EXP.NO : 1 (setting up python environment and lib-jupyter notebook)

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:')

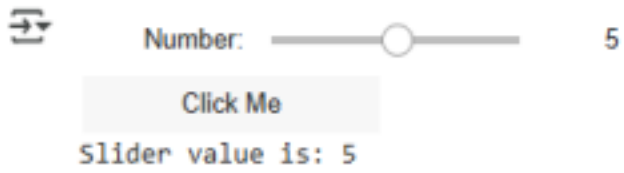
display(slider)

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:



EXP.NO : 2 (EDA – data import and export):

import pandas as pd

from sqlalchemy import create_engine

import sqlite3

import requests

```
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()

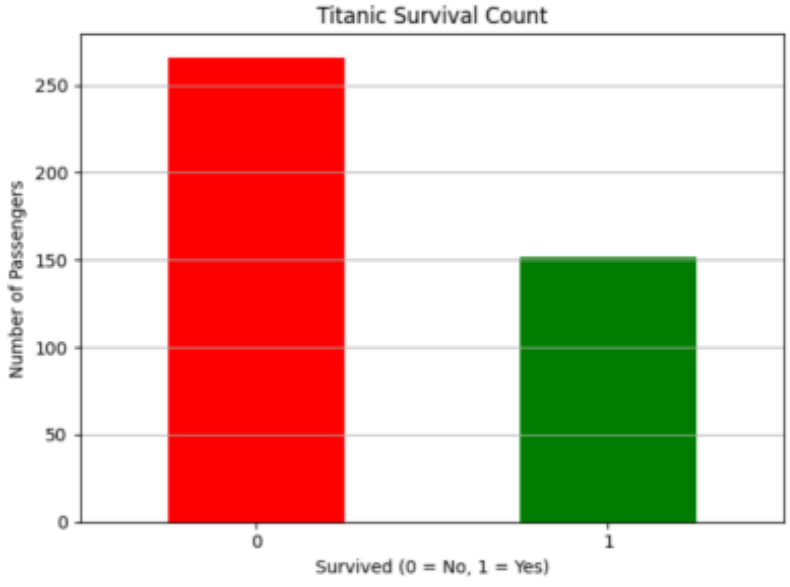
plt.show()

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
```

EXP.NO : 3 (EDA – data cleaning):

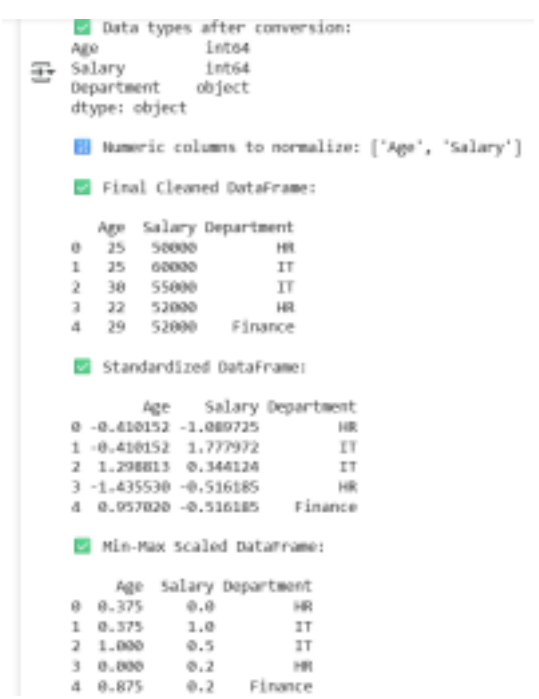
```
# Step 1: Import required libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler



def main():
    # 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("\nMissing values before cleaning:")
    print(df.isnull().sum())

    # Step 4: Fill missing values using forward fill
    df = df.ffill()
```

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

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

# Step 7: Remove duplicate rows
print("\nDuplicate rows before removal:", df.duplicated().sum())
df = df.drop_duplicates()
print("Duplicate rows after removal:", df.duplicated().sum())

# Step 8: Remove unnecessary columns
if "Name" in df.columns:
   df = df.drop(columns=["Name"])

# Step 9: Show data types before conversion
print("\nData 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("\nData types after conversion:")
print(df.dtypes)
```

```
    # Step 12: Normalize numerical columns
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    print("\nNumeric 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("\nFinal Cleaned DataFrame:\n")
    print(df)

    print("\nStandardized DataFrame:\n")
    print(df_standardized)
```

```
    print("\nMin-Max Scaled DataFrame:\n")
    print(df_minmax)
print("\nFiles saved: cleaned_data.csv, standardized_data.csv, minmax_scaled_data.csv")
if __name__ == "__main__":
    main()
```

output:



EXP.NO : 4 (EDA – data inspection and analysis):

```
# 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("\nInfo about DataFrame:")
print(df.info())

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

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

```
print("\nEmployees from IT Department:")
it_employees = df[df['Department'] == 'IT']
print(it_employees)

# Step 5: Descriptive statistics - Central Tendency
print("\nMean 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("\nRange 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("\nRange 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: 6082.76253029822
```

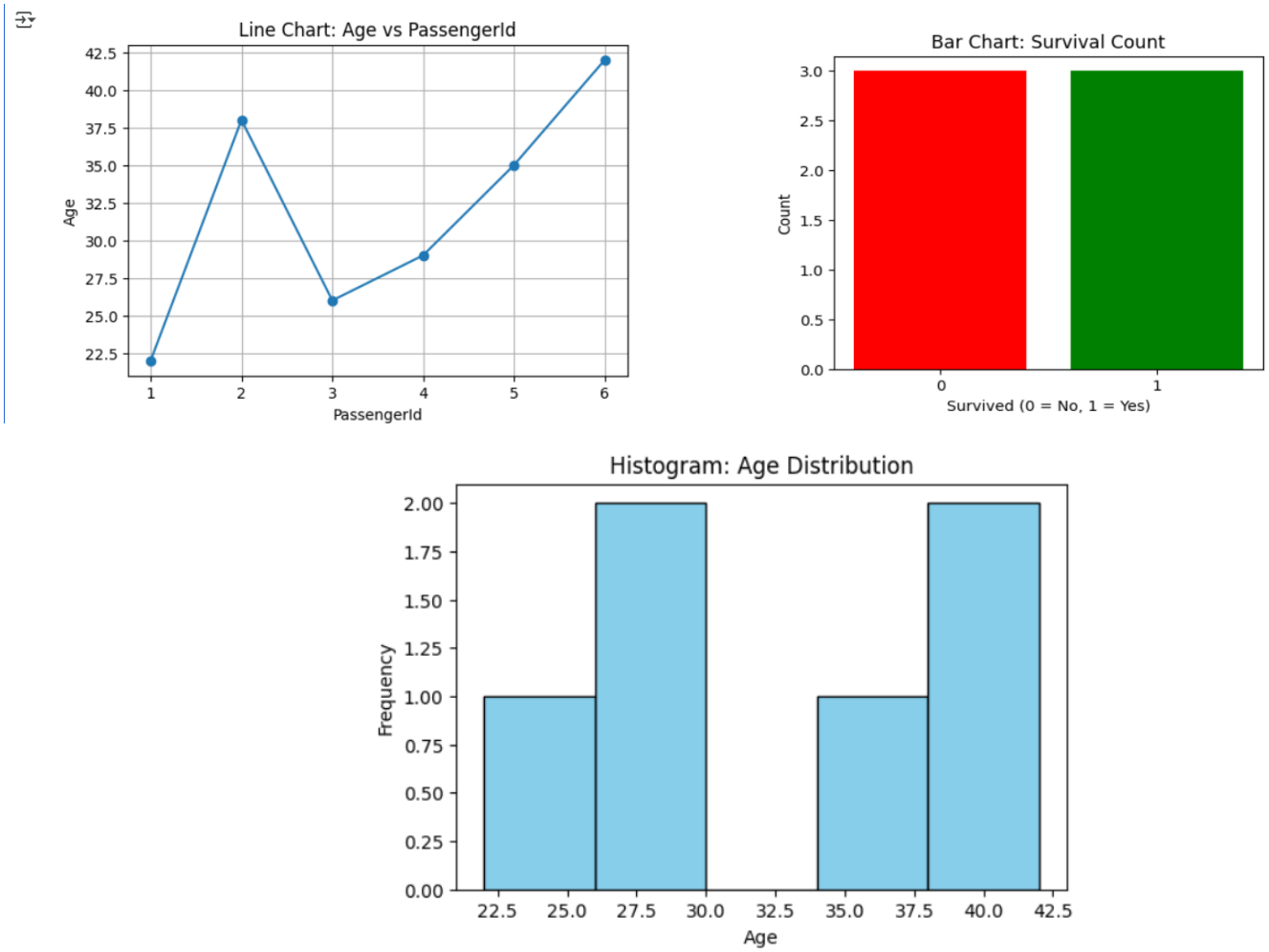
```
First 5 rows of the DataFrame:

      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.NO : 5 (EDA – data visualization with matplotlib):

```
# Step 1: Import libraries
import pandas as pd
import matplotlib.pyplot as plt
# Step 2: Create a sample DataFrame
data = {
'PassengerId': [1, 2, 3, 4, 5, 6],
'Survived': [0, 1, 1, 0, 1, 0],
'Name': ['Allen', 'Braund', 'Cumings', 'Dawson', 'Evans', 'Frank'],
'Age': [22, 38, 26, 29, 35, 42],
}
df = pd.DataFrame(data)
# Step 3: Line chart – Age vs PassengerId
plt.figure(figsize=(6, 4))
plt.plot(df['PassengerId'], df['Age'], marker='o')
plt.title('Line Chart: Age vs PassengerId')
plt.xlabel('PassengerId')
plt.ylabel('Age')
plt.grid(True)
plt.show()
# Step 4: Bar chart – Survived counts
survival_counts = df['Survived'].value_counts()
plt.figure(figsize=(5, 4))
plt.bar(survival_counts.index, survival_counts.values, color=['red', 'green'])
plt.title('Bar Chart: Survival Count')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Count')
```

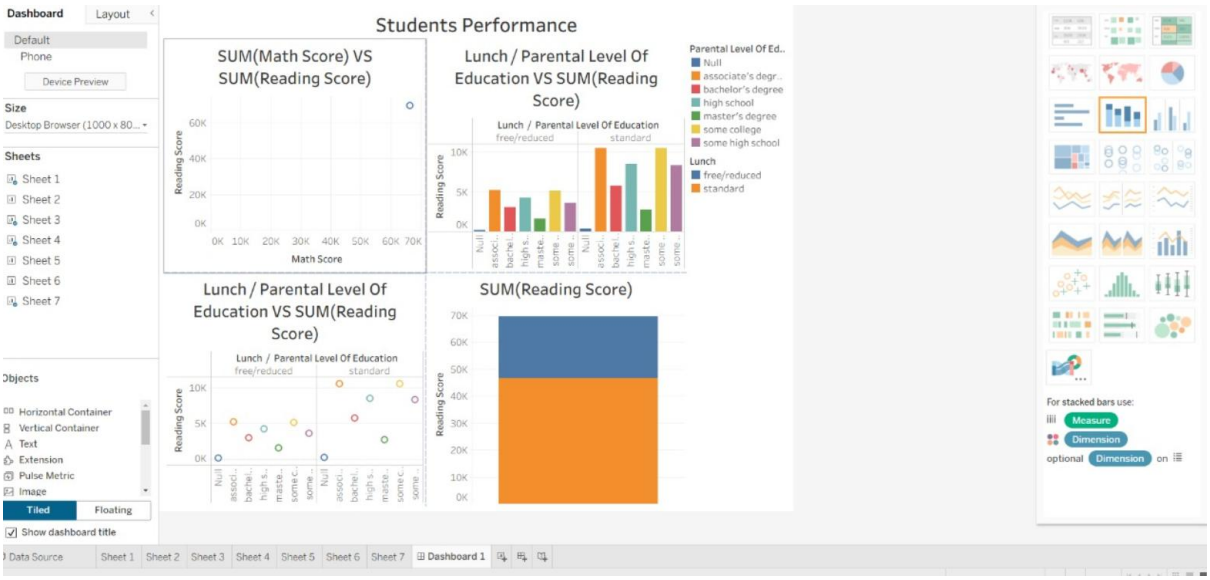
```
plt.xticks([0, 1])
plt.show()
# Step 5: Histogram – Age distribution
plt.figure(figsize=(6, 4))
plt.hist(df['Age'], bins=5, color='skyblue', edgecolor='black')
plt.title('Histogram: Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

output:



EXP.NO : 6:

EXP.NO : 7:

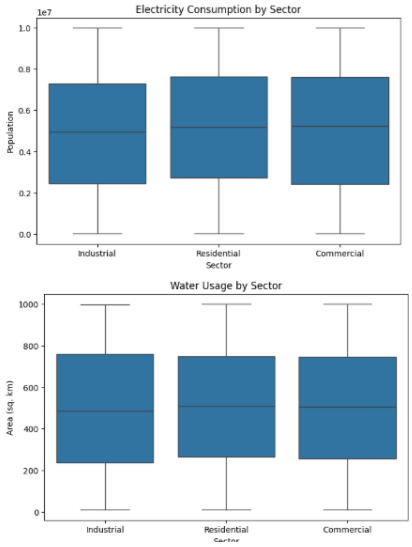


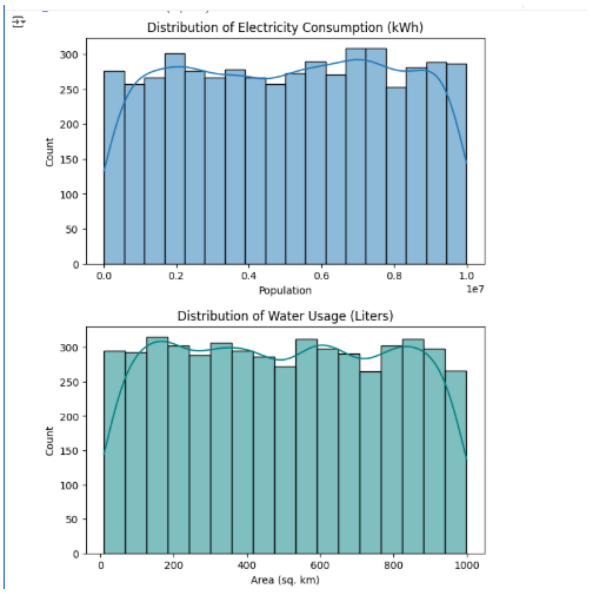
EXP NO 8 :

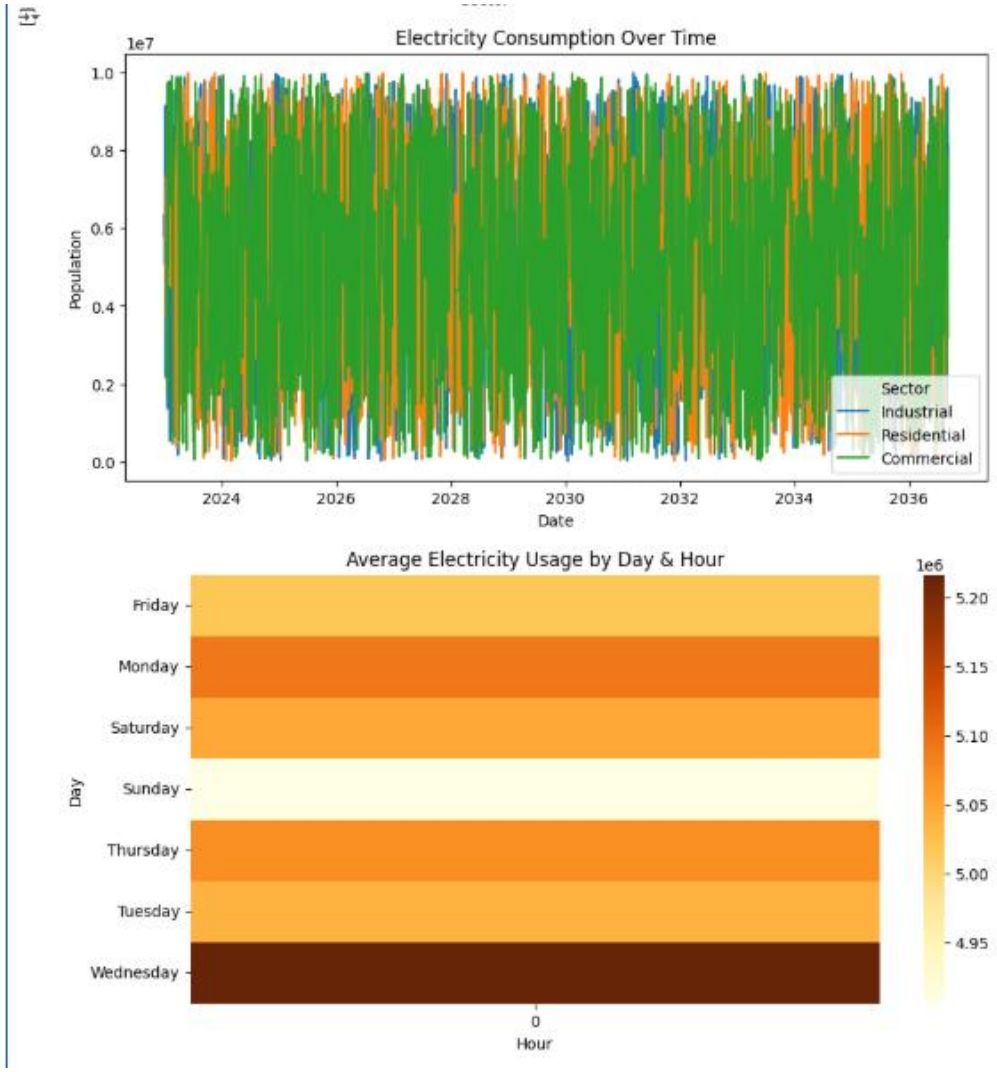
[5] ✓ 2s

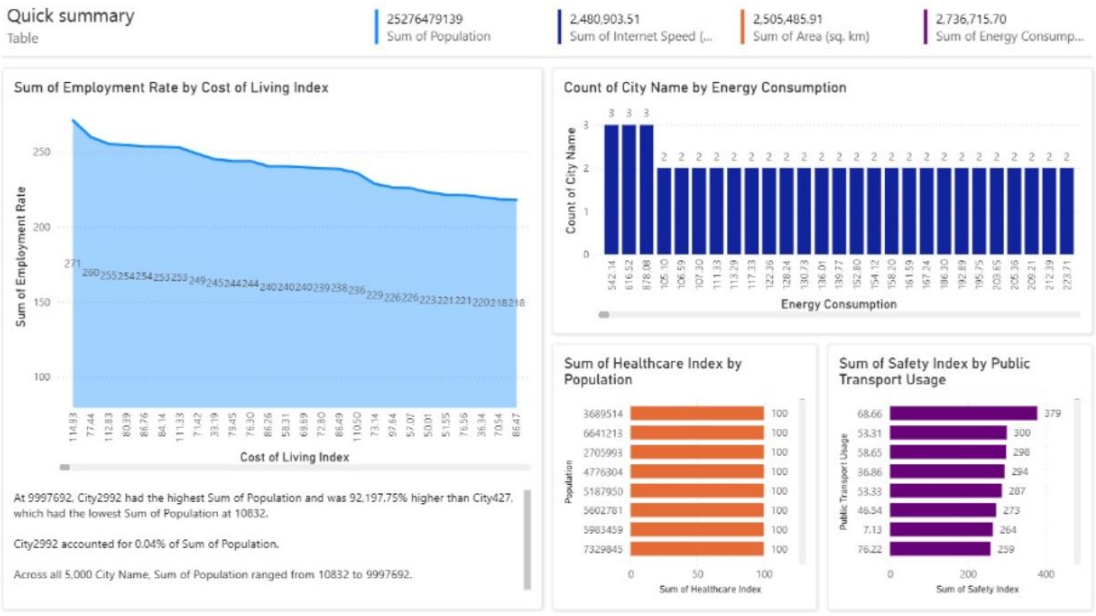
```
# 🧩 Step 1: Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ------------------------------------------------------------
# 🧩 Step 2: Load the dataset
df = pd.read_csv("/content/smart_city_dataset (1).csv")

# Check columns and basic info
print("✅ Columns in the dataset:")
print(df.columns.tolist())
print("\nDataset shape:", df.shape)
print("\nMissing values:\n", df.isna().sum())

# ------------------------------------------------------------
# 🧩 Step 3: Identify the correct date/time column
# Replace below with the actual date/time column name in your dataset
# (example: 'timestamp', 'date_time', 'recorded_at', etc.)
date_col = None
for col in df.columns:
    if 'date' in col.lower() or 'time' in col.lower():
        date_col = col
        break

if date_col:
    print(f"\n🕒 Using '{date_col}' as Date column.")
    df['Date'] = pd.to_datetime(df[date_col], errors='coerce')
else:
    print("\n⚠️ No date/time column found! Creating synthetic date column.")
    df['Date'] = pd.date_range(start='2023-01-01', periods=len(df), freq='D')

# ------------------------------------------------------------
# 🧩 Step 4: Basic data cleaning
# Remove duplicates
df = df.drop_duplicates()

# Fill numeric columns with median, categorical with mode
for col in df.select_dtypes(include=[np.number]).columns:
    df[col] = df[col].fillna(df[col].median())

for col in df.select_dtypes(exclude=[np.number]).columns:
    df[col] = df[col].fillna(df[col].mode()[0])

print("\n✅ Missing values after cleaning:", df.isna().sum().sum())

# ------------------------------------------------------------
# 🧩 Step 5: Feature engineering
df['Hour'] = df['Date'].dt.hour if df['Date'].dt.hour.notna().any() else np.nan
df['Day'] = df['Date'].dt.day_name()

# If sector info missing, assume “Residential” for demonstration
if 'Sector' not in df.columns and 'sector' not in df.columns:
    df['Sector'] = np.random.choice(['Residential', 'Commercial', 'Industrial'], len(df))
```

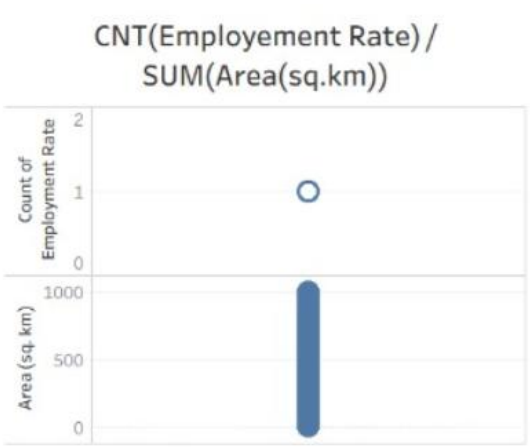
```python
# Rename likely consumption columns (auto-detect)
for col in df.columns:
    if 'electric' in col.lower():
        elec_col = col
    if 'water' in col.lower():
        water_col = col

# Fallbacks if not detected
elec_col = locals().get('elec_col', df.select_dtypes(include=[np.number]).columns[0])
water_col = locals().get('water_col', df.select_dtypes(include=[np.number]).columns[1])

print(f"\n⚡ Electricity Column: {elec_col}")
print(f"💧 Water Column: {water_col}")

# ------------------------------------------------------------
# 🧩 Step 6: Univariate Analysis

plt.figure(figsize=(7,4))
sns.histplot(df[elec_col], kde=True)
plt.title("Distribution of Electricity Consumption (kWh)")
plt.show()

plt.figure(figsize=(7,4))
sns.histplot(df[water_col], kde=True, color='teal')
plt.title("Distribution of Water Usage (Liters)")
plt.show()

# ------------------------------------------------------------
# 🧩 Step 7: Bivariate Analysis

plt.figure(figsize=(8,5))
sns.boxplot(x='Sector', y=elec_col, data=df)
plt.title("Electricity Consumption by Sector")
plt.show()

plt.figure(figsize=(8,5))
sns.boxplot(x='Sector', y=water_col, data=df)
plt.title("Water Usage by Sector")
plt.show()

# ------------------------------------------------------------
# 🧩 Step 8: Time-based Trend
plt.figure(figsize=(10,5))
sns.lineplot(x='Date', y=elec_col, hue='Sector', data=df)
plt.title("Electricity Consumption Over Time")
plt.show()

# ------------------------------------------------------------
# 🧩 Step 9: Heatmap – Peak Usage Hours
if df['Hour'].notna().any():
    pivot_data = df.pivot_table(values=elec_col, index='Day', columns='Hour', aggfunc='mean')
    plt.figure(figsize=(10,5))
    sns.heatmap(pivot_data, cmap='YlOrBr')
    plt.title("Average Electricity Usage by Day & Hour")
    plt.show()

# ------------------------------------------------------------
# 🧩 Step 10: Save cleaned dataset
df.to_csv("cleaned_smart_city_dataset.csv", index=False)
print("\n💾 Cleaned dataset saved as 'cleaned_smart_city_dataset.csv'")

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

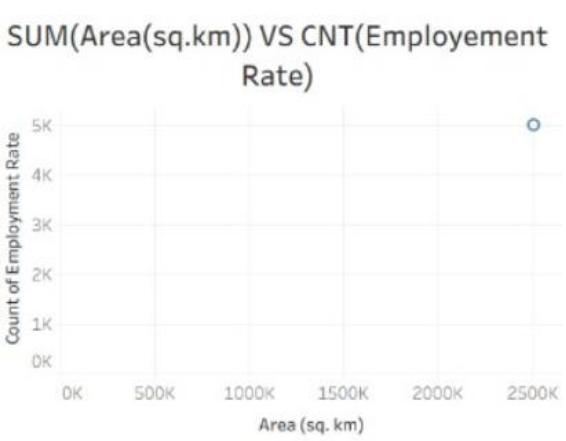

Electricity Consumption by Sector
1e7
Population
Industrial
Residential
Commercial
Sector
Water Usage by Sector
Area (sq. km)
Industrial
Residential
Commercial

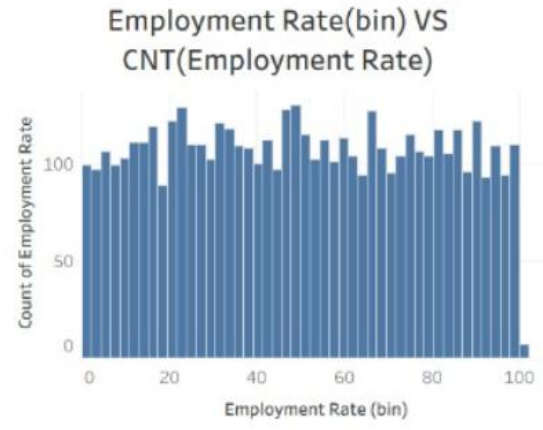

Distribution of Electricity Consumption (kWh)
Count
Population
1e7
Distribution of Water Usage (Liters)
Count
Area (sq. km)

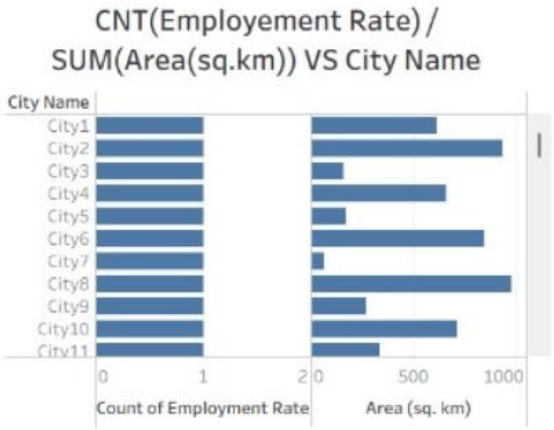

Electricity Consumption Over Time
1e7
Population
Date
Sector
Industrial
Residential
Commercial
Average Electricity Usage by Day & Hour
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
Day
Hour
1e6


Quick summary
Table
25276479139
Sum of Population
2,480,903.51
Sum of Internet Speed (...
2,505,485.91
Sum of Area (sq. km)
2,736,715.70
Sum of Energy Consump...
Sum of Employment Rate by Cost of Living Index
Sum of Employment Rate
Cost of Living Index
At 9997692, City2992 had the highest Sum of Population and was 92,197.75% higher than City427, which had the lowest Sum of Population at 10832.
City2992 accounted for 0.04% of Sum of Population.
Across all 5,000 City Name, Sum of Population ranged from 10832 to 9997692.
Count of City Name by Energy Consumption
Count of City Name
Energy Consumption
Sum of Healthcare Index by Population
Population
Sum of Healthcare Index
Sum of Safety Index by Public Transport Usage
Public Transport Usage
Sum of Safety Index


CNT(Employement Rate) / SUM(Area(sq.km))
Count of Employment Rate
Area (sq. km)


SUM(Area(sq.km)) VS CNT(Employement Rate)
Count of Employment Rate
Area (sq. km)


Employment Rate(bin) VS CNT(Employment Rate)
Count of Employment Rate
Employment Rate (bin)


CNT(Employement Rate) / SUM(Area(sq.km)) VS City Name
City Name
City1
City2
City3
City4
City5
City6
City7
City8
City9
City10
City11
Count of Employment Rate
Area (sq. km)