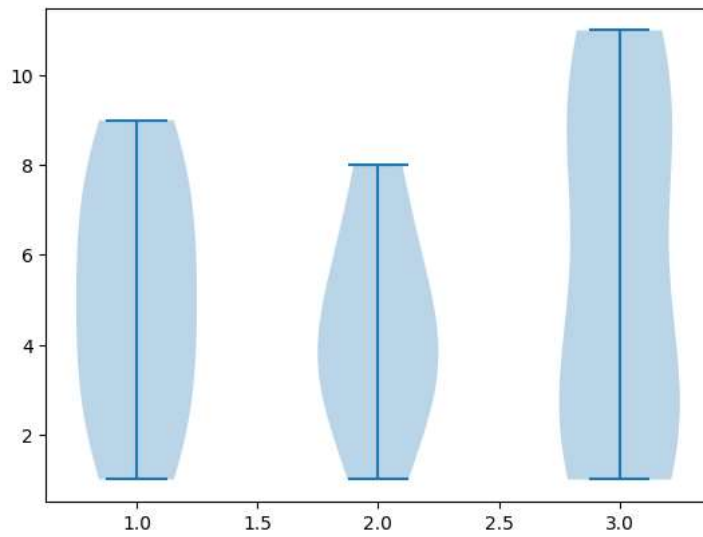


▼ Violin - plot

- Combines a box plot and a kernel density plot to show the distribution of data, providing information about both the summary statistics and the density.

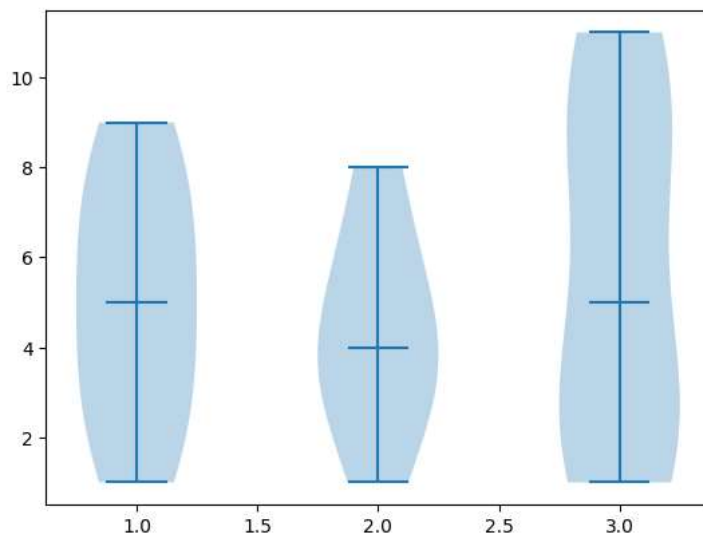
```
data1 = list([11, 12, 13])
plt.violinplot(data1)

plt.show()
```



```
plt.violinplot(data1, showmedians = True)

plt.show()
```

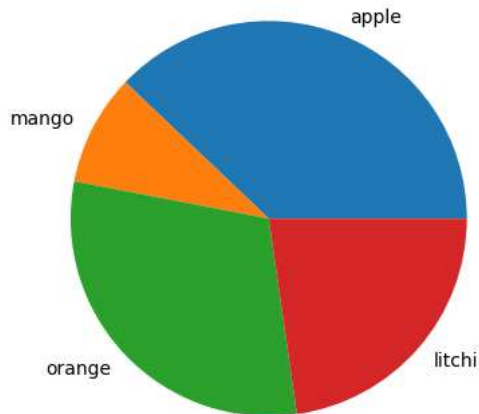


▾ Pie Chart

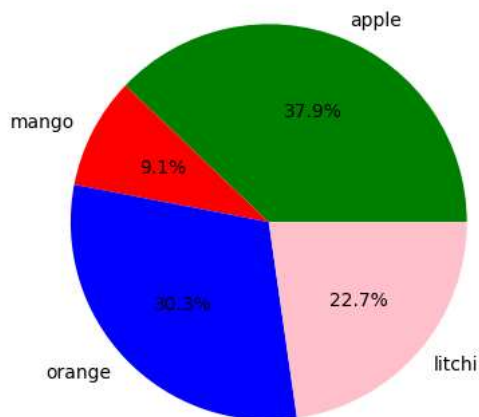
- Represents data as slices of a pie, where each slice represents a proportion of the whole, suitable for showing parts of a whole.

```
fruits = ['apple', 'mango', 'orange', 'litchi']  
quantity = [50, 12, 40, 30]
```

```
plt.pie(quantity, labels = fruits)  
plt.show()
```



```
plt.pie(quantity, labels = fruits, autopct = "%0.1f%", colors = ['green', 'red', 'blue', 'pink'] )  
plt.show()
```

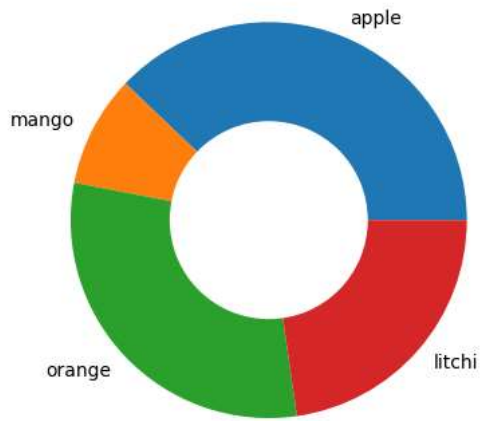


▾ Doughnut - chart

- A doughnut chart is a variation of a pie chart where there is a hole in the center, creating a "doughnut" shape. It is similar to a pie chart but allows for an additional level of categorization or comparison.

```
fruits = ['apple', 'mango', 'orange', 'litchi']  
quantity = [50, 12, 40, 30]
```

```
plt.pie(quantity, labels = fruits, radius = 1)  
plt.pie([1], colors = ['white'], radius = 0.5) # set quantity to any value, color = white and radius halaf of outer pe chart  
plt.show()
```



InternBix
Connect, learn and Earn