**Pratical No:- 1**

**Pratical Name:- Calculate the mean and standard deviation.**

# Roll No:-

import numpy as np # Example data

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

# Calculate mean mean = np.mean(data)

# Calculate standard deviation std\_dev = np.std(data)

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

**Pratical No:- 2**

# Pratical Name:- Read the CSV file. Roll No:-

import csv

# Open the CSV file

with open('your\_file.csv', mode='r') as file: csv\_reader = csv.reader(file)

# Skip the header row if necessary next(csv\_reader) # If there's a header in your CSV

# Read and print each row for row in csv\_reader:

print(row) import pandas as pd

# Read the CSV file into a DataFrame df = pd.read\_csv('your\_file.csv')

# Display the first few rows print(df.head())

**Pratical No:- 3**

# Pratical Name:- Perform data filtering, and calculate aggregate statistics Roll No:-

import pandas as pd

# Load the CSV file into a DataFrame df = pd.read\_csv('your\_file.csv')

filtered\_data = df[df['Age'] > 30]

filtered\_data = df[(df['Age'] > 30) & (df['Salary'] > 50000)] mean\_salary = filtered\_data['Salary'].mean()

print(f"Mean Salary: {mean\_salary}")

sum\_salary = filtered\_data['Salary'].sum() print(f"Total Salary: {sum\_salary}")

row\_count = filtered\_data.shape[0] print(f"Number of rows: {row\_count}")

# Group by 'Department' and calculate the mean of numeric columns grouped\_data = df.groupby('Department').mean()

print(grouped\_data)

aggregate\_stats = df.groupby('Department').agg({ 'Salary': ['mean', 'sum'],

'Age': 'mean'

})

print(aggregate\_stats) import pandas as pd # Load the CSV file

df = pd.read\_csv('your\_file.csv')

# Filter rows where 'Age' > 30 and 'Salary' > 50000 filtered\_data = df[(df['Age'] > 30) & (df['Salary'] > 50000)]

# Calculate aggregate statistics

mean\_salary = filtered\_data['Salary'].mean() total\_salary = filtered\_data['Salary'].sum() row\_count = filtered\_data.shape[0]

print(f"Mean Salary: {mean\_salary}") print(f"Total Salary: {total\_salary}") print(f"Number of rows: {row\_count}")

# Group by 'Department' and calculate mean salary and age grouped\_data = df.groupby('Department').agg({

'Salary': ['mean', 'sum'], 'Age': 'mean'

})

print(grouped\_data)

**Pratical No:- 4**

# Pratical Name:- Calculate total sales by month. Roll No:-

import pandas as pd

# Load the CSV file into a DataFrame df = pd.read\_csv('your\_sales\_data.csv')

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

# Create a new column for the month-year combination df['YearMonth'] = df['Date'].dt.to\_period('M')

# Group by 'YearMonth' and calculate the total sales

total\_sales\_by\_month = df.groupby('YearMonth')['Sales'].sum().reset\_index()

# Print the results print(total\_sales\_by\_month)

import pandas as pd

# Load the CSV file into a DataFrame df = pd.read\_csv('your\_sales\_data.csv')

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

# Create a new column for the month-year combination df['YearMonth'] = df['Date'].dt.to\_period('M')

# Group by 'YearMonth' and calculate the total sales

total\_sales\_by\_month = df.groupby('YearMonth')['Sales'].sum().reset\_index()

# Print the total sales by month print(total\_sales\_by\_month)

**Pratical No:- 5**

# Pratical Name:- Implement the Clustering using K-means. Roll No:-

# Import necessary libraries import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans from sklearn.datasets import make\_blobs

# Create synthetic data for clustering

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

# Initialize KMeans with the number of clusters you want kmeans = KMeans(n\_clusters=4)

# Fit the model to the data kmeans.fit(X)

# Get the cluster centers

centroids = kmeans.cluster\_centers\_

# Get the predicted labels (clusters) for each point labels = kmeans.labels\_

# Visualize the data points and centroids

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50) plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', c='red', s=200, label='Centroids')

plt.title('K-Means Clustering') plt.legend()

plt.show()

**Pratical No:- 6**

# Pratical Name:- Classification using Random Forest. Roll No:-

import numpy as np import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.datasets import load\_iris

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

# Load a sample dataset (Iris dataset for example) data = load\_iris()

X = data.data # Features (4 attributes for each flower) y = data.target # Labels (target classes: 0, 1, 2)

# Split the data into training and test sets (80% train, 20% test) X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the Random Forest Classifier with 100 trees (default) rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the model on the training data rf\_classifier.fit(X\_train, y\_train)

# Predict on the test data

y\_pred = rf\_classifier.predict(X\_test)

# Evaluate the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred)) print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Optionally, you can visualize feature importance: import matplotlib.pyplot as plt

# Get feature importance

importances = rf\_classifier.feature\_importances\_

# Plot feature importance plt.figure(figsize=(8, 6)) plt.barh(data.feature\_names, importances)

plt.xlabel("Feature Importance") plt.title("Random Forest Feature Importance") plt.show()

**Pratical No:- 7**

# Pratical Name:- Regression Analysis using Linear Regression. Roll No:-

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

# Generate some synthetic data for regression

# Example: Predicting house prices based on square footage (just as an example)

np.random.seed(42)

X = 2.5 \* np.random.randn(1000) + 25 # Feature (square footage)

y = 0.5 \* X + np.random.randn(1000) \* 5 + 15 # Target (house price)

# Reshape X to be a 2D array as required by sklearn X = X.reshape(-1, 1)

# Split the data into training and test sets (80% train, 20% test) X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the Linear Regression model lin\_reg = LinearRegression()

# Train the model lin\_reg.fit(X\_train, y\_train)

# Make predictions on the test set y\_pred = lin\_reg.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred) # Mean Squared Error r2 = r2\_score(y\_test, y\_pred) # R-squared value

# Print the evaluation metrics print(f"Mean Squared Error: {mse}") print(f"R-squared: {r2}")

# Plot the results plt.figure(figsize=(8, 6))

# Scatter plot of the actual data points (test data) plt.scatter(X\_test, y\_test, color='blue', label='Actual data')

# Plot the regression line (predicted values)

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

plt.title('Linear Regression: House Price Prediction') plt.xlabel('Square Footage')

plt.ylabel('Price') plt.legend() plt.show()

**Pratical No:- 8**

# Pratical Name:- Association Rule Mining using Apriori. Roll No:-

import pandas as pd

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

data = {

'Milk': [1, 1, 1, 0, 1],

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

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

'Beer': [0, 0, 1, 1, 1],

'Diapers': [1, 1, 0, 1, 1],

}

df = pd.DataFrame(data)

frequent\_itemsets = apriori(df, min\_support=0.6, use\_colnames=True) # Generate association rules with a minimum lift of 1.0 (default is 1.0)

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.0)

# Display the rules

print("Frequent Itemsets:\n", frequent\_itemsets) print("\nAssociation Rules:\n", rules)

# Optionally, you can display the rules in a more readable format: print("\nFormatted Association Rules:")

for index, rule in rules.iterrows():

print(f"Rule: {', '.join(list(rule['antecedents']))} -> {', '.join(list(rule['consequents']))}")

print(f"Support: {rule['support']}, Confidence: {rule['confidence']}, Lift:

{rule['lift']}") print("="\*50)

**Pratical No:- 9**

# Pratical Name:- Visualize the result of the clustering and compare. Roll No:-

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans from sklearn.datasets import make\_blobs

from sklearn.metrics import adjusted\_rand\_score

# Generate synthetic data for clustering (3 clusters)

X, y = make\_blobs(n\_samples=300, centers=3, random\_state=42)

# Apply KMeans clustering kmeans = KMeans(n\_clusters=3) kmeans.fit(X)

# Get the predicted labels (clusters) labels = kmeans.labels\_

# Get the cluster centers

centroids = kmeans.cluster\_centers\_

# Compare the clustering results with the true labels using Adjusted Rand Index ari = adjusted\_rand\_score(y, labels)

print(f"Adjusted Rand Index (ARI) between true labels and predicted clusters:

{ari}")

# Visualize the clustering results (predicted clusters) plt.figure(figsize=(12, 6))

# Plot the clustered data points plt.subplot(1, 2, 1)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', s=50) plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', c='red', s=200, label='Centroids')

plt.title('KMeans Clustering (Predicted Labels)') plt.xlabel('Feature 1')

plt.ylabel('Feature 2') plt.legend()

# Visualize the true labels (if available) plt.subplot(1, 2, 2)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', marker='o', s=50) plt.title('True Labels')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.tight\_layout() plt.show()

**Pratical No:- 10**

# Pratical Name:- Visualize the correlation matrix using a pseudocolor plot Roll No:-

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

# Generate some sample data

data = np.random.rand(10, 5) # 10 rows, 5 columns

df = pd.DataFrame(data, columns=[f'Feature {i+1}' for i in range(5)])

# Compute the correlation matrix corr\_matrix = df.corr()

# Plotting the correlation matrix using a heatmap plt.figure(figsize=(8, 6))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Matrix Heatmap') plt.show()

**Pratical No:- 11**

# Pratical Name:- Use of degrees distribution of a network. Roll No:-

import networkx as nx

import matplotlib.pyplot as plt import numpy as np

# Create a sample graph (e.g., Erdos-Renyi random graph)

G = nx.erdos\_renyi\_graph(n=100, p=0.05) # 100 nodes, probability of edge creation = 0.05

# Get the degree of each node

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

# Compute the frequency of each degree degree\_count = np.bincount(degrees)

# Plot the degree distribution plt.figure(figsize=(8, 6))

plt.bar(range(len(degree\_count)), degree\_count, width=0.8, color='b', alpha=0.7)

plt.xlabel('Degree') plt.ylabel('Frequency')

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

# Create a Barabási–Albert scale-free network

G = nx.barabasi\_albert\_graph(n=100, m=2) # 100 nodes, each new node connects to 2 existing nodes

# Get the degree of each node and calculate the degree distribution as before degrees = [G.degree(n) for n in G.nodes()]

degree\_count = np.bincount(degrees) # Plot the degree distribution plt.figure(figsize=(8, 6))

plt.bar(range(len(degree\_count)), degree\_count, width=0.8, color='r', alpha=0.7) plt.xlabel('Degree')

plt.ylabel('Frequency')

plt.title('Degree Distribution of Barabási–Albert Network') plt.show()

**Pratical No:- 12**

# Pratical Name:- Graph visualization of a network using maximum, minimum, median, first quartile and third quartile

**Roll No:-**

import networkx as nx

import matplotlib.pyplot as plt import numpy as np

# Create a random graph (e.g., Erdos-Renyi random graph)

G = nx.erdos\_renyi\_graph(n=100, p=0.05) # 100 nodes, probability 0.05

# Compute the degree of each node degrees = [G.degree(n) for n in G.nodes()]

# Compute degree statistics degree\_min = np.min(degrees) degree\_max = np.max(degrees) degree\_median = np.median(degrees) degree\_q1 = np.percentile(degrees, 25) degree\_q3 = np.percentile(degrees, 75)

# Print degree statistics

print(f"Minimum degree: {degree\_min}") print(f"Maximum degree: {degree\_max}") print(f"Median degree: {degree\_median}") print(f"First Quartile (Q1): {degree\_q1}") print(f"Third Quartile (Q3): {degree\_q3}")

# Classify nodes based on degree node\_colors = []

for node in G.nodes(): degree = G.degree(node) if degree == degree\_min:

node\_colors.append('blue') # Min degree elif degree == degree\_max:

node\_colors.append('red') # Max degree elif degree == degree\_median:

node\_colors.append('green') # Median degree elif degree <= degree\_q1:

node\_colors.append('purple') # First Quartile (Q1) elif degree >= degree\_q3:

node\_colors.append('orange') # Third Quartile (Q3) else:

node\_colors.append('gray') # Other degrees

# Visualize the network plt.figure(figsize=(10, 8))

pos = nx.spring\_layout(G) # Layout for node positioning

# Draw the graph with colored nodes based on degree

nx.draw(G, pos, node\_size=50, node\_color=node\_colors, with\_labels=False, edge\_color='lightgray')

# Add a title

plt.title('Network Visualization Based on Degree Statistics') plt.show()