Lab experiments

Experiment 1:

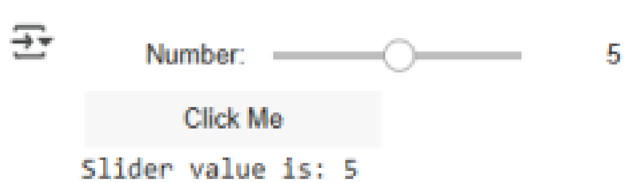
Setting up the Python environment and libraries-Juypter Notebook:
code:

```
1.print("Hello, Jupyter Notebook!")
```

O/P:Hello, Jupyter Notebook!

```
2.import ipywidgets as widgets
from IPython.display import display

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

button = widgets.Button(description="Click Me")
display(button)

def on_click(b):
 print(f"Slider value is: {slider.value}")

button.on_click(on_click)
```

O/P:



Experiment - 2:

EDA-Data Import and Export:

Code:

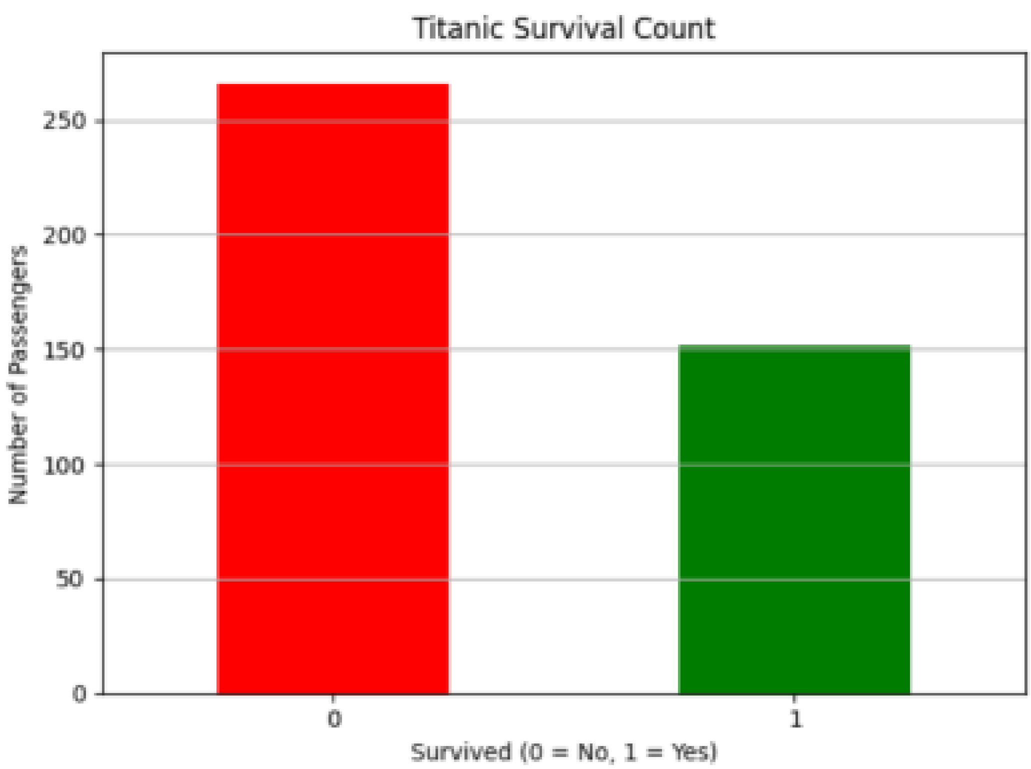
```
import pandas as pd
from sqlalchemy import create_engine
import sqlite3
import requests
```

```python
import matplotlib.pyplot as plt

# Load CSV
df_csv = pd.read_csv("/content/gender_submission.csv")
print("CSV Data:\n", df_csv.head())

# Export to Excel
excel_path = "titanic_full.xlsx"
df_csv.to_excel(excel_path, index=False)
print(f"Entire CSV exported to: {excel_path}")

# Read Excel file and preview
df_excel = pd.read_excel("/content/titanic_full.xlsx")
print("Excel Data Preview:")
display(df_excel.head()) # ✅ Now appears in a nice box

# SQL setup and query
engine = create_engine("sqlite:///:memory:")
df_csv.to_sql("titanic", engine, if_exists="replace", index=False)

query = """
SELECT Survived, COUNT(*) AS count
FROM titanic
GROUP BY Survived
"""
df_sql = pd.read_sql_query(query, engine)
print("SQL Query Result:\n", df_sql)
# Web scraping
url = "https://en.wikipedia.org/wiki/Titanic"
try:
 tables = pd.read_html(url)
 print("\nWeb Scraped Table Example:\n", tables[0].head())
except:
 print("\nUnable to scrape table from the web.")

# Plot survival count
df_sql.plot(kind='bar', x='Survived', y='count', legend=False,
color=['red', 'green'])
plt.title("Titanic Survival Count")
plt.xlabel("Survived (0 = No, 1 = Yes)")
plt.ylabel("Number of Passengers")
plt.xticks(rotation=0)
plt.grid(axis='y')
```

```
plt.tight_layout()
plt.show()

# Export SQL result to Excel
df_sql.to_excel("titanic_survival_summary.xlsx", index=False)
print("Survival summary exported to: titanic_survival_summary.xlsx")
```

Output:

```
SQL Query Result:
   Survived  count
0         0    266
1         1    152

Web Scraped Table Example:
                                                   0  \
0  RMS Titanic departing Southampton for the only...
1                           Location of Titanic wreck
2                                             History
3                                      United Kingdom
4                                                Name

                                                  1
0  RMS Titanic departing Southampton for the only...
1                           Location of Titanic wreck
2                                             History
3                                      United Kingdom
4                                         RMS Titanic
```




```
Survival summary exported to: titanic_survival_summary.xlsx
```

EXPERIMENT - 3

EDA-Data Cleaning:

CODE:

```
# �� Step 1: Import required libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# �� Step 2: Create a sample DataFrame
data = {
 'Name': ['Alice', 'Bob', 'Charlie', np.nan, 'Eve', 'Alice'],
'Age': [25, np.nan, 30, 22, 29, 25],
 'Salary': [50000, 60000, 55000, 52000, np.nan, 50000],
'Department': ['HR', 'IT', 'IT', 'HR', 'Finance', 'HR'] }
df = pd.DataFrame(data)
print("�� Original DataFrame:\n")
print(df)

# �� Step 3: Detect missing values
print("\n�� Missing values before cleaning:")
print(df.isnull().sum())

# �� Step 4: Fill missing values using forward fill (new
syntax) df.ffill(inplace=True)

# �� Step 5: Drop any remaining rows with missing
values df.dropna(inplace=True)

# �� Step 6: Show missing values after cleaning
print("\n✅ Missing values after filling and dropping:")
print(df.isnull().sum())

# �� Step 7: Remove duplicate rows
print("\n�� Duplicate rows before removal:", df.duplicated().sum())
df.drop_duplicates(inplace=True)
print("✅ Duplicate rows after removal:", df.duplicated().sum())

# �� Step 8: Remove unnecessary columns
df.drop(columns=['Name'], inplace=True)

# �� Step 9: Show data types before conversion
print("\n�� Data types before conversion:")
```

```
print(df.dtypes)

# �� Step 10: Convert data types for consistency
df['Age'] = df['Age'].astype(int)
df['Salary'] = df['Salary'].astype(int)

# �� Step 11: Show data types after conversion
print("\n✅ Data types after conversion:")
print(df.dtypes)

# �� Step 12: Normalize numerical columns
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
print("\n�� Numeric columns to normalize:", numeric_cols)

# Standardization (Z-score)
scaler_std = StandardScaler()
df_standardized = df.copy()
df_standardized[numeric_cols] =
scaler_std.fit_transform(df_standardized[numeric_cols])

# Min-Max Scaling (0-1 range)
scaler_mm = MinMaxScaler()
df_minmax = df.copy()
df_minmax[numeric_cols] = scaler_mm.fit_transform(df_minmax[numeric_cols])

# �� Step 13: Save cleaned data to CSV
df.to_csv("cleaned_data.csv", index=False)
df_standardized.to_csv("standardized_data.csv", index=False)
df_minmax.to_csv("minmax_scaled_data.csv", index=False)

# �� Step 14: Final Outputs
print("\n✅ Final Cleaned DataFrame:\n")
print(df)

print("\n✅ Standardized DataFrame:\n")
print(df_standardized)

print("\n✅ Min-Max Scaled DataFrame:\n")
print(df_minmax)

print("\n�� Files saved: cleaned_data.csv,
standardized_data.csv, minmax_scaled_data.csv")
```

OUTPUT:

```
✅ Data types after conversion:
Age              int64
Salary           int64
Department      object
dtype: object

🔢 Numeric columns to normalize: ['Age', 'Salary']

✅ Final Cleaned DataFrame:

   Age  Salary Department
0   25   50000         HR
1   25   60000         IT
2   30   55000         IT
3   22   52000         HR
4   29   52000    Finance

✅ Standardized DataFrame:

        Age    Salary Department
0 -0.410152 -1.089725         HR
1 -0.410152  1.777972         IT
2  1.298813  0.344124         IT
3 -1.435530 -0.516185         HR
4  0.957020 -0.516185    Finance

✅ Min-Max Scaled DataFrame:

     Age  Salary Department
0  0.375     0.0         HR
1  0.375     1.0         IT
2  1.000     0.5         IT
3  0.000     0.2         HR
4  0.875     0.2    Finance
```

```
📊 Original DataFrame:

      Name   Age   Salary Department
0    Alice  25.0  50000.0         HR
1      Bob   NaN  60000.0         IT
2  Charlie  30.0  55000.0         IT
3      NaN  22.0  52000.0         HR
4      Eve  29.0      NaN    Finance
5    Alice  25.0  50000.0         HR

🔍 Missing values before cleaning:
Name          1
Age           1
Salary        1
Department    0
dtype: int64

✅ Missing values after filling and dropping:
Name          0
Age           0
Salary        0
Department    0
dtype: int64

🟰 Duplicate rows before removal: 1
✅ Duplicate rows after removal: 0

🧾 Data types before conversion:
Age           float64
Salary        float64
Department     object
dtype: object
```

EXPERIMENT - 4:

EDA-Data Inspection and Analysis:

CODE:

```
# �� Step 1: Import necessary libraries
import pandas as pd
import numpy as np

# �� Step 2: Create a sample DataFrame
data = {
 'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
'Age': [25, 30, 22, 28, 35],
 'Salary': [50000, 60000, 52000, 58000, 65000],
```

```
'Department': ['HR', 'IT', 'HR', 'Finance', 'IT'] }
df = pd.DataFrame(data)

# �� Step 3: View and inspect the DataFrame
print("�� First 5 rows of the
DataFrame:\n") print(df.head())

print("\n�� Info about DataFrame:")
print(df.info())

print("\n�� Shape of DataFrame:", df.shape)
print("\n�� Column Names:",
df.columns.tolist())

# �� Step 4: Filter and subset data using
conditions print("\n�� Employees with Salary >
55000:") high_salary = df[df['Salary'] > 55000]
print(high_salary)

print("\n��     Employees     from     IT
Department:")         it_employees         =
df[df['Department']         ==         'IT']
print(it_employees)
# �� Step 5: Descriptive statistics - Central Tendency
print("\n�� Mean Age:", df['Age'].mean())
print("�� Median Age:", df['Age'].median())
print("�� Mode Age:", df['Age'].mode()[0])

# �� Step 6: Descriptive statistics - Dispersion
range_age = df['Age'].max() - df['Age'].min()
print("\n�� Range of Age:", range_age)

print("�� Variance of Age:", df['Age'].var())
print("�� Standard Deviation of Age:", df['Age'].std())

range_salary = df['Salary'].max() - df['Salary'].min()
print("\n�� Range of Salary:", range_salary)

print("�� Variance of Salary:", df['Salary'].var())
print("�� Standard Deviation of Salary:", df['Salary'].std())
```

OUTPUT:

```
📏 Shape of DataFrame: (5, 4)

📚 Column Names: ['Name', 'Age', 'Salary', 'Department']

📌 Employees with Salary > 55000:
    Name  Age  Salary Department
1    Bob   30   60000         IT
3  David   28   58000    Finance
4    Eve   35   65000         IT

📌 Employees from IT Department:
  Name  Age  Salary Department
1  Bob   30   60000         IT
4  Eve   35   65000         IT

📊 Mean Age: 28.0
📊 Median Age: 28.0
📊 Mode Age: 22

📈 Range of Age: 13
📈 Variance of Age: 24.5
📈 Standard Deviation of Age: 4.949747468305833

📈 Range of Salary: 15000
📈 Variance of Salary: 37000000.0
📈 Standard Deviation of Salary: 6082.762530298222
```

```
🔍 First 5 rows of the DataFrame:

      Name  Age  Salary Department
0    Alice   25   50000         HR
1      Bob   30   60000         IT
2  Charlie   22   52000         HR
3    David   28   58000    Finance
4      Eve   35   65000         IT

📋 Info about DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Name        5 non-null      object
 1   Age         5 non-null      int64
 2   Salary      5 non-null      int64
 3   Department  5 non-null      object
dtypes: int64(2), object(2)
memory usage: 292.0+ bytes
None

📏 Shape of DataFrame: (5, 4)

📚 Column Names: ['Name', 'Age', 'Salary', 'Department']

📌 Employees with Salary > 55000:
    Name  Age  Salary Department
1    Bob   30   60000         IT
3  David   28   58000    Finance
4    Eve   35   65000         IT
```

Exp 5:

EDA-DATA VISUALIZATION USING MATPLOTLIB

CODE:

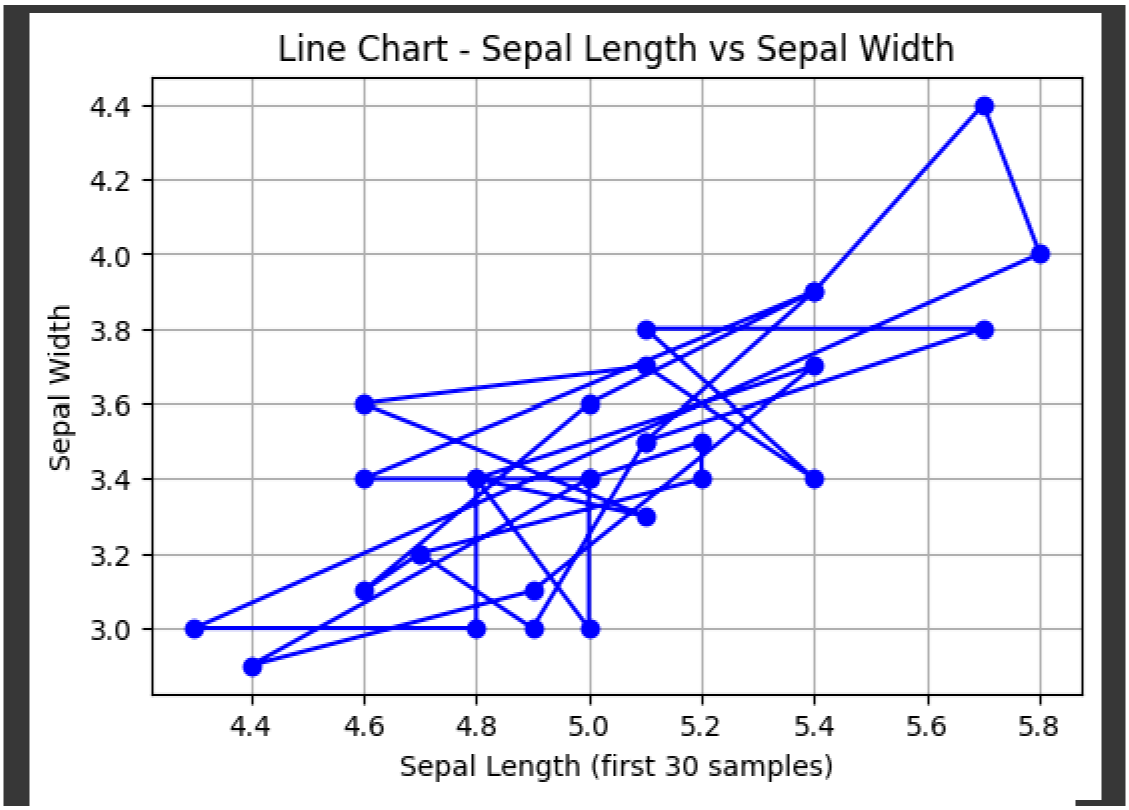
```
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

import numpy as np



# Load Iris dataset (built-in)

iris = load_iris()
```

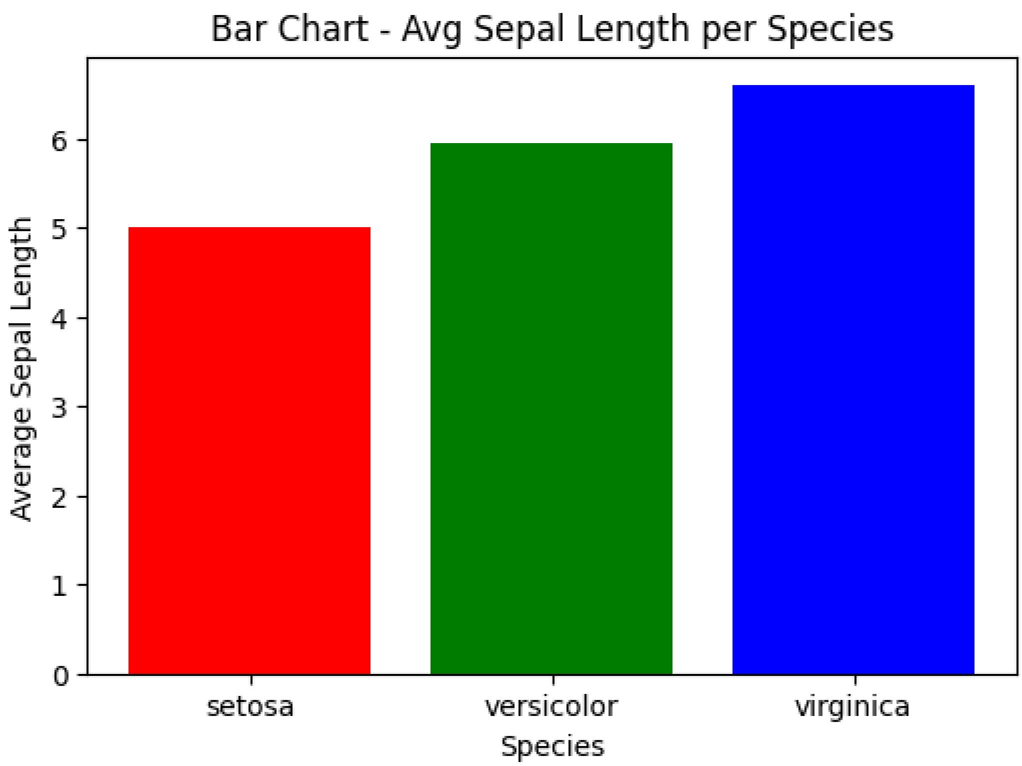
```
data = iris.data
feature_names = iris.feature_names


# Take first feature (sepal length) and second feature (sepal width)
sepal_length = data[:, 0]
sepal_width = data[:, 1]


# --------------------
# 1. Line Chart
# --------------------
plt.figure(figsize=(6,4))
plt.plot(sepal_length[:30], sepal_width[:30], marker='o', linestyle='-',
color='blue')
plt.title("Line Chart - Sepal Length vs Sepal Width")
plt.xlabel("Sepal Length (first 30 samples)")
plt.ylabel("Sepal Width")
plt.grid(True)
plt.show()


# --------------------
```

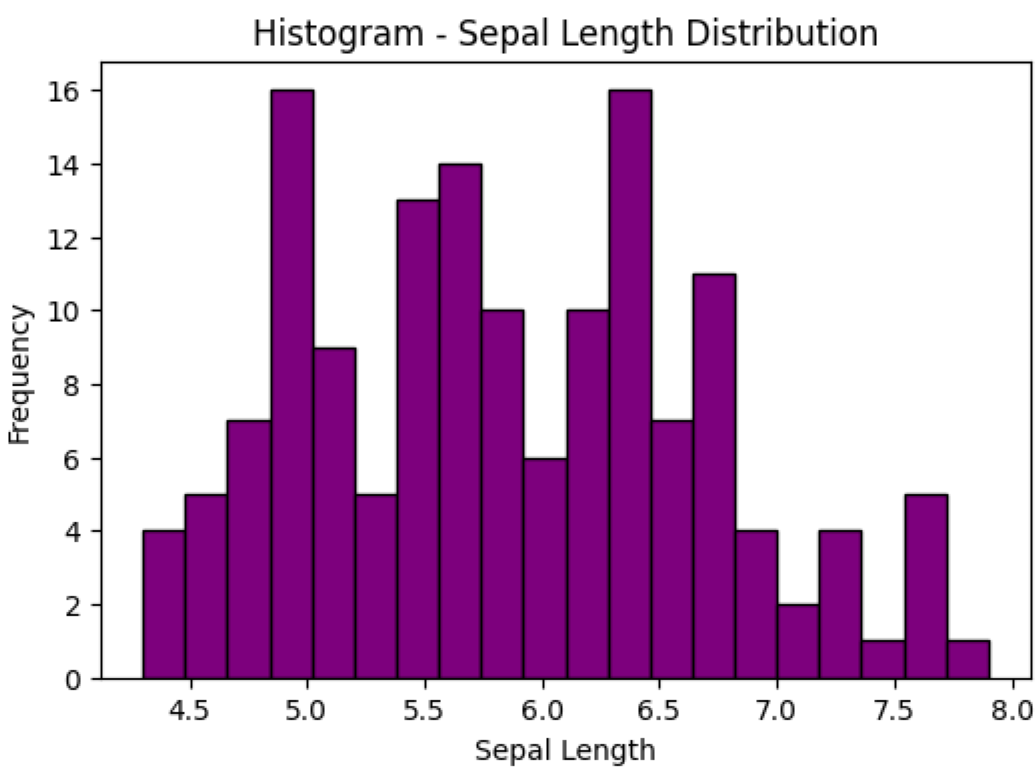
```
# 2. Bar Chart
# --------------------
# Average sepal length by species
species = iris.target
unique_species = np.unique(species)
avg_sepal_length = [sepal_length[species == s].mean() for s in unique_species]

plt.figure(figsize=(6,4))
plt.bar(iris.target_names, avg_sepal_length, color=['red', 'green', 'blue'])
plt.title("Bar Chart - Avg Sepal Length per Species")
plt.xlabel("Species")
plt.ylabel("Average Sepal Length")
plt.show()

# --------------------
# 3. Histogram
# --------------------
plt.figure(figsize=(6,4))
plt.hist(sepal_length, bins=20, color='purple', edgecolor='black')
plt.title("Histogram - Sepal Length Distribution")
```

```
plt.xlabel("Sepal Length")

plt.ylabel("Frequency")

plt.show()
```

OUTPUT:




Bar Chart - Avg Sepal Length per Species
Average Sepal Length
0
1
2
3
4
5
6
setosa
versicolor
virginica
Species


Histogram - Sepal Length Distribution
Frequency
0
2
4
6
8
10
12
14
16
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Sepal Length