


# Thadomal Shahani Engineering College

Bandra (W.), Mumbai- 400 050.

## ❧ CERTIFICATE ❧

Certify that ~~Mr.~~/Miss Niyati Savant  
of Computer Department, Semester V with  
Roll No. 2103156 has completed a course of the necessary  
experiments in the subject DWM under my  
supervision in the **Thadomal Shahani Engineering College**  
Laboratory in the year 2023 - 2024

  
Teacher In-Charge

Head of the Department

Date 21 October 2023

Principal

## CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1	Select a dataset and perform exploratory data analysis using Python	1-12	19/7/23	
2	One Case study on building Data warehouse. Write detailed Problem Statement and design Star and Snowflake schema	13-14	26/7/23	
3	Implementation of all dimension table and fact table based on Experiment 2 Case Study	17-44	2/8/23	
4	Implementation of OLAP operation Slice, Dice, Roll up, Drill down and Pivot based on Experiment 2.	45-49	23/8/23	① 21/10/23
5	Implementation of Bayesian Algorithm	50-57	6/9/23	
6	Perform data Preprocessing task and demonstrate performing Classification, Clustering, Association using WEKA tool	58-67	13/9/23	
7	Implementation of Clustering algorithm (K-means)	68-73	27/9/23	



## Experiment 1 :

**Aim :** Select a Dataset and perform Exploratory data analysis using Python

- Data cleaning- Missing Values,remove outliers
- Data transformation- Min-max normalisation, Z-score normalisation,Decimal
- Scaling
- Data Discretization- Binning
- Data analysis and Visualization

### **Steps:**

- 1) Load the libraries Download the data set from kaggle/ other sources
- 2) Read the file –select appropriate file read function according to data type of file
- 3) Describe the attributes name, count no of values, and find min, max, data type, range, quartile, percentile, box plot and outliers.
- 4) Perform cleaning,transformation , discretization and analysis
- 5) Give visualisation of statistical description of data – in form of histogram, scatter plot, pie chart,Give correlation matrix

### **Theory :**

So to be performing data analysis on a dataset related to housing in Mumbai. Here is a theory related to the concepts and processes involved in the code:

#### **1. Data Preprocessing:**

Data preprocessing is a crucial phase in the realm of data warehouse management (DWM), where raw data is transformed and refined into a structured format suitable for analysis and reporting. Effective data preprocessing sets the foundation for accurate insights and informed decision-making. In this context, here are eight essential points to consider when undertaking data preprocessing within a data warehouse management framework.

Points for Data Preprocessing in DWM:

##### **a.Data Collection and Integration:**

Data preprocessing begins with collecting data from various sources, including databases, APIs, and external files. Integrating this diverse data ensures a comprehensive view and minimizes data silos.

##### **b.Data Cleaning:**

Cleaning involves identifying and handling inconsistencies, missing values, and

errors in the data. Imputing missing values and rectifying anomalies ensures that downstream analyses are not compromised by flawed data.

**c.Data Transformation:**

Data often requires transformation to be usable. This could involve normalisation (scaling data to a standard range), encoding categorical variables, or aggregating data to a suitable granularity for analysis.

**d.Data Deduplication:**

Duplicate records can distort analysis results. Implementing deduplication techniques ensures that the same data point is not counted multiple times, improving accuracy.

**e.Data Reduction:**

In cases of large datasets, data reduction techniques like dimensionality reduction (PCA, t-SNE) can help retain essential information while reducing computational complexity.

**f.Outlier Detection and Handling:**

Outliers can skew analysis and modeling results. Identifying and dealing with outliers through techniques like statistical tests or clustering can lead to more reliable insights.

**g.Data Formatting and Validation:**

Ensuring data consistency and adherence to defined formats is crucial. Validation checks are applied to confirm that the data meets the required standards before it's loaded into the data warehouse.

**2.Data Transformation:**

Data transformation is a critical process in data preprocessing that involves converting raw data into a more suitable format for analysis, reporting, and modeling. It enhances the usability of the data by standardizing, scaling, and reorganizing it. Data transformation enables better insights and more accurate decision-making by preparing the data to be compatible with various algorithms and analytical techniques.

Five Points on Data Transformation:

**a.Normalisation:**

Normalisation scales numerical data to a common range, usually between 0 and 1. This eliminates the impact of differing scales on algorithms and ensures that each feature contributes equally to analysis.

**b.Encoding Categorical Variables:**

Categorical variables, such as gender or product category, need to be encoded into numerical values for analysis. Techniques like one-hot encoding or label encoding are used to represent categorical data in a format that algorithms can process.

**c.Aggregation:**

Aggregating data involves summarizing information by grouping it based on certain attributes. For instance, sales data might be aggregated by month to analyze monthly trends. Aggregation simplifies complex datasets and makes them more manageable.



**d.Feature Creation:**

Feature creation involves generating new attributes from existing data. These new features can capture patterns that are not immediately evident. For example, creating a "customer loyalty score" based on purchase frequency and total spending can provide valuable insights.

**e.Binning and Discretization:**

Binning involves dividing continuous data into discrete intervals or bins. This can simplify complex datasets and reveal trends that might not be apparent in the raw data. Binning can be particularly useful when dealing with numerical data that has a wide range.

**3.Data Discretization:**

Data discretization is a data transformation technique that involves converting continuous data into a discrete format by grouping values into intervals or bins. This process simplifies complex data, reduces noise, and can make it easier to uncover patterns and trends that might not be immediately apparent in the raw data.

Points on Data Discretization:

**a.Benefits of Discretization:**

Discretization can simplify analysis and modeling processes by converting continuous data into categories. This can make data more manageable, especially when dealing with large datasets. Additionally, it can reduce the impact of outliers and small fluctuations in the data.

**b.Methods of Discretization:**

There are various methods for discretizing data, including equal width binning, equal frequency binning, and clustering-based binning. Equal width binning divides the data range into intervals of equal width, while equal frequency binning ensures each interval contains approximately the same number of data points. Clustering-based binning groups data based on similarity measured by clustering algorithms.

**c.Impact on Analysis and Interpretation:**

Discretization can affect the results of data analysis and modelling. While it simplifies data, it may also lead to loss of information due to grouping similar values together. The choice of binning method and the number of bins should be guided by the underlying characteristics of the data and the objectives of the analysis.

**4.Data Visualization:**

Data visualisation is the practice of representing data in graphical or visual formats to aid understanding, analysis, and communication of insights. By translating raw data into intuitive visual representations, data visualisation enhances the ability to identify trends, patterns, and relationships within the data, making complex information more accessible and actionable.

Points on Data Visualization:

**a.Enhanced Understanding:**

Data visualisation transforms abstract data into visual cues, enabling viewers to quickly grasp information and identify trends that might not be apparent in raw data. It provides a clear overview of data distributions, correlations, and anomalies,

making it easier to derive insights.

### **b.Effective Communication:**

Visualisations are powerful tools for communicating findings to both technical and non-technical audiences. Complex data can be presented in a digestible manner, allowing stakeholders to make informed decisions based on the visual representation of information.

### **c.Visualisation Types:**

There are various types of visualisations, each suited for different data characteristics and objectives. Common types include bar charts, line graphs, scatter plots, histograms, and heatmaps. Choosing the right visualisation type depends on the data's nature and the specific insights you want to convey

## **Implementation:**

Loading and importing libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df= pd.read_csv('Mumbai.csv')

df.columns

Index(['Price', 'Area', 'Location', 'No. of Bedrooms', 'Resale',
      'MaintenanceStaff', 'Gymnasium', 'SwimmingPool', 'LandscapedGardens',
      'JoggingTrack', 'RainWaterHarvesting', 'IndoorGames', 'ShoppingMall',
      'Intercom', 'SportsFacility', 'ATM', 'ClubHouse', 'School',
      '24X7Security', 'PowerBackup', 'CarParking', 'StaffQuarter',
      'Cafeteria', 'MultipurposeRoom', 'Hospital', 'WashingMachine',
      'Gasconnection', 'AC', 'Wifi', 'Children'splayarea', 'LiftAvailable',
      'BED', 'VaastuCompliant', 'Microwave', 'GolfCourse', 'TV',
      'DiningTable', 'Sofa', 'Wardrobe', 'Refrigerator'],
      dtype='object')
```

## **Cleaning:**

```
1 df.duplicated().sum()
2 df.drop_duplicates(keep='first', inplace=True)
3 df.duplicated().sum()

1 df.replace(9, np.nan, inplace=True)
2 df.head()
```

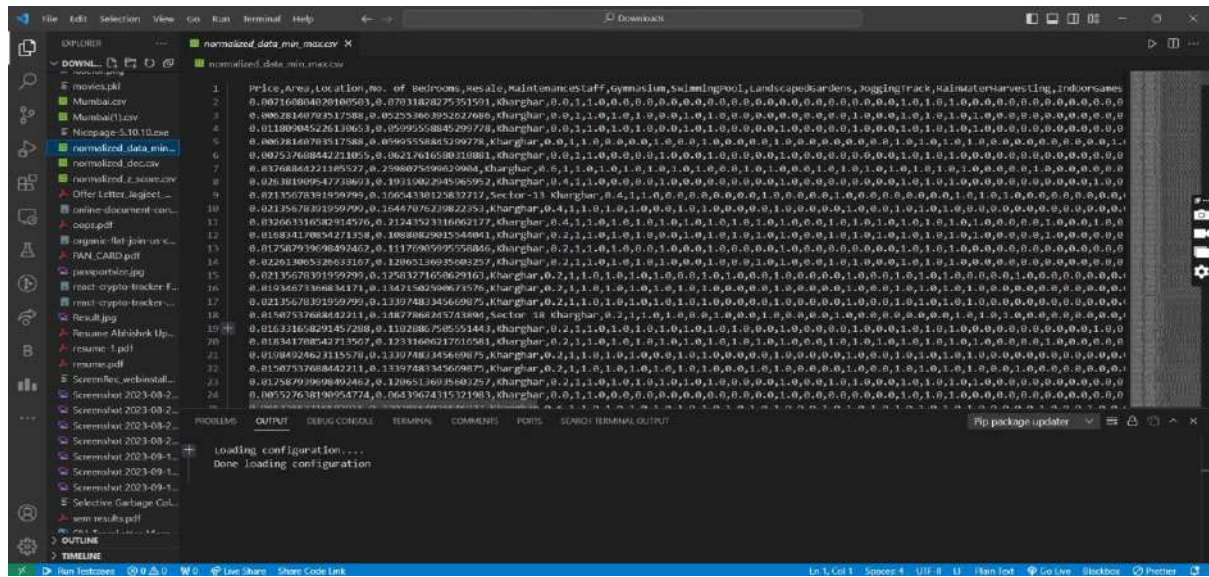
```
1 df.isnull().sum()
2 df.dropna(inplace=True)
3 df.isnull().sum()
```

## Transformation:

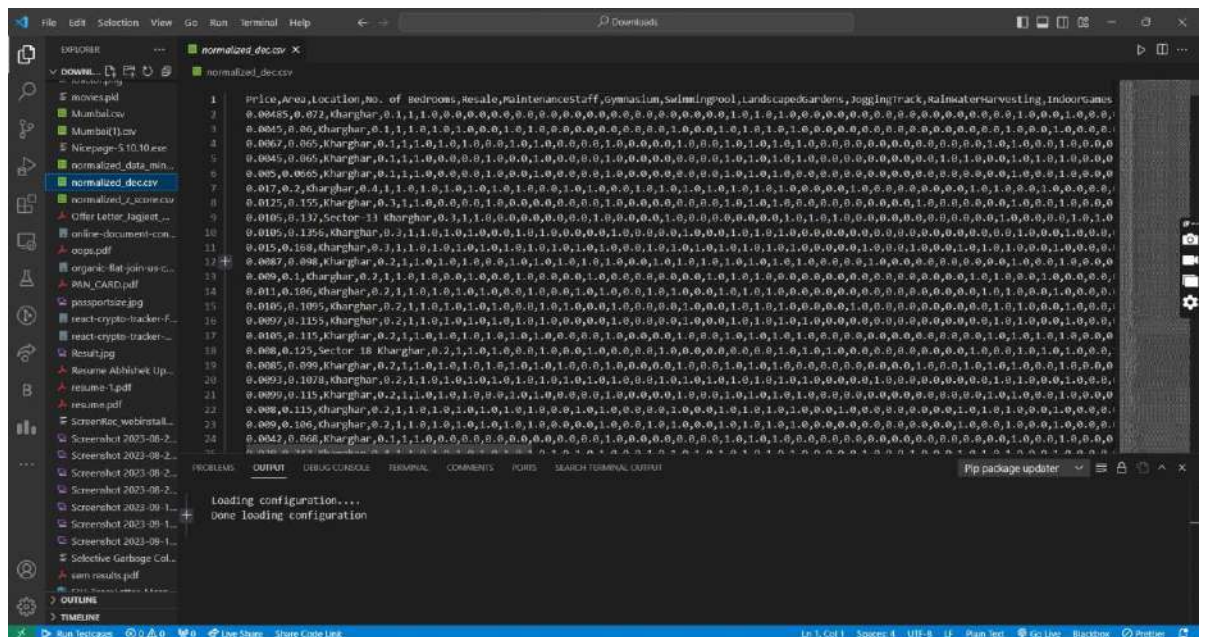
```
1 numericals_columns=['Price','Area','No. of Bedrooms']
2
3 # min-max normalization of data
4 def min_max_normalization(column):
5     return (column-column.min())/(column.max()-column.min())
6
7
8 data_min_max=df.copy()
9 data_min_max[numericals_columns]=data_min_max[numericals_columns].apply(min_max_normalization)
10
11 # z score for normalization
12 def z_score(column):
13     return (column-column.mean())/column.std()
14
15 data_z=df.copy()
16 data_z[numericals_columns]=data_z[numericals_columns].apply(z_score)
17
18 def dec_norm(column):
19     max_value=column.max()
20     num_dec=len(str(int(max_value)))
21     divisor=10**num_dec
22     return column/divisor
23
24 data_decimal=df.copy()
25 data_decimal[numericals_columns]=data_decimal[numericals_columns].apply(dec_norm)
26
27
28 #save data
29 data_min_max.to_csv('normalized_data_min_max.csv',index=False)
30 data_z.to_csv('normalized_z_score.csv',index=False)
31 data_decimal.to_csv('normalized_dec.csv',index=False)
32
```



**Output:**  
**Min\_max:**



## Decimal scaling :



**Z score:**



### Data Discretization:

```

1 num_of_bins = 3
2 column_to_desc = ['Price']
3 df=df.head(600)
4 def equal_width_binning(column, num_bin):
5     bin_width = (column.max() - column.min()) / num_bin
6     bins = [column.min() + i * bin_width for i in range(num_bin + 1)]
7     labels = [f'Bin {i + 1}' for i in range(num_bin)]
8     return pd.cut(column, bins=bins, labels=labels, include_lowest=True)
9
10 for c in column_to_desc:
11     df[f'{c}_bin'] = equal_width_binning(df[c], num_of_bins)
12
13 df.to_csv('discreted_data.csv', index=False)
14

```

**Output:**

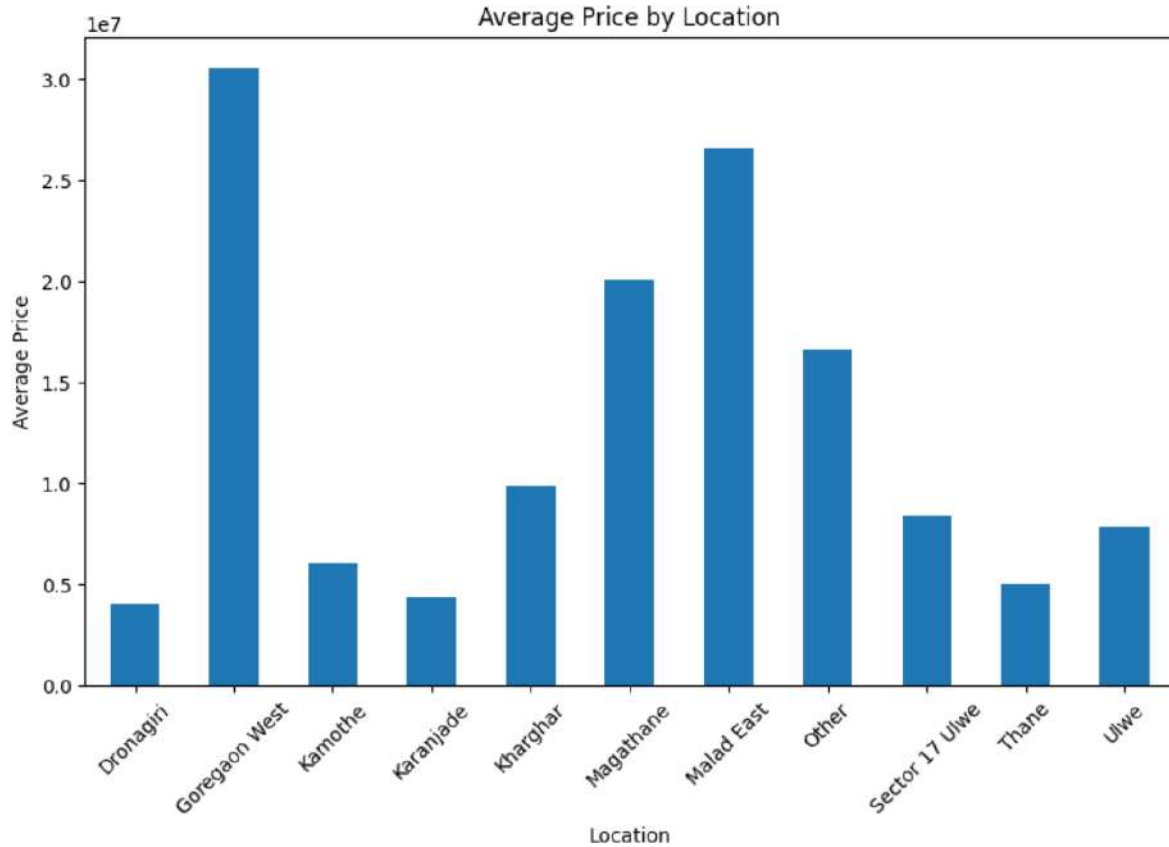
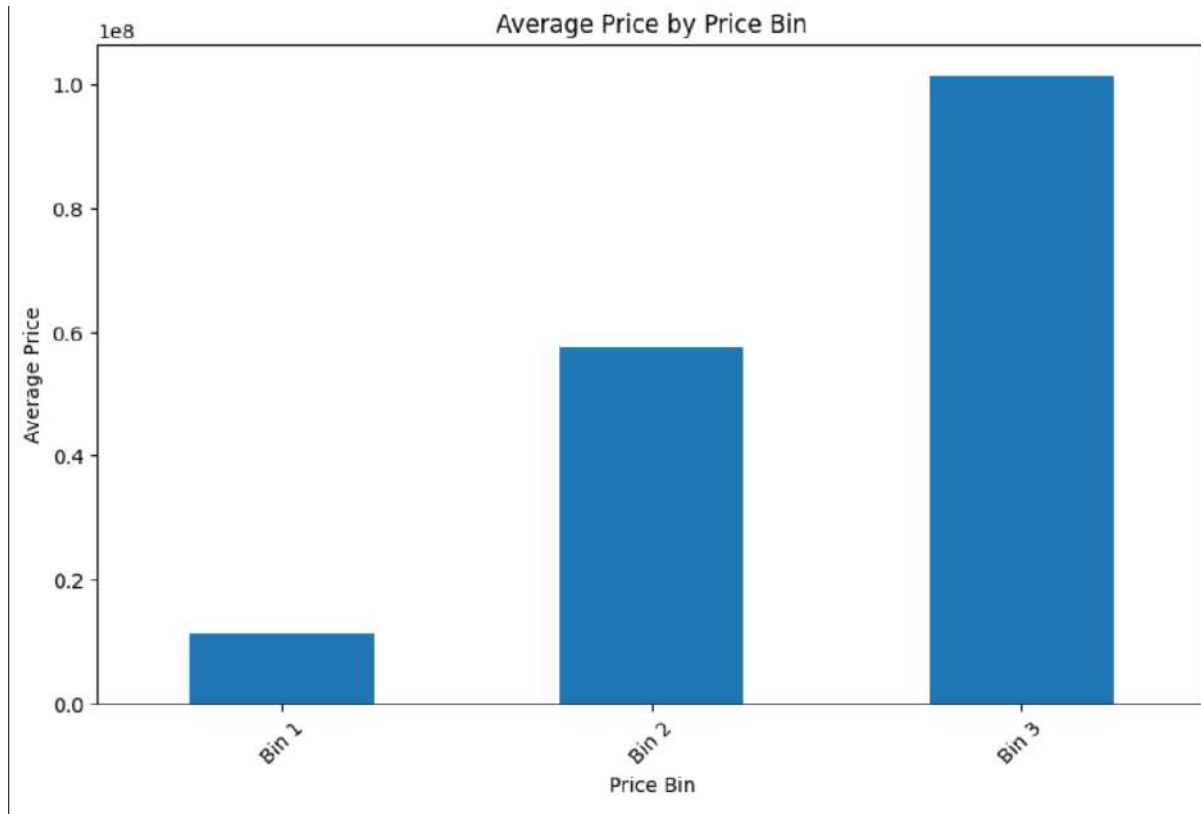




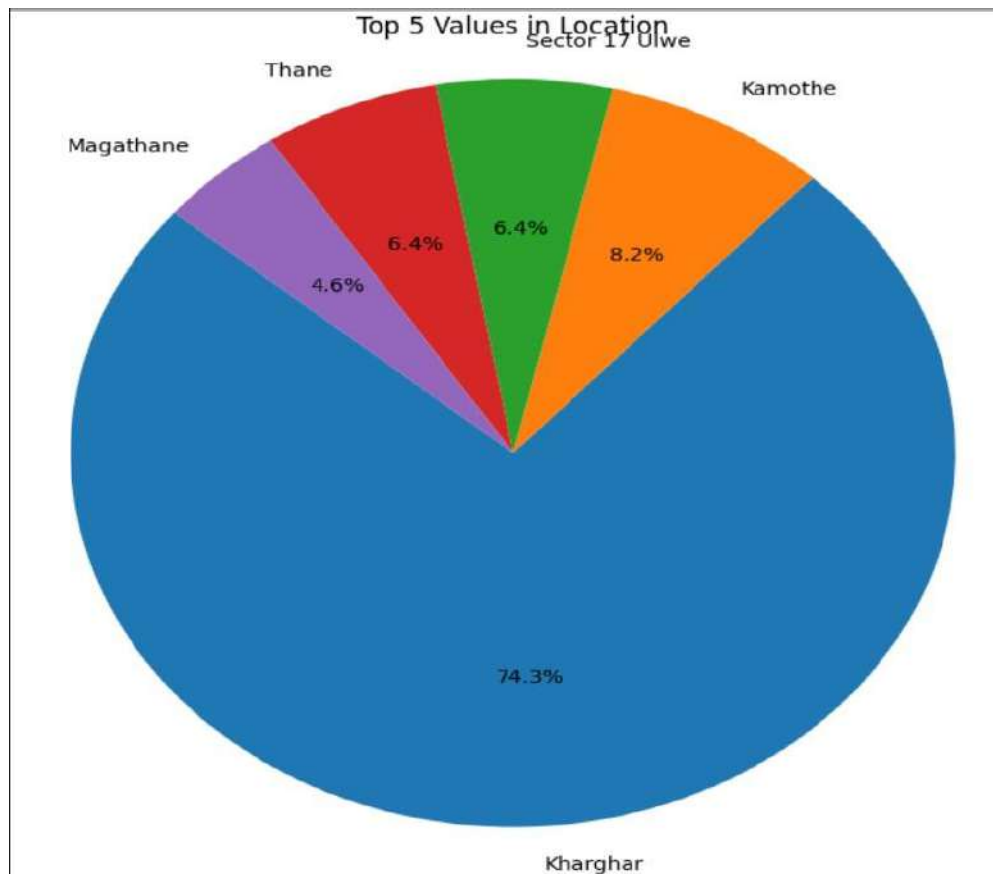
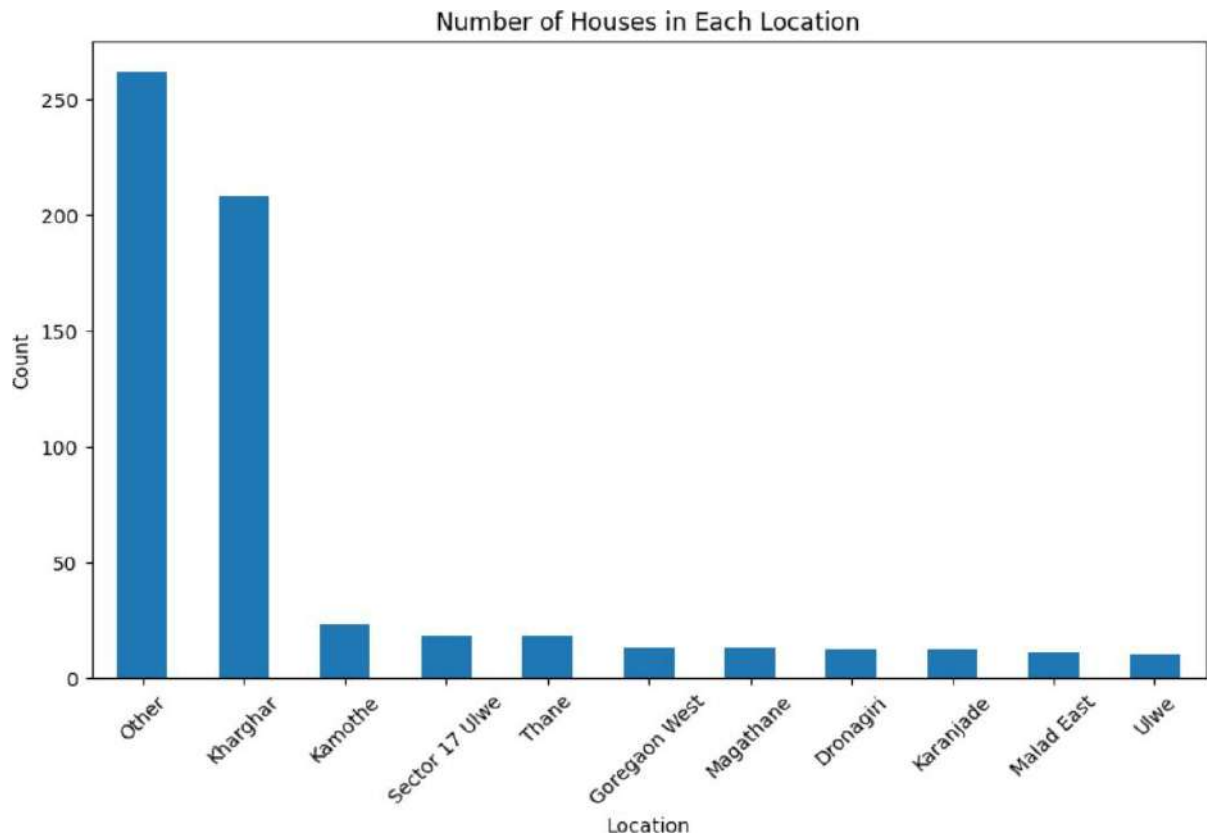
## Data Visualization and Cleaning:

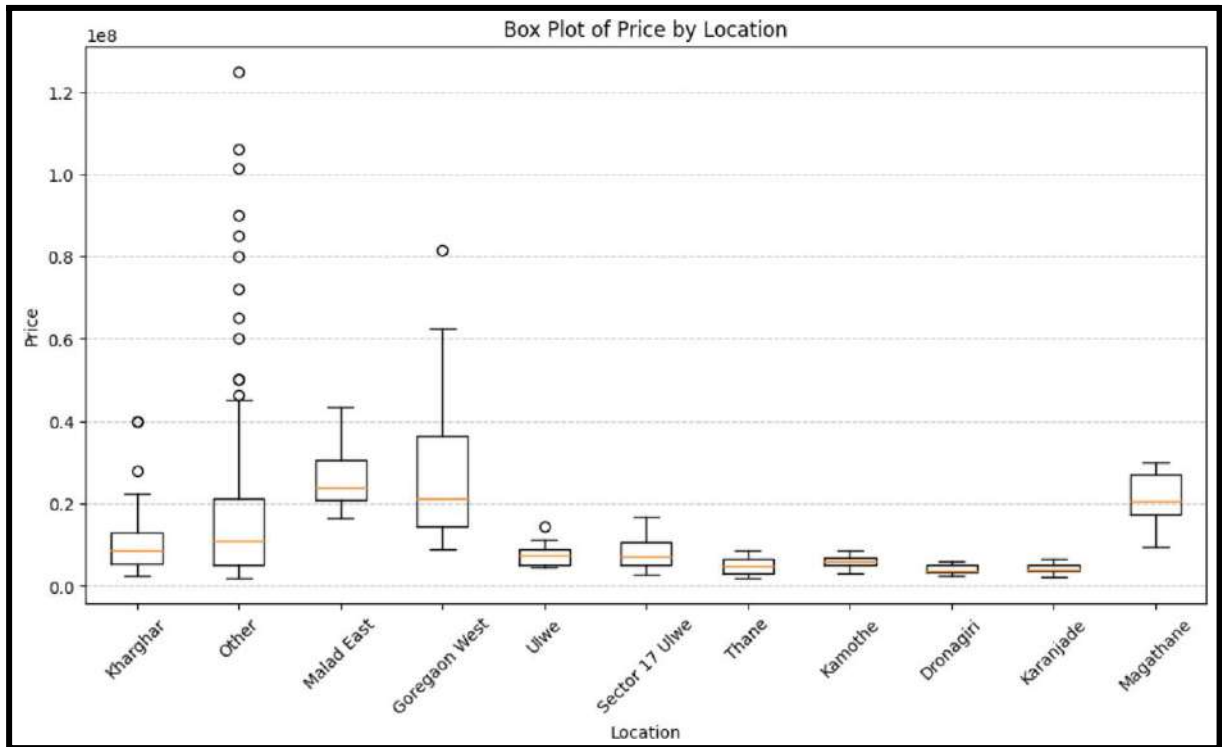
```
1 dcr = pd.read_csv('discreted_data.csv')
2 dcr=dcr.head(600)
3 location_counts = dcr['Location'].value_counts()
4 top_10_locations = location_counts.head(10).index.tolist()
5 dcr['Location'] = dcr['Location'].apply(lambda x: x if x in top_10_locations else 'Other')
6
7 # Calculate average price per bin
8 avg_price_per_bin = dcr.groupby('Price_bin')['Price'].mean()
9
10 # Count the number of houses in each location
11 category_counts = dcr['Location'].value_counts()
12
13 # Calculate average price by location
14 avg_price_by_area = dcr.groupby('Location')['Price'].mean()
15
16 # Plot the average price per bin
17 plt.figure(figsize=(10, 6))
18 avg_price_per_bin.plot(kind="bar")
19 plt.title('Average Price by Price Bin')
20 plt.xlabel('Price Bin')
21 plt.ylabel('Average Price')
22 plt.xticks(rotation=45)
23 plt.show()
24
25 # Plot the number of houses in each location
26 plt.figure(figsize=(10, 6))
27 category_counts.plot(kind="bar")
28 plt.title('Number of Houses in Each Location')
29 plt.xlabel('Location')
30 plt.ylabel('Count')
31 plt.xticks(rotation=45)
32 plt.show()
33
34 # Plot the average price by location
35 plt.figure(figsize=(10, 6))
36 avg_price_by_area.plot(kind="bar")
37 plt.title('Average Price by Location')
38 plt.xlabel('Location')
39 plt.ylabel('Average Price')
40 plt.xticks(rotation=45)
41 plt.show()
42
43
44 categorical_col = 'Location'
45 top_n = 5
46 top_values = df[categorical_col].value_counts().head(top_n)
47 plt.figure(figsize=(8, 8))
48 plt.pie(top_values, labels=top_values.index, autopct='%1.1f%%', startangle=140)
49 plt.axis('equal')
50 plt.title(f'Top {top_n} Values in {categorical_col}')
51 plt.show()
52
53 plt.figure(figsize=(12, 6))
54 plt.xticks(rotation=45)
55 plt.title('Box Plot of Price by Location')
56 plt.xlabel('Location')
57 plt.ylabel('Price')
58 plt.grid(axis='y', linestyle='--', alpha=0.7)
59 plt.boxplot([dcr[dcr['Location'] == location]['Price'] for location in dcr['Location'].unique()],
60             labels=dcr['Location'].unique())
61 plt.show()
62
```

## Output:









## Experiment 2

### **Aim- Write detailed problem statement and draw Star and Snowflake Schema Diagram**

#### **Theory –**

*Problem Statement:* Designing a Star Schema for House Price Prediction

We are designing a star schema to predict house prices accurately. This star schema will incorporate various dimensions that contribute to the pricing of houses enabling better decision-making for both buyers and sellers. The schema provides insights into how different factors such as property details, location, time, customer preferences, and amenities influence house prices. This will assist customers in making informed decisions by giving them an estimated price range for a particular house based on its attributes.

#### *Dimensions:*

We have identified the following dimensions that need to be incorporated into the star schema for house price prediction:

1. **Property Details Dimension:** Captures attributes of the property such as property type(building, row house, bungalow), number of rooms(BHK), Carpet area(CA),Built-Up area(BA),(SPA) and a binary code for Amenities, where 0 indicates amenity is absent while 1 shows presence. The amenities code is a 6-bit number indicating the status of lift, security services, gymnasium, swimming pool, garden and fire extinguisher in that order.
2. **Location Dimension:** Provide geographic information about the house's location, including area, locality, city, state, pin code.
3. **Time Dimension:** Includes attributes like year, month, quarter and date.
4. **Customer Dimension:** Capture customer-related attributes like name, email-address, phone number, salary range, number of family members.

All these dimensions are associated with an id.

#### *Fact Table:*

The central fact table of your star schema will contain the customer-id, property-id, location-id, time-id, area-id, predicted price, discount price, final price, commission

#### *Star Schema Theory*

A star schema is a type of data modelling technique used in data warehousing to represent data in a structured and intuitive way. In a star schema, data is organized into a central fact table that contains the measures of interest, surrounded by dimension tables that describe the attributes of the measures. The fact table in a star schema contains the measures or metrics that are of interest to the user or organization. The dimension tables in a star schema contain the descriptive attributes of the measures in the fact table. These attributes are used to slice and dice the data in the fact table, allowing users to analyze the data from different perspectives. For example, in a sales data warehouse, the dimension tables might include product, customer, time, and location. In a star schema, each dimension table is joined to the

fact table through a foreign key relationship. This allows users to query the data in the fact table using attributes from the dimension tables.

#### Advantages of Star Schema:

- **Simpler Queries** –Join logic of star schema is quite cinch in comparison to other join logic which are needed to fetch data from a transactional schema that is highly normalized.
- **Simplified Business Reporting Logic** – In comparison to a transactional schema that is highly normalized, the star schema makes simpler common business reporting logic, such as of reporting and period-over-period.
- **Feeding Cubes** – Star schema is widely used by all OLAP systems to design OLAP cubes efficiently. In fact, major OLAP systems deliver a ROLAP mode of operation which can use a star schema as a source without designing a cube structure.

#### Disadvantages of Star Schema:

- Data integrity is not enforced well since in a highly de-normalized schema state.
- Not flexible in terms if analytical needs as a normalized data model.
- Star schemas don't reinforce many-to-many relationships within business entities – at least not frequently.

#### Snowflake Schema Theory

The snowflake schema is a variant of the star schema. Here, the centralized fact table is connected to multiple dimensions. In the snowflake schema, dimensions are present in a normalized form in multiple related tables. The snowflake structure materialized when the dimensions of a star schema are detailed and highly structured, having several levels of relationship, and the child tables have multiple parent tables. The snowflake effect affects only the dimension tables and does not affect the fact tables. In a snowflake schema, the fact table is still located at the center of the schema, surrounded by the dimension tables. However, each dimension table is further broken down into multiple related tables, creating a hierarchical structure that resembles a snowflake.

The snowflake design is the result of further expansion and normalization of the dimension table. In other words, a dimension table is said to be snowflaked if the low-cardinality attribute of the dimensions has been divided into separate normalized tables. These tables are then joined to the original dimension table with referential constraints (foreign key constrain).

Generally, snowflaking is not recommended in the dimension table, as it hampers the understandability and performance of the dimension model as more tables would be required to be joined to satisfy the queries. The main difference between star schema and snowflake schema is that the dimension table of the snowflake schema is maintained in the normalized form to reduce redundancy. The advantage here is that such tables (normalized) are easy to maintain and save storage space. However, it also means that more joins will be needed to execute the query. This will adversely impact system performance.

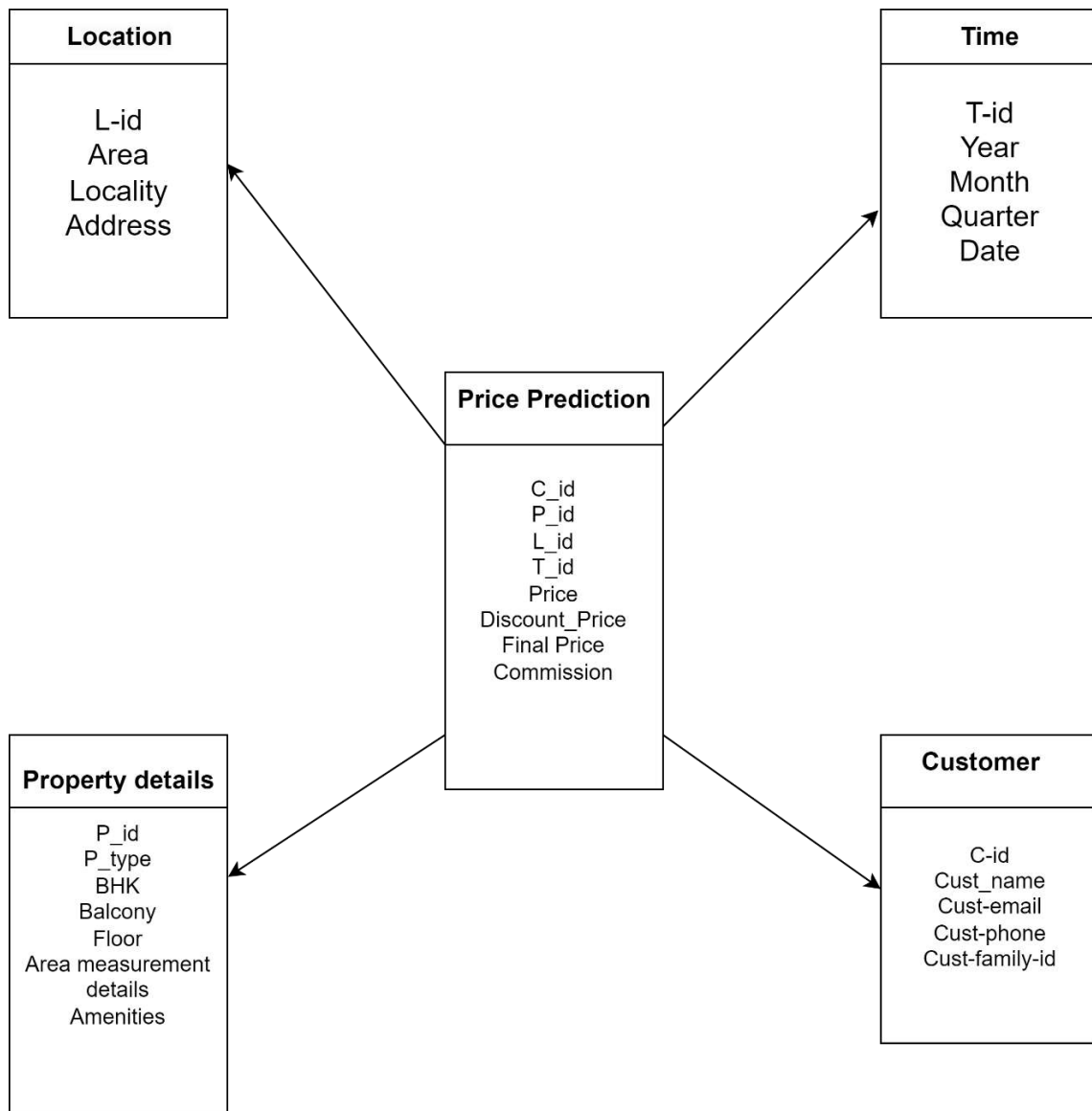
#### Advantages of Snowflake Schema

- It provides structured data which reduces the problem of data integrity.
- It uses small disk space because data are highly structured.

#### Disadvantages of Snowflake Schema

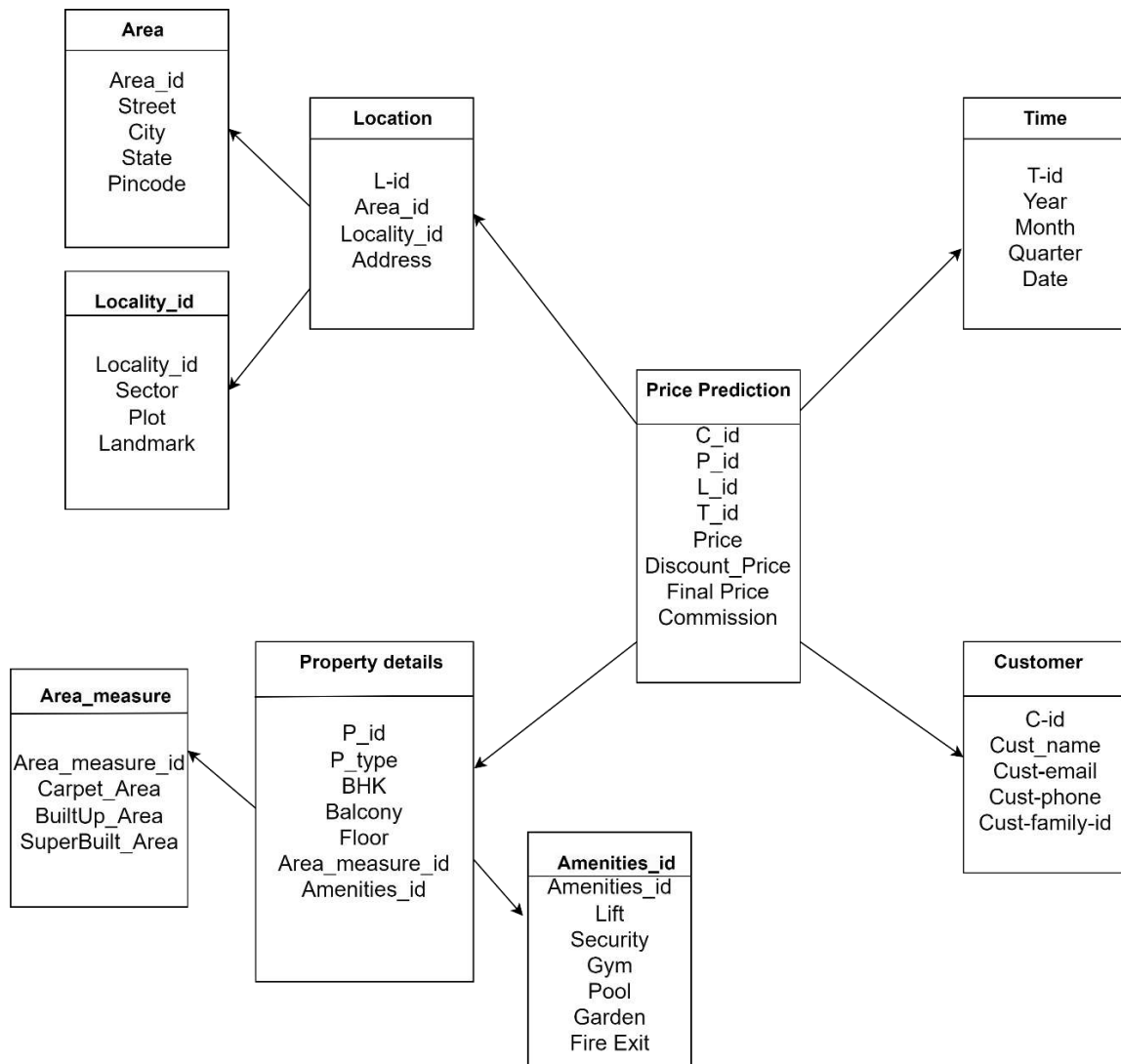
- Snowflaking reduces space consumed by dimension tables but compared with the entire data warehouse the saving is usually insignificant.
- Avoid snowflaking or normalization of a dimension table, unless required and appropriate.
- Do not snowflake hierarchies of dimension table into separate tables. Hierarchies should belong to the dimension table only and should never be snowflakes.
- Multiple hierarchies that can belong to the same dimension have been designed at the lowest possible detail.

#### STAR Schema





## Snowflake Schema



## EXPERIMENT NO : 3

**AIM:** Implementation of all dimension table and fact table based on experiment 2 case study.

**THEORY:** Fact tables and dimension tables are key components of a schema in a data warehouse. A fact table contains records that combine attributes from different dimension tables. These records allow users to analyze different aspects of their business, which can aid in decision-making and improving the business.

**Fact Table:** In a data warehouse, a fact table is a table that stores the measurements, metrics, or facts related to a business operation. It is located at the center of a star or snowflake schema and is surrounded by dimension tables.

- When multiple fact tables are used, they can be organized using a "fact constellation schema."
- A fact table has two types of columns: those that contain the facts and those that serve as foreign keys linking to dimension tables.
- The primary key of a fact table is often a composite key made up of all of the foreign keys in the table.
- Fact tables can hold various types of measurements, such as additive, non-additive, and partly additive measures, and store important information in the data warehouse.
- They are useful for evaluating dimensional attributes because they provide additive values that can act as independent variables.

**Dimension tables:** on the other hand, provide the context and background information for the measures recorded in the fact table. One of the main differences between fact tables and dimension tables is that dimension tables contain the attributes that the measures in the fact table are based on.

Dimension tables contain descriptions of the objects in a fact table and provide information about dimensions such as values, characteristics, and keys.

- These tables are usually small, with a number of rows ranging from a few hundred to a few thousand.
- The term "dimension table" refers to a set of data related to any quantifiable event and is the foundation for dimensional modeling.
- Dimension tables have a column that serves as a primary key, allowing each dimension row or record to be uniquely identified. This key is used to link the dimension table to the fact tables. A surrogate key, which is a system-generated key, is often used to uniquely identify the rows in the dimension table.

## DIFFERENCE BETWEEN FACT TABLE AND DIMENSION TABLE:

Features	Fact Table	Dimension Table
<b>Basic</b>	It has the values of the dimension table's attributes	It has the attributes that are used to calculate the metric in the fact table.
<b>Records</b>	It stores more records.	It stores fewer records.
<b>Attributes</b>	It has fewer attributes.	It has more attributes.
<b>Creation</b>	The fact table comes after the dimension table.	It comes before the fact table.
<b>Attribute format</b>	The fact table's attribute format is in both numerical and text format.	Dimension table's attribute format is only in text format.
<b>Schema</b>	The fact table has fewer numbers in the schema.	The dimension table has more numbers in the schema.
<b>Key</b>	It has a concatenated key, which is a mixture of all primary keys of the entire dimension table.	It has a primary key that is used to find each and every record in the table.
<b>Purpose</b>	Its main task is to analyze the purpose and decision-making.	Its main task is to hold the data about an organization and its process.
<b>Location</b>	It is situated in the center of the snowflake or star schema.	It is situated on the edges of the snowflake or star schema.
<b>Table Growth</b>	It grows vertically.	It grows horizontally.
<b>Hierarchy</b>	It doesn't contain a hierarchy.	It has a hierarchy.

## CODE:

### PROPERTY DETAILS DIMENSION TABLE

```
CREATE TABLE property_details (  
  Property_id INT PRIMARY KEY,  
  property_type VARCHAR2(255) NOT NULL,  
  bhk INT NOT NULL,  
  floor INT NOT NULL,  
  carpet_area INT NOT NULL,  
  buildup_area INT NOT NULL  
);
```

```
DESC property_details;
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,  
buildup_area)  
VALUES (1, 'Apartment', 2, 5, 900, 1000);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,  
buildup_area)  
VALUES (2, 'House', 3, 2, 1500, 1800);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (3, 'Apartment', 1, 8, 600, 650);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (4, 'House', 4, 1, 2000, 2400);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (5, 'Villa', 5, 3, 2500, 2800);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (6, 'Apartment', 2, 4, 950, 1100);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (7, 'House', 3, 2, 1400, 1600);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (8, 'Apartment', 1, 6, 700, 750);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (9, 'House', 4, 3, 1800, 2100);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (10, 'Villa', 5, 2, 2200, 2600);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (11, 'Apartment', 2, 7, 850, 950);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (12, 'House', 3, 4, 1300, 1500);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (13, 'Apartment', 1, 5, 680, 720);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (14, 'House', 4, 2, 1900, 2200);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (15, 'Villa', 5, 1, 2400, 2700);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (16, 'Apartment', 2, 6, 920, 1050);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (17, 'House', 3, 3, 1350, 1550);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (18, 'Apartment', 1, 4, 670, 710);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (19, 'House', 4, 1, 1950, 2250);
```

```
INSERT INTO property_details (property_id, property_type, bhk, floor, carpet_area,
builtup_area)
VALUES (20, 'Villa', 5, 2, 2300, 2600);
```

PROPERTY_ID	PROPERTY_TYPE	BHK	FLOOR	CARPET_AREA	BUILTUP_AREA
1	Apartment	2	5	900	1000
4	House	4	1	2000	2400
5	Villa	5	3	2500	2800
6	Apartment	2	4	950	1100
8	Apartment	1	6	700	750
9	House	4	3	1800	2100
11	Apartment	2	7	850	950
12	House	3	4	1300	1500



15	Villa	5	1	2400	2700
16	Apartment	2	6	920	1050
17	House	3	3	1350	1550
3	Apartment	1	8	600	650
7	House	3	2	1400	1600
13	Apartment	1	5	680	720
14	House	4	2	1900	2200
2	House	3	2	1500	1800
10	Villa	5	2	2200	2600
18	Apartment	1	4	670	710
19	House	4	1	1950	2250
20	Villa	5	2	2300	2600

### **LOCATION DIMENSION TABLE**

```
CREATE TABLE location (
    loc_id INT PRIMARY KEY,
    locality VARCHAR(255) NOT NULL,
    city VARCHAR(255) NOT NULL,
    state VARCHAR(255) NOT NULL,
    pincode INT NOT NULL
);
```

```
DESC LOCATION;
```

```
INSERT INTO location (loc_id, locality, city, state, pincode) VALUES (1, 'Park Street',
'Mumbai', 'Maharashtra', 700001);
```

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (2, 'MG Road', 'Pune', 'Maharashtra', 560001);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (3, 'Connaught Place', 'Mumbai', 'Maharashtra', 110001);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (4, 'Marine Drive', 'Mumbai', 'Maharashtra', 400001);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (5, 'Anna Nagar', 'Mumbai', 'Maharashtra', 600040);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (6, 'Lalbagh', 'Mumbai', 'Maharashtra', 226001);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (7, 'Sector 17', 'Mumbai', 'Maharashtra', 160017);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (8, 'Bodakdev', 'Mumbai', 'Maharashtra', 380054);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (9, 'Kadavanthra', 'Mumbai', 'Maharashtra', 682020);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (10, 'Gachibowli', 'Mumbai', 'Maharashtra', 500032);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (11, 'Sector 62', 'Mumbai', 'Maharashtra', 201301);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (12, 'Indiranagar', 'Mumbai', 'Maharashtra', 560038);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (13, 'Jawahar Nagar', 'Mumbai', 'Maharashtra', 302004);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (14, 'Ballygunge', 'Mumbai', 'Maharashtra', 700019);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (15, 'Sector 29', 'Mumbai', 'Maharashtra', 122002);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (16, 'Koregaon Park', 'Mumbai', 'Maharashtra', 411001);

INSERT INTO location (loc\_id, locality, city, state, pincode) VALUES (17, 'Malviya Nagar', 'Mumbai', 'Maharashtra', 110017);

```
INSERT INTO location (loc_id, locality, city, state, pincode) VALUES (18, 'Rajajinagar',  
'Mumbai', 'Maharashtra', 560010);
```

```
INSERT INTO location (loc_id, locality, city, state, pincode) VALUES (19, 'Piplod', 'Mumbai',  
'Maharashtra', 395007);
```

```
INSERT INTO location (loc_id, locality, city, state, pincode) VALUES (20, 'Jayanagar',  
'Mumbai', 'Maharashtra', 560041);
```

```
select * from location;
```

LOC_ID	LOCALITY	CITY	STATE	PINCODE
15	Sector 29	Mumbai	Maharashtra	122002
1	Park Street	Mumbai	Maharashtra	700001
3	Connaught Place	Mumbai	Maharashtra	110001
5	Anna Nagar	Mumbai	Maharashtra	600040
6	Lalbagh	Mumbai	Maharashtra	226001
7	Sector 17	Mumbai	Maharashtra	160017
12	Indiranagar	Mumbai	Maharashtra	560038
13	Jawahar Nagar	Mumbai	Maharashtra	302004
14	Ballygunge	Mumbai	Maharashtra	700019
20	Jayanagar	Mumbai	Maharashtra	560041
2	MG Road	Pune	Maharashtra	560001
10	Gachibowli	Mumbai	Maharashtra	500032
18	Rajajinagar	Mumbai	Maharashtra	560010
4	Marine Drive	Mumbai	Maharashtra	400001

8	Bodakdev	Mumbai	Maharashtra	380054
9	Kadavanthra	Mumbai	Maharashtra	682020
11	Sector 62	Mumbai	Maharashtra	201301
16	Koregaon Park	Mumbai	Maharashtra	411001
17	Malviya Nagar	Mumbai	Maharashtra	110017
19	Piplod	Mumbai	Maharashtra	395007

### **TIME DIMENSION TABLE:**

```
CREATE TABLE time (
    time_id INT PRIMARY KEY,
    tmonth VARCHAR(255) NOT NULL,
    tyear VARCHAR(255) NOT NULL,
    tdate DATE NOT NULL
);
```

```
DESC time;
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (1, 1, 2023,
TO_DATE('2023-01-01','YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (2, 2, 2023,
TO_DATE('2023-02-15', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (3, 3, 2023,
TO_DATE('2023-03-10', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (4, 4, 2023,
TO_DATE('2023-04-22', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (5, 5, 2023,
TO_DATE('2023-05-05', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (6, 6, 2023,
TO_DATE('2023-06-18', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (7, 7, 2023,
TO_DATE('2023-07-07', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (8, 8, 2023,
TO_DATE('2023-08-29', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (9, 9, 2023,
TO_DATE('2023-09-14', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (10, 10, 2023,
TO_DATE('2023-10-03', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (11, 11, 2023,
TO_DATE('2023-11-20', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (12, 12, 2023,
TO_DATE('2023-12-12', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (13, 1, 2024,
TO_DATE('2024-01-02', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (14, 2, 2024,
TO_DATE('2024-02-29', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (15, 3, 2024,
TO_DATE('2024-03-08', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (16, 4, 2024,
TO_DATE('2024-04-17', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (17, 5, 2024,
TO_DATE('2024-05-23', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (18, 6, 2024,
TO_DATE('2024-06-30', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (19, 7, 2024,
TO_DATE('2024-07-11', 'YYYY-MM-DD'));
```

```
INSERT INTO time (time_id, tmonth, tyear, tdate) VALUES (20, 8, 2024,
TO_DATE('2024-08-26', 'YYYY-MM-DD'));
```

```
select * from time;
```

TIME_ID	TMONTH	TYEAR	TDATE
1	1	2023	01-JAN-23



8	8	2023	29-AUG-23
14	2	2024	29-FEB-24
15	3	2024	08-MAR-24
2	2	2023	15-FEB-23
3	3	2023	10-MAR-23
4	4	2023	22-APR-23
5	5	2023	05-MAY-23
6	6	2023	18-JUN-23
7	7	2023	07-JUL-23
9	9	2023	14-SEP-23
12	12	2023	12-DEC-23
13	1	2024	02-JAN-24
16	4	2024	17-APR-24
17	5	2024	23-MAY-24
18	6	2024	30-JUN-24
20	8	2024	26-AUG-24
10	10	2023	03-OCT-23
11	11	2023	20-NOV-23
19	7	2024	11-JUL-24

### **Customer Dimension Table:**

```
CREATE TABLE customer (  
    cust_id INT PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    mail VARCHAR(255) NOT NULL,  
    phone VARCHAR(10) NOT NULL,  
    income DECIMAL(10, 2) NOT NULL  
);
```

desc customer;

```
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (1, 'Rajesh Kumar', 'rajesh.kumar@example.com', '9876543210', 60000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (2, 'Priya Sharma', 'priya.sharma@example.com', '8765432109', 55000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (3, 'Amit Verma', 'amit.verma@example.com', '7654321098', 70000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (4, 'Sneha Gupta', 'sneha.gupta@example.com', '6543210987', 80000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (5, 'Rahul Singh', 'rahul.singh@example.com', '5432109876', 75000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (6, 'Neha Patel', 'neha.patel@example.com', '4321098765', 62000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (7, 'Sandeep Joshi', 'sandeep.joshi@example.com', '3210987654', 68000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (8, 'Anita Sharma', 'anita.sharma@example.com', '2109876543', 58000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (9, 'Vikas Yadav', 'vikas.yadav@example.com', '1098765432', 72000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (10, 'Pooja Mishra', 'pooja.mishra@example.com', '9988776655', 65000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (11, 'Alok Tiwari', 'alok.tiwari@example.com', '8877665544', 70000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)  
VALUES  
    (12, 'Meena Reddy', 'meena.reddy@example.com', '7766554433', 54000.00)  
INSERT INTO customer (cust_id, name, mail, phone, income)
```

VALUES

(13, 'Sanjay Kumar', 'sanjay.kumar@example.com', '6655443322', 73000.00)

INSERT INTO customer (cust\_id, name, mail, phone, income)

VALUES

(14, 'Rashmi Singh', 'rashmi.singh@example.com', '5544332211', 62000.00)

INSERT INTO customer (cust\_id, name, mail, phone, income)

VALUES

(15, 'Gaurav Sharma', 'gaurav.sharma@example.com', '4433221100', 71000.00)

select \* from customer;

CUST_ID	NAME	MAIL	PHONE	INCOME
1	Rajesh Kumar	rajesh.kumar@example.com	9876543210	60000
2	Priya Sharma	priya.sharma@example.com	8765432109	55000
3	Amit Verma	amit.verma@example.com	7654321098	70000
4	Sneha Gupta	sneha.gupta@example.com	6543210987	80000
5	Rahul Singh	rahul.singh@example.com	5432109876	75000
8	Anita Sharma	anita.sharma@example.com	2109876543	58000
9	Vikas Yadav	vikas.yadav@example.com	1098765432	72000
11	Alok Tiwari	alok.tiwari@example.com	8877665544	70000
12	Meena Reddy	meena.reddy@example.com	7766554433	54000
7	Sandeep Joshi	sandeep.joshi@example.com	3210987654	68000
15	Gaurav Sharma	gaurav.sharma@example.com	4433221100	71000
6	Neha Patel	neha.patel@example.com	4321098765	62000
10	Pooja Mishra	pooja.mishra@example.com	9988776655	65000

13	Sanjay Kumar	sanjay.kumar@example.com	6655443322	73000
14	Rashmi Singh	rashmi.singh@example.com	5544332211	62000

### **FACT TABLE:**

CREATE TABLE fact\_table AS

SELECT

l.loc\_id,

p.property\_id,

t.time\_id,

-- Generate random price values between 100,000 and 1,000,000

FLOOR(DBMS\_RANDOM.VALUE() \* (10000000 - 100000 + 1) + 100000) AS price,

-- Generate random commission values between 5% and 10% of the price

FLOOR(DBMS\_RANDOM.VALUE() \* (10 - 5 + 1) + 5) / 100 \*

(FLOOR(DBMS\_RANDOM.VALUE() \* (1000000 - 100000 + 1) + 100000)) AS commission

FROM

location l

CROSS JOIN

property\_details p

CROSS JOIN

time t;

select \* from fact\_table

LOC_ID	PROPERTY_ID	TIME_ID	PRICE	COMMISSION
1	1	1	5875298	34788.8
1	1	2	6253151	55434.26
1	1	3	6415087	45699.29
1	1	4	3623252	70724.16
1	1	5	1416609	12082.63
1	1	6	8806155	8627.64

1	1	7	7135856	14627.05
1	1	8	6469149	85501.3
1	1	9	5772846	7987.8
1	1	10	2163982	6063.5
1	1	11	7886682	65783.36
1	1	12	2169094	54177.21
1	1	13	8971864	28221.44
1	1	14	7996135	99960.4
1	1	15	617785	78293.68
1	1	16	4887716	44246.25
1	1	17	6185813	9009.42
1	1	18	6754681	19332.36
1	1	19	2115813	40054.3
1	1	20	4147895	62190.72
1	2	1	7945066	61911.15
1	2	2	4226526	28441.84
1	2	3	4346223	5404.55
1	2	4	3679149	40903.83
1	2	5	351425	64951.29
1	2	6	1983332	38135.25

1	2	7	2813011	37455.78
1	2	8	780203	50138.6
1	2	9	2778247	72065.6
1	2	10	5857147	50786.72
1	2	11	2222743	85518.81
1	2	12	1609332	31940.82
1	2	13	2906402	42939.75
1	2	14	9854385	54821.8
1	2	15	8307228	16836
1	2	16	7254257	46993.3
1	2	17	9834032	28679.63
1	2	18	3678828	51773.26
1	2	19	1116332	23655.75
1	2	20	5656102	26448.5
1	3	1	8793927	59675.68
1	3	2	9541309	72181.52
1	3	3	8755177	19187.16
1	3	4	2686157	58493.1
1	3	5	2420154	42254.1
1	3	6	7081841	69191.01

1	3	7	8584514	31261.74
1	3	8	434019	18591.7
1	3	9	3118963	46583.16
1	3	10	3656562	26233.65

# BEFORE SLICE:

LOC_ID	PROPERTY_ID	TIME_ID	PRICE	COMMISSION
1	1	1	5875298	34788.8
1	1	2	6253151	55434.26
1	1	3	6415087	45699.29
1	1	4	3623252	70724.16
1	1	5	1416609	12082.63
1	1	6	8806155	8627.64
1	1	7	7135856	14627.05
1	1	8	6469149	85501.3
1	1	9	5772846	7987.8
1	1	10	2163982	6063.5
1	1	11	7886682	65783.36
1	1	12	2169094	54177.21

1	1	13	8971864	28221.44
1	1	14	7996135	99960.4
1	1	15	617785	78293.68

**AFTER SLICING :** Slicing based on location id 1

CREATE MATERIALIZED VIEW mv\_slice AS

SELECT \*

FROM fact\_table

WHERE loc\_id = 1;

SELECT \* FROM mv\_slice;

LOC_ID	PROPERTY_ID	TIME_ID	PRICE	COMMISSION
1	1	1	9052938	21898.5
1	1	2	944333	23203.3
1	1	3	8914803	19354.8
1	1	4	5003094	77314.95
1	1	5	1478730	40476.17
1	1	6	6650524	91185
1	1	7	7436204	37225.35
1	1	8	5647765	58056.72
1	1	9	9482649	53069.7
1	1	10	7286817	77813.92
1	1	11	4126820	38345.04



1	1	12	8169456	75662.19
1	1	13	7363031	54741.52
1	1	14	3417102	95547.3
1	1	15	6371278	98795
1	1	16	7569909	52527.78
1	1	17	9711570	99187.6
1	1	18	9909063	32094.55
1	1	19	9207394	27303.7
1	1	20	7752388	73007.46
1	2	1	3804140	51947.4
1	2	2	6720825	34061.4
1	2	3	6697205	87839.9
1	2	4	1887490	88527.6
1	2	5	8540552	69697.67
1	2	6	7057018	29985.66
1	2	7	8273816	98611.6
1	2	8	4748294	75801.78
1	2	9	5760218	28359.36
1	2	10	6175857	53375.92
1	2	11	6170731	41326.6

1	2	12	5050637	14704.8
1	2	13	4754795	49648.74
1	2	14	8529828	31404.2
1	2	15	7368122	38653.74
1	2	16	9863121	47983.95
1	2	17	558172	71425.71
1	2	18	1850686	34511.1
1	2	19	6187796	66144
1	2	20	2722895	54514.92
1	3	1	1583785	58957.86
1	3	2	6754233	9045.36
1	3	3	973094	42414.36
1	3	4	8054755	26000.48
1	3	5	6851831	61427.92
1	3	6	3571619	32848.83
1	3	7	3129371	13203.04
1	3	8	1929017	76275.27
1	3	9	1666221	8899.2
1	3	10	4367686	32192.28

**BEFORE DICE:**

LOC_ID	PROPERTY_ID	TIME_ID	PRICE	COMMISSION
1	1	1	5875298	34788.8
1	1	2	6253151	55434.26
1	1	3	6415087	45699.29
1	1	4	3623252	70724.16
1	1	5	1416609	12082.63
1	1	6	8806155	8627.64
1	1	7	7135856	14627.05
1	1	8	6469149	85501.3
1	1	9	5772846	7987.8
1	1	10	2163982	6063.5
1	1	11	7886682	65783.36
1	1	12	2169094	54177.21
1	1	13	8971864	28221.44
1	1	14	7996135	99960.4
1	1	15	617785	78293.68

**AFTER DICE:** Dicing based on location id 1 and property id 2

CREATE MATERIALIZED VIEW mv\_dice AS

SELECT \*

FROM fact\_table

WHERE loc\_id = 1 AND property\_id = 2;

SELECT \* FROM mv\_dice;

LOC_ID	PROPERTY_ID	TIME_ID	PRICE	COMMISSION
1	2	1	3804140	51947.4
1	2	2	6720825	34061.4
1	2	3	6697205	87839.9
1	2	4	1887490	88527.6
1	2	5	8540552	69697.67
1	2	6	7057018	29985.66
1	2	7	8273816	98611.6
1	2	8	4748294	75801.78
1	2	9	5760218	28359.36
1	2	10	6175857	53375.92
1	2	11	6170731	41326.6
1	2	12	5050637	14704.8
1	2	13	4754795	49648.74
1	2	14	8529828	31404.2
1	2	15	7368122	38653.74

1	2	16	9863121	47983.95
1	2	17	558172	71425.71
1	2	18	1850686	34511.1
1	2	19	6187796	66144
1	2	20	2722895	54514.92

# **PIVOT:**

CREATE MATERIALIZED VIEW mv\_pivot AS

SELECT

loc\_id,

MAX(CASE WHEN measure = 'Price' THEN value END) AS "Price",

MAX(CASE WHEN measure = 'Commission' THEN value END) AS "Commission"

FROM (

SELECT loc\_id, measure, value

FROM fact\_table

UNPIVOT INCLUDE NULLS (

value FOR measure IN (price AS 'Price', commission AS 'Commission')

)

)

GROUP BY loc\_id;

SELECT \* FROM mv\_pivot;

LOC_ID	Price	Commission
6	9995909	98284
14	9997296	99585.2
1	9996910	99718.6
7	9985666	95592.5
15	9991567	96400
2	9984496	99074.7

8	9995574	97922
11	9944100	99761.5
12	9971596	99662.3
17	9989337	98687.2
4	9998921	98662.6
5	9962178	99280.8
10	9922671	98133.8
18	9925171	97514.7
3	9988166	99200.6
9	9985697	97497.2
20	9994425	99118.5
13	9996290	97881.4
19	9996403	97271.6
16	9948577	98605.2

# BEFORE ROLLUP:

LOC_ID	PROPERTY_ID	TIME_ID	PRICE	COMMISSION
1	1	1	5875298	34788.8
1	1	2	6253151	55434.26
1	1	3	6415087	45699.29

1	1	4	3623252	70724.16
1	1	5	1416609	12082.63
1	1	6	8806155	8627.64
1	1	7	7135856	14627.05
1	1	8	6469149	85501.3
1	1	9	5772846	7987.8
1	1	10	2163982	6063.5
1	1	11	7886682	65783.36
1	1	12	2169094	54177.21
1	1	13	8971864	28221.44
1	1	14	7996135	99960.4
1	1	15	617785	78293.68

#### AFTER ROLLUP:

```

SELECT
  t.time_id,
  t.tmonth,
  t.tyear,
  SUM(ft.price)      AS      total_price,
  SUM(ft.commission) AS total_commission
FROM
  fact_table ft
JOIN
  time t ON ft.time_id = t.time_id
GROUP BY
  ROLLUP(t.tyear, t.tmonth, t.time_id)
HAVING
  t.time_id IS NOT NULL

```

AND t.tmonth IS NOT NULL  
ORDER BY  
t.tyear, t.tmonth, t.time\_id;

TIME_ID	TMONTH	TYEAR	TOTAL_PRICE	TOTAL_COMMISSION
1	1	2023	2101931826	16054151.42
10	10	2023	2066125677	16261859.5
11	11	2023	2004875961	16886698.53
12	12	2023	1962606227	16427641.04
2	2	2023	1976930790	17296714.17
3	3	2023	2141258409	16451173.29
4	4	2023	2063359066	16989850.22
5	5	2023	1945655186	16985115.77
6	6	2023	1947597776	16221779.82
7	7	2023	2139520417	16567968.25
8	8	2023	1964822828	16264206.68
9	9	2023	2051915543	15987983.58
13	1	2024	1877075336	16591696.08
14	2	2024	1975987593	16005548.02
15	3	2024	2041326917	17349342.08
16	4	2024	2037258268	15832799.57
17	5	2024	2017940068	16613674.28



18	6	2024	2079796036	16404641.14
19	7	2024	2054141046	15980059.78
20	8	2024	2034312589	16689025.74

#### BEFORE DRILL DOWN:

LOC_ID	PROPERTY_ID	TIME_ID	PRICE	COMMISSION
1	1	1	5875298	34788.8
1	1	2	6253151	55434.26
1	1	3	6415087	45699.29
1	1	4	3623252	70724.16
1	1	5	1416609	12082.63
1	1	6	8806155	8627.64
1	1	7	7135856	14627.05
1	1	8	6469149	85501.3
1	1	9	5772846	7987.8
1	1	10	2163982	6063.5
1	1	11	7886682	65783.36
1	1	12	2169094	54177.21
1	1	13	8971864	28221.44
1	1	14	7996135	99960.4

1	1	15	617785	78293.68
---	---	----	--------	----------

#### AFTER DRILL DOWN:

It groups properties by property\_type, bhk, and property\_id, providing a count for each combination.

```
SELECT
  property_type,
  bhk,
  property_id,
  COUNT(*) AS property_count
FROM
  property_details
GROUP BY
  ROLLUP(property_type, bhk, property_id);
```

PROPERTY_TYPE	BHK	PROPERTY_ID	PROPERTY_COUNT
House	3	1	1
House	3	1	1
House	3	1	1
House	3	1	1
House	3	1	4
House	4	1	1
House	4	1	1
House	4	1	1
House	4	1	1
House	4	1	4
House	-	1	8

Villa	5	1	1
Villa	5	1	1
Villa	5	1	1
Villa	5	1	1
Villa	5	1	4
Villa	-	1	4
Apartment	1	1	1
Apartment	1	1	1
Apartment	1	1	1
Apartment	1	1	1
Apartment	1	1	4
Apartment	2	1	1
Apartment	2	1	1
Apartment	2	1	1
Apartment	2	1	1

## EXPERIMENT NO : 4

**AIM:** Implementation of OLAP Operations : Slice, Dice, RollUp, Drilldown, Pivot based on experiment 2.

**THEORY:** OLAP stands for On-Line Analytical Processing. OLAP is a classification of software technology which authorises analysts, managers, and executives to gain insight into information through fast, consistent, interactive access in a wide variety of possible views of data that has been transformed from raw information to reflect the real dimensionality of the enterprise as understood by the clients.

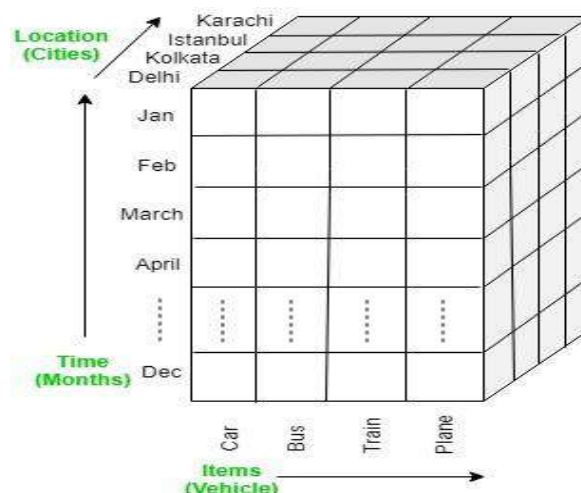
OLAP implements the multidimensional analysis of business information and support the capability for complex estimations, trend analysis, and sophisticated data modeling. It is rapidly enhancing the essential foundation for Intelligent Solutions containing Business Performance Management, Planning, Budgeting, Forecasting, Financial Documenting, Analysis, Simulation-Models, Knowledge Discovery, and Data Warehouses Reporting. OLAP enables end-clients to perform ad hoc analysis of record in multiple dimensions, providing the insight and understanding they require for better decision making.

There are several types of OLAP:

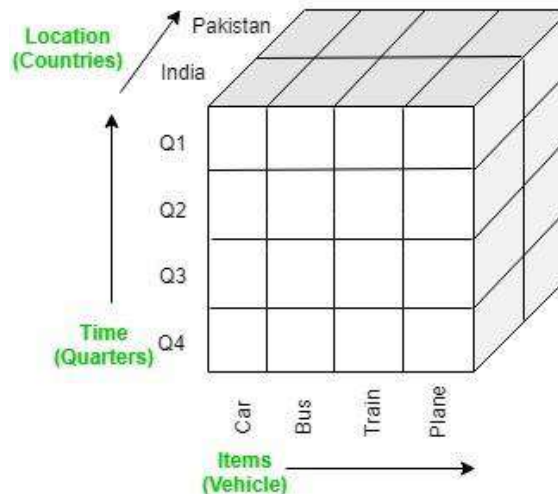
- ROLAP stands for Relational OLAP, an application based on relational DBMSs.
- MOLAP stands for Multidimensional OLAP, an application based on multidimensional DBMSs.
- HOLAP stands for Hybrid OLAP, an application using both relational and multidimensional techniques.

Several Operations on OLAP are:

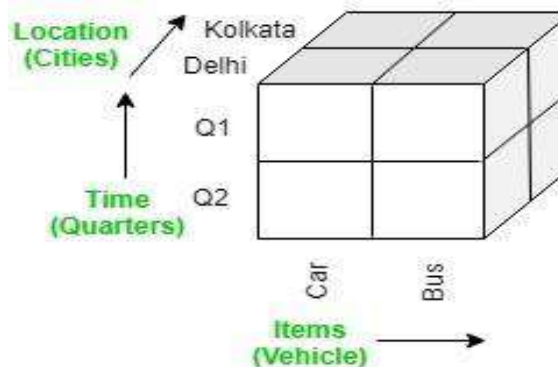
- **Drill down:** In drill-down operation, the less detailed data is converted into highly detailed data. It can be done by:
  - Moving down in the concept hierarchy
  - Adding a new dimension



- **Roll up:** It is just opposite of the drill-down operation. It performs aggregation on the OLAP cube. It can be done by:
  - Climbing up in the concept hierarchy
  - Reducing the dimensions



- **Dice:** It selects a sub-cube from the OLAP cube by selecting two or more dimensions. In the cube given in the overview section, a sub-cube is selected by selecting following dimensions with criteria:
  - Location = "Delhi" or "Kolkata"
  - Time = "Q1" or "Q2"
  - Item = "Car" or "Bus"



- **Slice:** It selects a single dimension from the OLAP cube which results in a new sub-cube creation. In the cube given in the overview section, Slice is performed on the dimension Time = "Q1".

Karachi				
Istanbul				
Kolkata				
Delhi				

Location (Cities) ↑

Items (Vehicle) →

Car Bus Train Plane

- **Pivot:** It is also known as rotation operation as it rotates the current view to get a new view of the representation. In the sub-cube obtained after the slice operation, performing pivot operation gives a new view of it.

Karachi				
Istanbul				
Kolkata				
Delhi				

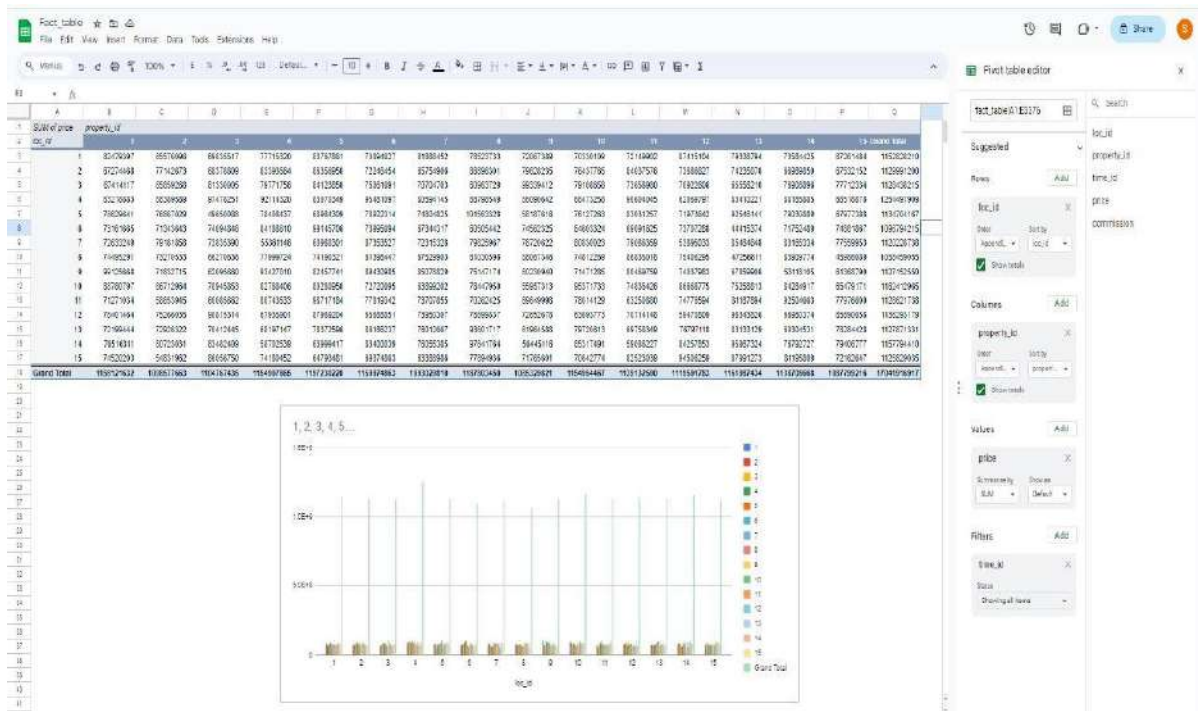
Location (Cities) ↑

Items (Vehicle) →

Car Bus Train Plane

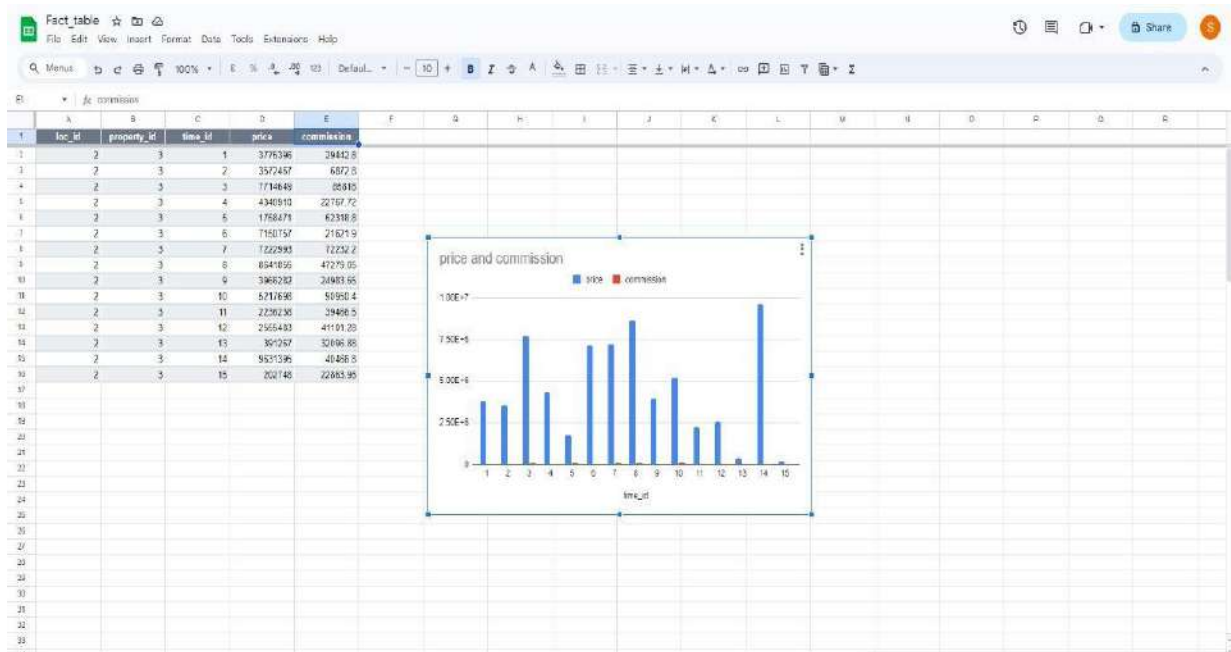
## ROLL UP:

The sum of house prices in all the locations and all property types , everytime.



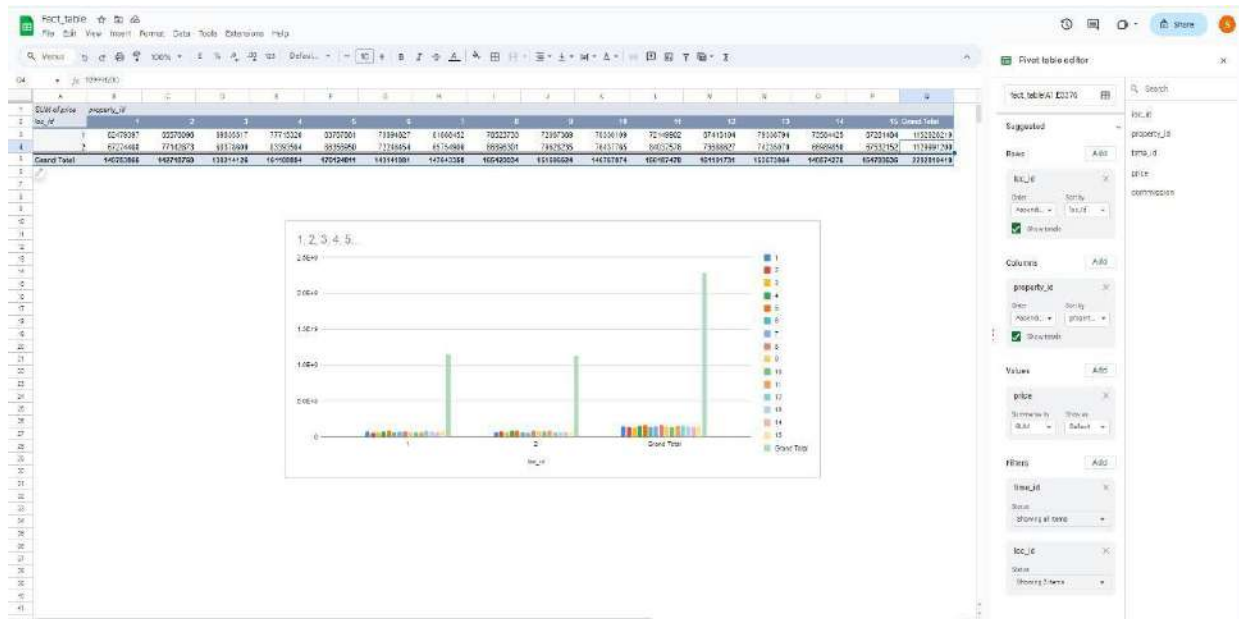
## DRILL DOWN:

The detailed data with location id = 2 and property id =3



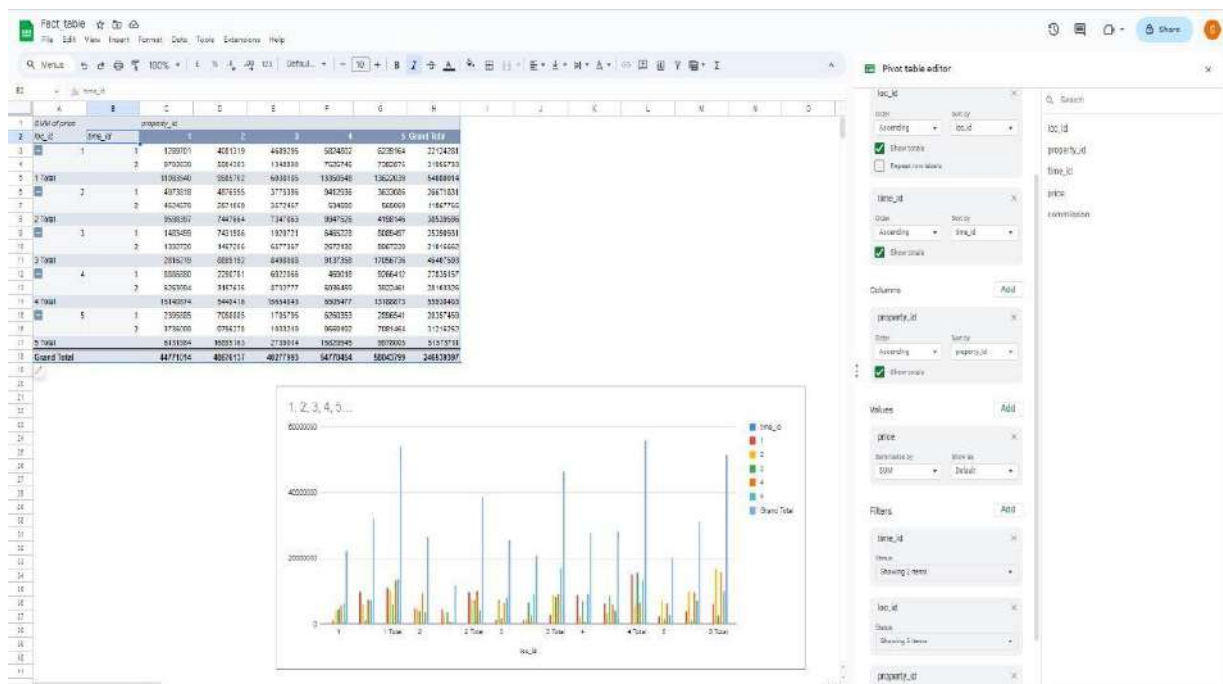
## SLICE:

Slicing the data where location\_id<=2 i.e. location\_id =1 and location\_id=2



## DICE:

Dicing the data by selecting only first 5 location\_id, first 5 property\_id and first 2 time\_id.





## **EXPERIMENT -5**

**Aim-** Implement Naive Bayes Theorem

### **Theory-**

The Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. Naive Bayes Classifier is one of the simplest and most effective Classification algorithms that help in building fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts based on the probability of an object. Some famous examples of Naive Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

The Naive Bayes algorithm is comprised of two words Naive and Bayes, Which can be described as:

- Naive: It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified based on color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability. The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

- $P(A|B)$  is Posterior probability: Probability of hypothesis A on the observed event B.
- $P(B|A)$  is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

- P(A) is Prior Probability: Probability of hypothesis before observing the evidence.
- P(B) is Marginal Probability: Probability of Evidence.

Advantages of Naive Bayes Classifier:

- Naive Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms
- It is the most popular choice for text classification problems.

Disadvantages of Naive Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

The dataset is divided into two parts, namely, feature matrix and the response vector.

- Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of dependent features. In our dataset, features are 'House Price', 'BHK', 'Owner Income' and 'Location'.
- Response vector contains the value of class variable(prediction or output) for each row of feature matrix. In our dataset, the class variable name is 'Purchased house'.

The fundamental Naive Bayes assumption is that each feature makes an independent and equal contribution to the outcome. The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

where, y is class variable and X is a dependent feature vector (of size n) where:

$$X = (x_1, x_2, x_3, \dots, x_n)$$

Now, its time to put a Naive assumption to the Bayes' theorem, which is, independence among the features. So now, we split evidence into the independent parts. Now, if any two events A and B are independent, then,

$$P(A,B) = P(A)P(B)$$

Hence, we reach to the result:

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)}$$

which can be expressed as:

$$P(y|x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i|y)}{P(x_1)P(x_2)\dots P(x_n)}$$

Now, as the denominator remains constant for a given input, we can remove that term:

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y).$$

. For this, we find the probability of given set of inputs for all possible values of the class variable y and pick up the output with maximum probability. This can be expressed mathematically as:

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

Now, consider the example –make a prediction for a customer  $X=(\text{'High'}, 2, \text{'Low'}, \text{'City'})$  -

### **Step 1: Calculate Prior Probabilities**

$$- P(\text{Yes}) = (12 \text{ 'Yes' instances}) / (21 \text{ total instances}) = 12/21 \approx 0.5714$$

$$- P(\text{No}) = (9 \text{ 'No' instances}) / (21 \text{ total instances}) = 9/21 \approx 0.4286$$

### **Step 2: Calculate Likelihood Probabilities**

$$- P(\text{'Price' = 'High' | 'Purchased' = 'Yes'}) = 7/12 \approx 0.5833$$

$$- P(\text{'Price' = 'High' | 'Purchased' = 'No'}) = 0/9 \approx 0.0$$

$$- P(\text{'BHK' = 2} \mid \text{'Purchased' = 'Yes'}) = 6/12 \approx 0.5$$

$$- P(\text{'BHK' = 2} \mid \text{'Purchased' = 'No'}) = 3/9 \approx 0.3333$$

$$- P(\text{'OwnerIncome' = 'Low'} \mid \text{'Purchased' = 'Yes'}) = 5/12 \approx 0.4167$$

$$- P(\text{'OwnerIncome' = 'Low'} \mid \text{'Purchased' = 'No'}) = 4/9 \approx 0.4444$$

$$- P(\text{'Location' = 'City'} \mid \text{'Purchased' = 'Yes'}) = 3/12 \approx 0.25$$

$$- P(\text{'Location' = 'City'} \mid \text{'Purchased' = 'No'}) = 2/9 \approx 0.2222$$

### **Step 3: Calculate the Posterior Probabilities**

For 'Purchased' = 'Yes':

$$P(\text{'Purchased' = 'Yes' / X}) \propto [P(\text{Yes}) * P(\text{'Price' = 'High'} \mid \text{'Purchased' = 'Yes'}) * \\ P(\text{'BHK' = 2} \mid \text{'Purchased' = 'Yes'}) * P(\text{'OwnerIncome' = 'Low'} \mid \text{'Purchased' = 'Yes'}) * \\ P(\text{'Location' = 'City'} \mid \text{'Purchased' = 'Yes'})]$$

$$- P(\text{'Purchased' = 'Yes' | X}) \propto 0.5714 * 0.4167 * 0.5 * 0.4167 * 0.25$$

For 'Purchased' = 'No':

$$P(\text{'Purchased' = 'No' / X}) \propto [P(\text{No}) * P(\text{'Price' = 'High'} \mid \text{'Purchased' = 'No'}) * \\ P(\text{'BHK' = 2} \mid \text{'Purchased' = 'No'}) * P(\text{'OwnerIncome' = 'Low'} \mid \text{'Purchased' = 'No'}) * \\ P(\text{'Location' = 'City'} \mid \text{'Purchased' = 'No'})]$$

$$- P(\text{'Purchased' = 'No' | X}) \propto 0.4286 * 0.2222 * 0.3333 * 0.4444 * 0.2222$$

### **Step 4: Calculate the final Probabilities**

$$- P(\text{'Purchased' = 'Yes' | X}) \approx 0.01240215771$$

$$- P(\text{'Purchased' = 'No' | X}) \approx 0.00313436551$$

Thus, the probability of 'Purchased' = 'Yes' is higher, so the prediction for customer X will be 'Yes'.

### Code:

The dataset (Head):

Price	BHK	OwnerIncome	Location	Purchased
Low	1	Low	Suburbs	Yes
Low	2	High	Suburbs	No
Average2		Average	City	Yes
Low	1	Low	Suburbs	No
High	3	High	Downtown	Yes

```
import pandas as pd
import math

data = pd.read_csv('bayes.csv')

price = input("Enter 'Price' (Low, Average, High): ")
bhk = int(input("Enter 'BHK' (1, 2, 3): "))
owner_income = input("Enter 'OwnerIncome' (Low, Average, High): ")
location = input("Enter 'Location' (Suburbs, City, Downtown): ")

print("\n")

# Prior probabilities
total_instances = len(data)
purchased_yes = len(data[data['Purchased'] == 'Yes'])
purchased_no = len(data[data['Purchased'] == 'No'])

p_yes = round(purchased_yes / total_instances, 4)
```

```

p_no = round(purchased_no / total_instances,4)

print(f"Total occurrences of Yes: {purchased_yes}")
print(f"Total occurrences of No: {purchased_no}")

print(f"Prior probability for Yes {p_yes}")
print(f"Prior probability for No: {p_no}")

# Likelihood probabilities
def calculate_likelihood(attribute, value, purchased):
    subset = data[data['Purchased'] == purchased]
    count = len(subset[subset[attribute] == value])
    total = len(subset)
    return count / total

print("\n")
print("\n")
print("The likelihood probabilities are: ")

p_price_given_yes = round(calculate_likelihood('Price', price,
'Yes'),4)
p_price_given_no =round(calculate_likelihood('Price', price, 'No'),4)
print(f"P(price = {price} / buy='Yes') = {p_price_given_yes}")
print(f"P(price = {price} / buy='No') = {p_price_given_no}")
print("\n")

p_bhk_given_yes =round(calculate_likelihood('BHK', bhk, 'Yes'),4)
p_bhk_given_no = round(calculate_likelihood('BHK', bhk, 'No'),4)
print(f"P(BHK = {bhk} / buy='Yes') = {p_bhk_given_yes}")
print(f"P(BHK = {bhk} / buy='No') = {p_bhk_given_no}")

```

```

print("\n")

p_owner_income_given_yes = round(calculate_likelihood('OwnerIncome',
owner_income, 'Yes'),4)

p_owner_income_given_no = round(calculate_likelihood('OwnerIncome',
owner_income, 'No'),4)

print(f"P(Owner Income = {bhk} / buy='Yes') =
{p_owner_income_given_yes}")

print(f"P(Owner Income = {bhk} / buy='No') =
{p_owner_income_given_no}")

print("\n")

p_location_given_yes = round(calculate_likelihood('Location', location,
'Yes'),4)

p_location_given_no = round(calculate_likelihood('Location', location,
'No'),4)

print(f"P(Location = {location} / buy='Yes') = {p_location_given_yes}")
print(f"P(Location = {location} / buy='No') = {p_location_given_no}")
print("\n")

# Posterior probabilities
print("The posterior probabilities are: ")

p_yes_given_x = round(p_yes * p_price_given_yes * p_bhk_given_yes *
p_owner_income_given_yes * p_location_given_yes,4)

p_no_given_x = round(p_no * p_price_given_no * p_bhk_given_no *
p_owner_income_given_no * p_location_given_no,4)

if p_yes_given_x > p_no_given_x:
    prediction = 'Yes'
else:
    prediction = 'No'

```

```
print(f'Probability of Purchased=Yes: {p_yes_given_x}')
print(f'Probability of Purchased=No: {p_no_given_x}')
print(f'Prediction: {prediction}')
```

## OUTPUT:

Enter 'Price' (Low, Average, High): Low  
Enter 'BHK' (1, 2, 3): 2  
Enter 'OwnerIncome' (Low, Average, High): High  
Enter 'Location' (Suburbs, City, Downtown): City

Total occurrences of Yes: 12  
Total occurrences of No: 9  
Prior probability for Yes 0.5714  
Prior probability for No: 0.4286

The likelihood probabilities are:  
 $P(\text{price} = \text{Low} / \text{buy} = \text{'Yes'}) = 0.0833$   
 $P(\text{price} = \text{Low} / \text{buy} = \text{'No'}) = 0.7778$

$P(\text{BHK} = 2 / \text{buy} = \text{'Yes'}) = 0.6667$   
 $P(\text{BHK} = 2 / \text{buy} = \text{'No'}) = 0.1111$

$P(\text{Owner Income} = 2 / \text{buy} = \text{'Yes'}) = 0.5$   
 $P(\text{Owner Income} = 2 / \text{buy} = \text{'No'}) = 0.1111$

$P(\text{Location} = \text{City} / \text{buy} = \text{'Yes'}) = 0.3333$   
 $P(\text{Location} = \text{City} / \text{buy} = \text{'No'}) = 0.3333$

The posterior probabilities are:  
Probability of Purchased=Yes: 0.0053  
Probability of Purchased=No: 0.0014  
Prediction: Yes

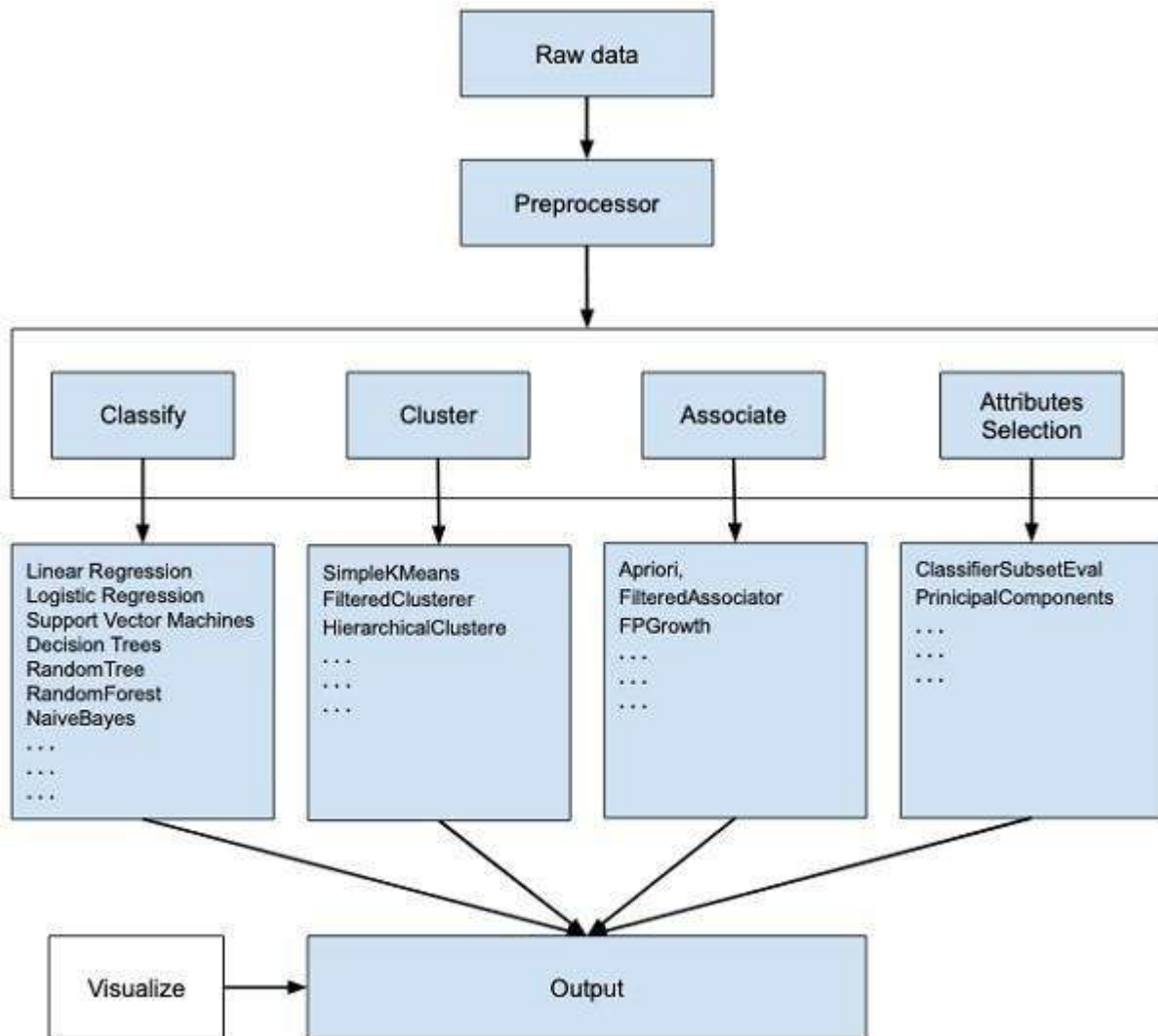


## **EXPERIMENT 6**

**Aim-** Use WEKA tool to perform Clustering, Classification, Association Algorithm

### **Theory-**

WEKA - an open source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualization tools so that you can develop machine learning techniques and apply them to real-world data mining problems. What WEKA offers is summarized in the following diagram –



First, you will start with the raw data collected from the field. This data may contain several null values and irrelevant fields. You use the data preprocessing tools provided in WEKA to cleanse the data.

Then, you would save the preprocessed data in your local storage for applying ML algorithms.

Next, depending on the kind of ML model that you are trying to develop you would select one of the options such as Classify, Cluster, or Associate. The Attributes Selection allows the automatic selection of features to create a reduced dataset.

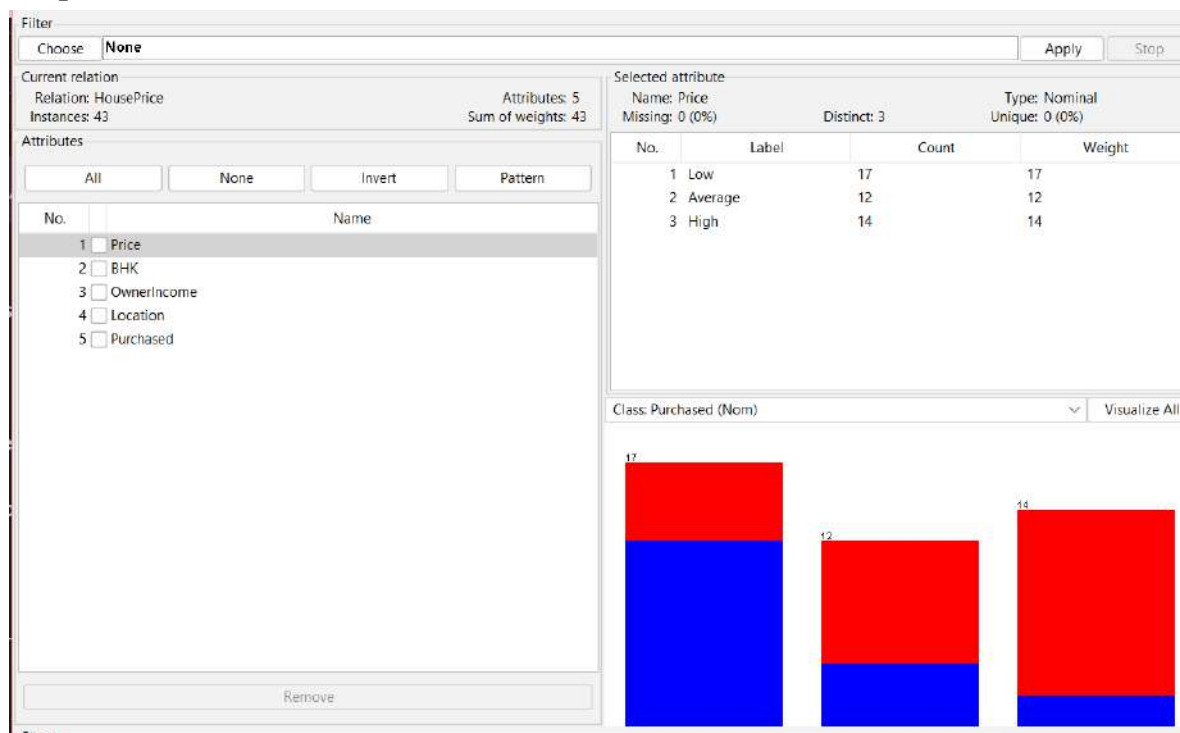
Note that under each category, WEKA provides the implementation of several algorithms. You would select an algorithm of your choice, set the desired parameters and run it on the dataset.

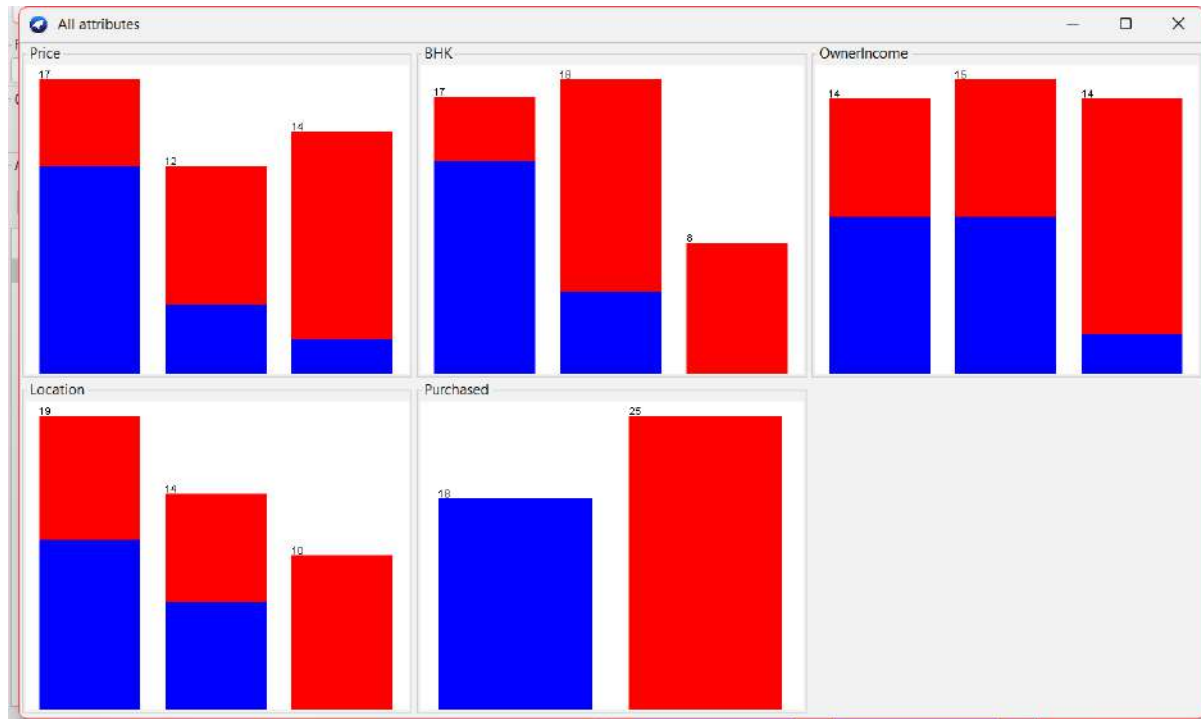
Then, WEKA would give you the statistical output of the model processing. It provides you a visualization tool to inspect the data.

The various models can be applied on the same dataset. You can then compare the outputs of different models and select the best that meets your purpose.

Thus, the use of WEKA results in a quicker development of machine learning models on the whole.

### **Output:**





## CLASSIFICATION-

- NAIVE BAYES

=== Run information ===

Scheme: weka.classifiers.bayes.NaiveBayes

Relation: HousePrice

Instances: 43

Attributes: 5

Price

BHK

OwnerIncome

Location

Purchased

Test mode: 10-fold cross-validation

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

Naive Bayes Classifier

Class

Attribute No Yes

(0.42) (0.58)

=====

Price

Low 13.0 6.0

Average 5.0 9.0

High	3.0	13.0
[total]	21.0	28.0

#### BHK

1	14.0	5.0
2	6.0	14.0
3	1.0	9.0
[total]	21.0	28.0

#### OwnerIncome

Low	9.0	7.0
Average	9.0	8.0
High	3.0	13.0
[total]	21.0	28.0

#### Location

Suburbs	12.0	9.0
City	8.0	8.0
Downtown	1.0	11.0
[total]	21.0	28.0

Time taken to build model: 0 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	32	74.4186 %
Incorrectly Classified Instances	11	25.5814 %
Kappa statistic	0.4941	
Mean absolute error	0.2917	
Root mean squared error	0.4378	
Relative absolute error	59.6401 %	
Root relative squared error	88.4302 %	
Total Number of Instances	43	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
Class								
	0.833	0.320	0.652	0.833	0.732	0.508	0.791	No
	0.680	0.167	0.850	0.680	0.756	0.508	0.791	Yes
Weighted Avg.	0.744	0.231	0.767	0.744	0.746	0.508	0.791	0.769

=== Confusion Matrix ===

a b <-- classified as

15 3 | a = No

8 17 | b = Yes

- ID3 – DECISION TREE

Scheme: weka.classifiers.trees.Id3

Relation: HousePrice

Instances: 43

Attributes: 5

Price

BHK

OwnerIncome

Location

Purchased

Test mode: 10-fold cross-validation

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

Id3

BHK = 1

| OwnerIncome = Low

| | Price = Low

| | | Location = Suburbs: No

| | | Location = City: No

| | | Location = Downtown: null

| | Price = Average: No

| | Price = High: null

| OwnerIncome = Average

| | Location = Suburbs

| | | Price = Low: No

| | | Price = Average: null

| | | Price = High: No

| | Location = City: No

| | Location = Downtown: null

| OwnerIncome = High: Yes

BHK = 2

| Location = Suburbs

| | Price = Low

| | | OwnerIncome = Low: No

| | | OwnerIncome = Average: Yes

| | | OwnerIncome = High: No

| | Price = Average: Yes

| | Price = High: Yes

| Location = City

| | Price = Low: No

| | Price = Average

```

| | | OwnerIncome = Low: Yes
| | | OwnerIncome = Average: Yes
| | | OwnerIncome = High: null
| | Price = High: No
| Location = Downtown: Yes
BHK = 3: Yes

```

Time taken to build model: 0 seconds

=== Stratified cross-validation ===

=== Summary ===

```

Correctly Classified Instances      29      67.4419 %
Incorrectly Classified Instances    13      30.2326 %
Kappa statistic                    0.3636
Mean absolute error                 0.3929
Root mean squared error             0.5722
Relative absolute error             81.9799 %
Root relative squared error        116.5666 %
UnClassified Instances              1       2.3256 %
Total Number of Instances          43

```

=== Detailed Accuracy By Class ===

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
No	0.611	0.250	0.647	0.611	0.629	0.364	0.619	0.492
Yes	0.750	0.389	0.720	0.750	0.735	0.364	0.589	0.668
Weighted Avg.	0.690	0.329	0.689	0.690	0.689	0.364	0.602	0.593

=== Confusion Matrix ===

```

a b <-- classified as
11 7 | a = No
6 18 | b = Yes

```

## **CLUSTERING**

- **K-MEANS CLUSTERING**

=== Run information ===

```

Scheme:   weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning
10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last"
-I 500 -num-slots 1 -S 10
Relation: HousePrice
Instances: 30
Attributes: 2

```

Price  
Area  
Test mode: split 70% train, remainder test

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

kMeans  
=====

Number of iterations: 6  
Within cluster sum of squared errors: 1.3317479191438764

Initial starting points (random):

Cluster 0: 650000,1300  
Cluster 1: 850000,1700

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Cluster#		
	Full Data	0	1
	(30.0)	(15.0)	(15.0)
=====			
Price	1225000	850000	1600000
Area	2450	1700	3200

Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on test split ===

kMeans  
=====

Number of iterations: 5  
Within cluster sum of squared errors: 0.9822588126159557

Initial starting points (random):

Cluster 0: 1900000,3800  
Cluster 1: 1350000,2700

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Cluster#	
	0	1
Full Data	(21.0)	(10.0)
	(10.0)	(11.0)
=====		
Price	1204761.9048	1575000 868181.8182
Area	2409.5238	3150 1736.3636

Time taken to build model (percentage split) : 0 seconds

Clustered Instances

0 5 ( 56%)  
1 4 ( 44%)

#### • HEIRARCHICAL CLUSTERING

=== Run information ===

Scheme: weka.clusterers.HierarchicalClusterer -N 2 -L SINGLE -P -A  
"weka.core.EuclideanDistance -R first-last"

Relation: HousePrice

Instances: 30

Attributes: 2

Price

Area

Test mode: evaluate on training data

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

Cluster 0

(((((1000.0:0.04877,1100.0:0.04877):0,(1200.0:0.04877,(1300.0:0.04877,1400.0:0.04877):0):0,(1500.0:0.04877,1600.0:0.04877):0):0,(((1700.0:0.04877,1800.0:0.04877):0,1900.0:0.04877):0,(2000.0:0.04877,2100.0:0.04877):0):0):0,((2200.0:0.04877,(2300.0:0.04877,2400.0:0.04877):0):0,(2500.0:0.04877,2600.0:0.04877):0,2700.0:0.04877):0):0,((2800.0:0.04877,2900.0:0.04877):0,((3000.0:0.04877,3100.0:0.04877):0,3200.0:0.04877):0):0)

Cluster 1

(((((3300.0:0.04877,3400.0:0.04877):0,3500.0:0.04877):0,(3600.0:0.04877,3700.0:0.04877):0):0,(3800.0:0.04877,3900.0:0.04877):0)



Time taken to build model (full training data) : 0 seconds

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

Clustered Instances

0 23 ( 77%)

1 7 ( 23%)

## **ASSOCIATION**

- Apriori

=== Run information ===

Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.6 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Relation: HousePrice

Instances: 43

Attributes: 5

Price

BHK

OwnerIncome

Location

Purchased

=== Associator model (full training set) ===

Apriori

=====

Minimum support: 0.25 (11 instances)

Minimum metric <confidence>: 0.6

Number of cycles performed: 15

Generated sets of large itemsets:

Size of set of large itemsets L(1): 12

Size of set of large itemsets L(2): 10

Best rules found:

1. Price=High 14 ==> Purchased=Yes 12 <conf:(0.86)> lift:(1.47) lev:(0.09) [3]  
conv:(1.95)

2. OwnerIncome=High 14 ==> Purchased=Yes 12 <conf:(0.86)> lift:(1.47) lev:(0.09) [3]  
conv:(1.95)

3. OwnerIncome=High 14 ==> Price=High 11 <conf:(0.79)> lift:(2.41) lev:(0.15) [6]  
conv:(2.36)
4. Price=High 14 ==> OwnerIncome=High 11 <conf:(0.79)> lift:(2.41) lev:(0.15) [6]  
conv:(2.36)
5. BHK=1 17 ==> Purchased=No 13 <conf:(0.76)> lift:(1.83) lev:(0.14) [5] conv:(1.98)
6. Purchased=No 18 ==> BHK=1 13 <conf:(0.72)> lift:(1.83) lev:(0.14) [5] conv:(1.81)
7. BHK=2 18 ==> Purchased=Yes 13 <conf:(0.72)> lift:(1.24) lev:(0.06) [2] conv:(1.26)
8. BHK=1 17 ==> Price=Low 12 <conf:(0.71)> lift:(1.79) lev:(0.12) [5] conv:(1.71)
9. Price=Low 17 ==> BHK=1 12 <conf:(0.71)> lift:(1.79) lev:(0.12) [5] conv:(1.71)
10. Price=Low 17 ==> Location=Suburbs 12 <conf:(0.71)> lift:(1.6) lev:(0.1) [4]  
conv:(1.58)

## Experiment 7:

**Aim:** Implement Clustering Algorithm (K-Means/K-Medoids)

### **Theory:**

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into K distinct, non-overlapping clusters. It is a centroid-based algorithm, meaning it assigns each data point to the cluster whose centroid (mean) is closest to it. K-means is widely used for various applications, including customer segmentation, image compression, and anomaly detection.

### **Key Concepts:**

#### **1. K:**

The number of clusters you want to create is a user-defined parameter. Selecting an appropriate value for K is often challenging and can impact the quality of clustering.

#### **2. Centroids:**

Each cluster is represented by a centroid, which is the mean (average) of all data points in the cluster. The centroid serves as the center of the cluster.

#### **3. Distance Metric:**

K-means uses a distance metric (usually Euclidean distance) to measure the similarity or dissimilarity between data points. It assigns data points to the cluster with the closest centroid based on this distance.

#### **4. Initialization:**

The algorithm starts by randomly initializing K centroids. The choice of initial centroids can affect the convergence and final clustering results. Common initialization methods include random initialization and K-means++.

#### **5. Assignment Step:**

In this step, each data point is assigned to the nearest centroid based on the chosen distance metric. This creates K clusters.

#### **6. Update Step:**

After data points have been assigned to clusters, the centroids are updated by calculating the mean of all data points in each cluster.

#### **7. Convergence:**

The assignment and update steps are repeated iteratively until convergence criteria are met. Common convergence criteria include a maximum number of iterations or when centroids no longer change significantly.

## **K-Means Algorithm:**

The K-means algorithm follows these steps:

1. Initialize K centroids randomly.
2. Repeat until convergence or a maximum number of iterations:
  - a. Assign each data point to the nearest centroid (assignment step).
  - b. Update the centroids by calculating the mean of data points in each cluster (update step).
3. End.

### **Example:**

You are given a dataset of houses with their respective prices and areas. Your goal is to use K-means clustering to group these houses into two clusters based on their price and area.

Data :

- Price Array: A list of house prices in thousands of dollars.  
price\_array = [150, 180, 200, 220, 250, 280, 300, 320, 350, 400]
- Area Array: A list of house areas in square meters.  
area\_array = [100, 120, 130, 150, 170, 180, 190, 200, 220, 240]

Steps to Solve:

#### 1. Initialization:

- Randomly initialise two centroids.
- For example, let's assume the initial centroids are:

Centroid 1: (150, 100)

Centroid 2: (320, 220)

#### 2. Iteration 1:

##### a. Assignment Step:

- Calculate the Euclidean distance between each house (price, area) and the centroids.
- Assign each house to the nearest centroid.

- Example assignments:

- House 1 (150, 100) is closer to Centroid 1.
- House 2 (180, 120) is closer to Centroid 1.
- House 3 (200, 130) is closer to Centroid 1.
- House 4 (220, 150) is closer to Centroid 1.
- House 5 (250, 170) is closer to Centroid 2.
- House 6 (280, 180) is closer to Centroid 2.
- House 7 (300, 190) is closer to Centroid 2.
- House 8 (320, 200) is closer to Centroid 2.
- House 9 (350, 220) is closer to Centroid 2.
- House 10 (400, 240) is closer to Centroid 2.

b. Update Step:

- Calculate the new centroids by taking the mean of houses in each cluster.

- New Centroid 1: Mean of Cluster 1 =  $((150+180+200+220)/4, (100+120+130+150)/4) = (187.5, 125)$

- New Centroid 2: Mean of Cluster 2 =  $((250+280+300+320+350+400)/6, (170+180+190+200+220+240)/6) = (315, 188.33)$

3. Iteration 2:

a. Assignment Step:

- Calculate the Euclidean distance between each house (price, area) and the centroids.

- Assign each house to the nearest centroid.

- Example assignments:

- House 1 (150, 100) is closer to New Centroid 1.
- House 2 (180, 120) is closer to New Centroid 1.
- House 3 (200, 130) is closer to New Centroid 1.
- House 4 (220, 150) is closer to New Centroid 1.
- House 5 (250, 170) is closer to New Centroid 2.
- House 6 (280, 180) is closer to New Centroid 2.
- House 7 (300, 190) is closer to New Centroid 2.
- House 8 (320, 200) is closer to New Centroid 2.
- House 9 (350, 220) is closer to New Centroid 2.
- House 10 (400, 240) is closer to New Centroid 2.

b. Update Step:

- Calculate the new centroids by taking the mean of houses in each cluster.

- New Centroid 1: Mean of Cluster 1 =  $((150+180+200+220)/4, (100+120+130+150)/4) = (187.5, 125)$

- New Centroid 2: Mean of Cluster 2 =  $((250+280+300+320+350+400)/6, (170+180+190+200+220+240)/6) = (315, 188.33)$

4. Convergence:

- Since the centroids did not change significantly between Iteration 1 and Iteration 2, the algorithm has converged.

5. Final Clusters:

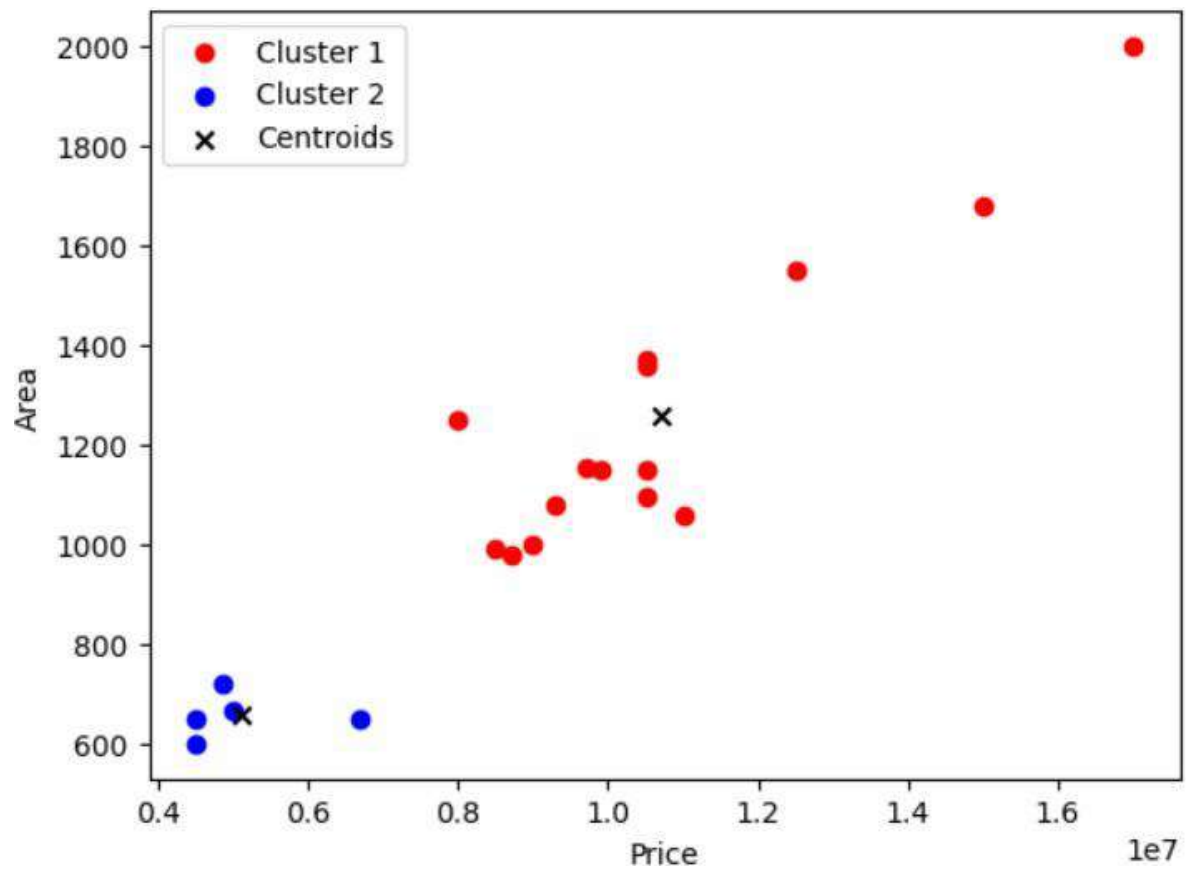
- Cluster 1: Houses with prices and areas similar to Centroid 1
- Cluster 2: Houses with prices and areas similar to Centroid 2

So the centroids are (315, 188.33) and (187.5,125) of respective clusters

## Code:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4
5 df=pd.read_csv('Mumbai.csv')
6
7 df=df.head(20)
8 np.random.seed(0)
9 prices = df['Price'].values
10 areas = df['Area'].values
11
12 data = np.vstack((prices, areas)).T
13
14 K = 2
15
16 centroids = data[np.random.choice(data.shape[0], K, replace=False)]
17
18 max_iters = 100
19
20 for _ in range(max_iters):
21     # Assign each data point to the nearest centroid
22     distances = np.sqrt(((data - centroids[:, np.newaxis])**2).sum(axis=2))
23     labels = np.argmin(distances, axis=0)
24
25     # Update the centroids based on the mean of points in each cluster
26     new_centroids = np.array([data[labels == k].mean(axis=0) for k in range(K)])
27
28     # Check for convergence
29     if np.all(centroids == new_centroids):
30         break
31
32     centroids = new_centroids
33
34 # Visualize the clusters
35 colors = ['r', 'b']
36 for k in range(K):
37     plt.scatter(data[labels == k][:, 0], data[labels == k][:, 1], c=colors[k], label=f'Cluster {k+1}')
38
39 plt.scatter(centroids[:, 0], centroids[:, 1], c='k', marker='x', label='Centroids')
40 plt.xlabel('Price')
41 plt.ylabel('Area')
42 plt.legend()
43 plt.show()
```

Output:





## Experiment 8:

**Aim:** Implementing any one of the Hierarchical Clustering Method

### **Theory :**

Hierarchical clustering is a versatile and widely used technique in data analysis and pattern recognition. Here's more detailed information on hierarchical clustering:

### **Hierarchical Clustering Process:**

#### **1. Initialization:**

Each data point is initially considered as a separate cluster. So, if you have  $N$  data points, you start with  $N$  clusters, each containing a single data point.

#### **2. Linkage Calculation:**

The algorithm calculates the pairwise distances (or dissimilarities) between all clusters or data points. The choice of linkage criterion determines how this distance is computed.

#### **3. Merging Clusters:**

The two clusters (or data points) with the smallest distance, according to the chosen linkage criterion, are merged into a single cluster. This step reduces the total number of clusters by one.

#### **4. Repeat :**

Steps 2 and 3 are repeated iteratively until only one cluster remains, containing all data points. At each step, the algorithm generates a dendrogram to visualize the hierarchy of cluster merges.

### **Linkage Criteria:**

Different linkage criteria result in different hierarchical structures. Here are some common linkage criteria:

#### **- Single Linkage:**

This method merges clusters based on the minimum distance between any two data points in the clusters. It tends to form elongated clusters and is sensitive to outliers.

#### **- Complete Linkage:**

It merges clusters based on the maximum distance between any two data points in the clusters. It tends to form compact, spherical clusters and is less sensitive to outliers.

### **- Average Linkage:**

This method calculates the average distance between all pairs of data points in the two clusters being merged. It strikes a balance between single and complete linkage.

### **- Ward Linkage:**

Ward's method minimises the variance of the distances between data points in the merged clusters. It often results in well-defined, balanced clusters.

### **Dendrogram:**

A dendrogram is a tree-like diagram that illustrates the hierarchical structure of clusters created during the clustering process. In a dendrogram:

- The vertical lines represent clusters or individual data points.
- The horizontal lines indicate the distance at which clusters or data points were merged.
- The height of each vertical line represents the level of dissimilarity at which the merge occurred.

### **Applications:**

Hierarchical clustering finds applications in various fields, including:

1. Biology: Clustering genes based on their expression patterns in microarray data.
2. Marketing: Segmenting customers based on their buying behavior.
3. Image Processing: Grouping similar regions in an image.
4. Text Analysis: Clustering documents based on their content.
5. Ecology: Classifying species based on similarities in ecological characteristics.

### **Pros and Cons:**

Pros:

- Does not require a predefined number of clusters.
- Provides a visual representation of hierarchical relationships.
- Sensitive to the underlying data structure.
- Can handle various distance metrics and linkage criteria.

Cons

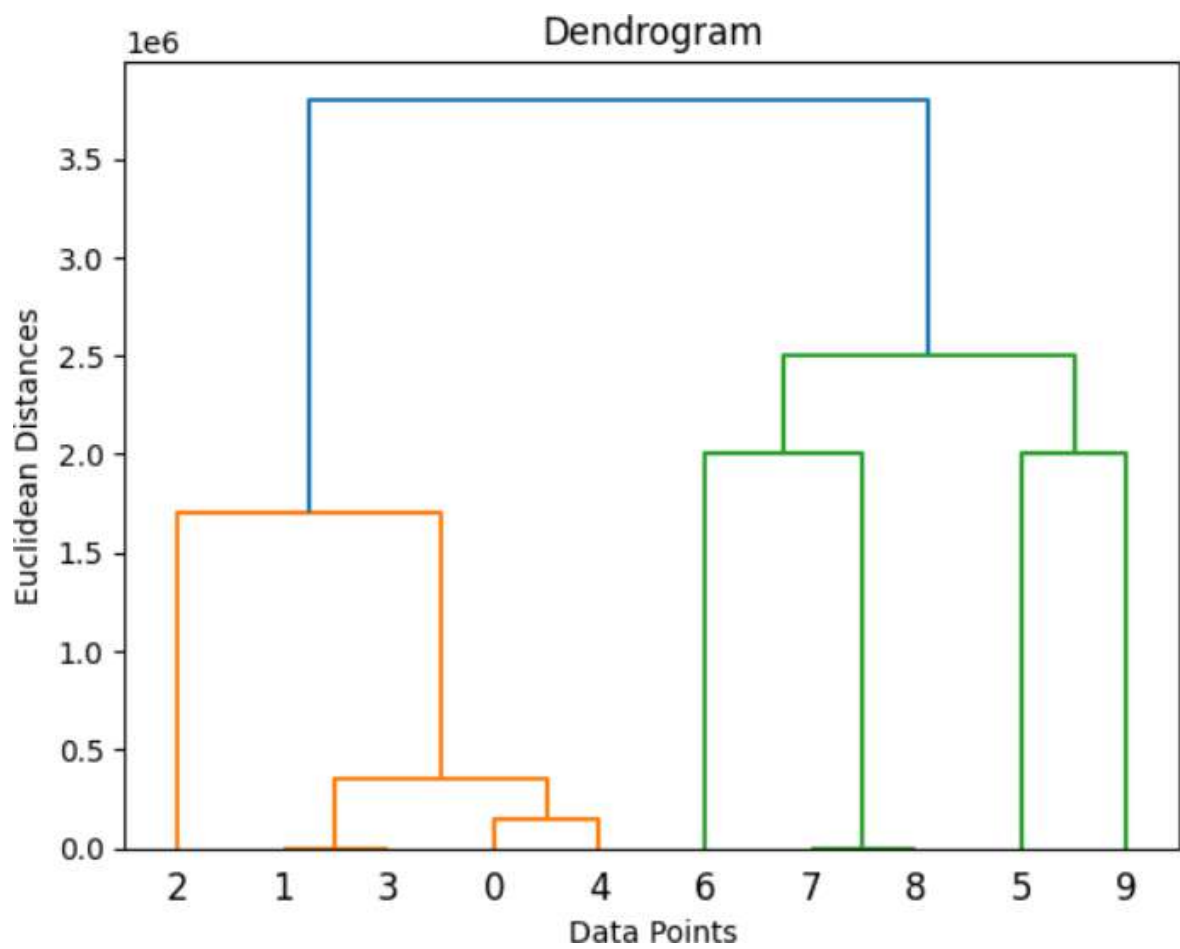
- Computationally expensive for large datasets.
- Dendrograms can be challenging to interpret for very large datasets.
- Prone to forming elongated or unbalanced clusters with single linkage.

In practice, hierarchical clustering is often used for exploratory data analysis and can guide subsequent clustering methods like k-means by helping to determine an appropriate number of clusters. It's essential to choose the linkage criterion and distance metric that best suit your data and problem.

## Code:

```
1 import pandas as pd
2 import scipy.cluster.hierarchy as sch
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # Load your house price dataset
7 data = pd.read_csv('./Mumbai.csv')
8
9 # Select the columns you want to use for clustering
10 columns_of_interest = ['Price', 'Area']
11
12 X = data[columns_of_interest].values
13
14 def euclidean_distance(point1, point2):
15     return np.sqrt(np.sum((point1 - point2) ** 2))
16
17 # Single linkage clustering
18 def single_linkage(X):
19     num_samples = X.shape[0]
20     linkage_matrix = np.zeros((num_samples - 1, 4))
21
22     clusters = [[i] for i in range(num_samples)]
23
24     for i in range(num_samples - 1):
25         min_distance = float('inf')
26         merge_candidates = (0, 0)
27
28         for cluster1_idx in range(len(clusters)):
29             for cluster2_idx in range(cluster1_idx + 1, len(clusters)):
30                 for sample1_idx in clusters[cluster1_idx]:
31                     for sample2_idx in clusters[cluster2_idx]:
32                         distance = euclidean_distance(X[sample1_idx], X[sample2_idx])
33                         if distance < min_distance:
34                             min_distance = distance
35                             merge_candidates = (cluster1_idx, cluster2_idx)
36
37         merged_cluster = clusters.pop(merge_candidates[1])
38         clusters[merge_candidates[0]].extend(merged_cluster)
39
40         linkage_matrix[i, 0] = merge_candidates[0]
41         linkage_matrix[i, 1] = merge_candidates[1]
42         linkage_matrix[i, 2] = min_distance
43         linkage_matrix[i, 3] = len(merged_cluster)
44
45     return linkage_matrix
46
47 linkage_matrix = single_linkage(X)
48
49 # Dendrogram
50 dendrogram = sch.dendrogram(linkage_matrix, labels=np.arange(X.shape[0]))
51 plt.title('Dendrogram')
52 plt.xlabel('Data Points')
53 plt.ylabel('Euclidean Distances')
54 plt.show()
55
```

**Output:**



## **EXPERIMENT :9**

AIM : Implementation of Association Rule mining algorithm(Apriori)

THEORY: Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. In other words, we can say that the apriori algorithm is an association rule learning that analyzes that people who bought product A also bought product B.

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions. Let's understand the apriori algorithm with the help of an example; suppose you go to Big Bazar and buy different products. It helps the customers buy their products with ease and increases the sales performance of the Big Bazar. In this tutorial, we will discuss the apriori algorithm with examples.

Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers buy at a Big Bazar.

Apriori algorithm helps the customers to buy their products with ease and increases the sales performance of the particular store.

Components of Apriori algorithm

The given three components comprise the apriori algorithm.

- Support
- Confidence

Let's take an example to understand this concept.

We have already discussed above; you need a huge database containing a large number of transactions. Suppose you have 4000 customers transactions in a Big Bazar. You have to 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

$$\text{Support (Biscuits)} = (\text{Transactions relating biscuits}) / (\text{Total transactions})$$

$$= 400/4000 = 10 \text{ percent.}$$

### **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,

$$\text{Confidence} = (\text{Transactions relating both biscuits and Chocolate}) / (\text{Total transactions involving Biscuits})$$

$$= 200/400$$

$$= 50 \text{ percent.}$$

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

### **Advantages of Apriori Algorithm**

- It is used to calculate large itemsets.
- Simple to understand and apply.

### **Disadvantages of Apriori Algorithms**

- 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:**

```
data = [  
    ['T100', ['Mira Road', 'Kharghar', 'Nerul']],  
    ['T200', ['Mira Road', 'Kharghar', 'Andheri', 'Nerul']],  
    ['T300', ['Mira Road', 'Andheri', 'Bandra']],  
    ['T400', ['Mira Road', 'Bandra', 'Kharghar']],  
    ['T500', ['Kharghar', 'Andheri']],  
    ['T600', ['Andheri', 'Bandra']],  
    ['T700', ['Andheri', 'Bandra', 'Mira Road']],  
    ['T800', ['Mira Road', 'Kharghar', 'Andheri', 'Bandra']],  
    ['T900', ['Mira Road', 'Andheri', 'Bandra', 'Kharghar']],  
    ['T1000', ['Mira Road', 'Andheri', 'Kharghar', 'Nerul']]  
]
```

```
init = []  
for i in data:  
    for q in i[1]:  
        if(q not in init):  
            init.append(q)  
init = sorted(init)  
print(init)
```

```
sp = 0.4  
s = int(sp*len(init))  
s
```

```
from collections import Counter
```

```
c = Counter()  
for i in init:  
    for d in data:  
        if(i in d[1]):  
            c[i]+=1  
print("C1:")  
for i in c:  
    print(str([i])+"": "+str(c[i]))  
print()  
# print(c)  
l = Counter()  
for i in c:  
    if(c[i] >= s):  
        l[frozenset([i])] += c[i]
```

```

print("L1:")
for i in l:
    print(str(list(i))+": "+str(l[i]))
print()
pl = l
pos = 1
for count in range (2,1000):
    nc = set()
    temp = list(l)
    for i in range(0,len(temp)):
        for j in range(i+1,len(temp)):
            t = temp[i].union(temp[j])
            if(len(t) == count):
                nc.add(temp[i].union(temp[j]))
    nc = list(nc)
    print(nc)
    c = Counter()
    for i in nc:
        c[i] = 0
        for q in data:
            temp = set(q[1])
            if(i.issubset(temp)):
                c[i]+=1
    print("C"+str(count)+":")
    for i in c:
        print(str(list(i))+": "+str(c[i]))
    print()
    l = Counter()
    for i in c:
        if(c[i] >= s):
            l[i]+=c[i]
    print("L"+str(count)+":")
    for i in l:
        print(str(list(i))+": "+str(l[i]))
    print()
    if(len(l) == 0):
        break
    pl = l
    pos = count
print("Result: ")
print("L"+str(pos)+":")
for i in pl:
    print(str(list(i))+": "+str(pl[i]))
print()

```



```

# print(pl)
confidence=60

from itertools import combinations
for l in pl:
    c = [frozenset(q) for q in combinations(l,len(l)-1)]
    mmax = 0
    for a in c:
        b = l-a
        ab = l
        sab = 0
        sa = 0
        sb = 0
        for q in data:
            temp = set(q[1])
            if(a.issubset(temp)):
                sa+=1
            if(b.issubset(temp)):
                sb+=1
            if(ab.issubset(temp)):
                sab+=1
        temp = sab/sa*100
        if(temp > confidence):
            mmax = temp
        temp = sab/sb*100
        if(temp > confidence):
            mmax = temp
        print(str(list(a))+" -> "+str(list(b))+" = "+str(sab/sa*100)+"%")
        print(str(list(b))+" -> "+str(list(a))+" = "+str(sab/sb*100)+"%")
    curr = 1
    print("choosing:", end=' ')
    for a in c:
        b = l-a
        ab = l
        sab = 0
        sa = 0
        sb = 0
        for q in data:
            temp = set(q[1])
            if(a.issubset(temp)):
                sa+=1
            if(b.issubset(temp)):
                sb+=1

```

```

        if(ab.issubset(temp)):
            sab+=1
        temp = sab/sa*100
        if(temp >= confidence):
            print(curr, end = ' ')
        curr += 1
        temp = sab/sb*100
        if(temp >= confidence):
            print(curr, end = ' ')
        curr += 1
    print()
    print()

```

### OUTPUT:

['Andheri', 'Bandra', 'Kharghar', 'Mira Road', 'Nerul']

C1:

['Andheri']: 8

['Bandra']: 6

['Kharghar']: 7

['Mira Road']: 8

['Nerul']: 3

L1:

['Andheri']: 8

['Bandra']: 6

['Kharghar']: 7

['Mira Road']: 8

['Nerul']: 3

C2:

['Kharghar', 'Bandra']: 3

['Mira Road', 'Andheri']: 6

['Andheri', 'Kharghar']: 5

['Mira Road', 'Bandra']: 5

['Nerul', 'Kharghar']: 3

['Mira Road', 'Nerul']: 3

['Mira Road', 'Kharghar']: 6

['Andheri', 'Bandra']: 5

['Nerul', 'Bandra']: 0

['Nerul', 'Andheri']: 2

L2:

['Kharghar', 'Bandra']: 3  
['Mira Road', 'Andheri']: 6  
['Andheri', 'Kharghar']: 5  
['Mira Road', 'Bandra']: 5  
['Nerul', 'Kharghar']: 3  
['Mira Road', 'Nerul']: 3  
['Mira Road', 'Kharghar']: 6  
['Andheri', 'Bandra']: 5  
['Nerul', 'Andheri']: 2

C3:

['Andheri', 'Kharghar', 'Bandra']: 2  
['Mira Road', 'Andheri', 'Bandra']: 4  
['Mira Road', 'Andheri', 'Kharghar']: 4  
['Mira Road', 'Nerul', 'Andheri']: 2  
['Nerul', 'Kharghar', 'Bandra']: 0  
['Nerul', 'Andheri', 'Kharghar']: 2  
['Mira Road', 'Nerul', 'Bandra']: 0  
['Mira Road', 'Kharghar', 'Bandra']: 3  
['Nerul', 'Andheri', 'Bandra']: 0  
['Mira Road', 'Nerul', 'Kharghar']: 3

L3:

['Andheri', 'Kharghar', 'Bandra']: 2  
['Mira Road', 'Andheri', 'Bandra']: 4  
['Mira Road', 'Andheri', 'Kharghar']: 4  
['Mira Road', 'Nerul', 'Andheri']: 2  
['Nerul', 'Andheri', 'Kharghar']: 2  
['Mira Road', 'Kharghar', 'Bandra']: 3  
['Mira Road', 'Nerul', 'Kharghar']: 3

C4:

['Nerul', 'Kharghar', 'Bandra', 'Andheri']: 0  
['Mira Road', 'Nerul', 'Bandra', 'Andheri']: 0  
['Mira Road', 'Kharghar', 'Bandra', 'Andheri']: 2  
['Mira Road', 'Nerul', 'Kharghar', 'Bandra']: 0  
['Mira Road', 'Nerul', 'Kharghar', 'Andheri']: 2

L4:

['Mira Road', 'Kharghar', 'Bandra', 'Andheri']: 2  
['Mira Road', 'Nerul', 'Kharghar', 'Andheri']: 2

C5:

['Bandra', 'Andheri', 'Mira Road', 'Nerul', 'Kharghar']: 0

L5:

Result:

L4:

['Mira Road', 'Kharghar', 'Bandra', 'Andheri']: 2

['Mira Road', 'Nerul', 'Kharghar', 'Andheri']: 2

['Mira Road', 'Kharghar', 'Bandra'] -> ['Andheri'] = 66.66666666666666%

['Andheri'] -> ['Mira Road', 'Kharghar', 'Bandra'] = 25.0%

['Mira Road', 'Kharghar', 'Andheri'] -> ['Bandra'] = 50.0%

['Bandra'] -> ['Mira Road', 'Kharghar', 'Andheri'] = 33.33333333333333%

['Mira Road', 'Andheri', 'Bandra'] -> ['Kharghar'] = 50.0%

['Kharghar'] -> ['Mira Road', 'Andheri', 'Bandra'] = 28.57142857142857%

['Andheri', 'Kharghar', 'Bandra'] -> ['Mira Road'] = 100.0%

['Mira Road'] -> ['Andheri', 'Kharghar', 'Bandra'] = 25.0%

choosing: 1 7

['Mira Road', 'Nerul', 'Kharghar'] -> ['Andheri'] = 66.66666666666666%

['Andheri'] -> ['Mira Road', 'Nerul', 'Kharghar'] = 25.0%

['Mira Road', 'Nerul', 'Andheri'] -> ['Kharghar'] = 100.0%

['Kharghar'] -> ['Mira Road', 'Nerul', 'Andheri'] = 28.57142857142857%

['Mira Road', 'Kharghar', 'Andheri'] -> ['Nerul'] = 50.0%

['Nerul'] -> ['Mira Road', 'Kharghar', 'Andheri'] = 66.66666666666666%

['Nerul', 'Kharghar', 'Andheri'] -> ['Mira Road'] = 100.0%

['Mira Road'] -> ['Nerul', 'Kharghar', 'Andheri'] = 25.0%

choosing: 1 3 6 7

## Experiment 10

Aim: implementation page rank/hits algorithm

Theory : Detail about the PageRank algorithm:

1. Link Analysis and Importance: PageRank is rooted in the concept that web pages are connected by hyperlinks, and the importance of a page can be determined by the number and quality of links it receives from other pages. In essence, it's a measure of a page's popularity and authority.

2. Random Surfer Model: Imagine a random web user who starts on a random web page and either clicks on links or jumps to other pages. PageRank models this as a Markov chain, where the probability that the surfer is on a particular page at any given time represents the PageRank of that page.

3. The PageRank Formula: The PageRank of a page is calculated using a recursive mathematical formula. It's based on the principle that a page's importance is shared among the pages it links to. The formula can be expressed as:

$$\text{PageRank}(A) = (1 - d) + d * (\text{PageRank}(B) / L(B) + \text{PageRank}(C) / L(C) + \dots)$$

Where:

- PageRank(A) is the PageRank of page A.
- d is the damping factor (typically set to 0.85).
- PageRank(B), PageRank(C), etc., are the PageRank values of pages linking to page A.
- L(B), L(C), etc., are the number of outbound links on pages B, C, etc.

4. Damping Factor: The damping factor (d) represents the probability that the random surfer will click on a link. It introduces a level of randomness to the model, ensuring that not all links are followed, and the surfer might occasionally jump to a random page.

5. Iterative Calculation: PageRank is not calculated in a single pass. Instead, it's an iterative process. The PageRank values are initially assumed to be equal for all pages, and then the formula is applied repeatedly until the values converge to a stable point. The process continues until the differences between successive iterations are very small.

6. Link Weighting: Not all links are considered equal. Inbound links from pages with higher PageRank contribute more to a page's PageRank. This means that links from authoritative and trusted sources carry more weight and influence.

7. Importance of Internal Links: PageRank isn't limited to external backlinks. It also applies to internal links within a website. Internal links can help distribute PageRank among different pages of a site, optimizing the flow of importance within the site's structure.

In practice, PageRank is just one of many factors used by search engines to rank web pages. Search engines, including Google, have since developed more sophisticated algorithms that take into account a wide range of factors, including content quality, user behavior, and more. However, the fundamental principles of PageRank continue to be important in understanding the concept of web authority and link analysis.

Code:

```
# n = int(input("Enter the total number of nodes in the graph:\n"))
n = 4
adj_mat = [[0, 1, 1, 1],
[0, 0, 0, 1],
[1, 0, 0, 1],
[1, 0, 1, 0]]
# adj_mat = []

# print("Enter the Adjacency Matrix of the Graph\n")
# for i in range(n):
#     row = []
#     for j in range(n):
#         val = int(input("Enter the value at row {} and column {} in the adjacency Matrix:
#.format(i+1,j+1)))
#         row.append(val)
#     adj_mat.append(row)

print("Entered Adjacency Matrix is\n")
for row in adj_mat:
    print(row)
```

```

# Calculate out-degree of each node
out_degree = []
for i in range(n):
    out_count = 0
    for j in range(n):
        if adj_mat[i][j] == 1:
            out_count += 1
    out_degree.append(out_count)

print("\nTotal Outgoing Links from Each Node:\n")
for i in range(n):
    print("Node {}: {}".format(i+1, out_degree[i]))

A = []
print(out_degree)
for i in range(n):
    A_row = []
    for j in range(n):
        if adj_mat[j][i] == 1:
            # print(f"deb: {i}, {j}")
            A_row.append(1/out_degree[j])
        else:
            A_row.append(0)
    A.append(A_row)

print("\nTransition Probability Matrix A:\n")
for row in A:
    print(row)
import numpy as np
X = np.ones((n, 1))
# print(X)
A = np.array(A)
prev = np.array([])
print("Final matrix after 3 iterations\n")
for _ in range(0, 3):
    X = A @ X
    prev = X
print(X)

```

```

ans = dict()
for _ in range(0, X.size):
    ans[f'Page {_+1}'] = X[_][0]
ans = dict(sorted(ans.items(), key=lambda item: -item[1]))
# print(ans)

```

```

ctn = 1
for i in ans:
    print(f"Rank {ctn}: {i}")
    ctn+=1

```

Output:

Entered Adjacency Matrix is

```

[0, 1, 1, 1]
[0, 0, 0, 1]
[1, 0, 0, 1]
[1, 0, 1, 0]

```

Total Outgoing Links from Each Node:

```

Node 1: 3
Node 2: 1
Node 3: 2
Node 4: 2
[3, 1, 2, 2]

```

Transition Probability Matrix A:

```

[0, 0, 0.5, 0.5]
[0.3333333333333333, 0, 0, 0]
[0.3333333333333333, 0, 0, 0.5]
[0.3333333333333333, 1.0, 0.5, 0]
Final matrix after 3 iterations

```

```

[[1.16666667]
 [0.44444444]
 [0.98611111]
 [1.40277778]]

```



Rank 1: Page 4

Rank 2: Page 1

Rank 3: Page 3

Rank 4: Page 2

## Written Assignment - 01

Q1 What is metadata? Explain data-warehouse Metadata with example.

→ Data warehouse metadata are pieces of information stored in one or more special-purpose metadata repositories that include -

- Information on contents of data warehouse, their location and their structure
- Information on the processes that take place in the data warehouse back-stage, concerning the refreshment of warehouse with clean, up-to-date, semantically and structurally reconciled data
- Information on implicit semantics of data along with any other kind of data that aids the user to exploit warehouse information
- Information on infrastructure and physical characteristics of components and sources of data warehouse
- Information including security, authentication, and usage statistics that aid administrator

Data Warehouse Metadata has specific requirements. Metadata that describes tables typically includes:

- Physical name
- Logical name
- Type : Fact, Dimension, Bridge
- Role : Legacy, OLTP, Stage
- DBMS : DB2, Informix, MS SQL Server, Oracle, Sybase

- Location
- Definition
- Notes

Meta data describes columns within tables

Physical Name	Logical Name
Order in Table	Data type
Length	Decimal positions
Nullable/Required	Default value
Edit Rules	Definition
Notes	

Example of Metadata for Customer sales data warehouse

- Entity name : Customer
- Create Date : June 25, 2010
- Last Update Date : June 01, 2015
- Update Cycle : Weekly
- Remarks : Entity includes regular, current and past Customers
- Data Quality Review : July 05, 2015
- Responsible User : Pallavi K.

AP

18/10



## TE-C31 Written Assignment-02

Q1] Write short notes on

a) Multilevel Association rules and Multidimensional Association rules

→ Association rules created from mining information at different degrees of reflection are called various level or staggered association rules. These can be mined effectively utilizing idea progressions under a help certainty system. Items are always in form of hierarchy and the leaf node items have lower support. Item can be generalized or specialized as per hierarchy and its level can be present in transactions.

The support of generalized item is more than the support of specialized item. Similarly the support of rules increases from specialized to generalized item sets. Confidence is not affected for general or specialized.

Approaches of Multilevel association rule.

- 1) Using uniform minimum support for all levels.
  - Consider the same minimum support for all levels of hierarchy.
  - As only one minimum support is set, so there is no necessity to examine the items of

Itemset whose ancestors do not have minimum support.

- If very high support is considered then many low level association may get missed.
- If very low support is considered then many high level association rules are generated.

2) Using reduced minimum support at lower level

- Consider separate minimum support at each level of hierarchy.

• As every level is having its own minimum support, support at lower level reduces

- There are 4 search strategies-

a) Level-by-level independent

It's a full-breadth search method and to examine a node, its parent node is checked whether it's frequent or not.

b) Level-cross filtering by single item

• Children of only frequent nodes are checked.

c) Level-cross filtering by k-itemset

Find frequent k-itemset at parent level and only k-itemset at next level is checked.

d) Controlled level-cross filtering by single item

It is the modified version of level-cross filtering by single item. Some minimum support threshold is set for lower level and called 'Level Passage Threshold' and items that don't satisfy minimum support are checked for the threshold.



## Mining Multidimensional Association Rules

i) Single-dimensional rules -

contains only one distinct predicate

Eg:  $\text{buys}(x, \text{"Butter"}) \Rightarrow \text{buys}(x, \text{"Milk"})$

ii) Multi-dimensional rules -

contains two or more predicates. Has two types

• Inter-dimension-rules doesn't have any repeated predicate. Eg:

$\text{gender}(x, \text{"Male"}) \wedge \text{salary}(x, \text{"High"}) \Rightarrow \text{buys}(x, \text{"Computer"})$

• Hybrid - rule may have occurrence of same predicate  
 $\text{gender}(x, \text{"Male"}) \wedge \text{buys}(x, \text{"TV"}) \Rightarrow \text{buys}(x, \text{"DVD"})$

iii) Categorical attributes -

Have finite number of possible values and there is no ordering among values.

Eg - brand, color

iv) Quantitative attributes -

Numeric values and implicit ordering

Eg - age, income

## Techniques for mining multidimensional associations

i) Using static discretization of quantitative attributes -

Use concept hierarchy to discretize quantitative attributes.

Convert numeric values by ranges or categorical values. To get all frequent  $k$ -predicate,  $k$  or  $k+1$  table scans are needed to generate a cube so that mining is faster. Thus, predicate sets are determined by cells of  $n$ -dimensional cuboid.

## 2) Quantitative Association rules

Numeric attributes are dynamically discretized such that confidence or compactness of rules mined is maximized. The LHS contains 2-D quantitative attribute and RHS has categorical attribute

$$A_{\text{quant.1}} \wedge A_{\text{quant.2}} \Rightarrow \text{Categorical}$$

Approach to find such rules is ARCS or Association Rule Clustering Systems that involves binning, finding frequent predicate set and clustering association rules.

## 3) Distance Based Association rules

It is a dynamic discretization process that considers the distance between data points and has two steps -

- Perform clustering to find interval of attributes involved
- Obtain association rules by searching for cluster groups that occur together.



## b) Web Usage Mining

- It is the type of web mining activity that predicts about which pages are likely to be visited in near future based on the active user's behavior. These pages can be pre-fetched to reduce access times.
- To better understand and serve needs of users, usage data record user's behaviour as they browse or make transactions. A large amount of data collected by organizations is automatically generated by Web servers and collected in server log.
  - Initial web analysis tools simply reported user activity as recorded in servers and no analysis. The data is that is now collected in a web server log (file created by server to record all its activities). For example when user enters a URL, when a page request is sent to web server, information about URL, user's IP address, status of request, time and date of request completion and referrer header are recorded.
  - This web usage data is used in
    - i) personalization, collaboration in Web-based systems
    - ii) Marketing
    - iii) Web site design and evaluation
    - iv) Decision support

AP  
18/10