📘 NumPy, Matplotlib, Pandas — Detailed Colab Material

🔷 NumPy

NumPy (**Numerical Python**) is the core library for scientific computing in Python.
It provides support for:

- Large **multidimensional arrays**
- Mathematical functions (linear algebra, statistics, etc.)
- Random number generation

🧾 NumPy Array Creation

- `np.array()` → Create array from list or tuple
- `np.arange(start, stop, step)` → Create range of numbers
- `np.linspace(start, stop, num)` → Create evenly spaced numbers

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)

print("arange:", np.arange(0, 10, 2))
print("linspace:", np.linspace(0, 1, 5))
```

```
Array: [1 2 3 4 5]
arange: [0 2 4 6 8]
linspace: [0.   0.25 0.5  0.75 1.  ]
```

🔄 Reshape Arrays

- `reshape(rows, cols)` → Change array shape without changing data

```
matrix = np.arange(1, 10).reshape(3, 3)
print("3x3 Matrix:\n", matrix)
```

```
3x3 Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

⭐ Special Arrays

- `np.zeros((m, n))` → Matrix of zeros
- `np.ones((m, n))` → Matrix of ones
- `np.eye(n)` → Identity matrix

```
print("Zeros:\n", np.zeros((2, 3)))
print("Ones:\n", np.ones((2, 3)))
print("Identity:\n", np.eye(3))
```

```
Zeros:
 [[0. 0. 0.]
 [0. 0. 0.]]
Ones:
 [[1. 1. 1.]
 [1. 1. 1.]]
Identity:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

🎲 Random Numbers

- `np.random.rand(m, n)` → Random numbers (0–1, uniform)
- `np.random.randn(m, n)` → Random numbers (normal distribution)

- `np.random.randint(low, high, size)` → Random integers

```
print("rand:\n", np.random.rand(2, 2))
print("randn:\n", np.random.randn(2, 2))
print("randint:\n", np.random.randint(1, 10, size=(2, 3)))
```

```
rand:
 [[0.73415628 0.62039303]
 [0.66954772 0.21417082]]
randn:
 [[-0.16787452 -1.94803001]
 [ 0.93631903  2.48482118]]
randint:
 [[6 5 7]
 [6 1 4]]
```

➕ Math Operations

- `np.mean()` → Mean
- `np.std()` → Standard deviation
- `np.dot(A, B)` → Matrix multiplication
- `np.linalg.inv(A)` → Inverse of matrix

```
matrix = np.array([[1, 2], [3, 4]])
print("Mean:", np.mean(matrix))
print("Std Dev:", np.std(matrix))
print("Dot Product:\n", np.dot(matrix, matrix))
print("Inverse:\n", np.linalg.inv(matrix))
```

```
Mean: 2.5
Std Dev: 1.118033988749895
Dot Product:
 [[ 7 10]
 [15 22]]
Inverse:
 [[-2.   1. ]
 [ 1.5 -0.5]]
```

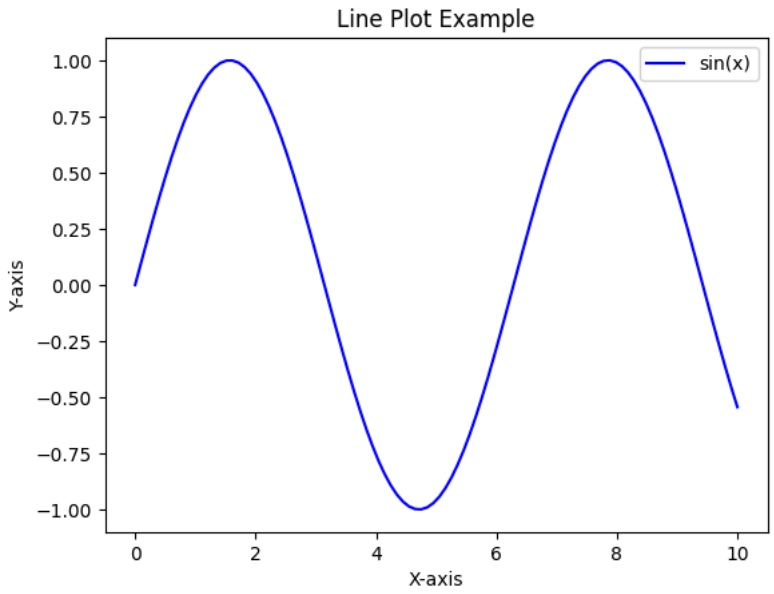
🔷 Matplotlib

Matplotlib is a **visualization library** for plots and charts.
We use it mostly via `pyplot`.

📈 Line Plot

- `plt.plot(x, y)` → Draw line graph
- `plt.xlabel(), plt.ylabel()` → Add labels
- `plt.title()` → Add title
- `plt.legend()` → Show legend

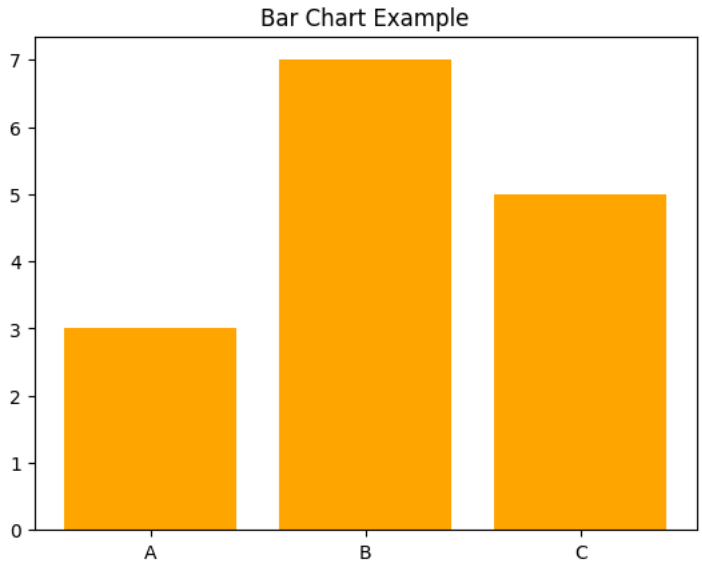
```
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y, label="sin(x)", color="blue")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Line Plot Example")
plt.legend()
plt.show()
```



📊 Bar Chart

- `plt.bar(categories, values)` → Vertical bar chart
- `plt.barh(categories, values)` → Horizontal bar chart

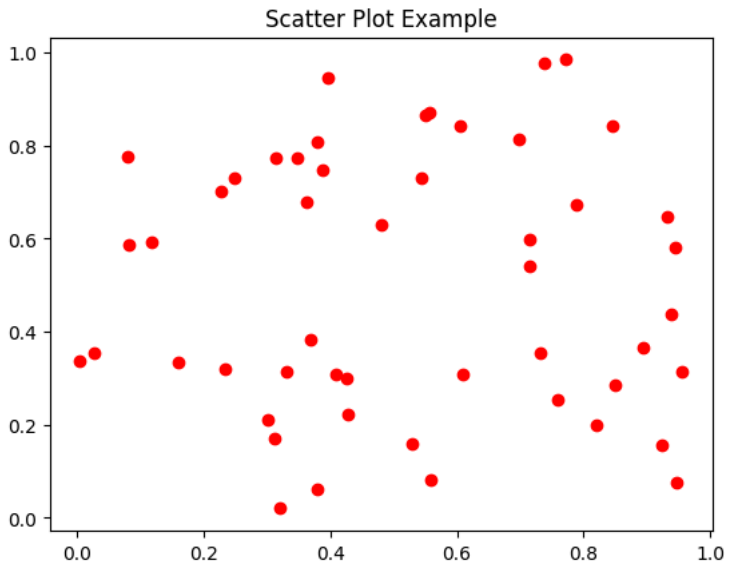
```
categories = ["A", "B", "C"]
values = [3, 7, 5]

plt.bar(categories, values, color="orange")
plt.title("Bar Chart Example")
plt.show()
```



🔵 Scatter Plot

- `plt.scatter(x, y)` → Scatter plot
  Useful for comparing two variables.

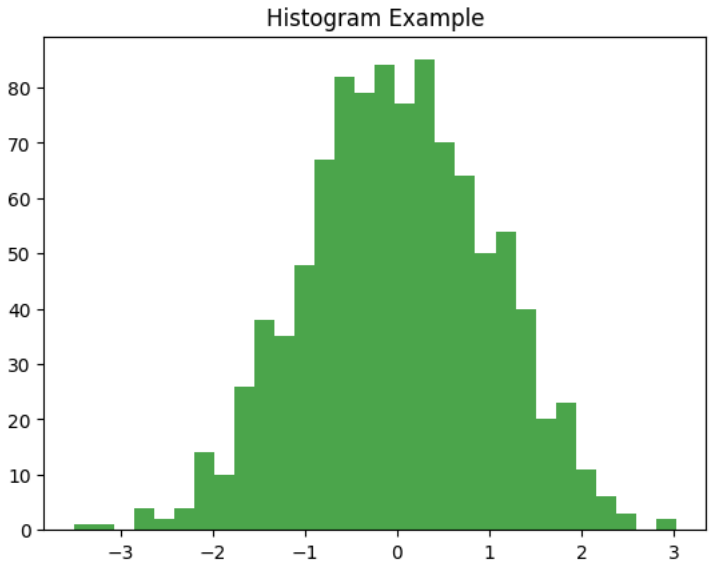
```
x = np.random.rand(50)
y = np.random.rand(50)

plt.scatter(x, y, color="red")
plt.title("Scatter Plot Example")
plt.show()
```



📦 Histogram

- `plt.hist(data, bins)` → Histogram of values
  Shows frequency distribution.

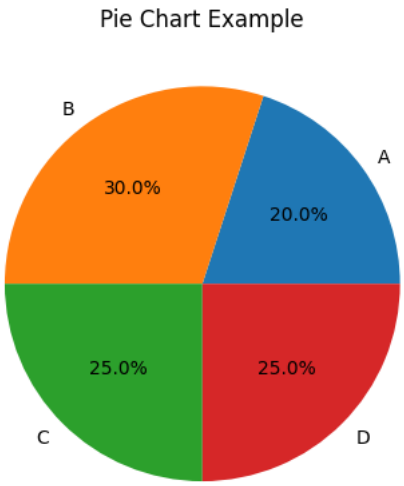
```
data = np.random.randn(1000)
plt.hist(data, bins=30, color="green", alpha=0.7)
plt.title("Histogram Example")
plt.show()
```



🥧 Pie Chart

- `plt.pie(values, labels, autopct)` → Pie chart
- `autopct='%1.1f%%'` → Show percentage

```
sizes = [20, 30, 25, 25]
labels = ["A", "B", "C", "D"]

plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title("Pie Chart Example")
plt.show()
```



🔷 Pandas

Pandas is a **data analysis library** with:

- `Series` → 1D data
- `DataFrame` → 2D tabular data

🧾 Create DataFrame

- `pd.DataFrame(dict)` → Create dataframe from dictionary
- `pd.Series()` → Create 1D labeled data

```
import pandas as pd

data = {
    "Name": ["Alice", "Bob", "Charlie", "David"],
    "Age": [24, 27, 22, 32],
    "Score": [85, 90, 88, 76]
}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age  Score
0    Alice   24     85
1      Bob   27     90
2  Charlie   22     88
3    David   32     76
```

👀 Viewing Data

- `df.head(n)` → First n rows
- `df.tail(n)` → Last n rows
- `df.info()` → Info about data
- `df.describe()` → Statistics

```
print(df.head(2))
print(df.tail(2))
print(df.info())
print(df.describe())
```

```
    Name  Age  Score
0  Alice   24     85
1    Bob   27     90
      Name  Age  Score
2  Charlie   22     88
3    David   32     76
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    4 non-null      object
```

```
 1   Age     4 non-null      int64
 2   Score   4 non-null      int64
dtypes: int64(2), object(1)
memory usage: 228.0+ bytes
None
             Age      Score
count   4.000000   4.000000
mean   26.250000  84.750000
std     4.349329   6.184658
min    22.000000  76.000000
25%    23.500000  82.750000
50%    25.500000  86.500000
75%    28.250000  88.500000
max    32.000000  90.000000
```

🎯 Selecting Data

- `df['col']` → Select column
- `df.loc[row, col]` → Select by label
- `df.iloc[row, col]` → Select by index

```
print(df["Name"])
print(df.loc[1, "Age"])
print(df.iloc[2])
```

```
0      Alice
1        Bob
2    Charlie
3      David
Name: Name, dtype: object
27
Name     Charlie
Age           22
Score         88
Name: 2, dtype: object
```

⚡ Operations

- `df.mean()` → Mean
- `df.groupby()` → Group data
- `df.sort_values()` → Sort data

```
print("Mean Age:", df["Age"].mean())
print("Sorted by Age:\n", df.sort_values("Age"))
```

```
Mean Age: 26.25
Sorted by Age:
       Name  Age  Score
2  Charlie   22     88
0    Alice   24     85
1      Bob   27     90
3    David   32     76
```

🚑 Handling Missing Data

- `df.fillna(value)` → Fill missing values
- `df.dropna()` → Remove missing values

```
df2 = df.copy()
df2.loc[2, "Score"] = None
print("With NaN:\n", df2)

print("After fillna:\n", df2.fillna(df2["Score"].mean()))
```

```
With NaN:
       Name  Age  Score
0    Alice   24   85.0
1      Bob   27   90.0
2  Charlie   22    NaN
3    David   32   76.0
After fillna:
       Name  Age      Score
```