B. K. Birla College (Autonomous), Kalyan
(Department of Information Technology)

CERTIFICATE

This is to certify that Mrs. **Ruchita S. Kamble**, Roll No. **15** ,

Exam Seat No. **23258715** has satisfactorily completed the Practical in **BIG DATA ANALYTICS** of Semester **II** for the fulfilment of M.SC.- IT(Cloud Computing) for the year **2024-2025.**

Signature of Examiners

____

Date:- 03 June 2025

INDEX

Practical 1: Exploring Big data Characteristics with Real Datasets

#load the dataset

[1] import pandas as pd
df=pd.read_csv("OnlineRetail.csv", encoding='ISO-8859-1') #encoding- to handle special characters in the dataset.
df.head()

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

[2] df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  object
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

 # Volume -- Size of Data
[3] print("Shape:", df.shape)
#shows the size: (rows, columns) → indicates how much data you have.

Shape: (541909, 8)

[4] print("Memoery Usage (MB):",df.memory_usage(deep=True).sum()/1024**2)
#calculates how much memory the DataFrame is consuming, in megabytes (MB)

Memoery Usage (MB): 173.12356662750244

#Velocity --Speed of Data Generation & Processing

[ ] df['InvoiceDate']=pd.to_datetime(df['InvoiceDate'],dayfirst=True, errors='coerce')
#Converts the InvoiceDate column from string to datetime.
#dayfirst=True handles dates in DD/MM/YYYY format.
#errors='coerce' converts invalid dates to NaT (missing date).

[5] df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  232959 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

#Variety --Different Data Types and Sources
[6] print("Data Type:\n", df.dtypes)
#df.dtypes shows the data type of each column (e.g., object, float64, datetime64)

```
Data Type:
 InvoiceNo              object
StockCode              object
Description            object
Quantity                int64
InvoiceDate    datetime64[ns]
UnitPrice             float64
CustomerID            float64
Country                object
dtype: object
```

[7] print("Variety Count:\n", df.dtypes.value_counts())
#.value_counts() on dtypes tells how many columns belong to each type.

```
Variety Count:
 object            4
float64           2
int64             1
datetime64[ns]    1
Name: count, dtype: int64
```

#Veracity-- Data Quality and Trustworthiness
[8] print("Missing Values:\n", df.isnull().sum())
#isnull().sum() counts how many missing values are in each column.

Missing Values:
 InvoiceNo 0
StockCode 0
Description 1454
Quantity 0
InvoiceDate 308950
UnitPrice 0
CustomerID 135080
Country 0
dtype: int64

[9] print("Duplicate Records:\n", df.duplicated().sum())
#duplicated().sum() counts how many duplicate rows exist.

Duplicate Records:
 5269

Cleaning Steps
#Removes rows where CustomerID or InvoiceDate is missing.
#This improves data quality (veracity).
[10] df_clean= df.dropna(subset=['CustomerID'])

[11] print("After cleaning:", df_clean.shape)

After cleaning: (406829, 8)

[12] print("Missing Values:\n", df_clean.isnull().sum())

Missing Values:
 InvoiceNo 0
StockCode 0
Description 0
Quantity 0
InvoiceDate 234047
UnitPrice 0
CustomerID 0
Country 0
dtype: int64

[13] df_clean1= df.dropna(subset=['InvoiceDate'])

[14] print("Missing Values:\n", df_clean1.isnull().sum())

Missing Values:
 InvoiceNo 0
StockCode 0
Description 658
Quantity 0
InvoiceDate 0
UnitPrice 0
CustomerID 60177
Country 0

dtype: int64

#Calculate Revenue per Transaction
[15] df['TotalPrice']=df['Quantity'] * df['UnitPrice']
#Creates a new column TotalPrice, representing revenue per line item.

[16] print("Top Products:\n", df['Description'].value_counts().head())
#Finds the most frequently sold products by counting values in the Description column.

```
Top Products:
 Description
WHITE HANGING HEART T-LIGHT HOLDER    2369
REGENCY CAKESTAND 3 TIER              2200
JUMBO BAG RED RETROSPOT               2159
PARTY BUNTING                         1727
LUNCH BAG RED RETROSPOT               1638
Name: count, dtype: int64
```

[17] print("Top Countries by Revenue:\n",
df.groupby('Country')['TotalPrice'].sum().sort_values(ascending=False).head())

```
Top Countries by Revenue:
 Country
United Kingdom    8187806.364
Netherlands        284661.540
EIRE               263276.820
Germany            221698.210
France             197403.900
Name: TotalPrice, dtype: float64
```

#Groups the data by Country.
#Sums the TotalPrice for each country.
#Sorts countries by revenue in descending order.
#Shows the top 5 countries with the highest sales revenue.

[18] df['TotalPrice'] = df['Quantity'] * df['UnitPrice']
[19] print("Top Products:\n", df['Description'].value_counts().head())

```
Top Products:
 Description
WHITE HANGING HEART T-LIGHT HOLDER    2369
REGENCY CAKESTAND 3 TIER              2200
JUMBO BAG RED RETROSPOT               2159
PARTY BUNTING                         1727
LUNCH BAG RED RETROSPOT               1638
Name: count, dtype: int64
```

Practical 2: Comparative Analysis of Traditional BI vs Big Data Tools

[1] #comapre analysis and data visulization
import pandas as pd
df = pd.read_csv("OnlineRetail.csv", encoding='ISO-8859-1')

[2] df.head()

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

[3]df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'],dayfirst=True,errors='coerce')

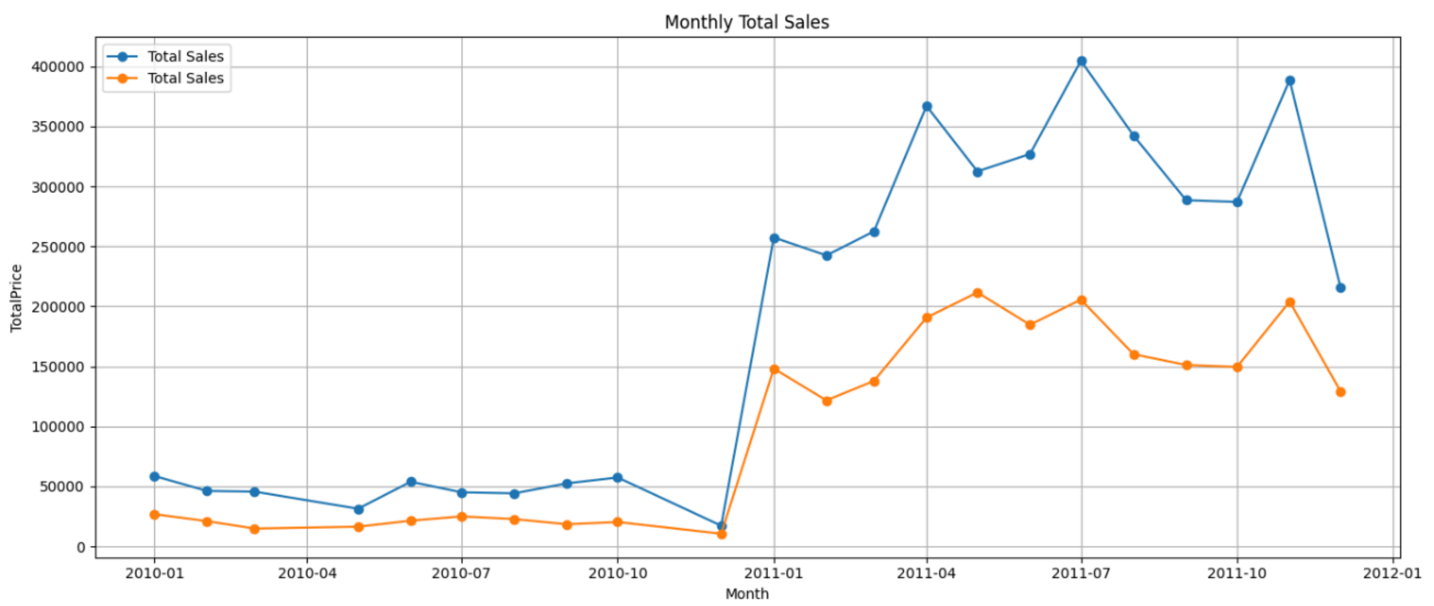
[4] # compute total price
df['TotalPrice'] = df['Quantity'] * df['UnitPrice']

[5]# group by the month of the InvoiceDate
```
monthly_sales = df.groupby(df['InvoiceDate'].dt.to_period('M')).agg({
    'Quantity': 'sum',
    'TotalPrice': 'sum'
}).reset_index()

print(monthly_sales.head())
```

```
InvoiceDate  Quantity  TotalPrice
0    2010-01     26814    58635.56
1    2010-02     21023    46207.28
2    2010-03     14830    45620.46
3    2010-05     16395    31383.95
4    2010-06     21419    53860.18
```

[6] # 1. Reset index so 'InvoiceDate' is a column
```
monthly_sales = monthly_sales.reset_index()

# 2. Convert the PeriodIndex column to Timestamps
monthly_sales['InvoiceDate'] = monthly_sales['InvoiceDate'].dt.to_timestamp()

# 3. Plot
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(14,6))
plt.plot(
    monthly_sales['InvoiceDate'],
    monthly_sales['TotalPrice'],
    marker='o',
    label='Total Sales'
)
plt.plot(
    monthly_sales['InvoiceDate'],
    monthly_sales['Quantity'],
    marker='o',
    label='Total Sales'
)
plt.title('Monthly Total Sales')
plt.xlabel('Month')
plt.ylabel('TotalPrice')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



Practical 3: Implement K-Means and DBSCAN Clustering

```
[1] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans,DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
```

```
[2] df = pd.read_csv('Mall_Customers.csv')
print(df.head())
```

```
CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0          1    Male   19                  15                      39
1          2    Male   21                  15                      81
2          3  Female   20                  16                       6
3          4  Female   23                  16                      77
4          5  Female   31                  17                      40
```

```
[3] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
[4] df.describe()
```

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

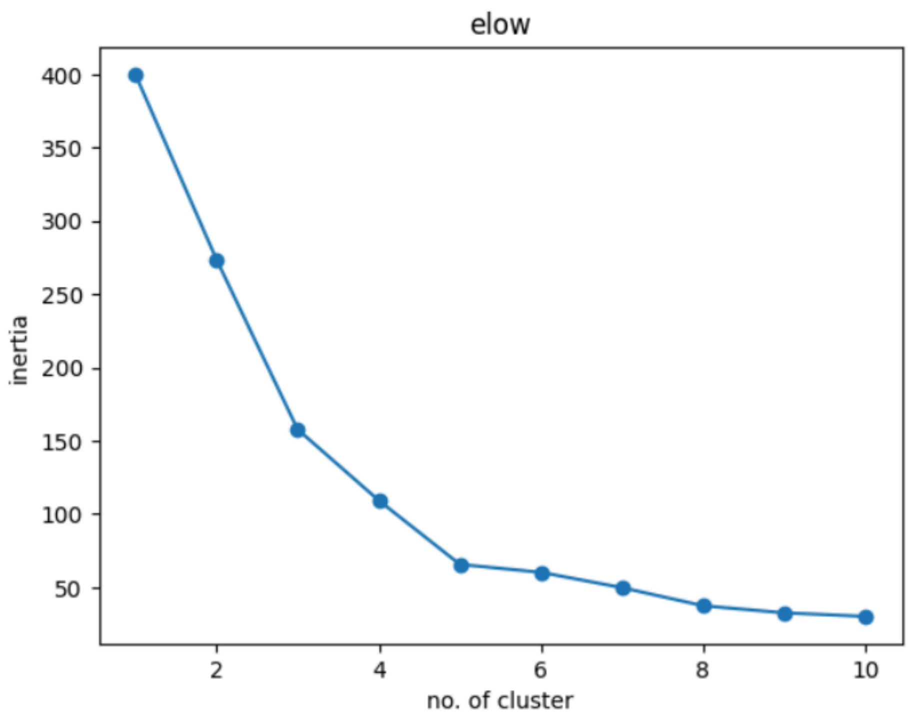
[5] df.isnull().sum()

```
CustomerID              0
Gender                  0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

[6] X = df[['Annual Income (k$)', 'Spending Score (1-100)']]
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

[7] inertia =[]

```
for k in range (1,11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.plot(range(1,11), inertia, marker='o')
plt.xlabel('no. of cluster')
plt.ylabel('inertia')
plt.title('elow')
plt.show()
```

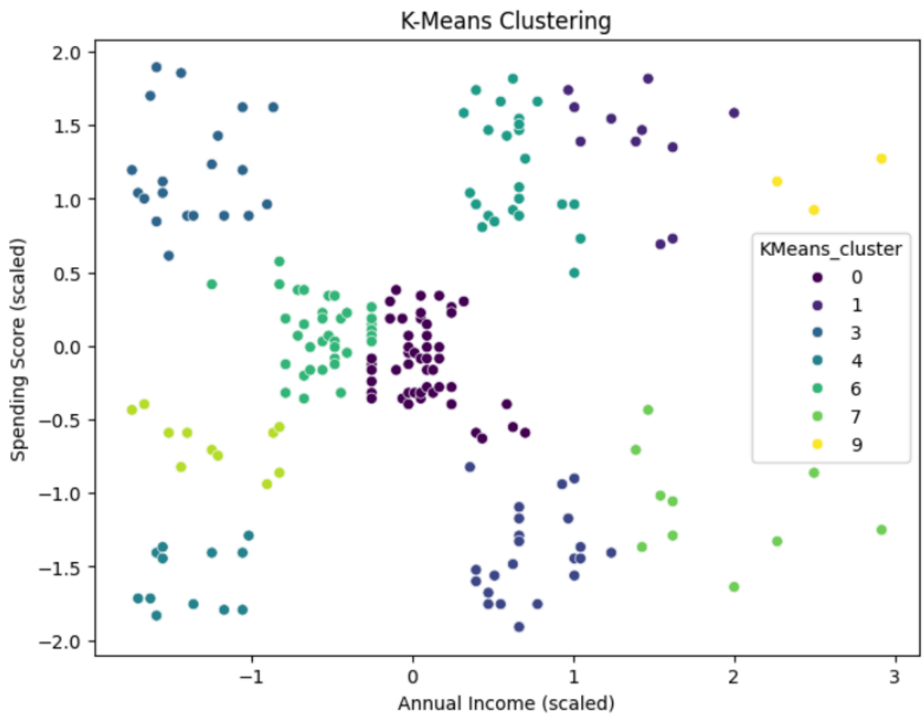


[8] from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd

[9] # Assuming X_scaled is a scaled version of your input data (e.g., with StandardScaler or MinMaxScaler)
and k is defined

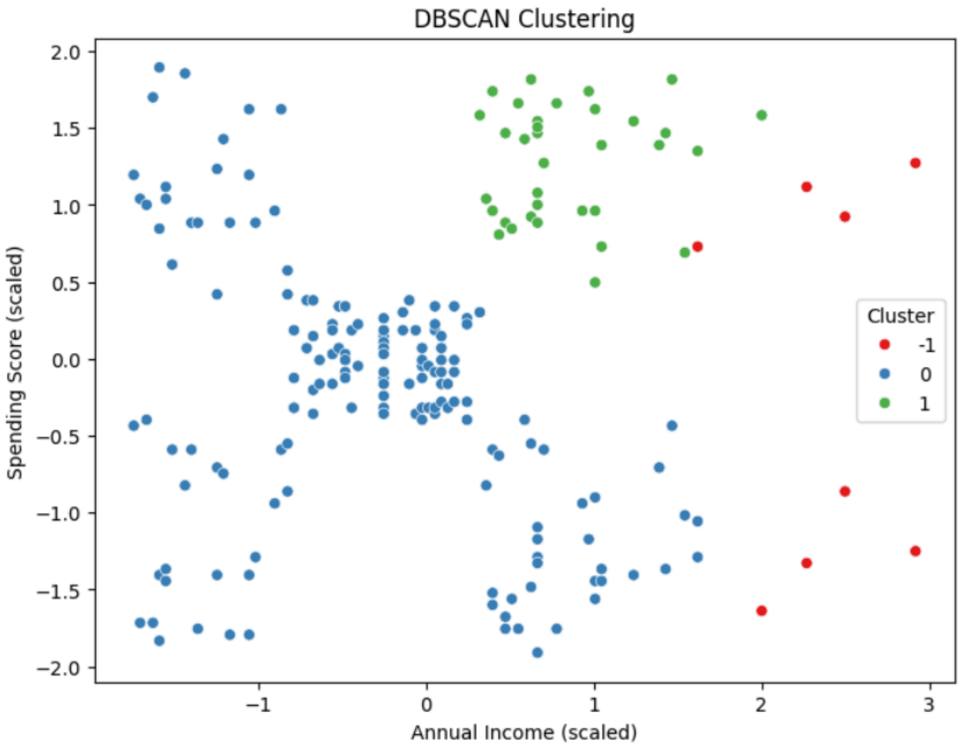
```
kmeans = KMeans(n_clusters=k, random_state=42)
df['KMeans_cluster'] = kmeans.fit_predict(X_scaled)
plt.figure(figsize=(8,6))
# Correct indexing for x and y: use column indices or names
sns.scatterplot(
    x=X_scaled[:, 0],  # 1st feature (e.g., Annual Income)
    y=X_scaled[:, 1],  # 2nd feature (e.g., Spending Score)
    hue=df['KMeans_cluster'],
    palette='viridis'
)

plt.xlabel('Annual Income (scaled)')
plt.ylabel('Spending Score (scaled)')
plt.title('K-Means Clustering')
plt.show()
```



[10] from sklearn.cluster import DBSCAN

```
# Create and fit the DBSCAN model
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Corrected: min_samples, not min_sample
df['DBSCAN_Cluster'] = dbscan.fit_predict(X_scaled)


# In[27]:
plt.figure(figsize=(8,6))
```

```
sns.scatterplot(
    x=X_scaled[:, 0],
    y=X_scaled[:, 1],
    hue=df['DBSCAN_Cluster'],
    palette='Set1'
)

plt.xlabel('Annual Income (scaled)')
plt.ylabel('Spending Score (scaled)')
plt.title('DBSCAN Clustering')
plt.legend(title='Cluster')
plt.show()
```



Practical 4: Association Rules Mining using Apriori Algorithm

[1] get_ipython().system('pip install mlxtend')

[2] import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

[3]transactions=[
    ['Milk','Bread','Butter'],
    ['Bread','Butter'],
    ['Milk','Diaper','Beer','Egg'],
    ['Bread','Butter','Diaper'],
    ['Milk','Diaper','Bread','Butter'],
    ['Beer','Diaper'],
    ['Milk','Butter','Diaper','Egg'],
]

[4] te=TransactionEncoder()
te_ary =te.fit(transactions).transform(transactions)
df_onehot =pd.DataFrame(te_ary, columns=te.columns_)
print("One-hot encoded dataset:")
display(df_onehot)

```
One-hot encoded dataset:
```

	Beer	Bread	Butter	Diaper	Egg	Milk
0	False	True	True	False	False	True
1	False	True	True	False	False	False
2	True	False	False	True	True	True
3	False	True	True	True	False	False
4	False	True	True	True	False	True
5	True	False	False	True	False	False
6	False	False	True	True	True	True

[5] frequent_itemset =apriori(df_onehot, min_support=0.4, use_colnames=True)
print("frequent_itemset")
display(frequent_itemset)

```
frequent_itemset
```

	support	itemsets
0	0.571429	(Bread)
1	0.714286	(Butter)
2	0.714286	(Diaper)
3	0.571429	(Milk)
4	0.571429	(Butter, Bread)
5	0.428571	(Butter, Diaper)
6	0.428571	(Butter, Milk)
7	0.428571	(Milk, Diaper)

```
[6] rules =association_rules(frequent_itemset, metric="confidence", min_threshold=0.7)
print(association_rules)
display(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

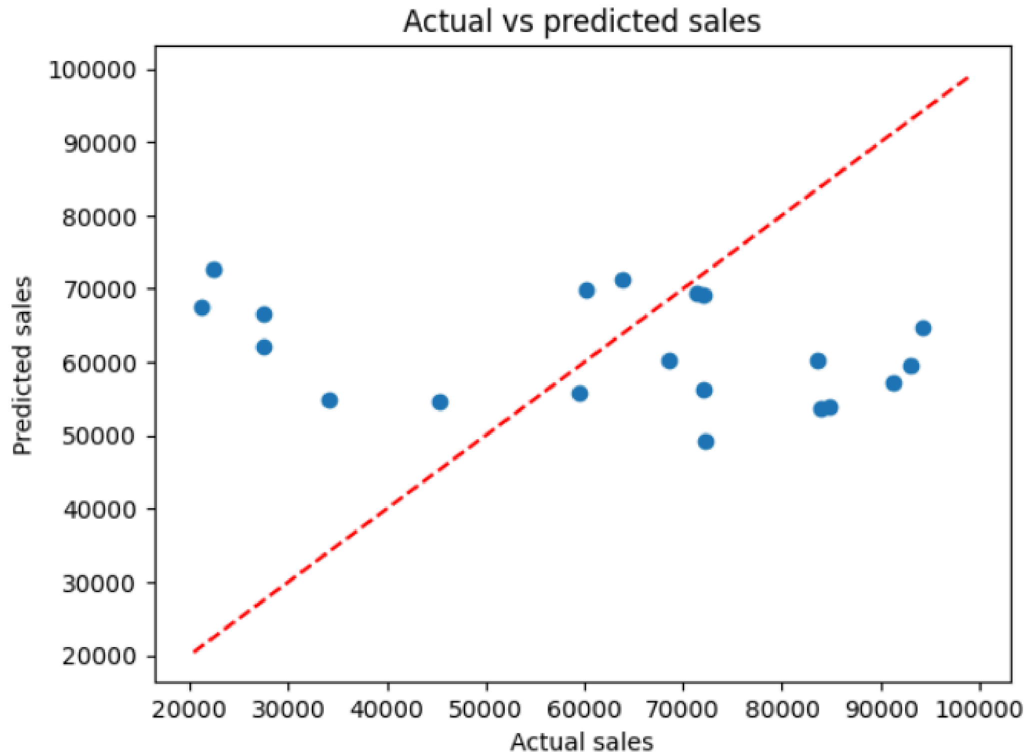
<function association_rules at 0x0000017170A9EC00>

	antecedents	consequents	support	confidence	lift
0	(Butter)	(Bread)	0.571429	0.80	1.40
1	(Bread)	(Butter)	0.571429	1.00	1.40
2	(Milk)	(Butter)	0.428571	0.75	1.05
3	(Milk)	(Diaper)	0.428571	0.75	1.05

Practical 5: Regression Analysis And Evaluation

```
[1] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

[2] np.random.seed(42)
data= pd.DataFrame({
   'Advertising_spend': np.random.uniform(1000, 5000, 100),
   'Store_size': np.random.uniform(500, 1500, 100),
   'Sales':  np.random.uniform(20000, 100000, 100),
})
x=data[['Advertising_spend','Store_size']]
y=data['Sales']

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)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))


r2= r2_score(y_test, y_pred)

print(f"Linear Regression -RMSE: {rmse:.2f}")
print(f"Linear Regression -R^2 score: {r2:.2f}")

plt.scatter(y_test, y_pred)

plt.xlabel("Actual sales")
plt.ylabel("Predicted sales")
plt.title("Actual vs predicted sales")
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.show()
```

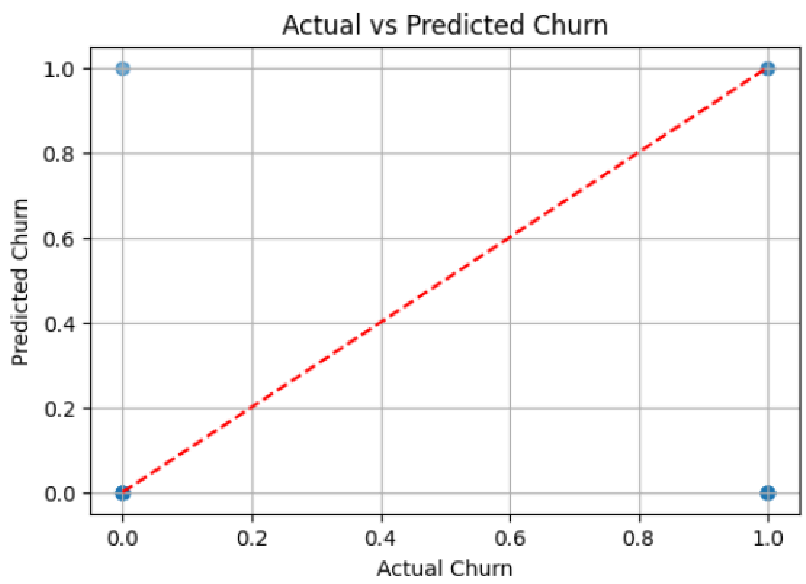
```
Linear Regression -RMSE: 26922.70
Linear Regression -R^2 score: -0.27
```




```
[3] import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generating synthetic churn data
churn_data = pd.DataFrame({
    'Monthly_charges': np.random.uniform(20, 120, 100),
    'Tenure': np.random.randint(1, 72, 100),
    'Churn': np.random.choice([0, 1], 100)  # 0 = no churn, 1 = churn
})

# Feature matrix and target vector
x = churn_data[['Monthly_charges', 'Tenure']]
y = churn_data['Churn']

# Splitting data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Logistic Regression model
log_model = LogisticRegression()
log_model.fit(x_train, y_train)

# Making predictions
```

```
y_pred = log_model.predict(x_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Logistic Regression - Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization (optional but illustrative)
plt.figure(figsize=(6, 4))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.xlabel("Actual Churn")
plt.ylabel("Predicted Churn")
plt.title("Actual vs Predicted Churn")
plt.plot([0, 1], [0, 1], 'r--')  # Diagonal reference line
plt.grid(True)
plt.show()
```

```
Logistic Regression - Accuracy: 0.50
Confusion Matrix:
[[8 1]
 [9 2]]

Classification Report:
              precision    recall  f1-score   support

           0       0.47      0.89      0.62         9
           1       0.67      0.18      0.29        11

    accuracy                           0.50        20
   macro avg       0.57      0.54      0.45        20
weighted avg       0.58      0.50      0.43        20
```



Practical 6: Building and evaluation Classification model

```
[1] import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
[2] iris=load_iris()##load data
X= iris.data
y=iris.target
```

```
[3] X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3, random_state=42)
```

```
[4] #Step 4: Initialize and Train Models
# Initialize models
dt_model = DecisionTreeClassifier(random_state=42)
nb_model = GaussianNB()
svm_model = SVC(random_state=42)
knn_model = KNeighborsClassifier()
```

```
[5] # Train models
dt_model.fit(X_train, y_train)
nb_model.fit(X_train, y_train)
svm_model.fit(X_train, y_train)
knn_model.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```
[6] #Step 5: Make Predictions
dt_pred = dt_model.predict(X_test)
nb_pred = nb_model.predict(X_test)
svm_pred = svm_model.predict(X_test)
knn_pred = knn_model.predict(X_test)
```

```
[7] #Step 6: Evaluate Models
#Use classification_report to get precision, recall, F1-score:

print("Decision Tree:\n", classification_report(y_test, dt_pred))
print("Naive Bayes:\n", classification_report(y_test, nb_pred))
print("SVM:\n", classification_report(y_test, svm_pred))
```

```
print("KNN:\n", classification_report(y_test, knn_pred))
print("Decision Tree Confusion Matrix:\n", confusion_matrix(y_test, dt_pred))
print("Naive Bayes Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))
print("SVM Confusion Matrix:\n", confusion_matrix(y_test, svm_pred))
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
```

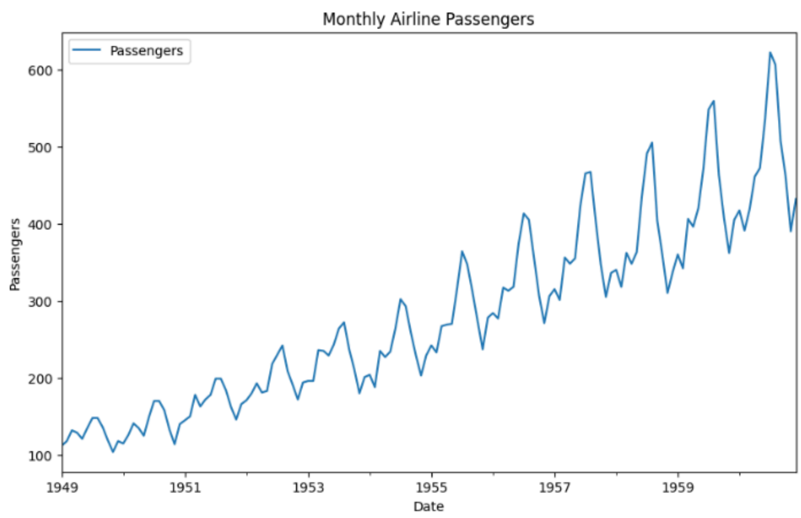
```
Decision Tree:
              precision    recall  f1-score   support

           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

Naive Bayes:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45

SVM:
              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

KNN:
              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


Decision Tree Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Naive Bayes Confusion Matrix:
 [[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
SVM Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
KNN Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

Practical 7: Time series Forecasting using ARIMA

```
# Step 1: Import Required Libraries
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error
import numpy as np

# Step 2: Load Time Series Dataset
df = pd.read_csv('airline_passengers.csv')
df.columns = ['Month', 'Passengers']  # Rename for consistency if needed
df['Month'] = pd.to_datetime(df['Month'])
df.set_index('Month', inplace=True)
print(df.head())
```

```
           Passengers
Month
1949-01-01        112
1949-02-01        118
1949-03-01        132
1949-04-01        129
1949-05-01        121
```

```
# Step 3: Visualize the Time Series
df.plot(figsize=(10, 6), title='Monthly Airline Passengers')
plt.xlabel("Date")
plt.ylabel("Passengers")
plt.show()
```

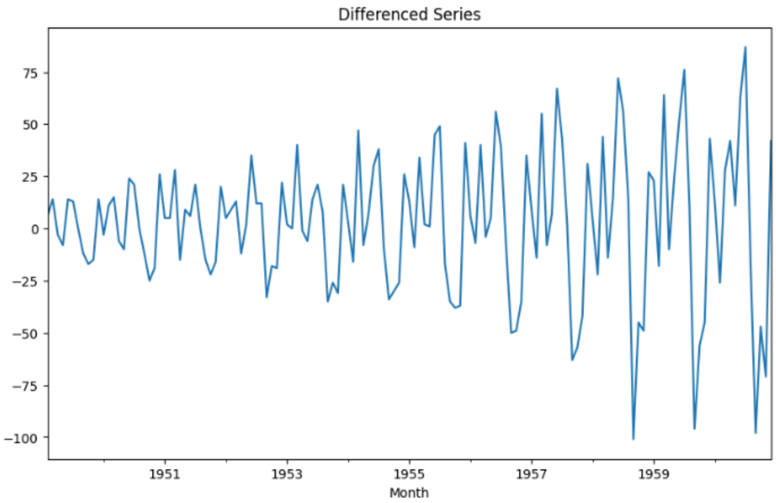



```
# Step 4: Check Stationarity
```

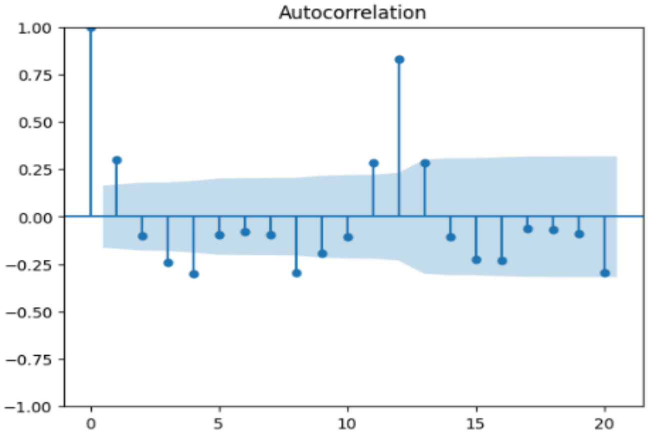
```
result = adfuller(df['Passengers'])
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
```

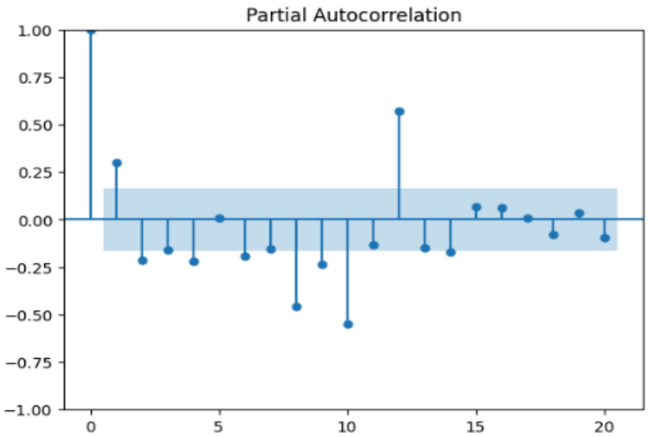
→ADF Statistic: 0.8153688792060463
p-value: 0.991880243437641

```
# Step 5: Differencing if not stationary
if result[1] > 0.05:
    df_diff = df['Passengers'].diff().dropna()
    df_diff.plot(figsize=(10, 6), title='Differenced Series')
    plt.show()
    data_to_model = df_diff
    d = 1
else:
    data_to_model = df['Passengers']
    d = 0
```




```
# Step 6: ACF and PACF plots to decide p and q
plot_acf(data_to_model, lags=20)
plot_pacf(data_to_model, lags=20)
plt.show()
```





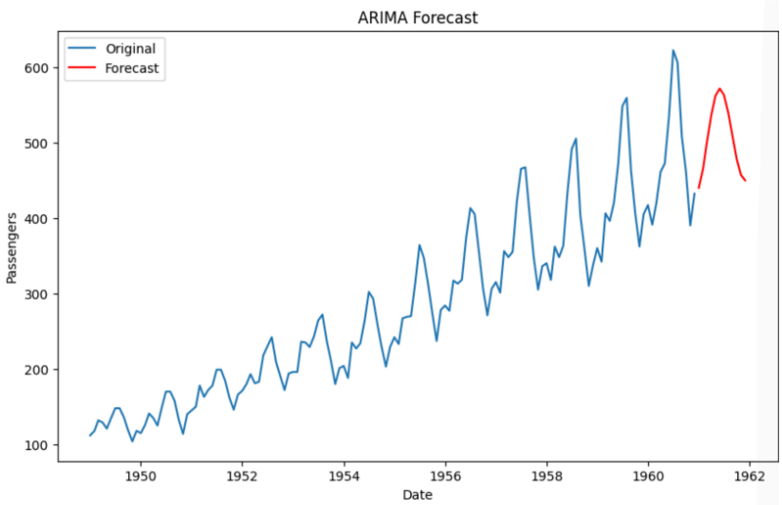

```
# Step 7: Fit ARIMA Model (Example: p=2, d=d, q=2)
model = ARIMA(df['Passengers'], order=(2, d, 2))
model_fit = model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:             Passengers   No. Observations:                  144
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -671.673
Date:                Fri, 23 May 2025   AIC                           1353.347
Time:                        23:01:55   BIC                           1368.161
Sample:                    01-01-1949   HQIC                          1359.366
                         - 12-01-1960
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.6850      0.020     83.061      0.000       1.645       1.725
ar.L2         -0.9549      0.017    -55.420      0.000      -0.989      -0.921
ma.L1         -1.8432      0.124    -14.852      0.000      -2.086      -1.600
ma.L2          0.9953      0.134      7.401      0.000       0.732       1.259
sigma2       665.9657    113.854      5.849      0.000     442.816     889.115
===================================================================================
Ljung-Box (L1) (Q):                   0.30   Jarque-Bera (JB):                 1.84
Prob(Q):                              0.59   Prob(JB):                         0.40
Heteroskedasticity (H):               7.38   Skew:                             0.27
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.14
===================================================================================
```

```
# Step 8: Forecast Future Values
forecast = model_fit.forecast(steps=12)
```

```
print("Forecast:\n", forecast)

# Plot original and forecast
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Passengers'], label='Original')
plt.plot(pd.date_range(df.index[-1], periods=13, freq='MS')[1:], forecast, label='Forecast', color='red')
plt.title('ARIMA Forecast')
plt.xlabel('Date')
plt.ylabel('Passengers')
plt.legend()
plt.show()
```

```
Forecast:
 1961-01-01    439.854649
1961-02-01    465.296295
1961-03-01    500.666061
1961-04-01    535.971971
1961-05-01    561.690397
1961-06-01    571.314654
1961-07-01    562.974471
1961-08-01    539.731319
1961-09-01    508.529683
1961-10-01    478.147936
1961-11-01    456.746897
1961-12-01    449.695697
Freq: MS, Name: predicted_mean, dtype: float64
```




```
# Step 9: Evaluate Model Performance (Train/Test Split)
train = df['Passengers'][:-12]
test = df['Passengers'][-12:]

model = ARIMA(train, order=(2, d, 2))
model_fit = model.fit()
pred = model_fit.forecast(steps=12)

rmse = np.sqrt(mean_squared_error(test, pred))
print(f'RMSE: {rmse:.2f}')→RMSE: 55.22
```

Practical 8: Text analysis and Sentiment Detection

```
[1]import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
# Download NLTK data (only first time)
nltk.download('punkt')
nltk.download('stopwords')
```

[2] #Step 2**: Load Sample Text Data**

```
#For demonstration, you can use a sample dataset of reviews, for example:
data = {'review': [
"I loved the movie! It was fantastic and thrilling.",
"The product is terrible, it broke after one use.",
"Not bad, but could be better.",
"Absolutely amazing! Highly recommend it.",
"Worst experience ever, will not buy again."
]}
df = pd.DataFrame(data)
```

Index	Review
0	I loved the movie! It was fantastic and thrilling.
1	The product is terrible, it broke after one use.
2	Not bad, but could be better.
3	Absolutely amazing! Highly recommend it.
4	Worst experience ever, will not buy again.

[3] #Step 3: **Text Preprocessing**

```
#Convert to lowercase
#Tokenize text
#Remove stopwords
#(Optional) Stemming or Lemmatization
stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    text = text.lower()
```

```
    words = word_tokenize(text)
    words = [w for w in words if w.isalpha()]  #| Remove punctuation/numbers
    words = [w for w in words if w not in stop_words]
    return ' '.join(words)
df['cleaned_review'] = df['review'].apply(preprocess_text)
print(df[['review', 'cleaned_review']])
```

Review	**Cleaned Review**
I loved the movie! It was fantastic and thrilling.	loved movie fantastic thrilling
The product is terrible, it broke after one use.	product terrible broke one use
Not bad, but could be better.	bad could better
Absolutely amazing! Highly recommend it.	absolutely amazing highly recommend
Worst experience ever, will not buy again.	worst experience ever buy

[4] #Step 4: **Compute TF-IDF Features**

```
tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(df['cleaned_review'])
print("TF-IDF feature names:\n", tfidf.get_feature_names_out())
print("TF-IDF matrix shape:", tfidf_matrix.shape)
```

```
TF-IDF feature names:
 ['absolutely' 'amazing' 'bad' 'better' 'broke' 'buy' 'could' 'ever'
 'experience' 'fantastic' 'highly' 'loved' 'movie' 'one' 'product'
 'recommend' 'terrible' 'thrilling' 'use' 'worst']
TF-IDF matrix shape: (5, 20)
```

[5] # Step 5: Perform Sentiment Analysis Using TextBlob:

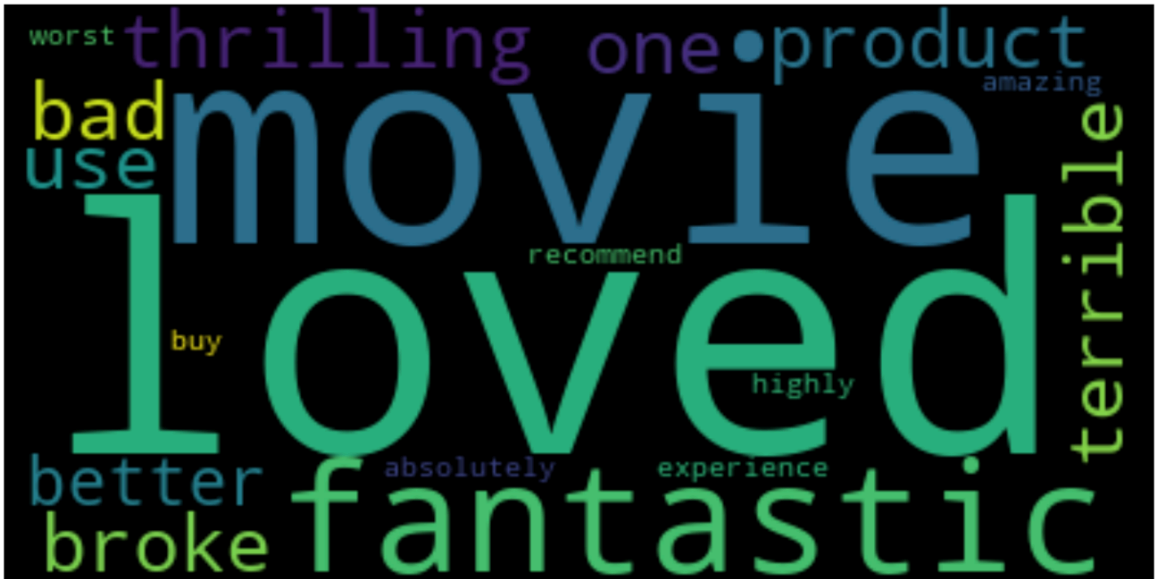
```
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

def get_textblob_sentiment(text):
    analysis = TextBlob(text)
    # Polarity ranges from -1 (negative) to 1 (positive)
    if analysis.sentiment.polarity > 0:
        return 'Positive'
    elif analysis.sentiment.polarity == 0:
        return 'Neutral'
    else:
        return 'Negative'
```

```
df['sentiment_textblob'] = df['review'].apply(get_textblob_sentiment)

# Using VADER (better for social media/reviews):
analyzer = SentimentIntensityAnalyzer()

def get_vader_sentiment(text):
    score = analyzer.polarity_scores(text)
    compound = score['compound']
    if compound >= 0.05:
        return 'Positive'
    elif compound <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

df['sentiment_vader'] = df['review'].apply(get_vader_sentiment)
```

```
[6] #Step 6: View Results
print(df[['review', 'sentiment_textblob', 'sentiment_vader']])
```

```
review sentiment_textblob  \
0  I loved the movie! It was fantastic and thrill...          Positive
1   The product is terrible, it broke after one use.          Negative
2                    Not bad, but could be better.          Positive
3          Absolutely amazing! Highly recommend it.          Positive
4        Worst experience ever, will not buy again.          Negative

  sentiment_vader
0       Positive
1       Negative
2       Positive
3       Positive
4       Negative
```

```
[7] from wordcloud import WordCloud
import matplotlib.pyplot as plt

def plot_cloud(docx):
    myWC = WordCloud().generate(docx)
    plt.figure(figsize=(20,10))
    plt.imshow(myWC, interpolation='bilinear')
    plt.axis('off')
    plt.show()

joy_docx = ' '.join(df['cleaned_review'])  # Combine all cleaned reviews into one string
plot_cloud(joy_docx)
```


movie
loved
fantastic
thrilling
one
product
bad
use
terrible
better
broke
worst
amazing
recommend
buy
highly
absolutely
experience

Practical 9: Hadoop HDFS and MapReduce Word Count

```
spark
```

SparkSession - hive

SparkContext

Spark UI

Version
v3.3.2
Master
local[8]
AppName
Databricks Shell

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.getOrCreate()

sc=spark.sparkContext

data=[1,2,3,4,5,6,7,8,9]

rdd=sc.parallelize(data,4)

rdd.collect()

Out[13]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
rdd.glom().collect()
Out[14]: [[1, 2], [3, 4], [5, 6], [7, 8, 9]]
result=rdd.count()
rdd1=sc.parallelize([1,2,3,4,5,6])
result=rdd1.filter(lambda x:x%2 == 0)
print(result.collect())
[2, 4, 6]

rdd2=sc.parallelize([1,2,3])
result=rdd2.flatMap(lambda x:(x,x*2))
print(result.collect())
[1, 2, 2, 4, 3, 6]

rdd3=sc.parallelize([('a',1),('b',2),('a',3)])
```

```
result=rdd3.reduceByKey(lambda x,y:x+y)
print(result.collect())
[('a', 4), ('b', 2)]

rdd4=sc.parallelize([('a',1),('b',2),('a',3),('b',4)])
result=rdd4.groupByKey().mapValues(list)
print(result.collect())
[('a', [1, 3]), ('b', [2, 4])]

rdd5=sc.parallelize([('a',1),('b',2)])
rdd6=sc.parallelize([('a',3),('b',4)])
result= rdd5.join(rdd6)
print(result.collect())
[('b', (2, 4)), ('a', (1, 3))]

rdd7=sc.parallelize([1,2,3,2,1])
result=rdd7.distinct()
print(result.collect())
[1, 2, 3]
#rdd action
rdd01=sc.parallelize([1,2,3,4])
result=rdd01.collect()
print(result)
[1, 2, 3, 4]

rdd01=sc.parallelize([1,2,3,4])
result=rdd01.count()
print(result)
4

rdd03=sc.parallelize([1,2,3,4])
result=rdd03.reduce(lambda x,y:x+y)
print(result)
10

result=rdd01.first()
print(result)
1

result=rdd01.take(3)
print(result)
[1, 2, 3]
```

Practical 10: Building a spark Application with PySpark

```
spark
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
sc=spark.sparkContext
data=[
   ("Alice",29,"Engineering"),
   ("Bob,",35,"Sales"),
   ("Charlie",40,"Engineering"),
   ("David",30,"HR"),
   ("Eva",25,"Sales")
]
columns=["name","age","department"]
df=spark.createDataFrame(data,columns)
df.display()
```

df: pyspark.sql.dataframe.DataFrame = [name: string, age: long ... 1 more field]

Table

	name	age	department
1	Alice	29	Engineering
2	Bob,	35	Sales
3	Charlie	40	Engineering
4	David	30	HR
5	Eva	25	Sales

5 rows

```
df.printSchema()
root
 |-- name: string (nullable = true)
 |-- age: long (nullable = true)
 |-- department: string (nullable = true)
df.show()
+-------+---+-----------+
|   name|age| department|
```

```
+-------+---+-----------+
|  Alice| 29|Engineering|
|   Bob,| 35|      Sales|
|Charlie| 40|Engineering|
|  David| 30|         HR|
|    Eva| 25|      Sales|
+-------+---+-----------+

df.select("name","age").show()
+-------+---+
|   name|age|
+-------+---+
|  Alice| 29|
|   Bob,| 35|
|Charlie| 40|
|  David| 30|
|    Eva| 25|
+-------+---+

df.filter(df.age>30).show()
+-------+---+-----------+
|   name|age| department|
+-------+---+-----------+
|   Bob,| 35|      Sales|
|Charlie| 40|Engineering|
+-------+---+-----------+

from pyspark.sql.functions import col
df.withColumn("age_plus",col("age")+5).show()
+-------+---+-----------+--------+
|   name|age| department|age_plus|
+-------+---+-----------+--------+
|  Alice| 29|Engineering|      34|
|   Bob,| 35|      Sales|      40|
|Charlie| 40|Engineering|      45|
|  David| 30|         HR|      35|
|    Eva| 25|      Sales|      30|
+-------+---+-----------+--------+
df.groupBy("department").avg("age").show()
+-----------+--------+
| department|avg(age)|
+-----------+--------+
```

```
|Engineering|    34.5|
|      Sales|    30.0|
|         HR|    30.0|
+-----------+--------+
df.orderBy("age",ascending=False).show()
+-------+---+-----------+
|   name|age| department|
+-------+---+-----------+
|Charlie| 40|Engineering|
|   Bob,| 35|      Sales|
|  David| 30|         HR|
|  Alice| 29|Engineering|
|    Eva| 25|      Sales|
+-------+---+-----------+

df.orderBy("age",decending=False).show()
+-------+---+-----------+
|   name|age| department|
+-------+---+-----------+
|    Eva| 25|      Sales|
|  Alice| 29|Engineering|
|  David| 30|         HR|
|   Bob,| 35|      Sales|
|Charlie| 40|Engineering|
+-------+---+-----------+

df.withColumnRenamed("age","employee_age").show()
+-------+------------+-----------+
|   name|employee_age| department|
+-------+------------+-----------+
|  Alice|          29|Engineering|
|   Bob,|          35|      Sales|
|Charlie|          40|Engineering|
|  David|          30|         HR|
|    Eva|          25|      Sales|
+-------+------------+-----------+

df.drop("department").show()
+-------+---+
|   name|age|
+-------+---+
|  Alice| 29|
```

```
|   Bob,| 35|
|Charlie| 40|
|  David| 30|
|    Eva| 25|
+-------+---+
df.withColumn("age",col("age").cast("string")).printSchema()
root
 |-- name: string (nullable = true)
 |-- age: string (nullable = true)
 |-- department: string (nullable = true)

df.describe()
Out[22]: DataFrame[summary: string, name: string, age: string, department: string]

from pyspark.sql.functions import when
df.withColumn("senior",when(col("age")>30,"Yes").otherwise("NO")).show()

+-------+---+-----------+------+
|   name|age| department|senior|
+-------+---+-----------+------+
|  Alice| 29|Engineering|    NO|
|   Bob,| 35|      Sales|   Yes|
|Charlie| 40|Engineering|   Yes|
|  David| 30|         HR|    NO|
|    Eva| 25|      Sales|    NO|
+-------+---+-----------+------+

df.select("department").distinct().show()
+-----------+
| department|
+-----------+
|Engineering|
|      Sales|
|         HR|
+-----------+
```