Experiment No. 1 – Setting up the Python Environment and Libraries

Date: 16-07-2025

Aim

To set up the Python environment using Jupyter Notebook, create and execute Python code cells and Markdown cells, and demonstrate the use of Jupyter Widgets and Jupyter AI for interactive programming.

Steps

1. Created a new Jupyter Notebook.
2. Added and ran Python code cells.
3. Created Markdown cells for documentation.
4. Imported and used libraries like NumPy and Matplotlib.
5. Used Jupyter Widgets for interactivity.
6. Demonstrated Jupyter AI for AI-assisted queries.

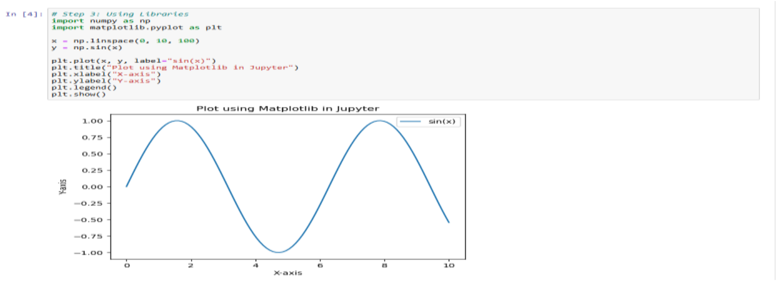
Code Sample

python

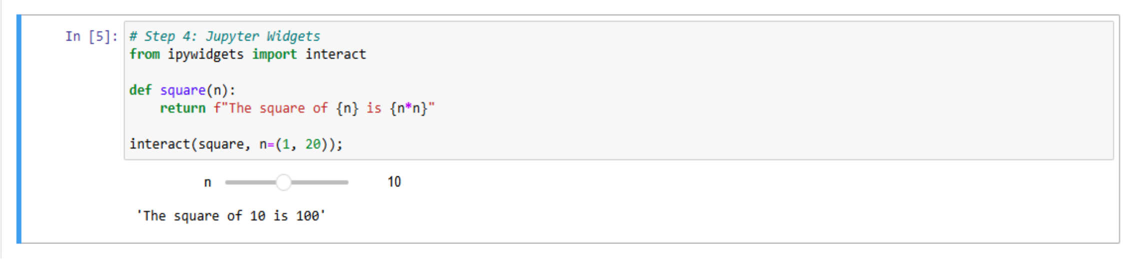
```
# Basic Python Execution
print("Hello, Jupyter Notebook!")

# Using Libraries (NumPy and Matplotlib)
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y, label="sin(x)")
plt.title("Plot using Matplotlib in Jupyter")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

```
# Using Jupyter Widgets
from ipywidgets import interact
def square(n):
    return f"The square of {n} is {n*n}"
interact(square, n=(1, 20));
```

OUTPUT:



```
In [4]: # Step 3: Using Libraries
        import numpy as np
        import matplotlib.pyplot as plt

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

        plt.plot(x, y, label="sin(x)")
        plt.title("Plot using Matplotlib in Jupyter")
        plt.xlabel("X-axis")
        plt.ylabel("Y-axis")
        plt.legend()
        plt.show()
```



```
In [5]: # Step 4: Jupyter Widgets
        from ipywidgets import interact

        def square(n):
            return f"The square of {n} is {n*n}"

        interact(square, n=(1, 20));
```

```
'The square of 10 is 100'
```

Result

- Successfully created and executed Python and Markdown cells.
- Plotted a sine wave using NumPy and Matplotlib.
- Created an interactive slider using Jupyter Widgets.
- Demonstrated AI-assisted query generation with Jupyter AI.

Experiment No. 2 – EDA: Data Import and Export

Date: 23-07-2025

Aim

To import data from various sources (CSV, Excel, SQL, web), handle different formats, and export a DataFrame to Excel.

Steps

1. Imported libraries (pandas, sqlite3, BeautifulSoup, requests).
2. Imported data from CSV and Excel files.
3. Imported data from an in-memory SQL database.
4. Web scraped data from Wikipedia tables.
5. Exported DataFrame to an Excel file.

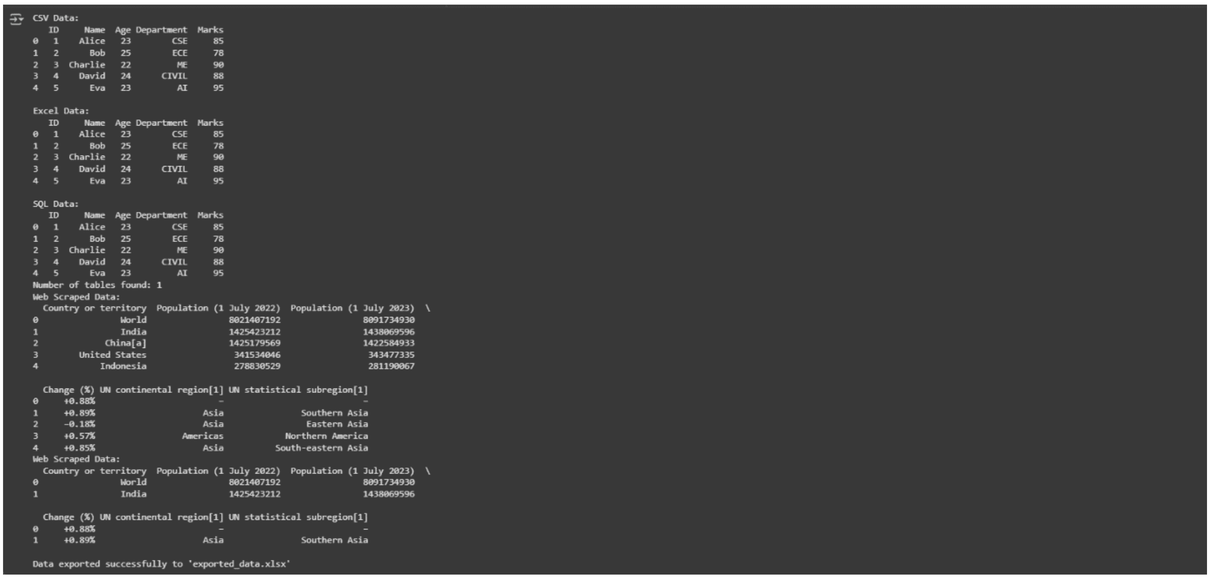
Code Sample

python

```
import pandas as pd
import sqlite3
from bs4 import BeautifulSoup
import requests
from io import StringIO

# Import CSV
csv_data = pd.read_csv("sample.csv")
print("CSV Data:")
print(csv_data.head())

# Import Excel
excel_data = pd.read_excel("sample.xlsx")
print("\nExcel Data:")
print(excel_data.head())
```

```
# Import from SQL Database
conn = sqlite3.connect(":memory:")
csv_data.to_sql("students", conn, index=False, if_exists="replace")
sql_data = pd.read_sql("SELECT * FROM students", conn)
print("\nSQL Data:")
print(sql_data.head())

# Web Scraping Wikipedia
url = "https://en.wikipedia.org/wiki/List_of_countries_by_population_(United_Nations)"
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, "html.parser")
tables_html = soup.find_all("table", {"class": "wikitable"})

if tables_html:
  tables = pd.read_html(StringIO(str(tables_html[0])))
  web_data = tables[0]
  print("Web Scraped Data:")
  print(web_data.head())
else:
  print("No tables found on the page.")

# Export to Excel
csv_data.to_excel("exported_data.xlsx", index=False)
print("\nData exported successfully to 'exported_data.xlsx'")
```

OUTPUT:



```
CSV Data:
   ID     Name  Age Department  Marks
0   1    Alice   23        CSE     85
1   2      Bob   25        ECE     78
2   3  Charlie   22         ME     90
3   4    David   24      CIVIL     88
4   5      Eva   23         AI     95

Excel Data:
   ID     Name  Age Department  Marks
0   1    Alice   23        CSE     85
1   2      Bob   25        ECE     78
2   3  Charlie   22         ME     90
3   4    David   24      CIVIL     88
4   5      Eva   23         AI     95

SQL Data:
   ID     Name  Age Department  Marks
0   1    Alice   23        CSE     85
1   2      Bob   25        ECE     78
2   3  Charlie   22         ME     90
3   4    David   24      CIVIL     88
4   5      Eva   23         AI     95
Number of tables found: 1
Web Scraped Data:
  Country or territory  Population (1 July 2022)  Population (1 July 2023)  \
0                World                8021407192                8091734930
1                India                1425423212                1438069596
2             China[a]                1425179569                1422584933
3        United States                 341534046                 343477335
4            Indonesia                 278830529                 281190067

  Change (%) UN continental region[1] UN statistical subregion[1]
0     +0.88%                        -                           -
1     +0.89%                     Asia               Southern Asia
2     -0.18%                     Asia                Eastern Asia
3     +0.57%                 Americas            Northern America
4     +0.85%                     Asia          South-eastern Asia
Web Scraped Data:
  Country or territory  Population (1 July 2022)  Population (1 July 2023)  \
0                World                8021407192                8091734930
1                India                1425423212                1438069596

  Change (%) UN continental region[1] UN statistical subregion[1]
0     +0.88%                        -                           -
1     +0.89%                     Asia               Southern Asia

Data exported successfully to 'exported_data.xlsx'
```

Result

- Successfully imported data from CSV, Excel, SQL, and web sources.
- Handled multiple data formats efficiently.
- Exported data to Excel file format.

Experiment No. 3 – EDA: Data Cleaning

Date: 30-07-2025

Aim

To perform data cleaning by handling missing values, removing duplicates, converting data types, and normalizing data.

Steps

1. Created a sample dataset with missing values and duplicates.
2. Detected and handled missing values by filling them with mean, mode, or placeholder.
3. Removed duplicate rows.
4. Converted data types as needed.
5. Applied normalization using MinMaxScaler and StandardScaler.

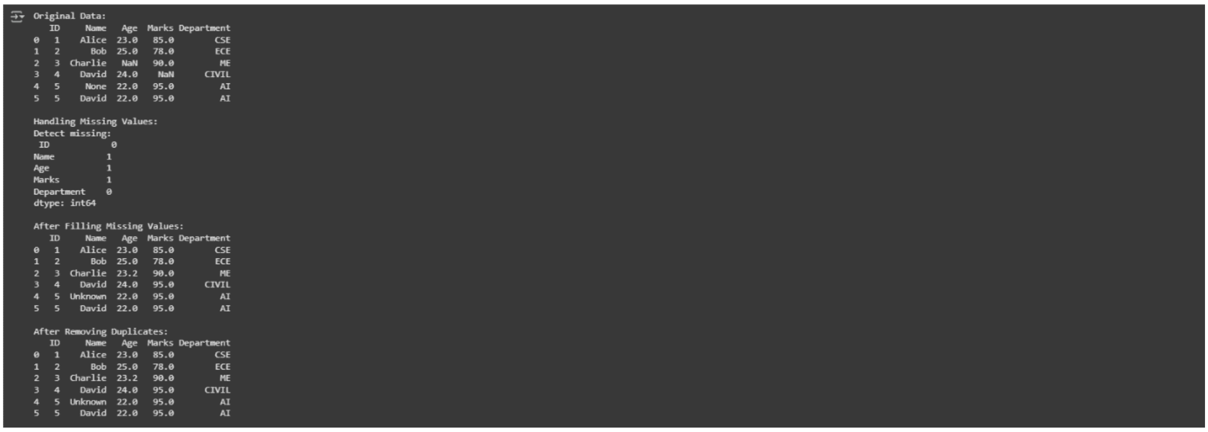
Code Sample

python

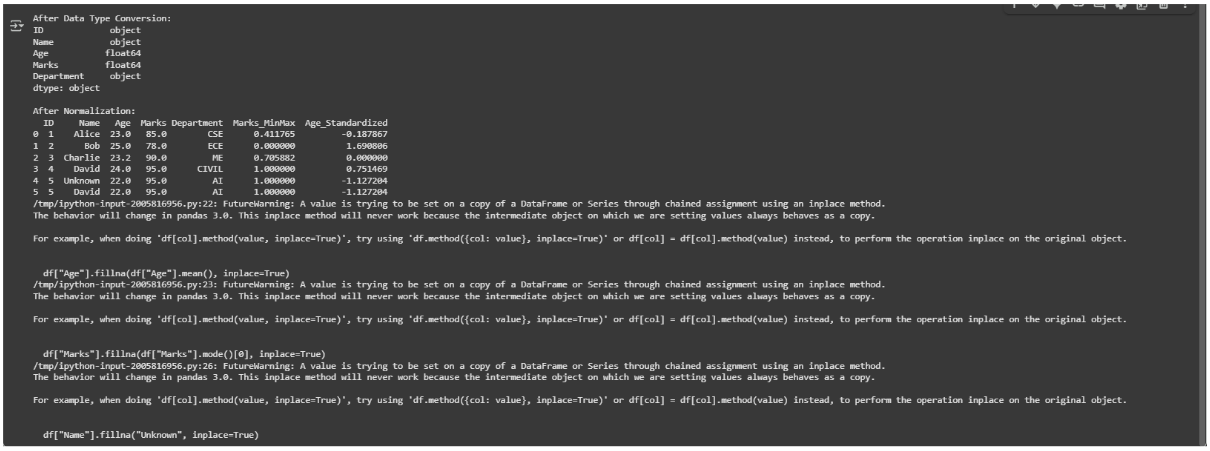
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Sample dataset
data = {
  "ID": [1, 2, 3, 4, 5, 5],
  "Name": ["Alice", "Bob", "Charlie", "David", None, "David"],
  "Age": [23, 25, np.nan, 24, 22, 22],
  "Marks": [85, 78, 90, np.nan, 95, 95],
  "Department": ["CSE", "ECE", "ME", "CIVIL", "AI", "AI"]
}
df = pd.DataFrame(data)
print("Original Data:")
print(df)
```

```
# Handle missing values
df["Age"].fillna(df["Age"].mean(), inplace=True)
df["Marks"].fillna(df["Marks"].mode()[0], inplace=True)
df["Name"].fillna("Unknown", inplace=True)


# Remove duplicates
df = df.drop_duplicates()


# Data type conversion
df["ID"] = df["ID"].astype(str)


# Normalization
scaler = MinMaxScaler()
df["Marks_MinMax"] = scaler.fit_transform(df[["Marks"]])
standard_scaler = StandardScaler()
df["Age_Standardized"] = standard_scaler.fit_transform(df[["Age"]])


print("\nCleaned and Normalized Data:")
print(df)
```

OUTPUT:



```
Original Data:
   ID     Name   Age  Marks Department
0   1    Alice  23.0   85.0        CSE
1   2      Bob  25.0   78.0        ECE
2   3  Charlie   NaN   90.0         ME
3   4    David  24.0    NaN      CIVIL
4   5     None  22.0   95.0         AI
5   5    David  22.0   95.0         AI

Handling Missing Values:
Detect missing:
 ID           0
Name         1
Age          1
Marks        1
Department   0
dtype: int64

After Filling Missing Values:
   ID     Name   Age  Marks Department
0   1    Alice  23.0   85.0        CSE
1   2      Bob  25.0   78.0        ECE
2   3  Charlie  23.2   90.0         ME
3   4    David  24.0   95.0      CIVIL
4   5  Unknown  22.0   95.0         AI
5   5    David  22.0   95.0         AI

After Removing Duplicates:
   ID     Name   Age  Marks Department
0   1    Alice  23.0   85.0        CSE
1   2      Bob  25.0   78.0        ECE
2   3  Charlie  23.2   90.0         ME
3   4    David  24.0   95.0      CIVIL
4   5  Unknown  22.0   95.0         AI
5   5    David  22.0   95.0         AI
```



```
After Data Type Conversion:
ID            object
Name          object
Age          float64
Marks        float64
Department    object
dtype: object

After Normalization:
  ID     Name   Age  Marks Department  Marks_MinMax  Age_Standardized
0  1    Alice  23.0   85.0        CSE      0.411765         -0.187867
1  2      Bob  25.0   78.0        ECE      0.000000          1.690806
2  3  Charlie  23.2   90.0         ME      0.705882          0.000000
3  4    David  24.0   95.0      CIVIL      1.000000          0.751469
4  5  Unknown  22.0   95.0         AI      1.000000         -1.127204
5  5    David  22.0   95.0         AI      1.000000         -1.127204
/tmp/ipython-input-2005816956.py:22: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df["Age"].fillna(df["Age"].mean(), inplace=True)
/tmp/ipython-input-2005816956.py:23: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df["Marks"].fillna(df["Marks"].mode()[0], inplace=True)
/tmp/ipython-input-2005816956.py:26: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df["Name"].fillna("Unknown", inplace=True)
```

Result

- Detected and filled missing values appropriately.
- Removed duplicate records.
- Converted data types correctly.
- Normalized numerical columns successfully.

Experiment No. 4 – EDA: Data Inspection and Analysis using Pandas

Date: 06-08-2025

Aim

To inspect and analyze data using Pandas through DataFrame viewing, filtering, and calculating descriptive statistics.

Steps

1. Created a sample DataFrame.
2. Viewed data, displayed info, first few rows, and column names.
3. Filtered data based on conditions.
4. Computed descriptive statistics including mean, median, mode, range, variance, and standard deviation.

Code Sample

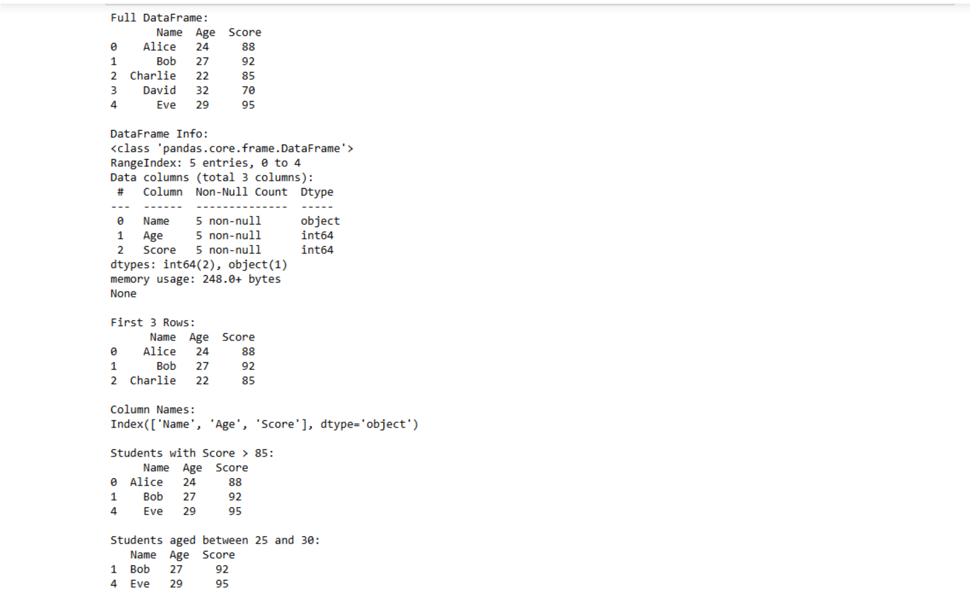
python

```python
import pandas as pd
import numpy as np
from scipy import stats

# Sample dataset
data = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
   'Age': [24, 27, 22, 32, 29],
   'Score': [88, 92, 85, 70, 95]
}
df = pd.DataFrame(data)

# Viewing data
print("Full DataFrame:\n", df)
print("\nDataFrame Info:")
print(df.info())
print("\nFirst 3 Rows:")
print(df.head(3))
print("\nColumn Names:")
print(df.columns)

# Filtering data
high_scores = df[df['Score'] > 85]
print("\nStudents with Score > 85:\n", high_scores)

age_range = df[(df['Age'] >= 25) & (df['Age'] <= 30)]
print("\nStudents aged between 25 and 30:\n", age_range)

# Descriptive statistics
print("\nDescriptive Statistics:")
print(df.describe())

mean_score = df['Score'].mean()
median_score = df['Score'].median()
mode_score = stats.mode(df['Score'], keepdims=False)

range_score = df['Score'].max() - df['Score'].min()
variance_score = df['Score'].var()
std_dev_score = df['Score'].std()

print(f"\nMean Score: {mean_score}")
print(f"Median Score: {median_score}")
print(f"Mode Score: {mode_score}")
print(f"Range of Scores: {range_score}")
```

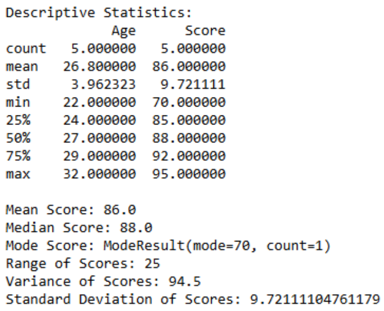
print(f"Variance of Scores: {variance_score}")

print(f"Standard Deviation of Scores: {std_dev_score}")

OUTPUT:



```
Full DataFrame:
       Name  Age  Score
0     Alice   24     88
1       Bob   27     92
2   Charlie   22     85
3     David   32     70
4       Eve   29     95

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    5 non-null      object
 1   Age     5 non-null      int64
 2   Score   5 non-null      int64
dtypes: int64(2), object(1)
memory usage: 248.0+ bytes
None

First 3 Rows:
      Name  Age  Score
0    Alice   24     88
1      Bob   27     92
2  Charlie   22     85

Column Names:
Index(['Name', 'Age', 'Score'], dtype='object')

Students with Score > 85:
     Name  Age  Score
0  Alice   24     88
1    Bob   27     92
4    Eve   29     95

Students aged between 25 and 30:
   Name  Age  Score
1  Bob   27     92
4  Eve   29     95
```



```
Descriptive Statistics:
             Age      Score
count   5.000000   5.000000
mean   26.800000  86.000000
std     3.962323   9.721111
min    22.000000  70.000000
25%    24.000000  85.000000
50%    27.000000  88.000000
75%    29.000000  92.000000
max    32.000000  95.000000

Mean Score: 86.0
Median Score: 88.0
Mode Score: ModeResult(mode=70, count=1)
Range of Scores: 25
Variance of Scores: 94.5
Standard Deviation of Scores: 9.72111104761179
```

Result

- Effectively viewed and inspected the data.
- Applied conditional filtering.
- Computed key descriptive statistics providing insight into data distribution.

Experiment No. 5 – EDA: Data Visualization with Matplotlib

Date: Not Provided

Aim

To understand and implement basic data visualization techniques using Matplotlib, including line charts, bar charts, and histograms as part of exploratory data analysis.

Steps

1. Created line chart displaying trends.
2. Created bar chart for categorical comparisons.
3. Created histogram to visualize data distribution.

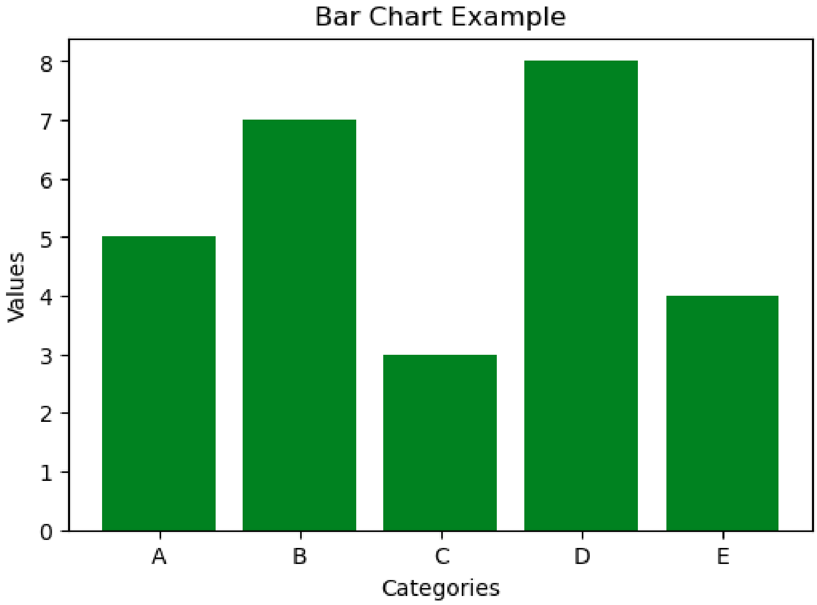
Code Sample

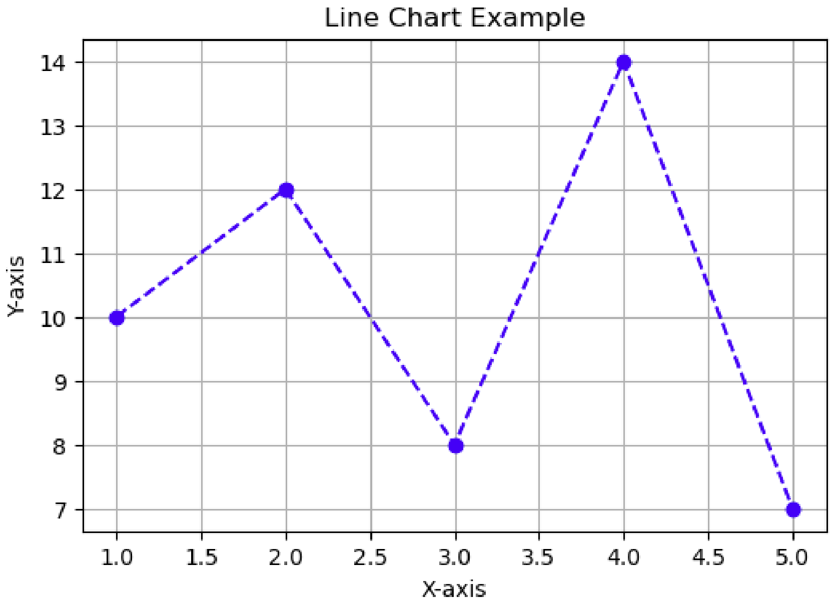
python

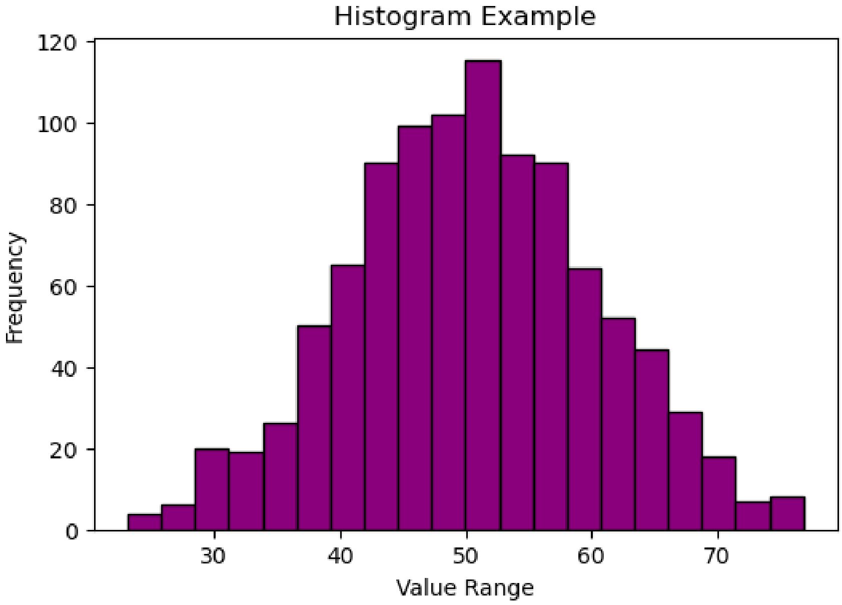
```
import matplotlib.pyplot as plt

import numpy as np


# Line Chart

x = [1, 2, 3, 4, 5]

y = [10, 12, 8, 14, 7]

plt.figure(figsize=(6, 4))

plt.plot(x, y, marker='o', color='blue', linestyle='--')

plt.title('Line Chart Example')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.grid(True)

plt.show()


# Bar Chart

categories = ['A', 'B', 'C', 'D', 'E']

values = [5, 7, 3, 8, 4]
```

```
plt.figure(figsize=(6, 4))

plt.bar(categories, values, color='green')

plt.title('Bar Chart Example')

plt.xlabel('Categories')

plt.ylabel('Values')

plt.show()


# Histogram

data = np.random.normal(50, 10, 1000) # mean=50, std=10

plt.figure(figsize=(6, 4))

plt.hist(data, bins=20, color='purple', edgecolor='black')

plt.title('Histogram Example')

plt.xlabel('Value Range')

plt.ylabel('Frequency')

plt.show()
```

OUTPUT:







Result

- Successfully implemented basic plotting techniques using Matplotlib.
- Visualized continuous data trends, categorical comparisons, and distribution frequency.