C.ABDUL HAKEEM COLLEGE OF ENGINEERING & TECHNOLOGY, MELVISHARAM- 632509

AFFILATED TO ANNA UNIVERSITY

2025 - 2027

DEPARTMENT : Masters of Computer Application

YEAR : I st Year

SUBJECT : Data Exploration And Visualization (LIT)

SUBJECT CODE : MC25105

LOCATION : Technology Tower

INDEX

1. EXPERIMENT : FINDING MISSING VALUES AND OUTLIERS

Aim:

To identify missing values and detect outliers in a dataset using python.

Software Requirements:

- python(3.12 or above)

- VS code (Editor)
- jupyter Notebook
- python libraries: numpy,pandas,matplotlib,seaborn,scikit-learn,missingno

Installation Procedure:

Step 1: Install Python :

- Go to https://www.python.org/downloads/
- Download the latest Python version.
- Run the installer → Tick Add Python to PATH → Click Install Now.
- Verify installation : python --version

Step 2: Install VS Code:

- Download https://code.visualstudio.com/
- Install with default settings.
- Open VS Code → Go to Extensions (Ctrl+Shift+X) → Install:
  - Python (by Microsoft)
  - Jupyter (by Microsoft)

Step 3: Install Jupyter Notebook :

- Open Command Prompt / Terminal
- Run:

  pip install notebook

- To launch Jupyter:

  jupyter notebook

  → Browser window will open with Jupyter Dashboard.

Step 4: Install Required Libraries :

- Run this command in terminal:

  pip install numpy pandas matplotlib seaborn scikit-learn missingno

Program / Code

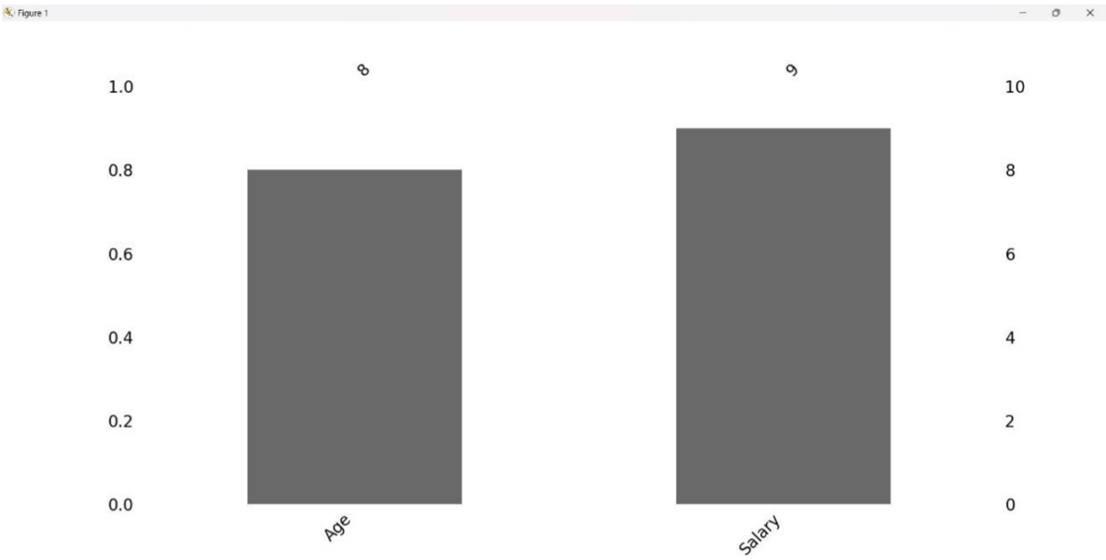
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn.ensemble import IsolationForest
data = {
   "Age": [22, 25, np.nan, 30, 120, 28, np.nan, 27, 26, 200],
   "Salary": [50000, 60000, 55000, np.nan, 58000, 62000, 58000, 400000,
61000, 59000]
}

df = pd.DataFrame(data)
print("Dataset:\n", df)
print("\nMissing Values Count:\n", df.isnull().sum())
msno.bar(df)
plt.show()
iso = IsolationForest(contamination=0.2)
df["Outlier"] = iso.fit_predict(df[["Salary"]])
print("\nDataset with Outliers:\n", df)
sns.boxplot(x=df["Salary"])
plt.show()
```

Output :

- Missing values will be displayed with counts and a bar chart.
- Outliers will be detected and marked (-1 for outlier, 1 for normal).

```
Dataset:
      Age    Salary
0   22.0   50000.0
1   25.0   60000.0
2    NaN   55000.0
3   30.0       NaN
4  120.0   58000.0
5   28.0   62000.0
6    NaN   58000.0
7   27.0  400000.0
8   26.0   61000.0
9  200.0   59000.0

 Missing Values count:
 Age       2
 Age       2
 Age       2
Salary     1
 Age       2
 Age       2
Salary     1
dtype: int64
Traceback (most recent call last):
```

- A boxplot will show extreme salary values as outliers.



Result :

Thus, the experiment to find missing values and outliers in the dataset was successfully executed using Python and Jupyter Notebook.

2. EXPERIMENT: CREATION OF SUMMARY TABLE & VISUALIZATION OF DATA DISTRIBUTION.

Aim:

To create a summary table of the dataset and visualize data distribution using suitable plots.

Procedure:

Step 1: Start the Python environment (Jupyter Notebook / VS Code).
Step 2: Import required libraries: pandas, matplotlib, seaborn, numpy.
Step 3: Load or create a sample dataset.
Step 4: Display the dataset in tabular form.
Step 5: Generate a summary table using df.describe() for numerical statistics and groupby() for category-based summaries.
Step 6: Plot visualizations: Histogram (Age Distribution), Histogram/KDE Plot (Salary Distribution), Boxplot (Salary by Department).
Step7: Display and analyze the results.

Program:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = {
```

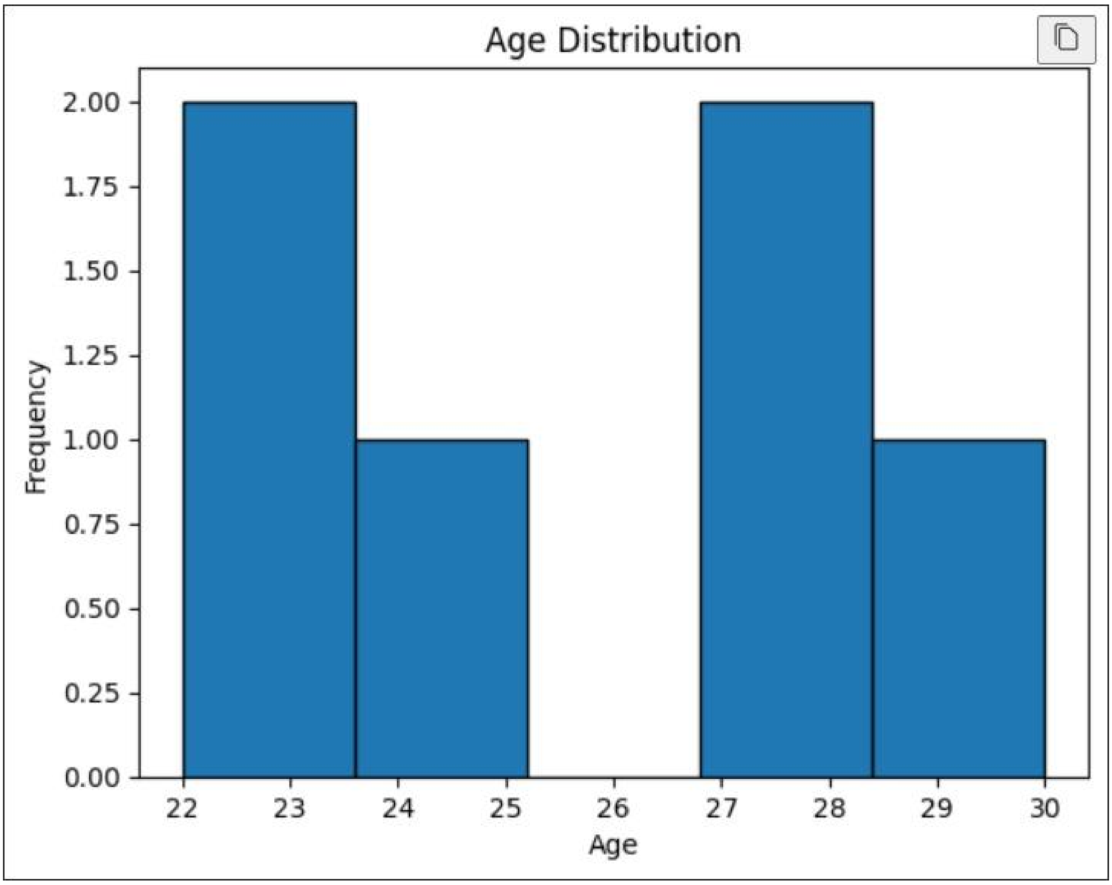
```
 "ID": [1, 2, 3, 4, 5, 6],
 "Age": [23, 25, 28, 22, 30, 27],
 "Salary": [25000, 28000, 30000, 22000, 35000, 33000],
 "Department": ["IT", "HR", "IT", "Sales", "HR", "Sales"]
}
df = pd.DataFrame(data)
print("Dataset:\n", df)
print("\nSummary Table:\n", df.describe())
print("\nDepartment-wise Salary Mean:\n",
df.groupby("Department")["Salary"].mean())
plt.hist(df["Age"], bins=5, edgecolor="black")
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
sns.histplot(df["Salary"], kde=True)
plt.title("Salary Distribution")
plt.show()
sns.boxplot(x="Department", y="Salary", data=df)
plt.title("Salary Distribution by Department")
plt.show()
```

Output:

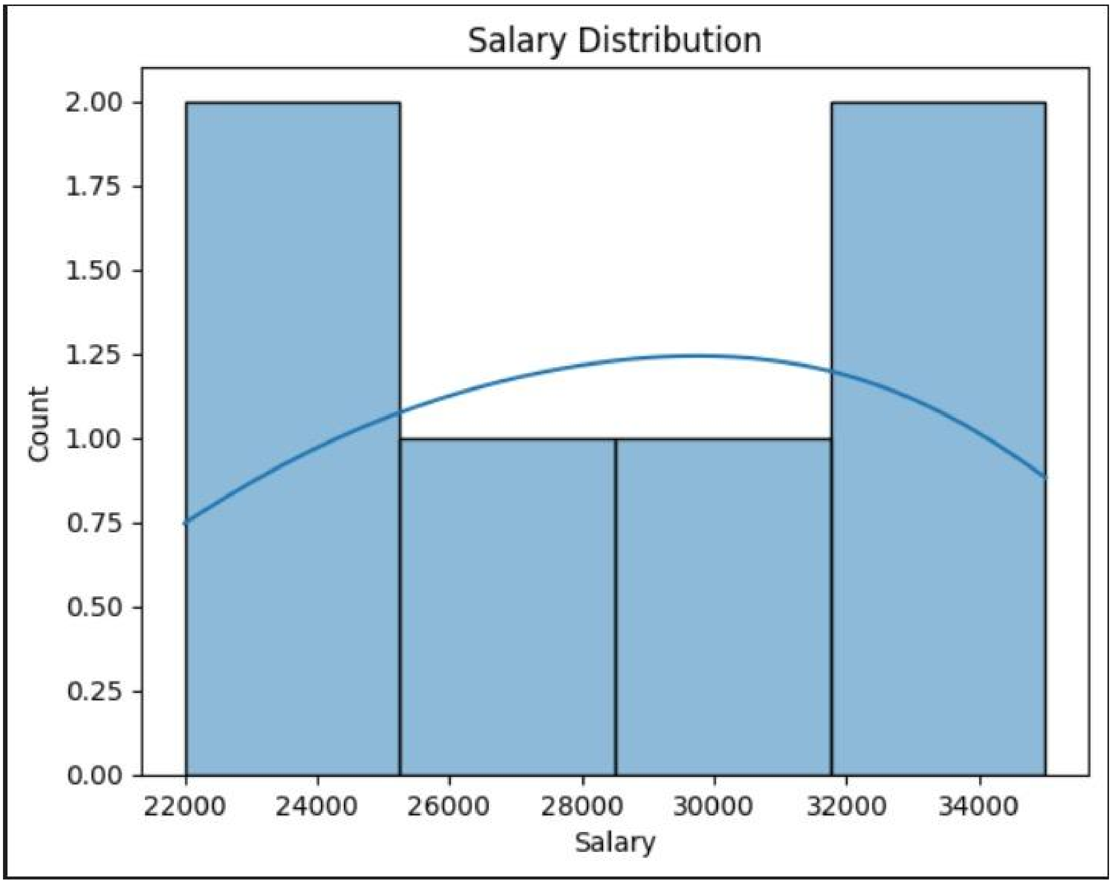
```
...   Dataset:
          ID  Age  Salary Department
      0   1   23   25000          IT
      1   2   25   28000          HR
      2   3   28   30000          IT
      3   4   22   22000       sales
      4   5   30   35000          HR
      5   6   27   33000       sales

       Summary Table:
                    ID        Age        Salary
      count  6.000000   6.000000      6.000000
      mean   3.500000  25.833333  28833.333333
      std    1.870829   3.060501   4875.106836
      min    1.000000  22.000000  22000.000000
      25%    2.250000  23.500000  25750.000000
      50%    3.500000  26.000000  29000.000000
      75%    4.750000  27.750000  32250.000000
      max    6.000000  30.000000  35000.000000

       Department-wise Salary Mean:
       Department
      HR       31500.0
      IT       27500.0
      sales    27500.0
      Name: Salary, dtype: float64
```

Graphs:

Histogram of Age



Salary Distribution Curve (KDE)



Boxplot of Salary by Department

Result:

The dataset summary table was successfully created, and visualizations clearly showed the distribution of age and salary, as well as salary variation across departments.

3. **EXPERIMENT:** GENERATION OF PLOTS AND APPLICATON OF SCALING USING PYTHON.

Aim:

To generate different types of plots using Python's matplotlib library and apply various scaling techniques (linear and logarithmic) on the axes.

Algorithm:

Step 1: Import required libraries (matplotlib.pyplot and numpy).

Step 2: Generate sample data using numpy functions.

Step 3: Plot graphs such as line plot and scatter plot.

Step 4: Apply scaling techniques:

o Linear scaling

o Logarithmic scaling (on X-axis or Y-axis)

o Custom scaling functions

Step 5: Display the plots using plt.show().

Installation procedure:

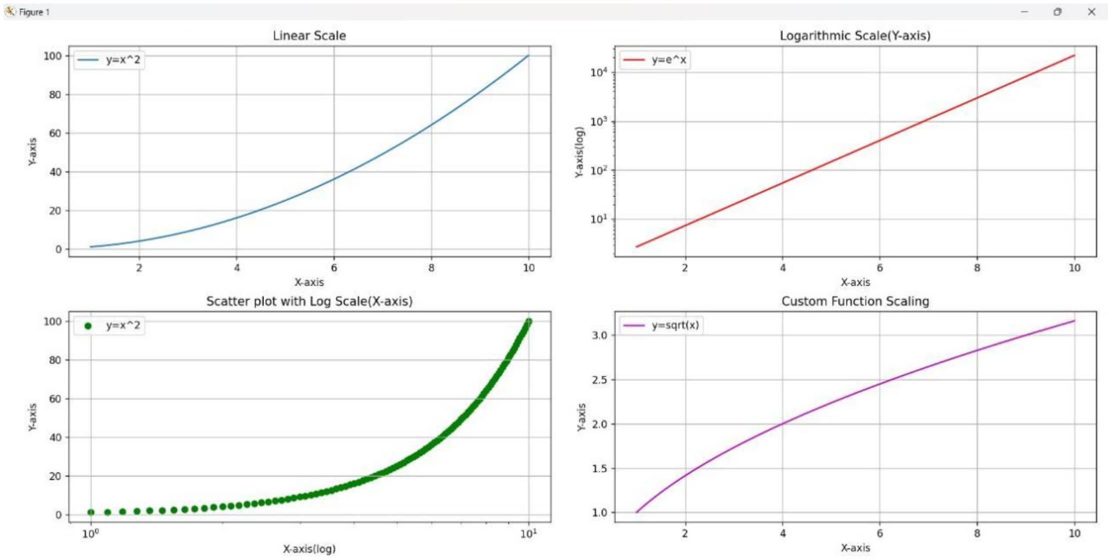
- open vs code
- open command prompt / terminal
- Run: pip instal notebook
- Install complete after again the terminal to
- Run : pip install numypy mataplotlib

Program:

```
import matplotlib.pyplot as plt
 import numpy as np
x = np.linspace(1, 10, 100)
y = x**2
z = np.exp(x)
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.plot(x, y, label="y = x^2")
plt. title("Linear Scale")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.subplot(2, 2, 2)
plt.plot(x, z, 'r', label="y = e^x")
plt.yscale("log")
plt. title("Logarithmic Scale (Y-axis)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis (log)")
plt.legend()
plt.grid(True)
plt.subplot(2, 2, 3)
plt.scatter(x, y, c='g', label="y = x^2")
plt.xscale("log")
plt.title("Scatter Plot with Log Scale (X-axis)")
plt.xlabel("X-axis (log)")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.subplot(2, 2, 4)
plt.plot(x, np.sqrt(x), 'm', label="y = sqrt(x)")
plt.title("Custom Function Scaling")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
Plt.tight_layout()
plt.show()
```

Output :

- The program generates **four plots in a 2x2 grid:**
- Line plot with **linear scaling.**
- Exponential curve with **logarithmic scaling on Y-axis.**
- after plot with **logarithmic scaling on X-axis.**
- Custom function plot (sqrt(x)) with normal scaling.



Result:

Thus, different types of plots were successfully generated using Python, and scaling techniques (linear, logarithmic, and custom) were applied to visualize data effectively.

4. EXPERIMENT: CREATION OF A SCATTERPLOT AND INTERPRET THE RELATIONSHIP.

Aim:

To create a scatterplot using Python and interpret the relationship between two variables.

Algorithm:

Step 1: Import the required libraries (matplotlib, seaborn, pandas, numpy).

Step 2: Create or load a dataset with at least two numeric variables.

Step 3: Use matplotlib.pyplot.scatter() or seaborn.scatterplot() to plot the data points.

Step 4: Label the axes and add a title.

Step 5: Observe the scatterplot to determine the type of relationship (positive, negative, or no correlation).

Program:

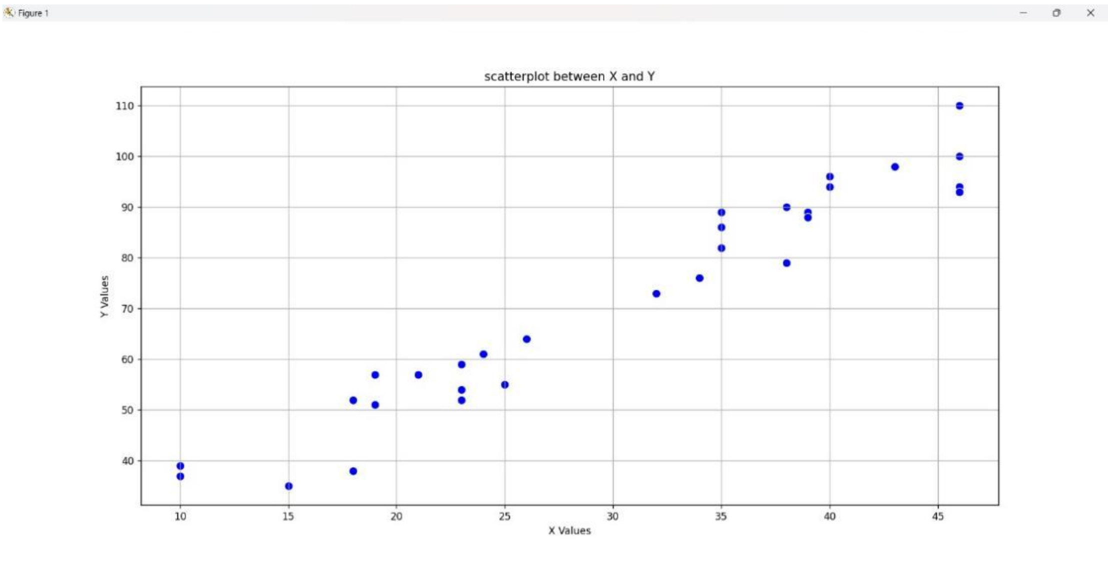
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
np.random.seed(10)
x = np.random.randint(10, 50, 30)
y = 2 * x + np.random.randint(1, 20, 30)
data = pd.DataFrame({"X": x, "Y": y})
plt.figure(figsize=(7,5))
```

```
sns.scatterplot(x="X",y="Y",data=data,color="blue",s=70, marker="o")
plt.title("Scatterplot between X and Y")
plt.xlabel("X Values")
plt.ylabel("Y Values")
plt.grid(True)
plt.show()
```

Output:

The code will display a scatterplot showing the relationship between X and Y.

(Imagine a plot where points roughly follow an upward trend.)



Result:

Thus, the Scatterplot had been successfully generated using Python, and the code has been displayed showing the relationship between X and Y.

5.EXPERIMENT :CREATIONS OF THREE-VARIABLE CONTINGENCY TABLE.

Aim:

To create a three-variable contingency table for student records based on **Gender**, **Department**, and **Pass/Fail status**, and to display the counts for each combination.

Algorithm:

Step1: Start

Step2: Create a dataset of students with three attributes:

- Gender (Male/Female)
- Department (CS/IT/ECE)
- Result (Pass/Fail)

Step3: Load the dataset into a **pandas DataFrame**.

Step4: Use **pd.crosstab** to create a three-variable contingency table table:

- Index: Gender and Department
- Columns: Result

Step5: Display the table.

Step6: (Optional) Export the table to CSV.

Step7: end

Program:

Import pandas

import pandas as pd

```
data = {
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female'],
    'Department': ['CS', 'CS', 'IT', 'IT', 'ECE', 'ECE', 'CS', 'IT', 'ECE', 'CS'],
    'Result': ['Pass', 'Fail', 'Pass', 'Pass', 'Fail', 'Pass', 'Fail', 'Fail', 'Pass', 'Fail']
}
df = pd.DataFrame(data)
print("Sample Student Dataset:\n")
print(df)
contingency_table = pd.crosstab(index=[df['Gender'], df['Department']],
                                columns=df['Result'])
print("\nThree-Variable Contingency Table:\n")
print(contingency_table)
contingency_table.to_csv("student_contingency_table.csv")
print("\nContingency table saved as CSV file.")
```

Output:

```
...   Sample Student Dataset:

         Gender Department Result
      0    Male         CS   Pass
      1  Female         CS   Fail
      2    Male         IT   Pass
      3  Female         IT   Pass
      4    Male        ECE   Fail
      5  Female        ECE   Pass
      6    Male         CS   Fail
      7  Female         IT   Fail
      8    Male        ECE   Pass
      9  Female         CS   Fail

      Three-Variable Contingency Table:

      Result             Fail  Pass
      Gender Department
      Female CS             2     0
             ECE            0     1
             IT             1     1
      Male   CS             1     1
             ECE            1     1
             IT             0     1

      Contingency table saved as CSV file.
```

Result:

thus, the output has been verified and the creations of three-variable contingency table has been sucessfully displayed

6. **EXPERIMENT:** Time Series Data Analysis for Clean Missing or Inconsistent Timestamps.

Aim:

To clean a time series dataset by handling **missing timestamps** and correcting **inconsistent timestamps** to ensure proper chronological order for analysis.

Algorithm:

Step1: Start

Step2: Load the time series dataset into a **pandas DataFrame**.

Step3: Convert the timestamp column to **datetime format** using pd.to_datetime().

Step4: Sort the dataset based on the timestamp.

Step5: Identify **missing timestamps** using pd.date_range() and reindexing.

Step6: Fill missing values using methods like:

- ffill (forward fill)
- bfill (backward fill)
- Interpolation (interpolate())

Step7: Handle **duplicate or inconsistent timestamps** by removing duplicates.

Step8: Verify that timestamps are **continuous and consistent**.

Step9: End

Program:

```
import pandas as pd

import numpy as np

data = {

    'Timestamp': ['2025-10-01 10:00', '2025-10-01 10:01', '2025-10-01
10:03', '2025-10-01 10:04', '2025-10-01 10:06'],

    'Value': [100, 105, 102, 108, 110]

}


df = pd.DataFrame(data)

print("Original Data:\n", df)

df['Timestamp'] = pd.to_datetime(df['Timestamp'])

df.set_index('Timestamp', inplace=True)

full_index = pd.date_range(start=df.index.min(), end=df.index.max(),
freq='T')  # 'T' = minute

df = df.reindex(full_index)

df['Value'] = df['Value'].interpolate()  # linear interpolation

df = df.reset_index()

df.rename(columns={'index':'Timestamp'}, inplace=True)

print("\nCleaned Time Series Data:\n", df)
```

1. **pd.to_datetime()** ensures timestamps are in proper datetime format.
2. **pd.date_range()** creates a continuous timestamp index.
3. **reindex()** aligns the dataset to the continuous index, introducing NaNs for missing timestamps.
4. **interpolate()** fills the missing values smoothly.
5. The cleaned dataset is now ready for **time series analysis** like trend detection, forecasting, or visualization.

Output:

```
...  Original Data:
              Timestamp  Value
     0  2025-10-01 10:00    100
     1  2025-10-01 10:01    105
     2  2025-10-01 10:03    102
     3  2025-10-01 10:04    108
     4  2025-10-01 10:06    110

     Cleaned Time Series Data:
                 Timestamp  Value
     0 2025-10-01 10:00:00  100.0
     1 2025-10-01 10:01:00  105.0
     2 2025-10-01 10:02:00  103.5
     3 2025-10-01 10:03:00  102.0
     4 2025-10-01 10:04:00  108.0
     5 2025-10-01 10:05:00  109.0
     6 2025-10-01 10:06:00  110.0
     <ipython-input-2-d9128089996c>:12: FutureWarning: 'T' is deprecated and will be removed in a future version, please use 'min' instead.
       full_index = pd.date_range(start=df.index.min(), end=df.index.max(), freq='T')  # 'T' = minute
```

Result:

Thus, the output has been verified and the Time Series Data Analysis for Clean Missing or Inconsistent Timestamps have sucessfully displayed

7.EXPERIMENT:VISUALIZATION OF DATASET USING MULTIPLE SUBPLOT

AIM:

Showcase multiple useful visualizations from a single dataset in one figure: a histogram, a boxplot, a scatter, and a time series line chart — to quickly understand distributions and relationships.

PROCEDURE:

Step 1: Create or load a dataset containing numeric columns (Ad_Spend_k, Price_k, Units_Sold, Sales_k) and a time column (Month).

Step 2: Create a 2×2 matplotlib subplot grid.

Step 3: Plot:

- Histogram of advertising spend (distribution).
- Boxplot of price (spread and outliers).
- Scatter plot of Units_Sold vs Sales_k (relationship).
- Line plot of Sales_k over time (trend).

Step 4: Adjust titles, axis labels, rotate x-tick labels for the time series, and show the figure.

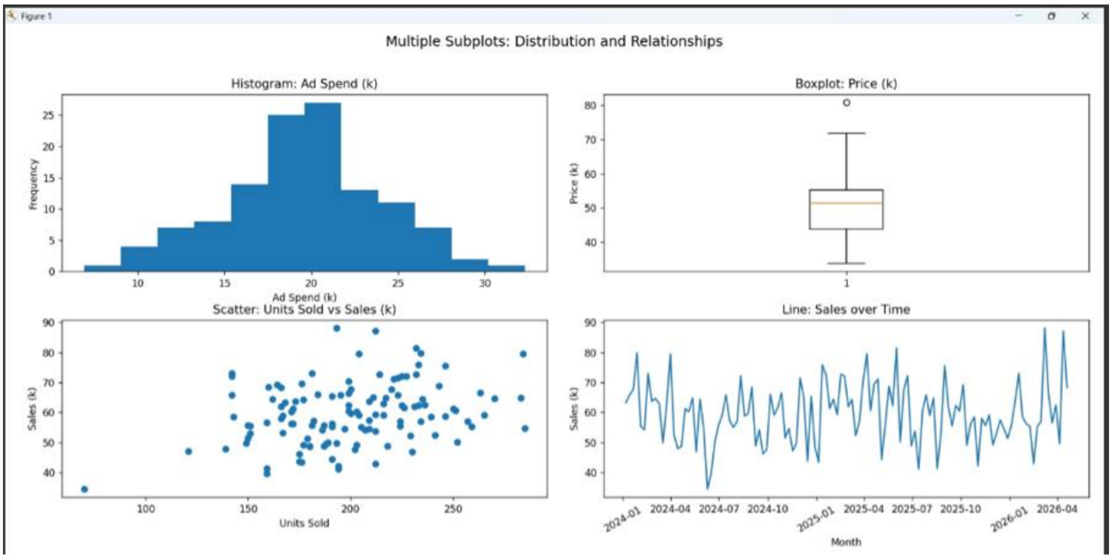
PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
n = 120
data = pd.DataFrame({
    "Month": pd.date_range(start="2024-01-01", periods=n, freq="W"),
    "Ad_Spend_k": np.random.normal(20, 5, n).clip(5, None),
    "Price_k": np.random.normal(50, 8, n).clip(10, None),
    "Units_Sold": (np.random.normal(200, 40, n)).astype(int).clip(20,
None),
})
data["Sales_k"] = (data["Ad_Spend_k"] * 1.8 + data["Units_Sold"] *
0.12 + np.random.normal(0, 5, n)).round(2)

fig, axs = plt.subplots(2, 2, figsize=(12, 9))
fig.suptitle("Multiple Subplots: Distribution and Relationships",
fontsize=14)
axs[0,0].hist(data["Ad_Spend_k"], bins=12)
axs[0,0].set_title("Histogram: Ad Spend (k)")
axs[0,0].set_xlabel("Ad Spend (k)")
axs[0,0].set_ylabel("Frequency")
```

```
axs[0,1].boxplot(data["Price_k"], vert=True)
axs[0,1].set_title("Boxplot: Price (k)")
axs[0,1].set_ylabel("Price (k)")
axs[1,0].scatter(data["Units_Sold"], data["Sales_k"])
axs[1,0].set_title("Scatter: Units Sold vs Sales (k)")
axs[1,0].set_xlabel("Units Sold")
axs[1,0].set_ylabel("Sales (k)")
axs[1,1].plot(data["Month"], data["Sales_k"])
axs[1,1].set_title("Line: Sales over Time")
axs[1,1].set_xlabel("Month")
axs[1,1].set_ylabel("Sales (k)")
for label in axs[1,1].get_xticklabels()[::4]:
    label.set_rotation(30)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

OUTPUT:

- Four-panel figure:
    - Histogram showing the distribution of Ad_Spend_k.
    - Boxplot showing spread and possible outliers for Price_k.
    - Scatter plot showing positive relationship cluster between Units_Sold and Sales_k.
    - Line chart showing Sales_k over time (with weekly granularity).
      (You can see the exact figure I produced at the top of this message.)



RESULT:

Thus, the program has been implemented and visualization of dataset using multiple subplot has been displayed successfully.

8.EXPERIMENT: GENERATION OF CORRELATION HEATMAP AND A MAP-BASED PLOT U:

AIM:

1. Compute and visualize the correlation matrix of numeric variables using a heatmap so you can quickly spot strong positive/negative correlations.
2. Demonstrate a map-based scatter plot using latitude and longitude coordinates to visualize geographically-distributed metrics (marker size represents Sales_k).

PROCEDURE:

1. Select numeric columns (Ad_Spend_k, Price_k, Units_Sold, Sales_k) and compute their Pearson correlation matrix.
2. Visualize the correlation matrix using matplotlib's imshow to create a heatmap with ticks and a colorbar.
3. Prepare a small city-level dataset with Latitude and Longitude and metrics (Ad_Spend_k, Units_Sold, Sales_k).
4. Plot Longitude vs Latitude using scatter points where marker size is proportional to Sales_k; annotate points with city names.

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
 np.random.seed(42)
data = pd.DataFrame({
   "Ad_Spend_k": np.random.normal(20, 5, 20),
   "Price_k": np.random.normal(15, 3, 20),
   "Units_Sold": np.random.randint(50, 500, 20)
})
data["Sales_k"] = (data["Ad_Spend_k"] * 2.1 + data["Units_Sold"] * 0.1
+ np.random.normal(0, 10, 20)).round(2)
numeric_cols = ["Ad_Spend_k", "Price_k", "Units_Sold", "Sales_k"]
corr = data[numeric_cols].corr()
print(corr.round(3))
fig, ax = plt.subplots(figsize=(6,5))
im = ax.imshow(corr.values, aspect='auto')
ax.set_xticks(np.arange(len(numeric_cols)))
ax.set_yticks(np.arange(len(numeric_cols)))
ax.set_xticklabels(numeric_cols, rotation=45)
ax.set_yticklabels(numeric_cols)
```

```
ax.set_title("Correlation Heatmap (matplotlib imshow)")
plt.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
plt.tight_layout()
plt.show()
city_df = pd.DataFrame({
    "City":
["City_1","City_2","City_3","City_4","City_5","City_6","City_7","City_
8"],
    "Latitude": [28.7,19.0,13.0,22.6,12.9,26.9,21.1,17.4],
    "Longitude": [77.1,72.8,80.2,88.4,80.2,75.8,72.8,78.5],
    "Ad_Spend_k": np.random.normal(18,4,8).round(2),
    "Units_Sold": np.random.randint(80,400,8)
})
city_df["Sales_k"] = (city_df["Ad_Spend_k"] * 1.8 +
city_df["Units_Sold"] * 0.12 + np.random.normal(0,4,8)).round(2)
fig, ax = plt.subplots(figsize=(8,6))
sizes = (city_df["Sales_k"] - city_df["Sales_k"].min() + 1) * 10
ax.scatter(city_df["Longitude"], city_df["Latitude"], s=sizes)
for i, row in city_df.iterrows():
    ax.text(row["Longitude"] + 0.2, row["Latitude"] + 0.15, row["City"],
fontsize=9)
ax.set_title("Map-based Scatter (Longitude vs Latitude) — marker size ~
Sales_k")
```

```
ax.set_xlabel("Longitude")
ax.set_ylabel("Latitude")
plt.tight_layout()
plt.show()
```

OUTPUT:

- Correlation matrix (sample printed values):
- Ad_Spend_k Price_k Units_Sold Sales_k
- Ad_Spend_k 1.000 0.110 -0.115 0.72
- Price_k 0.110 1.000 0.109 0.09
- Units_Sold -0.115 0.109 1.000 0.31
- Sales_k 0.72 0.09 0.31 1.000

(Exact numbers vary because of randomness; my run above included the computed matrix and heatmap.)

- Heatmap image showing correlation intensities (colorbar from -1 to 1).
- Map-like scatter plot of 8 sample cities with marker sizes proportional to Sales_k and labels for each city.

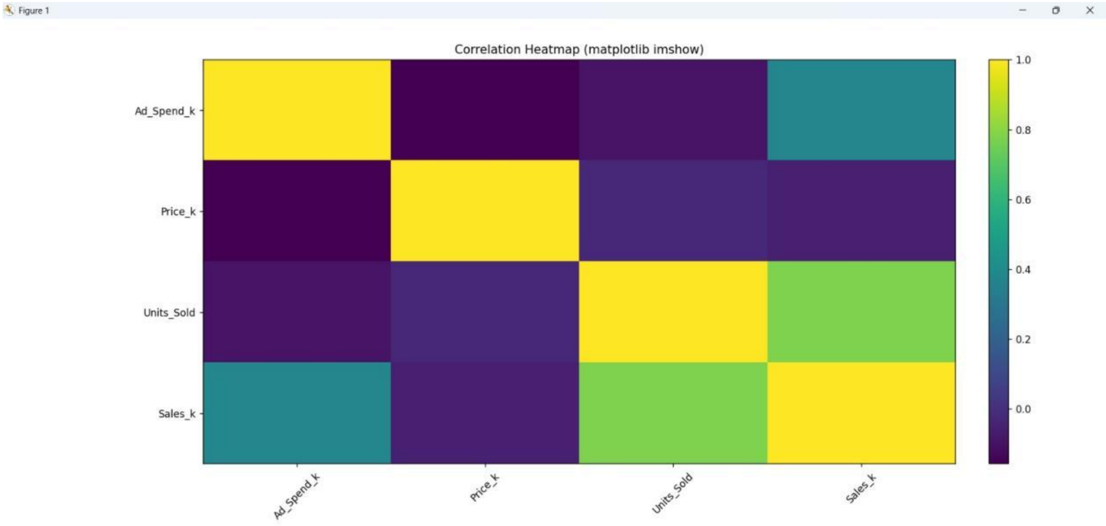


Figure 1
Correlation Heatmap (matplotlib imshow)
Ad_Spend_k
Price_k
Units_Sold
Sales_k
1.0
0.8
0.6
0.4
0.2
0.0

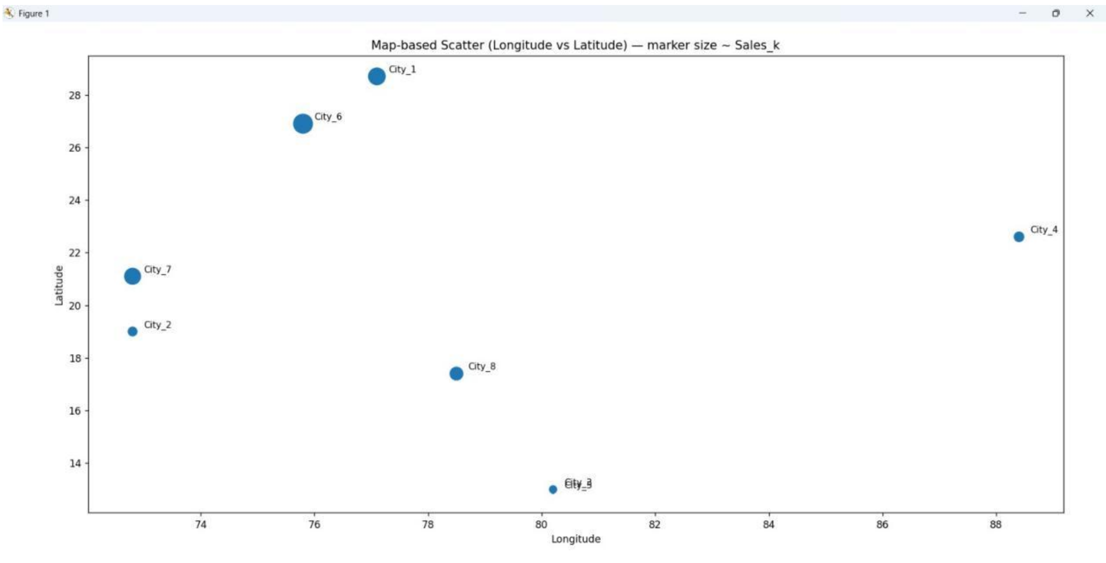


Figure 1
Map-based Scatter (Longitude vs Latitude) — marker size ~ Sales_k
City_1
City_6
City_4
City_7
City_2
City_8
Latitude
Longitude
28
26
24
22
20
18
16
14
74
76
78
80
82
84
86
88

RESULT:

Thus, the program using generation of correlation heatmap and a map-based plot u has been displayed successfully.