EXPERIMENT 1

AIM: Tutorials on design Star Schema & Snowflake Schema.

THEORY:

- SCHEMA

- A schema is a structure or blueprint of a database that defines how data is organised, including tables, columns, relationships, constraints & indexes. It determines how data is stored, accessed, & manipulated in a database. A schema can be made or can be divided into two types: STAR SCHEMA & SNOWFLAKE SCHEMA.

- STAR SCHEMA

- A star schema is a type of database schema commonly used in data warehousing. It consists of a central fact table connected to multiple dimension tables, forming a star-like structure. It is easy to make and simple to understand. It consists of the following components:

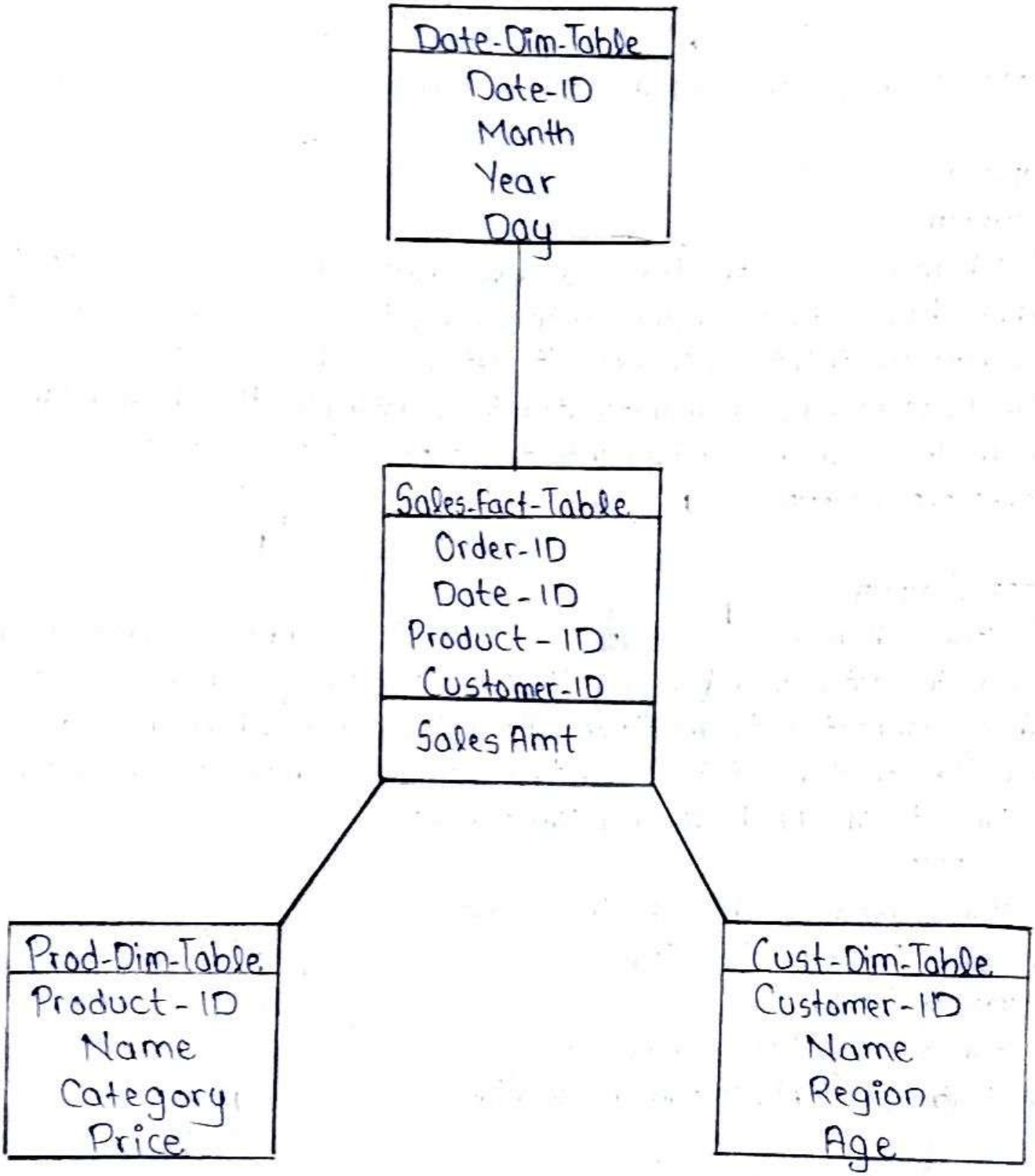
① FACT TABLE:
- It stores measureable business data.
- Eg: Sales, Revenue, Quantity.

② DIMENSION TABLES:
- It stores descriptive attributes.
- Eg: Date, product, customer, location.

- Example on Star Schema

i] Fact Table:
- Sales [Order-ID, Date-ID, Product-ID, Customer-ID, Sales-Amount]

STAR SCHEMA


Date-Dim-Table
Date-ID
Month
Year
Day
Sales-Fact-Table
Order-ID
Date-ID
Product-ID
Customer-ID
Sales Amt
Prod-Dim-Table
Product-ID
Name
Category
Price
Cust-Dim-Table
Customer-ID
Name
Region
Age

ii) Dimension Table:
- Date [Date-ID, Year, Month, Day]
- Product [Product-ID, Name, Category, Price]
- Customer [Customer-ID, Name, Region, Age]

- It has fast query performance.
- It suits good for OLAP (Online Analytical Processing)

- SNOWFLAKE SCHEMA
- A snowflake schema is an extension of the Star Schema where it consists of centralized Fact Table and Dimension Tables but these dimension tables are further normalized, reducing redundancy. It forms a snowflake-like structure. It consists of the following components:

① Fact Table:
- It stores measureable data.

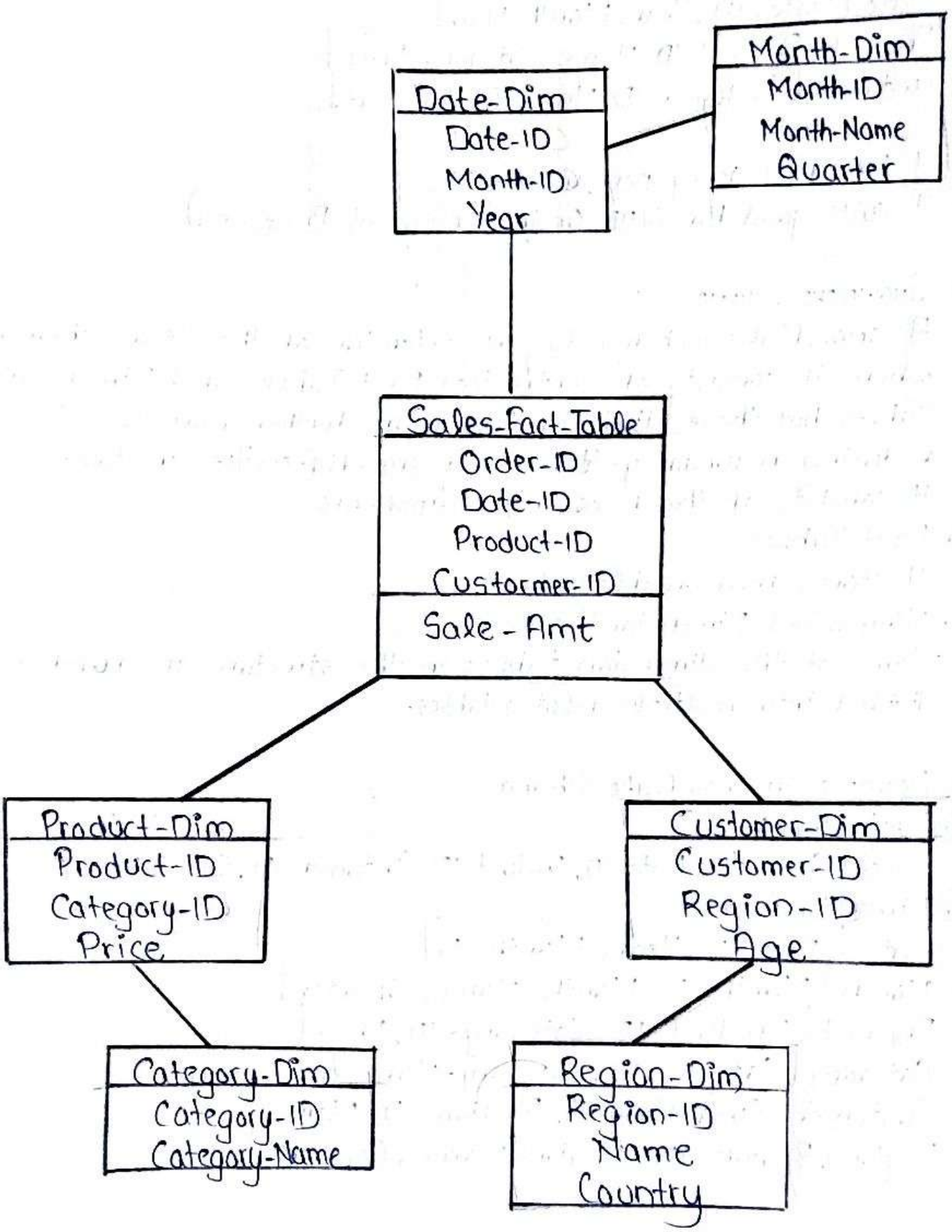
② Normalized Dimension Tables:
- Some of the dimension tables in the structure are further divided into multiple related tables.

- Example on Snowflake Schema

① Fact Table:
- Sales [Order-ID, Date-ID, Product-ID, Customer-ID, Sales-Amt]

② Dimension tables:
- Date [Date-ID, Year, Month-ID]
- Month [Month-ID, Month-Name, Quarter]
- Product [Product-ID, Category-ID, Price]
- Category [Category-ID, Category-Name]
- Customer [Customer-ID, Region-ID, Name]
- Region [Region-ID, Region-Name, Country]

Snowflake Schema



- It reduces data redundancy
- It saves storage space
- It is more organized & structured

CONCLUSION: Hence, we have successfully designed Star Schema and Snowflake Schema.

EXPERIMENT 2

AIM: Implement Data Exploration using tools or languages like JAVA / Python / R.

THEORY:

- DATA EXPLORATION

- Data Exploration is the initial step in data analysis, where raw data is examined to understand its structure, patterns, anomalies, and relationships. This process helps analysts and data scientists clean, pre-processes, and transform data for further analysis or modeling.

- Why is data exploration important?

① Identifies missing or inconsistent data.
② Detects outliers and anomalies.
③ Helps in feature selection & engineering.
④ Uncovers patterns, trends & distributions.
⑤ Guides data preprocessing and cleaning.

- STEPS FOR DATA EXPLORATION [PYTHON]

- STEP 1: Load the Dataset

Python provides libraries like 'pandas' to load datasets efficiently.

- import pandas as pd
df = pd.read_csv('data.csv')
df.head()

- STEP 2: Understand Data Structure

Check basic information about dataset

- df.info()
- df.describe()
df.shape()

df.columns

- STEP 3: Handle Missing Data

Find & Manage Missing data

```
df.isnull().sum()
df.fillna(df.mean(), inplace=True)
df.dropna(inplace=True)
```

- STEP 4: Identify Outliers & Data Distribution

Use visualizations to detect outliers & distribution.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.boxplot(x=df['column-name'])
sns.histplot(df['column-name'], bins=30, kde=True)
plt.show()
```

- STEP 5: Detect Correlations & Relationships

Use correlation matrices & scatter plots to find relationships.

```
correlation-matrix = df.corr()
sns.heatmap(correlation-matrix, annot=True, cmap='coolwarm')
plt.show()
sns.pairplot(df).
plt.show()
```

CONCLUSION: Hence, we have successfully implemented Data Exploration using tools/languages like JAVA/Python/R.

EXPERIMENT 3

AIM: Implement Data pre-processing using tools or languages like JAVA/Python/R

THEORY:

- **DATA PRE-PROCESSING**

- Data pre-processing is the process of cleaning, transforming, & organizing raw data before using it for analysis or machine learning. It ensures data quality, improves accuracy, and enhances model performance.

- Why is data pre-processing important

① Handles Missing Data: Fills or removes missing values.
② Removes Noise & Outliers: Eliminates inconsistencies.
③ Normalizes data: Converts data into a standard format.
④ Encodes Categorial Data: Converts non-numerical data into numerical values.
⑤ Improves Model Efficiency & Enhances prediction accuracy.

- **STEPS FOR DATA PRE-PROCESSING [PYTHON]**

- STEP 1: Load the dataset
Use 'pandas' to load & inspect the dataset.

```
import pandas as pd
df = pd.read_csv('data.csv')
df.head()
```

- STEP 2: Handle Missing Values
Find and fix missing data

```
df.isnull().sum()
df.fillna(df.mean(), inplace=True)
df.dropna(inplace=True)
```

- STEP 3: Handle Categorial Data
Convert non-numeric data into numeric format

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['Category'] = encoder.fit_transform(df['Category'])
```

- STEP 4 : Feature Scaling
Normalize or standardize numerical data.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['Feature 1', 'Feature 2']] = scaler.fit_transform(df[['Feature 1',
                                                          'Feature 2']])
```

- STEP 5: Remove Outliers
Use visualization & statistical methods to detect & remove outliers.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.boxplot(x=df['Feature 1'])
df = df[(df['Feature 1'] > df['Feature 1'].quantile(0.05)) &
        (df['Feature 1'] < df['Feature 1'].quantile(0.95))]
```

CONCLUSION: Hence, we have successfully implemented Data Pre-Processing using tools/languages like JAVA/ Python/R.

EXPERIMENT 4

AIM: Perform and Evaluate Classification Algorithms using any open source tools.

THEORY:

- Classification Algorithms

Classification is a Supervised Machine learning Technique where the goal is to predict categorial labels (eg: spam or not spam, disease or no disease). It maps input features to a predefined category.

Following are the types of classification Algorithms:

① Logistic Reasoning / Regression:
- A statistical model for binary classification (Yes/No, 0/1).

② Decision Tree:
- A tree-like model that splits data based on conditions.

③ Random Forest:
- An ensemble of multiple decision trees for better accuracy.

④ Support Vector Machine (SVM):
- Finds the best boundary (hyperplane) between classes.

⑤ Neural Networks:
- Deep learning-based approach for complex classifications.

- Open Source tools for Classification Algorithms

- Many tools support classification tasks, including:

① Sci-kit learn:
- Python-based library for machine learning

② Tensorflow / Keras:
- Deep learning frameworks for neural network-based classification.

③ Weka:
- Java-based data mining & machine learning tool.

④ Orange:
- GUI based Machine learning platform.

⑤ RapidMiner:
- Open-source data science platform for classification tasks.

• Steps to Perform & Evaluate Classification using Python (Sci-kit learn)

① STEP 1: Load Dataset

```
- from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
data = load_iris()
X_train, X_test, Y_train, Y_test = train_test_split(data.data)
```

② STEP 2: Train a classification model

```
- from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, Y_train)
```

③ STEP 3: Make predictions

```
- Y_pred = model.predict(X_test)
```

④ STEP 4: Evaluate the Model

```
- from sklearn.metrics import confusion_matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

CONCLUSION: Hence, we have successfully performed & evaluated classification algorithm using open source tools.

EXPERIMENT 5

AIM: Implement & Evaluate Classification Algorithms using languages like JAVA/Python/R

THEORY:

- STEPS to implement & Evaluate Classification Algorithms using Python.

STEP 1: Import Required libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
```

STEP 2: Load & Preprocess the Dataset

```
df = pd.read_csv('dataset.csv')
df.dropna(inplace=True)
X = df.drop('target', axis=1)
y = df['target']
```

STEP 3: Split data into Training & Testing Sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
                                   0.2, random=42)
```

STEP 4: Train a classification model

```
model = RandomForestClassifier(n_estimators=100, random=42)
model.fit(X_train, y_train)
```

STEP 5: Make Predictions

- y_pred = model.predict(X_test)

STEP 6: Evaluate Model Performance

- print("Confusion Matrix", confusion_matrix(y_test, y_pred))

CONCLUSION: Hence, we have successfully implemented & evaluated Classification Algorithm using languages like JAVA/Python/R.

EXPERIMENT 6

AIM: Perform & Evaluate Clustering Algorithms using any open source tools.

THEORY:

- Clustering Algorithms

Clustering is an unsupervised Machine learning technique that groups similar data points together based on their features. Unlike classification, clustering does not require labeled data.

Types of Clustering algorithms:

① K-Means Clustering:
- Partitions data in K-clusters based on centroids.

② Hierarchical Clustering:
- Forms a hierarchy of clusters (Agglomerative & Divisive)

③ DBSCAN (Density-Based Spatial Clustering of Apps with Noise):
- Groups dense regions while ignoring noise.

④ Mean Shift Clustering:
- Shifts points towards high-density points.

⑤ Gaussian Mixture Models (GMM):
- Probabilistic clustering using Gaussian distributions.

- Open Source tools for Clustering Algorithms

Several tools support clustering, including:

① Sci-kit learn:
- Provides implementation of K-Means, DBScans, etc.

② TensorFlow/Keras:
- Deep-learning based clustering

③ Weka:
- Supports multiple clustering algorithms.

④ Orange:
- GUI based Machine learning with clustering support

⑤ RapidMiner:
- Open-source data science platform.

• Steps to Perform & Evaluate Clustering using Python (Sci-kit learn)

① STEP 1: Load & Preprocess Data
```
- import pandas as pd
from sklearn.preprocessing import StandardScaler
dF = pd.read_csv('data.csv')
df.dropna(inplace = True)
X = StandardScaler().fit_transform(df)
```

② STEP 2: Apply K-Means Clustering
```
- from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 3, random = 42)
df['Cluster'] = kmeans.fit_predict(X)
```

③ STEP 3: Evaluate Clustering Performance
```
- from sklearn.metrics import silhouette_score
score = silhouette_score(X, df['Cluster'])
print("Silhouette Score:", score)
```

④ STEP 4: Visualize
```
- import matplotlib.pyplot as plt
import seaborn as sns
sns.scatterplot(x=X[0], y=X[:, 1], hue=df['Cluster']
plt.show()
```

CONCLUSION: Hence, we have successfully performed & evaluated ~~classification~~ clustering algorithms using opensource tools

EXPERIMENT 7

AIM: Implement & Evaluate Clustering Algorithms using languages like JAVA / Python / R

THEORY:

- Steps to Implement and Evaluate Clustering Algorithms using Python.

STEP 1: Import Required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as pyplot
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
```

STEP 2: Load and Pre-process the Dataset

```
df = pd.read_csv('data.csv')
df.dropna(inplace=True)
X = StandardScaler().fit_transform(df)
```

STEP 3: Apply Clustering Algorithm

```
kmeans = KMeans(n_cluster=3, random=42)
df['Cluster'] = kmeans.fit_predict(X)
```

STEP 4: Evaluate Clustering Performance

```
score = silhouette_score(X, df['Cluster'])
print("Silhouette Score:", score)
```

STEP 5: Visualize

```
sns.scatterplot(x=X[:,0], y=x[:,1], hue=dF['Cluster'])
plt.title('Visualization')
plt.show()
```

CONCLUSION: Hence, we have successfully implemented & evaluated Clustering Algorithms using languages like JAVA/Python/R.

EXPERIMENT 8

AIM: Perform & Evaluate Frequent Pattern Mining Algorithms using any open source tools.

THEORY:

- Frequent Pattern Mining Algorithms

Frequent Pattern Mining (FPM) is a data mining technique used to find patterns, associations, correlations or frequent itemsets in large datasets. It is widely used in market basket analysis, recommendation systems, & fraud detection.

Following are the types of Pattern Mining Algorithms:

① Apriori Algorithms:
- Generates frequent itemsets & associations rules using a breadth-first search

② FP Growth:
- An improved version of Apriori that uses a compact FP-Tree structure to reduce computation.

③ ECLAT:
- It uses depth-first search for faster itemset generation.

- Open source tools for Frequent Pattern Mining

- Several open-source tools support, including:

① MLxtend:
- Implements Apriori and FP-Growth for association rule mining.

② Orange 3:
- Offers built-in FPM for non-programmers.

③ Weka:
- Includes Apriori for association rule mining.

④ R:
- Provides tools for Apriori & FP Growth in R.

⑤ Rapid Miner:
- A GUI-based tool for data mining & frequent pattern extraction.

• Steps to perform & evaluate Frequent Pattern Mining using MLxtend.

① STEP 1: Import required libraries
```
- !pip install mlxtend
import pandas as pd
from mlxtend.frequent-patterns import apriori
```
② STEP 2: load & Preprocess the dataset
```
- df = pd.read-csv ('transaction.csv')
df = df.applymap (lambda x:1 if x else 0)
```
③ STEP 3: Apply Priori
```
- frequent_itemsets = apriori (df, min_sup = 0.2, use_col = True)
print (frequent_itemsets)
```

CONCLUSION: Hence, we have successfully performed & evaluated frequent pattern mining algorithms using open source tools.

EXPERIMENT 9

AIM: Implement & Evaluate Frequent Pattern Mining Algorithms using languages like JAVA/Python/R

THEORY:

- Steps to implement & Evaluate Frequent Pattern Mining Algorithms using Python.

STEP 1: Install & Import Libraries.

```
! pip install mlxtend
import pandas as pd
from mlxtend.frequent_patterns import apriori, rules.
```

STEP 2: Load & Preprocess the Dataset

```
df = pd.read_csv('transactions.csv')
df = df.applymap(lamba x: 1 if x else 0)
```

STEP 3: Apply Apriori

```
frequent_items = apriori(df, min_sup=0.2, use_col=True)
print(frequent_items)
```

STEP 4: Generate Rules

```
rules = association_rules(frequent_items, min=0.5)
print(rules)
```

STEP 5: Evaluate

```
strong_rules = rules.sort_values(by='lift', ascending=False)
print(strong_rules)
```

CONCLUSION: Hence, we have successfully implemented & evaluated FPM using languages like JAVA/Python/R