

Experiment No. 1

Aim: Select a dataset and perform exploratory data analysis using Python(data preprocessing, transformation, discretization and visualisation)

Theory:

A) DISCRETIZATION:

Data discretization refers to a method of converting a huge number of data values into smaller ones so that the evaluation and management of data become easy. In other words, data discretization is a method of converting attributes values of continuous data into a finite set of intervals with minimum data loss.

There are two forms of data discretization: first is supervised discretization, and the second is unsupervised discretization. Supervised discretization refers to a method in which the class data is used. Unsupervised discretization refers to a method depending upon the way which operation proceeds. It means it works on the top-down splitting strategy and bottom-up merging strategy.

SOME FAMOUS TECHNIQUES IN DATA DISCRETIZATION:

1. Histogram analysis

Histogram refers to a plot used to represent the underlying frequency distribution of a continuous data set. Histogram assists the data inspection for data distribution. For example, Outliers, skewness representation, normal distribution representation, etc.

2. Binning

Binning refers to a data smoothing technique that helps to group a huge number of continuous values into smaller values. For data discretization and the development of idea hierarchy, this technique can also be used.

3. Cluster Analysis

Cluster analysis is a form of data discretization. A clustering algorithm is executed by dividing the values of x numbers into clusters to isolate a computational feature of x. 4. Data discretization using decision tree analysis

Data discretization refers to a decision tree analysis in which a top-down slicing technique is used. It is done through a supervised procedure. In a numeric attribute discretization, first, you need to select the attribute that has the least entropy, and

then you need to run it with the help of a recursive process. The recursive process divides it into various discretized disjoint intervals, from top to bottom, using the same splitting criterion.

5. Data discretization using correlation analysis

Discretizing data by linear regression technique, you can get the best neighboring interval, and then the large intervals are combined to develop a larger overlap to form the final 20 overlapping intervals. It is a supervised procedure.

B) DATA VISUALIZATION:

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.

In the world of Big Data, data visualization tools and technologies are essential to analyze massive amounts of information and make data-driven decisions.

Types of data visualization charts : -

Now that we understand how data visualization can be used, let's apply the different types of data visualization to their uses. There are numerous tools available to help create data visualizations. Some are more manual and some are automated, but either way they should allow you to make any of the following types of visualizations.

1. Line chart

A line chart illustrates changes over time. The x-axis is usually a period of time, while the y-axis is quantity. So, this could illustrate a company's sales for the year broken down by month or how many units a factory produced each day for the past week.

2. Area chart

An area chart is an adaptation of a line chart where the area under the line is filled in to emphasize its significance. The color fill for the area under each line should be somewhat transparent so that overlapping areas can be discerned.

3. Bar chart

A bar chart also illustrates changes over time. But if there is more than one variable, a bar chart can make it easier to compare the data for each variable at each moment in time. For example, a

bar chart could compare the company's sales from this year to last year.

4. Histogram

A histogram looks like a bar chart, but measures frequency rather than trends over time. The x-axis of a histogram lists the "bins" or intervals of the variable, and the y-axis is frequency, so each bar represents the frequency of that bin. For example, you could measure the frequencies of each answer to a survey question. The bins would be the answer: "unsatisfactory," "neutral," and "satisfactory." This would tell you how many people gave each answer.

5. Scatter plot

Scatter plots are used to find correlations. Each point on a scatter plot means "when x = this, then y equals this." That way, if the points trend a certain way (upward to the left, downward to the right, etc.) there is a relationship between them. If the plot is truly scattered with no trend at all, then the variables do not affect each other at all.

6. Bubble chart

A bubble chart is an adaptation of a scatter plot, where each point is illustrated as a bubble whose area has meaning in addition to its placement on the axes. A pain point associated with bubble charts is the limitations on sizes of bubbles due to the limited space within the axes. So, not all data will fit effectively in this type of visualization.

7. Pie chart

A pie chart is the best option for illustrating percentages, because it shows each element as part of a whole. So, if your data explains a breakdown in percentages, a pie chart will clearly present the pieces in the proper proportions.

8. Gauge

A gauge can be used to illustrate the distance between intervals. This can be presented as a round clock-like gauge or as a tube type gauge resembling a liquid thermometer. Multiple gauges can be shown next to each other to illustrate the difference between multiple intervals.

9. Map

Much of the data dealt with in businesses has a location element, which makes it easy to illustrate on a map. An example of a map visualization is mapping the number of purchases customers made in each state in the U.S. In this example, each state would be shaded in and states with less purchases would be a lighter shade, while states with more purchases would be darker shades. Location information can also be very valuable for business leadership to understand, making this an important data visualization to use.

10. Heat map

A heat map is basically a color-coded matrix. A formula is used to color each cell of the matrix and is shaded to represent the relative value or risk of that cell. Usually heat map colors range from green to red, with green being a better result and red being worse. This type of visualization is helpful because colors are quicker to interpret than numbers.

11. Frame diagram

Frame diagrams are basically tree maps which clearly show hierarchical relationship structure. A frame diagram consists of branches, which each have more branches connecting to them with each level of the diagram consisting of more and more branches.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler,
StandardScaler

# Load the dataset

data = pd.read_csv("100 Sales Records.csv") # Replace
with the actual dataset filename

# Check for missing values

missing_values = data.isnull().sum()

print("Missing Values:")
```

```

print(missing_values)

# Data Transformation

# Min-Max Normalization
numerical_features = ["Unit Price", "Unit Cost", "Total
Revenue", "Total Cost", "Total Profit"]

scaler = MinMaxScaler()

data[numerical_features] =
scaler.fit_transform(data[numerical_features])

# Z-Score Normalization
mean = data[numerical_features].mean()

std = data[numerical_features].std()

data[numerical_features] = (data[numerical_features] -
mean) / std

# Decimal Scaling (Choose a decimal point position)
decimal_point = 3

data[numerical_features] = data[numerical_features] /
10**decimal_point

# Data Discretization (Binning - choose the number of
bins)
num_bins = 5

data['Total Revenue Binned'] = pd.cut(data['Total
Revenue'], bins=num_bins, labels=False)

data['Total Cost Binned'] = pd.cut(data['Total Cost'],
bins=num_bins, labels=False)

data['Total Profit Binned'] = pd.cut(data['Total Profit'],
bins=num_bins, labels=False)

# Data Analysis and Visualization

# Summary Statistics
summary_stats = data.describe()

print("Summary Statistics:")

print(summary_stats)

```

```

# Histograms
data[numerical_features].hist(bins=20, figsize=(12, 8))

plt.suptitle("Histograms of Numerical Features")

plt.show()

# Box Plots
sns.set(style="whitegrid")

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

sns.boxplot(data=data[numerical_features])

plt.title("Box Plots of Numerical Features")

plt.xticks(rotation=45)

plt.show()

# Pairplot for correlations
sns.pairplot(data[numerical_features], diag_kind="kde")

plt.suptitle("Pairplot of Numerical Features")

plt.show()

# Correlation Matrix
correlation_matrix = data[numerical_features].corr()

plt.figure(figsize=(10, 6))

sns.heatmap(correlation_matrix, annot=True,
            cmap="coolwarm", linewidths=.5)

plt.title("Correlation Matrix")

plt.show()

# Grouping and Aggregation (Example: Average Total Profit
by Order Priority)
average_profit_by_priority = data.groupby('Order
Priority')['Total Profit'].mean().reset_index()

print("Average Total Profit by Order Priority:")

print(average_profit_by_priority)

```

Output:

Missing Values:

Region 0

Country 0

Item Type 0

Sales Channel 0

Order Priority 0

Order Date 0

Order ID 0

Ship Date 0

Units Sold 0

Unit Price 0

Unit Cost 0

Total Revenue 0

Total Cost 0

Total Profit 0

dtype: int64

Summary Statistics:

Order ID Units Sold Unit Price Unit

Cost Total Revenue Total Cost Total Profit Total

Revenue Binned Total Cost Binned Total Profit Binned

count 1.000000e+02 100.000000 1.000000e+02

1.000000e+02 1.000000e+02 1.000000e+02 1.000000e+02

100.000000 100.000000 100.000000

mean 5.550204e+08 5128.710000 1.734723e-19 -5.204170e-

20 -3.903128e-20 1.452831e-19 7.155734e-20

0.740000 0.660000 0.880000

std 2.606153e+08 2794.484562 1.000000e-03 1.000000e-

03 1.000000e-03 1.000000e-03 1.000000e-03

1.106637 1.056199 1.191553

min 1.146066e+08 124.000000 -1.135145e-03 -9.783209e-

04 -9.373908e-04 -8.563158e-04 -1.004301e-03

0.000000 0.000000 0.000000

25% 3.389225e+08 2836.250000 -8.278341e-04 -8.246613e-

04 -7.566745e-04 -7.038571e-04 -7.302411e-04

0.000000 0.000000 0.000000

50% 5.577086e+08 5382.500000 -4.112245e-04 -4.451082e-

04 -4.254528e-04 -5.242359e-04 -3.441299e-04

0.000000 0.000000 0.000000

75% 7.907551e+08 7369.000000 6.810016e-04 3.840534e-

04 5.743428e-04 6.292462e-04 4.427139e-04

1.000000 1.000000 1.000000

max 9.940222e+08 9925.000000 1.661806e-03 1.774163e-

03 3.166765e-03 3.300915e-03 2.914777e-03

4.000000 4.000000 4.000000

C:\Users\umesh\AppData\Local\Programs\Python\Python311\Lib

\site-packages\seaborn\axisgrid.py:118: UserWarning: The

figure layout has changed to tight

self._figure.tight_layout(*args, **kwargs)

Average Total Profit by Order Priority:

Order Priority Total Profit

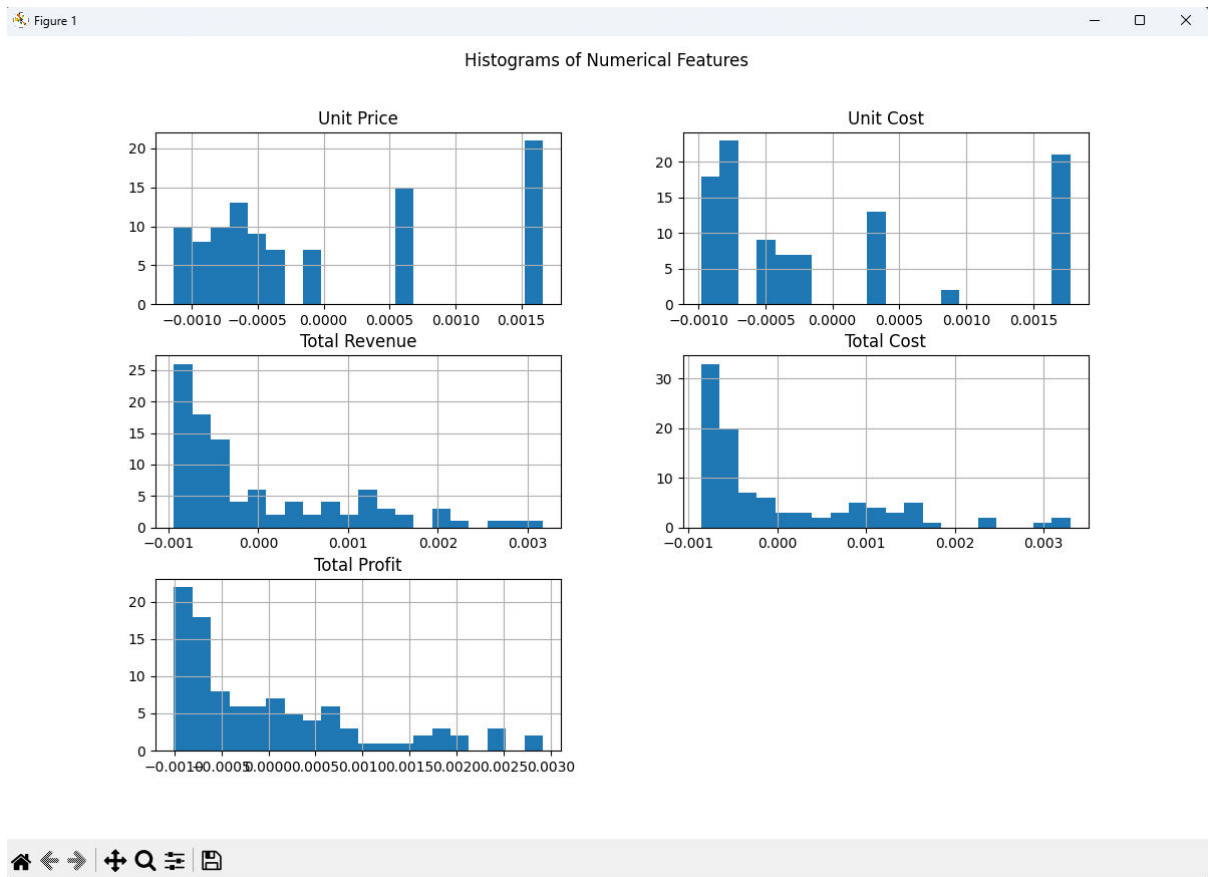
0 C -0.000308

1 H 0.000277

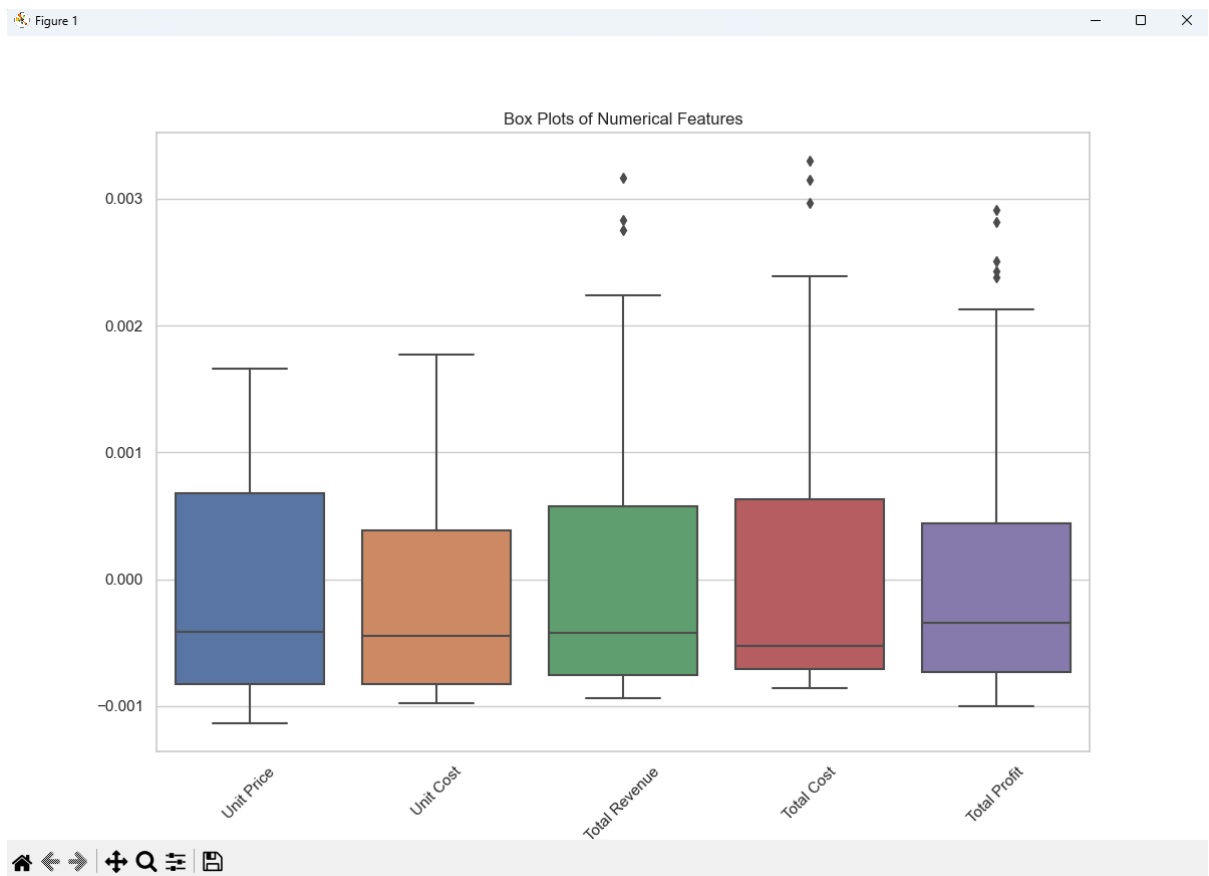
2 L -0.000090

3 M 0.000043

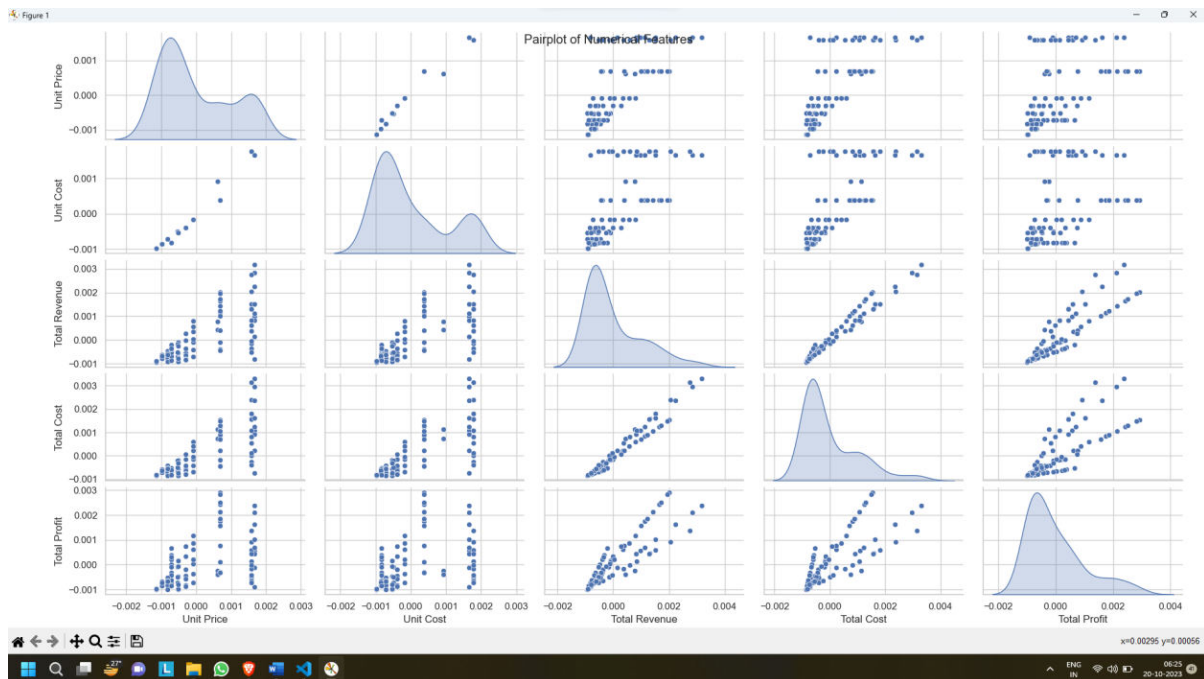
Histogram



Boxplot



Pairplot



Correlation matrix



Conclusion: Thus, we implemented exploratory data analysis using Python(data preprocessing, transformation, discretization and visualisation).

Experiment No. 2

Aim: Implement case study on building Data warehouse/Data Mart, write Detailed Problem statement and design dimensional modelling (creation of star and snowflake schema)

Theory:

Star Schema:

A star schema is a widely used data modelling technique in the field of data warehousing and business intelligence. It structures data in a way that simplifies querying and reporting for analytical purposes. The term "star schema" derives from the resemblance of its visual representation to a star, where a central fact table is connected to multiple dimension tables radiating outwards like star points. Here is an overview of the key aspects of a star schema:

1. Fact Table:

- At the centre of the star schema is the fact table, which holds quantitative and numeric data that represent business events or transactions, such as sales, orders, or revenue.
- Each row in the fact table corresponds to a specific event or transaction and contains measures or metrics to be analysed, such as sales amount or profit.

2. Dimension Tables:

- Surrounding the fact table are dimension tables, which contain descriptive attributes that provide context to the measures in the fact table.
- Dimension tables represent the various ways data can be sliced, diced, or analyzed. Examples include time, geography, products, customers, and more.
- Each dimension table typically contains a primary key that establishes a relationship with the fact table.

3. Relationships:

- The fact table is connected to each dimension table through foreign keys. These keys create relationships that allow analysts to correlate facts with relevant dimensions.
- Relationships enable data users to answer questions like "What were the sales in a specific region during a particular time period?"

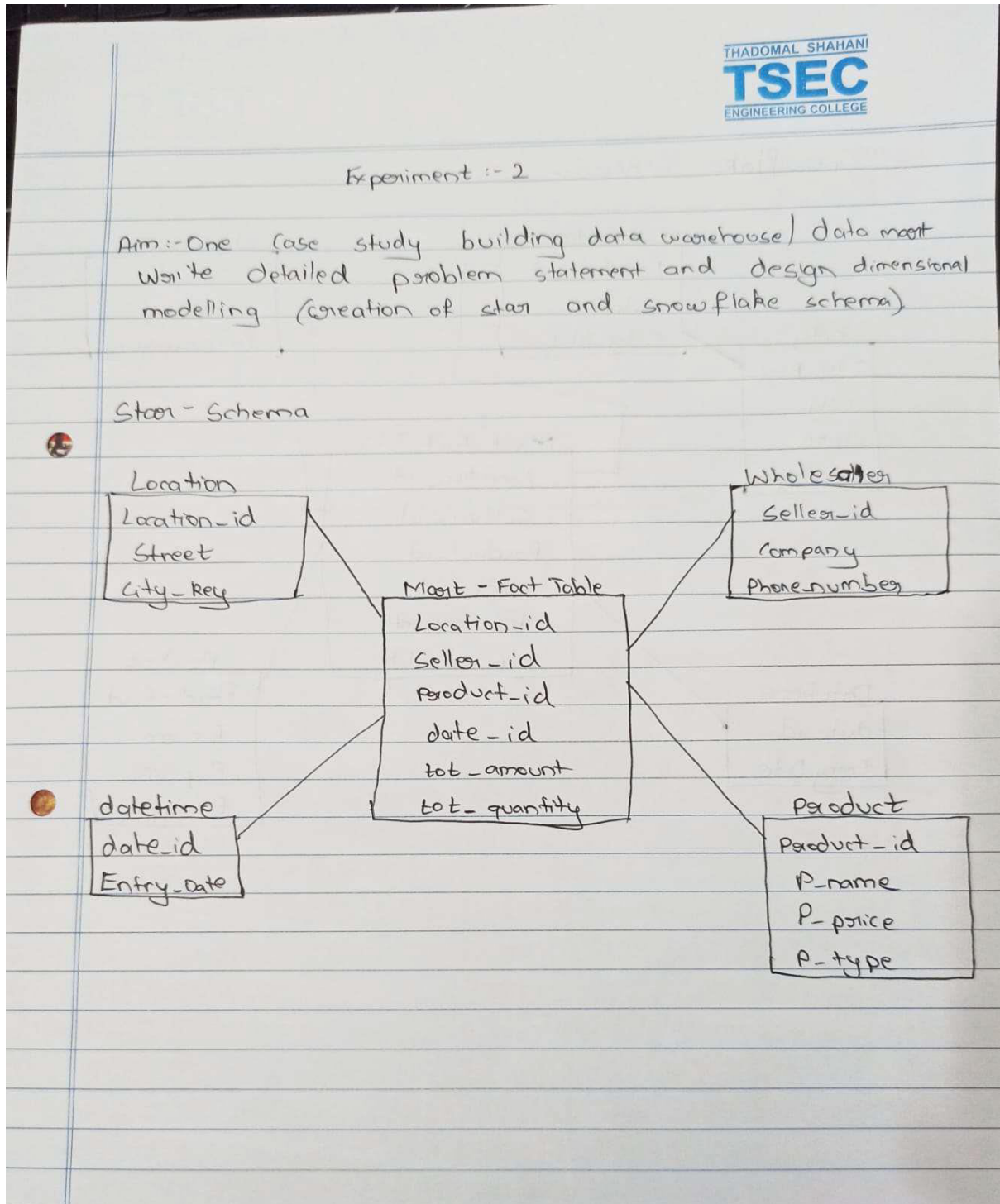
4. Benefits:

- Simplified Queries: Star schema simplifies querying by reducing the number of joins required for analysis. This enhances query performance and response time.

- Improved Understandability: The structure's clarity makes it easy for analysts to understand and navigate data relationships, facilitating effective analysis.

5. Drawbacks:

- Data Redundancy: Since dimension attributes are repeated in each row of the dimension table, there can be some redundancy in storage.



Snowflake schema:

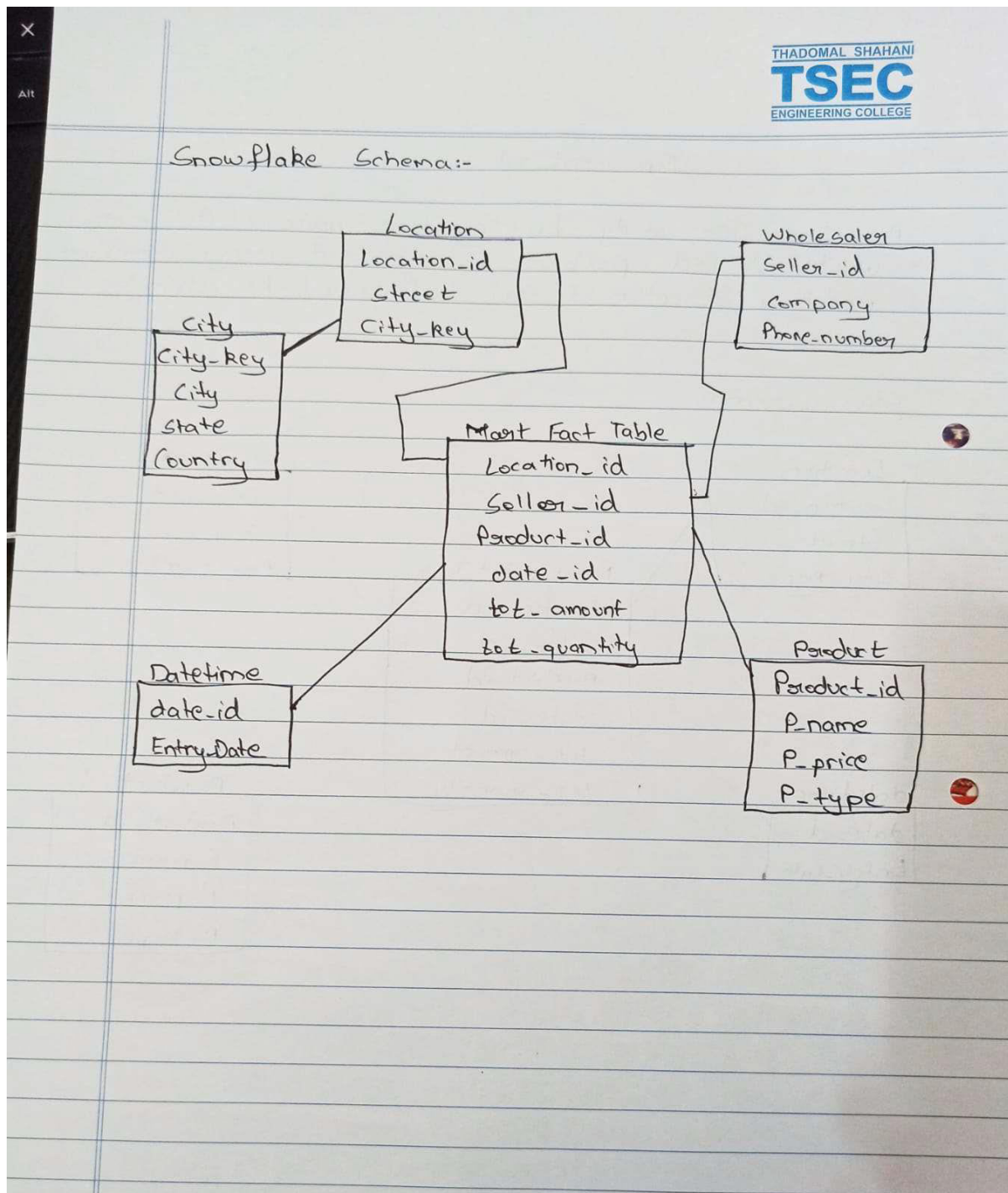
The Snowflake Schema is a normalized form of a multidimensional data model used in data warehousing and OLAP systems. It extends the concepts of the star schema and aims to reduce redundancy while enhancing data integrity and flexibility. The name "snowflake" comes from the shape of the schema when visualized, resembling a snowflake with its branches.

Key characteristics of the Snowflake Schema:

1. **Normalization:** Unlike the star schema, which denormalizes data into a single fact table and dimension tables, the snowflake schema further normalizes dimension tables. This means breaking down dimensions into sub-dimensions and related tables, reducing data redundancy.
2. **Hierarchical Structure:** The schema's structure resembles a hierarchy. Dimension tables are broken down into sub-dimensions and related attributes, forming a branching structure similar to the branches of a snowflake.
3. **Reduced Redundancy:** By normalizing data, the snowflake schema minimizes data duplication. This approach optimizes storage and maintains data consistency, reducing the chances of anomalies and update anomalies that can occur with redundant data.
4. **Data Integrity:** The normalized structure enhances data integrity by eliminating anomalies and inconsistencies associated with redundant data. Updates or modifications are made in fewer places, ensuring accuracy and reliability.
5. **Flexibility:** Snowflake schema allows for more flexible changes to attributes and relationships without impacting other parts of the schema. This makes it easier to adapt to evolving business needs.
6. **Complex Queries:** While snowflake schema promotes data integrity, it can lead to more complex query structures due to the need for multiple joins between related tables. This can affect query performance compared to simpler schemas like the star schema.
7. **Space Efficiency:** The normalized structure can lead to improved storage efficiency, especially for large datasets with repetitive attribute values.

8. Maintenance: Snowflake schema might require more complex maintenance due to the larger number of tables and relationships. However, modern database management tools have made managing such structures more manageable.

The Snowflake Schema is suitable when data integrity and accuracy are of prime importance, and storage efficiency is a consideration. It is often used in scenarios where frequent updates and changes are required, while maintaining data consistency. While it offers advantages in terms of data normalization and integrity, it is important to consider the trade-offs in query complexity and performance when choosing between different schema designs for a data warehouse or OLAP system.



Conclusion:

In conclusion, the Snowflake Schema offers a normalized and hierarchical approach to organizing data in data warehousing and OLAP systems. While enhancing data integrity, minimizing redundancy, and enabling flexibility, it introduces complexities in query structures and maintenance.

Organizations must weigh the benefits of improved data consistency against potential trade-offs in query performance when choosing the Snowflake Schema for their data modelling needs.

Experiment No. 3

Aim: Implementation of all dimension table and fact table based on experimental case study.

Theory:

Our experimental case study topic is “Restaurant Management System”. Implementing dimension tables and fact tables on them, basically we are implementing the following,

The structured SQL schema presented here models a restaurant's data using the star schema approach, a popular design for data warehousing. The star schema involves a central fact table, in this case named Restaurant, and dimension tables such as Customers, Menu_Items, Payment_Methods, Time, Locations, and Employees. Each dimension table holds unique attributes, while the fact table contains measurable metrics.

The Customers table captures customer information, including IDs, names, phone numbers, and email addresses. The Menu_Items table stores menu item details like IDs, names, categories, and prices. The Payment_Methods table records various payment methods for transactions.

The Time table provides temporal data, featuring IDs, order dates, months, quarters, and years. This allows for time-based analysis. The Locations table records location details, encompassing IDs, cities, states, and countries. The Employees table holds employee data, including IDs, names, departments, and join dates.

The central Restaurant fact table incorporates data from various dimensions. It includes order IDs, total amounts, references to customer IDs, menu item IDs, location IDs, employee IDs, and time IDs. These foreign keys link to the relevant entries in dimension tables.

This schema facilitates efficient querying and analysis for business intelligence purposes. The star schema's clear separation of facts and dimensions streamlines complex queries, providing a foundation for insightful reports and data-driven decisions in the restaurant domain.

Code:

Creating Tables:

1. Creating the tables

```
create table city(  
city_key int primary key,
```



```
city varchar2(20),
state varchar2(20),
country varchar2(20)
);
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.06 seconds

```
create table location(
location_id int primary key,
street varchar2(20),
city_key references city(city_key)
);
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.06 seconds

```
create table wholesaler(
seller_id int primary key,
company varchar2(20),
phone_number varchar2(20)
);
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.06 seconds

```
create table datetime(
date_id int primary key,
EntryDate TIMESTAMP
);
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.06 seconds

```
create table product(  
product_id int primary key,  
p_name varchar2(255),  
p_price int,  
p_type varchar2(20)  
);
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.06 seconds

```
create table mart(  
location_id REFERENCES location(location_id),  
seller_id REFERENCES wholesaler(seller_id),  
date_id REFERENCES datetime(date_id),  
product_id REFERENCES product(product_id),  
tot_amount int,  
tot_quantity int  
);
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Table created.

0.06 seconds

2. Inserting values in the tables created

```
insert into wholesaler values(1,'Lalwani Enterprises','1234567890');  
insert into wholesaler values(2,'Kursija Enterprises','0987654321');  
insert into wholesaler values(3,'Kumar Enterprises','5678901234');  
insert into wholesaler values(4,'Your Mart Wholesale','5433167899');
```

```

insert into wholesaler values(5,'Proper Wholesale','6789543321');
insert into wholesaler values(6,'High Cost Wholesale','1122334455');
insert into wholesaler values(7,'Wholesale Dealers','6677889900');
insert into wholesaler values(8,'Crafted Clothing','0678951234');
insert into wholesaler values(9,'Lifestyle Wholesale','2134590876');
insert into wholesaler values(10,'Super Star Wholesale','9087651234');

```

Results Explain Describe Saved SQL History

1 row(s) inserted.

0.00 seconds

Desc wholesaler;

Results Explain **Describe** Saved SQL History

Object Type **TABLE** Object **WHOLESALER**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
WHOLESALER	SELLER_ID	NUMBER	22	-	0	1	-	-	-
	COMPANY	VARCHAR2	20	-	-	-	✓	-	-
	PHONE_NUMBER	VARCHAR2	20	-	-	-	✓	-	-
1 - 3									

select * from wholesaler;

Results Explain Describe Saved SQL History

SELLER_ID	COMPANY	PHONE_NUMBER
1	Lalwani Enterprises	1234567890
2	Kursija Enterprises	0987654321
3	Kumar Enterprises	5678901234
4	Your Mart Wholesale	5433167899
5	Proper Wholesale	6789543321
6	High Cost Wholesale	1122334455
7	Wholesale Dealers	6677889900
8	Crafted Clothing	0678951234
9	Lifestyle Wholesale	2134590876
10	Super Star Wholesale	9087651234

10 rows returned in 0.00 seconds

[Download](#)

```

insert into city values(1,'Mumbai','Maharashtra','India');
insert into city values(2,'Ahemdabad','Gujarat','India');

```

```

insert into city values(3,'Udaipur','Rajasthan','India');
insert into city values(4,'Jaipur','Rajasthan','India');
insert into city values(5,'Patna','Bihar','India');
insert into city values(6,'Amritsar','Punjab','India');
insert into city values(7,'Pune','Maharashtra','India');
insert into city values(8,'Guwahati','Assam','India');
insert into city values(9,'Kolkata','West Bengal','India');
insert into city values(10,'Agra','Uttar Pradesh','India');

```

Results Explain Describe Saved SQL History

1 row(s) inserted.

0.00 seconds

Desc city;

Results Explain **Describe** Saved SQL History

Object Type **TABLE** Object **CITY**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
<u>CITY</u>	<u>CITY_KEY</u>	NUMBER	22	-	0	1	-	-	-
	<u>CITY</u>	VARCHAR2	20	-	-	-	✓	-	-
	<u>STATE</u>	VARCHAR2	20	-	-	-	✓	-	-
	<u>COUNTRY</u>	VARCHAR2	20	-	-	-	✓	-	-
1 - 4									

select * from city;

Results Explain Describe Saved SQL History

CITY_KEY	CITY	STATE	COUNTRY
1	Mumbai	Maharashtra	India
2	Ahemdabad	Gujarat	India
3	Udaipur	Rajasthan	India
4	Jaipur	Rajasthan	India
5	Patna	Bihar	India
6	Amritsar	Punjab	India
7	Pune	Maharashtra	India
8	Guwahati	Assam	India
9	Kolkata	West Bengal	India
10	Agra	Uttar Pradesh	India

10 rows returned in 0.00 seconds

[Download](#)

```

insert into location values(1,'Peddar Road',1);
insert into location values(2,'Carter Road',1);
insert into location values(3,'Manek Chowk',2);
insert into location values(4,'IIM Road',2);
insert into location values(5,'City Palace Road',3);
insert into location values(6,'Rani Road',3);
insert into location values(7,'Hawa Mahal Road',4);
insert into location values(8,'JLN Marg',4);
insert into location values(9,'Exhibition Road',5);
insert into location values(10,'Patna City',5);
insert into location values(11,'Hall Bazaar',6);
insert into location values(12,'Mall Road',6);
insert into location values(13,'MG Road',7);
insert into location values(14,'Kalyani Nagar',7);
insert into location values(15,'G.S. Road',8);
insert into location values(16,'A.T. Road',8);
insert into location values(17,'Park Street',9);
insert into location values(18,'Lake Road',9);
insert into location values(19,'Agra Fort Area',10);
insert into location values(20,'M.G. Road',10);

```

Results Explain Describe Saved SQL History

1 row(s) inserted.

0.00 seconds

Desc location;

Results Explain **Describe** Saved SQL History

Object Type **TABLE** Object **LOCATION**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
LOCATION	LOCATION_ID	NUMBER	22	-	0	1	-	-	-
	STREET	VARCHAR2	20	-	-	-	✓	-	-
	CITY_KEY	NUMBER	22	-	0	-	✓	-	-
1 - 3									

select * from location;

Results Explain Describe Saved SQL History

LOCATION_ID	STREET	CITY_KEY
1	Peddar Road	1
2	Carter Road	1
3	Manek Chowk	2
4	IIM Road	2
5	City Palace Road	3
6	Rani Road	3
7	Hawa Mahal Road	4
8	JLN Marg	4
9	Exhibition Road	5
10	Patna City	5

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.01 seconds

[Download](#)

```
insert into datetime values(1, TO_TIMESTAMP('2023-09-24 08:30:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(2, TO_TIMESTAMP('2023-07-14 18:30:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(3, TO_TIMESTAMP('2022-12-31 09:30:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(4, TO_TIMESTAMP('2022-10-06 09:00:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(5, TO_TIMESTAMP('2023-09-24 11:11:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(6, TO_TIMESTAMP('2022-12-15 10:30:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(7, TO_TIMESTAMP('2023-08-12 16:45:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(8, TO_TIMESTAMP('2022-10-14 19:23:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(9, TO_TIMESTAMP('2023-01-01 22:48:00', 'YYYY-MM-DD HH24:MI:SS'));
insert into datetime values(10, TO_TIMESTAMP('2023-04-16 17:56:00', 'YYYY-MM-DD HH24:MI:SS'));
```

Results Explain Describe Saved SQL History

1 row(s) inserted.

0.00 seconds

Desc datetime;

Results	Explain	Describe	Saved SQL	History					
Object Type TABLE Object DATETIME									
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
DATETIME	DATE_ID	NUMBER	22	-	0	1	-	-	-
	ENTRYDATE	TIMESTAMP(6)	11	-	6	-	✓	-	-
1 - 2									

select * from datetime;

Results Explain Describe Saved SQL History

DATE_ID	ENTRYDATE
1	24-SEP-23 08.30.00.000000 AM
2	14-JUL-23 06.30.00.000000 PM
3	31-DEC-22 09.30.00.000000 AM
5	24-SEP-23 11.11.00.000000 AM
4	06-OCT-22 09.00.00.000000 AM
6	15-DEC-22 10.30.00.000000 AM
7	12-AUG-23 04.45.00.000000 PM
8	14-OCT-22 07.23.00.000000 PM
9	01-JAN-23 10.48.00.000000 PM
10	16-APR-23 05.56.00.000000 PM

10 rows returned in 0.00 seconds Download

insert into product values(1,'Notebooks',200,'Educational Material');
insert into product values(2,'Bags',250,'Backpacks');
insert into product values(3,'Kitechenware',500,'Household Material');
insert into product values(4,'Shoes',800,'Footwear');
insert into product values(5,'Sandals',400,'Footwear');
insert into product values(6,'Television',900,'Household Material');
insert into product values(7,'Mobile',10200,'Electronic Gadgets');
insert into product values(8,'Dinner set',1500,'Kitchen');
insert into product values(9,'Camera Bag',600,'Backpacks');
insert into product values(10,'Dry Snacks',200,'Grocery');

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

1 row(s) inserted.

0.00 seconds

Desc product;

Results Explain Describe Saved SQL History

Object Type TABLE Object PRODUCT

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
PRODUCT	PRODUCT_ID	NUMBER	22	-	0	1	-	-	-
	P_NAME	VARCHAR2	255	-	-	-	✓	-	-
	P_PRICE	NUMBER	22	-	0	-	✓	-	-
	P_TYPE	VARCHAR2	20	-	-	-	✓	-	-
1 - 4									

select * from product;

Results	Explain	Describe	Saved SQL	History
PRODUCT_ID	P_NAME	P_PRICE	P_TYPE	
1	Notebooks	200	Educational Material	
2	Bags	250	Backpacks	
4	Shoes	800	Footwear	
5	Sandals	400	Footwear	
6	Television	900	Household Material	
7	Mobile	10200	Electronic Gadgets	
8	Dinner set	1500	Kitchen	
9	Camera Bag	600	Backpacks	
10	Dry Snacks	200	Grocery	
11	Cricket Bat	1000	Sports	
More than 10 rows available. Increase rows selector to view more rows.				
10 rows returned in 0.00 seconds Download				

```
insert into mart values(1,1,1,1,1000,5);
insert into mart values(2,2,2,2,1500,6);
insert into mart values(3,3,3,3,2000,4);
insert into mart values(4,4,4,4,2400,3);
insert into mart values(5,5,5,5,2000,5);
insert into mart values(6,6,6,6,6300,7);
insert into mart values(7,7,7,7,30600,3);
insert into mart values(8,8,8,8,10500,7);
insert into mart values(9,9,9,9,1800,3);
insert into mart values(10,10,10,10,3000,15);
insert into mart values(1,2,1,1,1200,6);
insert into mart values(2,3,10,3,1000,5);
insert into mart values(3,4,9,1,5000,5);
```


Results Explain Describe Saved SQL History

1 row(s) inserted.

0.00 seconds

Desc mart;

Results Explain **Describe** Saved SQL History

Object Type **TABLE** Object **MART**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
<u>MART</u>	<u>LOCATION_ID</u>	NUMBER	22	-	0	-	✓	-	-
	<u>SELLER_ID</u>	NUMBER	22	-	0	-	✓	-	-
	<u>DATE_ID</u>	NUMBER	22	-	0	-	✓	-	-
	<u>PRODUCT_ID</u>	NUMBER	22	-	0	-	✓	-	-
	<u>TOT_AMOUNT</u>	NUMBER	22	-	0	-	✓	-	-
	<u>TOT_QUANTITY</u>	NUMBER	22	-	0	-	✓	-	-
1 - 6									

select * from mart;

Results Explain Describe Saved SQL History

LOCATION_ID	SELLER_ID	DATE_ID	PRODUCT_ID	TOT_AMOUNT	TOT_QUANTITY
1	1	1	1	1000	5
2	2	2	2	1500	6
3	3	3	3	2000	4
4	4	4	4	2400	3
5	5	5	5	2000	5
6	6	6	6	6300	7
7	7	7	7	30600	3
8	8	8	8	10500	7
9	9	9	9	1800	3
10	10	10	10	3000	15
More than 10 rows available. Increase rows selector to view more rows.					

10 rows returned in 0.00 seconds

[Download](#)

Conclusion: Thus, in this experiment, we have implemented all the dimension tables and fact table based on experimental case study which is “Restaurant Management System”.

Experiment No. 4

Aim: Implementation of OLAP operations: Slice, Dice, Rollup, Drilldown and Pivot based on experiments.

Theory:

OLAP (Online Analytical Processing) operations are fundamental techniques used to interact with and analyze multidimensional data. These operations allow users to extract meaningful insights from complex data sets by aggregating, filtering, and transforming data across different dimensions. Here are the explanations of some common OLAP operations:

1. Slice:

- Definition: Slicing involves selecting a single value for a particular dimension while keeping other dimensions unchanged. It creates a new sub cube by fixing one dimension's value and looking at the data within that context.

- Use Case: For example, in a sales dataset, you could slice the data to focus only on a specific time period (e.g., a particular month) while maintaining other dimensions like products, regions, and customers.

2. Dice:

- Definition: Dicing involves selecting a subset of values from multiple dimensions to create a sub cube. This operation allows you to focus on a specific part of the data by narrowing down multiple dimensions.

- Use Case: Continuing with the sales dataset, you could dice the data to analyze sales for a specific time period and a particular region, while still considering products and customers.

3. Rollup:

- Definition: Rollup involves summarizing data across one or more dimensions. It reduces the level of granularity in data by moving from detailed data to aggregated data.

- Use Case: In a time-based dataset, you could roll up data to analyze sales quarterly instead of monthly. This operation is used to obtain higher-level insights from the data.

4. Drilldown:

- Definition: Drilldown is the opposite of rollup. It involves breaking down aggregated data into more detailed levels by adding more dimensions to the view.

- Use Case: If you have quarterly sales data, you can drill down to see the monthly data within each quarter. This operation provides a more granular view of the data.

5. Pivot:

- Definition: Pivoting involves rotating the data view to view it from a different perspective. It reorients dimensions, essentially changing rows to columns and columns to rows.

- Use Case: Consider a sales dataset with products, regions, and months. By pivoting, you can transform the data to view sales per product across different regions, making it easier to compare product performance.

These OLAP operations empower analysts and decision-makers to analyze complex datasets from various angles and levels of detail. By combining these operations, users can gain valuable insights into trends, patterns, and anomalies in multidimensional data, enabling informed decision-making and strategic planning.

Code:

Code for OLAP operations (Roll-Up, Drill-Down, Slice, and Dice) implemented using SQL queries on the provided restaurant data schema:

i. Slice:

```
select * from mart where date_id in(select date_id from datetime where  
extract(month from EntryDate)=9);
```

Results

Explain

Describe

Saved SQL

History

LOCATION_ID	SELLER_ID	DATE_ID	PRODUCT_ID	TOT_AMOUNT	TOT_QUANTITY
1	1	1	1	1000	5
5	5	5	5	2000	5
1	2	1	1	1200	6

3 rows returned in 0.01 seconds

Download

```
select * from mart where product_id in(select product_id from product where  
p_type='Educational Material');
```

Results Explain Describe Saved SQL History

LOCATION_ID	SELLER_ID	DATE_ID	PRODUCT_ID	TOT_AMOUNT	TOT_QUANTITY
1	1	1	1	1000	5
1	2	1	1	1200	6
3	4	9	1	5000	5

3 rows returned in 0.01 secondsDownload

ii. Dice:

```
select * from mart where location_id in(select location_id from location where  
city_key in(select city_key from city where city='Mumbai' AND  
state='Maharashtra'));
```

Results Explain Describe Saved SQL History

LOCATION_ID	SELLER_ID	DATE_ID	PRODUCT_ID	TOT_AMOUNT	TOT_QUANTITY
1	1	1	1	1000	5
2	2	2	2	1500	6
1	2	1	1	1200	6
2	3	10	3	1000	5

4 rows returned in 0.01 seconds Download

```
select * from mart where product_id in(select product_id from product where  
p_type='Educational Material' AND p_price=200);
```

Results Explain Describe Saved SQL History

LOCATION_ID	SELLER_ID	DATE_ID	PRODUCT_ID	TOT_AMOUNT	TOT_QUANTITY
1	1	1	1	1000	5
1	2	1	1	1200	6
3	4	9	1	5000	5

3 rows returned in 0.00 seconds Download

iii. Rollup:

```
select SUM(tot_amount) AS total_amount, SUM(tot_quantity) AS total_quantity  
from mart  
join datetime on mart.date_id = datetime.date_id;
```

Results	Explain	Describe	Saved SQL	History
TOTAL_AMOUNT TOTAL_QUANTITY				
68300	74			

1 rows returned in 0.00 seconds [Download](#)

```
select EntryDate, SUM(tot_amount) AS total_amount, SUM(tot_quantity) AS  
total_quantity  
from mart  
join datetime on mart.date_id = datetime.date_id  
group by EntryDate;
```

Results Explain Describe Saved SQL History

ENTRYDATE	TOTAL_AMOUNT	TOTAL_QUANTITY
24-SEP-23 08.30.00.000000 AM	2200	11
31-DEC-22 09.30.00.000000 AM	2000	4
16-APR-23 05.56.00.000000 PM	4000	20
14-JUL-23 06.30.00.000000 PM	1500	6
15-DEC-22 10.30.00.000000 AM	6300	7
06-OCT-22 09.00.00.000000 AM	2400	3
24-SEP-23 11.11.00.000000 AM	2000	5
12-AUG-23 04.45.00.000000 PM	30600	3
14-OCT-22 07.23.00.000000 PM	10500	7
01-JAN-23 10.48.00.000000 PM	6800	8

10 rows returned in 0.00 seconds

[Download](#)

iv. Drilldown:

```
select p_type, sum(tot_amount) AS total_amount, sum(tot_quantity) AS
total_quantity
from mart
join product on mart.product_id=product.product_id
group by p_type;
```

Results Explain Describe Saved SQL History

P_TYPE	TOTAL_AMOUNT	TOTAL_QUANTITY
Footwear	4400	8
Electronic Gadgets	30600	3
Backpacks	3300	9
Kitchen	10500	7
Household Material	9300	16
Educational Material	7200	16
Grocery	3000	15

7 rows returned in 0.01 seconds

[Download](#)

```
select p_price, sum(tot_amount) AS total_amount, sum(tot_quantity) AS
total_quantity
from mart
join product on mart.product_id=product.product_id
group by p_price;
```

Results Explain Describe Saved SQL History

P_PRICE	TOTAL_AMOUNT	TOTAL_QUANTITY
400	2000	5
250	1500	6
600	1800	3
500	3000	9
10200	30600	3
200	10200	31
900	6300	7
800	2400	3
1500	10500	7

9 rows returned in 0.01 seconds

[Download](#)

v. Pivot:

```
select p_price ,
sum(case when p_type='Educational material' then tot_amount else 0 end) as
P1_Product,
sum(case when p_type='Footwear' then tot_amount else 0 end) as P2_Product,
sum(case when p_type='Backpacks' then tot_amount else 0 end) as P3_Product,
sum(case when p_type='Household Material' then tot_amount else 0 end) as
P4_Product
from mart
join product on mart.product_id=product.product_id
group by p_price;
```

Results Explain Describe Saved SQL History

P_PRICE	P1_PRODUCT	P2_PRODUCT	P3_PRODUCT	P4_PRODUCT
400	0	2000	0	0
250	0	0	1500	0
600	0	0	1800	0
500	0	0	0	3000
10200	0	0	0	0
200	0	0	0	0
900	0	0	0	6300
800	0	2400	0	0
1500	0	0	0	0

9 rows returned in 0.00 seconds

[Download](#)

Conclusion:

Thus, OLAP operations—Slice, Dice, Rollup, Drilldown, and Pivot—provide essential techniques to analyze multidimensional data from various perspectives. These operations allow users to extract insights, identify trends, and make informed decisions by manipulating data across dimensions. By fixing, aggregating, or reorienting data, these operations offer flexibility in exploring complex datasets and enabling data-driven actions that drive business success. Ultimately, OLAP operations serve as invaluable tools for organizations seeking to maximize the value of their data assets.

Experiment No. 5

Aim: Implementation of Bayesian algorithm.

Theory:

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that is why it is considered as naive. This assumption is called class conditional independence.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- $P(h)$: the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h .
- $P(D)$: the probability of the data (regardless of the hypothesis). This is known as the prior probability.
- $P(h|D)$: the probability of hypothesis h given the data D . This is known as posterior probability.
- $P(D|h)$: the probability of data d given that hypothesis h was true. This is

known as posterior probability.

How does Naive Bayes classifier work?

Let us understand the working of Naive Bayes through an example.

Given an example of weather conditions and playing sports. You need to calculate the probability of playing sports. Now, you need to classify whether players will play or not, based on the weather condition.

Approach (In case of a single feature)

Naive Bayes classifier calculates the probability of an event in the following steps:

- Step 1: Calculate the prior probability for given class labels
- Step 2: Find Likelihood probability with each attribute for each class
- Step 3: Put these values in Bayes Formula and calculate posterior probability.
- Step 4: See which class has a higher probability, given the input belongs to the higher probability class.

For simplifying prior and posterior probability calculation you can use the two tables frequency and likelihood tables. Both tables will help you to calculate the prior and posterior probability. The Frequency table contains the occurrence of labels for all features. There are two likelihood tables. Likelihood Table 1 is showing prior probabilities of labels and Likelihood Table 2 is showing the posterior probability.

Whether	Play
Sunny	No
Sunny	No
Overcast	Yes
Rainy	Yes
Rainy	Yes
Rainy	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rainy	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Whether	No	Yes
Overcast		4
Sunny	2	3
Rainy	3	2
Total	5	9

Likelihood Table 1				
Whether	No	Yes		
Overcast		4	=4/14	0.29
Sunny	2	3	=5/14	0.36
Rainy	3	2	=5/14	0.36
Total	5	9		
	=5/14	=9/14		
	0.36	0.64		

Likelihood Table 2				
Whether	No	Yes	Posterior Probability for No	Posterior Probability for Yes
Overcast		4	0/5=0	4/9=0.44
Sunny	2	3	2/5=0.4	3/9=0.33
Rainy	3	2	3/5=0.6	2/9=0.22
Total	5	9		

Now suppose you want to calculate the probability of playing when the weather is overcast.

Probability of playing:

$$P(\text{Yes} \mid \text{Overcast}) = P(\text{Overcast} \mid \text{Yes}) P(\text{Yes}) / P(\text{Overcast})$$

.....(1)

1. Calculate Prior Probabilities:

$$P(\text{Overcast}) = 4/14 = 0.29$$

$$P(\text{Yes}) = 9/14 = 0.64$$

2. Calculate Posterior Probabilities:

$$P(\text{Overcast} \mid \text{Yes}) = 4/9 = 0.44$$

3. Put Prior and Posterior probabilities in equation (1)

$$P(\text{Yes} \mid \text{Overcast}) = 0.44 * 0.64 / 0.29 = 0.98(\text{Higher})$$

Similarly, you can calculate the probability of not playing

The probability of a 'Yes' class is higher. So, you can determine here if the weather is overcast than players will play the sport.

Code:

```
import java.util.*;

class Data {
    int count = 0;
    Map < String, Integer > freq = new HashMap < > ();
    double probVariable = 1;
    void calculateProbability(String s) {
        Integer frequency = freq.get(s);
        if (frequency != null && count != 0) {
            probVariable *= frequency / (double) count;
        }
    }
}

public class BayesAlgo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of row and column:");
        int n = sc.nextInt();
        int m = sc.nextInt();
        sc.next();
        String data[][] = new String[n][m];
        for (int i = 0; i < n; i++) {
            String row = sc.nextLine();
            String[] values = row.split(" ");
            if (values.length != m) {
```

```

        System.out.println("Invalid input. Number of columns does not
            match.
        ");
        sc.close();
        return;
    }
    for (int j = 0; j < m; j++) {
        data[i][j] = values[j];
    }
}

System.out.println("Enter the column name for class variable:");
String classVariable = sc.next();
Map < String, Data > distinct = new HashMap < > ();
for (int i = 0; i < data[0].length; i++) {
    if (data[0][i].equals(classVariable)) {
        for (int j = 1; j < data.length; j++) {
            String classValue = data[j][i];
            if (!distinct.containsKey(classValue)) {
                distinct.put(classValue, new Data());
            }
            Data classData = distinct.get(classValue);
            classData.count++;
            for (int k = 0; k < data[0].length; k++) {
                if (k == i)
                    continue;
                int cnt = classData.freq.getDefault(data[j][k], 0);
                classData.freq.put(data[j][k], ++cnt);
            }
        }
    }
}

```

```

    }
}
}
System.out.println("Enter the number of inputs:");
int col = sc.nextInt();
System.out.println("Enter the tuple in columnValue:");
for (int i = 0; i < col; i++) {
    String s = sc.next();
    for (Map.Entry < String, Data > it: distinct.entrySet()) {
        it.getValue().calculateProbability(s);
    }
}
String ans = "";
double ansProb = 0;
double total = 0;
for (Map.Entry < String, Data > it: distinct.entrySet()) {
    total += (it.getValue().probVariable * (it.getValue().count * 1.0) / (n - 1));
    if ((it.getValue().probVariable * (it.getValue().count * 1.0) / (n - 1)) >
ansProb) {
        ans = it.getKey();
        ansProb = (it.getValue().probVariable * (it.getValue().count * 1.0) / (n -
1));
    }
}
System.out.println("\nFinal class:" + ans);
System.out.println("Probability:" + (ansProb / total));
sc.close();
}

```

}

Output:

Enter the number of row and column:

6

4

Outlook Temperature Humidity PlayTennis

Sunny Hot High No

Sunny Hot High Yes

Overcast Hot High No

Rainy Mild High No

Rainy Cool Normal Yes

Enter the column name for class variable:

PlayTennis

Enter the number of inputs:

2

Enter the tuple in columnValue:

Rainy Cool

Final class: No

Probability:0.6666666666666666

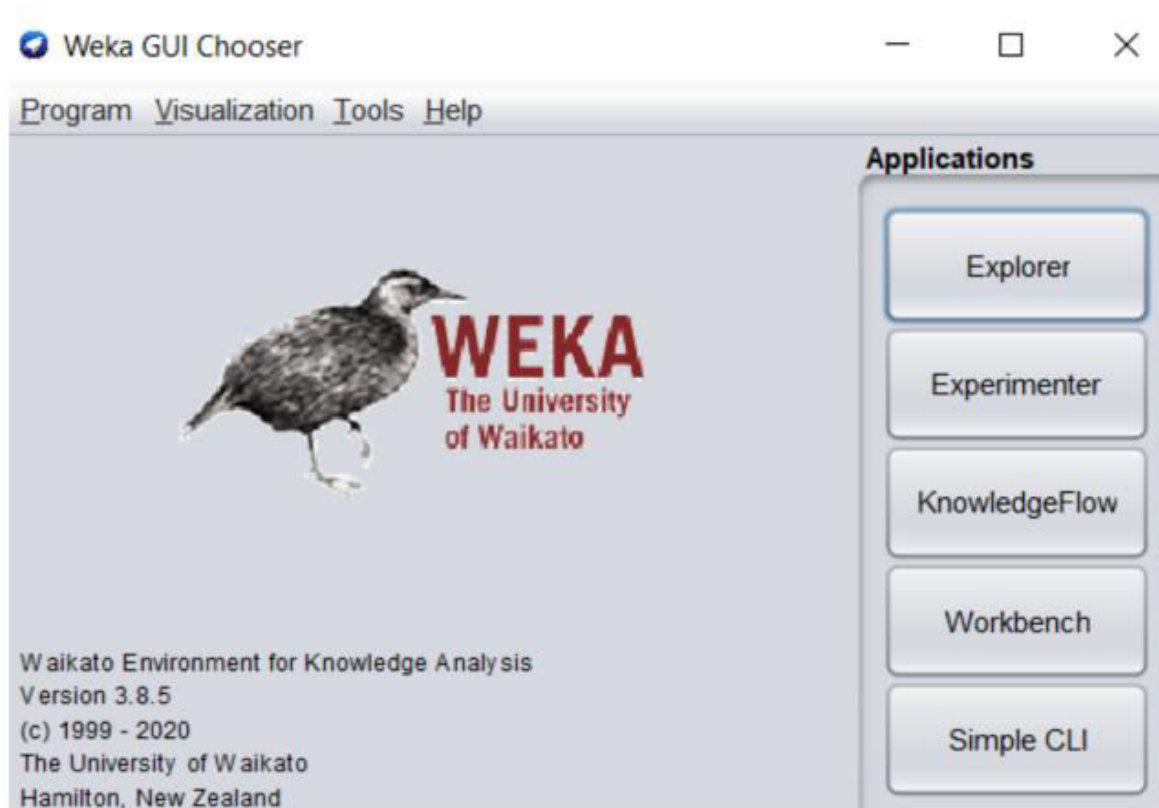
Conclusion: Thus, in this experiment, we have implemented Bayesian algorithm.

Experiment No. 6

Aim: Perform data Pre-processing task and demonstrate performing Classification, Clustering, Association algorithm on data sets using data mining tool (WEKA/R tool).

Theory:

The Weka GUI Chooser (class `weka.gui.GUIChooser`) provides a starting point for launching Weka's main GUI applications and supporting tools. If one prefers a MDI (—multiple document interface) appearance, then this is provided by an alternative launcher called —Main (class `weka.gui.Main`). The GUI Chooser consists of four buttons—one for each of the four major Weka applications— and four menus.



The buttons can be used to start the following applications:

Explorer - An environment for exploring data with WEKA

- a) Click on —explorer button to bring up the explorer window.
- b) Make sure the —preprocess tab is highlighted.
- c) Open a new file by clicking on —Open New file|| and choosing a file with —.arff|| extension from the —Data|| directory.
- d) Attributes appear in the window below.
- e) Click on the attributes to see the visualization on the right.
- f) Click —visualize all|| to see them all.

Experimenter - An environment for performing experiments and conducting statistical tests between learning schemes.

- a) Experimenter is for comparing results.
- b) Under the —set up tab click —New.
- c) Click on —Add New under —Data frame. Choose a couple of arff format files from —Data|| directory one at a time.
- d) Click on —Add New under —Algorithm frame. Choose several algorithms, one at a time by clicking —OK in the window and —Add New.
- e) Under the —Run tab click —Start||
- f) Wait for WEKA to finish.
- g) Under —Analyses tab click on —Experiment to see results.

Knowledge Flow - This environment supports essentially the same functions as the Explorer but with a drag-and-drop interface. One advantage is that it supports incremental learning.

SimpleCLI - Provides a simple command-line interface that

allows direct execution of WEKA commands for operating systems that do not provide their own command line interface.

Navigate the options available in the WEKA (ex. Select attributes panel, Preprocess panel, classify panel, Cluster panel, Associate panel and Visualize panel)

When the Explorer is first started only the first tab is active; the others are greyed out. This is because it is necessary to open (and potentially pre-process) a data set before starting to explore the data.

The tabs are as follows:

1. Preprocess. Choose and modify the data being acted on.
2. Classify. Train and test learning schemes that classify or perform regression.
3. Cluster. Learn clusters for the data.
4. Associate. Learn association rules for the data.
5. Select attributes. Select the most relevant attributes in the data.
6. Visualize. View an interactive 2D plot of the data. Once the tabs are active, clicking on them flicks between different screens, on which the respective actions can be performed. The bottom area of the window (including the status box, the log button, and the Weka bird) stays visible regardless of which section you are in

Loading Data:

The first four buttons at the top of the preprocess section enable you to load data into WEKA:

1. Open file.... Brings up a dialog box allowing you to browse for the datafile on the local file system.
2. Open URL.... Asks for a Uniform Resource Locator address for where the data is stored.
3. Open DB.....Reads data from a database. (Note that to make this work you might have to edit the file in `weka/experiment/DatabaseUtils.props`.)
4. Generate.... Enables you to generate artificial data from a variety of `DataGenerators`.

Using the Open file ...button you can read files in a variety of formats:

WEKA's ARFF format, CSV format, C4.5 format, or serialized Instances format. ARFF files typically have a `.arff` extension, CSV files a `.csv` extension, C4.5 files a `.data` and `.names` extension, and serialized Instances objects a `.bsi` extension.

Output:

K-means Clustering:

Clustering:

Weka Explorer

Preprocess

Classify

Cluster

Associate

Select attributes

Visualize

Clusterer

Choose

SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.

Cluster mode

☒ Use training set

☐ Supplied test set

Set...

☐ Percentage split

% 66

☐ Classes to clusters evaluation

(Nom) class

☒ Store clusters for visualization

Ignore attributes

Start

Stop

Result list (right-click for options)

10:34:25 - SimpleKMeans

Clusterer output

```

Scheme:      weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000
Relation:    pima_diabetes
Instances:    768
Attributes:   9
    preg
    plas
    pres
    skin
    insu
    mass
    pedi
    age
    class
Test mode:    evaluate on training data

=== Clustering model (full training set) ===

KMeans
=====

Number of iterations: 13
Within cluster sum of squared errors: 127.7202829520244

Initial starting points (random):

Cluster 0: 1,126,56,29,152,28.7,0.801,21,tested_negative
Cluster 1: 8,95,72,0,0,36.8,0.485,57,tested_negative
Cluster 2: 1,97,66,15,140,23.2,0.487,22,tested_negative

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute          Full Data          Cluster#          0          1          2
                   (768.0)          (132.0)          (268.0)          (368.0)
=====
preg                3.8451            6.8561            4.8657            2.0217
plas               120.8945          118.6136          141.2575          106.8832
pres               69.1055           75.7197           70.8246           65.481
skin               20.5365           14.9545           22.1642           21.3533
insu               79.7995           40.7727           100.3358           78.8424
mass               31.9926           30.5152           35.1425           30.2285
pedi                0.4719            0.4172            0.5505            0.4342
age                33.2409           46.9091           37.0672           25.5516
class              tested_negative tested_negative tested_positive tested_negative

Time taken to build model (full training data) : 0.03 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      132 ( 17%)
1      268 ( 35%)
2      368 ( 48%)

```

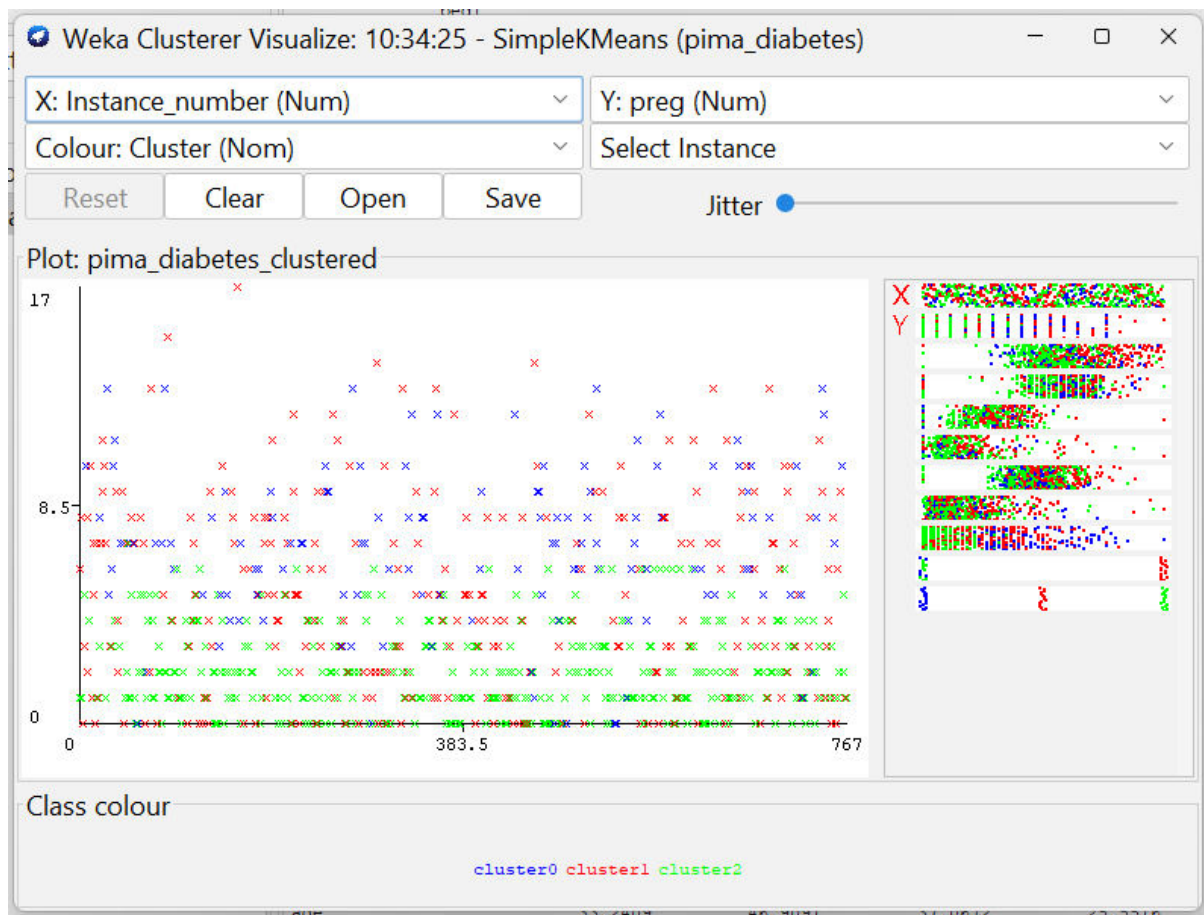
Status

OK

Log

x 0

Visualization:



Naïve Bayes Classification:

Classification:

Weka Explorer

Preprocess

Classify

Cluster

Associate

Select attributes

Visualize

Classifier

Choose

NaiveBayes

Test options

Use training set

Supplied test set

Set...

Cross-validation

Folds

10

Percentage split

%

66

More options...

(Nom) class

Start

Stop

Result list (right-click for options)

10:44:56 - bayes.NaiveBayes

Classifier output

sepal.length
sepal.width
petal.length
petal.width
class
Test mode: 10-fold cross-validation

=== Classifier model (full training set) ===

Naive Bayes Classifier

Attribute Class
 Iris-setosa Iris-versicolor Iris-virginica
 (0.33) (0.33) (0.33)
=====

sepal.length
 mean 4.9913 5.9379 6.5795
 std. dev. 0.355 0.5042 0.6353
 weight sum 50 50 50
 precision 0.1059 0.1059 0.1059

sepal.width
 mean 3.4015 2.7687 2.9629
 std. dev. 0.3925 0.3038 0.3088
 weight sum 50 50 50
 precision 0.1091 0.1091 0.1091

petal.length
 mean 1.4694 4.2452 5.5516
 std. dev. 0.1782 0.4712 0.5529
 weight sum 50 50 50
 precision 0.1405 0.1405 0.1405

petal.width
 mean 0.2743 1.3097 2.0343
 std. dev. 0.1096 0.1915 0.2646
 weight sum 50 50 50
 precision 0.1143 0.1143 0.1143

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances 144 96 %
Incorrectly Classified Instances 6 4 %
Kappa statistic 0.94
Mean absolute error 0.0342
Root mean squared error 0.155
Relative absolute error 7.6997 %
Root relative squared error 32.8794 %
Total Number of Instances 150

=== Detailed Accuracy By Class ===

 TP Rate FP Rate Precision Recall F-Measure MCC
 1.000 0.000 1.000 1.000 1.000 1.000
 0.960 0.040 0.923 0.960 0.941 0.911
 0.920 0.020 0.958 0.920 0.939 0.910
Weighted Avg. 0.960 0.020 0.960 0.960 0.960 0.940

=== Confusion Matrix ===

 a b c <-- classified as
50 0 0 | a = Iris-setosa
 0 48 2 | b = Iris-versicolor

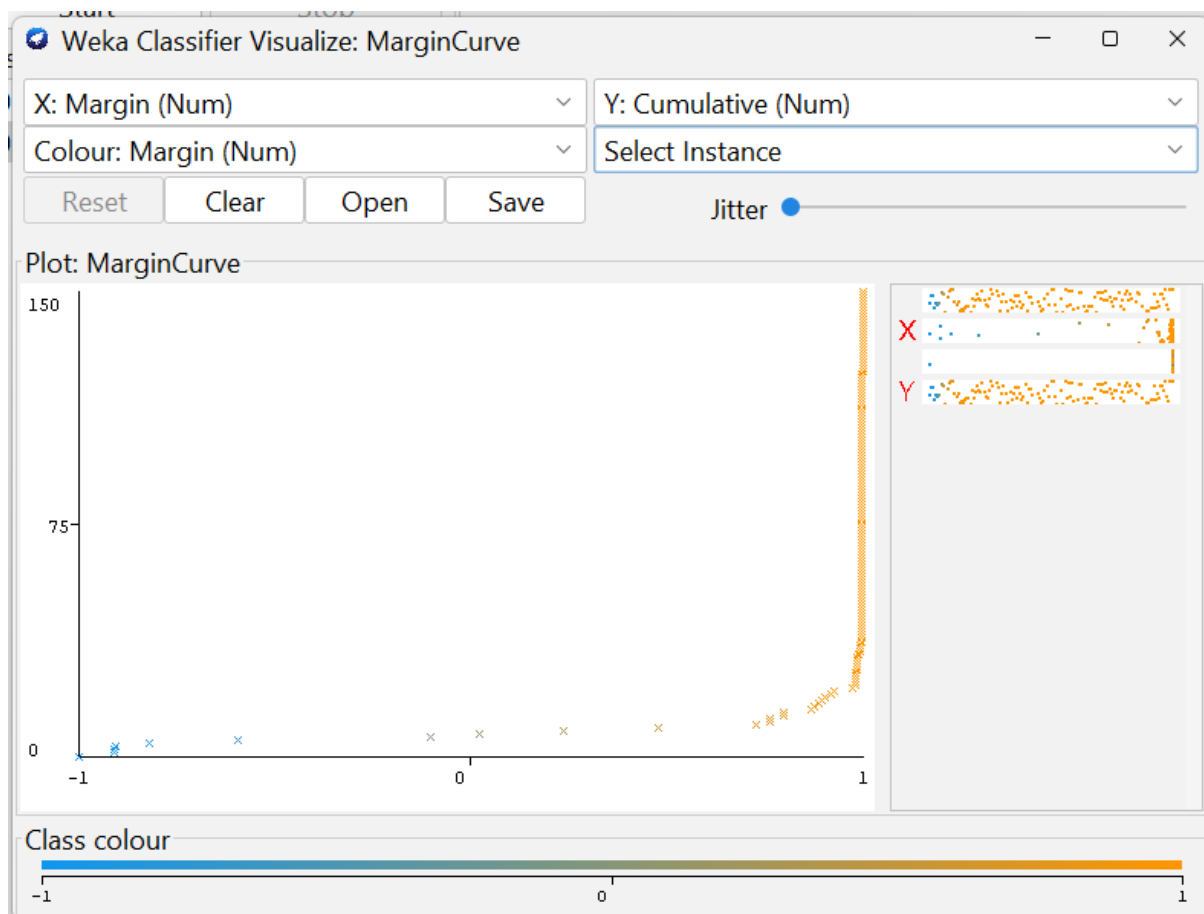
Status

OK

Log

x 0

Visualization:



Decision Tree Classification:

Classification:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier

Choose **J48 -C 0.25 -M 2**

Test options

☒ Use training set

☐ Supplied test set

☐ Cross-validation Folds

☐ Percentage split %

(Nom) class

Result list (right-click for options)

10:44:56 - bayes.NaiveBayes

10:48:54 - bayes.NaiveBayes

10:52:44 - trees.J48

10:56:02 - trees.J48

10:56:23 - trees.J48

10:57:36 - trees.J48

10:58:03 - trees.J48

11:00:26 - trees.J48

Classifier output

```

=== Run information ===

Scheme:      weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:    iris
Instances:   150
Attributes:  5
    sepallength
    sepalwidth
    petallength
    petalwidth
    class
Test mode:   evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree
-----
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (48.0/1.0)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3.0)
|   |   |   petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|   |   petalwidth > 1.7: Iris-virginica (46.0/1.0)

Number of Leaves   :    5
Size of the tree   :    9

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances      147          98      %
Incorrectly Classified Instances      3           2      %
Kappa statistic                    0.97
Mean absolute error                  0.0233
Root mean squared error              0.108
Relative absolute error              5.2482 %
Root relative squared error         22.9089 %
Total Number of Instances          150

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000    Iris-setosa
               0.980    0.020    0.961     0.980    0.970     0.955    0.990    0.969    Iris-versico
               0.960    0.010    0.980     0.960    0.970     0.955    0.990    0.970    Iris-virgini
Weighted Avg.    0.980    0.010    0.980     0.980    0.980     0.970    0.993    0.980


=== Confusion Matrix ===

  a  b  c  <-- classified as
50  0  0 | a = Iris-setosa
 0 49  1 | b = Iris-versicolor
 0  2 48 | c = Iris-virginica

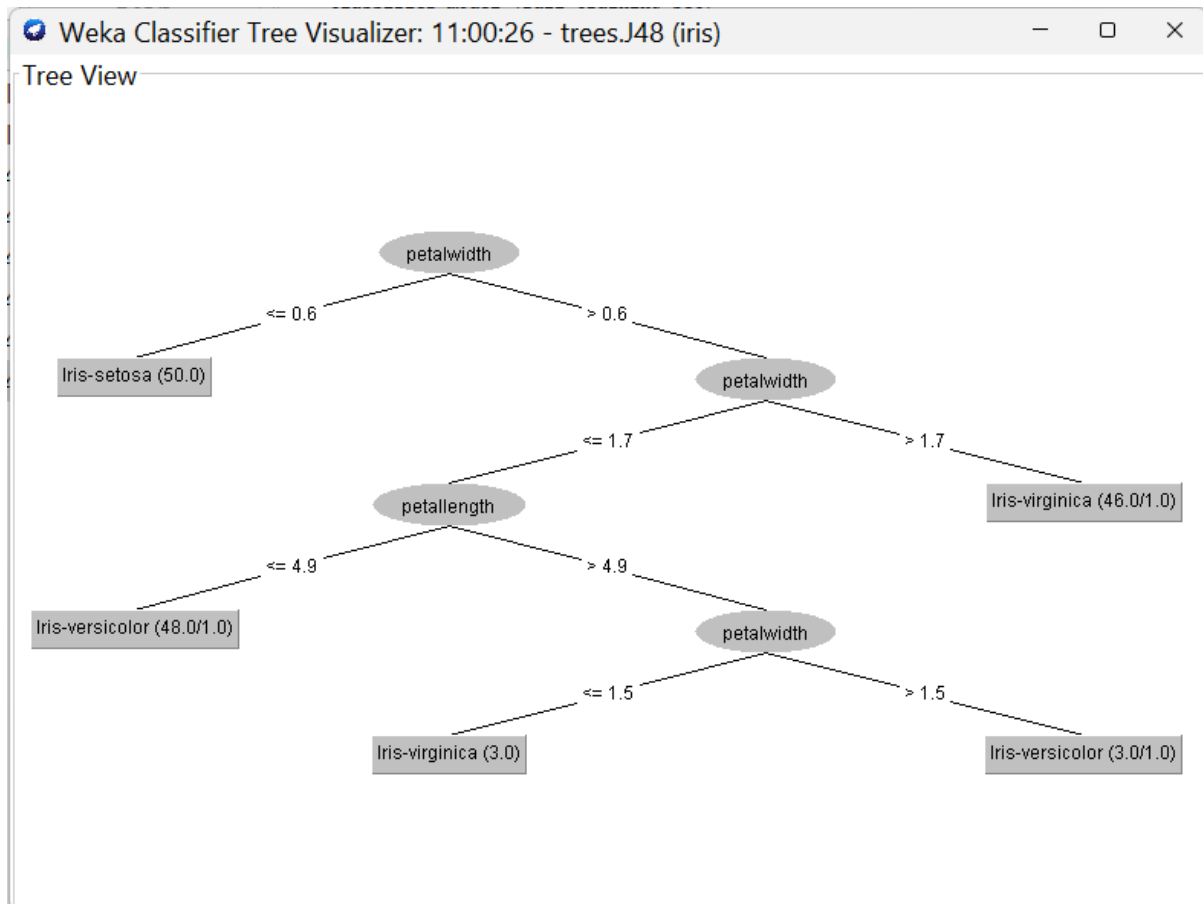
```

Status

OK

 x 0

Visualization:



Apriori:

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Associator

Choose **Apriori** -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Start Stop

Result list (right-click to expand)

11:04:15 - Apriori

Associator output

```
==== Run information ====
Scheme:      weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:    supermarket
Instances:    4627
Attributes:   217
              (list of attributes omitted)
==== Associator model (full training set) ====

Apriori
=====

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

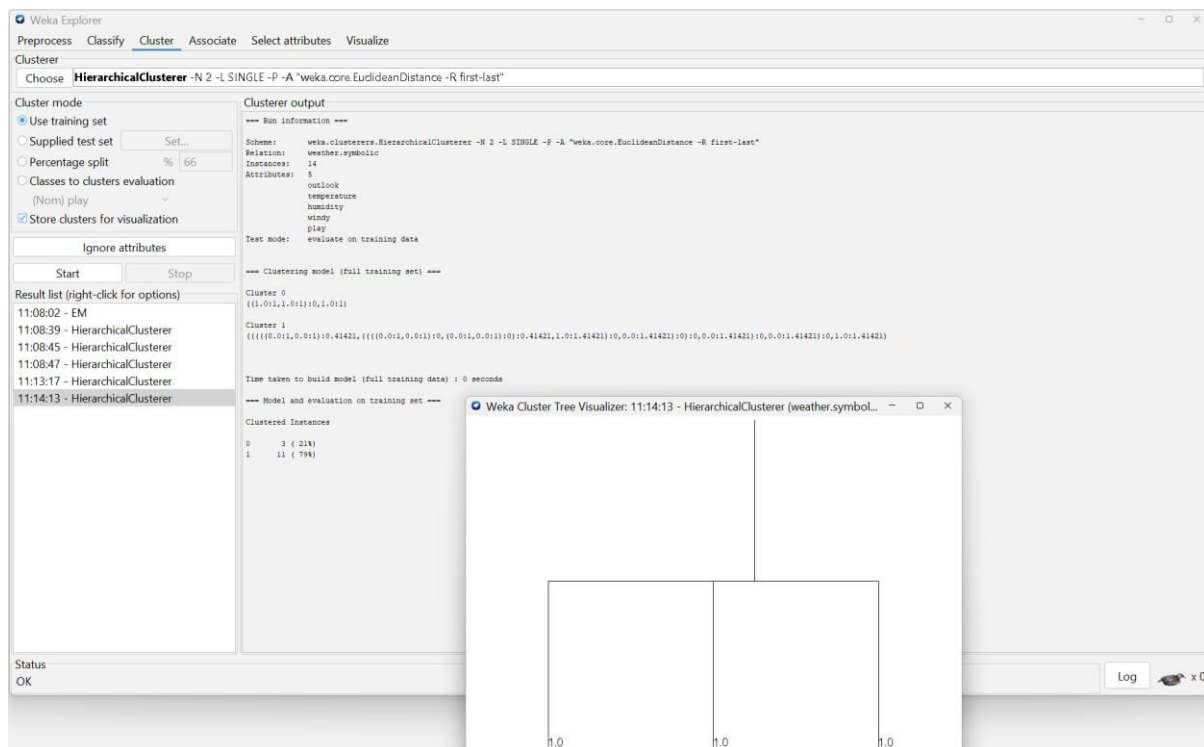
Size of set of large itemsets L(1): 44
Size of set of large itemsets L(2): 380
Size of set of large itemsets L(3): 910
Size of set of large itemsets L(4): 633
Size of set of large itemsets L(5): 105
Size of set of large itemsets L(6): 1

Best rules found:
1. biscuit=t frozen food=t fruit=t total=high 788 ==> bread and cake=t 723 <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
2. baking need=t biscuit=t fruit=t total=high 760 ==> bread and cake=t 696 <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.25)
3. baking need=t frozen food=t fruit=t total=high 770 ==> bread and cake=t 705 <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
4. biscuit=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746 <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
5. party snack food=t fruit=t total=high 854 ==> bread and cake=t 779 <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
6. biscuit=t frozen food=t vegetables=t total=high 797 ==> bread and cake=t 725 <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
7. baking need=t biscuit=t vegetables=t total=high 772 ==> bread and cake=t 701 <conf:(0.91)> lift:(1.26) lev:(0.03) [149] conv:(3.01)
8. biscuit=t fruit=t total=high 954 ==> bread and cake=t 866 <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
9. frozen food=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757 <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen food=t fruit=t total=high 969 ==> bread and cake=t 877 <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

Status OK

Log x 0

Agglomerative:



Conclusion: Thus, in this experiment, we have performed data Pre-processing task and demonstrated performing Classification, Clustering, Association algorithm on data sets using data mining tool (WEKA/R tool).

Experiment No. 7

Aim: Implementation of Clustering algorithm (K-means/K-medoids).

Theory:

K-Means is a popular clustering algorithm used for partitioning a dataset into distinct groups or clusters based on similarity. The goal of K-Means is to assign each data point to one of K clusters in a way that minimizes the total within-cluster variance. It is an iterative algorithm that gradually refines the cluster assignments by optimizing a cost function.

Algorithm Steps:

1. Initialization: Choose the number of clusters, K, and initialize K cluster centroids randomly or using a specific strategy, such as randomly selecting K data points as initial centroids.
2. Assignment: Assign each data point to the nearest cluster centroid based on a distance metric, commonly using Euclidean distance.
3. Update: Recalculate the centroids of each cluster by taking the mean of all data points assigned to that cluster.
4. Convergence Check: Check if the centroids have changed significantly from the previous iteration. If they have not, or a predetermined number of iterations has been reached, the algorithm stops.
5. Repeat Assignment and Update: Repeat the assignment and update steps iteratively until convergence or until a maximum number of iterations is reached.

Key Concepts:

- **Cluster Centroid:** Each cluster is represented by its centroid, which is the mean of all data points assigned to that cluster.
- **Within-Cluster Variance:** The total within-cluster variance is the sum of squared distances between each data point and its cluster centroid. Minimizing this variance results in tighter, more coherent clusters.
- **Choosing K:** The choice of the number of clusters, K, is critical. One common approach is the "elbow method," where the within-cluster variance is plotted for different values of K. The "elbow point" in the plot signifies a good trade-off between fitting the data closely and avoiding overfitting.

- Initialization Impact: The algorithm's final solution can be sensitive to the initial centroid placement. Poor initialization can lead to suboptimal solutions or slow convergence. Strategies like K-Means++ aim to mitigate this issue by initializing centroids in a smart way.

Advantages:

- Efficient for large datasets.
- Simple to implement and understand.
- Scales well to high-dimensional data.
- Can be applied to various types of data.

Limitations:

- Requires the number of clusters K to be specified beforehand.
- Sensitive to initial centroid placement, which can lead to convergence to local optima.
- Assumes clusters are spherical and equally sized, which might not always hold true.
- Not suitable for non-linear or complex cluster shapes.
- Prone to outliers, as they can significantly affect centroid placement.

Code:

```
import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

class Point {

    double x, y;

    public Point(double x, double y) {

        this.x = x;

        this.y = y;

    }

    // Override equals and hashCode methods to compare
```

points by rounded values

@Override

```
public boolean equals(Object obj) {  
    if (this == obj) return true;  
    if (obj == null || getClass() != obj.getClass())  
        return false;  
    Point other = (Point) obj;  
    return Double.compare(Math.round(x),  
        Math.round(other.x)) == 0 &&  
        Double.compare(Math.round(y),  
            Math.round(other.y)) == 0;  
}
```

@Override

```
public int hashCode() {  
    return Double.hashCode(Math.round(x)) +  
        Double.hashCode(Math.round(y));  
}  
}
```

```
class Cluster {  
    Point centroid;  
    List<Point> points;  
    public Cluster(Point centroid) {  
        this.centroid = centroid;  
        this.points = new ArrayList<>();  
    }  
    public void clearPoints() {  
        points.clear();  
    }  
    public void addPoint(Point point) {  
        points.add(point);  
    }  
}
```

```

}

public class KMeans {

private int k;

private int maxIterations;

private List<Point> data;

private List<Cluster> clusters;

public KMeans(int k, int maxIterations, List<Point>
data) {

this.k = k;

this.maxIterations = maxIterations;

this.data = data;

this.clusters = new ArrayList<>();

}

public void initializeClusters() {

Random rand = new Random();

if (data.size() < k) {

throw new IllegalArgumentException("Not enough
data points to initialize " + k + " clusters.");

}

for (int i = 0; i < k; i++) {

Point randomPoint =
data.get(rand.nextInt(data.size()));

Cluster cluster = new Cluster(randomPoint);

clusters.add(cluster);

}

}

public double distance(Point p1, Point p2) {

return Math.sqrt(Math.pow(p1.x - p2.x, 2) +
Math.pow(p1.y - p2.y, 2));

}

public void assignPointsToClusters() {

```

```

for (Cluster cluster : clusters) {
    cluster.clearPoints();
}
for (Point point : data) {
    Cluster nearestCluster = null;
    double minDistance = Double.MAX_VALUE;
    for (Cluster cluster : clusters) {
        double d = distance(point,
            cluster.centroid);
        if (d < minDistance) {
            minDistance = d;
            nearestCluster = cluster;
        }
    }
    if (nearestCluster != null) {
        nearestCluster.addPoint(point);
    }
}
// Reassign unassigned points
for (Point point : data) {
    if (!isAssigned(point)) {
        Cluster nearestCluster = null;
        double minDistance = Double.MAX_VALUE;
        for (Cluster cluster : clusters) {
            double d = distance(point,
                cluster.centroid);
            if (d < minDistance) {
                minDistance = d;
                nearestCluster = cluster;
            }
        }
    }
}

```

```

if (nearestCluster != null) {
    nearestCluster.addPoint(point);
}
}
}
}

public boolean isAssigned(Point point) {
    for (Cluster cluster : clusters) {
        if (cluster.points.contains(point)) {
            return true;
        }
    }
    return false;
}

public void updateClusterCentroids() {
    for (Cluster cluster : clusters) {
        double sumX = 0;
        double sumY = 0;
        List<Point> points = cluster.points;
        int numPoints = points.size();
        for (Point point : points) {
            sumX += point.x;
            sumY += point.y;
        }
        if (numPoints > 0) {
            cluster.centroid.x = sumX / numPoints;
            cluster.centroid.y = sumY / numPoints;
        }
    }
}

public void runKMeans() {

```

```

initializeClusters();

for (int iteration = 0; iteration < maxIterations;
iteration++) {
    assignPointsToClusters();
    updateClusterCentroids();
}
}

public List<Cluster> getClusters() {
    return clusters;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    List<Point> data = new ArrayList<>();
    List<Point> originalData = new ArrayList<>();
    System.out.println("Enter the number of data
points:");
    int numPoints = scanner.nextInt();
    for (int i = 0; i < numPoints; i++) {
        System.out.println("Enter data point " + (i +
1) + " (x y):");
        double x = scanner.nextDouble();
        double y = scanner.nextDouble();
        Point point = new Point(x, y);
        data.add(point);
        originalData.add(point);
    }
    System.out.println("\nEnter the number of clusters
(k):");
    int k = scanner.nextInt();
    System.out.println("\nEnter the maximum number of
iterations:");

```



```

int maxIterations = scanner.nextInt();

KMeans kMeans = new KMeans(k, maxIterations,
data);

kMeans.runKMeans();

List<Cluster> clusters = kMeans.getClusters();

for (int i = 0; i < clusters.size(); i++) {

Cluster cluster = clusters.get(i);

System.out.println("\nCluster " + (i + 1) + "

Centroid: (" + cluster.centroid.x + ", " +
cluster.centroid.y + ")");

System.out.println("Points in Cluster " + (i +
1) + ": " + cluster.points.size());

System.out.println("Original Points in Cluster
" + (i + 1) + " :");

for (Point originalPoint : originalData) {

if

(cluster.points.contains(originalPoint)) {

System.out.println("(" +
originalPoint.x + ", " + originalPoint.y + ")");

}

}

}

scanner.close();

}

}

```

OUTPUT:

Enter the number of data points:

6

Enter data point 1 (x y):

1 2

Enter data point 2 (x y):

2 3

Enter data point 3 (x y):

3 4

Enter data point 4 (x y):

10 12

Enter data point 5 (x y):

12 14

Enter data point 6 (x y):

11 13

Enter the number of clusters (k):

2

Enter the maximum number of iterations:

100

Cluster 1 Centroid: (11.5, 13.5)

Points in Cluster 1: 3

Original Points in Cluster 1:

(11.5, 13.5)

(12.0, 14.0)

(11.0, 13.0)

Cluster 2 Centroid: (1.5, 2.5)

Points in Cluster 2: 3

Original Points in Cluster 2:

(1.0, 2.0)

(2.0, 3.0)

(1.5, 2.5)

// K- means algorithm

Conclusion: Thus, we have implemented K-means algorithm (Clustering algorithm) using python programming language. K-Means is a widely used clustering algorithm due to its simplicity and efficiency. While it has its limitations, it serves as a foundation for more advanced clustering methods and provides valuable insights into data grouping and pattern discovery. Understanding its

theoretical underpinnings helps in making informed decisions about its application and potential modifications.

Experiment No. 8

Aim: Implementation of any one Hierarchical Clustering method.

Theory:

Hierarchical Clustering : -

There are two types of hierarchical clustering: Agglomerative and Divisive. In the former, data points are clustered using a bottom-up approach starting with individual data points, while in the latter top-down approach is followed where all the data points are treated as one big cluster and the clustering process involves dividing the one big cluster into several small clusters.

Steps to Perform Hierarchical Clustering : -

Following are the steps involved in agglomerative clustering:

- At the start, treat each data point as one cluster. Therefore, the number of clusters at the start will be K , while K is an integer representing the number of data points.
- Form a cluster by joining the two closest data points resulting in $K-1$ clusters.
- Form more clusters by joining the two closest clusters resulting in $K-2$ clusters.
- Repeat the above three steps until one big cluster is formed.
- Once the single cluster is formed, dendrograms are used to divide into multiple clusters depending upon the problem. We will study the concept of dendrogram in detail in an upcoming section.

There are different ways to find distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Following are some of the options to measure distance between two clusters:

- Measure the distance between the closest points of two clusters.
- Measure the distance between the farthest points of two clusters.
- Measure the distance between the centroids of two clusters.
- Measure the distance between all possible combination of points between the two clusters and take the mean.

Role of Dendrograms in Agglomerative Hierarchical Clustering

As we discussed in the last step, the role of dendrogram starts once the big cluster is formed. Dendrogram will be used to split the clusters into multiple cluster of related data points depending upon our problem.

Code:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
public class HierarchicalClusteringSimple {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of data points: ");
        int numPoints = scanner.nextInt();
        List<double[]> dataPoints = generateData(numPoints); // Generate userdefined
        data
        List<Cluster> clusters = initializeClusters(dataPoints);
        int clusterNumber = 1;
        while (clusters.size() > 1) {
            int[] closestPair = findClosestClusters(clusters);
            Cluster mergedCluster = mergeClusters(clusters.remove(closestPair[1]),
            clusters.remove(closestPair[0]));
            clusters.add(mergedCluster);
            System.out.println("Cluster " + clusterNumber + ":");
```

```

printCluster(mergedCluster, 1);
System.out.println();
// Calculate and print the distance matrix
double[][] distanceMatrix = calculateDistanceMatrix(clusters);
System.out.println("Distance Matrix:");
printDistanceMatrix(distanceMatrix);
System.out.println();
clusterNumber++;
}
// Handle the case when there's only one cluster left
if (clusters.size() == 1) {
System.out.println("Final Cluster:");
printCluster(clusters.get(0), 1);
}
}

private static List<double[]> generateData(int numPoints) {
List<double[]> data = new ArrayList<>();
Scanner scanner = new Scanner(System.in);
for (int i = 0; i < numPoints; i++) {
System.out.print("Enter the number of dimensions (2 or 3): ");
int numDimensions = scanner.nextInt();
double[] point = new double[numDimensions];
System.out.println("Enter " + numDimensions + " values for data point "
+ (i + 1) + ":");
for (int j = 0; j < numDimensions; j++) {
point[j] = scanner.nextDouble();
}
data.add(point);
}
return data;
}

private static List<Cluster> initializeClusters(List<double[]> data) {
List<Cluster> clusters = new ArrayList<>();
for (int i = 0; i < data.size(); i++) {
clusters.add(new Cluster(data.get(i), i));
}
return clusters;
}

private static double calculateDistance(Cluster cluster1, Cluster cluster2) {
double[] centroid1 = cluster1.getCentroid();
double[] centroid2 = cluster2.getCentroid();
double sumSquaredDifferences = 0;
for (int i = 0; i < centroid1.length; i++) {

```

```

double difference = centroid1[i] - centroid2[i];
sumSquaredDifferences += difference * difference;
}
return Math.sqrt(sumSquaredDifferences);
}
private static int[] findClosestClusters(List<Cluster> clusters) {
int[] closestPair = { 0, 1 };
double minDistance = calculateDistance(clusters.get(0), clusters.get(1));
for (int i = 0; i < clusters.size(); i++) {
for (int j = i + 1; j < clusters.size(); j++) {
double distance = calculateDistance(clusters.get(i),
clusters.get(j));
if (distance < minDistance) {
minDistance = distance;
closestPair[0] = i;
closestPair[1] = j;
}
}
}
return closestPair;
}
private static Cluster mergeClusters(Cluster cluster1, Cluster cluster2) {
double[] mergedCentroid = new double[cluster1.getCentroid().length];
for (int i = 0; i < mergedCentroid.length; i++) {
mergedCentroid[i] = (cluster1.getCentroid()[i] +
cluster2.getCentroid()[i]) / 2;
}
List<Integer> mergedDataPoints = new
ArrayList<>(cluster1.getDataPoints());
mergedDataPoints.addAll(cluster2.getDataPoints());
return new Cluster(mergedCentroid, mergedDataPoints);
}
private static void printCluster(Cluster cluster, int level) {
StringBuilder indentation = new StringBuilder();
for (int i = 0; i < level; i++) {
indentation.append(" ");
}
if (cluster.getDataPoints().size() > 1) {
System.out.println(indentation.toString() + "Data Points: " +
cluster.getDataPoints());
for (Cluster child : cluster.getChildren()) {
printCluster(child, level + 1);
}
}
}

```

```

    } else {
        System.out.println(indentation.toString() + "Data Point: " +
            cluster.getDataPoints().get(0));
    }
}

private static double[][] calculateDistanceMatrix(List<Cluster> clusters) {
    int numClusters = clusters.size();
    double[][] distanceMatrix = new double[numClusters][numClusters];
    for (int i = 0; i < numClusters; i++) {
        for (int j = 0; j < numClusters; j++) {
            if (i != j) {
                distanceMatrix[i][j] = calculateDistance(clusters.get(i),
                    clusters.get(j));
            } else {
                distanceMatrix[i][j] = 0.0; // Diagonal elements are 0
            }
        }
    }
    return distanceMatrix;
}

private static void printDistanceMatrix(double[][] distanceMatrix) {
    for (int i = 0; i < distanceMatrix.length; i++) {
        for (int j = 0; j < distanceMatrix[0].length; j++) {
            System.out.printf("%.2f ", distanceMatrix[i][j]);
        }
        System.out.println();
    }
}

public static class Cluster {
    private double[] centroid;
    private List<Integer> dataPoints;
    private List<Cluster> children;
    public Cluster(double[] centroid, int dataPoint) {
        this.centroid = centroid;
        this.dataPoints = new ArrayList<>();
        this.children = new ArrayList<>();
        dataPoints.add(dataPoint);
    }
    public Cluster(double[] centroid, List<Integer> dataPoints) {
        this.centroid = centroid;
        this.dataPoints = dataPoints;
        this.children = new ArrayList<>();
    }
}

```



```
public double[] getCentroid() {  
    return centroid;  
}  
public List<Integer> getDataPoints() {  
    return dataPoints;  
}  
public List<Cluster> getChildren() {  
    return children;  
}  
}  
}
```

Enter the number of data points: 10

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 1:

3604

502.54

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 2:

8435

159.42

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 3:

4848

364.69

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 4:

7225

364.69

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 5:

1975

117.11

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 6:

2542

364.69

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 7:

4398

502.54

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 8:

49

502.54

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 9:

4031

263.33

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 10:

7911

263.33

Cluster 1:

Data Points: [8, 6]

Distance Matrix:

0.00	4843.17	1251.61	3623.62	1673.98	1070.91	3555.00	4313.64	622.11
4843.17	0.00	3592.87	1227.29	6460.14	5896.57	8393.02	534.20	4226.41
1251.61	3592.87	0.00	2377.00	2883.65	2306.00	4800.98	3064.68	633.76
3623.62	1227.29	2377.00	0.00	5255.83	4683.00	7177.32	693.45	3010.56
1673.98	6460.14	2883.65	5255.83	0.00	618.70	1964.19	5937.80	2255.22
1070.91	5896.57	2306.00	4683.00	618.70	0.00	2496.81	5369.96	1672.60
3555.00	8393.02	4800.98	7177.32	1964.19	2496.81	0.00	7865.64	4167.22
4313.64	534.20	3064.68	693.45	5937.80	5369.96	7865.64	0.00	3698.43
622.11	4226.41	633.76	3010.56	2255.22	1672.60	4167.22	3698.43	0.00

Cluster 2:

Data Points: [9, 1]

Distance Matrix:

0.00	1251.61	3623.62	1673.98	1070.91	3555.00	622.11	4578.27
1251.61	0.00	2377.00	2883.65	2306.00	4800.98	633.76	3328.53
3623.62	2377.00	0.00	5255.83	4683.00	7177.32	3010.56	960.32
1673.98	2883.65	5255.83	0.00	618.70	1964.19	2255.22	6198.72
1070.91	2306.00	4683.00	618.70	0.00	2496.81	1672.60	5633.09
3555.00	4800.98	7177.32	1964.19	2496.81	0.00	4167.22	8129.22
622.11	633.76	3010.56	2255.22	1672.60	4167.22	0.00	3962.22
4578.27	3328.53	960.32	6198.72	5633.09	8129.22	3962.22	0.00

Cluster 3:

Data Points: [5, 4]

Distance Matrix:

0.00	1251.61	3623.62	3555.00	622.11	4578.27	1370.70
1251.61	0.00	2377.00	4800.98	633.76	3328.53	2592.46
3623.62	2377.00	0.00	7177.32	3010.56	960.32	4968.04
3555.00	4800.98	7177.32	0.00	4167.22	8129.22	2224.94
622.11	633.76	3010.56	4167.22	0.00	3962.22	1961.15

4578.27 3328.53 960.32 8129.22 3962.22 0.00 5914.57
1370.70 2592.46 4968.04 2224.94 1961.15 5914.57 0.00

Cluster 4:

Data Points: [8, 6, 0]

Distance Matrix:

0.00 2377.00 4800.98 3328.53 2592.46 941.99
2377.00 0.00 7177.32 960.32 4968.04 3316.67
4800.98 7177.32 0.00 8129.22 2224.94 3860.71
3328.53 960.32 8129.22 0.00 5914.57 4270.02
2592.46 4968.04 2224.94 5914.57 0.00 1663.04
941.99 3316.67 3860.71 4270.02 1663.04 0.00

Cluster 5:

Data Points: [8, 6, 0, 2]

Distance Matrix:

0.00 7177.32 960.32 4968.04 2846.64
7177.32 0.00 8129.22 2224.94 4330.75
960.32 8129.22 0.00 5914.57 3799.25
4968.04 2224.94 5914.57 0.00 2126.37
2846.64 4330.75 3799.25 2126.37 0.00

Cluster 6:

Data Points: [9, 1, 3]

Distance Matrix:

0.00 2224.94 4330.75 7653.01
2224.94 0.00 2126.37 5440.70
4330.75 2126.37 0.00 3322.39
7653.01 5440.70 3322.39 0.00

Cluster 7:

Data Points: [8, 6, 0, 2, 5, 4]

Distance Matrix:

0.00 7653.01 3274.53
7653.01 0.00 4380.57
3274.53 4380.57 0.00

Cluster 8:

Data Points: [8, 6, 0, 2, 5, 4, 7]

Distance Matrix:

0.00 6016.50
6016.50 0.00

Cluster 9:

Data Points: [8, 6, 0, 2, 5, 4, 7, 9, 1, 3]

Distance Matrix:

0.00

Final Cluster:

Data Points: [8, 6, 0, 2, 5, 4, 7, 9, 1, 3]

Conclusion: Thus, in this experiment, we have implemented Hierarchical Clustering method using agglomerative algorithm.

Experiment No. 9

Aim: Implementation of Association Rule Mining algorithm (Apriori).

Theory:

Association rule mining is a technique to identify underlying relations between different items. There are many methods to perform association rule mining. The Apriori algorithm is the most simple and straightforward approach. However, since it is the fundamental method, there are many different improvements that can be applied to it.

Components of Apriori algorithm

The given three components comprise the Apriori algorithm.

1. Support
2. Confidence
3. Lift

Let us take an example to understand this concept.

We have already discussed above; you need a huge database containing a large no of transactions. Suppose you have 4000 customers transactions in a Big Bazar. You must calculate the Support, Confidence, and Lift for two products, and you may say Biscuits and Chocolate. This is because customers frequently buy these two items together.

Out of 4000 transactions, 400 contain Biscuits, whereas 600 contain Chocolate, and these 600 transactions include a 200 that includes Biscuits and chocolates. Using this data, we will find out the support, confidence, and lift.

Support

Support refers to the default popularity of any product. You find the support as a quotient of the division of the number of transactions comprising that product by the total number of transactions. Hence, we get

Support (Biscuits) = (Transactions relating biscuits) / (Total transactions)

= 400/4000 = 10 percent.

Confidence

Confidence refers to the possibility that the customers bought both biscuits and chocolates together. So, you need to divide the number of transactions that comprise both biscuits and chocolates by the total number of transactions to get the confidence.

Hence,

Confidence = (Transactions relating both biscuits and Chocolate) / (Total transactions involving Biscuits)

= 200/400

= 50 percent.

It means that 50 percent of customers who bought biscuits bought chocolates also.

Advantages of Apriori Algorithm

It is used to calculate large itemsets.

Simple to understand and apply.

Disadvantages of Apriori Algorithm

Apriori algorithm is an expensive method to find support since the calculation has to pass through the whole database.

Sometimes, you need a huge number of candidate rules, so it becomes computationally more expensive.

Code:

```
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;
```

```
import java.util.HashMap;
import java.util.Map;
public class Apriori {
    public static void main(String[] args) {
        List< String[] > data = new ArrayList< > ();
        data.add(new String[] {
            "T100",
            "I1",
            "I2",
            "I5"
        });
        data.add(new String[] {
            "T200",
            "I2",
            "I4"
        });
        data.add(new String[] {
            "T300",
            "I2",
            "I3"
        });
        data.add(new String[] {
            "T400",
            "I1",
            "I2",
            "I4"
        });
        data.add(new String[] {
            "T500",
            "I1",
            "I3"
        });
        data.add(new String[] {
            "T600",
            "I2",
            "I3"
        });
        data.add(new String[] {
            "T700",
            "I1",
            "I3"
        });
        data.add(new String[] {
```

```

        "T800",
        "I1",
        "I2",
        "I3",
        "I5"
    });
    data.add(new String[] {
        "T900",
        "I1",
        "I2",
        "I3"
    });
    List < String > init = new ArrayList < > ();
    for (String[] itemSet: data) {
        for (int i = 1; i < itemSet.length; i++) {
            if (!init.contains(itemSet[i])) {
                init.add(itemSet[i]);
            }
        }
    }
    init.sort(null);
    double sp = 0.4;
    int s = (int)(sp * init.size());
    Map < String, Integer > c = new HashMap < > ();
    for (String i: init) {
        for (String[] d: data) {
            if (containsItem(d, i)) {
                c.put(i, c.getDefault(i, 0) + 1);
            }
        }
    }
    System.out.println("C1:");
    for (String key: c.keySet()) {
        System.out.println "[" + key + ": " + c.get(key));
    }
    System.out.println();
    Map < Set < String > , Integer > l = new HashMap < > ();
    for (String i: c.keySet()) {
        if (c.get(i) >= s) {
            Set < String > itemSet = new HashSet < > ();
            itemSet.add(i);
            l.put(itemSet, c.get(i));
        }
    }

```



```

}
System.out.println("L1:");
for (Set < String > key: l.keySet()) {
    System.out.println(key + ": " + l.get(key));
}
System.out.println();
Map < Set < String > , Integer > pl = l;
int pos = 1;
for (int count = 2; count < 1000; count++) {
    Set < Set < String >> nc = new HashSet < > ();
    List < Set < String >> temp = new ArrayList < > (l.keySet());
    for (int i = 0; i < temp.size(); i++) {
        for (int j = i + 1; j < temp.size(); j++) {
            Set < String > t = new HashSet < > (temp.get(i));
            t.addAll(temp.get(j));
            if (t.size() == count) {
                nc.add(t);
            }
        }
    }
}
Map < Set < String > , Integer > cMap = new HashMap < > ();
for (Set < String > i: nc) {
    cMap.put(i, 0);
    for (String[] q: data) {
        Set < String > tempSet = new HashSet < > (List.of(q).subList(1,
            q.length));
        if (i.stream().allMatch(tempSet::contains)) {
            cMap.put(i, cMap.get(i) + 1);
        }
    }
}
System.out.println("C" + count + ":");
for (Set < String > key: cMap.keySet()) {
    System.out.println(key + ": " + cMap.get(key));
}
System.out.println();
l = new HashMap < > ();
for (Set < String > i: cMap.keySet()) {
    if (cMap.get(i) >= s) {
        l.put(i, cMap.get(i));
    }
}
System.out.println("L" + count + ":");

```

```

for (Set < String > key: l.keySet()) {
    System.out.println(key + ": " + l.get(key));
}
System.out.println();
if (l.isEmpty()) {
    break;
}
pl = l;
pos = count;
}
System.out.println("Result:");
System.out.println("L" + pos + ":");
for (Set < String > key: pl.keySet()) {
    System.out.println(key + ": " + pl.get(key));
}
for (Set < String > lSet: pl.keySet()) {
    List < String > cList = new ArrayList < > (lSet);
    List < Set < String >> subsets = generateSubsets(cList);
    double mmax = 0;
    for (Set < String > a: subsets) {
        Set < String > b = new HashSet < > (cList);
        b.removeAll(a);
        int sab = 0;
        int sa = 0;
        int sb = 0;
        for (String[] q: data) {
            Set < String > tempSet = new HashSet < > (List.of(q).subList(1,
                q.length));
            if (a.stream().allMatch(tempSet::contains)) {
                sa++;
            }
            if (b.stream().allMatch(tempSet::contains)) {
                sb++;
            }
            if (cList.stream().allMatch(tempSet::contains)) {
                sab++;
            }
        }
        double temp = (double) sab / sa * 100;
        if (temp > mmax) {
            mmax = temp;
        }
        System.out.println(a + " -> " + b + " = " + temp + "%");
    }
}

```

```

        System.out.println(b + " -> " + a + " = " + (double) sab / sb * 100 +
            "%");
    }
    int curr = 1;
    System.out.print("Choosing: ");
    for (Set < String > a: subsets) {
        Set < String > b = new HashSet < > (cList);
        b.removeAll(a);
        int sab = 0;
        int sa = 0;
        int sb = 0;
        for (String[] q: data) {
            Set < String > tempSet = new HashSet < > (List.of(q).subList(1,
                q.length));
            if (a.stream().allMatch(tempSet::contains)) {
                sa++;
            }
            if (b.stream().allMatch(tempSet::contains)) {
                sb++;
            }
            if (cList.stream().allMatch(tempSet::contains)) {
                sab++;
            }
        }
        double temp = (double) sab / sa * 100;
        if (temp == mmax) {
            System.out.print(curr + " ");
        }
        curr++;
        temp = (double) sab / sb * 100;
        if (temp == mmax) {
            System.out.print(curr + " ");
        }
        curr++;
    }
    System.out.println();
}
}

private static boolean containsItem(String[] itemSet, String item) {
    for (int i = 1; i < itemSet.length; i++) {
        if (itemSet[i].equals(item)) {
            return true;
        }
    }
}

```

```

    }
    return false;
}
private static List < Set < String >> generateSubsets(List < String > originalSet) {
    List < Set < String >> subsets = new ArrayList < > ();
    int n = originalSet.size();
    for (int i = 0; i < (1 << n); i++) {
        Set < String > subset = new HashSet < > ();
        for (int j = 0; j < n; j++) {
            if ((i & (1 << j)) > 0) {
                subset.add(originalSet.get(j));
            }
        }
        subsets.add(subset);
    }
    subsets.remove(0); // Remove the empty set
    return subsets;
}
}

```

Output:

C1:

[11]: 6

[12]: 7

[13]: 6

[14]: 2

[15]: 2

L1:

[11]: 6

[12]: 7

[13]: 6

[14]: 2

[15]: 2

C2:

[11, 12]: 4

[11, 13]: 4

[11, 14]: 1

[12, 13]: 4

[11, 15]: 2

[12, 14]: 2

[12, 15]: 2

[13, 14]: 0

[13, 15]: 1

[I4, I5]: 0
L2:
[I1, I2]: 4
[I1, I3]: 4
[I2, I3]: 4
[I1, I5]: 2
[I2, I4]: 2
[I2, I5]: 2
C3:
[I2, I4, I5]: 0
[I1, I2, I3]: 2
[I1, I2, I4]: 1
[I1, I2, I5]: 2
[I1, I3, I5]: 1
[I2, I3, I4]: 0
[I2, I3, I5]: 1
L3:
[I1, I2, I3]: 2
[I1, I2, I5]: 2
C4:
[I1, I2, I3, I5]: 1
L4:

Result:

L3:
[I1, I2, I3]: 2
[I1, I2, I5]: 2
[I1] -> [I2, I3] = 33.33333333333333%
[I2, I3] -> [I1] = 50.0%
[I2] -> [I1, I3] = 28.57142857142857%
[I1, I3] -> [I2] = 50.0%
[I1, I2] -> [I3] = 50.0%
[I3] -> [I1, I2] = 33.33333333333333%
[I3] -> [I1, I2] = 33.33333333333333%
[I1, I2] -> [I3] = 50.0%
[I1, I3] -> [I2] = 50.0%
[I2] -> [I1, I3] = 28.57142857142857%
[I2, I3] -> [I1] = 50.0%
[I1] -> [I2, I3] = 33.33333333333333%
[I1, I2, I3] -> [] = 100.0%
[] -> [I1, I2, I3] = 22.22222222222222%
Choosing: 13

[I1] -> [I2, I5] = 33.33333333333333%
[I2, I5] -> [I1] = 100.0%
[I2] -> [I1, I5] = 28.57142857142857%
[I1, I5] -> [I2] = 100.0%
[I1, I2] -> [I5] = 50.0%
[I5] -> [I1, I2] = 100.0%
[I5] -> [I1, I2] = 100.0%
[I1, I2] -> [I5] = 50.0%
[I1, I5] -> [I2] = 100.0%
[I2] -> [I1, I5] = 28.57142857142857%
[I2, I5] -> [I1] = 100.0%
[I1] -> [I2, I5] = 33.33333333333333%
[I1, I2, I5] -> [] = 100.0%
[] -> [I1, I2, I5] = 22.22222222222222%
Choosing: 2 4 6 7 9 11 13

Conclusion: Thus, in this experiment, we have implemented Association Rule Mining algorithm using Apriori algorithm.

Experiment No. 10

Aim: Implementation of Page rank/HITS algorithm.

Theory:

Hyperlink Induced Topic Search (HITS) is an algorithm used in link analysis. It could discover and rank the webpages relevant for a particular search. The idea of this algorithm originated from the fact that an ideal website should link to other relevant sites and also being linked by other important sites.

HITS uses hubs and authorities to define a recursive relationship between web pages.

- Authority: A node is high-quality if many high-quality nodes link to it
- Hub: A node is high-quality if it links to many high-quality nodes

Algorithm Steps

- Initialize the hub and authority of each node with a value of 1
- For each iteration, update the hub and authority of every node in the graph
- The new authority is the **sum of the hub** of its parents
- The new hub is the **sum of the authority** of its children
- Normalize the new authority and hub

The first step that the algorithm takes is to retrieve the data of the

search query. This is the information people type on search engines to obtain a specific result. Then, it performs a computation only regarding these results, without taking into consideration other websites.

After that, authoritative and hub values are defined, and a process of iteration begins. In the iteration process, two updates are done: the authority update and the hub update. For HITS, an authoritative website is a site that has valuable content. These types of websites tend to rank higher on the search engine results page because they are considered 'expert' pages. Much like PageRank (another algorithm that identifies and ranks sites), HITS takes the linkage of documents on the web into account.

However, HITS differentiates itself from PageRank on a few aspects:

- It is executed at query time, not at indexing time. The hub and authority scores assigned to a page are query-specific. This means that the ranking will always consider the keyword or content that people are searching for.
- Whereas algorithms like PageRank compute one score per document, HITS compute two.
- It is processed on a small subset of documents, instead of all documents, like PageRank does.
- Search engines do not commonly use it.

Code:

```
import numpy as np
```



```

def pagerank(M, num_iterations, d=0.85):
    N = M.shape[1]
    page_rank = np.ones(N) / N

    for _ in range(num_iterations):
        new_page_rank = (1 - d) / N + d * np.dot(M, page_rank)
        if np.allclose(new_page_rank, page_rank, atol=1e-6):
            break
        page_rank = new_page_rank

    return page_rank

web_graph = np.array([
    [0, 1, 1, 0],
    [1, 0, 1, 1],
    [1, 0, 0, 1],
    [0, 1, 1, 0]
], dtype=float)

web_graph /= web_graph.sum(axis=0, keepdims=True)

page_rank_scores = pagerank(web_graph, num_iterations=100)

rank_order = np.argsort(page_rank_scores)[::-1]

for i, (score, rank) in enumerate(zip(page_rank_scores, rank_order), start=1):
    print(f"Page {i}: PageRank Score={score:.4f}, Rank={rank + 1}")

```

Output:

Page 1: PageRank Score=0.2320, Rank=2

Page 2: PageRank Score=0.3012, Rank=3

Page 3: PageRank Score=0.2347, Rank=4

Page 4: PageRank Score=0.2320, Rank=1

**** Process exited - Return Code: 0 ****

Press Enter to exit terminal

Conclusion: Thus, in this experiment, we have implemented Page Rank/HITS algorithm.