Experiment No: 1

Aim: a)Write a program convert Text delimited CSV to Hours format

b) Convert a Time Column into Hours Format (HH:MM:SS or Decimal)

Theory:

(a) Convert Text-Delimited CSV to Hours Format

Text-delimited CSV files store data using a specific delimiter (e.g., comma ,, tab \t, or pipe |). To convert this data into an **hours format**, we follow these steps:

1. **Read the CSV file** using pandas.read_csv(), specifying the delimiter.
2. **Extract the time-related column** containing durations (e.g., hh:mm:ss).
3. **Convert the duration into total hours** using pd.to_timedelta() and extract hours.
4. **Save or process the modified data** in hours format.

(b) Convert a Time Column into Hours Format (HH:MM:SS or Decimal)

A time column (e.g., hh:mm:ss) can be converted into hours format in two ways:

1. **HH:MM:SS Format** → Keep the format unchanged but ensure it's properly recognized as a time object using pd.to_datetime().
2. **Decimal Hours Format** → Convert hh:mm:ss into a decimal value using: Hours=Total Seconds/3600
3. Total Seconds This helps in numerical analysis, like computing work hours or aggregating time durations.

Both conversions are useful in time series analysis, payroll calculations, and scheduling systems.

Conclusion: We have successfully converted Text delimited CSV to Hours format

b) Convert a Time Column into Hours Format (HH:MM:SS or Decimal)

	A	B	C	D
	id	name	Duration	
	1	Ram	2:30:00	
	2	Sham	1:15:30	
	3	Krishna	0:45:15	

```
[66] import pandas as pd
```

```
[67] df1 = pd.read_csv("/content/pract1.csv")
```

```
[68] df1['hours'] = pd.to_timedelta(df['Duration']).dt.total_seconds() / 3600
```

```
print(df1)
```

```
   id     name Duration  Unnamed: 3  Unnamed: 4     hours
0   1      Ram  2:30:00         NaN         NaN  2.500000
1   2     Sham  1:15:30         NaN         NaN  1.258333
2   3  Krishna  0:45:15         NaN         NaN  0.754167
```

b) Convert a Time Column into Hours Format (HH:MM:SS or Decimal)

	A	B	C
	id	name	Timestamp
	1	Ram	14:45:30
	2	Sham	09:15:00
	3	Krishna	23:05:45

```
[51] import pandas as pd
```

```
[52] df = pd.read_csv("/content/pract3.csv")
```

```
[56] df["hours"] = pd.to_timedelta(df["Timestamp"]).dt.total_seconds() / 3600
```

```
[60] print(df)
```

```
   id     name Timestamp      hours
0   1      Ram  14:45:30  14.758333
1   2     Sham  09:15:00   9.250000
2   3  Krishna  23:05:45  23.095833
```

Experiment No.2

Aim-Data binning or Bucketing

Title :Write python code for Data binning or Bucketing

Theory :

Data binning (also called **bucketing**) is a technique to group numerical data into intervals (bins). This is useful in:

- **Reducing noise in the data**
- **Handling continuous variables**
- **Creating categorical features for machine learning**
- pd.cut() → Divides data into **fixed** intervals (equal width or custom-defined bins).
- pd.qcut() → Divides data into **equal-sized groups** (quantiles).

Example: Binning Age Data

Let’s assume we have a dataset with ages, and we want to **group (bin) them** into categories like:

- **Child (0-12)**
- **Teenager (13-19)**
- **Adult (20-59)**
- **Senior (60 and above)**

```python
import pandas as pd

# Sample dataset with ages
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Emma'],
        'Age': [5, 17, 34, 70, 25]}

df = pd.DataFrame(data)

# Define bin edges and labels
bins = [0, 12, 19, 59, 100]  # Bin edges
labels = ['Child', 'Teenager', 'Adult', 'Senior']  # Bin labels

# Apply binning
df['Age Group'] = pd.cut(df['Age'], bins=bins, labels=labels)

# Display the result
print(df)
```

```
      Name  Age Age Group
0    Alice    5     Child
1      Bob   17  Teenager
2  Charlie   34     Adult
3    David   70    Senior
4     Emma   25     Adult
```

Example: Binning Data into Equal-Width Bins

```python
df['Age Bin'] = pd.cut(df['Age'], bins=3)  # 3 equal-width bins
print(df)
```

```
      Name  Age Age Group           Age Bin
0    Alice    5     Child   (4.935, 26.667]
1      Bob   17  Teenager   (4.935, 26.667]
2  Charlie   34     Adult  (26.667, 48.333]
3    David   70    Senior    (48.333, 70.0]
4     Emma   25     Adult   (4.935, 26.667]
```

If you don't define custom bins, you can divide data into **equal-width bins** automatically.

Example: Binning Data into Equal-Frequency Bins

Instead of **equal-width bins**, we can use **equal-frequency bins** (quantiles).

```python
df['Quantile Bin'] = pd.qcut(df['Age'], q=3, labels=['Low', 'Medium', 'High'])  # 3 bins
print(df)
```

```
      Name  Age Age Group           Age Bin Quantile Bin
0    Alice    5     Child   (4.935, 26.667]          Low
1      Bob   17  Teenager   (4.935, 26.667]          Low
2  Charlie   34     Adult  (26.667, 48.333]         High
3    David   70    Senior    (48.333, 70.0]         High
4     Emma   25     Adult   (4.935, 26.667]       Medium
```

Conclusion: We have successfully implemented python code for Data binning or Bucketing

Experiment No:3

Aim: Write python code for averaging data.

Theory: Averaging data means calculating the **mean** of a set of numbers. The mean is the sum of all data points divided by the total number of data points. The general formula for calculating the average (arithmetic mean) is:

$$\mathbf{Mean} = \frac{\sum X_i}{N}$$

where:

- Xi represents individual data points.
- N is the total number of data points

Methods to Calculate the Average in Python-Python provides multiple ways to calculate the average:

- Using Basic Arithmetic
  - Manually sum up all elements and divide by the count.
- Using the sum() and len() Functions
  - The built-in sum() function computes the sum, and len() gives the count.
- Using the statistics.mean() Function
  - Python's statistics module provides a mean() function.
- Using NumPy for Large Datasets
  - The NumPy library has an optimized numpy.mean() function.

Conclusion: By using python code we have perform summarization operations on data.

https://colab.research.google.com/drive/1TenvJ6EgVRIbY0Eeung5drMXjTth2diE#scrollTo=b-_3yFG19_h0

Exp3_mscit.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

[1]
```python
import pandas as pd
import numpy as np
```

[2]
```python
#Python Code for Averaging Data
#Method 1: Using Basic Arithmetic
# Sample data
data = [10, 20, 30, 40, 50]

# Compute average
average = sum(data) / len(data)

print("Average:", average)
```

```
Average: 30.0
```

[3]
```python
#Method 2: Using statistics.mean()
import statistics

# Sample data
data = [10, 20, 30, 40, 50]

# Compute average using statistics module
average = statistics.mean(data)

print("Average:", average)
```

```
Average: 30
```

```python
#Method 3: Using NumPy (For Large Datasets)
import numpy as np

# Sample data
data = np.array([10, 20, 30, 40, 50])

# Compute average using NumPy
average = np.mean(data)

print("Average:", average)
```

```
Average: 30.0
```

[5]
```python
import numpy as np
import pandas as pd
```

[6]
```python
data=[10,20,45,35,87]
```

[7]
```python
total=np.sum(data)
print("Total=",total)
```

```
Total= 197
```

[8]
```python
mn=np.mean(data)
print("Mean of Data",mn)
```

```
Mean of Data 39.4
```

Exp3_mscit.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

[9]
```
mdn=np.median(data)
print("Median of Data",mdn)
```

```
Median of Data 35.0
```

[10]
```
std_dev=np.std(data)
print("Standard Deviation of Data",std_dev)
```

```
Standard Deviation of Data 26.672832620477337
```

[11]
```
mini=np.min(data)
print("Minimum of Data",mini)
```

```
Minimum of Data 10
```

[12]
```
max=np.max(data)
print("Maximum of Data",max)
```

```
Maximum of Data 87
```

[13]
```
q1=np.percentile(data,25)
print("First Quartile",q1)
```

```
First Quartile 20.0
```

[14]
```
import pandas as pd
```

[15]
```
data1=[20,30,50,70,100]
print("list elements=",data1)
```

```
list elements= [20, 30, 50, 70, 100]
```

[16]
```
df=pd.DataFrame(data1)
print(df.head)
```

```
<bound method NDFrame.head of      0
0   20
1   30
2   50
3   70
4  100>
```

Exp3_mscit.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

[17]
```
df.describe()
```

	0
count	5.000000
mean	54.000000
std	32.093613
min	20.000000
25%	30.000000
50%	50.000000
75%	70.000000
max	100.000000

```
mn=df.mean()
mdn=df.median()
std_dev=df.std()
mini=df.min()
max=df.max()
print("Mean=",mn)
print("Median=",mdn)
print("Standard Deviation=",std_dev)
print("Minimum=",mini)
print("Maximum=",max)
```

```
Mean= 0    54.0
dtype: float64
Median= 0    50.0
dtype: float64
Standard Deviation= 0    32.093613
dtype: float64
Minimum= 0    20
dtype: int64
Maximum= 0    100
dtype: int64
```

Experiment No:4

Aim: Write python program to build acyclic graph

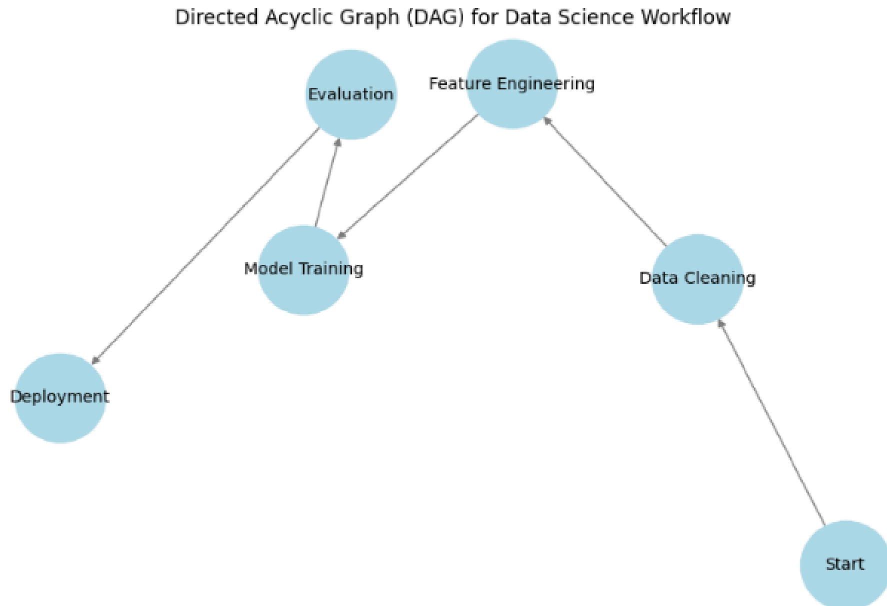
Theory:-An **Acyclic Graph (DAG - Directed Acyclic Graph)** is a directed graph with no cycles. It is widely used in **data science workflows**, **task scheduling**, and **dependency resolution** (e.g., Apache Airflow).

- 🎬 **Created a directed graph** using networkx.DiGraph().
- 🎬 **Added edges** representing a typical **data science pipeline**.
- 🎬 **Checked if the graph is acyclic** using nx.is_directed_acyclic_graph().
- 🎬 **Visualized the DAG** using matplotlib.

Conclusion:

```
import networkx as nx
import matplotlib.pyplot as plt

# Create a Directed Graph (DAG)
DAG = nx.DiGraph()

# Add edges (No cycles allowed)
edges = [
    ('Start', 'Data Cleaning'),
    ('Data Cleaning', 'Feature Engineering'),
    ('Feature Engineering', 'Model Training'),
    ('Model Training', 'Evaluation'),
    ('Evaluation', 'Deployment')
]

DAG.add_edges_from(edges)

# Check if the graph is acyclic
if nx.is_directed_acyclic_graph(DAG):
    print("The graph is a valid DAG.")
else:
    print("The graph contains cycles.")

# Draw the DAG
plt.figure(figsize=(8, 5))
pos = nx.spring_layout(DAG)  # Layout for better visualization
nx.draw(DAG, pos, with_labels=True, node_size=3000, node_color="lightblue", edge_color="gray", font_size=10
plt.title("Directed Acyclic Graph (DAG) for Data Science Workflow")
plt.show()
```

```
The graph is a valid DAG.
```



Experiment No:5

Aim: Write a python program using data science via clustering to determine new warehouse using given data

Theory:

Clustering is a **data science technique** used to group data points based on similarity. In this case, we will use **K-Means clustering** to determine the best location for a **new warehouse** based on existing customer locations.

Steps to Solve the Problem

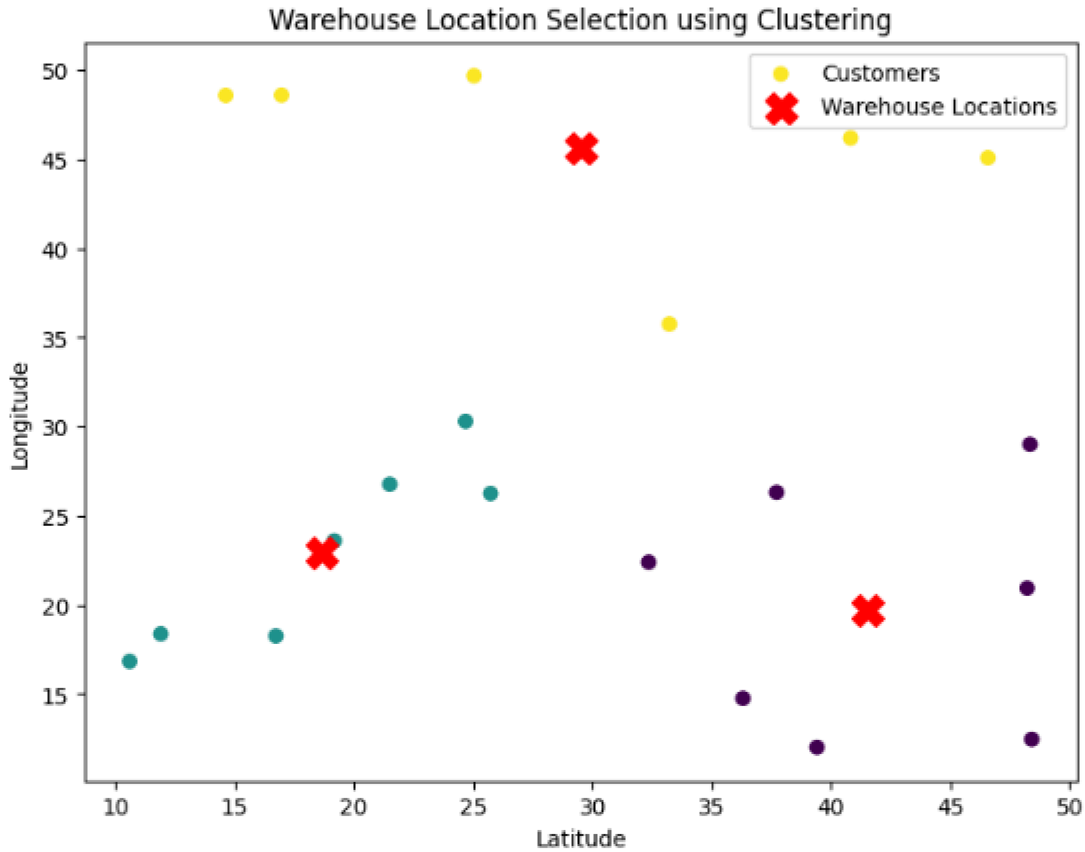
1. Load or simulate customer location data (latitude, longitude).
2. Apply K-Means clustering to identify customer groups.
3. Determine the cluster centers (potential warehouse locations).
4. Visualize the clusters and suggested warehouse locations.

🎬 Simulated customer location data (random latitudes/longitudes).
🎬 Used K-Means clustering to identify k=3 customer clusters.
🎬 Computed cluster centers, which represent optimal warehouse locations.
🎬 Plotted customers and warehouses using matplotlib.

🎬 Printed warehouse coordinates.

Conclusion- We have implemented **K-Means clustering** to determine the best location for a **new warehouse** based on existing customer locations.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from google.colab import files

#  Step 1: Load or Simulate Customer Location Data
# Simulated data (Customer locations: latitude, longitude)
data = {
    'Customer_ID': range(1, 21),
    'Latitude': np.random.uniform(10, 50, 20),  # Random latitudes
    'Longitude': np.random.uniform(10, 50, 20)  # Random longitudes
}
df = pd.DataFrame(data)

#  Step 2: Apply K-Means Clustering (Choose 3 clusters for warehouse locations)
k = 3
kmeans = KMeans(n_clusters=k, random_state=42)
df['Cluster'] = kmeans.fit_predict(df[['Latitude', 'Longitude']])

#  Step 3: Get Cluster Centers (Warehouse Locations)
warehouse_locations = kmeans.cluster_centers_

#  Step 4: Visualization
plt.figure(figsize=(8, 6))
plt.scatter(df['Latitude'], df['Longitude'], c=df['Cluster'], cmap='viridis', label='Customers')
plt.scatter(warehouse_locations[:, 0], warehouse_locations[:, 1], color='red', marker='X', s=200, label='Warehouse Locations')

plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Warehouse Location Selection using Clustering')
plt.legend()
plt.show()

# Step 5: Print Suggested Warehouse Locations
for i, loc in enumerate(warehouse_locations):
    print(f"Warehouse {i+1} Location: Latitude {loc[0]:.2f}, Longitude {loc[1]:.2f}")
```




```
Warehouse 1 Location: Latitude 41.54, Longitude 19.70
Warehouse 2 Location: Latitude 18.62, Longitude 22.91
Warehouse 3 Location: Latitude 29.54, Longitude 45.66
```

Experiment No:6

Aim: Write python program to build time hub, link and satellite

Theory: In Data Vault Modeling, a Hub, Link, and Satellite (HLS) structure is used for scalable, auditable, and flexible data warehouses.

- Hub → Stores unique business keys (e.g., Product ID, Customer ID).
- Link → Defines relationships between hubs (e.g., Customer-Order mapping).
- Satellite → Stores descriptive attributes and historical changes.

Steps to Implement Time-Based Hub, Link, and Satellite

1. Create a "Hub" table → Stores unique time-related business keys (e.g., Event ID).
2. Create a "Link" table → Connects time-related events and entities (e.g., Event ↔ Location).
3. Create a "Satellite" table → Stores descriptive details and historical changes (e.g., Event Metadata).
4. Load sample data and demonstrate the relationships.

Conclusion:

```python
import pandas as pd
import datetime

# Step 1: Create the Time Hub (Unique Events)
hub_time = pd.DataFrame({
    'Event_ID': [101, 102, 103, 104],  # Unique keys
    'Event_Name': ['Order Placed', 'Order Shipped', 'Payment Processed', 'Order Delivered'],
    'Event_Timestamp': [datetime.datetime(2024, 2, 1, 10, 30),
                         datetime.datetime(2024, 2, 1, 11, 0),
                         datetime.datetime(2024, 2, 1, 11, 15),
                         datetime.datetime(2024, 2, 1, 12, 0)]
})

# Step 2: Create the Link Table (Relationships)
link_event_location = pd.DataFrame({
    'Event_ID': [101, 102, 103, 104],  # Foreign key from hub
    'Location_ID': ['L001', 'L002', 'L003', 'L004'],  # Location where event happened
    'Link_Hash': ['H1', 'H2', 'H3', 'H4']  # Unique relationship hash
})

# Step 3: Create the Satellite Table (Descriptive Attributes)
sat_event_details = pd.DataFrame({
    'Event_ID': [101, 102, 103, 104],  # Foreign key from hub
    'Description': ['Order received from user', 'Package shipped via DHL',
                    'Payment confirmed', 'Package delivered successfully'],
    'Recorded_At': [datetime.datetime(2024, 2, 1, 10, 35),
                    datetime.datetime(2024, 2, 1, 11, 5),
                    datetime.datetime(2024, 2, 1, 11, 20),
                    datetime.datetime(2024, 2, 1, 12, 5)]
})

# Step 4: Print DataFrames
print("🔷 Time Hub (Event Information)")
print(hub_time)

print("\n🔷 Link Table (Event-Location Relationship)")
print(link_event_location)

print("\n🔷 Satellite Table (Event Details)")
print(sat_event_details)
```

```
🔹 Time Hub (Event Information)
   Event_ID         Event_Name     Event_Timestamp
0       101       Order Placed 2024-02-01 10:30:00
1       102      Order Shipped 2024-02-01 11:00:00
2       103  Payment Processed 2024-02-01 11:15:00
3       104    Order Delivered 2024-02-01 12:00:00

🔹 Link Table (Event-Location Relationship)
   Event_ID Location_ID Link_Hash
0       101        L001        H1
1       102        L002        H2
2       103        L003        H3
3       104        L004        H4

🔹 Satellite Table (Event Details)
   Event_ID                    Description         Recorded_At
0       101        Order received from user 2024-02-01 10:35:00
1       102         Package shipped via DHL 2024-02-01 11:05:00
2       103               Payment confirmed 2024-02-01 11:20:00
3       104  Package delivered successfully 2024-02-01 12:05:00
```

Experiment No:7

Aim: Data visualization using power Bi and python

Theory: **Power BI** is a powerful **data visualization tool** that integrates well with Python for advanced analytics.
Here, we will explore **Histograms** and **Scatter Plots**, which are commonly used in **data science**.

What is a Histogram?

A histogram shows the distribution of a numerical variable by dividing the data into bins. It helps understand:

- Frequency distribution
- Data skewness
- Outliers and spread

📌 Example Use Case: Analyzing the distribution of customer ages in a dataset.

What is a Scatter Plot?

A scatter plot visualizes the relationship between two numerical variables. It helps identify:

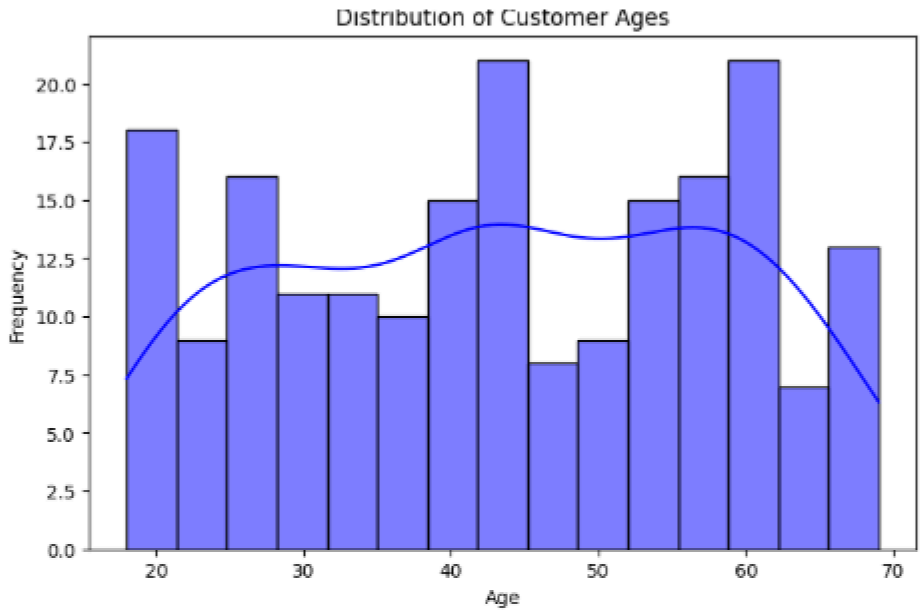
- Correlations
- Clusters in data
- Patterns and anomalies

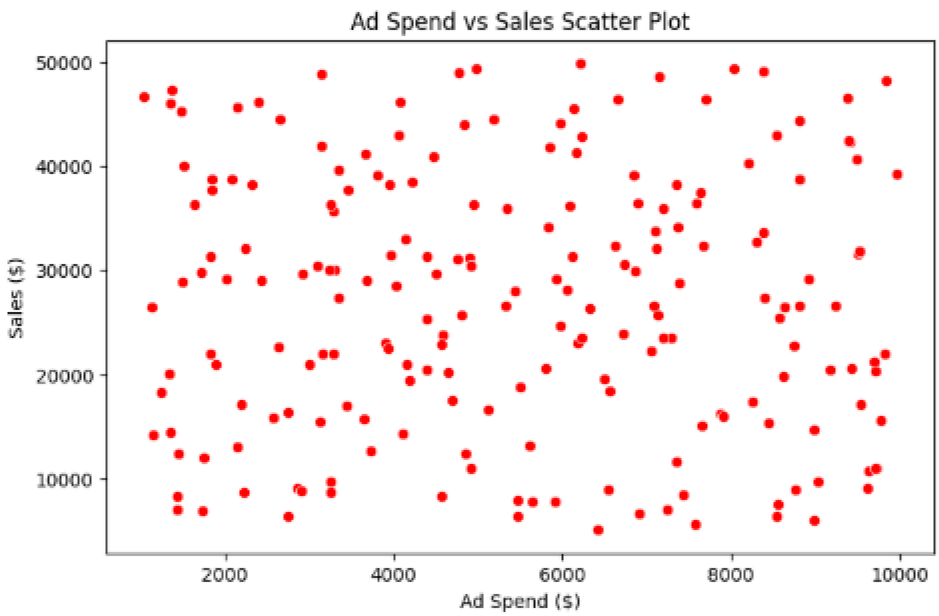
📌 Example Use Case: Understanding the relationship between advertising spend and sales.

Conclusion:- Plot histogram and scatter plot python.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#  Step 1: Create Sample Data
np.random.seed(42)
data = pd.DataFrame({
    'Age': np.random.randint(18, 70, 200),  # Random ages
    'Income': np.random.randint(20000, 120000, 200),  # Random income
    'Ad_Spend': np.random.uniform(1000, 10000, 200),  # Advertising spend
    'Sales': np.random.uniform(5000, 50000, 200)  # Sales generated
})

#  Step 2: Create Histogram (Distribution of Age)
plt.figure(figsize=(8, 5))
sns.histplot(data['Age'], bins=15, kde=True, color='blue')
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.title("Distribution of Customer Ages")
plt.show()

# Step 3: Create Scatter Plot (Ad Spend vs Sales)
plt.figure(figsize=(8, 5))
sns.scatterplot(x=data['Ad_Spend'], y=data['Sales'], color='red')
plt.xlabel("Ad Spend ($)")
plt.ylabel("Sales ($)")
plt.title("Ad Spend vs Sales Scatter Plot")
plt.show()
```

[89]


Distribution of Customer Ages
Frequency
Age
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
20
30
40
50
60
70


Ad Spend vs Sales Scatter Plot
Sales ($)
Ad Spend ($)
10000
20000
30000
40000
50000
2000
4000
6000
8000
10000

Experiment No:8

Aim: Write python code to organize data in data science.

Theory: Organizing data is a crucial step in data science to ensure clean, structured, and analyzable datasets.

This process includes:

- Handling missing values
- Removing duplicates
- Sorting and filtering data
- Renaming and restructuring columns

Explanation

1. **Created a raw dataset** with missing values and duplicates.
2. **Removed duplicates** using drop_duplicates().
3. **Filled missing values** using:
   - o Mean for **Age** (fillna(df['Age'].mean())).
   - o Median for **Salary** (fillna(df['Salary'].median())).
4. **Sorted the data** by Age using sort_values().
5. **Renamed columns** for better readability.

Conclusion: We have implanted python code to organize data

```
[ ] import pandas as pd
    import numpy as np
```

```
[93] data = {
         'ID': [102, 101, 104, 103, 105, 102],  # Duplicate ID 102
         'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Alice'],
         'Age': [25, np.nan, 35, 40, 29, 25],  # Missing Age
         'Salary': [50000, 60000, np.nan, 70000, 55000, 50000],  # Missing Salary
         'Department': ['HR', 'IT', 'Finance', 'IT', 'HR', 'HR']
     }
```

```
[94] df3= pd.DataFrame(data)
```

```
[97] print(df3)
```

```
    ID     Name   Age   Salary Department
0  102    Alice  25.0  50000.0         HR
1  101      Bob   NaN  60000.0         IT
2  104  Charlie  35.0      NaN    Finance
3  103    David  40.0  70000.0         IT
4  105      Eve  29.0  55000.0         HR
```

```
[95]
     #  Step 2: Remove Duplicates
     df3 = df3.drop_duplicates()
```

```
[96] print(df3)
```

```
    ID     Name   Age   Salary Department
0  102    Alice  25.0  50000.0         HR
1  101      Bob   NaN  60000.0         IT
2  104  Charlie  35.0      NaN    Finance
3  103    David  40.0  70000.0         IT
4  105      Eve  29.0  55000.0         HR
```

```python
# Step 3: Handle Missing Values (Fill with Mean or Default)
df3['Age'].fillna(df['Age'].mean(), inplace=True)
```

```
<ipython-input-115-887b8b456c1d>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series througl
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we :

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c

  df3['Age'].fillna(df['Age'].mean(), inplace=True)
```

```python
print(df3)
```

```
    ID     Name    Age   Salary Department
0  102    Alice  25.00  50000.0         HR
1  101      Bob  32.25  60000.0         IT
2  104  Charlie  35.00      NaN    Finance
3  103    David  40.00  70000.0         IT
4  105      Eve  29.00  55000.0         HR
```

[108]

```python
# Step 4: Sort Data by Age
df3 = df3.sort_values(by='Age')

# Step 5: Rename Columns for Clarity
df3.rename(columns={'ID': 'Employee_ID', 'Salary': 'Monthly_Salary'}, inplace=True)

# Step 6: Display Organized Data
print("Organized Data:")
print(df3)
```

```
Organized Data:
   Employee_ID     Name    Age  Monthly_Salary Department
0          102    Alice  25.00         50000.0         HR
4          105      Eve  29.00         55000.0         HR
1          101      Bob  32.25         60000.0         IT
2          104  Charlie  35.00             NaN    Finance
3          103    David  40.00         70000.0         IT
```