Practical Question

Working with various database

1)Reading and writing data in,

a. CSV format

b. Text format

c. Excel format

Code:

```
import pandas as pd

# Create simple data

data = {

    "Name": ["A", "B", "C"],

    "Marks": [80, 90, 70]

}


df = pd.DataFrame(data)

df.to_csv("data.csv", index=False)

df_csv = pd.read_csv("data.csv")

df.to_csv("data.txt", sep="\t", index=False)

df_txt = pd.read_csv("data.txt", sep="\t")

df.to_excel("data.xlsx", index=False)

df_excel = pd.read_excel("data.xlsx")
print("CSV Data:\n", df_csv)
print("\nText Data:\n", df_txt)
print("\nExcel Data:\n", df_excel)
```

2. Sending data in unstructured file format

Code

```
reviews = """ Customer Review 1:
Product is very good and easy to use.
Customer Review 2:
Service was slow but product quality is excellent.
Customer Review 3:
Satisfied with the purchase. Value for money."""

with open("customer_reviews.txt", "w") as file:
    file.write(reviews)
print("Customer reviews stored successfully in unstructured text file")
```

3. Managing data from a relational database

a. Reading SQL table

b. SQL queries

c. Dataframe

Code:

```
import sqlite3
import pandas as pd
# Connect to database
conn = sqlite3.connect("college_new.db")
cursor = conn.cursor()
# Create table
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT,
    marks INTEGER
)
""")

print("Database and table created successfully")
# Insert data (no id mentioned)
cursor.execute("INSERT INTO students (name, marks) VALUES ('Amit', 78)")
cursor.execute("INSERT INTO students (name, marks) VALUES ('Neha', 85)")
cursor.execute("INSERT INTO students (name, marks) VALUES ('Rahul', 65)")
# Commit before reading
conn.commit()
print("Data inserted successfully")
# Read full table
df = pd.read_sql("SELECT * FROM students", conn)
print(df)
# SQL query
query = "SELECT name, marks FROM students WHERE marks > 70"
df1 = pd.read_sql(query, conn)
print(df1)
# DataFrame operations
print(df1.head())
print(df1.describe())
```

```
# Close connection
conn.close()
```

5. Accessing data from the web

Code ('XMLData.xml)

```
<MyDataset>
   <Record>
      <Number>1</Number>
      <String>First</String>
      <Boolean>True</Boolean>
   </Record>
   <Record>
      <Number>2</Number>
      <String>Second</String>
      <Boolean>False</Boolean>
   </Record>
</MyDataset>
```

```
from lxml import objectify
import pandas as pd

xml = objectify.parse(open('XMLData.xml'))
root = xml.getroot()
rows = []
for record in root.getchildren():
   children = record.getchildren()

   rows.append({
      'Number': children[0].text,
      'String': children[1].text,
      'Boolean': children[2].text
```

```
    })
df = pd.DataFrame(rows)
print(df)
```

Preparing Dataset

1. Getting the data

2. Aggregate the data

3. Create data subset

4. Clean the data

5. Develop a single dataset by merging variable dataset together

6. Validating dataset

7. Extract valid data

Code

```
import pandas as pd
data1 = pd.DataFrame({
    "ID": [1, 2, 3, 4],
    "Marks": [80, 90, None, 70]
})

data2 = pd.DataFrame({
    "ID": [1, 2, 3, 4],
    "City": ["Ahd", "Surat", None, "Baroda"]
})

print("Data1:\n", data1)
print("\nData2:\n", data2)


# 2. Aggregate the data

print("\nAverage Marks:", data1["Marks"].mean())

# 3. Create data subset

subset = data1[data1["Marks"] > 75]
```

```
print("\nSubset (Marks > 75):\n", subset)

# 4. Clean the data (handle missing)
data1["Marks"] = data1["Marks"].fillna(data1["Marks"].mean())
data2["City"] = data2["City"].fillna("Unknown")
# 5. Merge datasets
df = pd.merge(data1, data2, on="ID")
# 6. Validate dataset
print("\nMissing Values:\n", df.isnull())
# 7. Extract valid data
valid_data = df.dropna()
print("\nValid Data:\n", valid_data)
```

Creating Data map and Data plan

1. Removing redundant variable

2. Removing possible errors

3. Handling missing values

4. Variable transformation

5. Manipulating categorical variable

6. Dealing with date and time

Code

```
import pandas as pd
data = {
    "ID": [1, 2, 2, 4],
    "Marks": [80, None, 150, 70],
    "City": ["Ahd", None, "Surat", "Baroda"],
```

```
    "Date": ["2024-01-01", "2024-02-05", "wrong", "2024-03-10"]
}
df = pd.DataFrame(data)
print("Original Data:\n", df)
# 1. Remove redundant column
df = df.drop("ID", axis=1)
# 2. Remove errors
df = df.drop_duplicates()
df.loc[df["Marks"] > 100, "Marks"] = None
# 3. Handle missing values
df["Marks"] = df["Marks"].fillna(df["Marks"].mean())
df["City"] = df["City"].fillna("Unknown")
# 4. Variable transformation
df["Marks"] = df["Marks"] / 100
# 5. Categorical variable (using map)
df["City"] = df["City"].map({
    "Ahd": 0,
    "Surat": 1,
    "Baroda": 2,
    "Unknown": 3
})
# 6. Date handling
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
print("\nFinal Data:\n", df)
```

Missing data

1. Dealing with missing data

2. Encoding missing data

3. Imputing missing data

Code

```
import pandas as pd
from sklearn.impute import SimpleImputer
data = {
    "Name": ["A", "B", "C", "D"],
    "Marks": [80, None, 70, None],   # only None
    "City": ["Ahmedabad", None, "Surat", "Vadodara"]
}

df = pd.DataFrame(data)
print("Original Data:\n", df)
print("\nMissing Values:\n", df.isnull())
df_removed = df.dropna()

print("\nAfter Removing Missing Data:\n", df_removed)

df["City"] = df["City"].fillna("Unknown")

imputer = SimpleImputer(strategy="mean")

df["Marks"] = imputer.fit_transform(df[["Marks"]])

print("\nFinal Data after Handling Missing Values:\n", df)
```

Conditioning with data

Code

```
import pandas as pd
df=pd.DataFrame({
    'Name': ['Asha', 'Ravi', 'Meena', 'Ravi'],
    'Marks': [85, None, 90, 85]
})
print(df)

df['Marks'] = df['Marks'].fillna(df['Marks'].mean())
```

```
print(df)
df1=df.drop_duplicates()

print(df1)
```

Filtering and selecting data

1. Aggregating data

Code

```
import pandas as pd
df = pd.DataFrame({
'Customer': ['C101','C102','C101','C103'],
'Account_Type':['Savings','Savings','Current','Current'],
'Transaction_Amount': [5000, 12000, 7000, 15000]
})
print(df)
print("Mean is ",df['Transaction_Amount'].mean())
print(df.groupby('Account_Type')['Transaction_Amount'].agg(['min', 'max', 'mean', 'count']))
print(df.groupby(['Account_Type','Customer'])['Transaction_Amount']. mean())
result = pd.pivot_table(df, values='Transaction_Amount', index='Account_Type',
aggfunc='mean' )
print(result)

df2 = pd.DataFrame({

'Date': pd.date_range('2024-01-01', periods=6),
'Transaction_Amount': [5000, 12000, 7000, 15000, 8000, 6000]
})
print(df2.set_index('Date').resample('ME').sum())


def transaction_range(x):
    return x.max() - x.min()
print(df.groupby('Account_Type')['Transaction_Amount'].agg(transaction_range))
```

2. Passing XML and HTML data

Code: - [price.html]

```
<html>

  <body>

    <h1>Product</h1>

    <p class="price">Rs. 600</p>

  </body>

</html>
```

Code: - [price.py]

```
from bs4 import BeautifulSoup
with open("price.html", "r") as file:
    html = file.read()
soup = BeautifulSoup(html, "html.parser")
price = soup.find("p", class_="price").text
print(price)
```

3. **Working with Raw Text:**

```
Code
import re
text = "Swayam course is Important 😅 pls registered!!!"
text=text.lower()
print(text)
Step 2: Remove Emojis and Special Characters
text=re.sub(r'[^a-zA-Z]', ' ', text)
print(text)
Step 3: Tokenization (Split into Words)
words=text.split()
print(words)
stop_words=["is"]
clean_words=[w for w in words if w not in stop_words]
print(clean_words)
```

4. **Graph data (Direct Graph, Undirect Graph)**

Code

```
import networkx as nx
import matplotlib.pyplot as plt
DG = nx.DiGraph()
DG.add_nodes_from(["A", "B", "C", "D"])
DG.add_edges_from([("A", "B"), ("A", "C"), ("B", "D"), ("C", "D")])
nx.draw(DG, with_labels=True, node_color='lightgreen', edge_color='blue', node_size=1500,
arrowsize=20)
plt.title("Directed Graph")
plt.show()
```

undirected Graph

Code

```
import networkx as nx
import matplotlib.pyplot as plt
# Create an empty undirected graph
G = nx.Graph()
# Add nodes
G.add_nodes_from(["A", "B", "C", "D"])
# Add edges (connections)
G.add_edges_from([("A", "B"), ("A", "C"), ("B", "D"), ("C", "D")])
```

```
# Draw the graph
nx.draw(G, with_labels=True, node_color='lightblue', edge_color='gray', node_size=1500)
plt.title("Undirected Graph")
plt.show()
```

Visualizing data

1) Creating Graphs and Charts

Code:

```
import matplotlib.pyplot as plt

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]

plt.plot(range (1, 11), values)

plt.show()
```

->Multiple lines

Code:

```
import matplotlib.pyplot as plt

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]

values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]

plt.plot(range(1, 11), values)
```

```
plt.plot(range(1, 11), values2)

plt.show()
```

-> Getting the axes: -

Code

```
import matplotlib.pyplot as plt
values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
ax = plt.axes()     # get the axes
#ax.set_xlim(1, 5)
plt.plot(range (1,11), values)
plt.show()
```

->Line Appearance

Code

```
import matplotlib.pyplot as plt

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]

values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]

plt.plot (range (1,11), values, '--')

plt.plot (range (1,11), values2, ':')

plt.show()
```

->Using colors

Code

```
import matplotlib.pyplot as plt
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
plt.plot(range(1, 11), values, 'r')
plt.plot(range(1, 11), values2, 'm')
```

```
plt.show()
```

-> Annotating the Chart

Code

```
import matplotlib.pyplot as plt

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]

plt.annotate('First Entry', xy = (1, 1), color='orange')

plt.plot(range(1, 11), values)

plt.show()
```

->Using legend()

Code

```
import matplotlib.pyplot as plt

values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]

values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]

line1 = plt.plot(range(1, 11), values)

line2 = plt.plot(range(1, 11), values2)

plt.legend(['Sales- 2025', 'Sales- 2026'], loc=4)

    plt.show()
```

1.Creating pie charts

Code

```
import matplotlib.pyplot as plt
values = [5, 8, 9, 10, 4, 7]
labels = ['A', 'B', 'C', 'D', 'E', 'F']
colors = ['b', 'g', 'r', 'c', 'm', 'y']
explode = (0, 0.2, 0, 0, 0, 0)  # highlight second slice
# Create pie chart
```

plt.pie(values, labels=labels, colors=colors, explode=explode, autopct='%1.2 f%%', shadow=True)
plt.title('Values Pie Chart')
plt.show()

2.Creating bar charts

Code

import matplotlib.pyplot as plt
values = [5, 8, 9, 10, 4, 7]
widths = [0.7, 0.8, 0.7, 0.7, 0.7, 0.7]
colors = ['b', 'r', 'b', 'b', 'b', 'b']
plt.bar(range(0,6), values, width=widths, color=colors, align='center')
plt.show()

3.Creating histograms charts

Code:

import numpy as np
import matplotlib.pyplot as plt
x = 20 * np.random.randn(10000)
plt.hist(x, 25, range= (-50, 50), histtype='stepfilled',
align='mid', color='g', label='Test Data')
plt.legend()
plt.title('Step Filled Histogram')
plt.show()

4.Creating boxplot

Code

import matplotlib.pyplot as plt
data = [2, 4, 5, 6, 7, 8, 9]
plt.boxplot(data)
plt.show()

5.Creating Scatter plot

Code: -

import matplotlib.pyplot as plt

```
x = [1,2,3,4,5]

y = [40, 45, 50, 60, 65]

plt.scatter(x, y)

 plt.xlabel("Study Hours")

plt.ylabel("Marks")

plt.show()
```

6.Creating *Advanced* Scatter plot

Code:

```
x = [1,2,3,4]
classA = [60,65,70,75]
classB = [55,60,68,72]
plt.scatter(x, classA, color='blue', label='Class A', marker='^')
plt.scatter(x, classB, color='red', label='Class B' ,marker='^')
plt.xlabel("Student Number")
plt.ylabel("Marks")
plt.legend()
plt.show()
```

7.Plotint time series

Code: -

```
import pandas as pd
import matplotlib.pyplot as plt
dates = pd.date_range('2026-03-01','2026-03-10')
sales = [10,20,30,40,25,35,15,45,20,30]
plt.plot(dates, sales)
plt.xlabel("Sales Date")
plt.ylabel("Sales Value")
plt.title("Time Series Plot")
plt.xticks(rotation=45)
plt.show()
```

8. Working with geographical data

Code: -

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
# Longitude values
lon = [72.5714, 77.1025, 72.8777, 80.2707, 88.3639]
# Latitude values
lat = [23.0225, 28.7041, 19.0760, 13.0827, 22.5726]
# City names
cities = ["Ahmedabad", "Delhi", "Mumbai", "Chennai", "Kolkata"]
# Create map
m = Basemap(projection='merc',
            llcrnrlat=5, urcrnrlat=35,
            llcrnrlon=65, urcrnrlon=95,
            resolution='l')
# Draw map details
m.drawcoastlines()
m.drawcountries()
m.drawstates()
# Convert lat/lon to map coordinates
x, y = m(lon, lat)
# Plot points
m.scatter(x, y, color='red', marker='o')

# Add city names
for i in range(len(cities)):
    plt.text(x[i], y[i], cities[i])
plt.title("Geographical Locations of Major Indian Cities")
plt.show()
```

Exploring data analysis for the given dataset

1)Measuring central tendency, variance, range, and percentile

Code

```
import numpy as np
data = np.array([10, 20, 30, 40, 50])
# Central Tendency
mean = np.mean(data)
median = np.median(data)
# Variance
variance = np.var(data)
# Range
data_range = np.max(data) - np.min(data)
# Percentile
percentile_25 = np.percentile(data, 25)
percentile_75 = np.percentile(data, 75)
print("Data:", data)
print("Mean:", mean)
print("Median:", median)
print("Variance:", variance)
print("Range:", data_range)
print("25th Percentile:", percentile_25)
print("75th Percentile:", percentile_75)
```

2)Measures of normality

(A) Skewness

Code

```
from scipy import stats
```

```
data = [5, 10, 15, 50, 60]
skew_value = round(stats.skew(data), 2)
print("Skewness:", skew_value)
```

(B) Kurtosis

Code

```
from scipy import stats
data = [10, 20, 30, 40, 50]
k = stats.kurtosis(data, fisher=False) # fisher=False gives k (not excess)
print("Kurtosis:", round(k,2))
```

Working with Categorical Data

1) get_dummies()

Code:

```
import pandas as pd
data = {
    'Name': ['Amit', 'Riya', 'John', 'Sneha', 'Raj'],
    'Placement': ['Placed', 'Not Placed', 'Placed', 'Not Placed', 'Placed']
}

df = pd.DataFrame(data)
print(df)
dummy = pd.get_dummies(df, columns=['Placement'], dtype=int)

print(dummy)
```

2) using LabelEncoder

Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

```
data = {
   'Name': ['Amit', 'Riya', 'John', 'Sneha', 'Raj'],
   'Placement': ['Placed', 'Not Placed', 'Placed', 'Not Placed', 'Placed']
}

df = pd.DataFrame(data)
le = LabelEncoder()
df['Placement_Encoded'] = le.fit_transform(df['Placement'])
print(df)
```

Visualization for EDA

1) Using boxplots

Code:

```
import matplotlib.pyplot as plt

data1 = [2,4,5,6,7]

data2 = [3,5,6,8,9]

plt.boxplot([data1, data2])

plt.show()
```

2) t-tests after boxplots

Code:

```
import matplotlib.pyplot as plt

from scipy.stats import ttest_ind

group1 = [45, 47, 50, 46, 48]

group2 = [55, 58, 60, 57, 59]
```

```
plt.boxplot([group1, group2], labels=['Group 1', 'Group 2'])
plt.title("Boxplot Comparison")
plt.show()
t_stat, p_value = ttest_ind(group1, group2)
print("T-statistic:", round(t_stat, 2))
print("P-value:", round(p_value, 4))
if p_value < 0.05:
    print("Significant difference between groups")
else:
    print("No significant difference")
```

3)Working with correlation

Code:

```
import pandas as pd
data = {
    'Marks': [40, 50, 60, 70, 80],
    'Hours': [2, 3, 4, 5, 6]
}
df = pd.DataFrame(data)
corr = df.corr()
print(corr)
```

4) Working with Data Distributions

Code:

```
import numpy as np
from sklearn.preprocessing import scale
```

```
sales = np.array([45, 47, 50, 46, 48, 49, 120])
z_scores = scale(sales)
print("Original Sales:", sales)
print("Z-scores:", np.round(z_scores, 2))
# Transformations
transformations = {
    'Original': lambda x: x,
    'Log': lambda x: np.log(x + 1),
    'Reciprocal': lambda x: 1/x,
    'Square Root': lambda x: np.sqrt(x),
    'Square': lambda x: x**2
}
print("\nTransformed Sales:")
for name, func in transformations.items():
    transformed = func(sales)
    print(f"{name}: {np.round(transformed, 2)}")
```

Dimensionality Reducing with Principal Component Analysis

Code:

```
import numpy as np
X = np.array([
    [40, 2],
    [50, 4],
    [60, 6]
])
print("Original Data:\n", X)
mean = np.mean(X, axis=0)
```

```
std = np.std(X, axis=0)
Z = (X - mean) / std
print("\nMean:\n", mean)
print("Standard Deviation:\n", std)
print("\nStandardized Data (Z):\n", Z)
cov_matrix = np.cov(Z.T)
print("\nCovariance Matrix:\n", cov_matrix)
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print("\nEigenvalues:\n", eigenvalues)
print("Eigenvectors:\n", eigenvectors)
idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
print("\nSorted Eigenvalues:\n", eigenvalues)
print("Sorted Eigenvectors:\n", eigenvectors)
PC = np.dot(Z, eigenvectors)
print("\nPrincipal Components (Projected Data):\n", PC)
```

Clustering with K-means

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
marks = [40, 45, 50, 90, 95, 100]
students = ['A', 'B', 'C', 'D', 'E', 'F']
X = np.array(marks).reshape(-1, 1)
```

```
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
for i in range(len(marks)):
    if labels[i] == labels[0]:
        plt.scatter(i, marks[i], color='red')
    else:
        plt.scatter(i, marks[i], color='green')
     plt.text(i, marks[i]+1, students[i], ha='center')
# Plot centroids
plt.scatter([1,4], centroids.flatten(), color='black', marker='X', s=200)
# Axis labels
plt.xticks(range(len(students)), students)
plt.xlabel("Students")
plt.ylabel("Marks")
plt.title("K-Means Clustering with Groups")
plt.show()
```

Univariate method

1) Z-Score

```
import numpy as np
data = np.array([10, 12, 13, 15, 18, 100])
mean = np.mean(data)
std = np.std(data)
z_scores = (data - mean) / std
print("Z-scores:", z_scores)
```

```
outliers = data[np.abs(z_scores) > 3]

print("Outliers:", outliers)
```

2) IQR

Code:

```
import numpy as np
data = np.array([10, 12, 13, 15, 18, 100])
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
outliers = data[(data < lower_limit) | (data > upper_limit)]
print("Q1:", Q1)
print("Q3:", Q3)
print("IQR:", IQR)
print("Lower Limit:", lower_limit)
print("Upper Limit:", upper_limit)
print("Outliers:", outliers)
```

using boxplot

Code:

```
import numpy as np
import matplotlib.pyplot as plt
data = np.array([10, 12, 13, 15, 18, 100])
plt.boxplot(data)
plt.title("Box Plot for Outlier Detection")
```

```
plt.show()
```