

❖ **13.10.2025 :**

- Introduction
- **Project Name:** BudgetWise AI based Expense Forecasting Tool

❖ **14.10.2025 :**

- **Artificial Intelligence (AI):** John McCarthy is the father of AI. He coined the term Artificial Intelligence in the year 1956. He defined AI as the science & engineering of making intelligent machines (Computer programs).
  - The theory & development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making and translation between languages.
  - Programming Languages for AI : Python, Java, R, Lisp
  - Artificial + Intelligence = Artificial Intelligence
  - Example: Number plate recognition, Machine playing chess, Siri
  - Artificial Narrow Intelligence (ANI): It is also known as weak AI. It involves applying AI only to specific tasks. Example: Alexa order a cheese pizza.
- **Machine Learning (ML):** Arthur Samuel is the father of ML. He coined the term Machine Learning in the year 1959. He defined Machine Learning as “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P, improves with experience E”.
  - **Types of ML:**
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
  - **Process of ML:** The ML process involves building a Predictive model that can be used to find a solution for a Problem Statement.
    - Define Objective
    - Data Gathering
    - Preparing Data
    - Data Exploration
    - Building a Model
    - Model Evaluation
    - Prediction
  - **Applications:**
    - Speech Recognition
    - Image Recognition
    - Traffic Prediction
    - Online Fraud Detection
    - Medical Diagnosis
  - **Life Cycle :**
    - Gathering Data
    - Data Preparation
    - Data Wrangling
    - Analyse Data
    - Train Model
    - Test Model

- Deployment
- Definitions :
  - **Algorithm:** A set of rules & statistical techniques used to learn patterns from data.
  - **Model:** It is trained by using a Machine Learning Algorithm.
  - **Predictor Variable:** It is a feature of the data that can be used to predict the output.
  - **Response Variable:** It is the feature of the output variable that needs to be predicted by using the predictor variable.
  - **Training Data:** The Machine Learning Model is built using the training data.
  - **Testing Data:** The Machine Learning Model is evaluated using the testing data.

❖ **15.10.2025 :**

- **Python NumPy:** NumPy stands for Numerical Python. It is the core library for numeric & scientific computing. It consists of multi-dimensional array objects and a collection of routines for processing those arrays.
- **Creating NumPy Array:**
  - Single-dimensional Array:

```
[4]: import numpy as np
n1 = np.array([10,20,30,40])
n1
[4]: array([10, 20, 30, 40])
```

- Multi-dimensional Array:

```
[6]: import numpy as np
n2 = np.array([[10,20,30,40], [40,30,20,10]])
n2
[6]: array([[10, 20, 30, 40],
       [40, 30, 20, 10]])
```

- **Initialising NumPy Array:**

- Initialising NumPy Array with zeros:

```
[9]: import numpy as np
n1 = np.zeros((1,2))
n1
[9]: array([[0., 0.]])
```

```
[10]: import numpy as np
n1 = np.zeros((5,5))
n1
```

```
[10]: array([[0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.]])
```

- Initialising NumPy Array with same number:

```
[14]: import numpy as np
n1 = np.full((2,2),10)
n1
```

```
[14]: array([[10, 10],
 [10, 10]])
```

- Initialising NumPy Array within a range:

```
[17]: import numpy as np
n1 = np.arange(10,20)
n1
```

```
[17]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
[18]: import numpy as np
n1 = np.arange(10,50,5)
n1
```

```
[18]: array([10, 15, 20, 25, 30, 35, 40, 45])
```

- Initialising NumPy Array with random numbers:

```
[21]: import numpy as np
n1 = np.random.randint(1,100,5)
n1
```

```
[21]: array([ 4,  5, 56, 77, 42], dtype=int32)
```

- NumPy-Shape:

- Checking the shape of NumPy Arrays:

```
[25]: import numpy as np
n1 = np.array([[1,2,3], [4,5,6]])
n1.shape
```

```
[25]: (2, 3)
```

```
[26]: n1.shape = (3,2)
n1.shape
```

```
[26]: (3, 2)
```

- **Joining NumPy Array:**

- **vstack()** : Vertical Stack

```
[29]: import numpy as np
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])
np.vstack((n1,n2))
```

```
[29]: array([[10, 20, 30],
 [40, 50, 60]])
```

- **hstack()** : Horizontal Stack

```
[31]: import numpy as np
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])
np.hstack((n1,n2))
```

```
[31]: array([10, 20, 30, 40, 50, 60])
```

- **column\_stack()** : Column Stack

```
[33]: import numpy as np
n1 = np.array([10,20,30])
n2 = np.array([40,50,60])
np.column_stack((n1,n2))
```

```
[33]: array([[10, 40],
 [20, 50],
 [30, 60]])
```

❖ **16.10.2025 :**

- **NumPy Intersection & Difference:**

```
[2]: import numpy as np
n1 = np.array([10,20,30,40,50,60])
n2 = np.array([50,60,70,80,90])
```

- **Intersection:**

```
[3]: np.intersect1d(n1,n2)
```

```
[3]: array([50, 60])
```

- Difference:

```
[4]: np.setdiff1d(n1,n2)  
[4]: array([10, 20, 30, 40])  
  
[5]: np.setdiff1d(n2,n1)  
[5]: array([70, 80, 90])
```

- NumPy Array Mathematics:

- Addition of NumPy Arrays:

```
[8]: import numpy as np  
n1 = np.array([10,20])  
n2 = np.array([30,40])  
  
np.sum([n1,n2])  
  
[8]: np.int64(100)  
  
[9]: np.sum([n1,n2], axis=0)  
[9]: array([40, 60])  
  
[10]: np.sum([n1,n2], axis=1)  
[10]: array([30, 70])
```

- Basic Addition:

```
[13]: import numpy as np  
n1 = np.array([10,20,30])  
n1 = n1 + 1  
n1  
  
[13]: array([11, 21, 31])
```

- Basic Subtraction:

```
[15]: import numpy as np  
n1 = np.array([10,20,30])  
n1 = n1 - 1  
n1  
  
[15]: array([ 9, 19, 29])
```

- Basic Multiplication:

```
[17]: import numpy as np  
n1 = np.array([10,20,30])  
n1 = n1 * 2  
n1
```

```
[17]: array([20, 40, 60])
```

- Basic Division:

```
[19]: import numpy as np  
n1 = np.array([10,20,30])  
n1 = n1 / 2  
n1
```

```
[19]: array([ 5., 10., 15.])
```

- Mean:

```
[21]: import numpy as np  
n1 = np.array([10,20,30,40,50,60])  
np.mean(n1)
```

```
[21]: np.float64(35.0)
```

- Median:

```
[24]: import numpy as np  
n1 = np.array([11,44,5,96,67,85])  
np.median(n1)
```

```
[24]: np.float64(55.5)
```

- Standard Deviation:

```
[26]: import numpy as np  
n1 = np.array([1,5,3,100,4,48])  
np.std(n1)
```

```
[26]: np.float64(36.59424666377065)
```

- NumPy Matrix:

```
[28]: import numpy as np  
n1 = np.array([[1,2,3], [4,5,6], [7,8,9]])  
n1
```

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

-

```
[29]: n1[0]
[29]: array([1, 2, 3])

[30]: n1[1]
[30]: array([4, 5, 6])
```

```
[31]: n1[:,1]
[31]: array([2, 5, 8])

[32]: n1[:,2]
[32]: array([3, 6, 9])
```

❖ 17.10.2025 :

- NumPy Transpose:

```
[28]: import numpy as np
n1 = np.array([[1,2,3], [4,5,6], [7,8,9]])
n1
[28]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])

    • Transpose:
[9]: n1.transpose()
[9]: array([[1, 4, 7],
             [2, 5, 8],
             [3, 6, 9]])
```

- Matrix Multiplication:

```
[11]: n1 = np.array([[1,2,3], [4,5,6], [7,8,9]])
n1
[11]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])

    •
[12]: n2 = np.array([[9,8,7], [6,5,4], [3,2,1]])
n2
[12]: array([[9, 8, 7],
             [6, 5, 4],
             [3, 2, 1]])
```

```
[13]: n1.dot(n2)
```

```
[13]: array([[ 30,  24,  18],
       [ 84,  69,  54],
       [138, 114,  90]])
```

```
[14]: n2.dot(n1)
```

```
[14]: array([[ 90, 114, 138],
       [ 54,  69,  84],
       [ 18,   24,   30]])
```

- **NumPy Save & Load:**

- **Save:**

```
[16]: import numpy as np
n1 = np.array([10,20,30,40,50,60])
np.save('my_numpy', n1)
```

- **Load:**

```
[17]: n2 = np.load('my_numpy.npy')
n2
```

```
[17]: array([10, 20, 30, 40, 50, 60])
```

- **Python Pandas:** Pandas stands for Panel Data. It is the core library for data manipulation & data analysis. It consists of single & multi-dimensional data structures for data manipulation.

- **Pandas Data Structures:**

- **Single-dimensional:**

- Series Object

- **Multi-dimensional:**

- Data-Frame

- **Pandas Series Object:** Series Object is one-dimensional labeled array.

```
[20]: import pandas as pd
s1 = pd.Series([1,2,3,4,5])
s1
```

```
[20]: 0    1
      1    2
      2    3
      3    4
      4    5
      dtype: int64
```

- **Changing Index:**

```
[22]: import pandas as pd  
s1 = pd.Series([1,2,3,4,5], index=['a','b','c','d','e'])  
s1
```

```
[22]: a    1  
      b    2  
      c    3  
      d    4  
      e    5  
      dtype: int64
```

- **Series Object from Dictionary:**

```
[24]: import pandas as pd  
pd.Series({'a':10, 'b':20, 'c':30})
```

```
[24]: a    10  
      b    20  
      c    30  
      dtype: int64
```

- **Changing index position:**

```
[27]: import pandas as pd  
pd.Series({'a':10, 'b':20, 'c':30}, index=['b','c','d','a'])
```

```
[27]: b    20.0  
      c    30.0  
      d    NaN  
      a    10.0  
      dtype: float64
```

- **Extracting Individual Elements:**

- **Extracting a single element:**

```
[30]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])  
s1[3]
```

```
[30]: np.int64(4)
```

- **Extracting a sequence of elements:**

```
[32]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])  
s1[:4]
```

```
[32]: 0    1  
      1    2  
      2    3  
      3    4  
      dtype: int64
```

- Extracting elements from back:

```
[34]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])
s1[-3:]
```

```
[34]: 6    7
      7    8
      8    9
      dtype: int64
```

❖ 21.10.2025 :

- Basic Math Operations on Series:

```
[13]: import pandas as panda
s1 = pd.Series([1,2,3,4,5,6,7,8,9])
```

- ▪ Adding a scalar value to Series Elements:

```
[15]: s1 + 5
[15]: 0    6
      1    7
      2    8
      3    9
      4   10
      5   11
      6   12
      7   13
      8   14
      dtype: int64
```

- Adding two Series Objects:

```
[18]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])
s2 = pd.Series([10,20,30,40,50,60,70,80,90])
s1 + s2
```

```
[18]: 0    11
      1    22
      2    33
      3    44
      4    55
      5    66
      6    77
      7    88
      8    99
      dtype: int64
```

- **Pandas Data-Frame:** Data-frame is a 2-dimensional labelled data-structure. It comprises of rows and columns.

- **Creating a Data-Frame:**

```
[22]: import pandas as pd
pd.DataFrame({"Name":['Bob', 'Sam', 'Anne'], "Marks":[76,25,92]})
```

	Name	Marks
0	Bob	76
1	Sam	25
2	Anne	92

- **DataFrame:**

```
[24]: df = pd.DataFrame({"Name":["Sam", "Anne", "Jennifer"], "Marks":[50,60,70]})
df
```

	Name	Marks
0	Sam	50
1	Anne	60
2	Jennifer	70

- **type():**

```
[25]: type(df)
```

```
[25]: pandas.core.frame.DataFrame
```

- **DataFrame In-Built Functions:**

```
[27]: import pandas as pd
ds = pd.read_csv('Iris.csv')
```

- **head():**

```
[28]: ds.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[30]: `ds.head(10)`

	<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>
<b>0</b>	1	5.1	3.5	1.4	0.2	Iris-setosa
<b>1</b>	2	4.9	3.0	1.4	0.2	Iris-setosa
<b>2</b>	3	4.7	3.2	1.3	0.2	Iris-setosa
<b>3</b>	4	4.6	3.1	1.5	0.2	Iris-setosa
<b>4</b>	5	5.0	3.6	1.4	0.2	Iris-setosa
<b>5</b>	6	5.4	3.9	1.7	0.4	Iris-setosa
<b>6</b>	7	4.6	3.4	1.4	0.3	Iris-setosa
<b>7</b>	8	5.0	3.4	1.5	0.2	Iris-setosa
<b>8</b>	9	4.4	2.9	1.4	0.2	Iris-setosa
<b>9</b>	10	4.9	3.1	1.5	0.1	Iris-setosa

- **tail():**

[29]: `ds.tail()`

	<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>
<b>145</b>	146	6.7	3.0	5.2	2.3	Iris-virginica
<b>146</b>	147	6.3	2.5	5.0	1.9	Iris-virginica
<b>147</b>	148	6.5	3.0	5.2	2.0	Iris-virginica
<b>148</b>	149	6.2	3.4	5.4	2.3	Iris-virginica
<b>149</b>	150	5.9	3.0	5.1	1.8	Iris-virginica

[31]: `ds.tail(10)`

	<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>
<b>140</b>	141	6.7	3.1	5.6	2.4	Iris-virginica
<b>141</b>	142	6.9	3.1	5.1	2.3	Iris-virginica
<b>142</b>	143	5.8	2.7	5.1	1.9	Iris-virginica
<b>143</b>	144	6.8	3.2	5.9	2.3	Iris-virginica
<b>144</b>	145	6.7	3.3	5.7	2.5	Iris-virginica
<b>145</b>	146	6.7	3.0	5.2	2.3	Iris-virginica
<b>146</b>	147	6.3	2.5	5.0	1.9	Iris-virginica
<b>147</b>	148	6.5	3.0	5.2	2.0	Iris-virginica
<b>148</b>	149	6.2	3.4	5.4	2.3	Iris-virginica
<b>149</b>	150	5.9	3.0	5.1	1.8	Iris-virginica

- **shape():**

[32]: `ds.shape`

[32]:	(150, 6)
-------	----------

- **describe():**

```
[33]: ds.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
<b>count</b>	150.000000	150.000000	150.000000	150.000000	150.000000
<b>mean</b>	75.500000	5.843333	3.054000	3.758667	1.198667
<b>std</b>	43.445368	0.828066	0.433594	1.764420	0.763161
<b>min</b>	1.000000	4.300000	2.000000	1.000000	0.100000
<b>25%</b>	38.250000	5.100000	2.800000	1.600000	0.300000
<b>50%</b>	75.500000	5.800000	3.000000	4.350000	1.300000
<b>75%</b>	112.750000	6.400000	3.300000	5.100000	1.800000
<b>max</b>	150.000000	7.900000	4.400000	6.900000	2.500000

- ❖ 22.10.2025 :

- ILOC:

```
[24]: ds.iloc[0:3, 0:2]
```

	Id	SepalLengthCm
0	1	5.1
1	2	4.9
2	3	4.7

```
[25]: ds.iloc[0:3, 2:5]
```

	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	3.5	1.4	0.2
1	3.0	1.4	0.2
2	3.2	1.3	0.2

- **LOC:**

	SepalWidthCm	PetalLengthCm	PetalWidthCm
135	3.0	6.1	2.3
136	3.4	5.6	2.4
137	3.1	5.5	1.8
138	3.0	4.8	1.8
139	3.1	5.4	2.1
140	3.1	5.6	2.4
141	3.1	5.1	2.3
142	2.7	5.1	1.9
143	3.2	5.9	2.3
144	3.3	5.7	2.5
145	3.0	5.2	2.3
146	2.5	5.0	1.9
147	3.0	5.2	2.0
148	3.4	5.4	2.3
149	3.0	5.1	1.8

- 

	SepalLengthCm	PetalLengthCm
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4
5	5.4	1.7

-

[38]: `ds.loc[135:150, ('SepalLengthCm', 'PetalLengthCm')]`

	SepalLengthCm	PetalLengthCm
135	7.7	6.1
136	6.3	5.6
137	6.4	5.5
138	6.0	4.8
139	6.9	5.4
140	6.7	5.6
141	6.9	5.1
142	5.8	5.1
143	6.8	5.9
144	6.7	5.7
145	6.7	5.2
146	6.3	5.0
147	6.5	5.2
148	6.2	5.4
149	5.9	5.1

- Dropping Columns:

[43]: `ds.drop('SepalLengthCm', axis=1)`

	Id	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	3.5	1.4	0.2	Iris-setosa
1	2	3.0	1.4	0.2	Iris-setosa
2	3	3.2	1.3	0.2	Iris-setosa
3	4	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	146	3.0	5.2	2.3	Iris-virginica
146	147	2.5	5.0	1.9	Iris-virginica
147	148	3.0	5.2	2.0	Iris-virginica
148	149	3.4	5.4	2.3	Iris-virginica
149	150	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

- **Dropping Rows:**

[44]:	<code>ds.drop([1,2,3], axis=0)</code>
[44]:	
0	Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species
1	5.1 3.5 1.4 0.2 Iris-setosa
2	5.0 3.6 1.4 0.2 Iris-setosa
3	5.4 3.9 1.7 0.4 Iris-setosa
4	4.6 3.4 1.4 0.3 Iris-setosa
5	5.0 3.4 1.5 0.2 Iris-setosa
6	...
7	...
8	...
...	...
145	146 6.7 3.0 5.2 2.3 Iris-virginica
146	147 6.3 2.5 5.0 1.9 Iris-virginica
147	148 6.5 3.0 5.2 2.0 Iris-virginica
148	149 6.2 3.4 5.4 2.3 Iris-virginica
149	150 5.9 3.0 5.1 1.8 Iris-virginica
147 rows × 6 columns	

- **More Pandas Functions:**

[1]:	<code>import pandas as pd pd.read_csv('iris.csv')</code>
[1]:	
0	Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species
1	5.1 3.5 1.4 0.2 Iris-setosa
2	4.9 3.0 1.4 0.2 Iris-setosa
3	4.7 3.2 1.3 0.2 Iris-setosa
4	4.6 3.1 1.5 0.2 Iris-setosa
5	5.0 3.6 1.4 0.2 Iris-setosa
...	...
145	146 6.7 3.0 5.2 2.3 Iris-virginica
146	147 6.3 2.5 5.0 1.9 Iris-virginica
147	148 6.5 3.0 5.2 2.0 Iris-virginica
148	149 6.2 3.4 5.4 2.3 Iris-virginica
149	150 5.9 3.0 5.1 1.8 Iris-virginica
150 rows × 6 columns	

- Minimum:

```
[57]: iris.min()

[57]:   Id              1
      SepalLengthCm    4.3
      SepalWidthCm     2.0
      PetalLengthCm    1.0
      PetalWidthCm     0.1
      Species          Iris-setosa
      dtype: object
```

- Maximum:

```
[58]: iris.max()

[58]:   Id             150
      SepalLengthCm   7.9
      SepalWidthCm    4.4
      PetalLengthCm   6.9
      PetalWidthCm    2.5
      Species         Iris-virginica
      dtype: object
```

❖ 23.10.2025 :

- Line Plot:

```
[2]: import numpy as np
      from matplotlib import pyplot as plt

[3]: x = np.arange(1,11)
      x

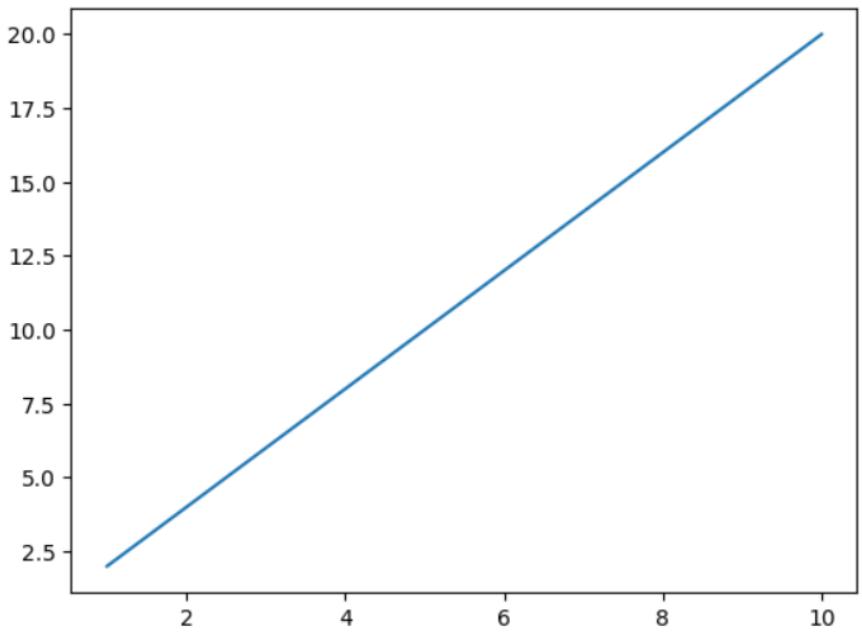
[3]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])

[4]: y = 2*x
      y

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

[5]:

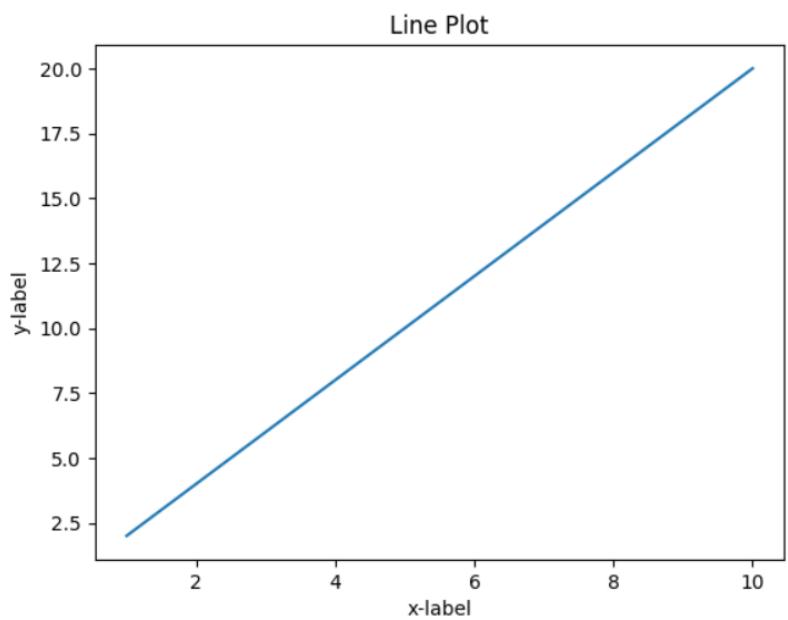
```
plt.plot(x,y)
plt.show()
```



o **Adding Title and Labels:**

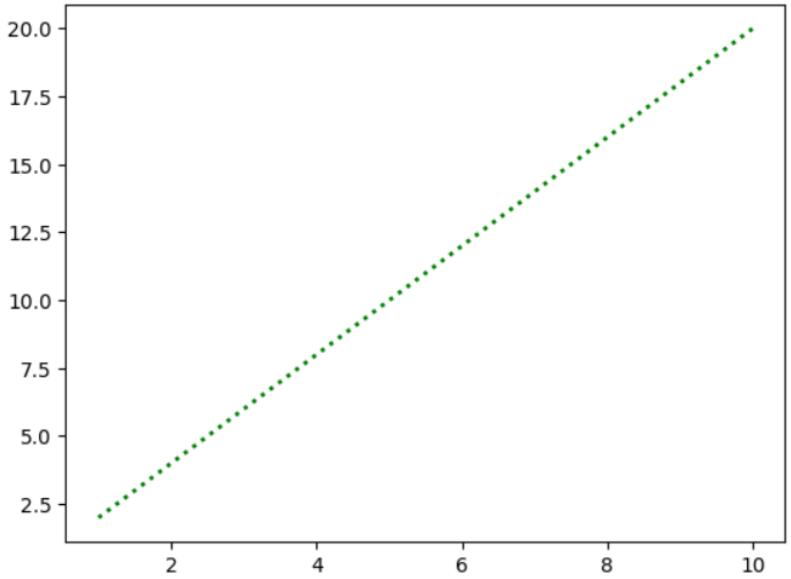
[8]:

```
plt.plot(x,y)
plt.title("Line Plot")
plt.xlabel("x-label")
plt.ylabel("y-label")
plt.show()
```



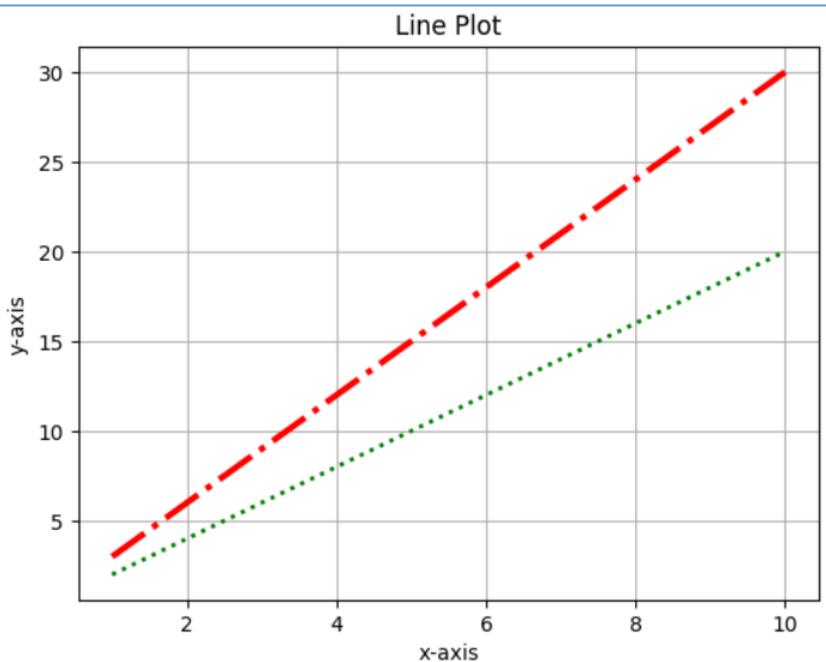
- **Changing Line Aesthetics:**

```
[10]: plt.plot(x, y, color='g', linestyle=':', linewidth=2)  
plt.show()
```



- **Adding two lines in the same plot:**

```
[15]: x = np.arange(1,11)  
y1 = 2*x  
y2 = 3*x  
  
plt.plot(x, y1, color='g', linestyle=':', linewidth=2)  
plt.plot(x, y2, color='r', linestyle='-.', linewidth=3)  
plt.title("Line Plot")  
plt.xlabel("x-axis")  
plt.ylabel("y-axis")  
plt.grid(True)  
plt.show()
```



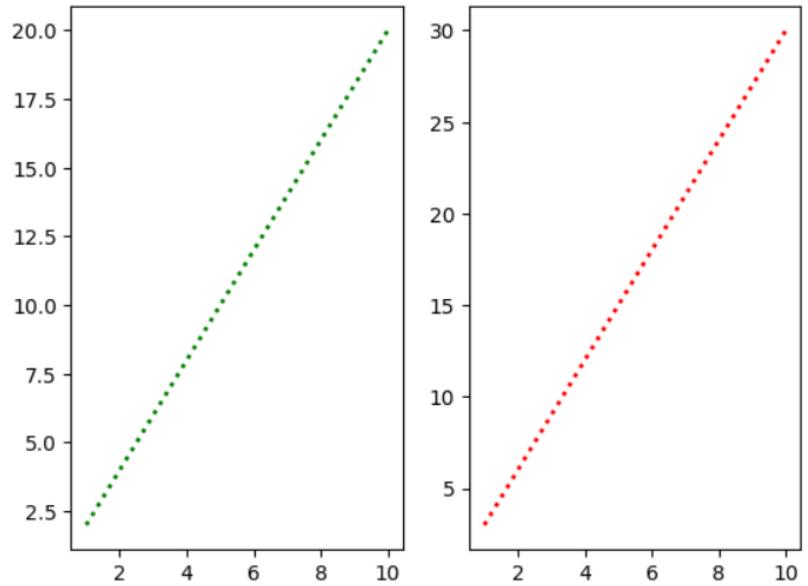
- Adding sub-plots:

```
[17]: x = np.arange(1,11)
y1 = 2*x
y2 = 3*x

plt.subplot(1,2,1)
plt.plot(x, y1, color='g', linestyle=':', linewidth=2)

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

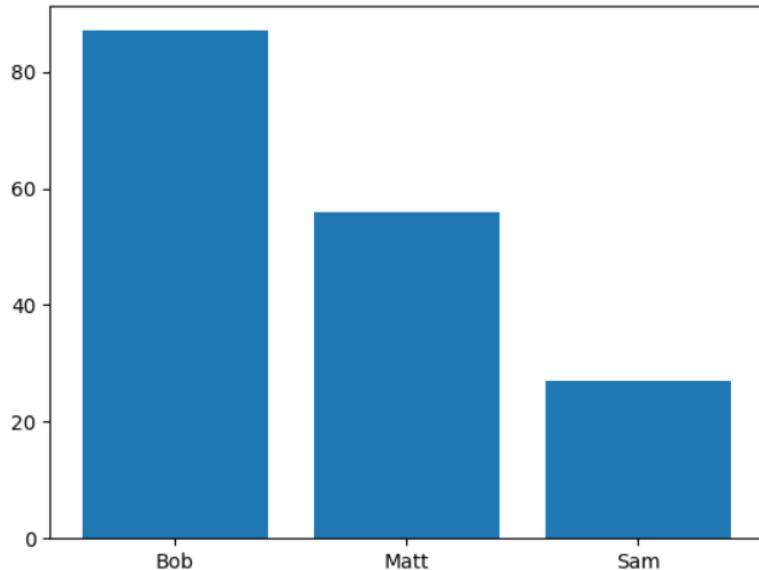
plt.show()
```



- Bar Plot:

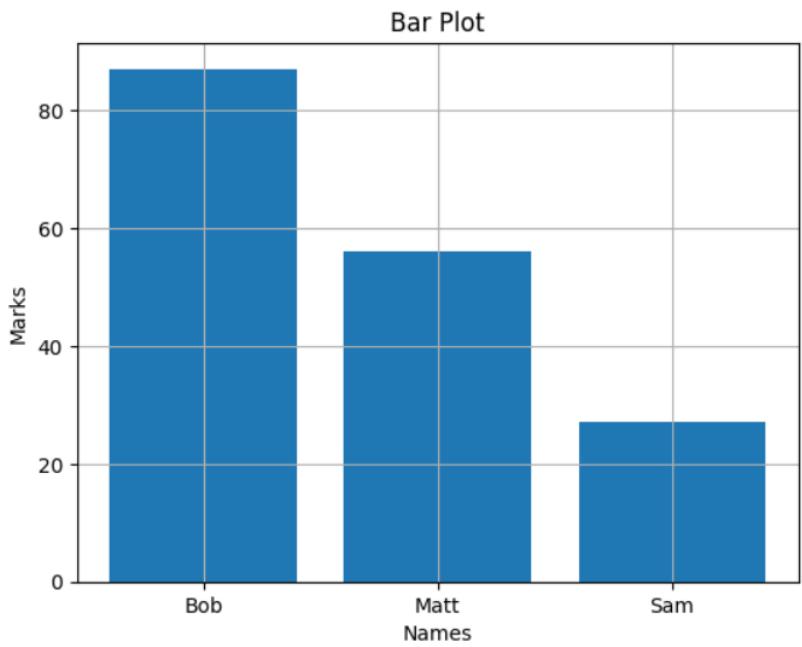
```
[19]: student = {"Bob":87, "Matt":56, "Sam":27}
names = list(student.keys())
values = list(student.values())
```

```
[20]: plt.bar(names, values)
plt.show()
```



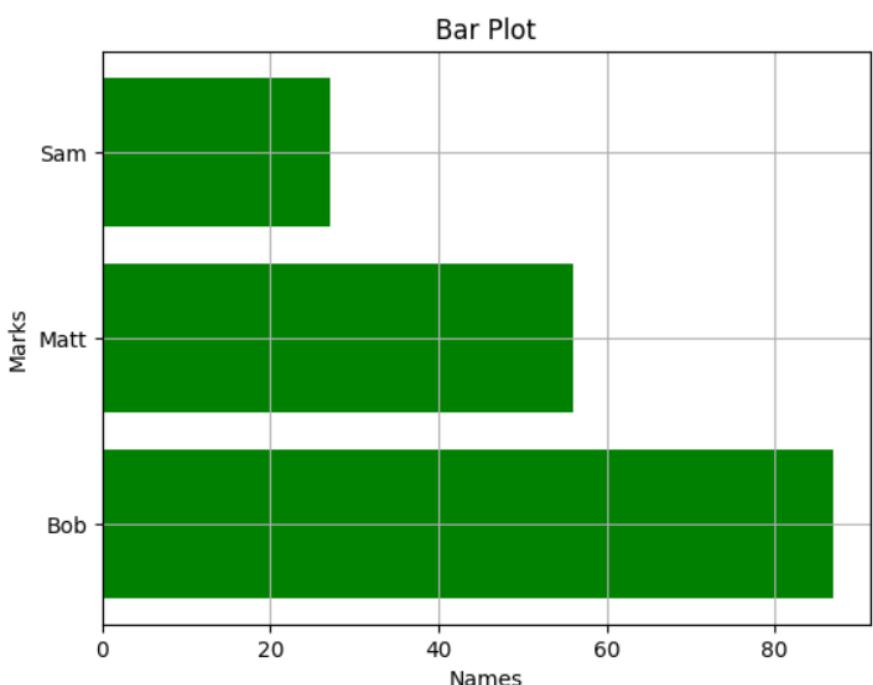
- **Adding Title and Labels:**

```
[22]: plt.bar(names, values)
plt.title("Bar Plot")
plt.xlabel("Names")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```



- **Horizontal Bar Plot:**

```
[24]: plt.bars(names, values, color='g')
plt.title("Bar Plot")
plt.xlabel("Names")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```

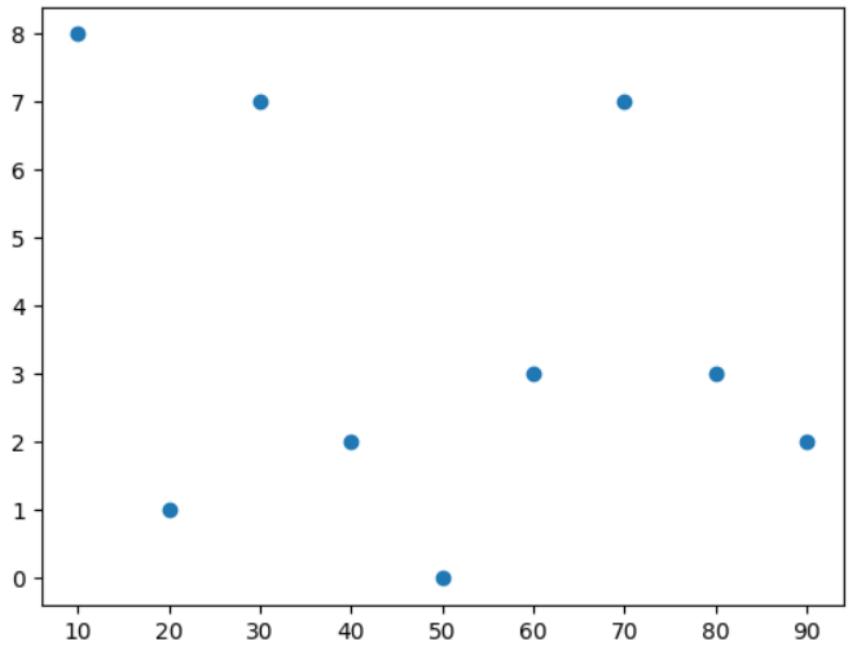


❖ 24.10.2025 :

- Scatter Plot:

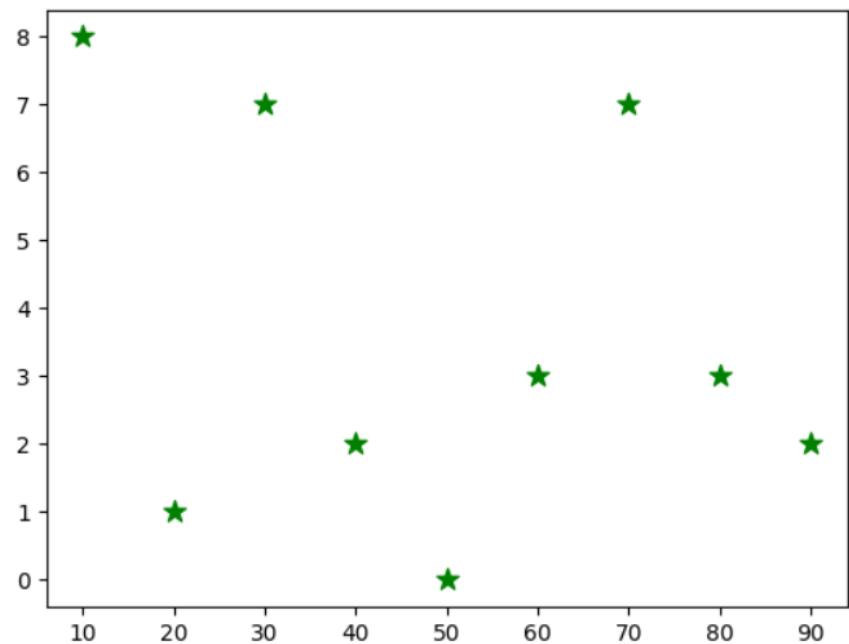
- Creating a basic scatter-plot:

```
[7]: x = [10,20,30,40,50,60,70,80,90]
a = [8,1,7,2,0,3,7,3,2]
plt.scatter(x,a)
plt.show()
```



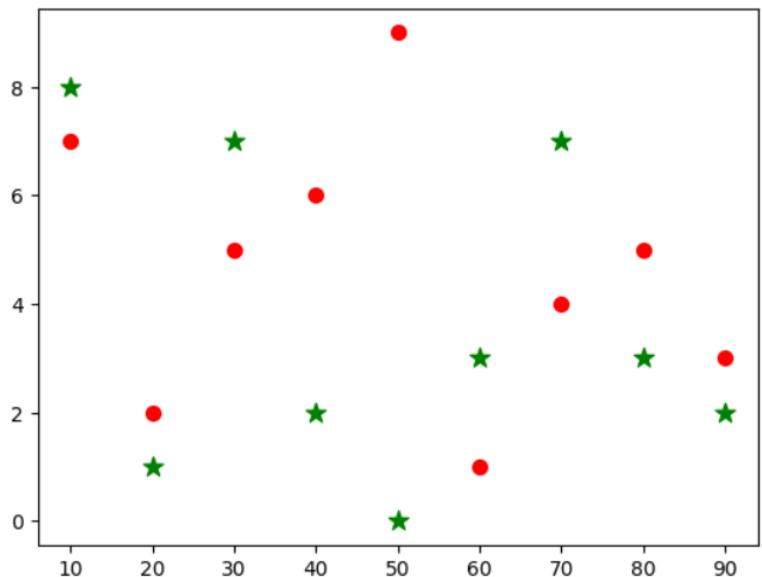
- Changing Mark Aesthetics:

```
[8]: x = [10,20,30,40,50,60,70,80,90]
a = [8,1,7,2,0,3,7,3,2]
plt.scatter(x, a, marker="*", c="g", s=100)
plt.show()
```



○ Adding two markers in the same plot:

```
[10]: x = [10,20,30,40,50,60,70,80,90]
a = [8,1,7,2,0,3,7,3,2]
b = [7,2,5,6,9,1,4,5,3]
plt.scatter(x, a, marker="*", c="g", s=100)
plt.scatter(x, b, marker=".", c="r", s=200)
plt.show()
```



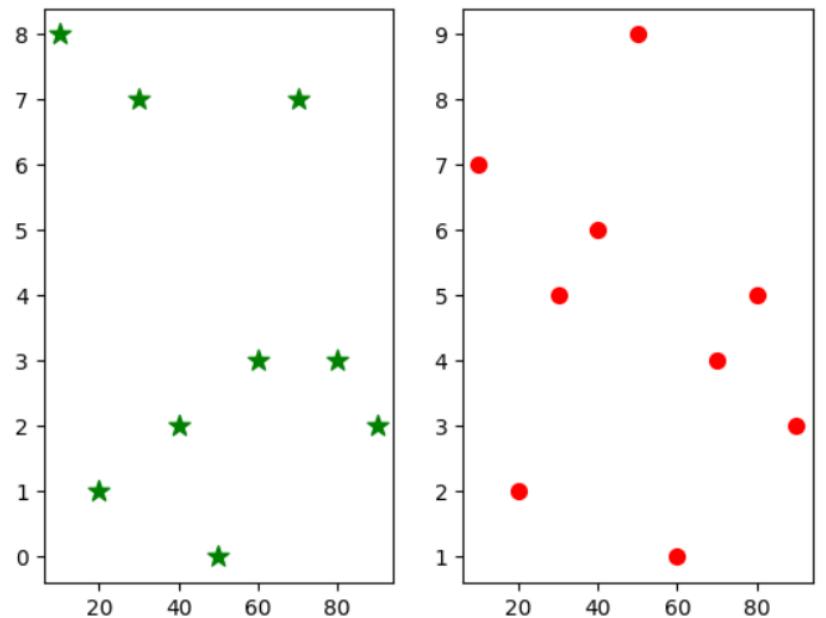
○ Adding sub-plots:

```
[14]: x = [10,20,30,40,50,60,70,80,90]
a = [8,1,7,2,0,3,7,3,2]
b = [7,2,5,6,9,1,4,5,3]

plt.subplot(1,2,1)
plt.scatter(x, a, marker="*", c="g", s=100)

plt.subplot(1,2,2)
plt.scatter(x, b, marker=".", c="r", s=200)

plt.show()
```



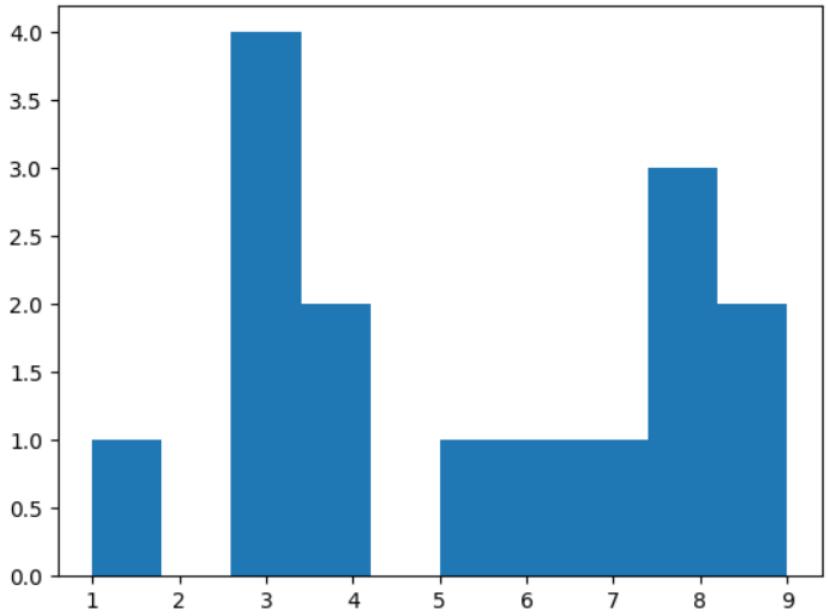
- **Histogram:**

- **Creating Data:**

```
[18]: #Creating data  
data = [1,3,3,3,3,9,9,5,4,4,8,8,8,6,7]
```

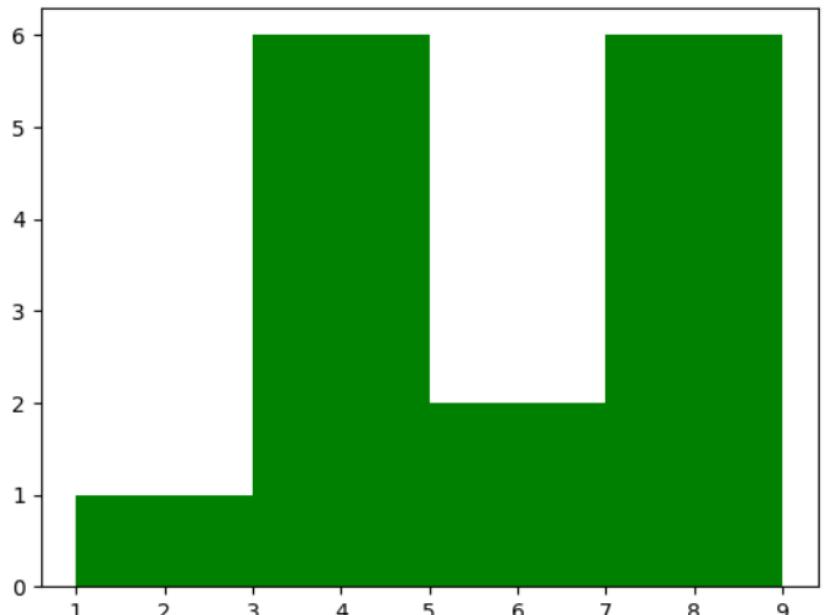
- **Making Histogram:**

```
[22]: plt.hist(data)  
plt.show()
```



- **Changing Aesthetics:**

```
[28]: plt.hist(data, color="g", bins=4)  
plt.show()
```

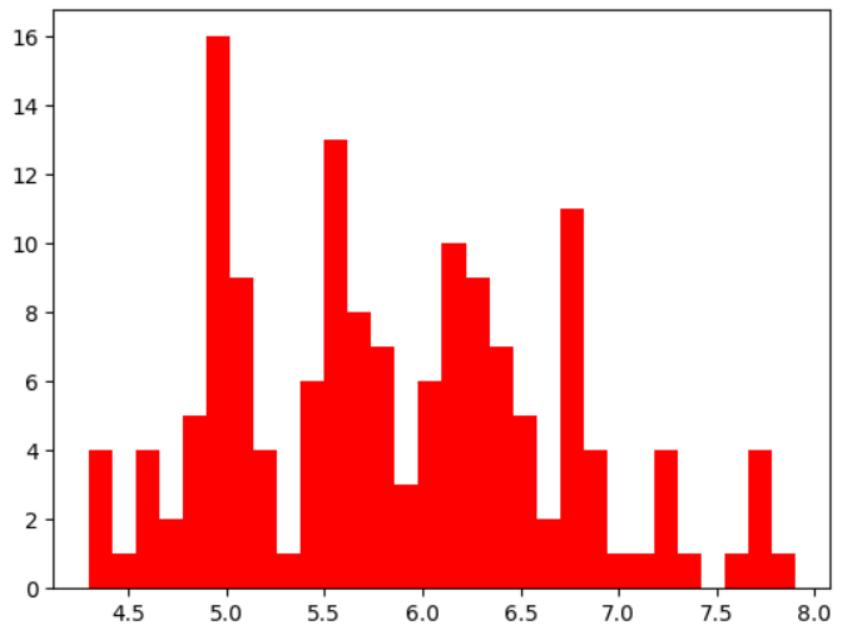


- Working with a dataset:

```
[34]: iris = pd.read_csv('iris.csv')
iris.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
[39]: plt.hist(iris['SepalLengthCm'], bins=30, color='r')
plt.show()
```



❖ 27.10.2025 :

- **Box Plot:**

- **Creating data:**

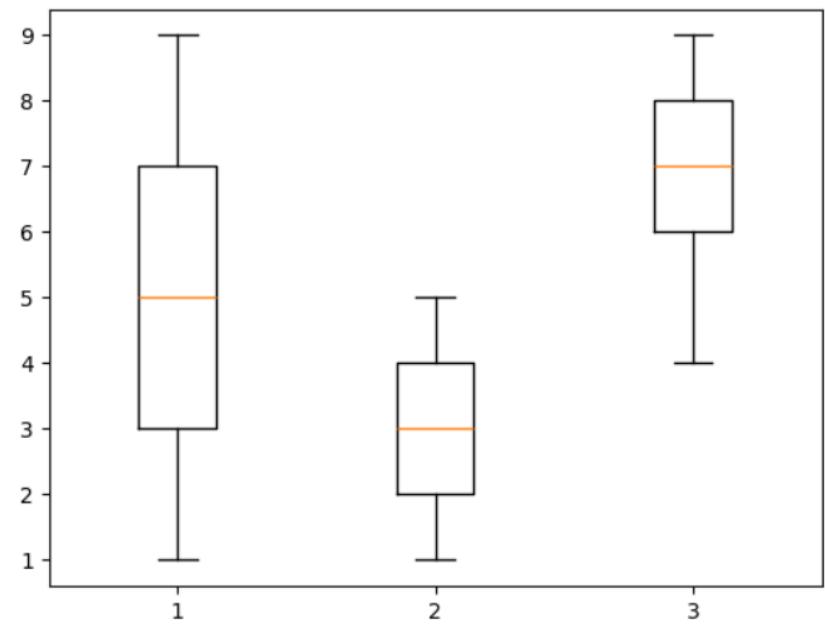
```
[5]: import matplotlib.pyplot as plt

[6]: one = [1,2,3,4,5,6,7,8,9]
      two = [1,2,3,4,5,4,3,2,1]
      three = [6,7,8,9,8,7,6,5,4]

      data = list([one,two,three])
```

- **Making Plot:**

```
[9]: # Box-Plot
      plt.boxplot(data)
      plt.show()
```



- **Violin Plot:**

- **Creating data:**

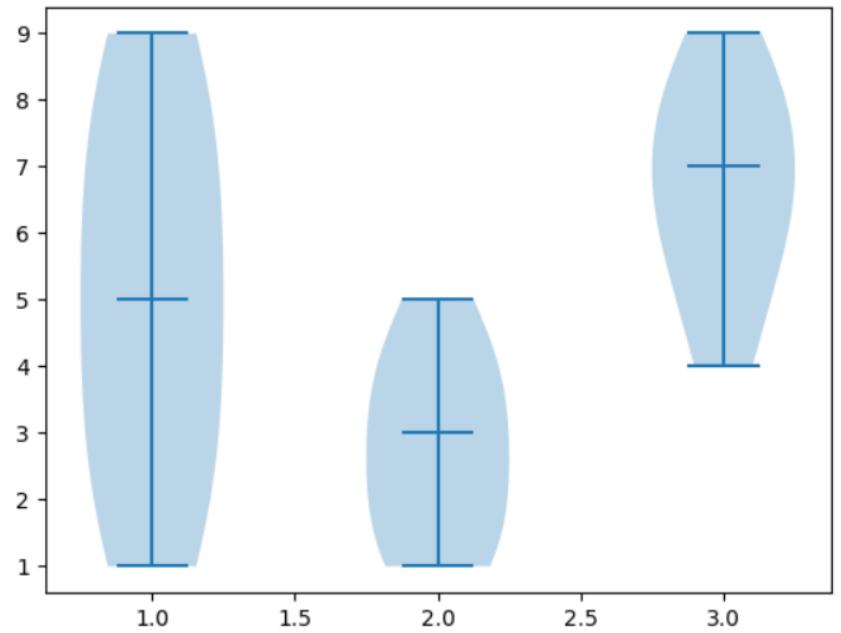
```
[5]: import matplotlib.pyplot as plt

[6]: one = [1,2,3,4,5,6,7,8,9]
      two = [1,2,3,4,5,4,3,2,1]
      three = [6,7,8,9,8,7,6,5,4]

      data = list([one,two,three])
```

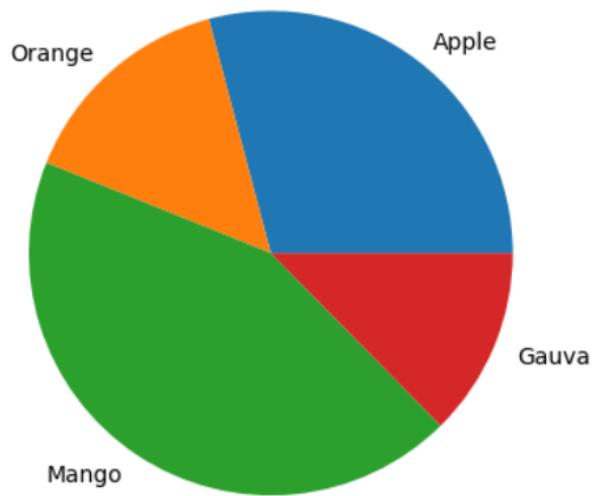
- **Making Plot:**

```
[10]: # Violin-Plot  
plt.violinplot(data, showmedians=True)  
plt.show()
```

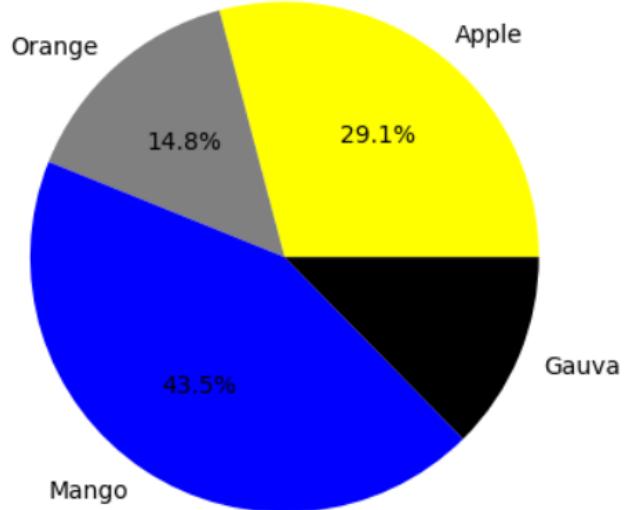


- **Pie Chart:**

```
[11]: # Pie-Chart  
fruit = ['Apple', 'Orange', 'Mango', 'Gauva']  
quantity = [67, 34, 100, 29]  
  
plt.pie(quantity, labels=fruit)  
plt.show()
```



```
[12]: plt.pie(quantity, labels=fruit, autopct='%.1f%%',
           colors=['yellow', 'grey', 'blue', 'black' ] )
plt.show()
```

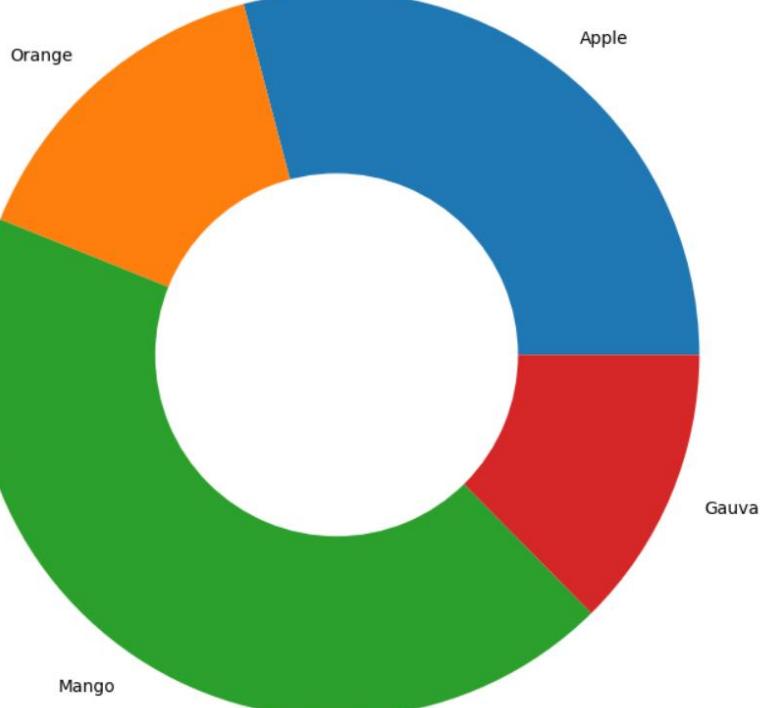


- **Doughnut Chart:**

```
[3]: # Doughnut-Chart

fruit = ['Apple', 'Orange', 'Mango', 'Gauva']
quantity = [67, 34, 100, 29]

plt.pie(quantity, labels=fruit, radius=2)
plt.pie([1], colors=['w'], radius=1)
plt.show()
```



- SeaBorn Line Plot:

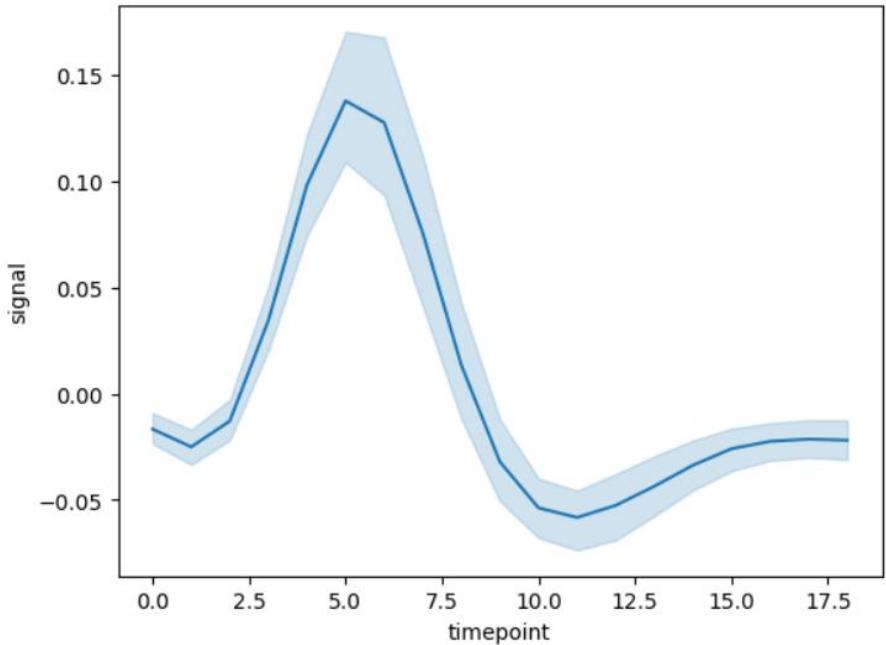
```
[14]: # SeaBorn Line Plot

import seaborn as sns
from matplotlib import pyplot as plt

fmri = sns.load_dataset("fmri")
fmri.head()
```

```
[14]:   subject  timepoint  event  region      signal
  0     s13        18  stim  parietal -0.017552
  1      s5         14  stim  parietal -0.080883
  2     s12        18  stim  parietal -0.081033
  3     s11        18  stim  parietal -0.046134
  4     s10        18  stim  parietal -0.037970
```

```
[17]: sns.lineplot(x="timepoint", y="signal", data=fmri)
plt.show()
```



❖ 28.10.2025 :

- **Seaborn Line Plot:** Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

```
[2]: # seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

[3]: # Visualizing statistical relationships

[4]: ds = sns.load_dataset('tips')

[5]: ds.head()

[5]:
   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

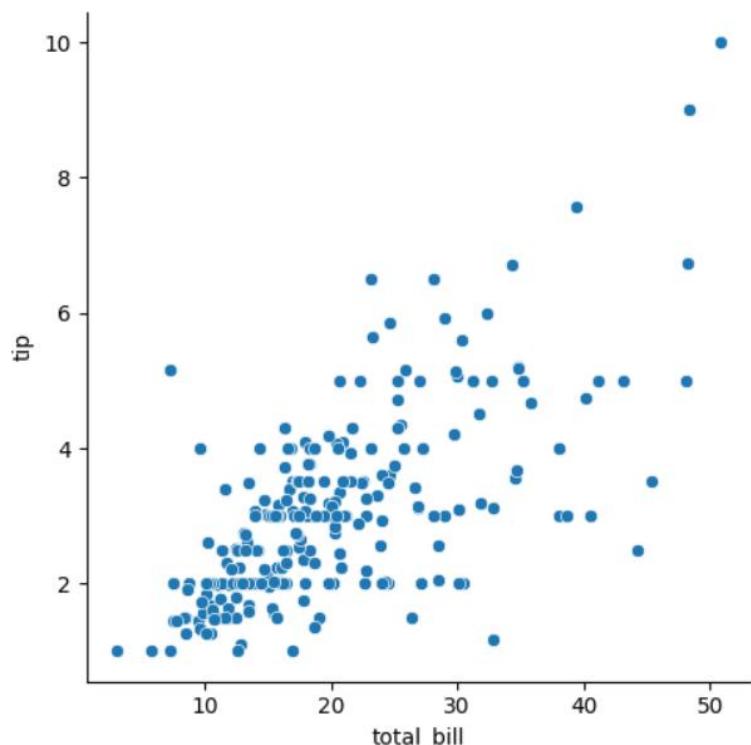
[6]: ds.shape

[6]: (244, 7)
```

○ Relating variables with scatter plots:

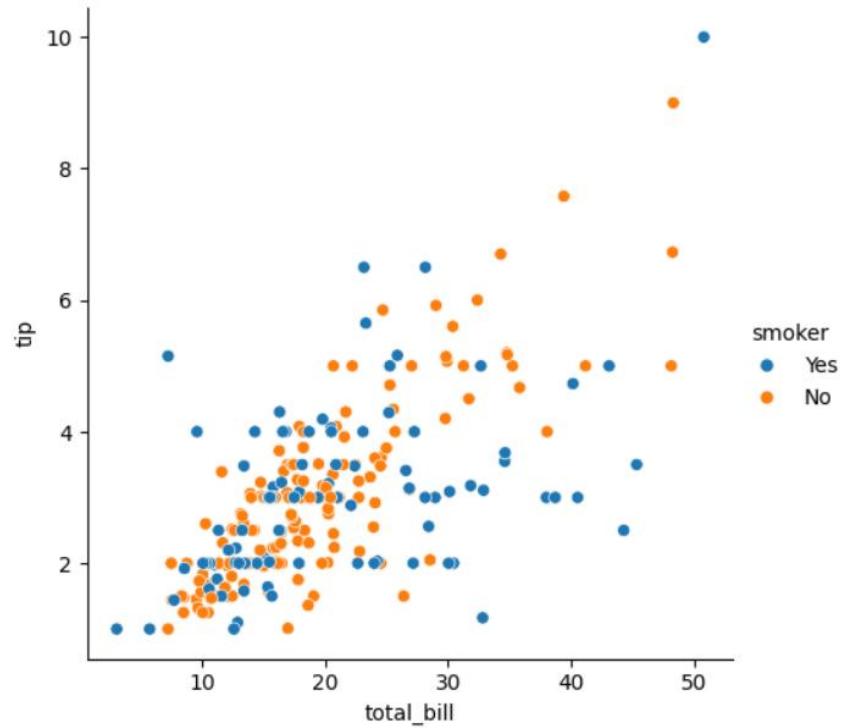
```
[8]: #Relating variables with scatter plots.
sns.relplot(data=ds, x='total_bill', y='tip')

[8]: <seaborn.axisgrid.FacetGrid at 0x1968d114110>
```



```
[9]: sns.relplot(data=ds, x='total_bill', y='tip', hue='smoker')
```

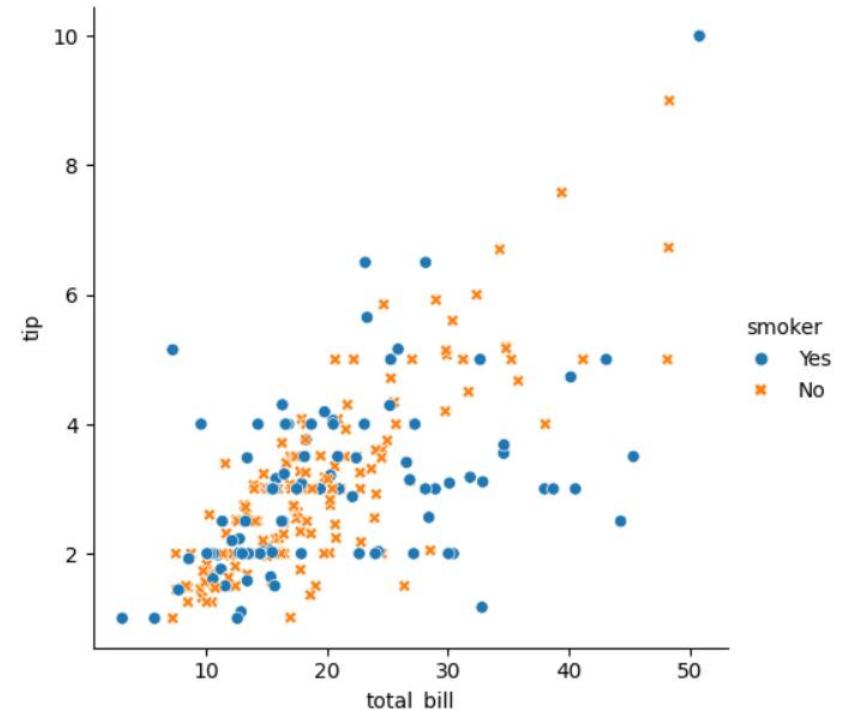
```
[9]: <seaborn.axisgrid.FacetGrid at 0x1968d1a4050>
```



- **Marker:**

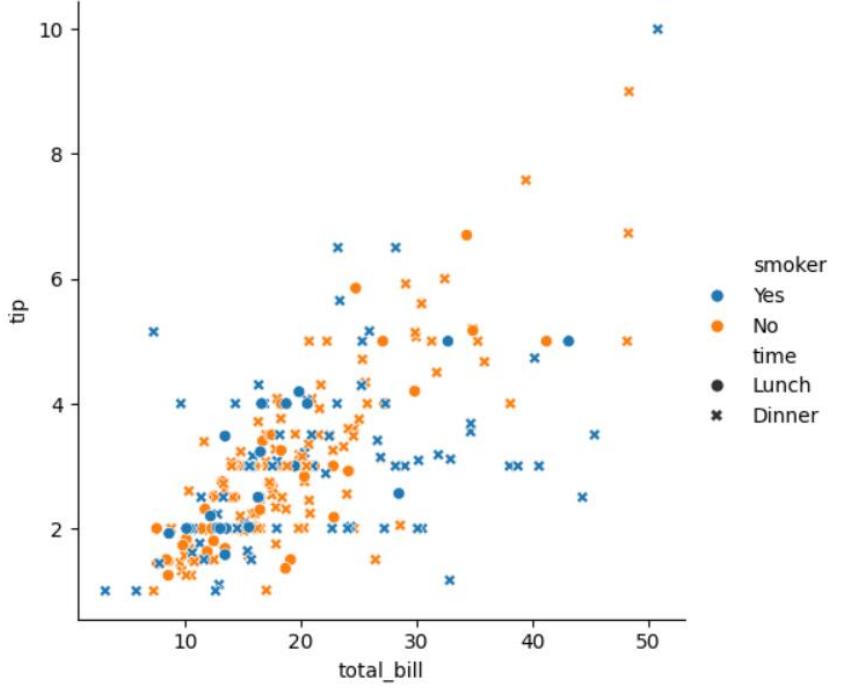
```
[11]: # Marker  
sns.relplot(data=ds, x='total_bill', y='tip', hue='smoker', style='smoker')
```

```
[11]: <seaborn.axisgrid.FacetGrid at 0x1968d245940>
```

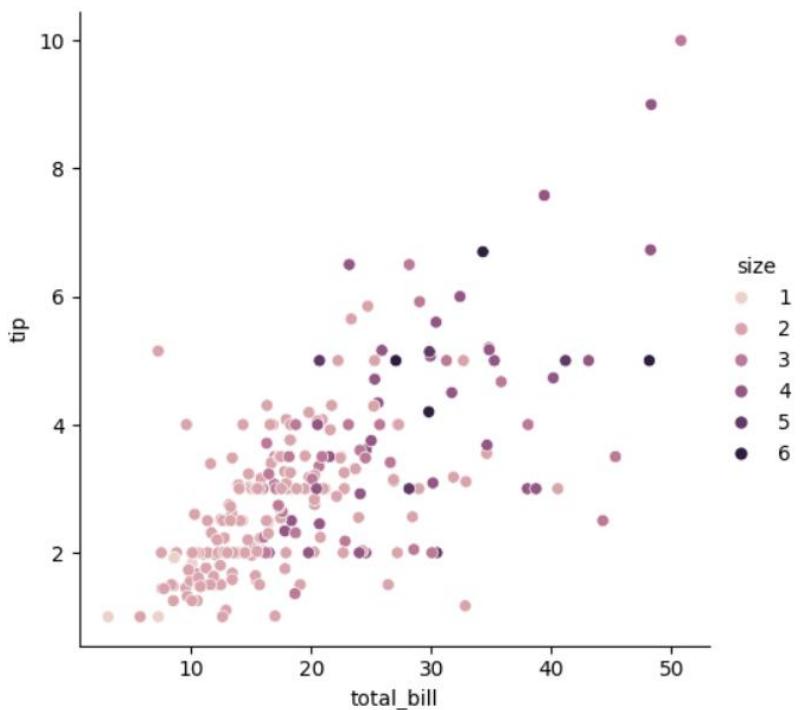


○ Hue:

```
[12]: sns.relplot(  
    data=ds,  
    x='total_bill', y='tip',  
    hue='smoker', style='time'  
)  
[12]: <seaborn.axisgrid.FacetGrid at 0x1968d150380>
```



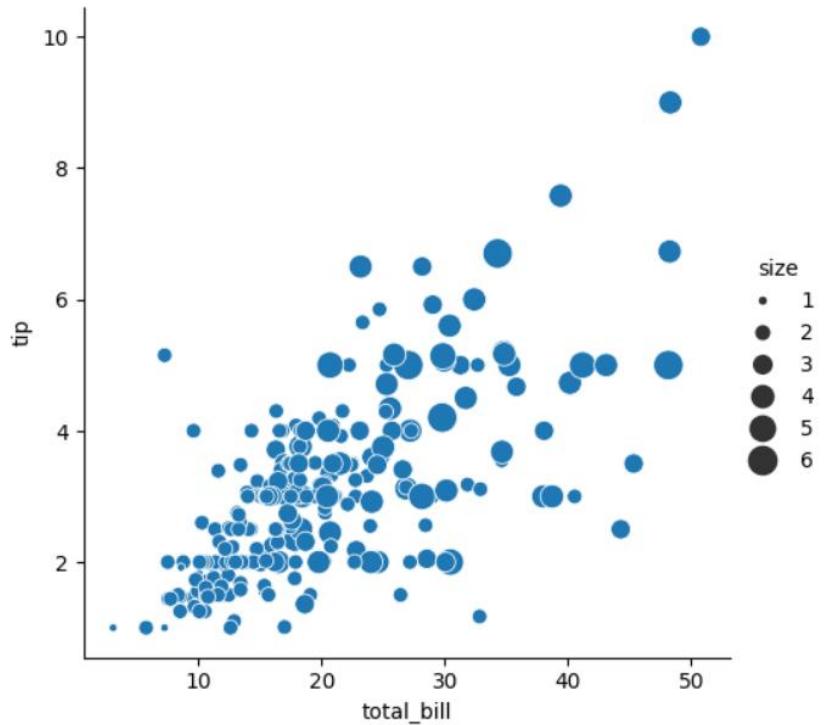
```
[16]: sns.relplot(  
    data=ds, x='total_bill', y='tip', hue='size'  
)  
[16]: <seaborn.axisgrid.FacetGrid at 0x19690cb84a0>
```



- **Size:**

```
[17]: sns.relplot(  
    data=ds, x='total_bill', y='tip',  
    size='size', sizes=(15,200)  
)
```

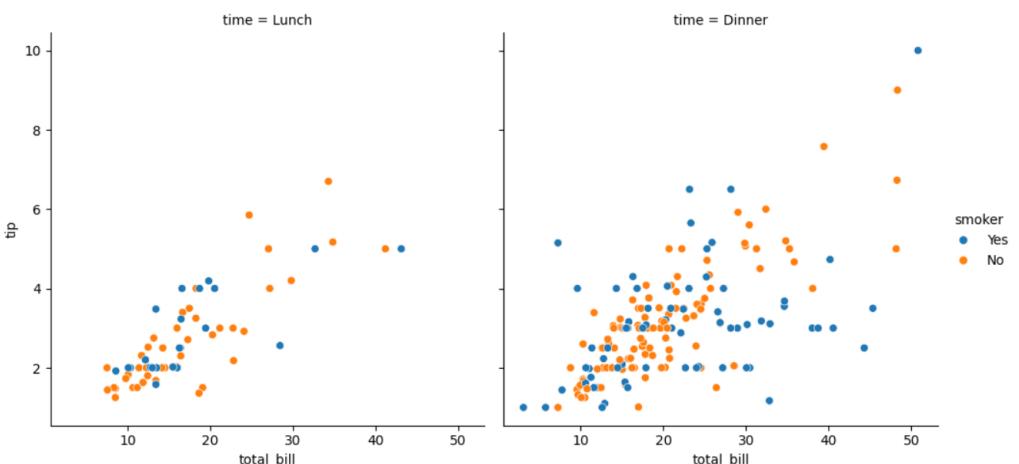
```
[17]: <seaborn.axisgrid.FacetGrid at 0x1968d253bf0>
```



- **Col:**

```
[18]: sns.relplot(  
    data=ds, x='total_bill', y='tip',  
    hue='smoker', col='time'  
)
```

```
[18]: <seaborn.axisgrid.FacetGrid at 0x19690e35a30>
```



❖ 29.10.2025 :

- SeaBorn Line Plot:

```
[2]: # seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

[22]: # More complex datasets will have multiple measurements for the same value of the x variable.

[4]: fmri = sns.load_dataset('fmri')

[5]: fmri.head()

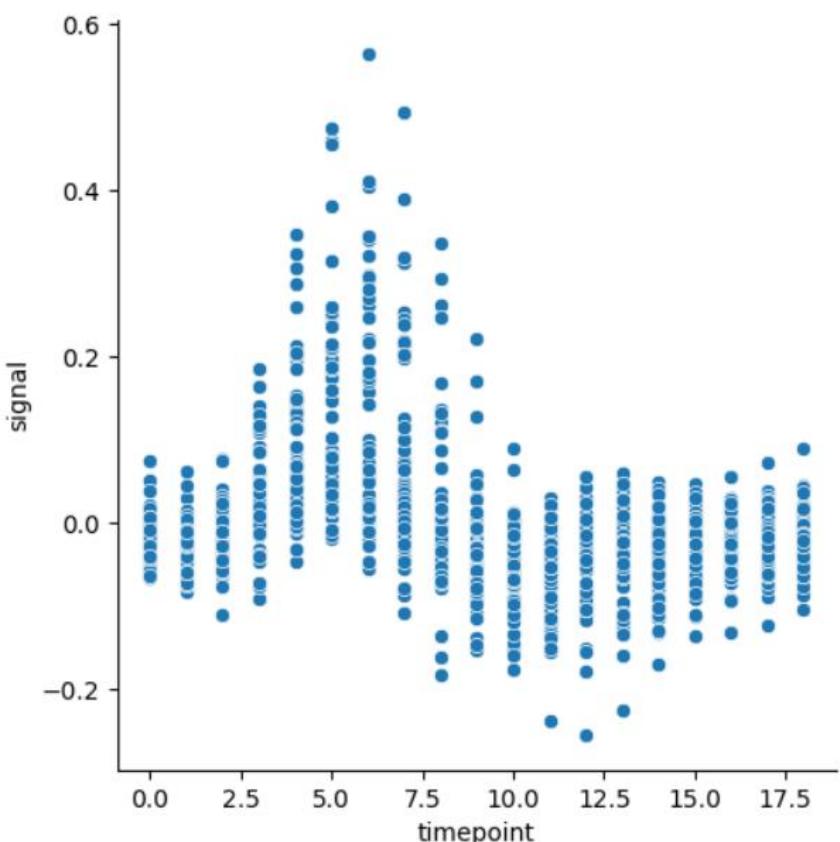
[5]:
  subject  timepoint  event  region    signal
0     s13        18  stim  parietal -0.017552
1      s5        14  stim  parietal -0.080883
2     s12        18  stim  parietal -0.081033
3     s11        18  stim  parietal -0.046134
4     s10        18  stim  parietal -0.037970

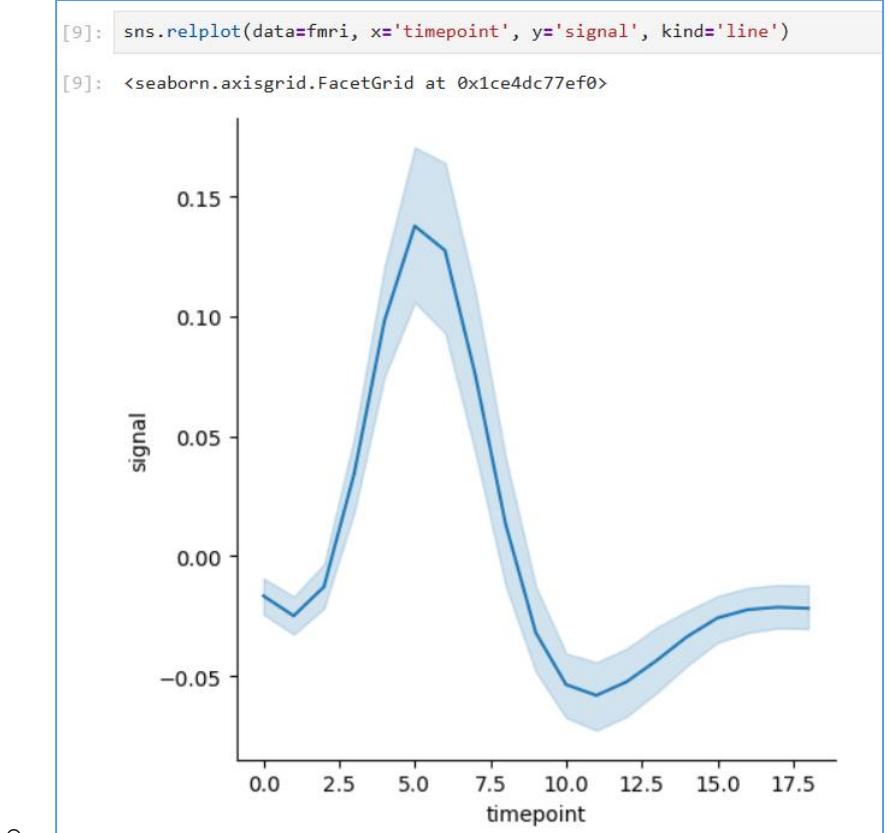
[6]: fmri.shape

[6]: (1064, 5)
```

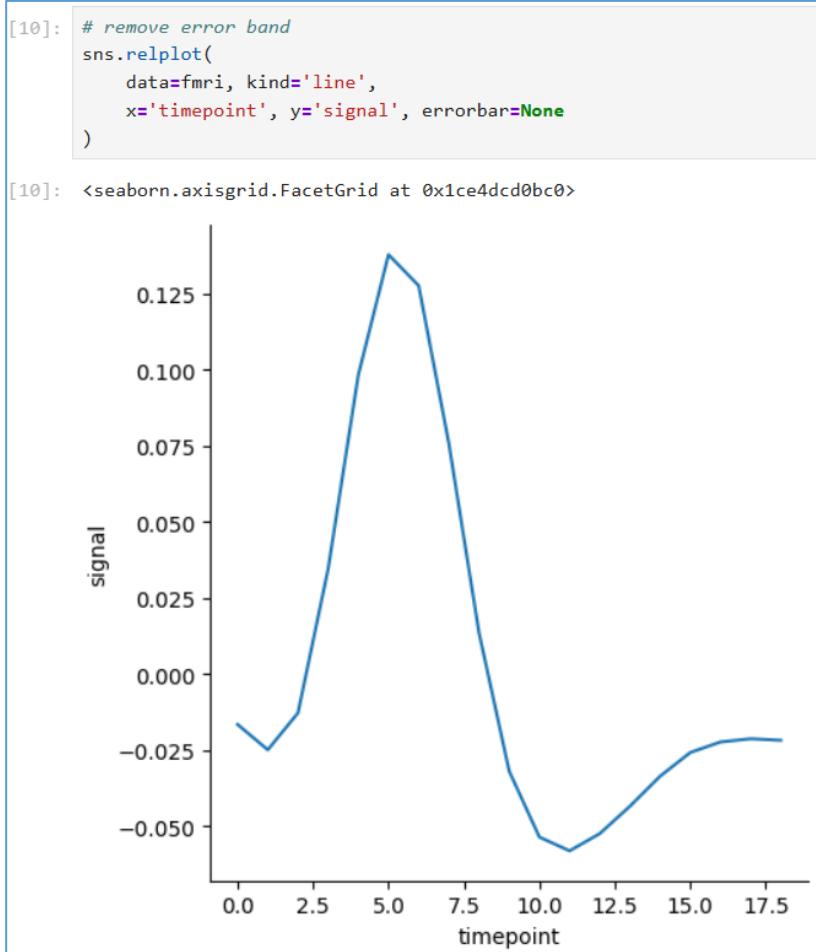
```
[8]: sns.relplot(data=fmri, x='timepoint', y='signal')
```

```
[8]: <seaborn.axisgrid.FacetGrid at 0x1ce4da68f20>
```

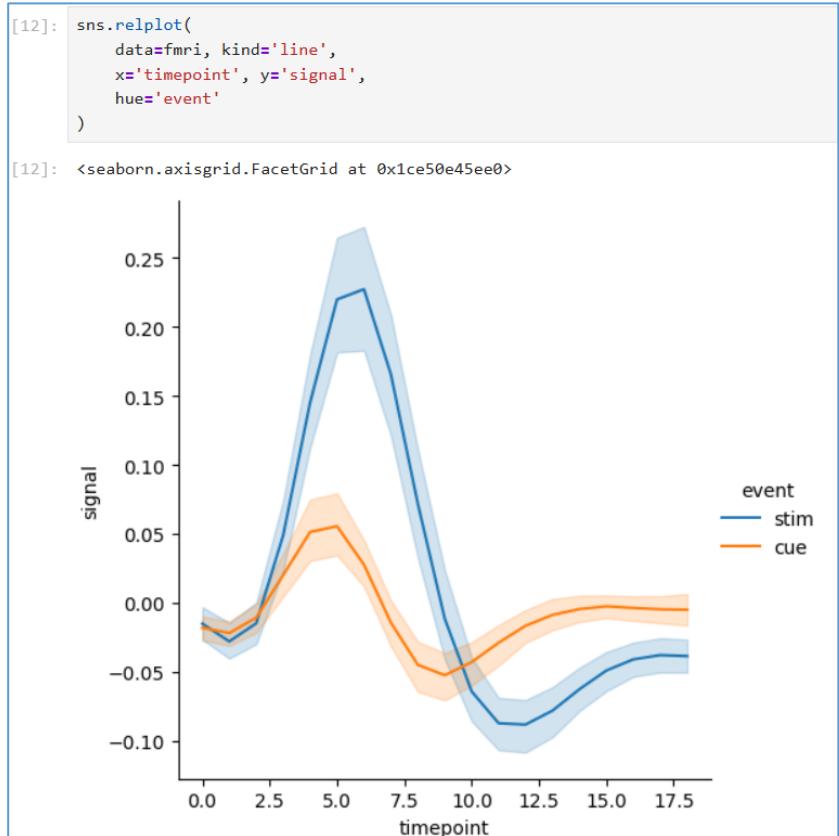




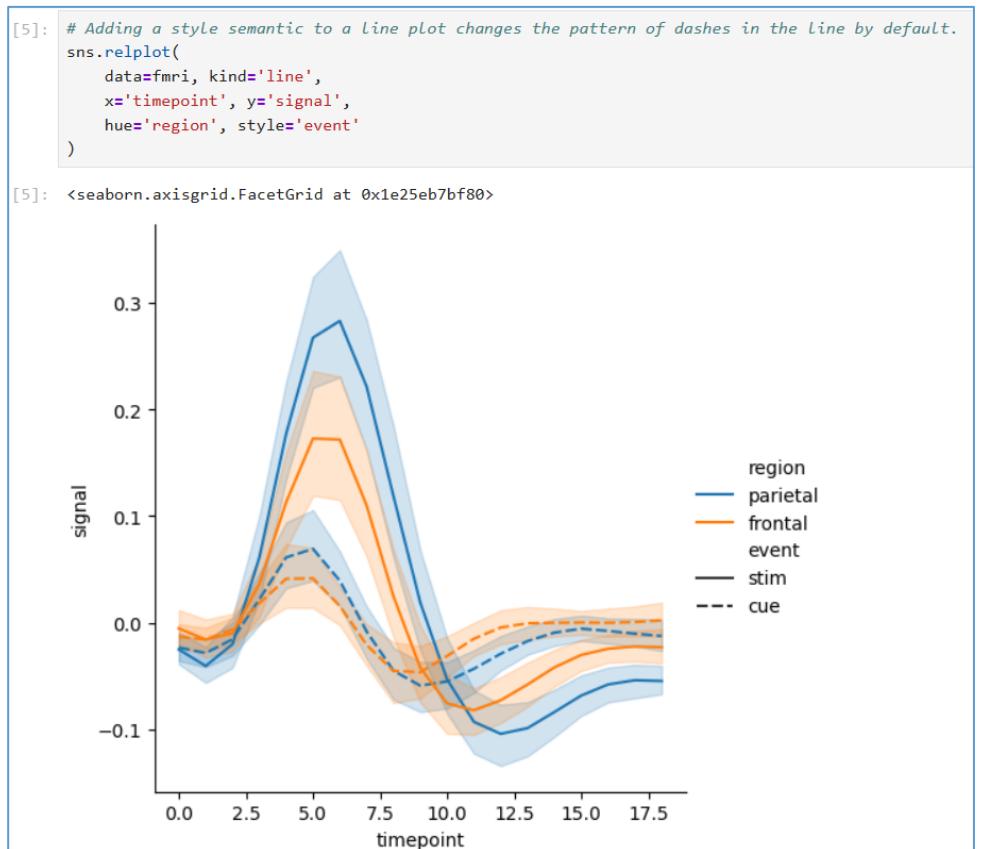
○ Remove Error Band:



- **Adding a hue semantic with 2 level splits:** They plot into two lines & error bands.



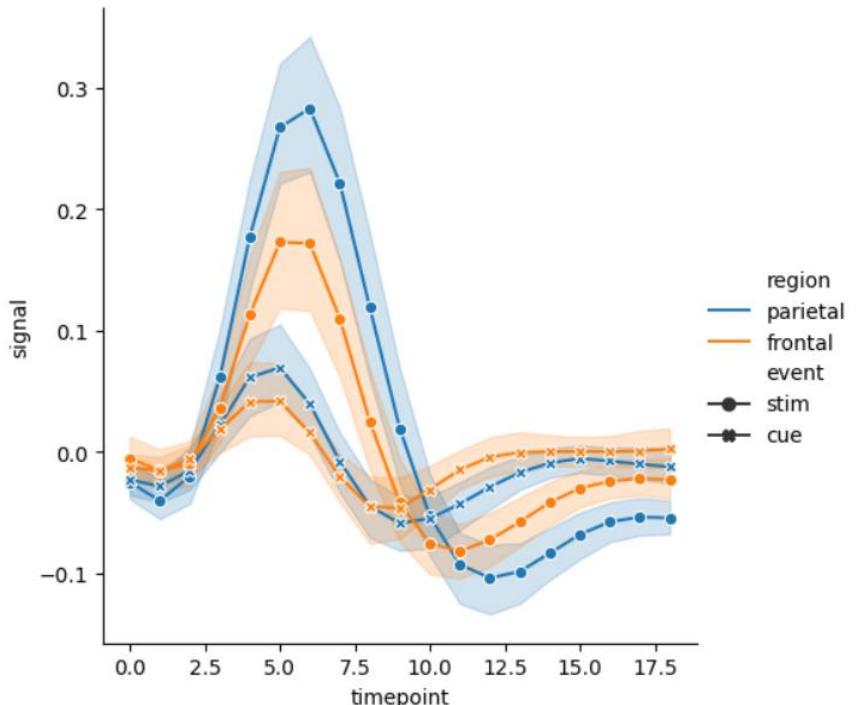
- **Adding a style semantic to a line plot changes the pattern of dashes in the line by default.**



- Identify subsets by the markers used at each observation:

```
[14]: sns.relplot(
    data=fMRI, kind='line',
    x='timepoint', y='signal',
    hue='region', style='event',
    dashes=False, markers=True
)
```

```
[14]: <seaborn.axisgrid.FacetGrid at 0x1ce50f60f20>
```



### ❖ 30.10.2025 :

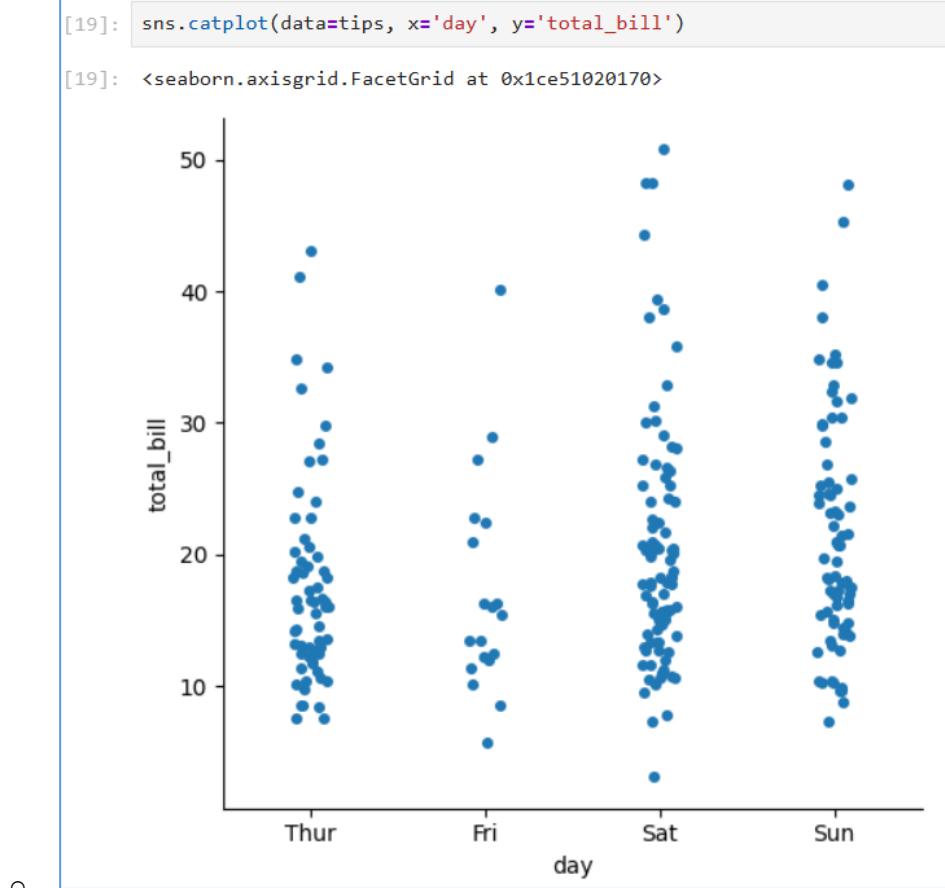
- **Categorical Scatter Plot:** The default representation of the data in catplot() uses a scatterplot. There are actually two different categorical scatter plots in seaborn.

```
[16]: # Categorical scatterplots
#The default representation of the data in catplot() uses a scatterplot.
#There are actually two different categorical scatter plots in seaborn.
```

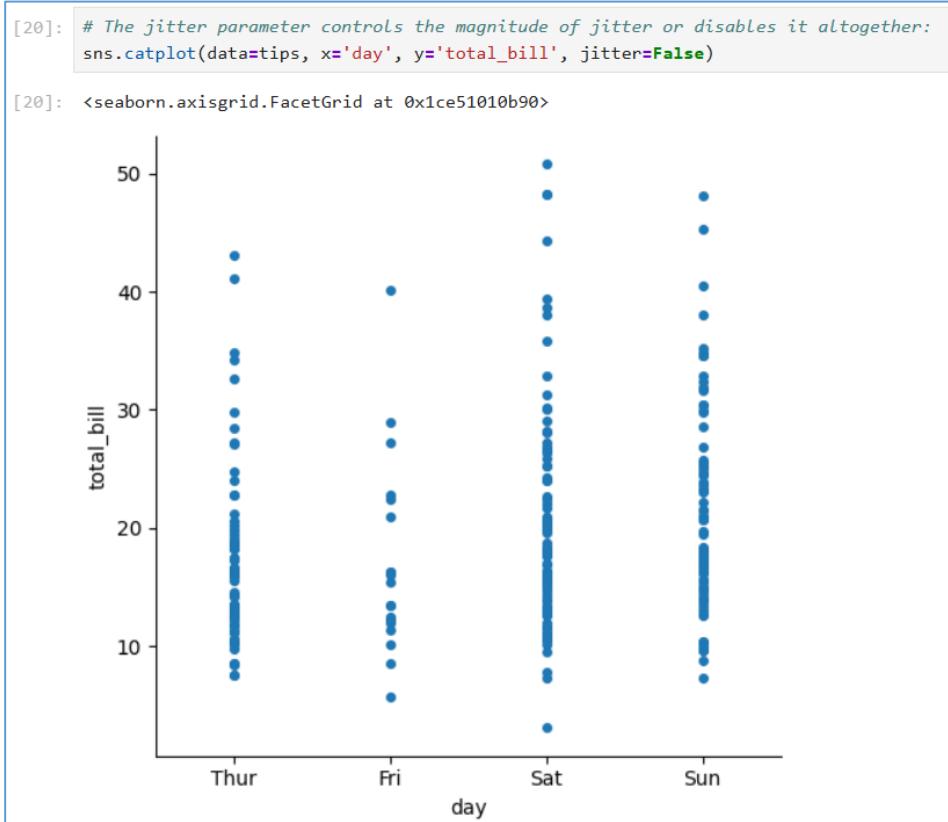
```
[17]: import seaborn as sns
```

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

	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



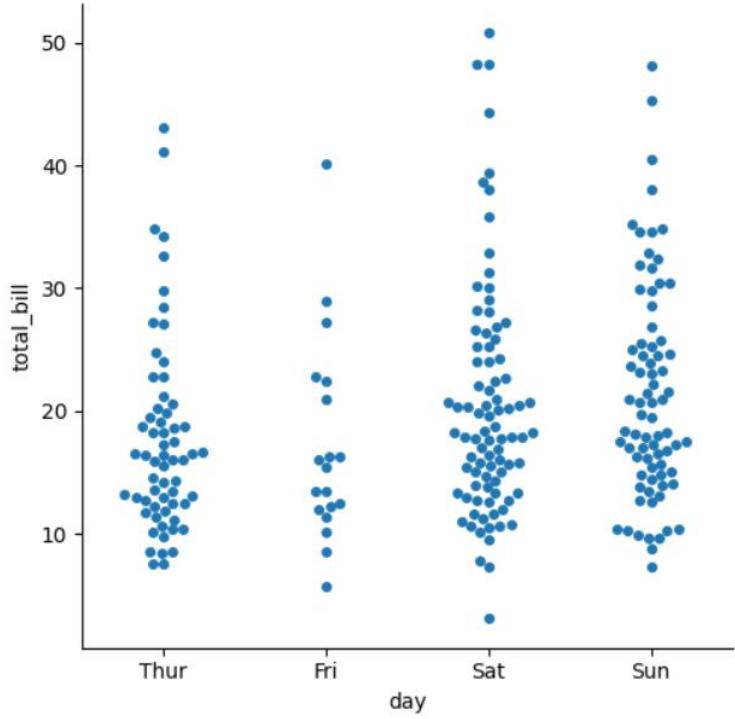
- **Jitter parameters:** The jitter parameter controls the magnitude of jitter or disables it altogether.



- **Swarm Plot:** Prevent from overlapping.

```
[21]: # prevent from overlapping (swarm plot)
sns.catplot(data=tips, x='day', y='total_bill', kind='swarm')
```

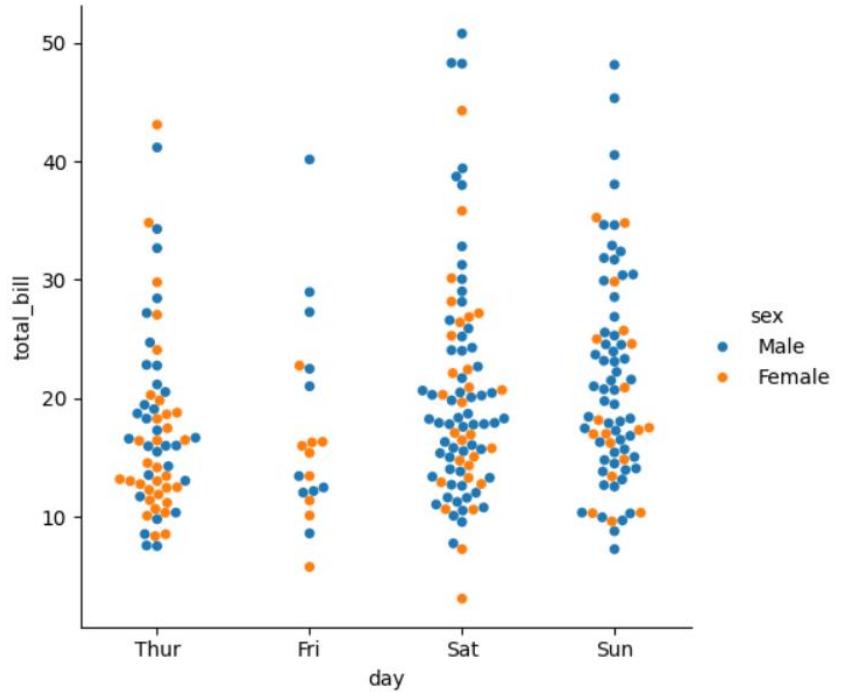
```
[21]: <seaborn.axisgrid.FacetGrid at 0x1ce51438d70>
```



- **Add the hue semantic:**

```
[7]: # add the hue semantic
sns.catplot(data=tips, x='day', y='total_bill', hue='sex', kind='swarm')
```

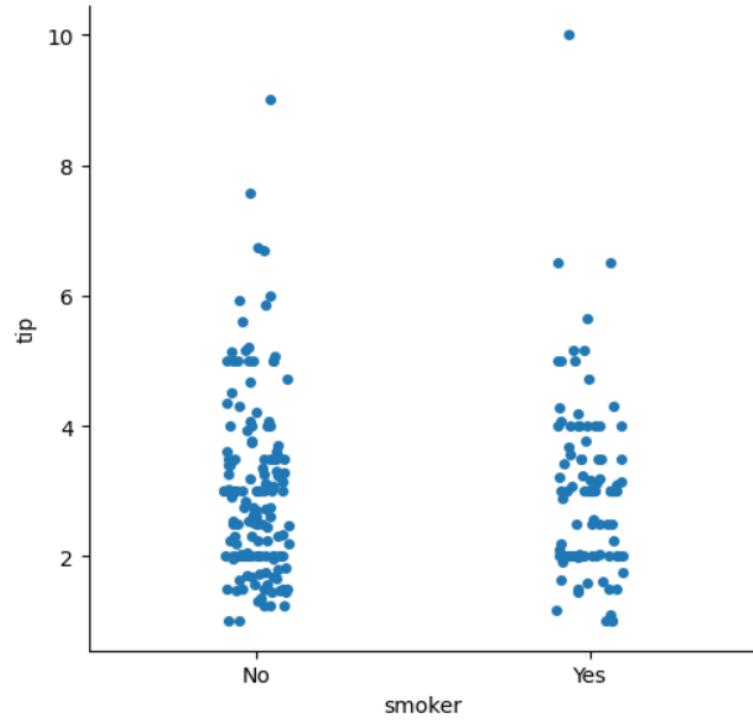
```
[7]: <seaborn.axisgrid.FacetGrid at 0x1ea88db3530>
```



- **Order parameter:** It is used to display multiple categorical plot in the same figure.

```
[8]: # order parameter - to display multiple categorical plot in the same figure
sns.catplot(data=tips, x='smoker', y='tip', order=['No', 'Yes'])
```

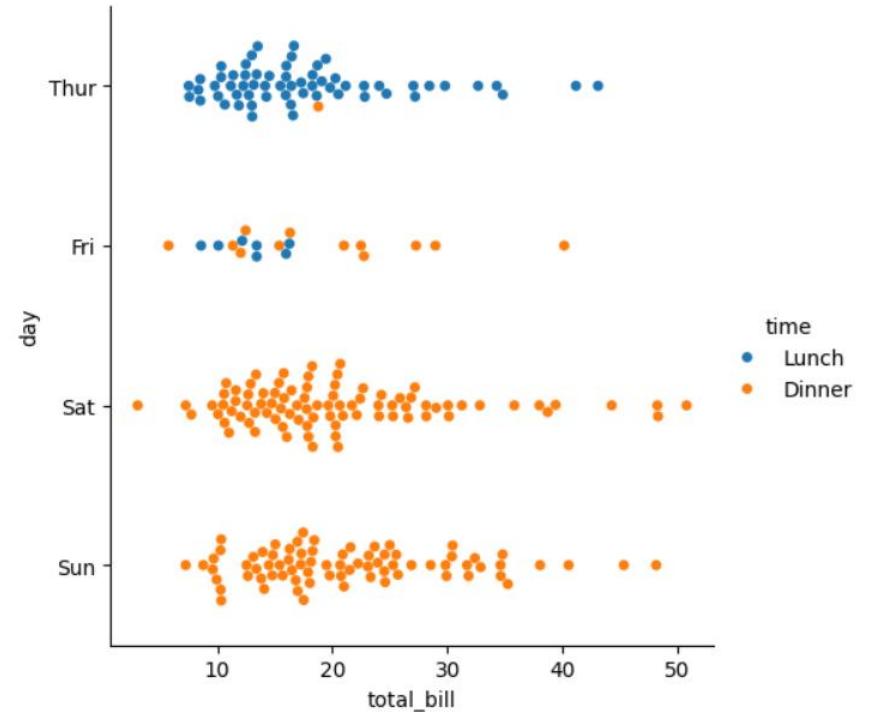
```
[8]: <seaborn.axisgrid.FacetGrid at 0x1ea89ee1cd0>
```



- **Categorical plot on vertical axis:**

```
[9]: # Categorical plot on vertical axis
sns.catplot(data=tips, x='total_bill', y='day', hue='time', kind='swarm')
```

```
[9]: <seaborn.axisgrid.FacetGrid at 0x1ea89ebb410>
```

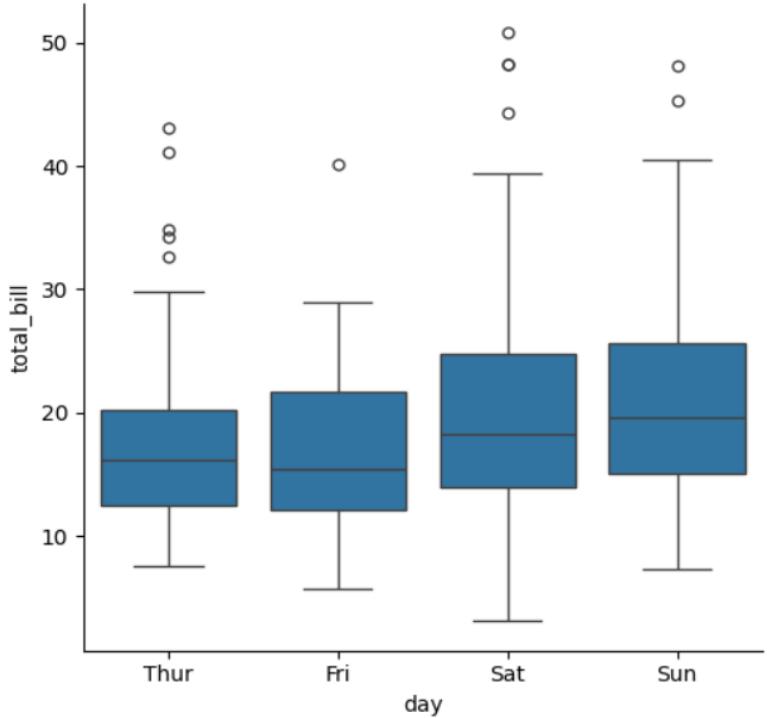


- Comparing Distribution:

- Box Plot:

```
[13]: # Box Plots  
sns.catplot(data=tips, x='day', y='total_bill', kind='box')
```

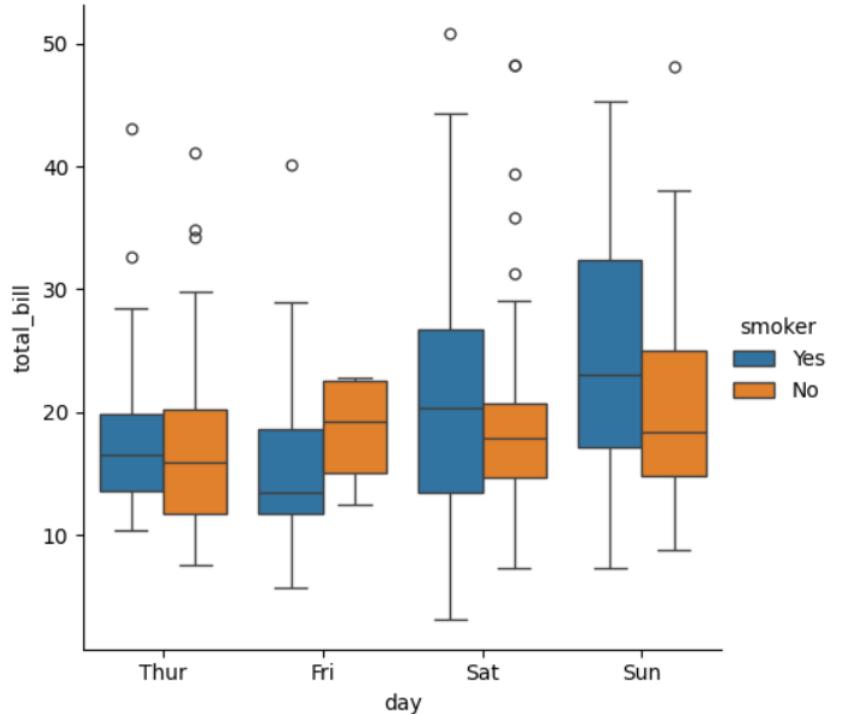
```
[13]: <seaborn.axisgrid.FacetGrid at 0x1ea89f790d0>
```



- Adding hue semantic:

```
[12]: # adding hue semantic  
sns.catplot(data=tips, x='day', y='total_bill', hue='smoker', kind='box')
```

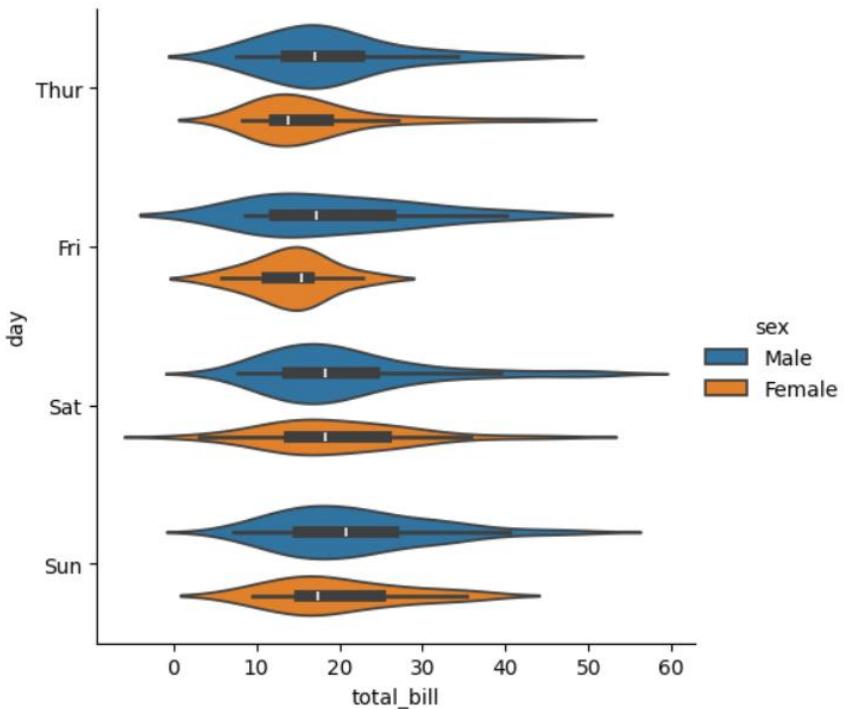
```
[12]: <seaborn.axisgrid.FacetGrid at 0x1ea8a056f90>
```



- **Violin Plot:**

```
[14]: # Violin Plot  
sns.catplot(data=tips, x='total_bill', y='day', hue='sex', kind='violin')
```

```
[14]: <seaborn.axisgrid.FacetGrid at 0x1ea89fe3020>
```



- **Split in the violin plot:**

```
[15]: # split in the violin plot  
sns.catplot(data=tips, x='day', y='total_bill', hue='sex', kind='violin', split=True)
```

```
[15]: <seaborn.axisgrid.FacetGrid at 0x1ea89f186b0>
```

