



Skill Match Resume Matcher and Skill Recommender

Milestone 1

**Submitted By
V.GNANENDHIRAN**

ARTIFICIAL INTELLIGENCE

Introduction:

- Jhon McCarthy first coined the term Artificial Intelligence in the year 1956.
- Artificial Intelligence (AI) as a field was first introduced in 1956 at the Dartmouth Conference.

What is Artificial Intelligence?

Artificial Intelligence (AI) means making computers or machines smart enough to do tasks that normally need human intelligence.

It helps machines to think, learn, and make decisions like humans.

Examples of AI:

- **Google Assistant / Siri / Alexa** → Answering questions, playing music.
- **Netflix / YouTube** → Suggesting movies and videos.
- **Self-driving Cars** → Cars that can drive on their own. Ex: Tesla
- **Healthcare** → AI predicting diseases.
- **Chatbots** → Talking to customers on websites.

Programming Languages Used in AI

1. **Python** – Most popular, easy to learn, used for machine learning and deep learning.
2. **R** – Used for data analysis and statistics.
3. **Java** – Used in big applications like chatbots and banking systems.
4. **C++** – Used in games and fast AI programs.

How AI is used in technology?

AI is used in many technologies around us to make systems smarter and more efficient. It helps in automation, decision-making, prediction, and personalization.

Examples: Smartphones, Social-Media, Healthcare, Transportation, Banking and Finance, E-commerce and entertainment.

MACHINE LEARNING

- Arthur Samuel first coined the term in Machine Learning in the year 1959.

What is Machine Learning?

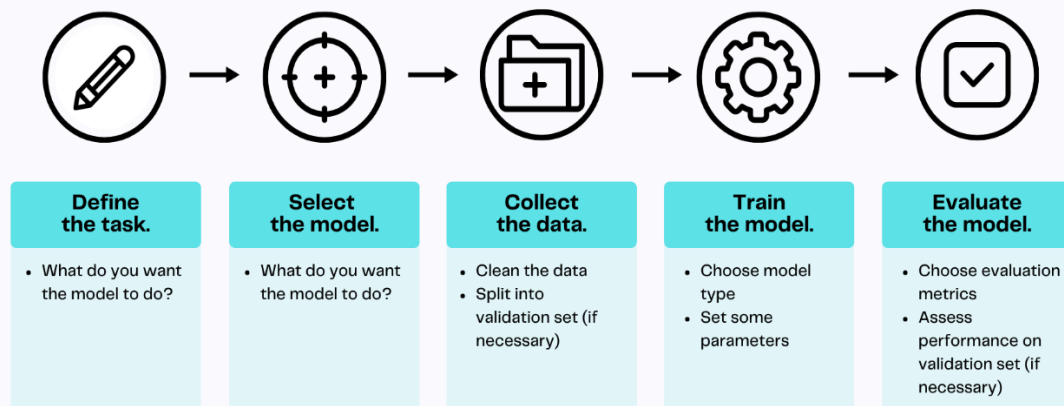
Machine Learning (ML) is a part of Artificial Intelligence (AI) that allows computers to learn from data and improve their performance without being directly programmed.

Example: Just like humans learn from experience, machines learn from data.

Machine Learning Process:

The ML process involves building a predictive model that can be used to find a solution for a problem statement.

Machine learning process



1. Define Objective

- Identify the problem you want to solve.
- Example: Predict student marks, forecast sales, detect spam, etc.

2. Data Gathering

- Collect relevant data from sources (databases, files, sensors, etc).

3. Preparing Data

- Clean the data (handle missing values, duplicates, errors).
- Transform into suitable format for analysis.

4. Data Exploration

- Analyse patterns, relationships, and insights in the data.
- Use statistical tools and visualization.

5. Building a Model

- Choose a machine learning algorithm (e.g., regression, decision tree, neural networks).
- Train the model on the prepared dataset.

6. Model Evaluation

- Test the model using metrics like accuracy, precision, recall, etc.
- Compare with baseline or other models.

7. Predictions

- Deploy the trained model to make predictions on new data.
- Continuously monitor and update the model for better performance.

MACHINE LEARNING DEFINITIONS

Algorithm: A set of rules and statistical techniques used to learn patterns from data

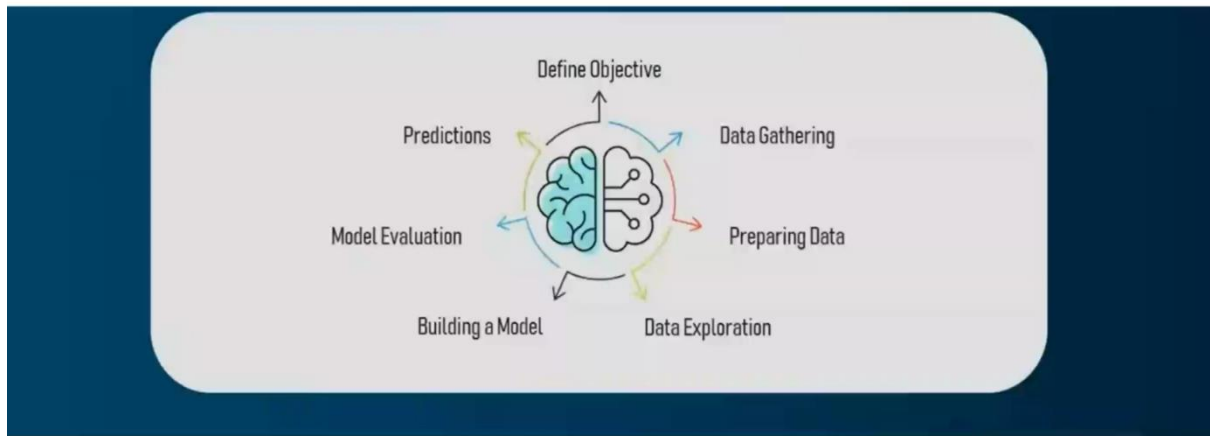
Model: A model is trained by using a Machine Learning Algorithm.

Predictor Variable: It is a feature(s) of the data that can be used to predict the output.

Response Variable: It is the feature or the output variable that needs to be predicted by using the predictor variable(s).

Training Data: The Machine Learning model is built using the training data.

Testing Data: The Machine Learning model is evaluated using the testing data.



MACHINE LEARNING PROCESS

*The Machine Learning process involves building a **Predictive model** that can be used to find a **solution** for a **Problem Statement**.*

NUMPY

Definition:

→ NumPy (Numerical Python) is a Python library used for working with numbers and arrays.

→ It helps in doing mathematical, statistical, and scientific calculations quickly and easily.

Practical tasks:

```
1 #Numpy arrays
2
3 #list
4 list_1 = [1,2,3,4,5]
5 print(list_1)
6 type(list_1)
```

[6]

Python

... [1, 2, 3, 4, 5]

... list

▷ ▾

```
1 np_array = np.array([1,2,3,4,5])
2 print(np_array)
3 type(np_array)
```

[7]

Python

... [1 2 3 4 5]

... numpy.ndarray

```
1 #creating a one dimensional array
2 a = np.array([1,2,3,4])
3 print(a)
```

[8]

Python

... [1 2 3 4]

▷ ▾

```
import numpy as np
n1=np.array([10,20,30,40,50,60])
n2=np.array([50,60,70,80,90])
print(np.intersect1d(n1,n2))
print(np.setdiff1d(n1,n2))
print(np.setdiff1d(n2,n1))
```

⌵ ▶ ▾ □ ... 🗑

[4]

✓ 0.0s

Python

... [50 60]

[10 20 30 40]

[70 80 90]

⌵ ◀ ▶ ⌵

```
1 a.shape #shape represents no of rows and columns in a particular array
```

[9] Python

... (4,)

```
1 b = np.array([(1,2,3,4),(5,6,7,8)])
2 print(b)
```

[10] Python

...
[[1 2 3 4]
 [5 6 7 8]]

▷ ▾

```
1 b.shape
```

[13] Python

... (2, 4)

🔗 Generate + Code + Markdown

```
1 c = np.array([(1,2,3,4),(5,6,7,8)],dtype = float)
2 print(c)
```

[15] Python

...
[[1. 2. 3. 4.]
 [5. 6. 7. 8.]]

- `np.vstack()` → stacks arrays vertically (row-wise)

```
| # Horizontal stack
print(np.hstack((n1, n2)))
```

[6] ✓ 0.0s Python

... [10 20 30 40 50 60]

```
print(np.column_stack((n1, n2))) # Column-wise stack
```

[7] ✓ 0.0s Python

...
[[10 40]
 [20 50]
 [30 60]]


```
import numpy as np

n1 = np.random.randint(1, 100, 5)
print(n1)
```

[3] ✓ 0.0s

Python

... [64 92 93 92 89]

```
import numpy as np

n1 = np.array([[1, 2, 3], [4, 5, 6]])
print(n1.shape)
```

[4] ✓ 0.0s

Python

... (2, 3)

```
import numpy as np

n1 = np.array([10, 20, 30])
n2 = np.array([40, 50, 60])
# Vertical stack
print(np.vstack((n1, n2)))
```

[5] ✓ 0.0s

Python

... [[10 20 30]
[40 50 60]]

- `np.hstack()` → stacks arrays horizontally (column-wise)
- `np.column_stack()` → stacks arrays as columns side by side

```

1 #reshaping a array
2 a = np.random.randint(0,10,(2,3))
3 print(a)
4 print(a.shape)

```

[35]

Python

```

... [[7 5 1]
      [9 6 6]]
      (2, 3)

```

```

1 b = a.reshape(3,2)
2 print(b)
3 print(b.shape)

```

[36]

Python

```

... [[7 5]
      [1 9]
      [6 6]]
      (3, 2)

```

Python IDE interface showing code execution:

```

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Array:\n", arr)
print("Dimensions (ndim):", arr.ndim)
print("Shape:", arr.shape)

```

[15] ✓ 0.0s Python

```

... Array:
      [[1 2 3]
       [4 5 6]]
      Dimensions (ndim): 2
      Shape: (2, 3)

```

- `n1`
→ A 3×3 2D array.
- `n1[0]`
→ First row (index 0) = [1 2 3].
- `n1[:,1]`
→ All rows (:) but only the 2nd column (1) → [2 5 8].
- `n1.transpose()`
→ Swaps rows ↔ columns.

Original rows → become columns

Original columns → become rows

```
import numpy as np

n1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(n1)
print(n1[0])
```

[8] ✓ 0.0s

Python

```
... [[1 2 3]
      [4 5 6]
      [7 8 9]]
      [1 2 3]
```

```
print(n1[:,1])
```

[10] ✓ 0.0s

Python

```
... [2 5 8]
```

```
print(n1.transpose())
```

[11] ✓ 0.0s

Python

```
... [[1 4 7]
      [2 5 8]
      [3 6 9]]
```

```
> ~
import numpy as np

# Create two arrays
a = np.array([10, 20, 30])
b = np.array([1, 2, 3])

print("Addition:", a + b)      # or np.add(a, b)
print("Subtraction:", a - b)  # or np.subtract(a, b)
print("Multiplication:", a * b) # or np.multiply(a, b)
print("Division:", a / b)     # or np.divide(a, b)
print("Power:", a ** b)       # or np.power(a, b)
print("Modulus:", a % b)      # or np.mod(a, b)

[14] ✓ 0.0s Python
```

```
... Addition: [11 22 33]
Subtraction: [ 9 18 27]
Multiplication: [10 40 90]
Division: [10. 10. 10.]
Power: [ 10 400 27000]
Modulus: [0 0 0]
```

- `n1.shape`
→ (2, 3) means 2 rows and 3 columns.
- `np.mean(n2)`
→ Average = $(10 + 20 + 30 + 60) / 4 = 30.0$
- `np.std(n2)` (Standard Deviation)
→ Measures how spread out the numbers are ≈ 18.71
- `np.median(n2)`
→ Middle value when sorted = [10, 20, 30, 60] → middle between 20 and 30 → 25.0

▷ ▾

⌵ ⏪ ⏩ 📄 ⋮ 🗑

```
import numpy as np

n1 = np.array([[1, 2, 3], [4, 5, 6]])
print(n1.shape)

n2 = np.array([10, 20, 30, 60])
print(np.mean(n2))
print(np.std(n2))
print(np.median(n2))
```

[12] ✓ 0.0s Python

... (2, 3)
30.0
18.708286933869708
25.0

> ▾

⌵ ⏪ ⏩ 📄 ⋮ 🗑

```
# Matrix Multiplication
n1 = np.array([[1, 2], [3, 4], [5, 6]]) # Shape: (3, 2)
n2 = np.array([[7, 8, 9], [10, 11, 12]]) # Shape: (2, 3)

print(n1.dot(n2))
```

[13] ✓ 0.0s Python

... [[27 30 33]
 [61 68 75]
 [95 106 117]]

PANDAS

→Pandas is a Python library used for data analysis and data manipulation.

Practical Tasks:

```
[16] ✓ 17.2s Python
import pandas as pd
s1 = pd.Series([1, 2, 4, 5])
print(s1)

.. 0    1
   1    2
   2    4
   3    5
dtype: int64
```

```
> ✓ 0.0s Python
s2 = pd.Series([6, 7, 9, 2], index=['a', 'b', 'c', 'd'])
print(s2)

.. a    6
   b    7
   c    9
   d    2
dtype: int64
```

```
> ✓ 0.0s Python
import pandas as pd

n1 = pd.Series({'a': 10, 'b': 20, 'c': 30})
print(n1)

... a    10
   b    20
   c    30
dtype: int64
```

- `pd.Series({key: value})` → creates a Series from a dictionary.
- The keys become the index labels (a, b, c)

- The values become the data (10, 20, 30).

```
import pandas as pd

n1 = pd.Series({'a': 10, 'b': 20, 'c': 30}, index=['d', 'c', 'a'])
print(n1)
```

[19] ✓ 0.0s Python

```
... d      NaN
   c    30.0
   a    10.0
   dtype: float64
```

- Pandas tries to match it with the dictionary keys:
 - 'c' → found → value 30
 - 'a' → found → value 10
 - 'd' → not found → NaN

->The output dtype becomes float64 because of the NaN.

```
print(s1.iloc[-1]) # Output: 5
print(s1[-1:])    # Output: Series containing last element
```

[25] ✓ 0.0s Python

```
... 5
   3  5
   dtype: int64
```

```
import pandas as pd

s1 = pd.Series([1, 2, 4, 5])
s2 = pd.Series([6, 7, 9, 2], index=['a', 'b', 'c', 'd'])

# First 3 elements
print(s1[:3])
```

3] ✓ 0.0s

Python

```
· 0    1
  1    2
  2    4
dtype: int64
```

```
# Last 2 elements
print(s1[-2:])

# Last element
print(s1.iloc[-1]) # prints single value
print(s1[-1:])     # prints as a Series
```

4] ✓ 0.0s

Python

```
· 2    4
  3    5
dtype: int64
5
3    5
dtype: int64
```

```
import pandas as pd

# Create a DataFrame
data = {'Name': ['Arun', 'Anbu', 'Bala'],
        'Age': [25, 30, 35],
        'City': ['Delhi', 'Mumbai', 'Bangalore']}

df = pd.DataFrame(data)

# Display DataFrame
print(df)

# Select 'Age' column
print(df['Age'])

# Compute average age
print(df['Age'].mean())
```

1] ✓ 0.0s

Python

```
   Name  Age   City
0  Arun   25  Delhi
1  Anbu   30  Mumbai
2  Bala   35  Bangalore
0     25
1     30
2     35
Name: Age, dtype: int64
30.0
```



```
import pandas as pd

ds = pd.read_csv('Iris.csv')
ds.head()
```

✓ 0.0s

Python

| | 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 |

```
ds.tail()
```

✓ 0.0s

Python

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|-----|-----|---------------|--------------|---------------|--------------|----------------|
| 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 |

```
ds.head(2)
```

[35] ✓ 0.0s

Python

```
...
```

| | 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 |

```
ds.tail(3)
```

[36] ✓ 0.0s

Python

```
...
```

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|-----|-----|---------------|--------------|---------------|--------------|----------------|
| 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 |

```
> ~
ds.describe()

[38] ✓ 0.0s Python

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

ds.shape

[37] ✓ 0.0s Python

.. (150, 6)
```

Summary Table

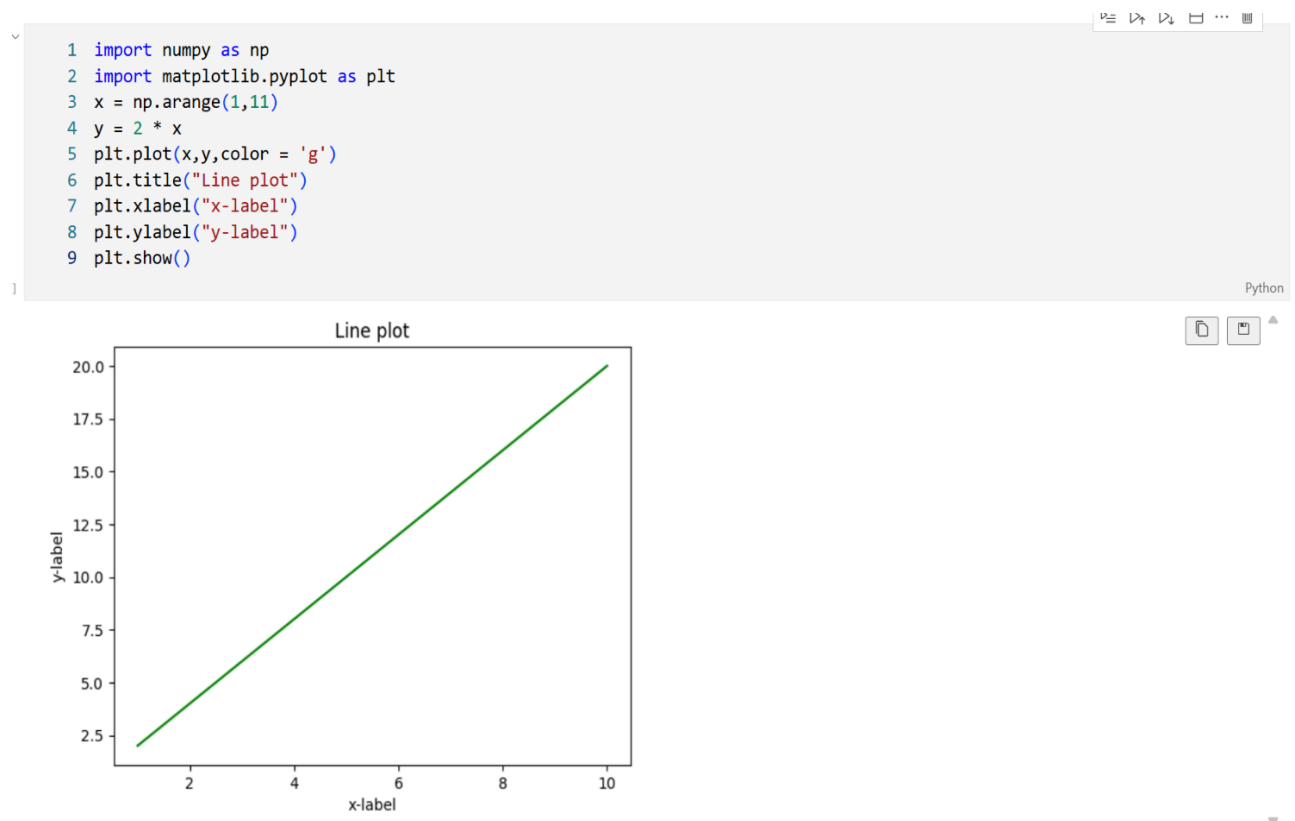
| <u>Function</u> | <u>Purpose</u> | |
|-------------------|--|--|
| <u>read_csv()</u> | <u>Load CSV file into a DataFrame</u> | |
| <u>head(n)</u> | <u>Show first <i>n</i> rows (default 5)</u> | |
| <u>tail(n)</u> | <u>Show last <i>n</i> rows (default 5)</u> | |
| <u>shape</u> | <u>Get number of rows and columns</u> | |
| <u>describe()</u> | <u>Get summary statistics of numeric columns</u> | |

MATPLOTLIB

→ Matplotlib is a Python library used for creating graphs, plots, and charts.

→ It helps to visualize data in different forms like line charts, bar graphs, pie charts, scatter plots, etc.

Practical Tasks:



```
1 import numpy as np
2 #Adding two lines in the same plot
3 x = np.arange(1,11)
4 y1= 2 * x
5 y2 = 3 * x
6 x
```

[21] ✓ 0.0s Python

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

```
1
2 #Adding the subplot
3 plt.subplot(1,2,1)
4 plt.plot(x,y1,color='violet',linestyle=':',linewidth = 2)
5 plt.title('Lineplot')
6 plt.xlabel('x-axis')
7 plt.ylabel('y-axis')
```

[22] ✓ 0.0s Python

... Text(0, 0.5, 'y-axis')



```

> ~
import numpy as np
from matplotlib import pyplot as plt
# Create an array from 1 to 10
x = np.arange(1, 11)
print(x)
y = 2 * x
print(y)
plt.plot(x, y)
plt.show()

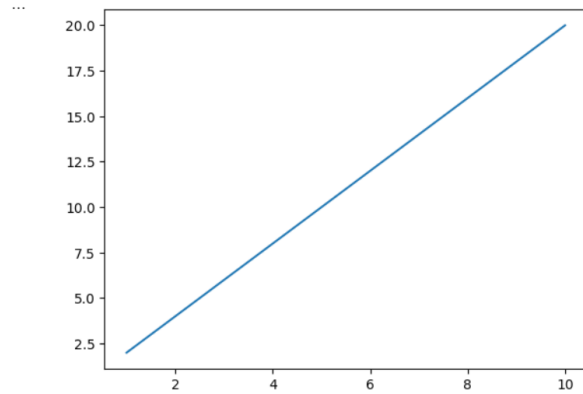
```

[2] ✓ 0.0s Python

```

... [ 1  2  3  4  5  6  7  8  9 10]
     [ 2  4  6  8 10 12 14 16 18 20]

```

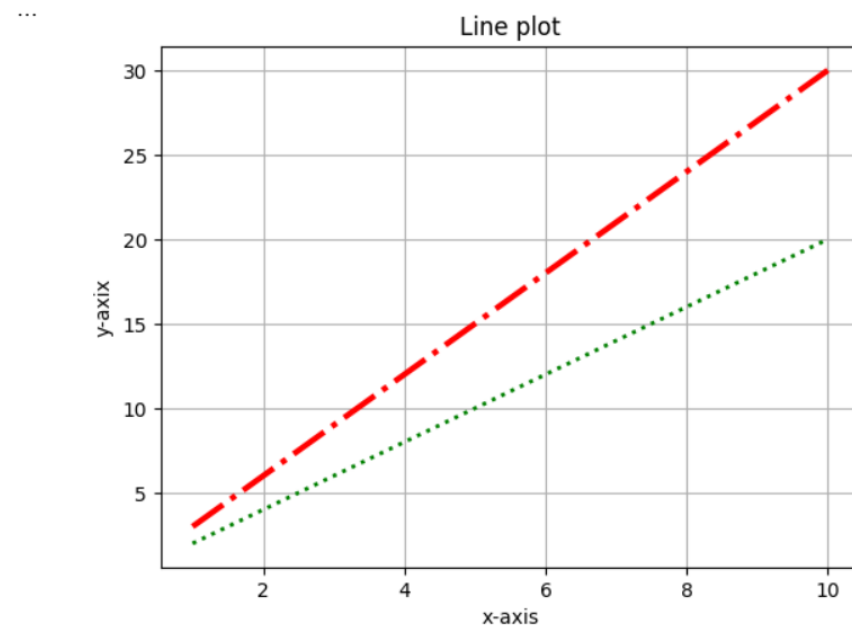


```

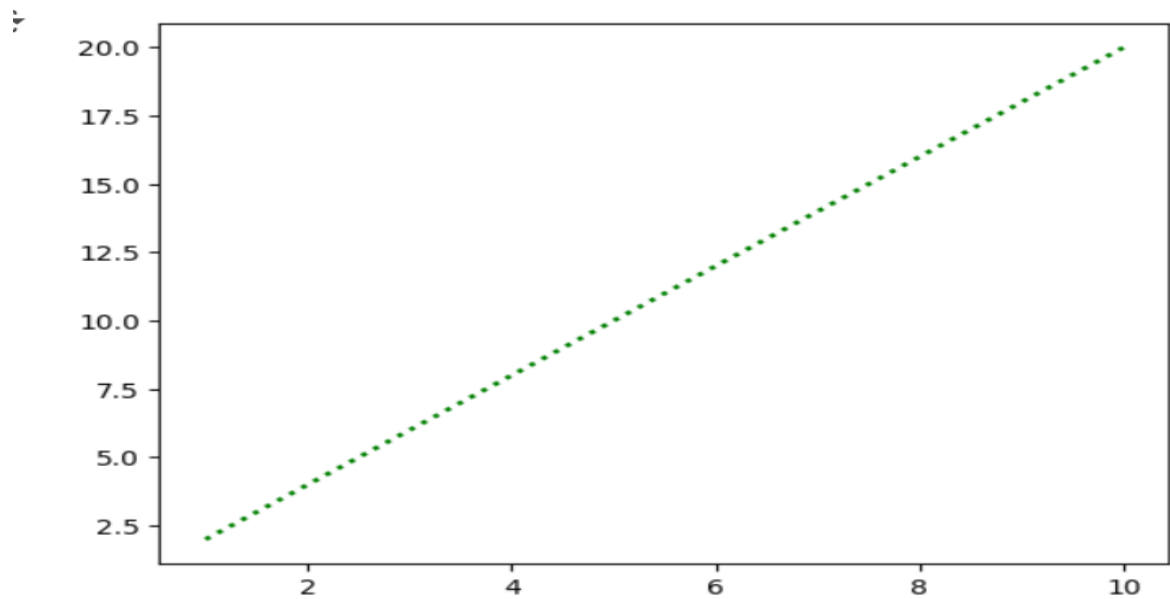
▷ ~
x=np.arange(1,11)
y1=2*x
y2=3*x
plt.plot(x,y1,color='g',linestyle=':',linewidth=2)
plt.plot(x,y2,color='r',linestyle='-.',linewidth=3)
• plt.title("Line plot")
  plt.xlabel("x-axis")
  plt.ylabel("y-axis")
  plt.grid(True)
  plt.show()

```

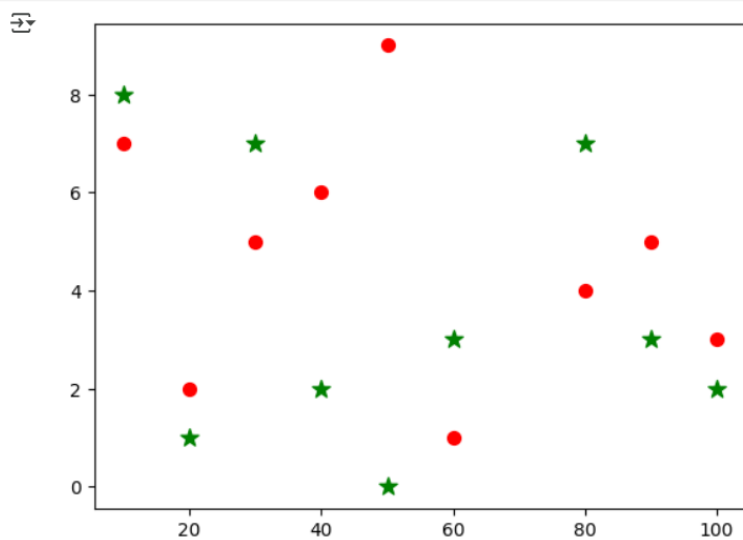
[5] ✓ 0.1s Python



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

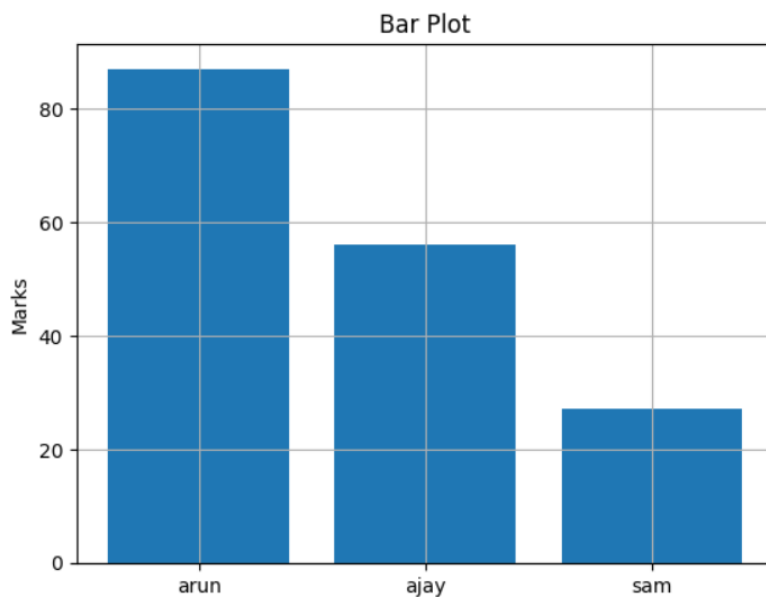


```
import matplotlib.pyplot as plt
a = [8, 1, 7, 2, 0, 3, 7, 3, 2]
x = [10, 20, 30, 40, 50, 60, 80, 90, 100]
b = [7, 2, 5, 6, 9, 1, 4, 5, 3]
plt.scatter(x, a, marker="*", c="g", s=100)
plt.scatter(x, b, marker=".", c="r", s=200)
plt.show()
```



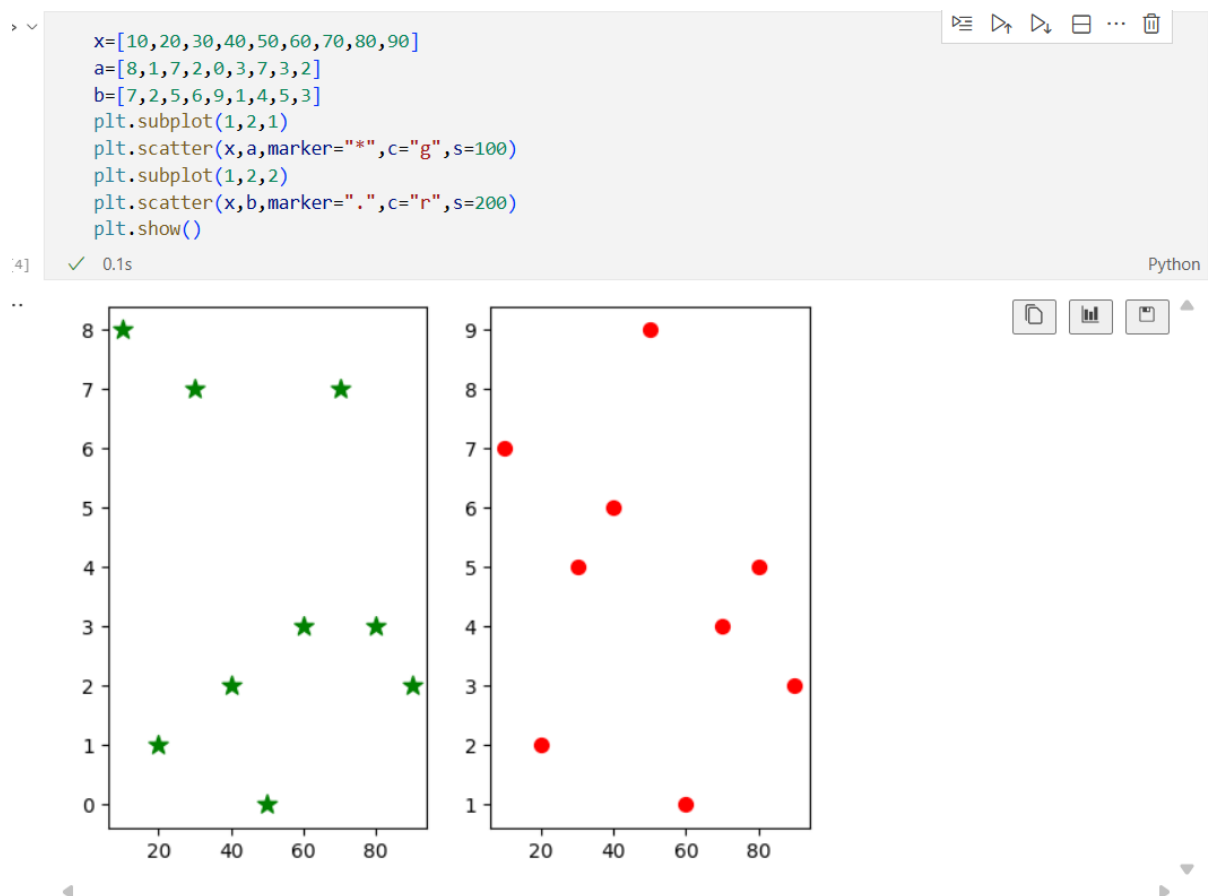
```
> from matplotlib import pyplot as plt
student = {"arun": 87, "ajay": 56, "sam": 27}
# Extract names and marks
names = list(student.keys())
values = list(student.values())
plt.bar(names, values)
plt.title("Bar Plot")
plt.xlabel("Names")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```

[3] ✓ 0.0s Python



- student → dictionary with names as keys and marks as values.
- list(student.keys()) → converts names to a list → ['arun', 'ajay', 'sam'].
- list(student.values()) → converts marks to a list → [87, 56, 27].
- plt.bar(names, values) → draws vertical bars for each student.
- plt.barh(names, values) → draws horizontal bars for students.
- plt.show() → displays the chart.
- plt.bar(names, values) → plots bars for each student.
- plt.title() → adds a heading on top.
- plt.xlabel() and plt.ylabel() → label the axes.

- `plt.grid(True)` → shows grid lines for better readability.

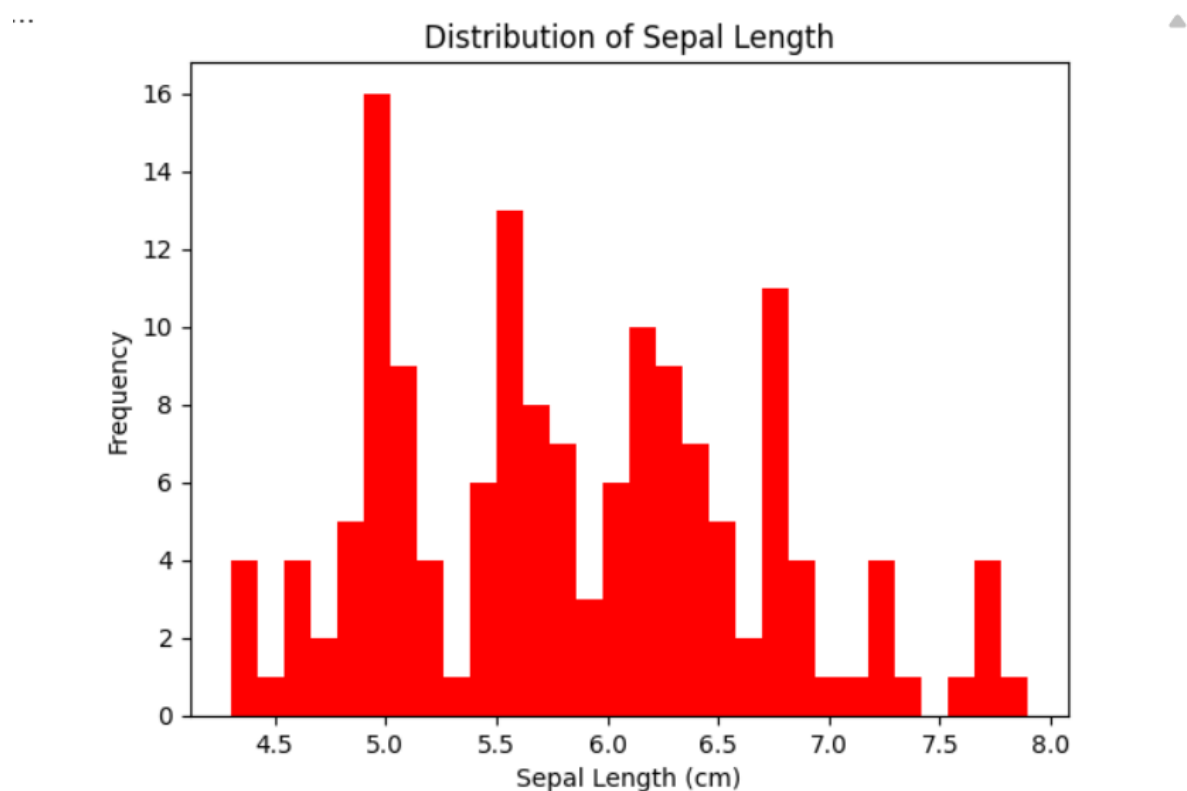


- `plt.subplot(1, 2, 1)` → means **1 row, 2 columns**, first plot on the **left**.
- `plt.subplot(1, 2, 2)` → second plot on the **right**.
- `plt.scatter()` → creates a scatter plot.
- `marker`, `c`, and `s` → control **shape**, **color**, and **size** of points.


```
> iris = pd.read_csv('iris.csv')
print(iris.head(2))
plt.hist(iris['SepalLengthCm'], bins=30, color="r")
plt.title("Distribution of Sepal Length")
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Frequency")
plt.show()
```

[16] ✓ 0.1s Python

```
...   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
```

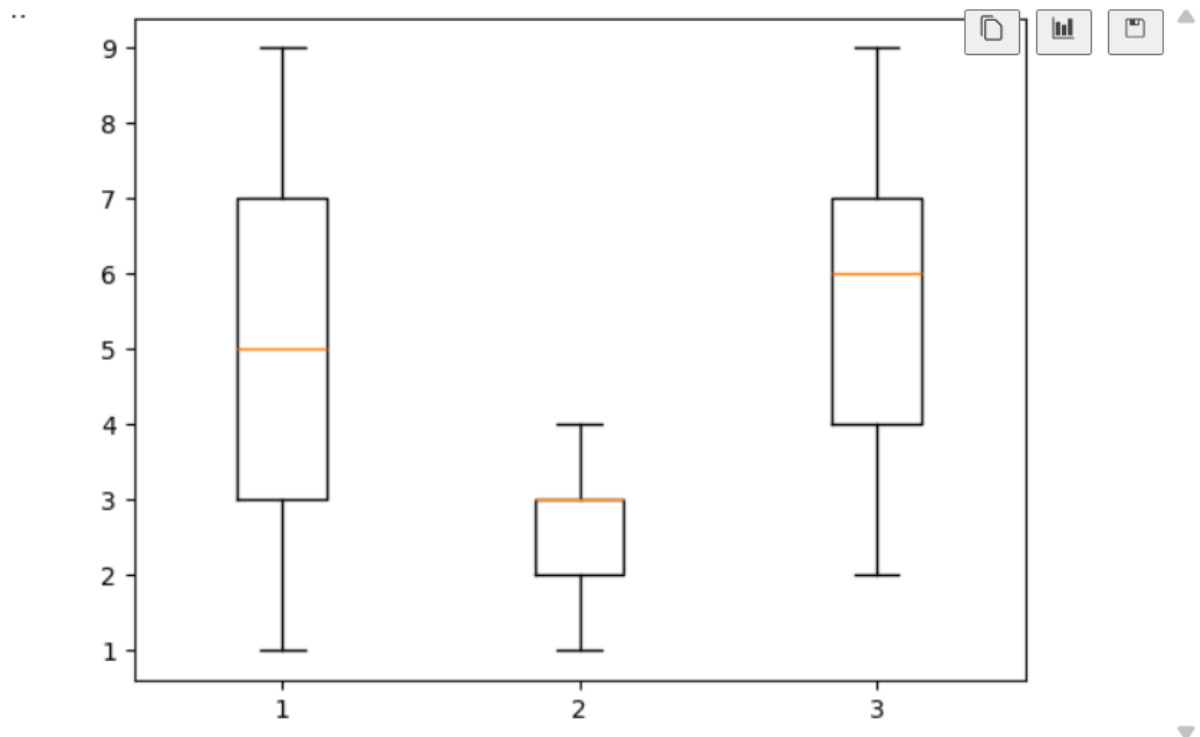


- `iris['Sepal.Length']` → selects the column for Sepal Length.
- `plt.hist(..., bins=30)` → divides data into 30 intervals to show the frequency distribution.

```

> ✓
a=[1,2,3,4,5,6,7,8,9]
b=[1,2,3,4,2,3,4,2,3]
c=[2,4,7,2,6,7,8,9,5]
data=list([a,b,c])
plt.boxplot(data)
plt.show()
[7] ✓ 0.0s Python

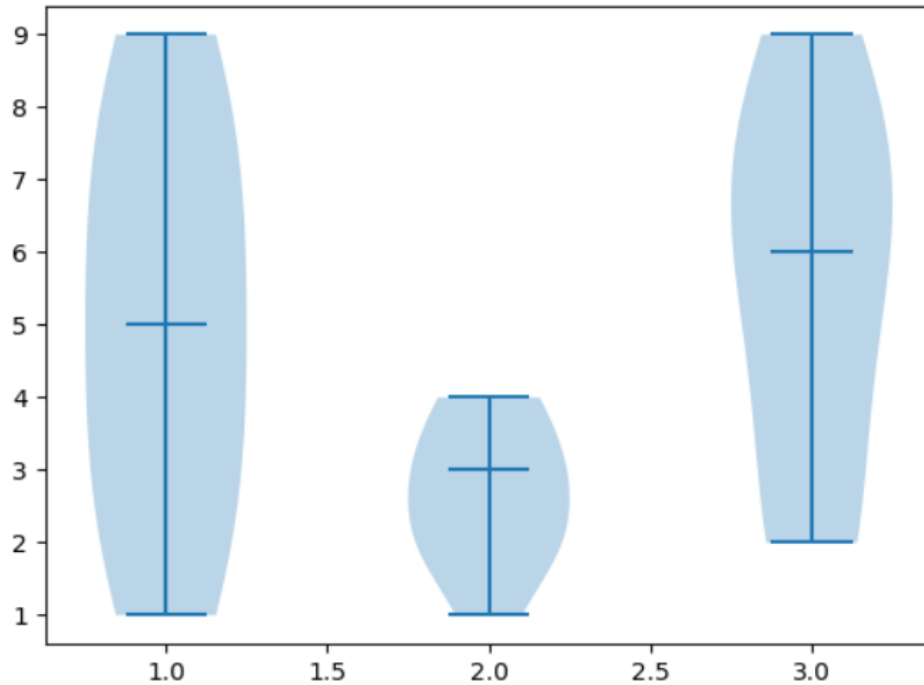
```



- `violin plot(data)` → draws a violin plot showing the data's distribution and density
- `data` → list of lists (each list is one dataset)
- `Show medians=True` → displays a horizontal line for the median inside each violin.
- The plot shows how spread out and concentrated each dataset's values are
- A boxplot (or box-and-whisker plot) shows how data is distributed.

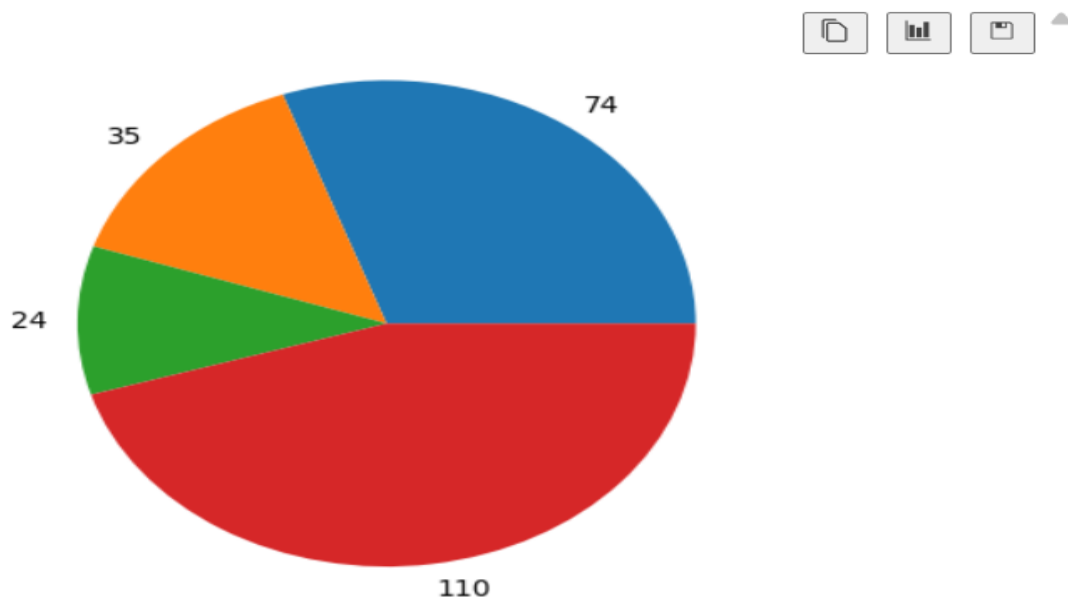
```
data=list([a,b,c])
plt.violinplot(data,showmedians=True)
plt.show()
```

18] ✓ 0.1s Python



```
f=['apple','orange','mango','guava']
quantity=[74,35,24,110]
plt.pie(quantity,labels=quantity)
plt.show()
```

9] ✓ 0.0s Python

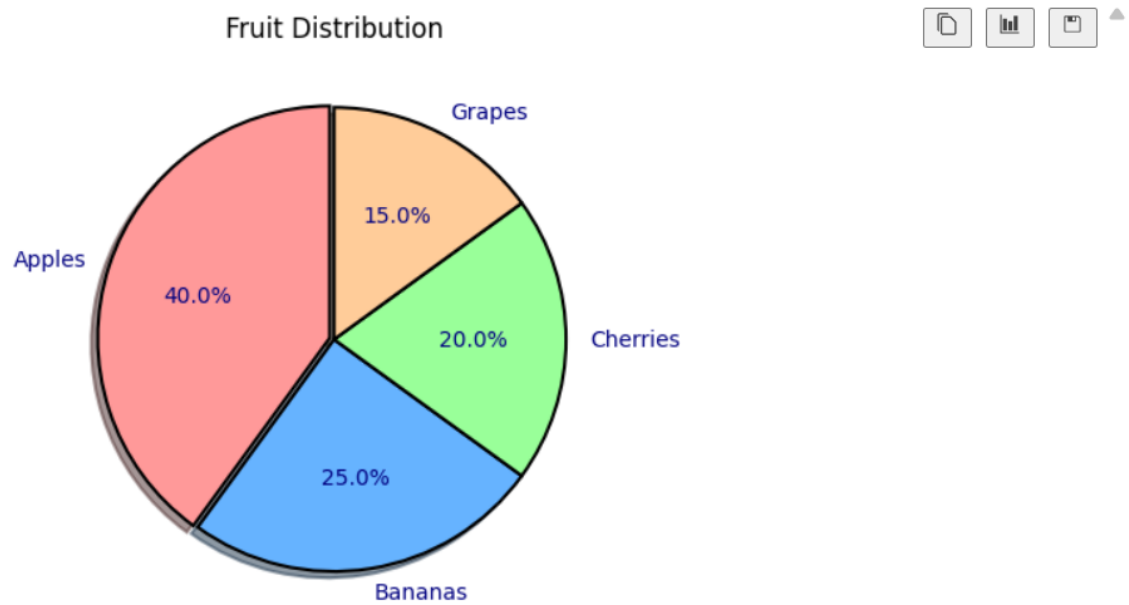


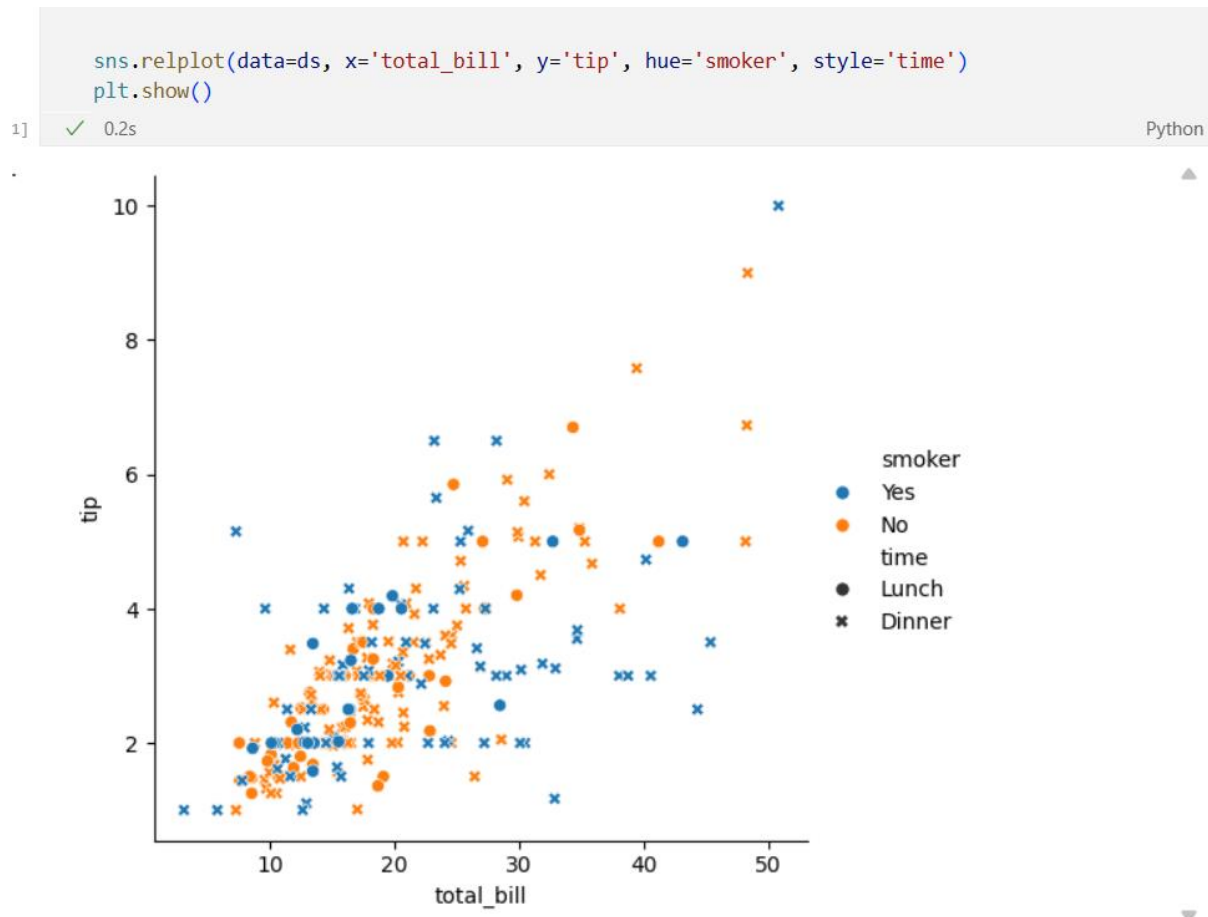
```

f = ['Apples', 'Bananas', 'Cherries', 'Grapes']
quantity = [40, 25, 20, 15]
colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99']
plt.pie(
    quantity, labels=f, colors=colors,
    autopct='%0.1f%%',
    explode=[0.02, 0, 0, 0], shadow=True,
    startangle=90,
    textprops={'fontsize':10, 'color':'navy'},
    wedgeprops={'edgecolor':'black', 'linewidth':1.5}
)
plt.title('Fruit Distribution')
plt.show()

```

2] ✓ 0.0s Python



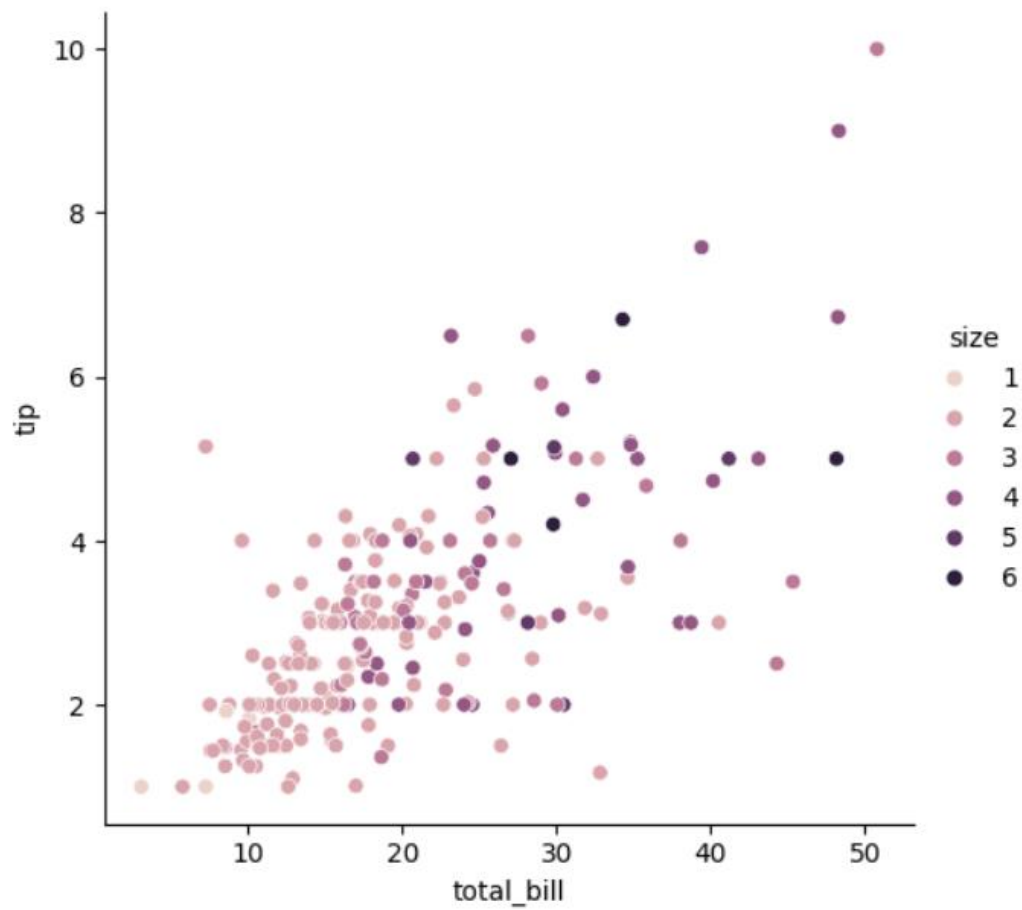


| Part | Meaning |
|-----------------------------|---|
| <code>sns.relplot()</code> | Creates a <i>relational plot</i> , showing the relationship between two numeric variables. |
| <code>data=ds</code> | The dataset (usually a DataFrame, like tips). |
| <code>x='total_bill'</code> | Variable for the X-axis. |
| <code>y='tip'</code> | Variable for the Y-axis. |
| <code>hue='smoker'</code> | Adds color based on whether the person is a smoker or not. Different colors represent different categories. |
| <code>style='time'</code> | Changes the marker shape based on another variable (like time, day, etc.). |

```
sns.relplot(data=ds, x="total_bill", y="tip", hue="size")  
plt.show()
```

✓ 0.1s

Python



Parameter Description

data=ds The dataset (e.g., Seaborn's built-in tips dataset).

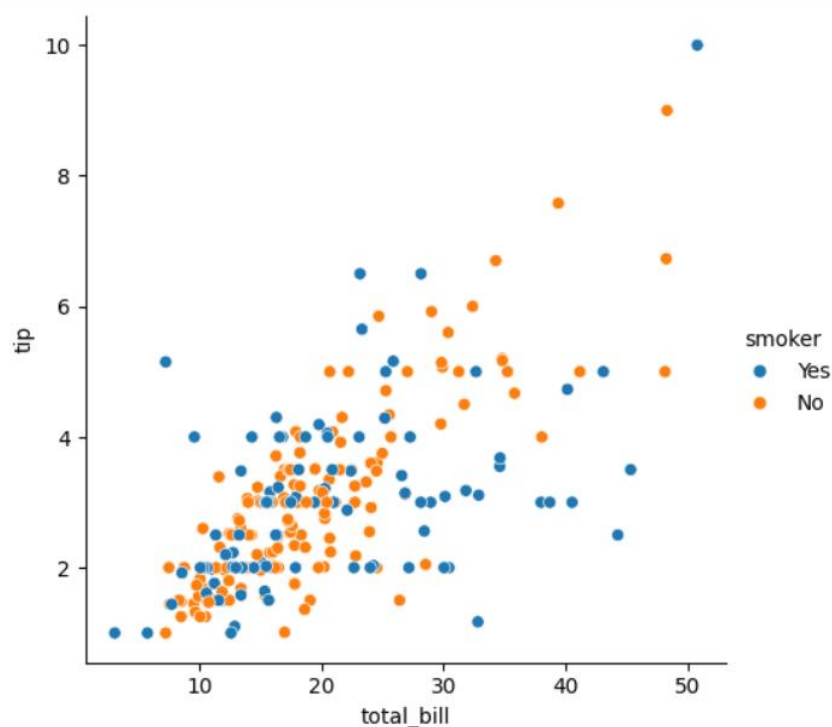
x="total_bill" X-axis → total amount of the bill.

y="tip" Y-axis → tip amount given.

hue="size" Color of each point represents the party size
(number of people). Different sizes → different
colors.

```
import matplotlib.pyplot as plt
sns.relplot(data=ds, x='total_bill', y='tip', hue='smoker')
plt.show()
```

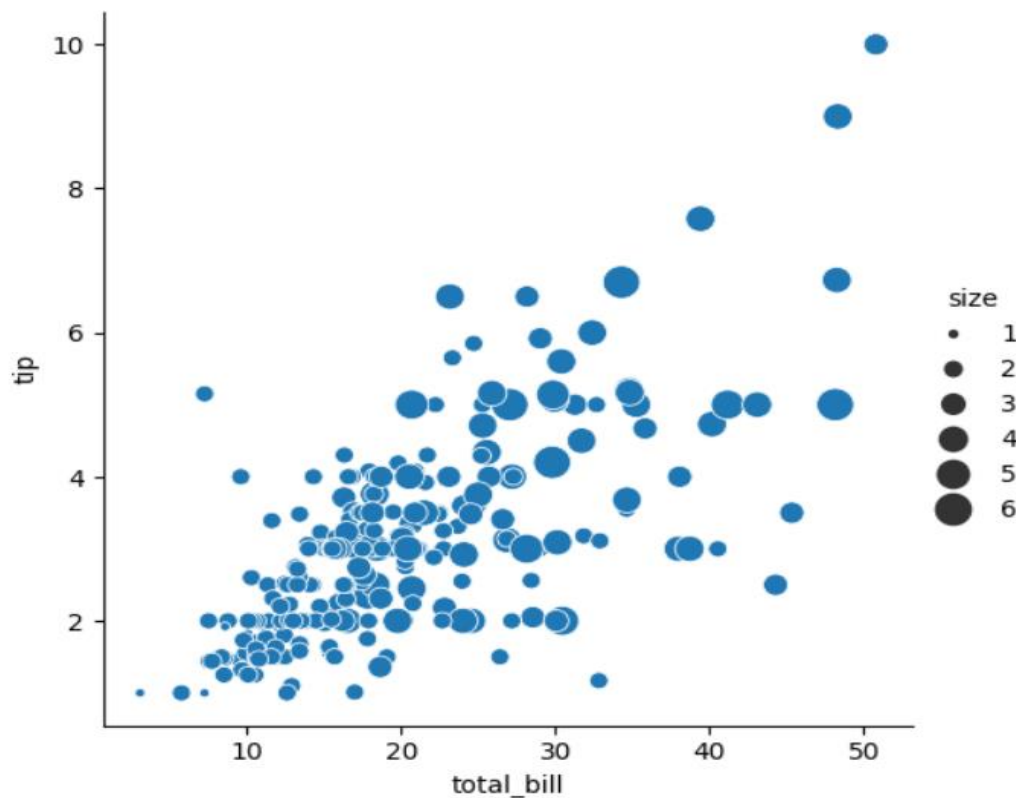
[12] ✓ 0.1s Python



```
sns.relplot(data=ds, x="total_bill", y="tip", size="size", sizes=(15,200))  
plt.show()
```

✓ 0.2s

Python

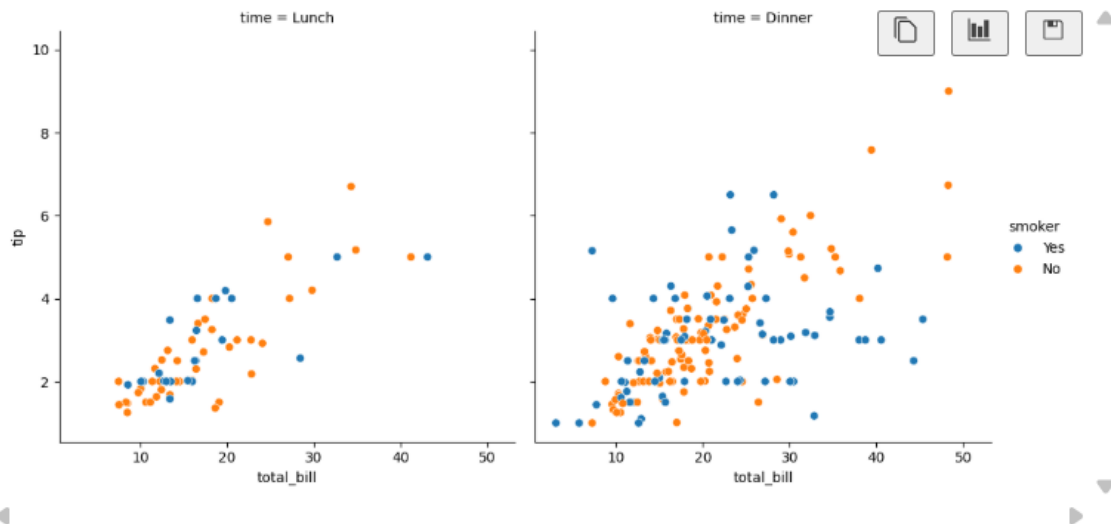


| Parameter | Description |
|------------------------------|--|
| <code>data=ds</code> | The dataset (for example, Seaborn's built-in tips dataset). |
| <code>x="total_bill"</code> | Values for the x-axis — total bill amount. |
| <code>y="tip"</code> | Values for the y-axis — tip amount. |
| <code>size="size"</code> | Uses the size column to vary the point size — larger dots mean larger party sizes. |
| <code>sizes=(15, 200)</code> | Sets the range of dot sizes (min 15px, max 200px). |


```
sns.relplot(data=ds, x="total_bill", y="tip", hue="smoker", col="time")  
plt.show()
```

[5] ✓ 0.4s

Python



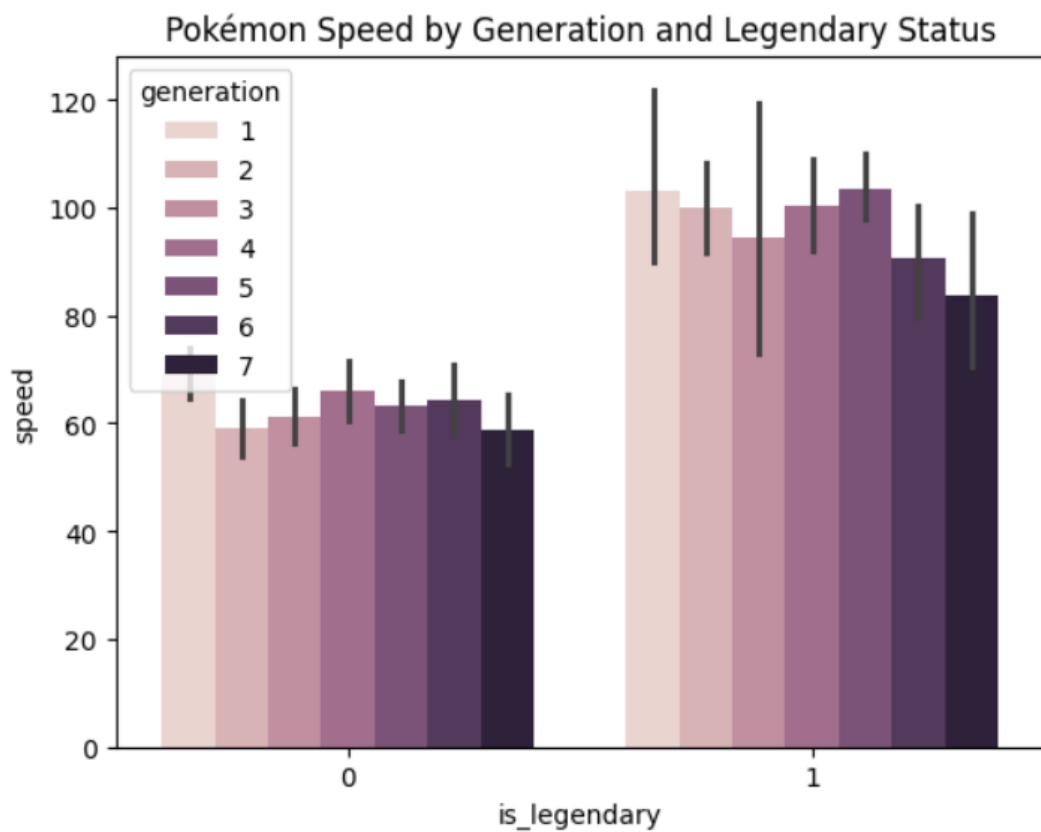
col="time"

Creates separate plots (columns) for each unique value in the time column (Lunch and Dinner)

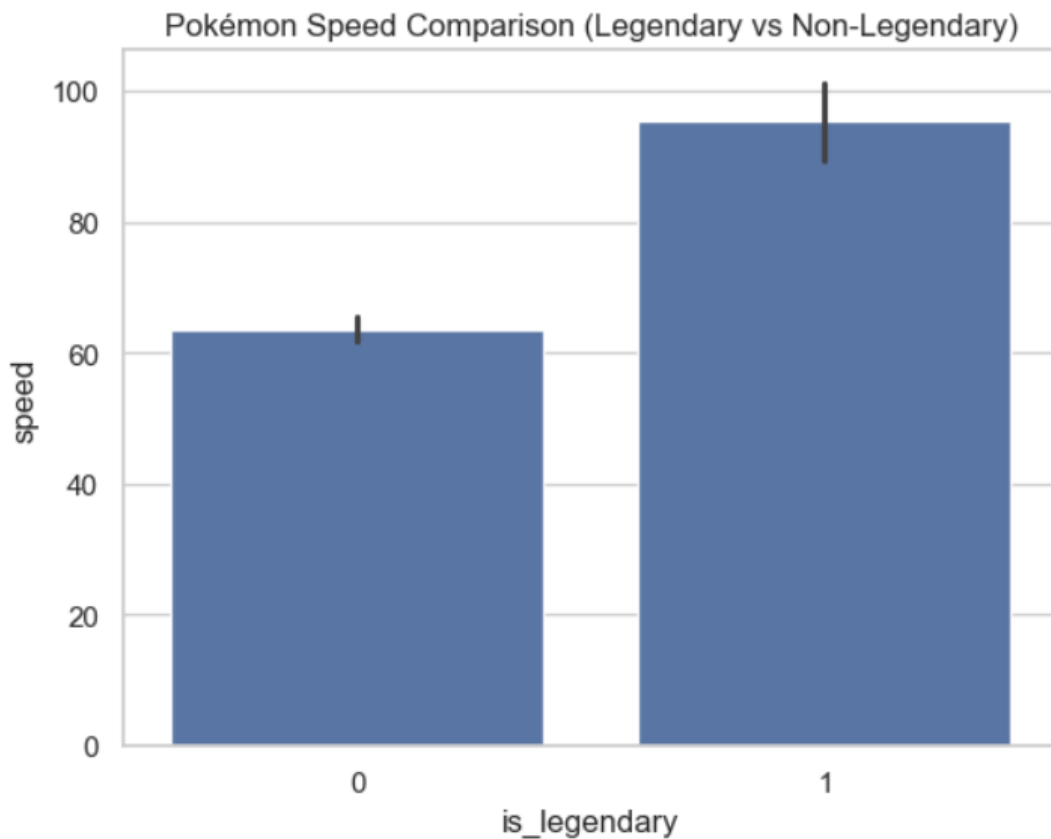
```

> ~
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
pokemon = pd.read_csv("Pokemon.csv")
sns.barplot(x="is_legendary", y="speed", hue="generation", data=pokemon)
plt.title("Pokémon Speed by Generation and Legendary Status")
plt.show()
21] ✓ 0.4s Pythor

```

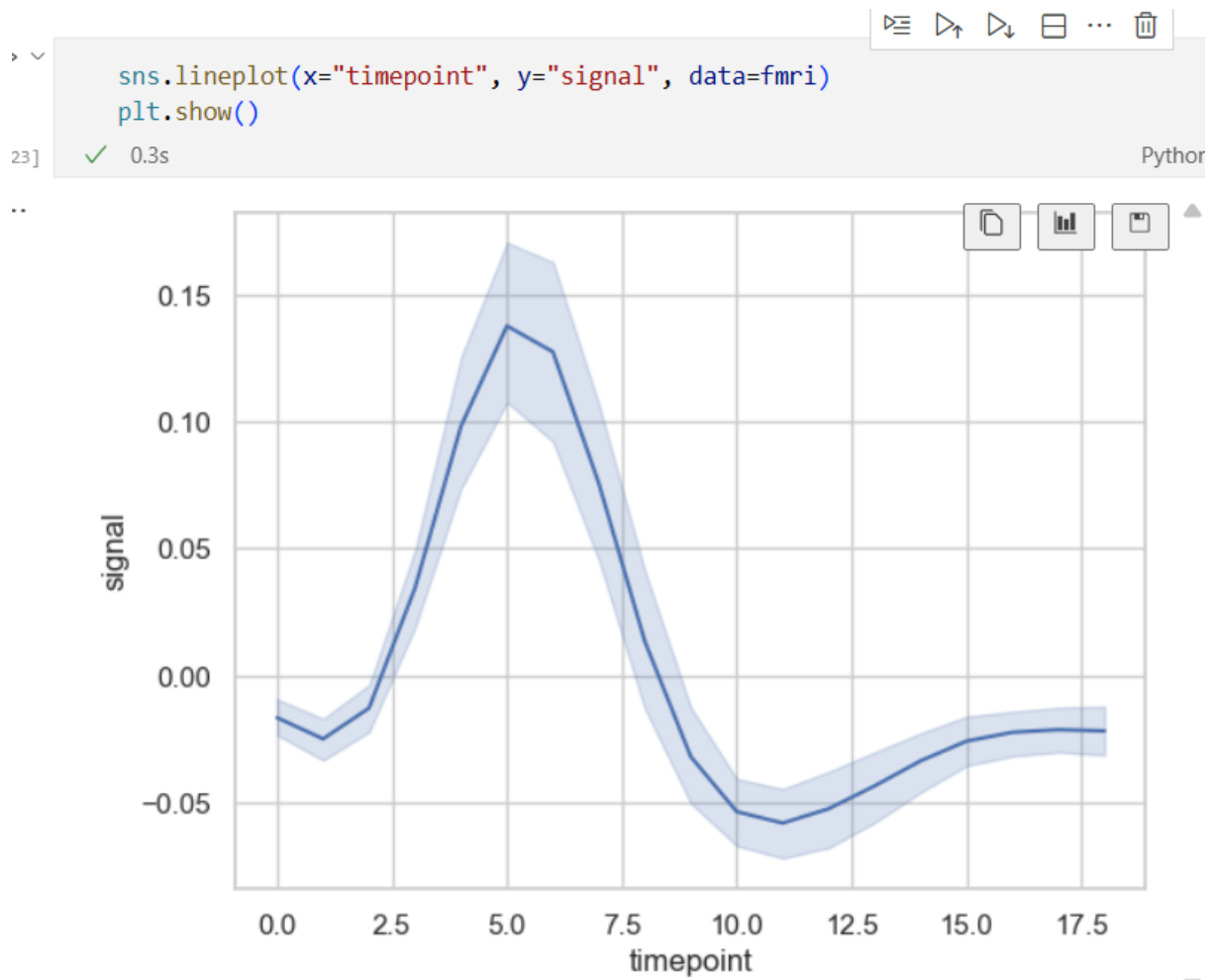


```
> \n\nsns.set(style="whitegrid")\n# Create bar plot\nsns.barplot(x="is_legendary", y="speed", data=pokemon)\nplt.title("Pokémon Speed Comparison (Legendary vs Non-Legendary)")\nplt.show()\n22] ✓ 0.1s Python
```



`sns.set(style="whitegrid")`

Sets a clean background with white grid lines for better readability



| Parameter | Description |
|-----------|-------------|
|-----------|-------------|

| | |
|---------------|---|
| x="timepoint" | X-axis → time points during the experiment. |
|---------------|---|

| | |
|------------|------------------------------------|
| y="signal" | Y-axis → brain signal measurement. |
|------------|------------------------------------|

| | |
|-----------|--------------------------------------|
| data=fmri | Uses Seaborn's built-in fMRI dataset |
|-----------|--------------------------------------|

```
fmri = sns.load_dataset('fmri')
fmri.head()
fmri.shape
sns.relplot(data=fmri, x='timepoint', y='signal')
plt.show()
```

16]

✓ 1.5s

Python

