**RCP ACS SHIRPUR**

**BCA 607(C) Data Mining Assignments:**

**1] Calculate the mean and standard deviation.**

import numpy as np

data = [1, 2, 3, 4, 5]

mean = np.mean(data)

std\_dev = np.std(data)

print("Mean:",mean)

print("Standard Deviation:",std\_dev)

**Output:**

Mean: 3.0

Standard Deviation: 1.4142135623730951

**2] Read the CSV file.**

import pandas as pd

df = pd.read\_csv("company-sales.csv")

print(df.head())

**Output:**

month\_number facecream facewash toothpaste bathingsoap shampoo \

0 1 2500 1500 5200 9200 1200

1 2 2630 1200 5100 6100 2100

2 3 2140 1340 4550 9550 3550

3 4 3400 1130 5870 8870 1870

4 5 3600 1740 4560 7760 1560

moisturizer total\_units total\_profit

0 1500 21100 211000

1 1200 18330 183300

2 1340 22470 224700

3 1130 22270 222700

4 1740 20960 209600

**3. Perform data filtering, and calculate aggregate statistics.**

import pandas as pd

# Creating DataFrame

data = {

'Name': ["harshu", "chetana", "jayeshri"],

'Age': [25, 30, 45],

'Salary': [500000, 10000, 50000],

'Department': ['HR', "Finance", 'Production'],

}

df = pd.DataFrame(data)

print("\nOriginal Data\n")

print(df)

# Filtering Employees by Department

hr\_emp = df[df["Department"] == "HR"]

fn\_emp = df[df["Department"] == "Finance"]

pr\_emp = df[df["Department"] == "Production"]

# Filtering Employees by Salary

high\_salary\_emp = df[df["Salary"] > 55000]

low\_salary\_emp = df[df["Salary"] < 55000]

print("\nEmployees in HR Department\n")

print(hr\_emp)

print("\nEmployees in Finance Department\n")

print(fn\_emp)

print("\nEmployees in Production Department\n")

print(pr\_emp)

print("\nEmployees with a Salary > 55000\n")

print(high\_salary\_emp)

print("\nEmployees with a Salary < 55000\n")

print(low\_salary\_emp)

# Aggregate Statistics

aggregate\_stats = {

'Age Mean': df['Age'].mean(),

'Salary Mean': df["Salary"].mean(),

'Age Sum': df['Age'].sum(),

'Salary Sum': df['Salary'].sum(),

'Age Count': df['Age'].count(),

'Salary Count': df['Salary'].count(),

}

print("\nAggregate Statistics\n")

for key, value in aggregate\_stats.items():

print(f"{key}: {value}\n")

**Output:**

Original Data

Name Age Salary Department

0 harshu 25 500000 HR

1 chetana 30 10000 Finance

2 jayeshri 45 50000 Production

Employees in HR Department

Name Age Salary Department

0 harshu 25 500000 HR

Employees in Finance Department

Name Age Salary Department

1 chetana 30 10000 Finance

Employees in Production Department

Name Age Salary Department

2 jayeshri 45 50000 Production

Employees with a Salary > 55000

Name Age Salary Department

0 harshu 25 500000 HR

Employees with a Salary < 55000

Name Age Salary Department

1 chetana 30 10000 Finance

2 jayeshri 45 50000 Production

**Aggregate Statistics**

Age Mean: 33.333333333333336

Salary Mean: 186666.66666666666

Age Sum: 100

Salary Sum: 560000

Age Count: 3

Salary Count: 3

**4] Calculate total sales by month**

import pandas as pd

data = {

"Months": ["jan", "Feb", "Mar", "April", "May","Jun", "July", "Aug", "Sep", "Oct", "Nov","Dec"],

"Sales": [1000, 2000, 3000, 4000, 5000, 6000, 90000, 80000, 90000, 10000, 11000, 12000],

}

df = pd.DataFrame(data)

print(df)

high\_Sale = df[df["Sales"] >= 5000]

low\_Sale = df[df["Sales"] <= 3000]

print('Highest Sales')

print(high\_Sale)

print("Lowest Sales")

print(low\_Sale)

**Output:**

Months Sales

0 jan 1000

1 Feb 2000

2 Mar 3000

3 April 4000

4 May 5000

5 Jun 6000

6 July 90000

7 Aug 80000

8 Sep 90000

9 Oct 10000

10 Nov 11000

11 Dec 12000

Highest Sales

Months Sales

4 May 5000

5 Jun 6000

6 July 90000

7 Aug 80000

8 Sep 90000

**5] Implement the Clustering using K-means.**

import numpy as np

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=1.0, random\_state=42)

# Apply K-means clustering

kmeans = KMeans(n\_clusters=4, random\_state=42)

kmeans.fit(X)

labels = kmeans.labels\_

centroids = kmeans.cluster\_centers\_

# Plot the clustered dataplt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', edgecolor='k')

plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='x', s=200, label='Centroids')

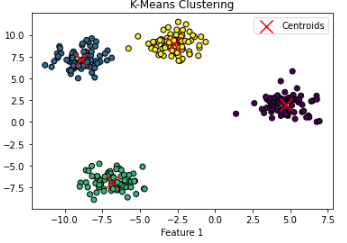
plt.title('K-Means Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()

****

**6] Classification using Random Forest.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make\_classification

from sklearn.metrics import accuracy\_score, classification\_report

# Generate synthetic dataset

X, y = make\_classification(n\_samples=500, n\_features=5, random\_state=42)

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest Classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

clf.fit(X\_train, y\_train)

# Predict on test data

y\_pred = clf.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# Print classification report

print('Classification Report:')

print(classification\_report(y\_test, y\_pred))

**Output:**

Accuracy: 0.96

Classification Report:

precision recall f1-score support

0 0.96 0.96 0.96 53

1 0.96 0.96 0.96 47

accuracy 0.96 100

**7] Regression Analysis using Linear Regression.**

import numpy as np

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

# Generate synthetic dataset

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Linear Regression model

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train, y\_train)

# Predict on test data

y\_pred = lin\_reg.predict(X\_test)

# Calculate performance metrics

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse:.2f}')

print(f'R-squared Score: {r2:.2f}')

# Plot regression line

plt.scatter(X\_test, y\_test, color='blue', label='Actual')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Predicted')

plt.title('Linear Regression Analysis')

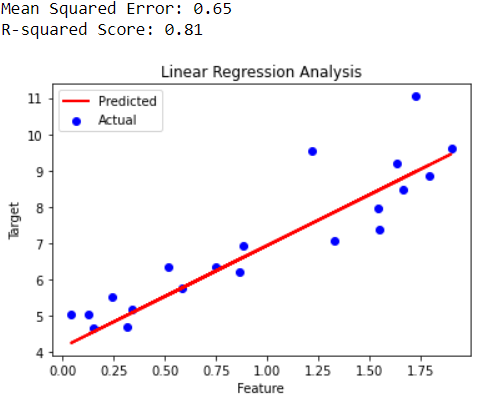
plt.xlabel('Feature')

plt.ylabel('Target')

plt.legend()

plt.show()

**Output:**



**8] Association Rule Mining using Apriori.**

from mlxtend.frequent\_patterns import apriori, association\_rules

import pandas as pd

# Sample dataset (Transaction Data)

data = {'Milk': [1, 0, 1, 1, 0],

'Bread': [1, 1, 1, 0, 1],

'Butter': [0, 1, 1, 1, 0],

'Cheese': [1, 0, 0, 1, 1]}

# Convert dataset into DataFrame

df = pd.DataFrame(data)

# Ensure data is in boolean format

df = df.astype(bool)

# Apply Apriori algorithm with lower min\_support

frequent\_itemsets = apriori(df, min\_support=0.4, use\_colnames=True)

# Generate association rules with lower confidence threshold

rules = association\_rules(frequent\_itemsets, metric='lift', min\_threshold=0.8)

# Display results

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(rules if not rules.empty else "No strong association rules found.")

**Output:**

Frequent Itemsets:

support itemsets

0 0.6 (Milk)

1 0.8 (Bread)

2 0.6 (Butter)

3 0.6 (Cheese)

4 0.4 (Bread, Milk)

5 0.4 (Butter, Milk)

6 0.4 (Cheese, Milk)

7 0.4 (Bread, Butter)

8 0.4 (Bread, Cheese)

Association Rules:

antecedents consequents antecedent support consequent support support \

0 (Bread) (Milk) 0.8 0.6 0.4

1 (Milk) (Bread) 0.6 0.8 0.4

2 (Butter) (Milk) 0.6 0.6 0.4

3 (Milk) (Butter) 0.6 0.6 0.4

4 (Cheese) (Milk) 0.6 0.6 0.4

5 (Milk) (Cheese) 0.6 0.6 0.4

6 (Bread) (Butter) 0.8 0.6 0.4

7 (Butter) (Bread) 0.6 0.8 0.4

8 (Bread) (Cheese) 0.8 0.6 0.4

9 (Cheese) (Bread) 0.6 0.8 0.4

**9] Visualize the result of the clustering and compare.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make\_blobs

# Generate data

X, \_ = make\_blobs(n\_samples=300, centers=4, random\_state=42)

# Apply K-Means

kmeans = KMeans(n\_clusters=4, random\_state=42)

labels = kmeans.fit\_predict(X)

# Plot clusters

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', alpha=0.6)

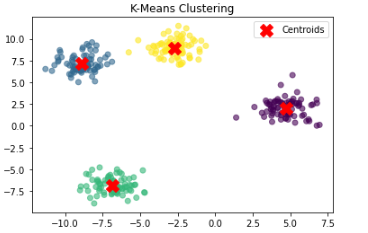
plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], c='red', marker='X', s=200, label='Centroids')

plt.title('K-Means Clustering')

plt.legend()

plt.show()

**Output:**



**10. Visualize the correlation matrix using a pseudocolor plot.**

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Generate sample data

data = np.random.rand(10, 10) # 10x10 random dataset

df = pd.DataFrame(data, columns=[f'Var{i}' for i in range(10)])

# Compute correlation matrix

corr\_matrix = df.corr()

# Plot correlation matrix using a pseudocolor plot

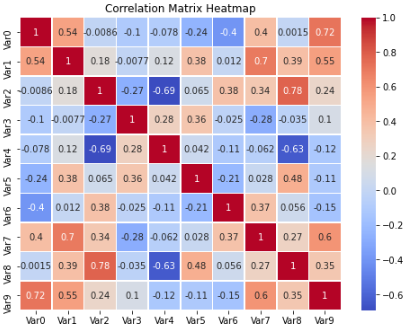
plt.figure(figsize=(8, 6))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix Heatmap')

plt.show()

**Output:**



**11] Use of degrees distribution of a network.**

import networkx as nx

import matplotlib.pyplot as plt

# Create a random graph with a fixed seed for reproducibility

G = nx.erdos\_renyi\_graph(n=50, p=0.1, seed=42)

# Compute degree distribution

degrees = [d for n, d in G.degree()]

# Plot degree distribution

plt.hist(degrees, bins=range(min(degrees), max(degrees) + 2), alpha=0.7, color='blue', edgecolor='black')

plt.xlabel('Degree')

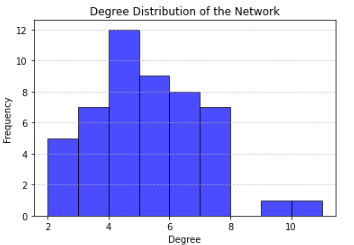
plt.ylabel('Frequency')

plt.title('Degree Distribution of the Network')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()

**Output:**



**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

# Step 1: Create a network (graph)

G = nx.erdos\_renyi\_graph(100, 0.1)

# Step 2: Calculate the degree of each node

degree\_sequence = [d for n, d in G.degree()]

# Step 3: Calculate the statistical measures

min\_degree = np.min(degree\_sequence)

max\_degree = np.max(degree\_sequence)

median\_degree = np.median(degree\_sequence)

Q1 = np.percentile(degree\_sequence, 25)

Q3 = np.percentile(degree\_sequence, 75)

# Print the statistics

print("Min degree:", min\_degree)

print("Max degree:", max\_degree)

print("Median degree:", median\_degree)

print("1st Quartile (Q1):", Q1)

print("3rd Quartile (Q3):", Q3)

# Step 4: Visualize the network, with node color based on degree

plt.figure(figsize=(12, 8))

# Choose node color based on degree

node\_color = [d for n, d in G.degree()]

# Create a spring layout for the graph

pos = nx.spring\_layout(G)

# Draw the graph with node size proportional to degree

nx.draw(G, pos, with\_labels=True, node\_size=[d \* 50 for d in degree\_sequence],

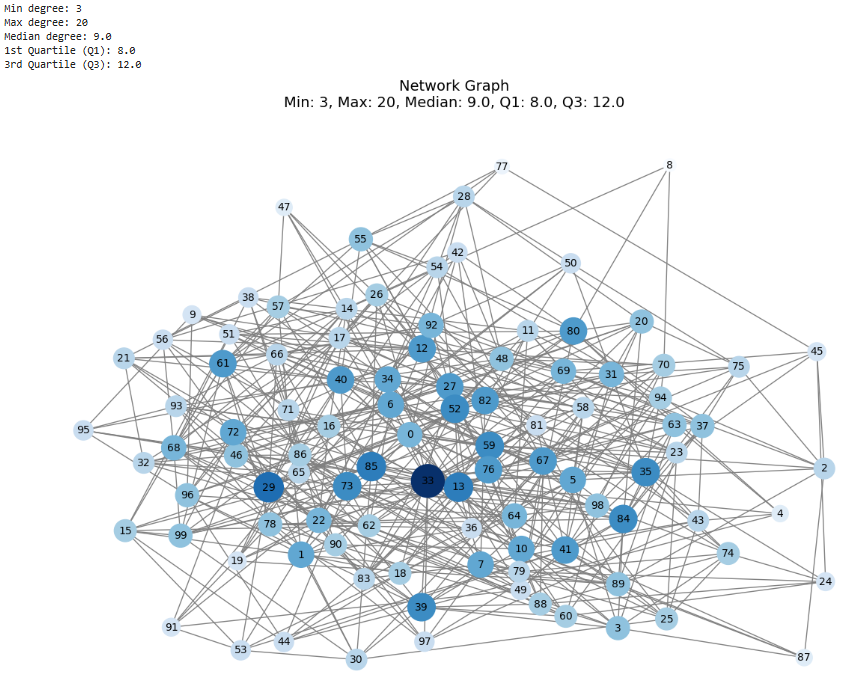
node\_color=node\_color, cmap=plt.cm.Blues, edge\_color='gray', font\_size=10)

# Add labels for the statistics on the plot

plt.title(f"Network Graph\nMin: {min\_degree}, Max: {max\_degree}, Median: {median\_degree}, Q1: {Q1}, Q3: {Q3}", fontsize=14)

plt.show()

**Output:**

****