**Practical No. 1:-** Calculate the mean and standard deviation.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

import numpy as np

# Sample data

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

# Calculate mean

mean\_value = np.mean(data)

# Calculate standard deviation

std\_dev = np.std(data, ddof=1) # Set ddof=0 for population standard deviation

# Display results

print(f"Mean: {mean\_value}")

print(f"Standard Deviation: {std\_dev}")

**Practical No. 2:-** Read CSV File

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

import pandas as pd

# Load the data from a CSV file

file\_path = 'sales\_data.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)

**Practical No. 3:-** Perform data filtering, and calculate aggregate statistics.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

We have a dataset (sales\_data.csv) with the following columns:

| **Date** | **Region** | **Product** | **Sales** |
| --- | --- | --- | --- |
| 2023-01-05 | North | A | 100 |
| 2023-01-12 | South | B | 200 |
| 2023-02-01 | North | A | 150 |
| 2023-02-14 | East | C | 300 |
| 2023-02-21 | West | B | 250 |

**Python Code**

import pandas as pd

# Load the data from a CSV file

file\_path = 'sales\_data.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)

**Explanation**

1. **Loading Data**: Uses pd.read\_csv() to read the dataset.
2. **Data Filtering**: sales\_data[sales\_data['Sales'] > 150] filters rows where sales are greater than 150.
3. **Grouping and Aggregating by Region**:  
   groupby('Region')['Sales'].agg(['sum', 'mean']) computes total and average sales per region.
4. **Grouping and Aggregating by Product**:  
   groupby('Product')['Sales'].agg(['sum', 'count']) computes total sales and transaction count per product.

**Output Example**

First few rows of the dataset:

Date Region Product Sales

0 2023-01-05 North A 100

1 2023-01-12 South B 200

Filtered Data (Sales > 150):

Date Region Product Sales

1 2023-01-12 South B 200

3 2023-02-14 East C 300

4 2023-02-21 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

**Practical No. 4:-** Calculate total sales by month.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

**Python Code**

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)

**Explanation:**

1. **Convert Date to Datetime**: Ensures the Date column is recognized as a datetime object.
2. **Extract Year and Month**: Uses .dt.to\_period('M') to extract the year-month period.
3. **Group and Aggregate**: Groups the data by YearMonth and calculates the sum of sales for each month.

**Practical No. 5 :-** Implement the Clustering using K-means.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

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()

**Practical No. 6 :-** Classification using Random Forest.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

**# Import necessary libraries**

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

**# Load the Iris dataset**

data = load\_iris()

X = data.data # Features

y = data.target # Target variable (class labels)

**# Split the data into training and testing sets (70% train, 30% test)**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

**# Initialize the Random Forest Classifier**

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

**# Train the model**

rf\_classifier.fit(X\_train, y\_train)

**# Predict on the test set**

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

**# Evaluate the model**

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

print(f"Accuracy: {accuracy \* 100:.2f}%")

**# Detailed classification report (Precision, Recall, F1-score for each class)**

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

**# Feature Importance: Display the importance of each feature**

print("Feature Importance:")

for feature, importance in zip(data.feature\_names, rf\_classifier.feature\_importances\_):

print(f"{feature}: {importance:.4f}")

**Practical No. 7 :-** Regression Analysis using Linear Regression.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

# Step 1: Import libraries  
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  
  
# Step 2: Load a sample dataset (for example, 'salary vs experience')  
# You can replace this with your own dataset  
# For this example, let's create a simple dataset  
  
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)  
  
# Step 3: Prepare the data  
# X = input features, y = target variable  
X = df[['YearsExperience']] # Independent variable (Features)  
y = df['Salary'] # Dependent variable (Target)  
  
# Step 4: Split the data 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)  
  
# Step 5: Create the model and train it  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Step 6: Predict the target variable using the test set  
y\_pred = model.predict(X\_test)  
  
# Step 7: Evaluate the model  
# Mean Squared Error  
mse = mean\_squared\_error(y\_test, y\_pred)  
print(f"Mean Squared Error: {mse}")  
  
# R-squared (Coefficient of determination)  
r2 = r2\_score(y\_test, y\_pred)  
print(f"R-squared: {r2}")  
  
# Step 8: Plotting the results  
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()  
  
# Step 9: Make predictions (Example)  
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}")

**Practical No. 8 :-** Association Rule Mining using Apriori.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

**# Import necessary libraries**

import pandas as pd

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

**# Sample dataset (You can load your own dataset as well)**

**# Here, each row represents a transaction with items purchased (1 if item is bought, 0 if not)**

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)

**# Apply the Apriori algorithm to find frequent itemsets with a minimum support of 0.6**

frequent\_itemsets = apriori(df, min\_support=0.6, use\_colnames=True)

**# Generate association rules with a minimum confidence of 0.7**

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

**# Display the frequent itemsets**

print("Frequent Itemsets:")

print(frequent\_itemsets)

**# Display the association rules**

print("\nAssociation Rules:")

print(rules)

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

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans, AgglomerativeClustering

from sklearn.metrics import silhouette\_score

**# Generate synthetic data**

X, y\_true = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.6, random\_state=42)

**# Apply K-Means clustering**

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

kmeans\_labels = kmeans.fit\_predict(X)

**# Apply Agglomerative Clustering**

agg = AgglomerativeClustering(n\_clusters=4)

agg\_labels = agg.fit\_predict(X)

**# Plot the original data and clustering results**

fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharex=True, sharey=True)

**# Original dataset**

axes[0].scatter(X[:, 0], X[:, 1], c=y\_true, cmap='viridis', s=30, edgecolor='k')

axes[0].set\_title("Original Data (Ground Truth)")

**# K-Means Clustering**

axes[1].scatter(X[:, 0], X[:, 1], c=kmeans\_labels, cmap='viridis', s=30, edgecolor='k')

axes[1].set\_title("K-Means Clustering")

**# Agglomerative Clustering**

axes[2].scatter(X[:, 0], X[:, 1], c=agg\_labels, cmap='viridis', s=30, edgecolor='k')

axes[2].set\_title("Agglomerative Clustering")

for ax in axes:

ax.set\_xlabel("Feature 1")

ax.set\_ylabel("Feature 2")

plt.tight\_layout()

plt.show()

**# Compare the quality using Silhouette Scores**

kmeans\_silhouette = silhouette\_score(X, kmeans\_labels)

agg\_silhouette = silhouette\_score(X, agg\_labels)

print(f"Silhouette Score for K-Means: {kmeans\_silhouette:.2f}")

print(f"Silhouette Score for Agglomerative Clustering: {agg\_silhouette:.2f}")

**Practical No. 10:-** Visualize the result of the clustering and compare.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

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()

**Practical No. 11:-** Use of degrees distribution of a network.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

import networkx as nx  
import matplotlib.pyplot as plt  
  
# Step 1: Create a sample graph (e.g., Erdos-Renyi graph)  
# n = number of nodes, p = probability of edge creation  
G = nx.erdos\_renyi\_graph(n=100, p=0.05) # A random graph with 100 nodes and a 5% chance of edge creation  
  
# Step 2: Get the degree of each node  
degree\_sequence = [G.degree(node) for node in G.nodes()]  
  
# Step 3: Plot the degree distribution  
# Count how many nodes have each degree  
degree\_count = {}  
for degree in degree\_sequence:  
 degree\_count[degree] = degree\_count.get(degree, 0) + 1  
  
# Prepare data for plotting  
degrees = list(degree\_count.keys()) # List of degrees  
counts = list(degree\_count.values()) # Corresponding counts of nodes with each degree  
  
# Step 4: Create a bar plot of the degree distribution  
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()  
  
# Step 5: Plot degree distribution in a log-log scale (for power-law distribution observation)  
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()

**Practical No. 12:-** Graph visualization of a network using maximum, minimum, median, first quartile and third quartile.

**Name**: Mohit Ravindra Chaudhari

**Batch:** B1 **Date :**

import networkx as nx

import matplotlib.pyplot as plt

import numpy as np

# Step 1: Create a sample graph

G = nx.erdos\_renyi\_graph(n=100, p=0.1) # A random graph with 100 nodes and a 10% chance of edge creation

# Step 2: Get the degree of each node

degree\_sequence = [G.degree(node) for node in G.nodes()]

# Step 3: Calculate statistical measures

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}")

# Step 4: Assign colors based on degree relative to the quartiles and other statistics

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

# Step 5: Visualize the network with degree-based coloring

plt.figure(figsize=(10, 8))

# Draw the graph with the node colors based on degree

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')

# Add a legend to the plot

import matplotlib.lines as mlines

# Create legend manually

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")

# Show the plot

plt.title("Network Visualization with Degree Statistics Coloring")

plt.show()