Lab Experiments

ROLL NO:231501186

EXP-1 Setting up the Python environment and libraries-Juypter Notebook

PROGRAM:

```
print("Hello, Google Colab!")


**Bold Text** and *Italic Text*

- Bullet 1
- Bullet 2

`Inline code`

[Google](https://www.google.com)



import ipywidgets as widgets
from IPython.display import display

# Slider example
slider = widgets.IntSlider(value=5, min=0, max=10, step=1,
description='Slider:')
display(slider)

# Textbox and button
text = widgets.Text(value='Hello', description='Name:')
button = widgets.Button(description='Greet')

def on_button_clicked(b):
    print(f"Hello, {text.value}!")

button.on_click(on_button_clicked)

display(text, button)
```

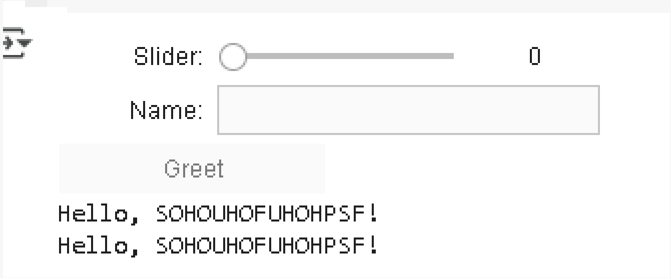
```
OUTPUT:

Hello, Google Colab!
```

Bold Text and *Italic Text*

- Bullet 1
- Bullet 2

Inline code

Google



```
Hello, SOHOUHOFUHOHPSF!
Hello, SOHOUHOFUHOHPSF!
```

EXP-2 Data Import and Export

PROGRAM:

```
import pandas as pd

# Replace with your CSV file URL
url = 
'https://raw.githubusercontent.com/kwaldenphd/eda-pandas/main/data
/titanic.csv'
df_csv = pd.read_csv(url)

# Display the first few rows
df_csv.head()
```

```
df_excel = pd.read_excel("/content/output.xlsx")  # Replace with
uploaded file name
print("Excel Data:")
print(df_excel.head())

from google.colab import drive
drive.mount('/content/drive')
```

```
# Create sample SQLite database and table (for demo)
engine = create_engine('sqlite://', echo=False)
df_sample = pd.DataFrame({
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35]
})
df_sample.to_sql("people", con=engine, index=False)

# Read from the SQL table
df_sql = pd.read_sql("SELECT * FROM people", engine)
print("SQL Data:")
print(df_sql)

# Read HTML table from a webpage
url =
"https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)"
tables = pd.read_html(url)

# Display the first table
df_web = tables[0]
print("Web Table Data:")
print(df_web.head())

import pandas as pd

# Sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'San Francisco', 'Los Angeles']}
df = pd.DataFrame(data)

# Export to Excel
df.to_excel('output1.xlsx', index=False)

OUTPUT:
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings,	female	38.0	1	0	PC 17599	71.2833	C85	C

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
				Mrs. John Bradley (Florence Briggs Th...								
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
Excel Data:
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
```

```
3     0          113803  53.1000  C123       S
4     0          373450   8.0500   NaN       S
```

addCode
addText
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
addCode
addText

```
SQL Data:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
```

```
Web Table Data:
                                                    0
0  Largest economies in the world by GDP (nominal...
```

EXP-3 Data Cleaning

PROGRAM:

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

# Sample dataset creation (you can replace this with your own
dataset)
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Edward',
'Alice'],
    'Age': [25, np.nan, 30, 22, 35, 25],
    'Salary': [50000, 60000, np.nan, 52000, 58000, 50000],
    'Department': ['HR', 'IT', 'IT', np.nan, 'Finance', 'HR'],
    'JoinDate': ['2010-01-10', '2012-05-15', '2011-08-20',
'2013-07-30', '2010-11-25', '2010-01-10']
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)

print("\nMissing values in each column:")
print(df.isnull().sum())

print("\nMissing values in each column:")
print(df.isnull().sum())
```

```
df.dropna(subset=['Salary'], inplace=True)

df.drop_duplicates(inplace=True)

df.drop(columns=['JoinDate'], inplace=True)

df['Age'] = df['Age'].astype(int)
df['Salary'] = df['Salary'].astype(int)

df['Department'] = df['Department'].astype('category')

scaler = StandardScaler()
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age',
'Salary']])
print("\nAfter Standardization:")
print(df[['Age', 'Salary']])

minmax_scaler = MinMaxScaler()
df[['Age', 'Salary']] = minmax_scaler.fit_transform(df[['Age',
'Salary']])
print("\nAfter Min-Max Scaling:")
print(df[['Age', 'Salary']])
```

OUTPUT:

```
Original DataFrame:
      Name   Age   Salary Department    JoinDate
0    Alice  25.0  50000.0         HR  2010-01-10
1      Bob   NaN  60000.0         IT  2012-05-15
2  Charlie  30.0      NaN         IT  2011-08-20
3    David  22.0  52000.0        NaN  2013-07-30
4   Edward  35.0  58000.0    Finance  2010-11-25
5    Alice  25.0  50000.0         HR  2010-01-10

Missing values in each column:
Name          0
Age           1
Salary        1
Department    1
JoinDate      0
dtype: int64

/tmp/ipython-input-4-2707674413.py:1: 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-4-2707674413.py:2: 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['Department'].fillna(df['Department'].mode()[0],
inplace=True)
```

```
After Standardization:

     Age    Salary

0 -0.467257 -1.212678

1 -0.051917  1.212678

3 -1.090266 -0.727607

4  1.609440  0.727607


After Min-Max Scaling:

     Age  Salary

0  0.230769    0.0

1  0.384615    1.0

3  0.000000    0.2

4  1.000000    0.8
```

EXP-4 -Data Inspection and Analysis

PROGRAM:

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_iris


# Load the Iris dataset from sklearn

iris = load_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Add the species column

df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

df.head()       # View first 5 rows

df.tail()       # View last 5 rows

df.info()       # Summary: data types, nulls

df.describe()    # Quick stats for numerical columns

df.columns     #colummn names

df.shape        # Rows and columns count


df[df['species'] == 'setosa']


df[(df['species'] == 'setosa') & (df['sepal length (cm)'] > 5.0)]

df[['sepal length (cm)', 'sepal width (cm)']]
```

```
df['sepal length (cm)'].mean()    # Mean
df['sepal length (cm)'].median()  # Median
df['sepal length (cm)'].mode()    # Mode (returns a Series)
df['sepal length (cm)'].min(), df['sepal length (cm)'].max()  # Range
df['sepal length (cm)'].var()       # Variance
df['sepal length (cm)'].std()       # Standard Deviation
df.corr(numeric_only=True)
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
(150, 4)
```

	sepal length (cm)	**sepal width (cm)**	**petal length (cm)**	**petal width (cm)**	**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
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa

	sepal length (cm)	**sepal width (cm)**	**petal length (cm)**	**petal width (cm)**	**specie s**
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa
20	5.4	3.4	1.7	0.2	setosa
21	5.1	3.7	1.5	0.4	setosa
22	4.6	3.6	1.0	0.2	setosa
23	5.1	3.3	1.7	0.5	setosa
24	4.8	3.4	1.9	0.2	setosa
25	5.0	3.0	1.6	0.2	setosa
26	5.0	3.4	1.6	0.4	setosa
27	5.2	3.5	1.5	0.2	setosa
28	5.2	3.4	1.4	0.2	setosa
29	4.7	3.2	1.6	0.2	setosa

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
30	4.8	3.1	1.6	0.2	setosa
31	5.4	3.4	1.5	0.4	setosa
32	5.2	4.1	1.5	0.1	setosa
33	5.5	4.2	1.4	0.2	setosa
34	4.9	3.1	1.5	0.2	setosa
35	5.0	3.2	1.2	0.2	setosa
36	5.5	3.5	1.3	0.2	setosa
37	4.9	3.6	1.4	0.1	setosa
38	4.4	3.0	1.3	0.2	setosa
39	5.1	3.4	1.5	0.2	setosa
40	5.0	3.5	1.3	0.3	setosa
41	4.5	2.3	1.3	0.3	setosa
42	4.4	3.2	1.3	0.2	setosa
43	5.0	3.5	1.6	0.6	setosa
44	5.1	3.8	1.9	0.4	setosa

	sepal length (cm)	**sepal width (cm)**	**petal length (cm)**	**petal width (cm)**	**species**
45	4.8	3.0	1.4	0.3	setosa
46	5.1	3.8	1.6	0.2	setosa
47	4.6	3.2	1.4	0.2	setosa
48	5.3	3.7	1.5	0.2	setosa
49	5.0	3.3	1.4	0.2	setosa

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
10	5.4	3.7	1.5	0.2	setosa
14	5.8	4.0	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa
20	5.4	3.4	1.7	0.2	setosa
21	5.1	3.7	1.5	0.4	setosa
23	5.1	3.3	1.7	0.5	setosa

27	5.2	3.5	1.5	0.2	setosa
28	5.2	3.4	1.4	0.2	setosa
31	5.4	3.4	1.5	0.4	setosa
32	5.2	4.1	1.5	0.1	setosa
33	5.5	4.2	1.4	0.2	setosa
36	5.5	3.5	1.3	0.2	setosa
39	5.1	3.4	1.5	0.2	setosa
44	5.1	3.8	1.9	0.4	setosa
46	5.1	3.8	1.6	0.2	setosa
48	5.3	3.7	1.5	0.2	setosa

sepal length (cm)	**sepal width (cm)**	
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...	...	...
145	6.7	3.0

	sepal length (cm)	**sepal width (cm)**
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4
149	5.9	3.0

150 rows × 2 columns

	sepal length (cm)
0	5.0

dtype: float64

0.8280661279778629

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

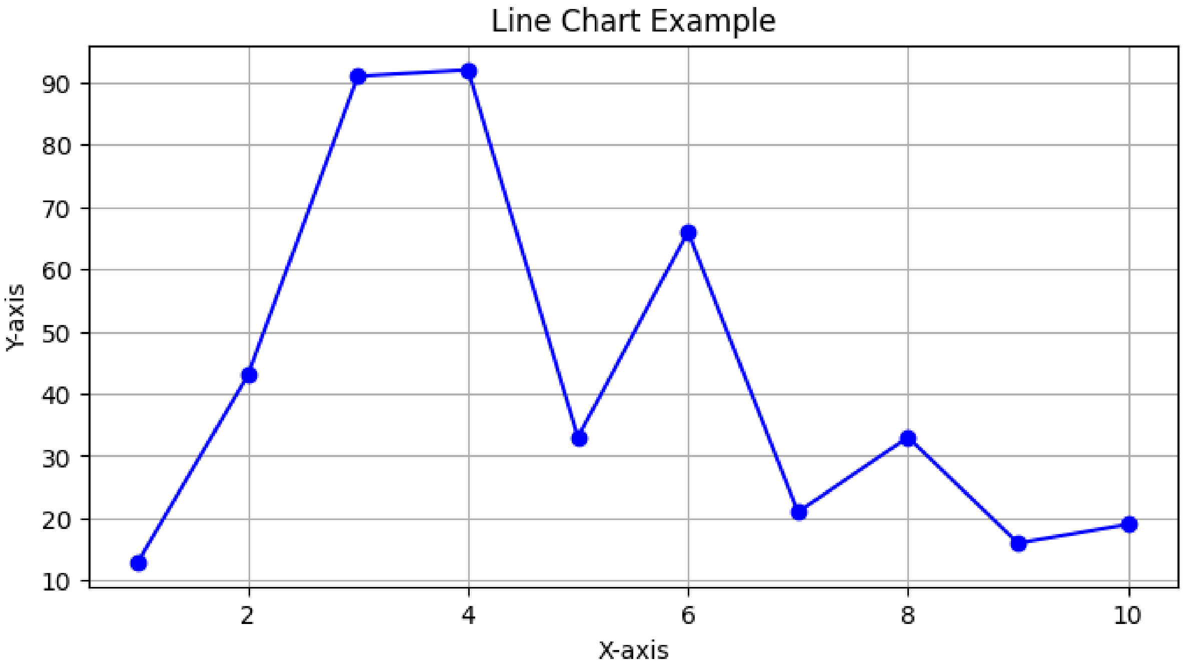
EXP-5 Data Visualization with matplotlib

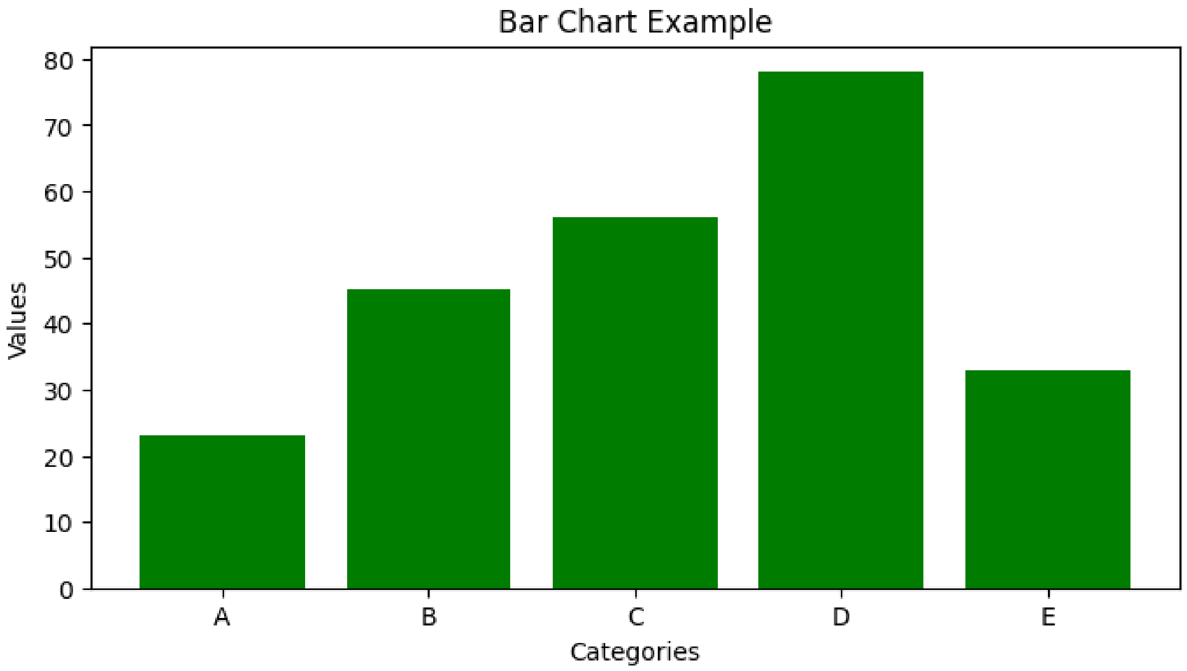
PROGRAM:

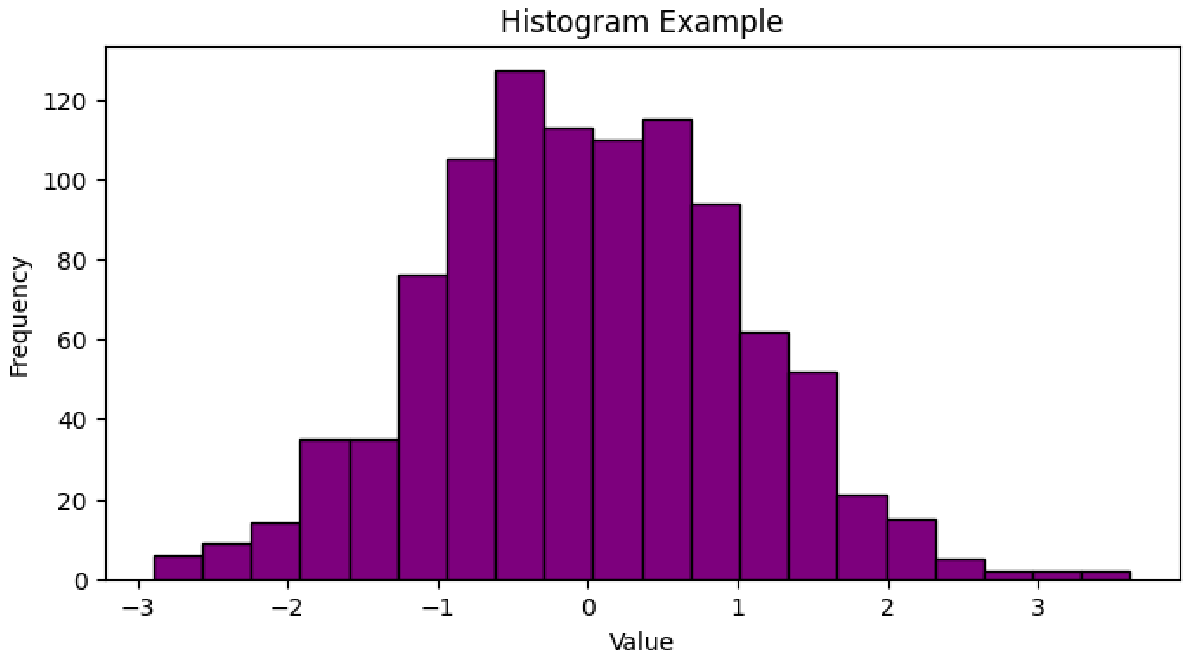
```
# EDA - Data Visualization with Matplotlib

# Install matplotlib if not already (usually preinstalled in Colab)
# !pip install matplotlib

import matplotlib.pyplot as plt
import numpy as np


# Sample data
x = np.arange(1, 11)
y = np.random.randint(10, 100, size=10)
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 78, 33]
hist_data = np.random.randn(1000)  # Normal distribution
# 1. Line Chart
plt.figure(figsize=(8, 4))
plt.plot(x, y, marker='o', linestyle='-', color='blue')
plt.title('Line Chart Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid(True)
plt.show()
```

```
# 2. Bar Chart

plt.figure(figsize=(8, 4))

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

plt.title('Bar Chart Example')

plt.xlabel('Categories')

plt.ylabel('Values')

plt.show()


# 3. Histogram

plt.figure(figsize=(8, 4))

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

plt.title('Histogram Example')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()
```

OUTPUT




Bar Chart Example
Values
0
10
20
30
40
50
60
70
80
A
B
C
D
E
Categories


Histogram Example
Frequency
0
20
40
60
80
100
120
−3
−2
−1
0
1
2
3
Value