**NAME: DATE:**

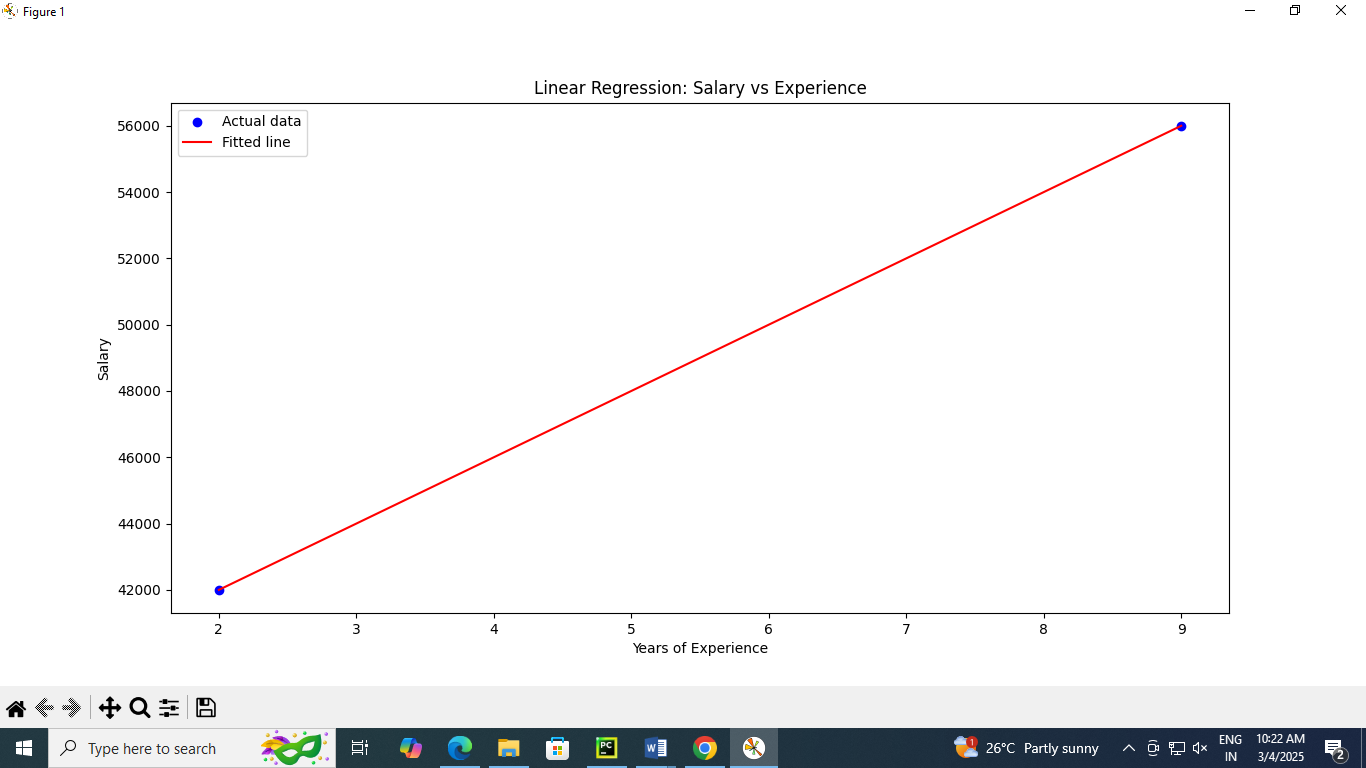
**ROLL\_NO: BATCH:**

**PRACT\_NAME: 7 regression analysis using linear regression**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
data = {  
 'YearsExperience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  
 'Salary': [40000, 42000, 44000, 46000, 48000, 50000, 52000, 54000, 56000, 58000]  
}  
  
df = pd.DataFrame(data)  
  
  
X = df[['YearsExperience']] # Independent variable (Features)  
y = df['Salary'] # Dependent variable (Target)  
  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
  
y\_pred = model.predict(X\_test)  
  
  
  
mse = mean\_squared\_error(y\_test, y\_pred)  
print(f"Mean Squared Error: {mse}")  
  
  
r2 = r2\_score(y\_test, y\_pred)  
print(f"R-squared: {r2}")  
  
  
plt.scatter(X\_test, y\_test, color='blue', label='Actual data')  
plt.plot(X\_test, y\_pred, color='red', label='Fitted line')  
plt.xlabel('Years of Experience')  
plt.ylabel('Salary')  
plt.title('Linear Regression: Salary vs Experience')  
plt.legend()  
plt.show()

new\_data = np.array([[11]]) # Example: Predict salary for 11 years of experience  
predicted\_salary = model.predict(new\_data)  
print(f"Predicted Salary for 11 years of experience: ${predicted\_salary[0]:,.2f}")

**Output:**



**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME: 8 association rule mining using apriori.**

import pandas as pd  
from mlxtend.frequent\_patterns import apriori, association\_rules  
  
data = {'Milk': [1, 1, 0, 1, 1],  
 'Bread': [1, 1, 1, 1, 0],  
 'Butter': [0, 1, 1, 1, 1],  
 'Cheese': [1, 0, 1, 1, 1]}  
  
df = pd.DataFrame(data)  
  
frequent\_itemsets = apriori(df, min\_support=0.6, use\_colnames=True)  
  
  
rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)  
  
  
print("Frequent Itemsets:")  
print(frequent\_itemsets)  
  
  
print("\nAssociation Rules:")  
print(rules)

**OUTPUT:**

Frequent Itemsets:

support itemsets

0 0.8 (Milk)

1 0.8 (Bread)

2 0.8 (Butter)

3 0.8 (Cheese)

4 0.6 (Milk, Bread)

5 0.6 (Milk, Butter)

6 0.6 (Cheese, Milk)

7 0.6 (Butter, Bread)

8 0.6 (Cheese, Bread)

9 0.6 (Cheese, Butter)

Association Rules:

antecedents consequents antecedent support ... jaccard certainty kulczynski

0 (Milk) (Bread) 0.8 ... 0.6 -0.25 0.75

1 (Bread) (Milk) 0.8 ... 0.6 -0.25 0.75

2 (Milk) (Butter) 0.8 ... 0.6 -0.25 0.75

3 (Butter) (Milk) 0.8 ... 0.6 -0.25 0.75

4 (Cheese) (Milk) 0.8 ... 0.6 -0.25 0.75

5 (Milk) (Cheese) 0.8 ... 0.6 -0.25 0.75

6 (Butter) (Bread) 0.8 ... 0.6 -0.25 0.75

7 (Bread) (Butter) 0.8 ... 0.6 -0.25 0.75

8 (Cheese) (Bread) 0.8 ... 0.6 -0.25 0.75

9 (Bread) (Cheese) 0.8 ... 0.6 -0.25 0.75

10 (Cheese) (Butter) 0.8 ... 0.6 -0.25 0.75

11 (Butter) (Cheese) 0.8 ... 0.6 -0.25 0.75

[12 rows x 14 columns]

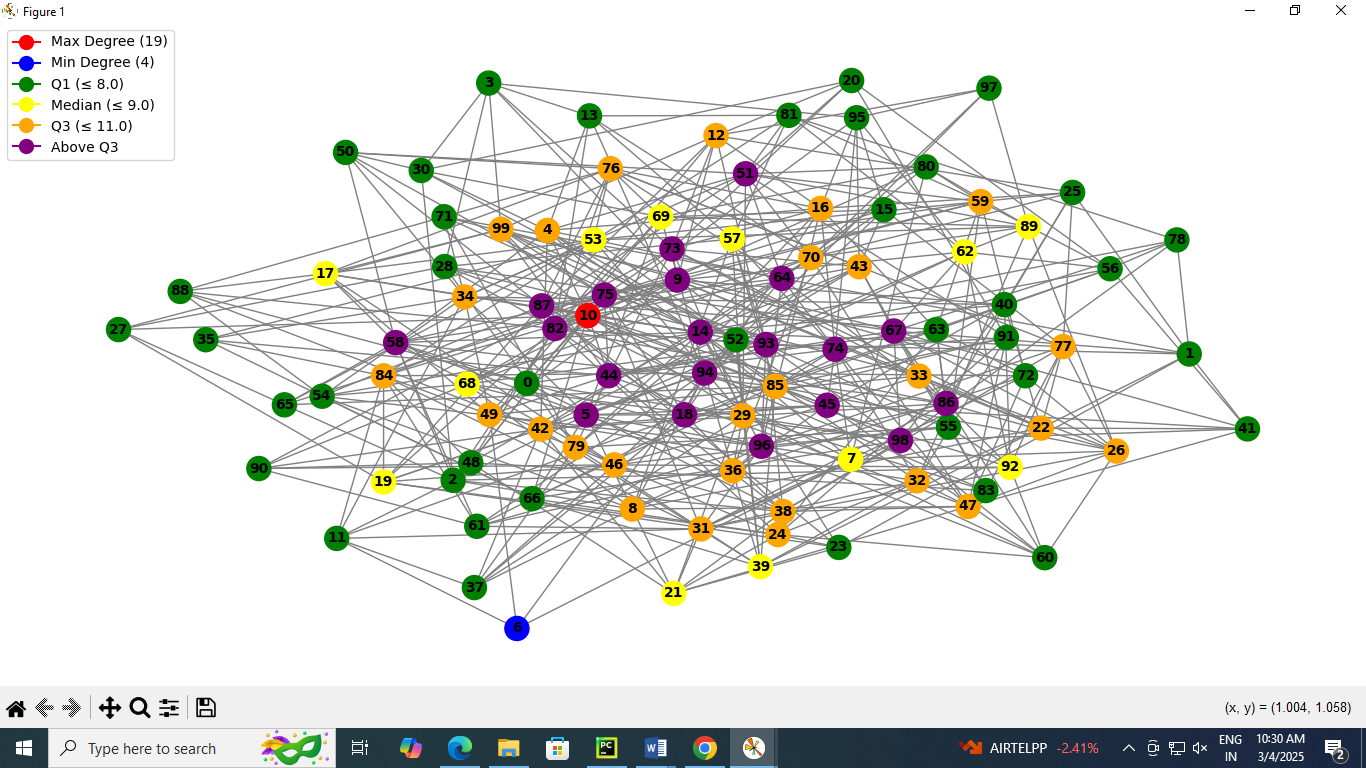
**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME: 12.graph visualization of a network using maximum,minimum,median,first quartile,and third quartile.**

import networkx as nx  
import matplotlib.pyplot as plt  
import numpy as np  
  
G = nx.erdos\_renyi\_graph(n=100, p=0.1) # A random graph with 100 nodes and a 10% chance of edge creation  
  
  
degree\_sequence = [G.degree(node) for node in G.nodes()]  
  
  
max\_degree = np.max(degree\_sequence)  
min\_degree = np.min(degree\_sequence)  
median\_degree = np.median(degree\_sequence)  
q1 = np.percentile(degree\_sequence, 25)  
q3 = np.percentile(degree\_sequence, 75)  
  
print(f"Maximum Degree: {max\_degree}")  
print(f"Minimum Degree: {min\_degree}")  
print(f"Median Degree: {median\_degree}")  
print(f"First Quartile (Q1): {q1}")  
print(f"Third Quartile (Q3): {q3}")  
  
node\_colors = []  
for degree in degree\_sequence:  
 if degree == max\_degree:  
 node\_colors.append('red') # High-degree node (maximum degree)  
 elif degree == min\_degree:  
 node\_colors.append('blue') # Low-degree node (minimum degree)  
 elif degree <= q1:  
 node\_colors.append('green') # Nodes in the first quartile  
 elif degree <= median\_degree:  
 node\_colors.append('yellow') # Nodes up to the median degree  
 elif degree <= q3:  
 node\_colors.append('orange') # Nodes up to the third quartile  
 else:  
 node\_colors.append('purple') # Nodes greater than the third quartile  
  
  
plt.figure(figsize=(10, 8))  
  
pos = nx.spring\_layout(G, seed=42) # Positions for all nodes  
nx.draw(G, pos, with\_labels=True, node\_size=300, node\_color=node\_colors, font\_size=10, font\_weight='bold', edge\_color='gray')  
  
  
import matplotlib.lines as mlines  
  
max\_label = mlines.Line2D([], [], color='red', marker='o', markersize=10, label=f"Max Degree ({max\_degree})")  
min\_label = mlines.Line2D([], [], color='blue', marker='o', markersize=10, label=f"Min Degree ({min\_degree})")  
q1\_label = mlines.Line2D([], [], color='green', marker='o', markersize=10, label=f"Q1 (≤ {q1})")  
median\_label = mlines.Line2D([], [], color='yellow', marker='o', markersize=10, label=f"Median (≤ {median\_degree})")  
q3\_label = mlines.Line2D([], [], color='orange', marker='o', markersize=10, label=f"Q3 (≤ {q3})")  
above\_q3\_label = mlines.Line2D([], [], color='purple', marker='o', markersize=10, label=f"Above Q3")  
  
plt.legend(handles=[max\_label, min\_label, q1\_label, median\_label, q3\_label, above\_q3\_label], loc="upper left")  
  
  
plt.title("Network Visualization with Degree Statistics Coloring")  
plt.show()

**OUTPUT:**



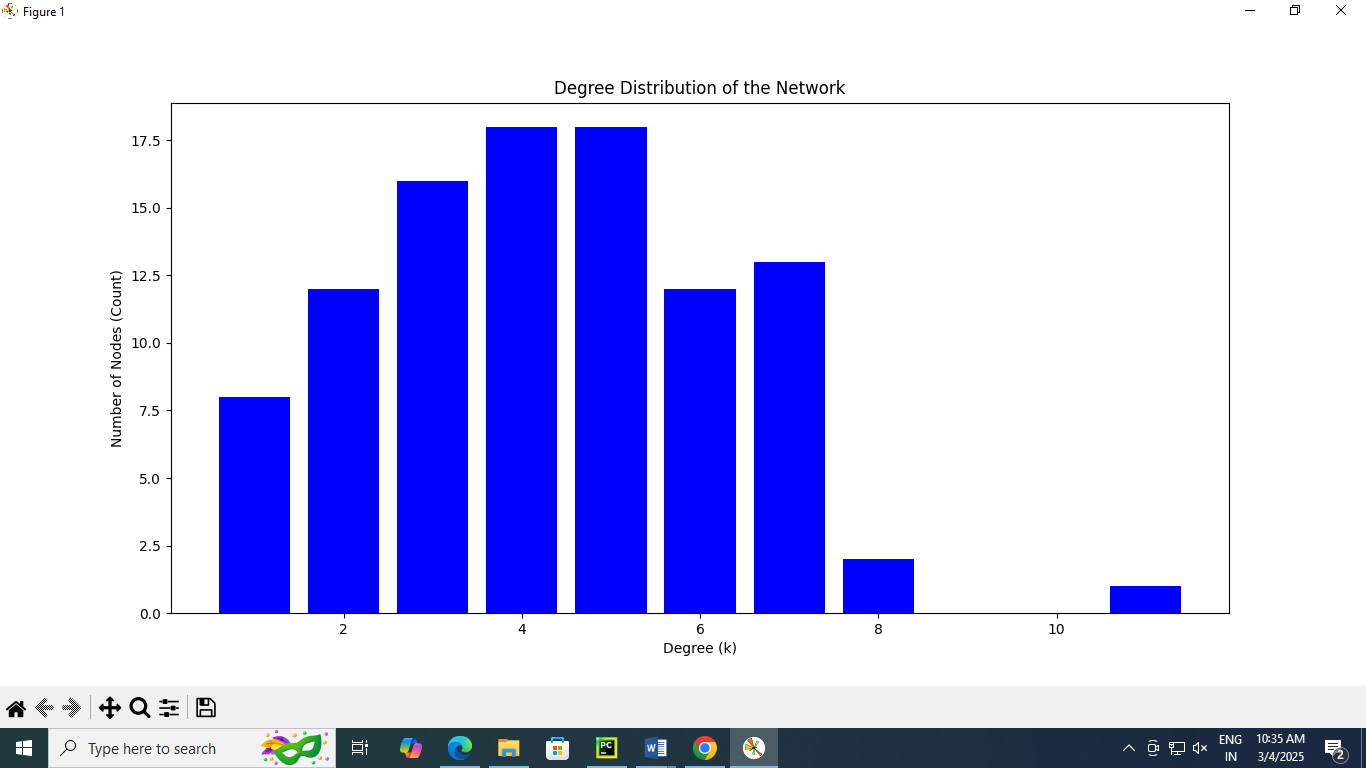
**NAME: DATE:**

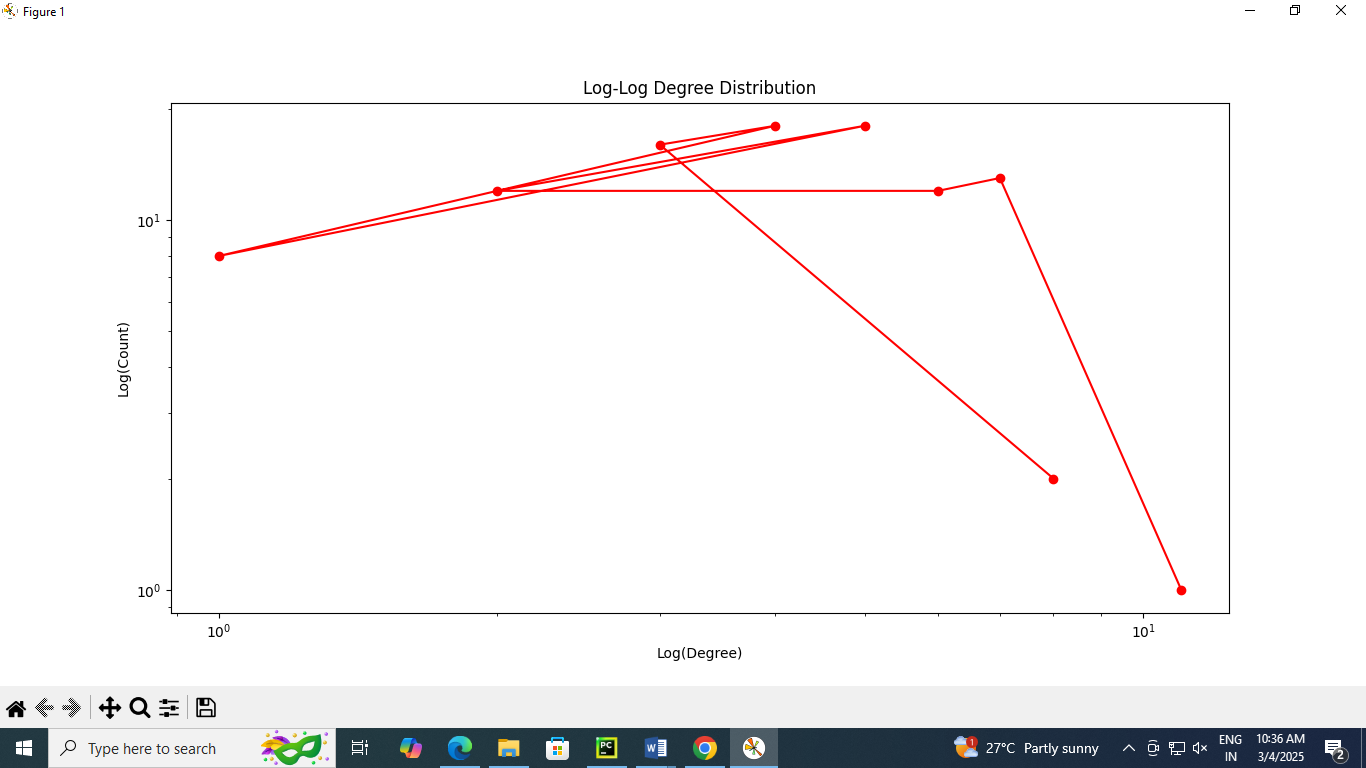
**ROLL\_NO: BATCH:**

**PRACT\_NAME:11 use of degree distribution of a network.**

import networkx as nx  
import matplotlib.pyplot as plt  
  
G = nx.erdos\_renyi\_graph(n=100, p=0.05) # A random graph with 100 nodes and a 5% chance of edge creation  
  
degree\_sequence = [G.degree(node) for node in G.nodes()]  
  
degree\_count = {}  
for degree in degree\_sequence:  
 degree\_count[degree] = degree\_count.get(degree, 0) + 1  
  
degrees = list(degree\_count.keys()) # List of degrees  
counts = list(degree\_count.values()) # Corresponding counts of nodes with each degree  
  
  
plt.figure(figsize=(8, 6))  
plt.bar(degrees, counts, color='b')  
plt.xlabel('Degree (k)')  
plt.ylabel('Number of Nodes (Count)')  
plt.title('Degree Distribution of the Network')  
plt.show()  
  
plt.figure(figsize=(8, 6))  
plt.loglog(degrees, counts, marker='o', color='r')  
plt.xlabel('Log(Degree)')  
plt.ylabel('Log(Count)')  
plt.title('Log-Log Degree Distribution')  
plt.show()

**OUTPUT:**





**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME: 6 classification using random forest.**

from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.datasets import load\_iris  
from sklearn.metrics import accuracy\_score, classification\_report  
  
  
data = load\_iris()  
X = data.data # Features  
y = data.target # Target variable (class labels)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
  
rf\_classifier.fit(X\_train, y\_train)  
  
y\_pred = rf\_classifier.predict(X\_test)  
  
  
accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {accuracy \* 100:.2f}%")  
  
print("Classification Report:\n", classification\_report(y\_test, y\_pred))  
  
print("Feature Importance:")  
for feature, importance in zip(data.feature\_names, rf\_classifier.feature\_importances\_):  
 print(f"{feature}: {importance:.4f}")

**OUTPUT:**

Accuracy: 100.00%

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 19

1 1.00 1.00 1.00 13

2 1.00 1.00 1.00 13

accuracy 1.00 45

macro avg 1.00 1.00 1.00 45

weighted avg 1.00 1.00 1.00 45

Feature Importance:

sepal length (cm): 0.1041

sepal width (cm): 0.0446

petal length (cm): 0.4173

petal width (cm): 0.4340

**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME: 2.read the csv file**

import pandas as pd  
  
# Load the data from a CSV file  
file\_path = 'hh.csv' # Replace with your file path  
sales\_data = pd.read\_csv(file\_path)  
  
# Display basic data structure  
print("Display rows of the dataset:")  
print(sales\_data)

**OUTPUT:**

C:\Users\comp\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/comp/PycharmProjects/pythonProject/jj.py

Display rows of the dataset:

Date Region Product Sales

0 1/5/2023 North A 100

1 1/12/2023 South B 200

2 2/1/2023 North A 150

3 2/14/2023 East C 300

4 2/21/2023 West B 250

**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME: 3.perfrom data filtering , and calculate aggregate statistics.**

import pandas as pd  
  
# Load the data from a CSV file  
file\_path = 'hh.csv' # Replace with your file path  
sales\_data = pd.read\_csv(file\_path)  
  
# Display basic data structure  
print("First few rows of the dataset:")  
print(sales\_data.head())  
  
# \*\*Data Filtering\*\*: Select data where Sales > 150  
filtered\_data = sales\_data[sales\_data['Sales'] > 150]  
print("\nFiltered Data (Sales > 150):")  
print(filtered\_data)  
  
# \*\*Aggregate Statistics\*\*: Calculate total and average sales by region  
region\_sales = sales\_data.groupby('Region')['Sales'].agg(['sum', 'mean']).reset\_index()  
print("\nTotal and Average Sales by Region:")  
print(region\_sales)  
  
# \*\*Aggregate Statistics\*\*: Calculate total sales and count by product  
product\_stats = sales\_data.groupby('Product')['Sales'].agg(['sum', 'count']).reset\_index()  
print("\nTotal Sales and Transaction Count by Product:")  
print(product\_stats)

**OUTPUT:**

C:\Users\comp\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/comp/PycharmProjects/pythonProject/3rd.py

First few rows of the dataset:

Date Region Product Sales

0 1/5/2023 North A 100

1 1/12/2023 South B 200

2 2/1/2023 North A 150

3 2/14/2023 East C 300

4 2/21/2023 West B 250

s

Filtered Data (Sales > 150):

Date Region Product Sales

1 1/12/2023 South B 200

3 2/14/2023 East C 300

4 2/21/2023 West B 250

Total and Average Sales by Region:

Region sum mean

0 East 300 300.0

1 North 250 125.0

2 South 200 200.0

3 West 250 250.0

Total Sales and Transaction Count by Product:

Product sum count

0 A 250 2

1 B 450 2

2 C 300 1

**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME: 4.calculate total sales by month**

import pandas as pd

# Sample data (replace with your actual data source, e.g., CSV file)

data = {  
 "Date": ["2023-01-15", "2023-01-20", "2023-02-10", "2023-02-15", "2023-03-01"],  
 "Sales": [200, 150, 300, 250, 400],  
}

# Convert the data to a DataFrame  
df = pd.DataFrame(data)

# Ensure the Date column is in datetime format  
df['Date'] = pd.to\_datetime(df['Date'])

# Extract year and month from the Date column  
df['YearMonth'] = df['Date'].dt.to\_period('M')

# Group by YearMonth and calculate total sales  
monthly\_sales = df.groupby('YearMonth')['Sales'].sum()

# Print the result  
print(monthly\_sales)

**OUTPUT:**

YearMonth

2023-01 350

2023-02 550

2023-03 400

Freq: M, Name: Sales, dtype: int64

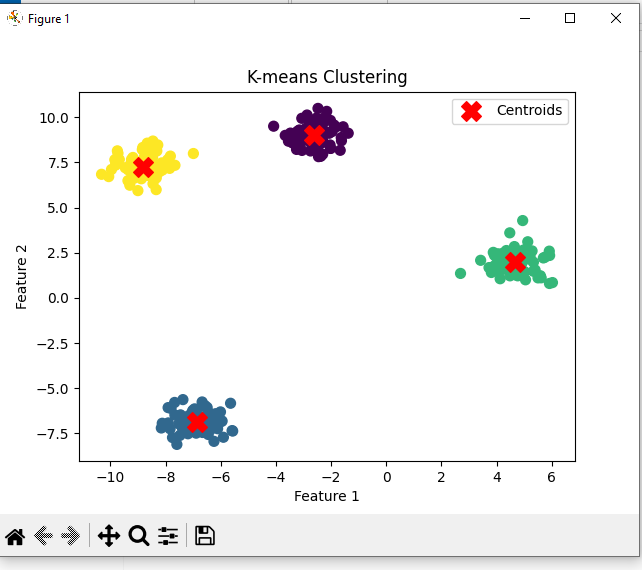
**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME: 5.Implement the clustering using K-means**

import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn.datasets import make\_blobs  
  
# Generate sample data  
X, y = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.6, random\_state=42)  
  
# Visualize the raw data  
plt.scatter(X[:, 0], X[:, 1], s=50, c='gray', marker='o')  
plt.title("Raw Data")  
plt.xlabel("Feature 1")  
plt.ylabel("Feature 2")  
plt.show()  
  
# Apply K-means clustering  
kmeans = KMeans(n\_clusters=4, random\_state=42)  
kmeans.fit(X)  
  
# Get cluster labels and centroids  
labels = kmeans.labels\_  
centroids = kmeans.cluster\_centers\_  
  
# Visualize the clustered data  
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)  
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X', label='Centroids')  
plt.title("K-means Clustering")  
plt.xlabel("Feature 1")  
plt.ylabel("Feature 2")  
plt.legend()  
plt.show()

**OUTPUT:**

****

**Name : Roll No :**

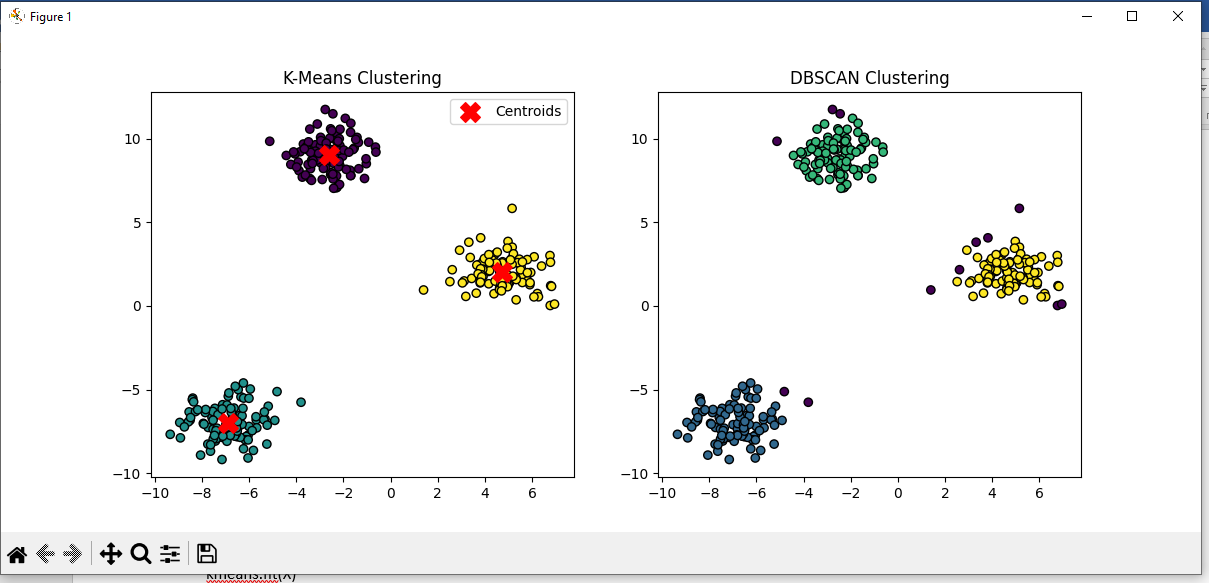
**Batch : Date :**

**Practical No 9 :- Visualize the result of the clustering and compare.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.datasets import make\_blobs  
from sklearn.cluster import KMeans, DBSCAN  
  
# Generate synthetic dataset  
X, y = make\_blobs(n\_samples=300, centers=3, cluster\_std=1.0, random\_state=42)  
  
# Apply K-Means clustering  
kmeans = KMeans(n\_clusters=3, random\_state=42)  
kmeans\_labels = kmeans.fit\_predict(X)  
  
# Apply DBSCAN clustering  
dbscan = DBSCAN(eps=0.8, min\_samples=5)  
dbscan\_labels = dbscan.fit\_predict(X)  
  
# Plot the results  
fig, axes = plt.subplots(1, 2, figsize=(12, 5))  
  
# K-Means Clustering  
axes[0].scatter(X[:, 0], X[:, 1], c=kmeans\_labels, cmap='viridis', marker='o', edgecolor='k')  
axes[0].scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], c='red', marker='X', s=200, label="Centroids")  
axes[0].set\_title("K-Means Clustering")  
axes[0].legend()  
  
# DBSCAN Clustering  
axes[1].scatter(X[:, 0], X[:, 1], c=dbscan\_labels, cmap='viridis', marker='o', edgecolor='k')  
axes[1].set\_title("DBSCAN Clustering")  
  
plt.show()

Output:



**NAME: DATE:**

**ROLL\_NO: BATCH:**

**PRACT\_NAME:10 visualize correaltion matrix using a pseudocolor plot.**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
# Generate a random dataset  
np.random.seed(42)  
data = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])  
  
# Compute the correlation matrix  
corr\_matrix = data.corr()  
  
# Create a pseudocolor plot (heatmap)  
plt.figure(figsize=(8, 6))  
plt.pcolormesh(corr\_matrix, cmap='coolwarm', edgecolors='k')  
plt.colorbar(label='Correlation Coefficient')  
  
# Add labels at the center of each grid  
plt.xticks(np.arange(0.5, len(corr\_matrix.columns), 1), corr\_matrix.columns)  
plt.yticks(np.arange(0.5, len(corr\_matrix.index), 1), corr\_matrix.index)  
plt.title('Correlation Matrix Heatmap')  
  
plt.show()

**OUTPUT:**

