

# Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

## ❧ CERTIFICATE ❧

Certify that Mr./Miss Shirish Shetty  
of Comps Department, Semester V with  
Roll No. 2103164 has completed a course of the necessary  
experiments in the subject Data warehousing & Mining under my  
supervision in the **Thadomal Shahani Engineering College**  
Laboratory in the year 2023 - 2024

Teacher In- Charge

  
Head of the Department

Date 23rd Oct 2023

Principal

## CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1)	Select a dataset and perform exploratory data analysis using Python (data preprocessing, transformation, discretization & visualization)		2/8/23	
2)	One case study on building Datawarehouse/ Data Mart. Write Detailed Problem Statement and design dimensional modelling (creation of star and snowflake schema)		31/6/23	
3)	Implementation of all dimension table & fact table based on experiment 1 case study		10/8/23	
4)	Implementation of OLAP operations: Slice, Dice, Rollup, DrillDown and Pivot based on experiment 1		24/8/23	
5)	Implementation of Bayesian algorithm		31/8/23	
6)	Perform data Pre-processing task and Demonstrate performing Classification, Clustering, Association algorithm on data set using mining tool (WEKA/ R tool)		5/9/23	
7)	Implementation of Clustering algorithm (K-means/K-medoids)		14/9/23	
8)	Implementation of any one Hierarchical Clustering method.		21/9/23	
9)	Implementation of Association Rule Mining Algorithm (Apriori)		9/10/23	
10)	Implementation of Page rank/HITS algorithm.		19/10/23	
	Assignment - 1		21/10/23	
	Assignment - 2		21/10/23	

## EXPERIMENT NO. 1

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

### THEORY :

#### Data Cleaning -

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset.

#### Data Transformation -

Data transformation is the process of converting data from one format, such as a database file, XML document or Excel spreadsheet, into another. Transformations typically involve converting a raw data source into a cleansed, validated and ready-to-use format. Data transformation is crucial to data management processes that include data integration, data migration, data warehousing and data preparation. The process of data transformation can also be referred to as extract/transform/load (ETL).

#### Data Visualization:

Data visualization is the graphical representation of information and data. It involves using visual elements like charts, graphs, and maps to present complex data in a more accessible and understandable format. Data visualization is a powerful tool for exploring, analyzing, and communicating insights from data, making it an essential part of data analysis.

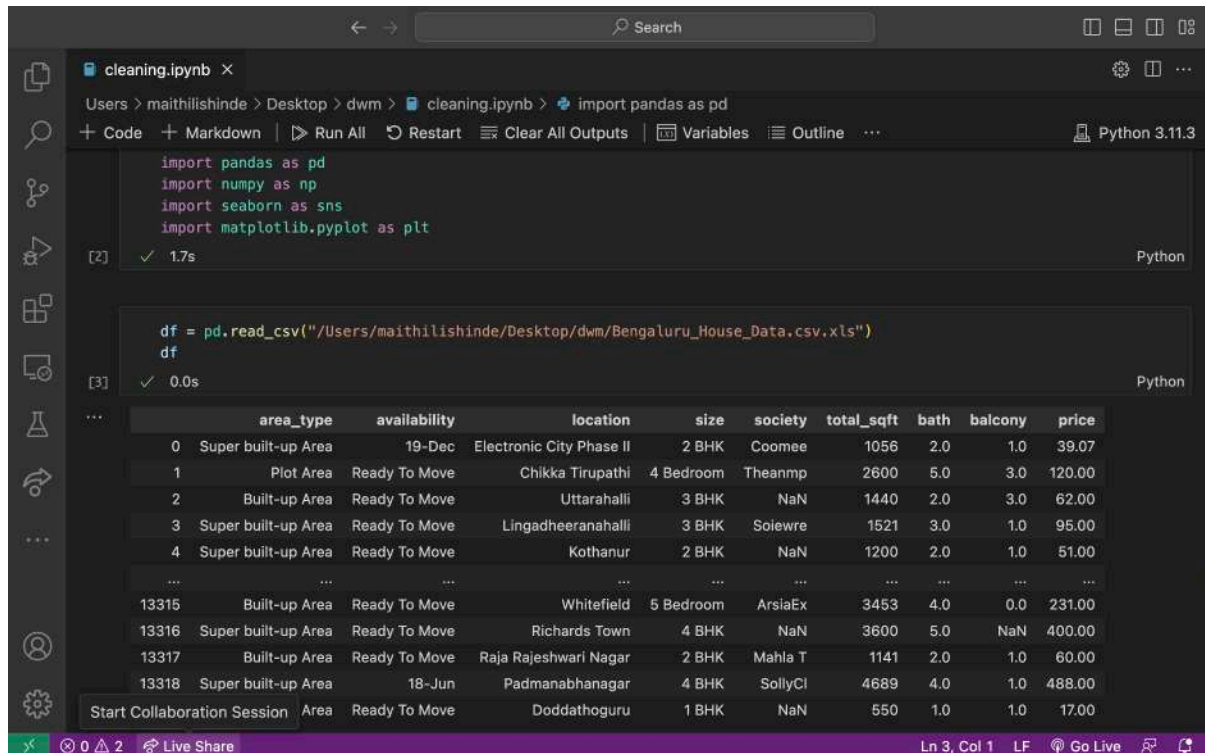
#### **Data Analysis:**

Data analysis is the process of inspecting, cleaning, transforming, and interpreting data to extract meaningful insights, discover patterns, and make informed decisions. It involves using various techniques, tools, and methodologies to understand the underlying structure of data, identify trends, relationships, and anomalies, and derive actionable information from the data.



## 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



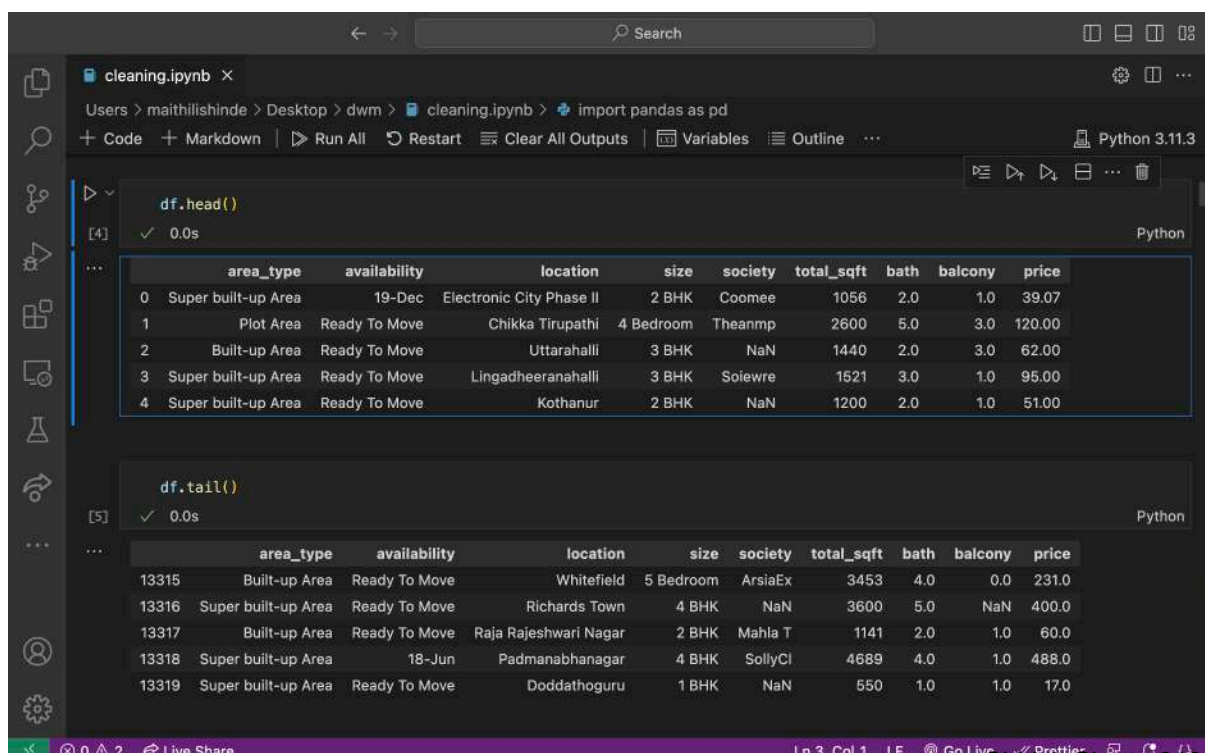
The screenshot shows a Jupyter Notebook with two code cells. The first cell imports pandas, numpy, seaborn, and matplotlib. The second cell reads a CSV file into a DataFrame. The output of the second cell is a preview of the DataFrame.

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

```
df = pd.read_csv("/Users/maithilishinde/Desktop/dwm/Bengaluru_House_Data.csv.xls")
df
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Solewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00
...	...	...	...	...	...	...	...	...	...
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.00
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.00
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.00
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCI	4689	4.0	1.0	488.00
...	...	...	...	...	...	...	...	...	...
13319	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	NaN	550	1.0	1.0	17.00

3. Describe the attributes name, count no of values, and find min, max, data type, range, quartile, percentile.



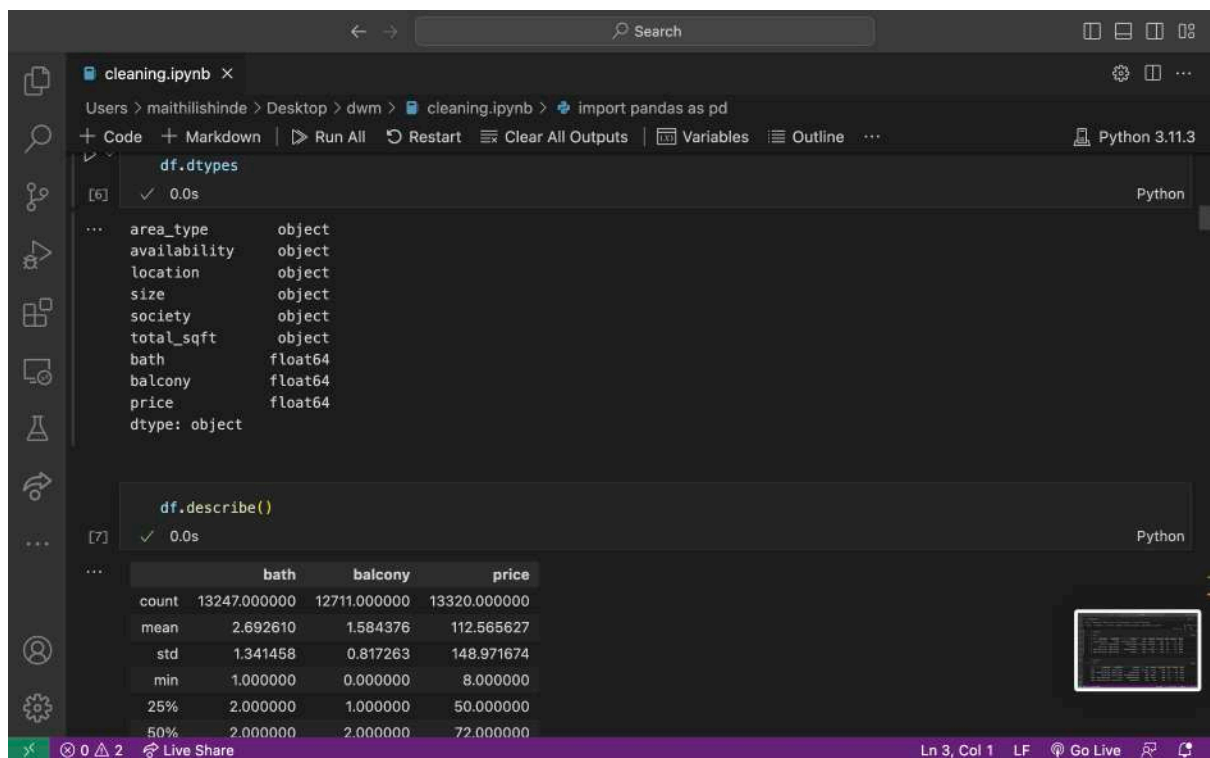
The screenshot shows a Jupyter Notebook with two code cells. The first cell uses head() to view the first five rows of the DataFrame. The second cell uses tail() to view the last five rows of the DataFrame.

```
df.head()
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Solewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00

```
df.tail()
```

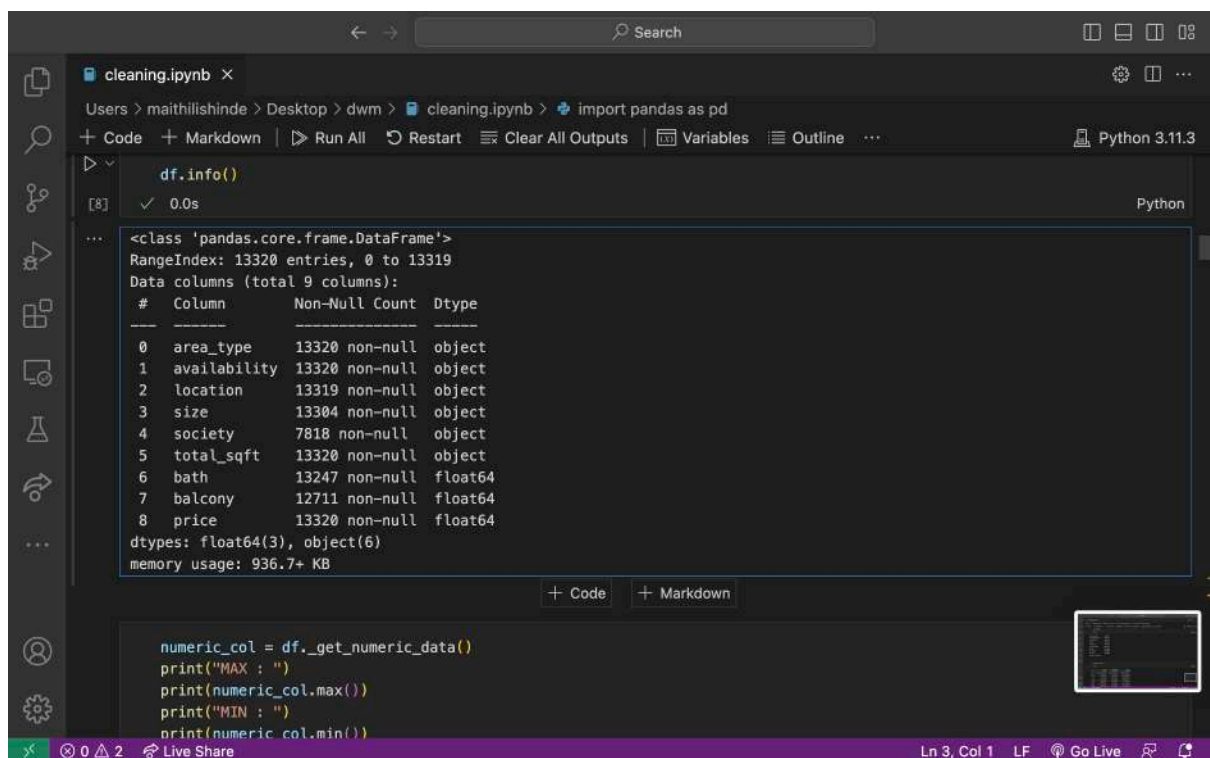
	area_type	availability	location	size	society	total_sqft	bath	balcony	price
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.00
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.00
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.00
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCI	4689	4.0	1.0	488.00
13319	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	NaN	550	1.0	1.0	17.00



The screenshot shows a Jupyter Notebook interface with a file named 'cleaning.ipynb'. The notebook is open to a cell containing the code `df.dtypes`. The output of this code is displayed below the cell, showing the data types for each column in the DataFrame. The columns and their corresponding dtypes are:

Column	Dtype
area_type	object
availability	object
location	object
size	object
society	object
total_sqft	object
bath	float64
balcony	float64
price	float64
dtype:	object

The notebook interface also shows a search bar at the top, a sidebar with icons for file operations, and a status bar at the bottom indicating the current line and column (Ln 3, Col 1).



The screenshot shows a Jupyter Notebook interface with a file named 'cleaning.ipynb'. The notebook is open to a cell containing the code `df.info()`. The output of this code is displayed below the cell, providing a summary of the DataFrame's structure and content. The output includes the class name, range index, total number of columns, and a detailed breakdown of each column's data type and non-null count.

#	Column	Non-Null Count	Dtype
0	area_type	13320 non-null	object
1	availability	13320 non-null	object
2	location	13319 non-null	object
3	size	13304 non-null	object
4	society	7818 non-null	object
5	total_sqft	13320 non-null	object
6	bath	13247 non-null	float64
7	balcony	12711 non-null	float64
8	price	13320 non-null	float64

The output also includes the dtypes: float64(3), object(6) and the memory usage: 936.7+ KB. The notebook interface also shows a search bar at the top, a sidebar with icons for file operations, and a status bar at the bottom indicating the current line and column (Ln 3, Col 1).

cleaning.ipynb ×

Users > maithilishinde > Desktop > dwm > cleaning.ipynb > import pandas as pd

+ Code + Markdown | ▶ Run All ⏮ Restart ⏭ Clear All Outputs | 📄 Variables 📄 Outline ... Python 3.11.3

```
numeric_col = df._get_numeric_data()
print("MAX : ")
print(numeric_col.max())
print("MIN : ")
print(numeric_col.min())
```

[9] ✓ 0.0s Python

MAX :

bath	40.0
balcony	3.0
price	3600.0

dtype: float64

MIN :

bath	1.0
balcony	0.0
price	8.0

dtype: float64

+ Code + Markdown

```
print("RANGE : ")
(numeric_col.max() - numeric_col.min())
```

[22] ✓ 0.0s Python

RANGE :

0 0 2 Live Share Ln 3, Col 1 LF Go Live

cleaning.ipynb ×

Users > maithilishinde > Desktop > dwm > cleaning.ipynb > import pandas as pd

+ Code + Markdown | ▶ Run All ⏮ Restart ⏭ Clear All Outputs | 📄 Variables 📄 Outline ... Python 3.11.3

bath 39.0  
balcony 3.0  
price 3592.0  
dtype: float64

```
print("QUARTILE : ")
print(numeric_col.quantile([0.25,0.50,0.75,1.00]))
```

[23] ✓ 0.0s Python

QUARTILE :

	bath	balcony	price
0.25	2.0	1.0	50.0
0.50	2.0	2.0	72.0
0.75	3.0	2.0	120.0
1.00	40.0	3.0	3600.0

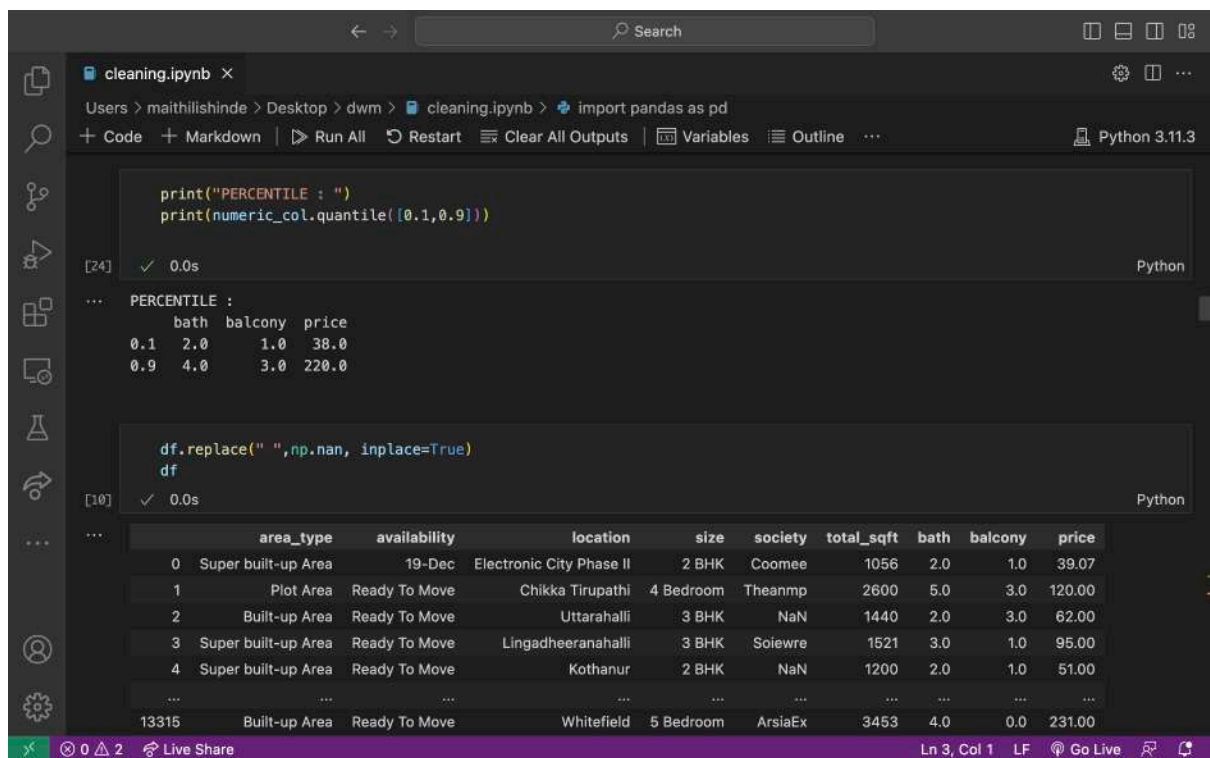
+ Code + Markdown

```
print("PERCENTILE : ")
print(numeric_col.quantile([0.1,0.9]))
```

[24] ✓ 0.0s Python

0 0 2 Live Share Ln 3, Col 1 LF Go Live

#### 4. Perform cleaning ,transformation , discretization and analysis



The screenshot shows a Jupyter Notebook with two code cells. The first cell calculates percentiles for 'bath', 'balcony', and 'price' columns. The second cell replaces empty strings with NaN in the DataFrame. The output of the first cell shows the percentiles, and the output of the second cell shows the first few rows of the DataFrame.

```
print("PERCENTILE : ")
print(numeric_col.quantile([0.1,0.9]))
```

PERCENTILE :

	bath	balcony	price
0.1	2.0	1.0	38.0
0.9	4.0	3.0	220.0

```
df.replace(" ",np.nan, inplace=True)
df
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00
...	...	...	...	...	...	...	...	...	...
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.00

```
cleaning.ipynb x
Users > maithilishinde > Desktop > dwm > cleaning.ipynb > import pandas as pd
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ... Python 3.11.3

df['size'].value_counts().idxmax
df['size'].replace(np.nan, '2 BHK', inplace=True)
df['size']

[16] ✓ 0.0s Python

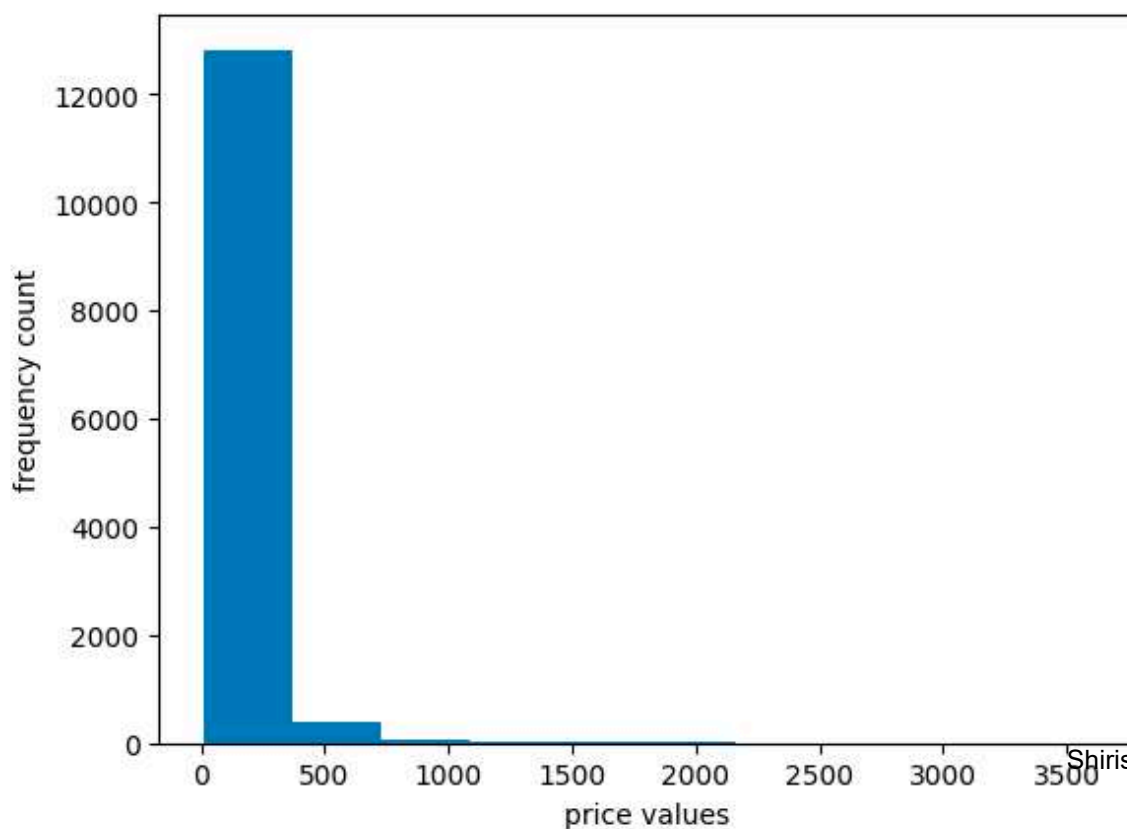
... 0 2 BHK
1 4 Bedroom
2 3 BHK
3 3 BHK
4 2 BHK
...
13315 5 Bedroom
13316 4 BHK
13317 2 BHK
13318 4 BHK
13319 1 BHK
Name: size, Length: 13320, dtype: object

df['location'].value_counts().idxmax
df['location'].replace(np.nan, 'Whitefield', inplace=True)
df['location']

[17] ✓ 0.0s Python
13320 rows x 8 columns
```

### Performing binning on price column:

```
df['price']=df['price'].astype('float')
plt.hist(df['price'])
plt.xlabel("price values")
plt.ylabel("frequency count")
```





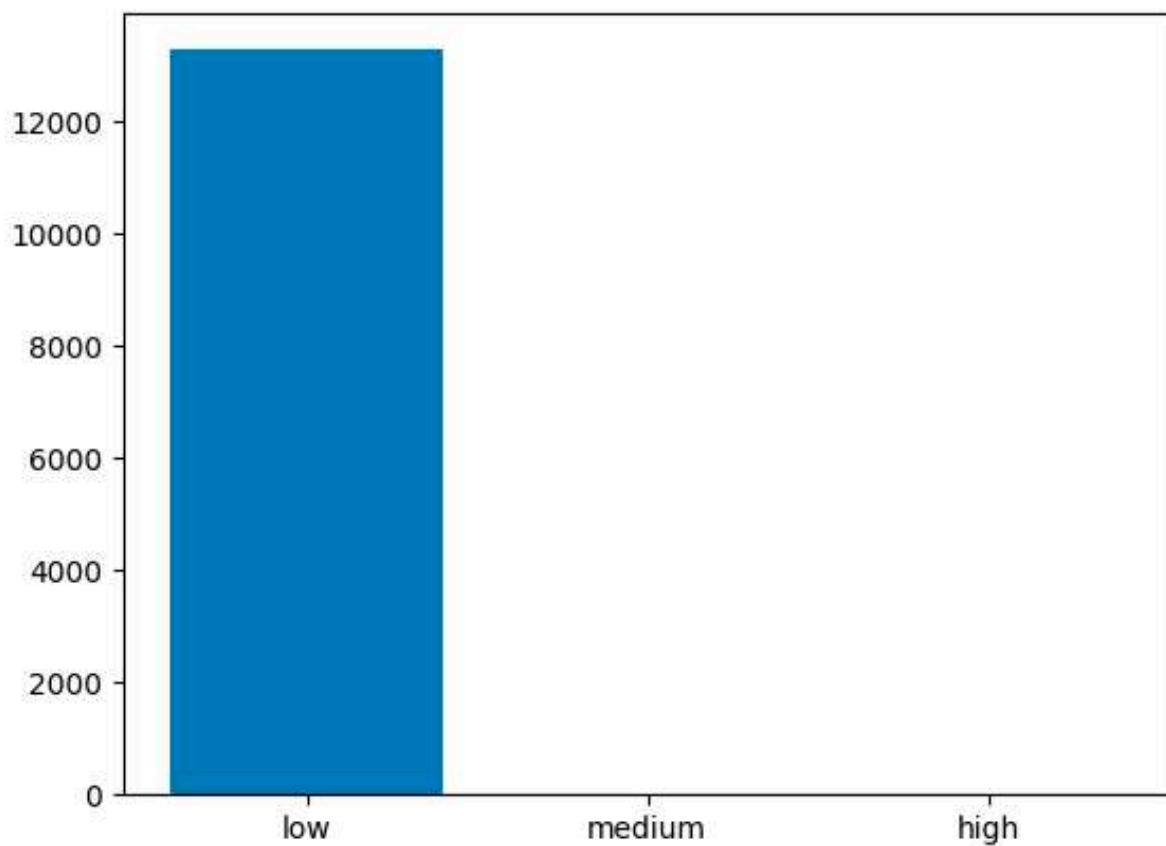
```
bins=np.linspace(min(df['price']),max(df['price']),4)
```

```
array([ 8.          , 1205.33333333, 2402.66666667, 3600.
])
```

```
df['price-binned'].value_counts()
```

```
price-binned
low          13284
medium         29
high           7
Name: count, dtype: int64
```

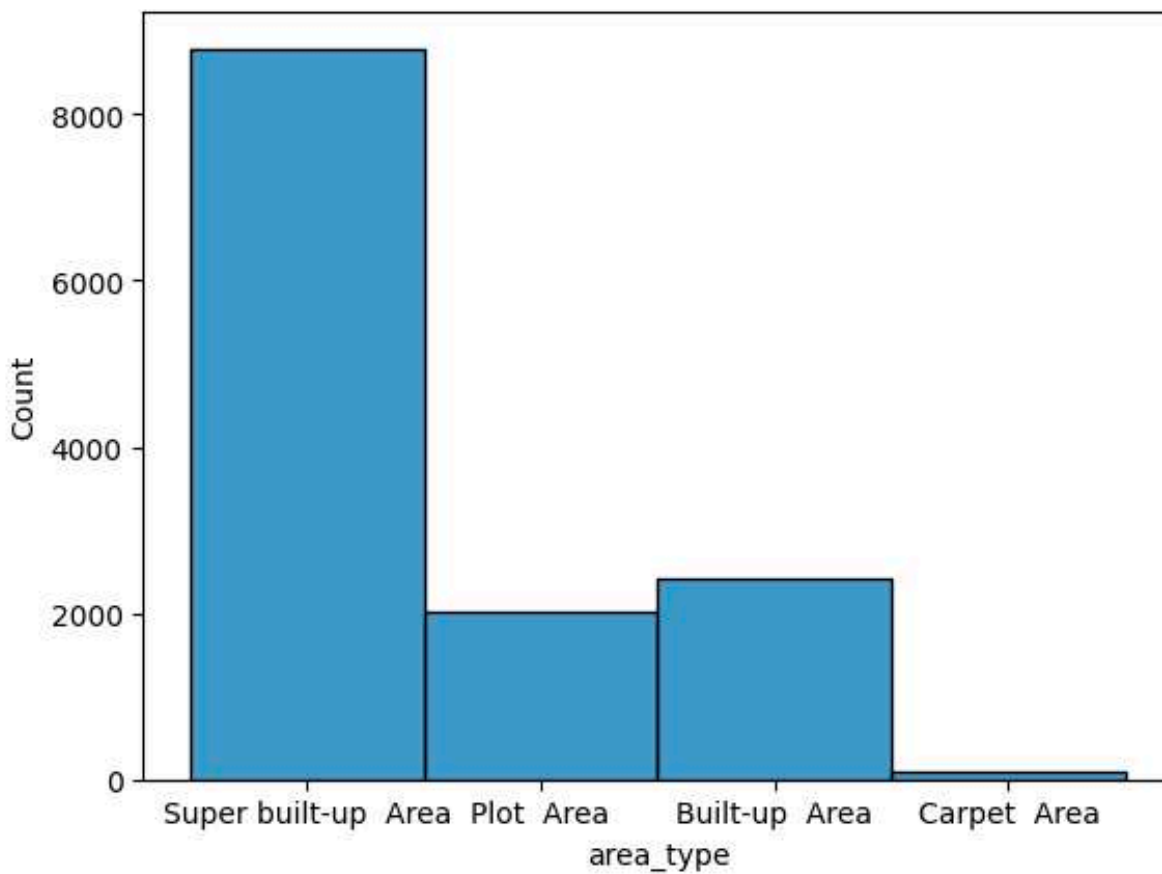
```
plt.bar(groups_name,df['price-binned'].value_counts())
```



5. Give visualization of statistical description of data – in form of histogram, scatter plot, pie chart, Give correlation matrix

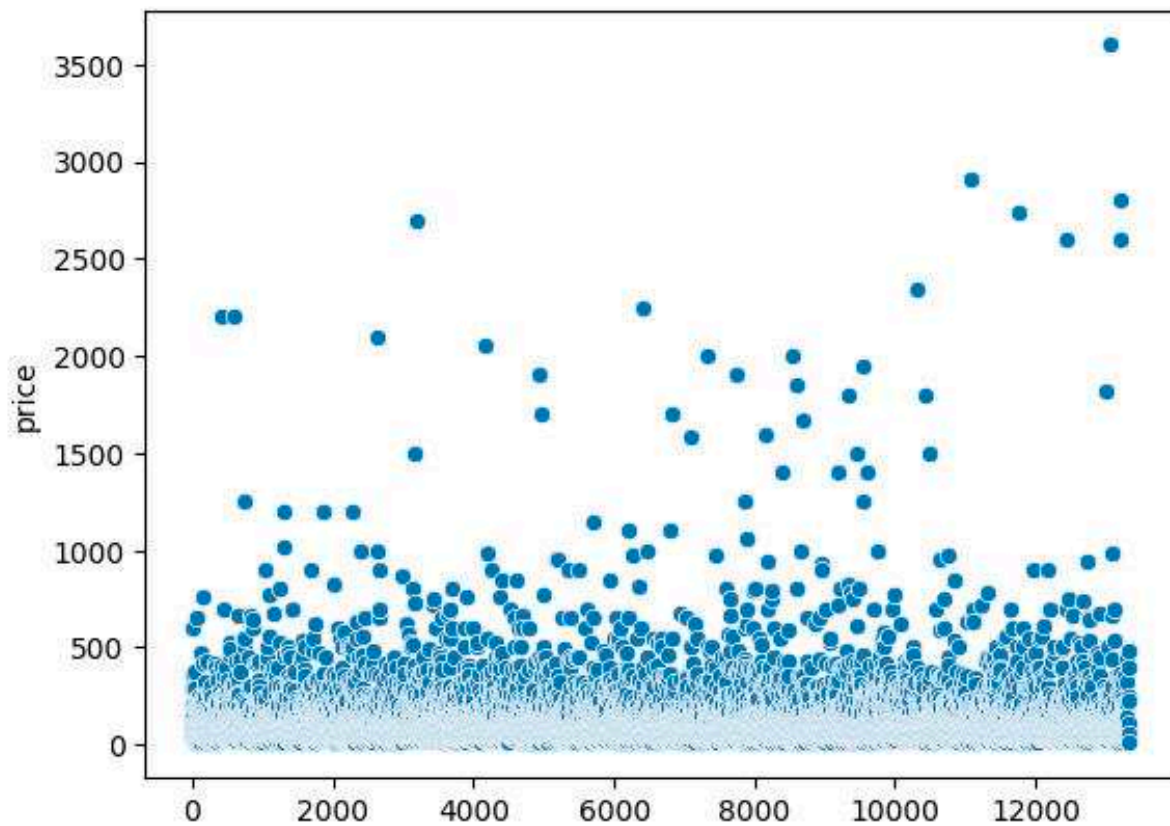
**HISTOGRAM :**

```
sns.histplot(df['area_type'])
```



**SCATTER PLOT :**

```
sns.scatterplot(df['price'])
```

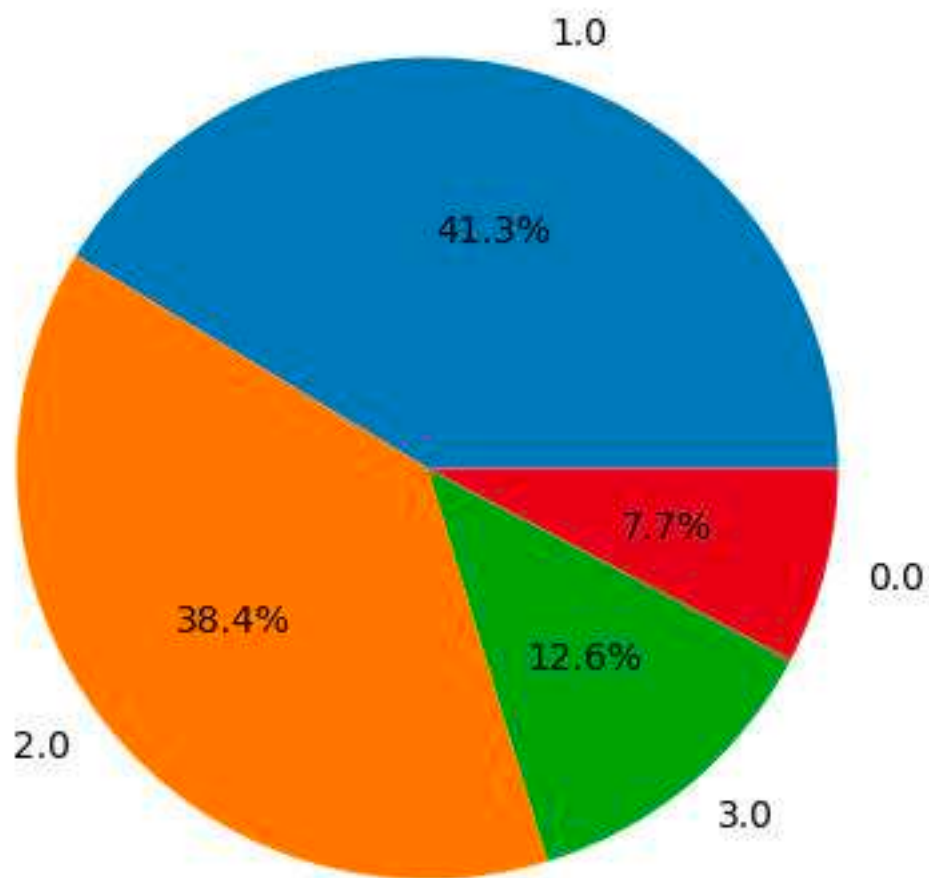


#### PIE CHART:

```
data = df['balcony']
neighborhood_counts = data.value_counts() #creates a map of
neighborhood_name: number_of_times_it_appears

#create pie chart
plt.figure(figsize=(5,5)) #size of chart
plt.pie(neighborhood_counts, labels = 
neighborhood_counts.index, autopct='%1.1f%%',
textprops={'color':"black"})
plt.title('Balcony', color="black",fontsize="24")
plt.show()
```

# Location



## EXPERIMENT NO 2

**AIM:** One case study on building Data warehouse/Data Mart. Write Detailed Problem statement and design dimensional modelling (creation of star and snowflake schema)

**PROBLEM STATEMENT:** This study aims to conduct a comprehensive analysis of Bengaluru's housing market to gain insights into the factors that influence house prices. The analysis will involve exploring various features such as location, size, amenities, and neighborhood characteristics, and their impact on property prices. By examining historical data and employing predictive modeling techniques, the goal is to develop a robust understanding of the dynamics driving house prices in Bengaluru and potentially create a model that can predict future property values with reasonable accuracy.

### THEORY:

#### What is star schema?

The star schema is a type of database schema used in data warehousing. It consists of a central fact table that holds quantitative data and is connected to dimension tables. Dimension tables contain descriptive information that provides context to the data in the fact table. This structure helps in organising and querying data efficiently for reporting and analysis purposes. The name "star schema" comes from the visualisation of the schema, where the fact table is at the centre and the dimension tables radiate out like points of a star.

The star schema is a widely used design in data warehousing and business intelligence applications. It's designed to optimise query performance for analytical purposes. Here's a bit more detail:

**Fact Table:** This is the central table in the schema and contains the quantitative measures or metrics that the organisation wants to analyse. These could be sales revenue, profit, quantity sold, etc. Each row in the fact table represents a specific event or transaction.

**Dimension Tables:** These tables contain descriptive attributes that provide context to the data in the fact table. They help in understanding the various aspects or dimensions of the data. For example, in a sales scenario, dimension tables might include information about time (date, month, year), products (product name, category, brand), geography (country, city), and more.



**Star-Like Structure:** The star schema gets its name from its visual representation. The fact table is located at the center of the schema, and the dimension tables surround it, resembling the points of a star. This layout makes it easy to navigate and query the data.

**Denormalization:** In a star schema, dimension tables are often denormalized, meaning redundant data is stored within them. This redundancy improves query performance by reducing the number of joins required for analysis. However, this approach might lead to increased storage requirements.

**Query Performance:** One of the primary advantages of the star schema is its ability to optimize query performance. Since most queries in analytics involve aggregations and grouping based on dimensions, the star schema's structure reduces the complexity of join operations, leading to faster query execution times.

**Ease of Use:** Star schemas are user-friendly for analysts and business users because they mirror the way people naturally think about data. The separation of facts and dimensions simplifies the process of formulating queries and understanding the results.

**Data Integrity:** While star schemas prioritise query performance, they might not be as efficient for maintaining data integrity as highly normalised schemas. Changes to dimension data need to be carefully managed to avoid inconsistencies.

## **What is snowflake schema?**

The snowflake schema is a database schema design used in data warehousing, similar to the star schema. It's named after its visual representation, which resembles a snowflake due to its normalized structure. In a snowflake schema, dimension tables are organized into a hierarchy of related tables, reducing data redundancy compared to the denormalized dimension tables in a star schema.

In a snowflake schema:

**Normalization:** Dimension tables are normalized by breaking down data into multiple related tables. For example, attributes of a dimension might be stored in separate tables, creating a hierarchy of related tables. This reduces data redundancy and can lead to more efficient storage.

**Hierarchy:** Each level of the hierarchy is stored in a separate table, and these tables are linked through foreign key relationships. This creates a branching structure where tables are connected in a way that resembles a snowflake.

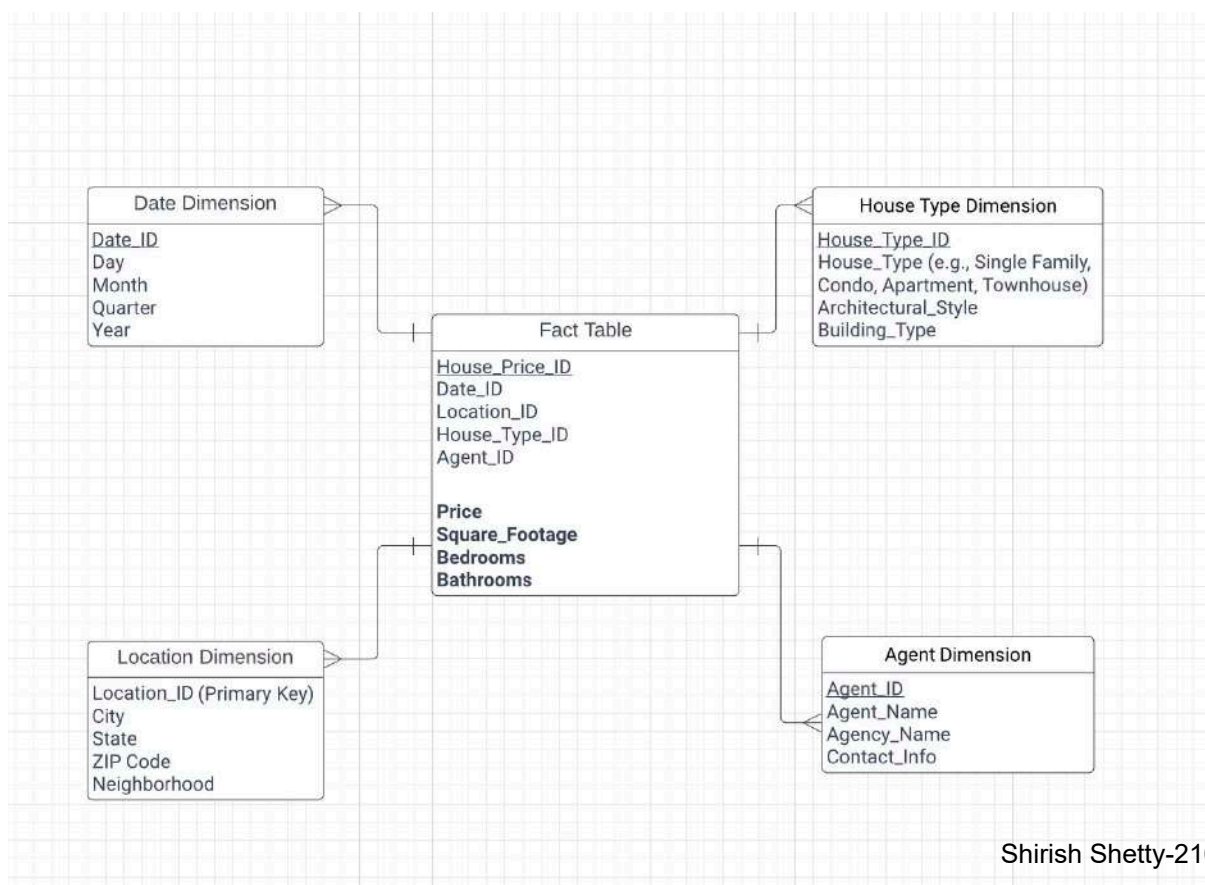
**Complexity:** Snowflake schemas can be more complex to navigate and query compared to star schemas. Queries might involve more joins and subqueries due to the normalized structure.

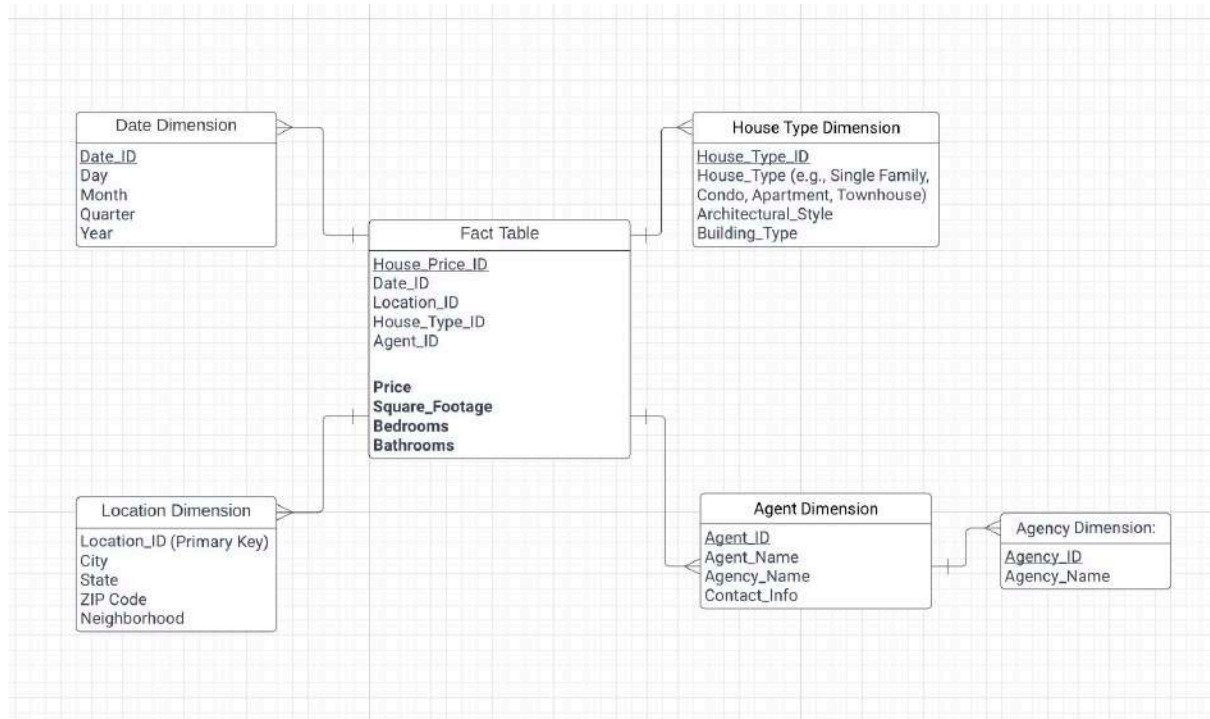
**Storage Efficiency:** The snowflake schema can save storage space because data redundancy is minimized. However, this benefit needs to be balanced against potential query complexity.

**Data Integrity:** Due to the normalized structure, maintaining data integrity can be easier compared to star schemas. Changes to data only need to be made in one place.

**Query Performance:** Query performance in snowflake schemas might be slower compared to star schemas because of the increased number of joins and complexity involved in querying normalized data.

## STAR SCHEMA





## Snowflake Schema

## EXPERIMENT NO 3

### AIM : Implementation of all dimension table and fact table

#### THEORY :

In the realm of data warehousing and dimensional modelling, fact tables and dimension tables are fundamental components that help organize and structure data for efficient querying and analysis. They are key elements of a star schema or snowflake schema, which are commonly used in data warehouses for multidimensional analysis and reporting.

#### Fact Table:

A fact table is the central table in a dimensional model that contains quantitative data, also known as measures or facts. These facts are typically numerical and represent the business metrics you want to analyze, such as sales revenue, profit, quantity sold, etc. Each row in the fact table corresponds to a specific event, transaction, or observation in the business process. Facts are usually associated with various dimensions, which provide context to the measures.

#### Key characteristics of a fact table:

1. Measures/Facts: Contains the numerical data you want to analyze.
2. Foreign Keys: Contains foreign keys referencing the dimension tables to establish relationships and provide context to the measures.
3. Granularity: Defines the level of detail at which facts are recorded (e.g., daily, monthly, by product, by region).
4. Aggregation: Fact tables often store data at a detailed level, but they can also include aggregated values for faster query performance.
5. Additive, Semi-Additive, Non-Additive Measures: Facts can be additive (summarizable), semi-additive (summarizable for some dimensions), or non-additive (not summarizable).

#### Dimension Table:

Dimension tables provide descriptive context for the data in the fact table. They contain attributes that help in filtering, grouping, and analyzing the facts. Dimensions provide the "who, what, where, when, and how" aspects of the data, allowing you to slice and dice the facts to gain insights. Dimension tables are usually smaller in size compared to fact tables.

#### Key characteristics of a dimension table:

1. Attributes: Contains textual or categorical data describing the dimensions of the business (e.g., time, product, customer, location).
2. Surrogate Keys: A unique identifier for each dimension row, used to establish relationships

3. with the fact table. This key is typically different from the natural key (e.g., product code, customer ID).
4. Hierarchies: Dimension attributes can be organized into hierarchies for drilling down or rolling up data.
5. Slowly Changing Dimensions (SCDs): Dimension tables can accommodate changes overtime, such as updates to attribute values or new entries.
6. Conformed Dimensions: In data warehousing, it's beneficial to have consistent dimensions shared across multiple fact tables to enable cross-functional analysis

In summary, a fact table holds numerical measures and is linked to dimension tables via foreign keys, providing the context needed for meaningful analysis. Dimension tables contain descriptive attributes that help classify and filter the facts. Together, these components create a structured and efficient framework for querying and analyzing large volumes of data in a data warehouse.

### **Implementation :**

Implementation of Customer and Sales of an organization in Oracle –

### **DIMENSION TABLES :**

#### **DATEDIMENSION TABLE =>**

```
CREATE TABLE DateDimension (
```

```
    Date_ID INT PRIMARY KEY,
```

```
    Date DATE,
```

```
    Day INT,
```

```
    Month INT,
```

```
    Quarter INT,
```

```
    Year INT
```

```
);
```

```
INSERT INTO DateDimension (Date_ID, Date, Day, Month, Quarter, Year)
```

```
VALUES
```

```
(1, '2023-01-01', 1, 1, 1, 2023),
```

```
(2, '2023-01-02', 2, 1, 1, 2023),
```

```
(3, '2023-01-03', 3, 1, 1, 2023),
```

```
(4, '2023-01-04', 4, 1, 1, 2023),
```

```
(5, '2023-01-05', 5, 1, 1, 2023),
```



```
(6, '2023-02-01', 1, 2, 1, 2023),
(7, '2023-02-02', 2, 2, 1, 2023),
(8, '2023-02-03', 3, 2, 1, 2023),
(9, '2023-02-04', 4, 2, 1, 2023),
(10, '2023-02-05', 5, 2, 1, 2023),
(11, '2023-03-01', 1, 3, 1, 2023),
(12, '2023-03-02', 2, 3, 1, 2023),
(13, '2023-03-03', 3, 3, 1, 2023),
(14, '2023-03-04', 4, 3, 1, 2023),
(15, '2023-03-05', 5, 3, 1, 2023);
```

```
select * from DateDimension;
```

	Date_ID	Date	Day	Month	Quarter	Year
▶	1	2023-01-01	1	1	1	2023
	2	2023-01-02	2	1	1	2023
	3	2023-01-03	3	1	1	2023
	4	2023-01-04	4	1	1	2023
	5	2023-01-05	5	1	1	2023
	6	2023-02-01	1	2	1	2023
	7	2023-02-02	2	2	1	2023
	8	2023-02-03	3	2	1	2023
	9	2023-02-04	4	2	1	2023
	10	2023-02-05	5	2	1	2023
	11	2023-03-01	1	3	1	2023
	12	2023-03-02	2	3	1	2023
	13	2023-03-03	3	3	1	2023
	14	2023-03-04	4	3	1	2023
	15	2023-03-05	5	3	1	2023
•	NULL	NULL	NULL	NULL	NULL	NULL

### LocationDimension Table=>

```
CREATE TABLE LocationDimension (
    Location_ID INT PRIMARY KEY,
    City VARCHAR(255),
    State VARCHAR(255),
    ZIP_Code VARCHAR(10),
```

```

Neighborhood VARCHAR(255)

);

INSERT INTO LocationDimension (Location_ID, City, State, ZIP_Code, Neighborhood)
VALUES

(1, 'New York', 'NY', '10001', 'Downtown'),
(2, 'Los Angeles', 'CA', '90001', 'Hollywood'),
(3, 'Chicago', 'IL', '60601', 'Loop'),
(4, 'Houston', 'TX', '77001', 'Downtown'),
(5, 'Miami', 'FL', '33101', 'South Beach'),
(6, 'San Francisco', 'CA', '94101', 'Financial District'),
(7, 'Boston', 'MA', '02101', 'Back Bay'),
(8, 'Seattle', 'WA', '98101', 'Downtown'),
(9, 'Atlanta', 'GA', '30301', 'Midtown'),
(10, 'Dallas', 'TX', '75201', 'Uptown'),
(11, 'Denver', 'CO', '80201', 'Downtown'),
(12, 'Phoenix', 'AZ', '85001', 'Downtown'),
(13, 'Philadelphia', 'PA', '19101', 'Center City'),
(14, 'Las Vegas', 'NV', '89101', 'The Strip'),
(15, 'Austin', 'TX', '78701', 'Downtown');

```

```
Select * from LocationDimension;
```

Location_ID	City	State	ZIP_Code	Neighborhood
1	New York	NY	10001	Downtown
2	Los Angeles	CA	90001	Hollywood
3	Chicago	IL	60601	Loop
4	Houston	TX	77001	Downtown
5	Miami	FL	33101	South Beach
6	San Francisco	CA	94101	Financial District
7	Boston	MA	02101	Back Bay
8	Seattle	WA	98101	Downtown
9	Atlanta	GA	30301	Midtown
10	Dallas	TX	75201	Uptown
11	Denver	CO	80201	Downtown
12	Phoenix	AZ	85001	Downtown
13	Philadelphia	PA	19101	Center City
14	Las Vegas	NV	89101	The Strip
15	Austin	TX	78701	Downtown

### HOUSETYPE DIMENSION TABLE=>

```
CREATE TABLE HouseTypeDimension (  
    House_Type_ID INT PRIMARY KEY,  
    House_Type VARCHAR(255),  
    Architectural_Style VARCHAR(255),  
    Building_Type VARCHAR(255)  
);
```

```
INSERT INTO HouseTypeDimension (House_Type_ID, House_Type, Architectural_Style,  
Building_Type)
```

```
VALUES
```

```
(1, 'Single Family', 'Colonial', 'House'),  
(2, 'Condo', 'Modern', 'Apartment'),  
(3, 'Apartment', 'Contemporary', 'Apartment'),  
(4, 'Townhouse', 'Victorian', 'Townhouse'),  
(5, 'Condo', 'Rustic', 'Apartment'),  
(6, 'Single Family', 'Ranch', 'House'),  
(7, 'Apartment', 'Art Deco', 'Apartment'),  
(8, 'Townhouse', 'Modern', 'Townhouse'),  
(9, 'Single Family', 'Victorian', 'House'),  
(10, 'Condo', 'Colonial', 'Apartment'),  
(11, 'Single Family', 'Cottage', 'House'),  
(12, 'Apartment', 'Art Nouveau', 'Apartment'),  
(13, 'Townhouse', 'Craftsman', 'Townhouse'),  
(14, 'Single Family', 'Cape Cod', 'House'),  
(15, 'Condo', 'Mediterranean', 'Apartment');
```

```
Select * from HouseTypeDimension;
```

House_Type_ID	House_Type	Architectural_Style	Building_Type
1	Single Family	Colonial	House
2	Condo	Modern	Apartment
3	Apartment	Contemporary	Apartment
4	Townhouse	Victorian	Townhouse
5	Condo	Rustic	Apartment
6	Single Family	Ranch	House
7	Apartment	Art Deco	Apartment
8	Townhouse	Modern	Townhouse
9	Single Family	Victorian	House
10	Condo	Colonial	Apartment
11	Single Family	Cottage	House
12	Apartment	Art Nouveau	Apartment
13	Townhouse	Craftsman	Townhouse
14	Single Family	Cape Cod	House
15	Condo	Mediterranean	Apartment

### **AGENT TYPE DIMENSION TYPE =>**

```
CREATE TABLE AgentDimension (
    Agent_ID INT PRIMARY KEY,
    Agent_Name VARCHAR(255),
    Agency_Name VARCHAR(255),
    Contact_Info VARCHAR(255)
);
```

```
INSERT INTO AgentDimension (Agent_ID, Agent_Name, Agency_Name, Contact_Info)
VALUES
    (1, 'John Doe', 'ABC Realty', 'john.doe@example.com'),
    (2, 'Jane Smith', 'XYZ Real Estate', 'jane.smith@example.com'),
    (3, 'David Johnson', 'Dream Homes', 'david.johnson@example.com'),
    (4, 'Emily White', 'Elite Properties', 'emily.white@example.com'),
    (5, 'Michael Brown', 'Best Homes', 'michael.brown@example.com'),
    (6, 'Sarah Davis', 'Sunshine Realty', 'sarah.davis@example.com'),
    (7, 'Robert Lee', 'New Horizons Realty', 'robert.lee@example.com'),
    (8, 'Linda Wilson', 'Golden Gate Realty', 'linda.wilson@example.com'),
    (9, 'William Taylor', 'Prime Properties', 'william.taylor@example.com'),
    (10, 'Susan Turner', 'Sunset Real Estate', 'susan.turner@example.com'),
    (11, 'Richard Clark', 'Blue Sky Realty', 'richard.clark@example.com'),
```

(12, 'Patricia Anderson', 'Skyline Properties', 'patricia.anderson@example.com'),  
 (13, 'Charles Hall', 'Harbor Homes', 'charles.hall@example.com'),  
 (14, 'Margaret Garcia', 'Majestic Realty', 'margaret.garcia@example.com'),  
 (15, 'Thomas Martinez', 'Evergreen Estates', 'thomas.martinez@example.com');

select \* from AgentDimension;

Agent_ID	Agent_Name	Agency_Name	Contact_Info
1	John Doe	ABC Realty	john.doe@example.com
2	Jane Smith	XYZ Real Estate	jane.smith@example.com
3	David Johnson	Dream Homes	david.johnson@example.com
4	Emily White	Elite Properties	emily.white@example.com
5	Michael Brown	Best Homes	michael.brown@example.com
6	Sarah Davis	Sunshine Realty	sarah.davis@example.com
7	Robert Lee	New Horizons Realty	robert.lee@example.com
8	Linda Wilson	Golden Gate Realty	linda.wilson@example.com
9	William Taylor	Prime Properties	william.taylor@example.com
10	Susan Turner	Sunset Real Estate	susan.turner@example.com
11	Richard Clark	Blue Sky Realty	richard.clark@example.com
12	Patricia Ander...	Skyline Properties	patricia.anderson@example....
13	Charles Hall	Harbor Homes	charles.hall@example.com
14	Margaret Garcia	Majestic Realty	margaret.garcia@example.com
15	Thomas Marti...	Evergreen Estates	thomas.martinez@example.c...

## HousePricesFact Table=>

CREATE TABLE HousePricesFact (

House\_Price\_ID INT PRIMARY KEY,

Date\_ID INT,

Location\_ID INT,

House\_Type\_ID INT,

Agent\_ID INT,

Price DECIMAL(10,2),

Square\_Footage INT,

Bedrooms INT,

Bathrooms INT,

FOREIGN KEY (Date\_ID) REFERENCES DateDimension (Date\_ID),

FOREIGN KEY (Location\_ID) REFERENCES LocationDimension (Location\_ID),



```

FOREIGN KEY (House_Type_ID) REFERENCES HouseTypeDimension (House_Type_ID),
FOREIGN KEY (Agent_ID) REFERENCES AgentDimension (Agent_ID)
);

-- Insert values into the HousePricesFact table

INSERT INTO HousePricesFact (House_Price_ID, Date_ID, Location_ID, House_Type_ID, Agent_ID,
Price, Square_Footage, Bedrooms, Bathrooms)
VALUES

(1, 1, 1, 1, 1, 500000.00, 2000, 3, 2),
(2, 2, 2, 2, 2, 350000.00, 1500, 2, 2),
(3, 3, 3, 3, 3, 250000.00, 1200, 1, 1),
(4, 4, 4, 4, 4, 600000.00, 2500, 4, 3),
(5, 5, 5, 5, 5, 450000.00, 1800, 3, 2),
(6, 6, 6, 6, 6, 700000.00, 2800, 4, 3),
(7, 7, 7, 7, 7, 300000.00, 1400, 2, 2),
(8, 8, 8, 8, 8, 550000.00, 2100, 3, 2),
(9, 9, 9, 9, 9, 400000.00, 1600, 3, 2),
(10, 10, 10, 10, 10, 750000.00, 3000, 4, 3),
(11, 11, 11, 11, 11, 320000.00, 1300, 2, 2),
(12, 12, 12, 12, 12, 480000.00, 1900, 3, 2),
(13, 13, 13, 13, 13, 850000.00, 3500, 4, 3),
(14, 14, 14, 14, 14, 400000.00, 1600, 3, 2),
(15, 15, 15, 15, 15, 600000.00, 2400, 3, 3);

SELECT * FROM HousePricesFact;

```

House_Price_ID	Date_ID	Location_ID	House_Type_ID	Agent_ID	Price	Square_Footage	Bedrooms	Bathrooms
1	1	1	1	1	500000.00	2000	3	2
2	2	2	2	2	350000.00	1500	2	2
3	3	3	3	3	250000.00	1200	1	1
4	4	4	4	4	600000.00	2500	4	3
5	5	5	5	5	450000.00	1800	3	2
6	6	6	6	6	700000.00	2800	4	3
7	7	7	7	7	300000.00	1400	2	2
8	8	8	8	8	550000.00	2100	3	2
9	9	9	9	9	400000.00	1600	3	2
10	10	10	10	10	750000.00	3000	4	3
11	11	11	11	11	320000.00	1300	2	2
12	12	12	12	12	480000.00	1900	3	2
13	13	13	13	13	850000.00	3500	4	3
14	14	14	14	14	400000.00	1600	3	2
15	15	15	15	15	600000.00	2400	3	3

### //creating workshop to perform olap

```
CREATE ANALYTIC WORKSPACE PriceAnalysis;
```

```
-- Create the cube
```

```
CREATE CUBE house_prices_cube
```

```
  DIMENSION BY (DateDimension, LocationDimension, HouseTypeDimension, AgentDimension)
```

```
  MEASURES (Price, Square_Footage, Bedrooms, Bathrooms)
```

```
  USING PriceAnalysis;
```

```
//Olap operations:
```

```
//drill
```

```
SELECT EXTRACT(YEAR FROM dd.Date) AS Year, SUM(hpf.Price) AS Total_Price
```

```
FROM HousePricesFact hpf
```

```
JOIN DateDimension dd ON hpf.Date_ID = dd.Date_ID
```

```
GROUP BY EXTRACT(YEAR FROM dd.Date)
```

```
ORDER BY Year;
```

	Year	Total_Price
▶	2023	7500000.00

```
//roll up
```

```
SELECT DateDimension.Quarter, SUM(HousePricesFact.Price) AS TotalPrice
```

```

FROM HousePricesFact
JOIN DateDimension ON HousePricesFact.Date_ID = DateDimension.Date_ID
GROUP BY DateDimension.Quarter
ORDER BY DateDimension.Quarter;

```

	Quarter	TotalPrice
▶	1	7500000.00

//dice

```

SELECT
    DateDimension.Year,
    LocationDimension.City,
    HouseTypeDimension.House_Type,
    AgentDimension.Agent_Name,
    SUM(HousePricesFact.Price) AS Total_Price
FROM
    HousePricesFact
INNER JOIN DateDimension ON HousePricesFact.Date_ID = DateDimension.Date_ID
INNER JOIN LocationDimension ON HousePricesFact.Location_ID = LocationDimension.Location_ID
INNER JOIN HouseTypeDimension ON HousePricesFact.House_Type_ID =
HouseTypeDimension.House_Type_ID
INNER JOIN AgentDimension ON HousePricesFact.Agent_ID = AgentDimension.Agent_ID
WHERE
    DateDimension.Year = 2023
    AND LocationDimension.City = 'New York'
    AND HouseTypeDimension.House_Type = 'Single Family'
    AND AgentDimension.Agent_Name = 'John Doe'
GROUP BY
    DateDimension.Year,
    LocationDimension.City,
    HouseTypeDimension.House_Type,

```

AgentDimension.Agent\_Name;

	Year	City	House_Type	Agent_Name	Total_Price
▶	2023	New York	Single Family	John Doe	500000.00

//slice

SELECT ht.House\_Type AS House\_Type,

SUM(hpf.Price) AS Total\_Revenue

FROM HousePricesFact hpf

JOIN DateDimension dd ON hpf.Date\_ID = dd.Date\_ID

JOIN HouseTypeDimension ht ON hpf.House\_Type\_ID = ht.House\_Type\_ID

WHERE dd.Year = 2023

GROUP BY ht.House\_Type;

House_Type	Total_Revenue
Single Family	2320000.00
Condo	2150000.00
Apartment	1030000.00
Townhouse	2000000.00

//pivot

SELECT

Year,

MAX(CASE WHEN House\_Type = 'Single Family' THEN Price ELSE NULL END) AS  
Single\_Family\_Price,

MAX(CASE WHEN House\_Type = 'Condo' THEN Price ELSE NULL END) AS Condo\_Price,

MAX(CASE WHEN House\_Type = 'Apartment' THEN Price ELSE NULL END) AS Apartment\_Price,

MAX(CASE WHEN House\_Type = 'Townhouse' THEN Price ELSE NULL END) AS Townhouse\_Price

FROM (

SELECT

YEAR(Date) AS Year,

House\_Type,

```

SUM(Price) AS Price
FROM HousePricesFact
INNER JOIN DateDimension ON HousePricesFact.Date_ID = DateDimension.Date_ID
INNER JOIN HouseTypeDimension ON HousePricesFact.House_Type_ID =
HouseTypeDimension.House_Type_ID
GROUP BY YEAR(Date), House_Type
) AS PivotData
GROUP BY Year
ORDER BY Year;

```

	Year	Single_Family_Price	Condo_Price	Apartment_Price	Townhouse_Price
►	2023	2320000.00	2150000.00	1030000.00	2000000.00

## CONCLUSION :

In this experiment we have learnt about Fact table and Dimension Tables, their features and advantages. We also built database schema of Fact and Dimension tables in Oracle.



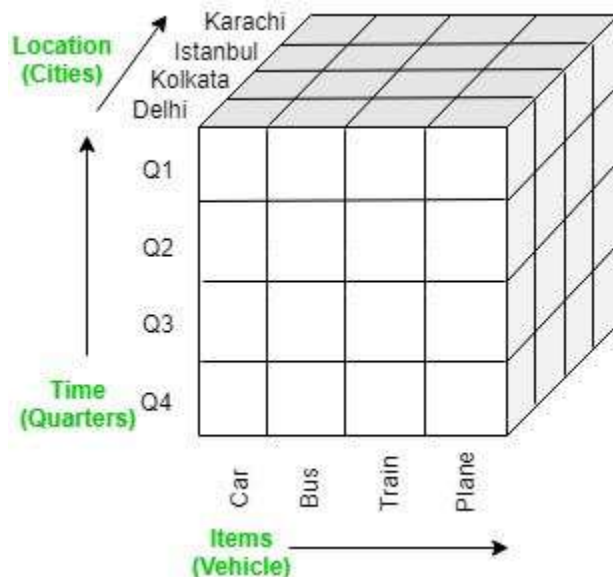
## EXPERIMENT NO 4

**Aim:** Implementation of OLAP operations: Slice, Dice, Rollup, Drilldown, and Pivot based on experiment 1.

### Theory:

#### OLAP Operations

OLAP stands for *Online Analytical Processing* Server. It is a software technology that allows users to analyze information from multiple database systems at the same time. It is based on multidimensional data model and allows the user to query on multi-dimensional data (eg. Delhi -> 2018 -> Sales data). OLAP databases are divided into one or more cubes and these cubes are known as *Hyper-cubes*.



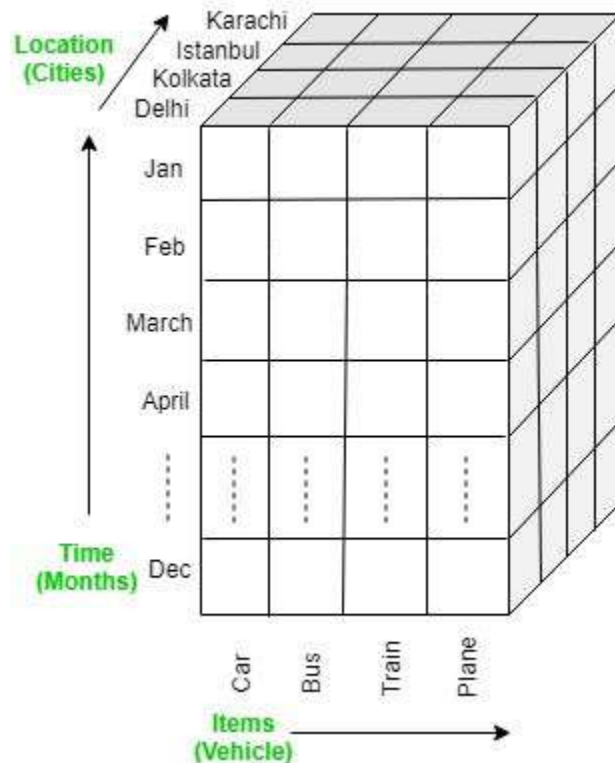
OLAP operations:

There are five basic analytical operations that can be performed on an OLAP cube:

1. 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

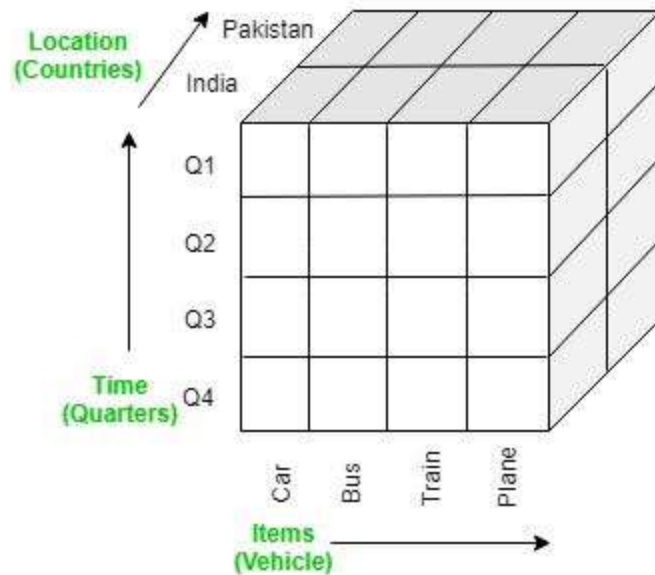
- In the cube given in overview section, the drill down operation is performed by moving down in the concept hierarchy of *Time* dimension (Quarter -> Month).



**2.Roll up:** It is just opposite of the drill-down operation. It performs aggregation on the OLAP cube. It can be done by:

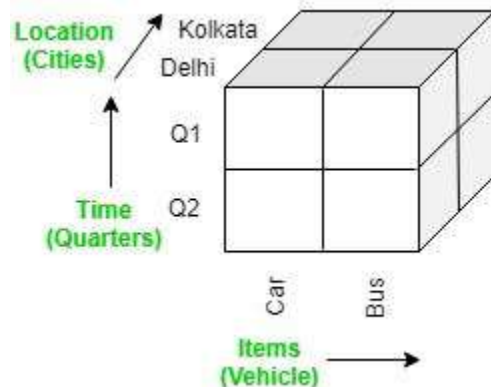
- Climbing up in the concept hierarchy
- Reducing the dimensions

. In the cube given in the overview section, the roll-up operation is performed by climbing up in the concept hierarchy of *Location* dimension (City -> Country).



3.**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”



4.**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				
	Car	Bus	Train	Plane

**5.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.

Car				
Bus				
Train				
Plane				
	Delhi	Kolkata	Istanbul	Karachi

**SLICE:**

**Average Price by Area Type and Availability:**

**Row Labels:** `area_type`

**Column Labels:** `availability`

**Values:** `price (average)`

AVERAGE of price	availability					
area_type	18-Feb	18-M ay	18-N ov	19-Dec c	Ready To Move	Grand Total
Built-up Area					39	39
Plot Area					262.6	262.6
Super built-up Area	70	130	25.2 5	80.53 5	125.645833 3	112.448333 3
<b>Grand Total</b>	<b>70</b>	<b>130</b>	<b>25.2 5</b>	<b>80.53 5</b>	<b>146.8875</b>	<b>132.848846 2</b>

This setup will display the average price of properties in different area types and their availability status.

DICE:

location	18-Feb	18-May	18-Nov	19-Dec	Ready To Move	Grand Total
7th Phase JP Nagar					38	38
Bellandur					103	103
Binny Pete				122		122
Bisuvanahalli					48	48

Chikka Tirupathi					120	120
Electronic City Phase II				39.07		39.07
Gandhi Bazar					370	370
Gottigere					40	40
Kengeri					15	15
Kothanur					51	51
Lingadheeranahalli					95	95
Manayata Tech Park					48	48
Mangammanapalya		56				56
Marathahalli					63.25	63.25
Mysore Road					73.5	73.5
Old Airport Road		204				204
Raja Rajeshwari Nagar					60	60
Rajaji Nagar					600	600
Ramakrishnappa Layout					290	290
Sarjapur					148	148
Thanisandra			25.25		380	405.25
Uttarahalli					62	62
Whitefield	70				333	403
<b>Grand Total</b>	<b>70</b>	<b>260</b>	<b>25.25</b>	<b>161.07</b>	<b>2937.75</b>	<b>3454.07</b>

**This combination would show house prices by location and availability status. You can see how prices differ in each location and whether properties are ready to move or have specific availability dates**

**ROLL UP:**

### Area Type Roll-Up (Higher to Lower):

- **Rows:** area\_type
- **Values:** price

area_type	SUM of price
Built-up Area	117
Plot Area	1313
Super built-up Area	2024.07
<b>Grand Total</b>	<b>3454.07</b>

### DRILL DOWN:

Location Drill Down:

- **Rows:** location
- **Values:** price

7th Phase JP Nagar	38
Bellandur	103
Binny Pete	122
Bisuvanahalli	48
Chikka Tirupathi	120
Electronic City Phase II	39.07
Gandhi Bazar	370
Gottigere	40
Kengeri	15
Kothanur	51



Lingadheeranahalli	95
Manayata Tech Park	48
Mangammanapaly a	56
Marathahalli	63.25
Mysore Road	73.5
Old Airport Road	204
Raja Rajeshwari Nagar	60
Rajaji Nagar	600
Ramakrishnappa Layout	290
Sarjapur	148
Thanisandra	202.625
Uttarahalli	62
Whitefield	134.3333333
<b>Grand Total</b>	<b>132.8488462</b>

## EXPERIMENT NO 5

**Aim:** Implementation of Bayesian algorithm

**Theory:**

Naive Bayes Algorithm:

Naïve 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.

Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which

helps in building the fast machine learning models that can make quick predictions.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

**Naive:** It is called Naïve 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 on the bases of 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. Bayes' Theorem:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, 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:

$P(A|B)$  = Conditional Probability of A given B

$P(B|A)$  = Conditional Probability of A given B

$P(A)$  = Probability of event A

$P(B)$  = Probability of event A

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.

### Problem Description:

Algorithm:

1. Load the "dataset.csv" dataset.
2. For each feature in the dataset create a frequency table to tally the occurrences of each value concerning the positive and negative class outcomes (Accident = Yes or No).
3. After counting, compute the probabilities for all feature values.
4. Accept user input for the specific conditions requiring a prediction
5. Calculate two probabilities using the feature values and the "Sale" variable, considering one case with "Accident" as "Yes" and another with "Accident" as "No."
6. To make a prediction, normalise each probability from Step 5 by dividing it by the sum of both probabilities, ensuring a valid probability value.
7. If the probability value for "Sale" being "Yes" is higher, it suggests an accident is likely; otherwise, the outcome is negative, indicating no accident.

### Code:

```
import pandas as pd

class NaiveBayesClassifier:
    def __init__(self, dataset):
        self.dataset = dataset
        self.class_probabilities = {}
        self.feature_probabilities = {}

    def preprocess(self):
        # Process dataset and calculate probabilities
        total_samples = len(self.dataset)
        outcomes = self.dataset['outcome'].unique()

        for outcome in outcomes:
```

```

        outcome_samples = self.dataset[self.dataset['outcome'] == outcome]
        self.class_probabilities[outcome] = len(outcome_samples) / total_samples

    for feature in self.dataset.columns[:-1]:
        self.feature_probabilities[feature] = {}
        for value in self.dataset[feature].unique():
            feature_outcome_samples = outcome_samples[outcome_samples[feature]
== value]

            self.feature_probabilities[feature][f"{value}_{outcome}"] =
len(feature_outcome_samples) / len(outcome_samples)

    def predict(self, input_tuple):
        outcome_probabilities = {outcome: 1 for outcome in self.class_probabilities}

        for feature_value in input_tuple:
            for outcome in outcome_probabilities:
                feature_outcome_key = f"{feature_value}_{outcome}"
                if feature_outcome_key in self.feature_probabilities:
                    outcome_probabilities[outcome] *=
self.feature_probabilities[feature_value][feature_outcome_key]

        for outcome in outcome_probabilities:
            outcome_probabilities[outcome] *= self.class_probabilities[outcome]

        return max(outcome_probabilities, key=outcome_probabilities.get)

def main():

    dataset = pd.read_csv('newdata.csv')

    nb_classifier = NaiveBayesClassifier(dataset)
    nb_classifier.preprocess()

    test_input = input("Enter comma-separated test tuple (e.g.,
Rain,Average,High,Yes): ")

    test_data = test_input.split(',')

    predicted_outcome = nb_classifier.predict(test_data)

```

```

probs_outcome = {}
for outcome in nb_classifier.class_probabilities:
    outcome_probabilities = 1

    for feature_value in test_data:
        feature_outcome_key = f"{feature_value}_{outcome}"
        if feature_outcome_key in nb_classifier.feature_probabilities:
            outcome_probabilities *=
nb_classifier.feature_probabilities[feature_value][feature_outcome_key]

    outcome_probabilities *= nb_classifier.class_probabilities[outcome]
    probs_outcome[outcome] = outcome_probabilities

print("Probability of Positive Outcome:", probs_outcome['positive'])
print("Probability of Negative Outcome:", probs_outcome['negative'])

print("Predicted Outcome:", predicted_outcome)

if __name__ == "__main__":
    main()

```

## OUTPUT:

```

Enter comma-separated test tuple:
Super built-up Area,19-Dec,Electronic City Phase II,2 BHK,1056,2.0,1.0,39.07

```

```

Probability of Positive Outcome: 0.5882352941176471
Probability of Negative Outcome: 0.4117647058823529
Predicted Outcome: positive

```

## EXPERIMENT NO 6

**AIM:** Perform data Pre-processing task and Demonstrate performing Classification, Clustering, and Association algorithm on data sets using a data mining tool (WEKA/R tool)

### WEKA:

WEKA - an open-source software that 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. Weka is a popular and widely used open-source machine learning and data mining software tool. It was developed by the Machine Learning Group at the University of Waikato in New Zealand. Weka provides a graphical user interface (GUI) for working with various machine learning algorithms and data preprocessing techniques, making it accessible to both beginners and experienced data scientists.

Key features of Weka include:

1. **Data Preprocessing:** Weka offers a wide range of tools for data preprocessing, including options for data cleaning, transformation, and feature selection. You can prepare your data for machine learning tasks easily within the software.
2. **Machine Learning Algorithms:** Weka includes a large collection of machine learning algorithms for tasks such as classification, regression, clustering, association rule mining, and more. These algorithms are implemented and readily available for use.
3. **Visual Interface:** Weka's graphical user interface allows users to interact with the software visually, making it easier to experiment with different algorithms and configurations. You can build, train, and evaluate machine learning models without writing code.
4. **Scripting:** Advanced users can also access Weka's functionality through scripting using the Weka API, which is available in Java. This allows for more programmatic and automated machine learning workflows.
5. **Integration:** Weka can be integrated with other data analysis and visualization tools, making it part of a larger data analytics ecosystem.
6. **Community Support:** Weka has a strong user community, and there are many resources available, including documentation, tutorials, and forums where users can seek help and share their experiences.
7. **Extensibility:** Weka allows for the development and integration of custom machine learning algorithms and data preprocessing techniques.

Weka is commonly used in education, research, and industry for tasks such as data analysis, classification, regression, and more. It's a versatile tool that can be useful for both learning about machine learning concepts and conducting practical data analysis and modeling projects.

## 1. Naive Bayes Algorithm:

- **Algorithm Type:** Supervised Learning (Classification)
- **Description:** Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem. It's particularly well-suited for classification tasks. Despite its simplicity, it often performs surprisingly well in practice.
- **How it Works:** Naive Bayes calculates the probability of a data point belonging to a particular class based on the conditional probabilities of its features given that class. It makes the "naive" assumption that all features are conditionally independent, which simplifies the calculation. This assumption, though often not strictly true in real-world data, works reasonably well in many cases.
- **Applications:** Naive Bayes is commonly used in text classification (e.g., spam detection or sentiment analysis), email filtering, and medical diagnosis.

## 2. J48 Algorithm (C4.5 Decision Tree):

- **Algorithm Type:** Supervised Learning (Classification)
- **Description:** J48, also known as C4.5, is a decision tree algorithm used for classification. It constructs a tree structure where each node represents a decision based on a specific feature.
- **How it Works:** The algorithm iteratively selects the best feature to split the data into more homogeneous groups (classes). It uses criteria such as information gain or Gini impurity to determine the best splits. The process continues until a stopping criterion is met, resulting in a decision tree.
- **Applications:** Decision trees are used for a wide range of classification tasks, including customer segmentation, fraud detection, and medical diagnosis.

## 3. K-Means Algorithm:

- **Algorithm Type:** Unsupervised Learning (Clustering)

**Description:** K-Means is a clustering algorithm used to group data points into clusters based on their similarity. It aims to minimize the distance between data points within the same cluster (inertia) and maximize the distance between clusters.

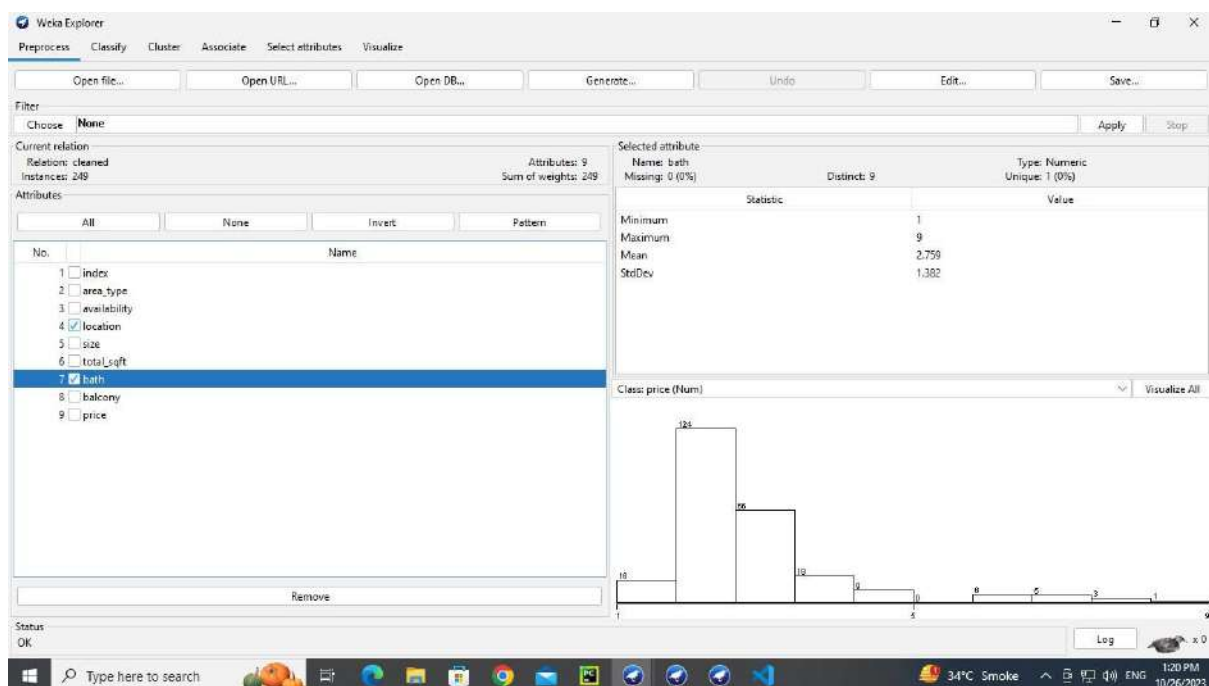
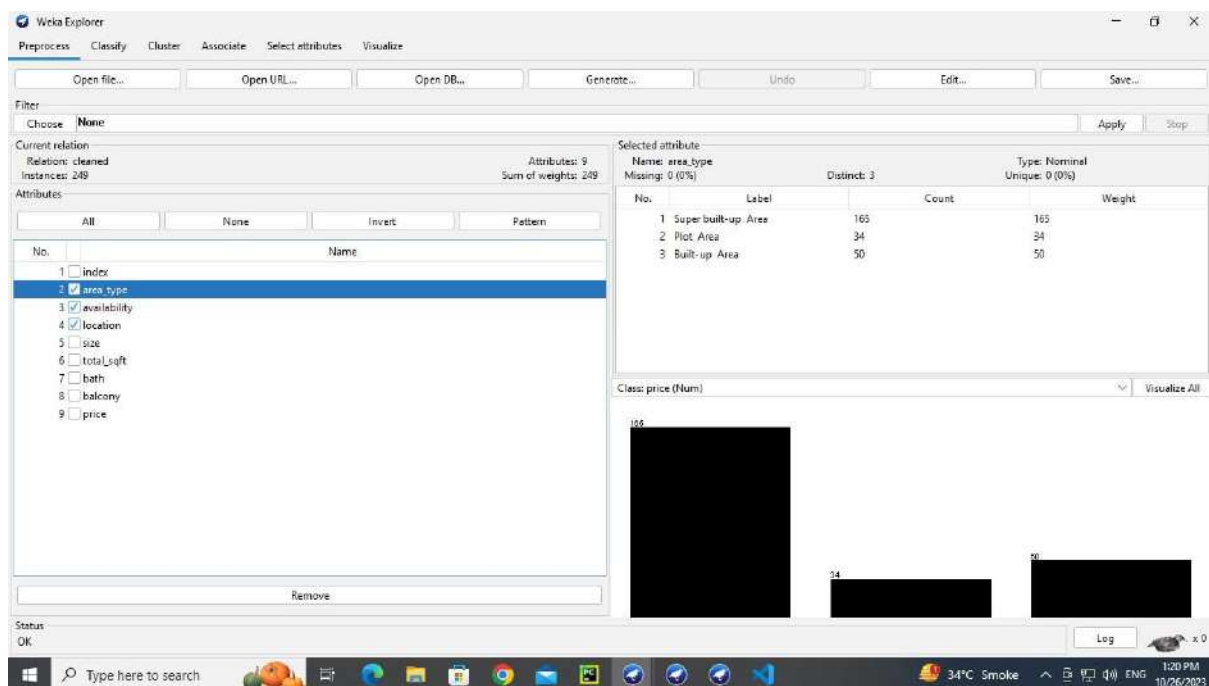
- **How it Works:** K-Means starts by randomly initializing cluster centroids. It then assigns each data point to the nearest centroid and updates the centroids based on the mean of the data points in each cluster. This process repeats until convergence.
- **Applications:** K-Means is used for customer segmentation, image compression, anomaly detection, and recommendation systems.

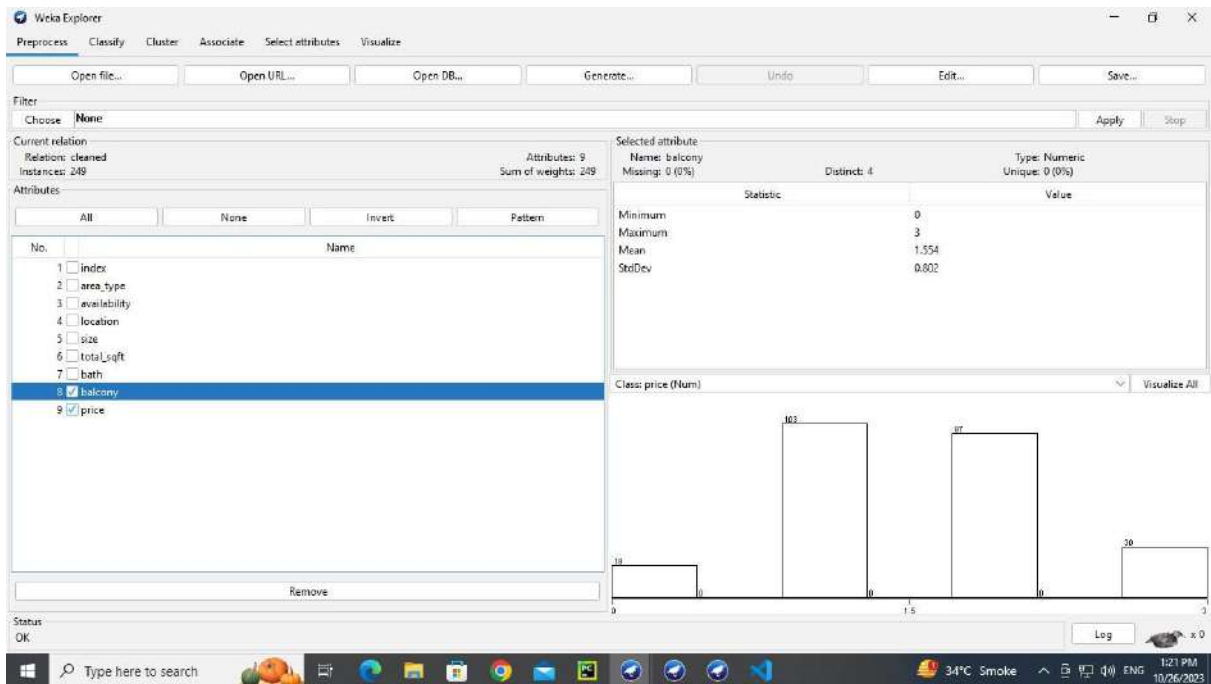
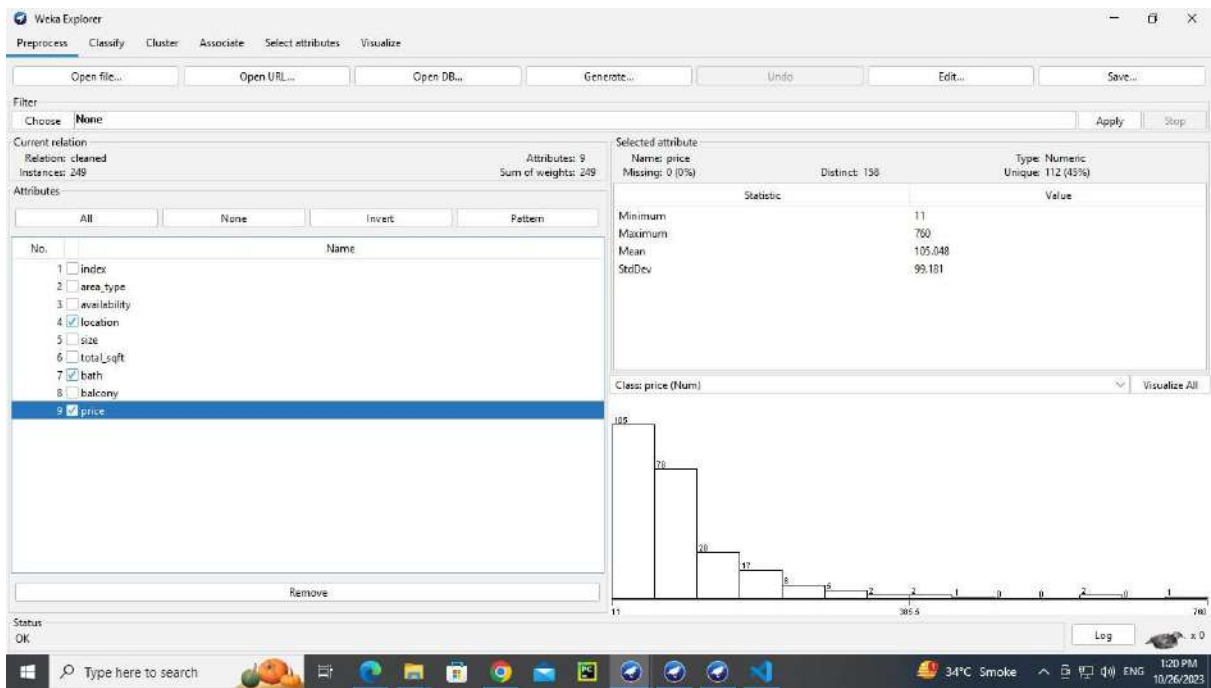
## 4. Filtered Associator Algorithm:

- **Algorithm Type:** Data Mining (Association Rule Mining)



- **Description:** The Filtered Associator combines association rule mining with filtering techniques to discover interesting patterns and associations in data.
- **How it Works:** It starts by discovering association rules in a dataset. Association rules are "if-then" statements that describe relationships between different items in a dataset. The algorithm then applies filters to these rules to extract the most relevant and interesting associations.
- **Applications:** Filtered Associator is used in market basket analysis (e.g., identifying items that are frequently purchased together in a store), recommendation systems, and fraud detection.





Weka Explorer

Preprocess    Classify    **Cluster**    Associate    Select attributes    Visualize

Clusterer  
Choose: HierarchicalClusterer -N 2-L SINGLE-P -A 'weka.core.EuclideanDistance' -R first-last

Cluster mode  
☒ Use training set  
☐ Supplied test set:   
☐ Percentage split: %   
☐ Classes to clusters evaluation (Num) price  
☒ Store clusters for visualization

Ignore attributes

Result list (right-click for options)  
 13:26:50 HierarchicalClusterer

Cluster output  
 Relation: cleaned  
 Instances: 249  
 Attributes: 9  
 index  
 area\_type  
 availability  
 location  
 size  
 total\_sqft  
 bath  
 balcony  
 price  
 Test mode: evaluate on training data

```

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

Cluster 0
{(((((1056:1.1753,1020:1.1753):0.23506,(((2600:1.02772,(2785:0.51666,5700:0.51666):0.51107):0.03339,2800:

Time taken to build model (full training data) : 0.1 seconds

--- Model and evaluation on training set ---

Clustered Instances

0      248 (100%)
1       1 ( 0%)
  
```

Status  
OK

Log

Windows taskbar: Type here to search, 34°C Smoke, 1:26 PM 10/26/2023

Windows taskbar showing search bar, taskbar icons (Edge, File Explorer, Mail, PC, etc.), system tray (34°C, Smoke, ENG, 1:30 PM, 10/26/2023).

Windows taskbar showing search bar, taskbar icons (File Explorer, Edge, Chrome, Mail, PC), system tray (34°C, Smoke, ENG, 10/26/2023).

## Experiment 7

### Aim: - Implementation of Clustering Algorithm (K-Means)

#### Theory:

K-means is a popular clustering algorithm in machine learning and data analysis. It is used to group similar data points or objects together into clusters based on their features or attributes. The primary goal of K-means clustering is to partition a dataset into K distinct, non-overlapping clusters, with each data point belonging to the cluster with the nearest mean (centroid).

K-means algorithm

k : number of clusters

n : sample feature vectors  $x_1, x_2, \dots, x_n$

$m_i$  : the mean of the vectors in cluster i

Assume  $k < n$ .

Make initial guesses for the means  $m_1, m_2, \dots, m_k$ .

Until there are no changes in any mean.

Use the estimated means to classify the samples into clusters. o for  $i = 1$  to k

Replace  $m_i$  with the mean of all of the samples for cluster i

o end\_for

o end\_until

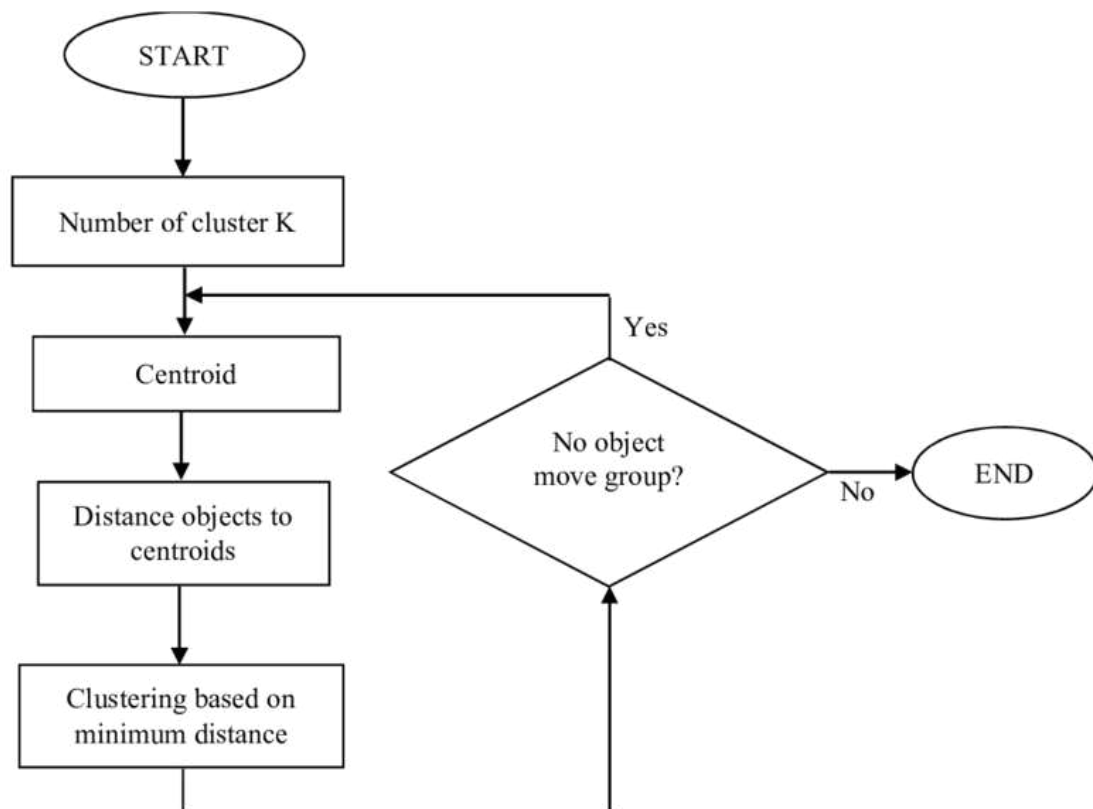
Following three steps are repeated until convergence :

Iterate till no object moves to a different group :

1. Find the centroid coordinate.
2. Find the distance of each object to the centroids.
3. Based on minimum distance, group the objects.

K-means is an iterative algorithm that aims to minimize the within-cluster variance, which measures the distance between data points within the same cluster. It is a relatively simple and efficient algorithm but has some limitations, such as its sensitivity to the initial placement of centroids and its assumption that clusters have similar sizes and shapes (spherical).

K-means clustering is widely used in various fields, including image segmentation, customer segmentation, anomaly detection, and more, for discovering patterns and groupings within data.



Code:

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import pandas as pd
import random

def most_common(lst):
    """
    Return the most frequently occurring element in a list.
    """
    return max(set(lst), key=lst.count)

def euclidean(point, data):
    """
    Return Euclidean distances between a point and a dataset.
    """
    return np.sqrt(np.sum((point - data) ** 2, axis=1))

class KMeans:
    def __init__(self, n_clusters=8, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
  
```

```

def fit(self, X_train):
    # Initialize centroids using random data points
    self.centroids = random.sample(list(X_train), self.n_clusters)

    iteration = 0
    prev_centroids = None
    while iteration < self.max_iter:
        # Assign data points to the nearest centroid
        labels = [np.argmin([np.linalg.norm(x - c) for c in self.centroids]) for
x in X_train]

        # Update centroids as the mean of assigned data points
        new_centroids = [np.mean(X_train[np.array(labels) == i], axis=0) for i
in range(self.n_clusters)]

        # Check for convergence
        if prev_centroids is not None and all((prev == new).all() for prev, new
in zip(prev_centroids, new_centroids)):
            break

        self.centroids = new_centroids
        prev_centroids = new_centroids
        iteration += 1

def evaluate(self, X):
    centroids = []
    centroid_idx = []
    for x in X:
        dists = euclidean(x, self.centroids)
        centroid_idx = np.argmin(dists)
        centroids.append(self.centroids[centroid_idx])
        centroid_idx.append(centroid_idx)
    return centroids, centroid_idx

# Read the dataset from a CSV file
data = pd.read_csv('/Users/maithilishinde/Desktop/dwm/hierarchical clustering/
cleaned.csv')

# Select the columns you want to use for clustering (excluding non-numeric columns)
selected_data = data[['bath', 'balcony', 'price']]

# Convert the selected data to a NumPy array
X = selected_data.values

# Standardize the data

```



```

means = np.mean(X, axis=0)
stds = np.std(X, axis=0)
X = (X - means) / stds

# Determine the number of clusters (you can adjust this)
n_clusters = 3 # Adjust the number of clusters as needed

# Maximum number of iterations
max_iterations = 100 # Adjust the maximum number of iterations as needed

# Initialize KMeans with the desired number of clusters and max iterations
kmeans = KMeans(n_clusters=n_clusters, max_iter=max_iterations)

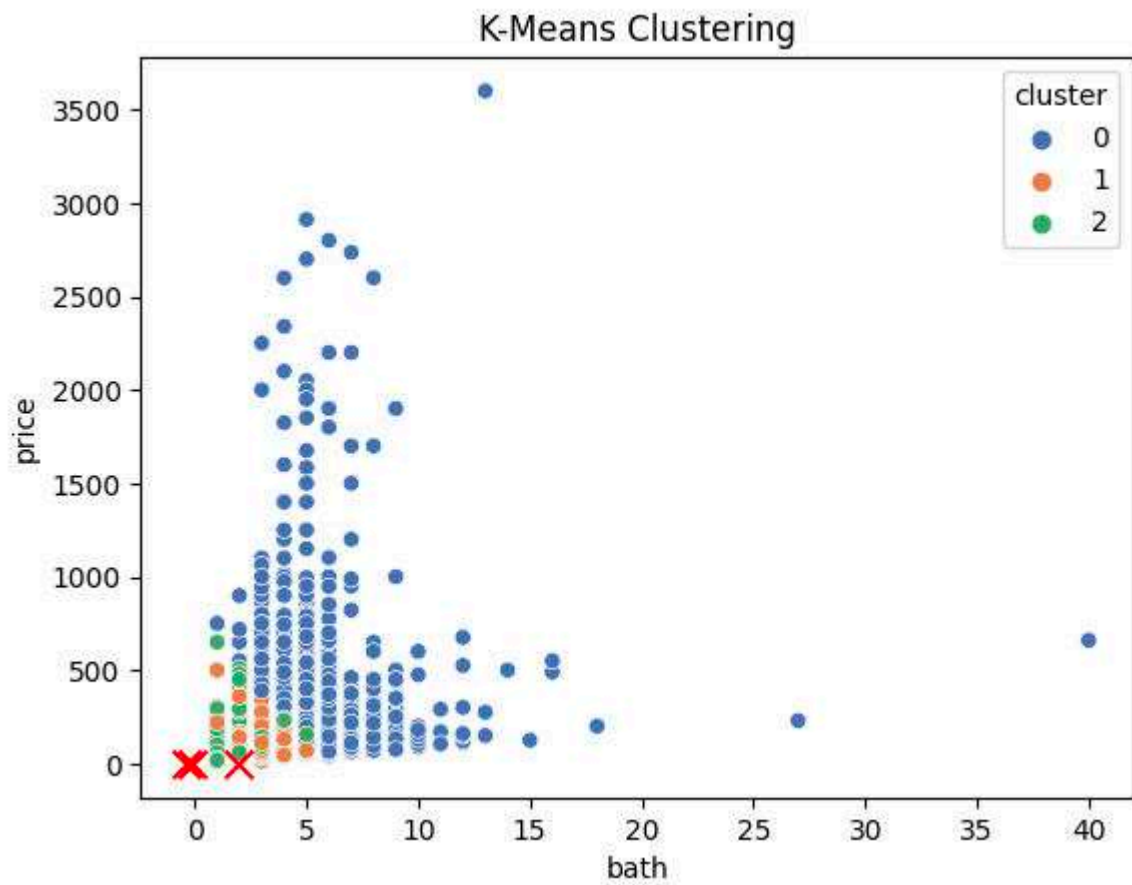
# Fit the data
kmeans.fit(X)

# Get the cluster labels from your KMeans object
labels = kmeans.evaluate(X)[1]

# Add cluster labels to the original dataset
data['cluster'] = labels

# View results
sns.scatterplot(data=data, x='bath', y='price', hue='cluster', palette="deep",
legend="full")
plt.scatter([centroid[0] for centroid in kmeans.centroids], [centroid[3] for
centroid in kmeans.centroids], c='red', marker='x', s=100)
plt.title("K-Means Clustering")
plt.show()

```



## EXPERIMENT NO 8

**Aim:** Implementation of any one Hierarchical Clustering method

**Theory:**

### Theory of Hierarchical Clustering:

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

Steps to Perform Hierarchical Clustering

Following are the steps involved in agglomerative clustering:

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

There

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

are different ways to find distance between the clusters. The distance

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

**Code and output:**

```
import pandas as pd
```

```
import numpy as np
```

```
import scipy.cluster.hierarchy as sch
```

```
import matplotlib.pyplot as plt
```

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

```

# Step 1: Data Preprocessing

# Load the dataset

df = pd.read_csv("newdata.csv")

# Perform one-hot encoding for categorical columns

df = pd.get_dummies(df, columns=['area_type', 'availability'])

# Extract the numeric attributes 'total_sqft', 'bath', 'balcony', 'price'

X = df[['total_sqft', 'bath', 'balcony', 'price']].values


class Distance_computation_grid(object):

    def __init__(self):

        pass

    def compute_distance(self, samples):

        Distance_mat = np.zeros((len(samples), len(samples)))

        for i in range(Distance_mat.shape[0]):

            for j in range(Distance_mat.shape[0]):

                if i != j:

                    Distance_mat[i, j] = float(self.distance_calculate(samples[i],
samples[j]))

                else:

                    Distance_mat[i, j] = 10**4

            return Distance_mat

    def distance_calculate(self, sample1, sample2):

        dist = []

        for i in range(len(sample1)):

```

```

        for j in range(len(sample2)):

            try:

                dist.append(np.linalg.norm(np.array(sample1[i]) -
np.array(sample2[j])))

            except:

                dist.append(self.intersampledlist(sample1[i], sample2[j]))

        return min(dist)

def intersampledlist(self, s1, s2):

    if str(type(s2[0])) != '<class \'list\'>':

        s2 = [s2]

    if str(type(s1[0])) != '<class \'list\'>':

        s1 = [s1]

    m = len(s1)

    n = len(s2)

    dist = []

    if n >= m:

        for i in range(n):

            for j in range(m):

                if (len(s2[i]) >= len(s1[j])) and str(type(s2[i][0])) != '<class
\'list\'>':

                    dist.append(self.interclusterdist(s2[i], s1[j]))

                else:

                    dist.append(np.linalg.norm(np.array(s2[i]) -
np.array(s1[j])))

            else:

                for i in range(m):

```

```

        for j in range(n):

            if (len(s1[i]) >= len(s2[j])) and str(type(s1[i][0])) != '<class
\'list\''>':

                dist.append(self.interclusterdist(s1[i], s2[j]))

            else:

                dist.append(np.linalg.norm(np.array(s1[i]) -
np.array(s2[j])))

        return min(dist)

def interclusterdist(self, cl, sample):

    if str(type(sample[0])) != '<class \'list\''>':

        sample = [sample]

    dist = []

    for i in range(len(cl)):

        for j in range(len(sample)):

            dist.append(np.linalg.norm(np.array(cl[i]) - np.array(sample[j])))

        return min(dist)

# Initialize progression and samples
progression = [[i] for i in range(X.shape[0])]

samples = [[list(X[i])] for i in range(X.shape[0])]

m = len(samples)

distcal = Distance_computation_grid()

while m > 1:

    print('Sample size before clustering :- ', m)

    Distance_mat = distcal.compute_distance(samples)

```

```

sample_ind_needed = np.where(Distance_mat == Distance_mat.min())[0]

value_to_add = samples.pop(sample_ind_needed[1])

samples[sample_ind_needed[0]].append(value_to_add)


print('Cluster Node 1 :-', progression[sample_ind_needed[0]])

print('Cluster Node 2 :-', progression[sample_ind_needed[1]])

progression[sample_ind_needed[0]].append(progression[sample_ind_needed[1]])

progression[sample_ind_needed[0]] = [progression[sample_ind_needed[0]]

v = progression.pop(sample_ind_needed[1])

m = len(samples)


print('Progression(Current Sample) :-', progression)

print('Cluster attained :-', progression[sample_ind_needed[0]])

print('Sample size after clustering :-', m)

print('\n')


# After clustering is done, you can proceed to create a dendrogram

Z = linkage(X, 'single')

fig = plt.figure(figsize=(25, 10))

dn = dendrogram(Z)


plt.show()

Sample size before clustering :- 11

Cluster Node 1 :- [4]

Cluster Node 2 :- [5]

Progression(Current Sample) :- [[0], [1], [2], [3], [[4, [5]]], [6], [7], [8], [9],

```

```

[10]]

Cluster attained :- [[4, [5]]]

Sample size after clustering :- 10


Sample size before clustering :- 10

Cluster Node 1 :- [2]

Cluster Node 2 :- [3]

Progression(Current Sample) :- [[0], [1], [[2, [3]]], [[4, [5]]], [6], [7], [8],
[9], [10]]

Cluster attained :- [[2, [3]]]

Sample size after clustering :- 9


Sample size before clustering :- 9

Cluster Node 1 :- [[4, [5]]]

Cluster Node 2 :- [8]

Progression(Current Sample) :- [[0], [1], [[2, [3]]], [[[4, [5]], [8]]], [6], [7],
[9], [10]]

Cluster attained :- [[[4, [5]], [8]]]

Sample size after clustering :- 8


Sample size before clustering :- 8

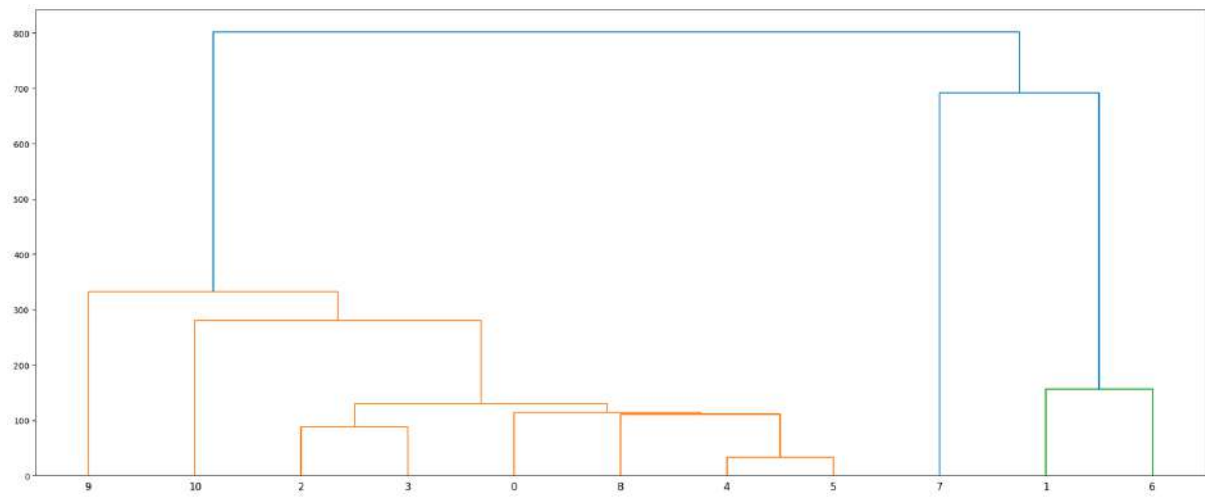
...

Cluster attained :- [[[[[0, [[[4, [5]], [8]]]], [[2, [3]]]], [10]], [9]], [[[1,
[6]], [7]]]]]

Sample size after clustering :- 1

```





## Experiment No. 9

**Aim:** To implement association rule mining algorithm (Apriori Algorithm).

### Theory:

**Apriori algorithm** is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets. To improve the efficiency of level-wise generation of frequent itemsets, an important property is used called *Apriori property* which helps by reducing the search space.

### Problem Description:

This association rule generator is used to mine frequent pattern on a data set that consists of transaction IDs and the posts liked by users (on a social media platform) in that transaction. This program aims to calculate the association between the posts liked by users and the hence determines the likelihood of users liking similar posts in the future.

### Algorithm:

The algorithm used was Apriori algorithm. It generates association rules based upon the prior knowledge of the frequent item set patterns for Boolean association rule generation.

1. Import the CSV file of the dataset.
2. Accept the minimum support and minimum confidence as input from the user.
3. Create a list of all items and assign them to variable keyList.
4. Find all possible combinations of elements in keyList and filter them to only keep those which have support greater than the minimum support.
5. Reassign keyList and repeat step 4 until the length of frequency variable is less than or equal to 1.
6. Find the association rules by finding the power set of the frequency variable's elements.
7. Find the confidence of all the association rules as  $|LHS \cup RHS| / |LHS|$ .
8. Keep only those rules which have confidence  $\geq$  minimum confidence (strong association rules).
9. Print the strong association rules.

Code:

```
import csv
file_path = "/Users/maithilishinde/Desktop/dwm/hierarchical
clustering/newdata2.csv"
from itertools import combinations

def findFrequency(string, freqs):
    for freq_dict in freqs:
        if string in freq_dict:
```

```

        return freq_dict[string]
    return 0 # Return 0 if the string is not found in any frequency dictionary

def powerSet(string):
    powerset = set()
    for i in range(0, len(string) + 1):
        for element in combinations(string, i):
            powerset.add(' '.join(element))
    powerset.discard(' '.join(string))
    powerset = list(powerset)
    powerset.sort()
    return powerset

def association(keys, confidence, allFreqs):
    finalRules = []
    for key in keys:
        lhs = powerSet(key.split(' '))
        rhs = []
        elements = set(key) - {' '}
        for elem in lhs:
            toJoin = list(elements - set(elem) - {' '})
            toJoin.sort()
            rhs.append(' '.join(toJoin))
        for l, r in zip(lhs, rhs):
            rule = l + ' -> ' + r
            string = l + ' ' + r
            tempLst = string.split(' ')
            tempLst.sort()
            string = ' '.join(tempLst)
            freq_string = findFrequency(string, allFreqs)
            freq_l = findFrequency(l, allFreqs)
            if freq_l == 0:
                continue # Skip this rule if denominator is zero
            conf = (freq_string / freq_l) * 100
            if conf >= confidence:
                print(rule + " with a confidence of " + str(conf) + "%")

def combine(keys, keyLen):
    finalKeys = []
    for lst1 in keys:
        for lst2 in keys:
            tempLst = lst1[:]
            tempLst.extend(lst2)

```

```

        tempLst = list(set(tempLst))
        tempLst.sort()
        if len(tempLst) == keyLen + 1:
            if tempLst not in finalKeys:
                finalKeys.append(tempLst)
    return finalKeys

def filterComb(keys, data, support):
    finalKeys = []
    frequency = {}
    for key in keys:
        freq = 0
        for value in data.values():
            if all(item in value for item in key):
                freq += 1
        if freq >= support:
            frequency[' '.join(key)] = freq
    return frequency

items = []
itemSet = []
data = {}
previous = {}
support = int(input("Enter the minimum support (in percent): "))
confidence = int(input("Enter the minimum confidence (in percent): "))
frequency = {}
keyLen = 1
allFreqs = []
with open('newdata.csv', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    for row in csv_reader:
        data[row['total_sqft']] = row['bath']
        items.extend(data[row['total_sqft']])

itemSet = list(set(items))
itemSet.sort()
support = (support / 100) * len(data)
for item in itemSet:
    frequency[item] = items.count(item)
frequency = dict(filter(lambda elem: elem[1] >= support, frequency.items()))
keyList = []
for key in list(frequency.keys()):
    keyl = [key]
    keyList.append(keyl)
while True:

```

```

keyList = combine(keyList, keyLen)
allFreqs.append(frequency)
previous = frequency.copy()
frequency = filterComb(keyList, data, support)
keyList = []
for key in list(frequency.keys()):
    keyl = key.split()
    keyList.append(keyl)
keyLen += 1
if len(keyList) <= 1:
    break

print("The strong association rules are as follows: -")
if not frequency:
    association(previous, confidence, allFreqs)
else:
    allFreqs.append(frequency)
    association(frequency, confidence, allFreqs)

```

## Results:

```

Enter the minimum support (in percent): 30
Enter the minimum confidence (in percent):70

```

```

The strong association rules are as follows:
1056 -> 1440    with a confidence of 100.0%
1440 -> 1056 with a confidence of 95.65217391304348%

```

In the given output, the association rules represent relationships between two items: "1056" and "1440." The confidence values associated with these rules indicate the strength of the relationships.

### 1. "1056 -> 1440" with a confidence of 100.0%:

- This rule means that if you have item "1056," there's a 100% confidence that you will also have item "1440" in the same transaction. In other words, "1056" implies "1440," and this relationship is very strong.

### 2. "1440 -> 1056" with a confidence of 95.65217391304348%:

- This rule means that if you have item "1440," there's a 95.65% confidence that you will also have item "1056" in the same transaction. The high confidence value suggests a strong relationship between "1440" and "1056" in the dataset.

These association rules are often used in market basket analysis and can provide insights into which items are frequently purchased together. In the context of your dataset, it suggests a strong statistical relationship between the presence of "1056" and "1440" in transactions.



**SUBJECT : Data-Warehouse And Mining**

**AIM :** Implementation of Page rank/HITS algorithm

**THEORY :**

### **Hyperlink-Induced Topic Search (HITS) Algorithm**

HITS is an algorithm used in link analysis to discover and rank webpages relevant to a search. It utilizes the concept that a quality website should both link to other relevant sites and be linked by other important sites.

Key Concepts:

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

Algorithm Steps:

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

How HITS Works:

1. Retrieve the search query data.

2. Computation is performed considering the search results, excluding other websites.
3. Define authoritative and hub values.
4. Initiate an iterative process involving authority and hub updates.
5. Authoritative websites are those with valuable content.
6. HITS is executed at query time and considers the keyword or content people are searching for.
7. HITS computes two scores per document (authority and hub).
8. It is processed on a small subset of documents, not all documents like PageRank.
9. HITS is not commonly used by search engines.

This structured format provides a concise summary of the HITS algorithm, its key concepts, and how it differs from PageRank.

### Code and output:

```
import networkx as nx
import matplotlib.pyplot as plt

# Create a directed graph
graph = nx.DiGraph()

# Define relationships (customize this based on your dataset)
edges = [
    ('A', 'D'), ('B', 'C'), ('B', 'E'), ('C', 'A'),
    ('D', 'C'), ('E', 'D'), ('E', 'B'), ('E', 'F'),
    ('F', 'C'), ('F', 'H'), ('G', 'A'),
    ('I', 'C'), ('H', 'A')
]

graph.add_edges_from(edges)

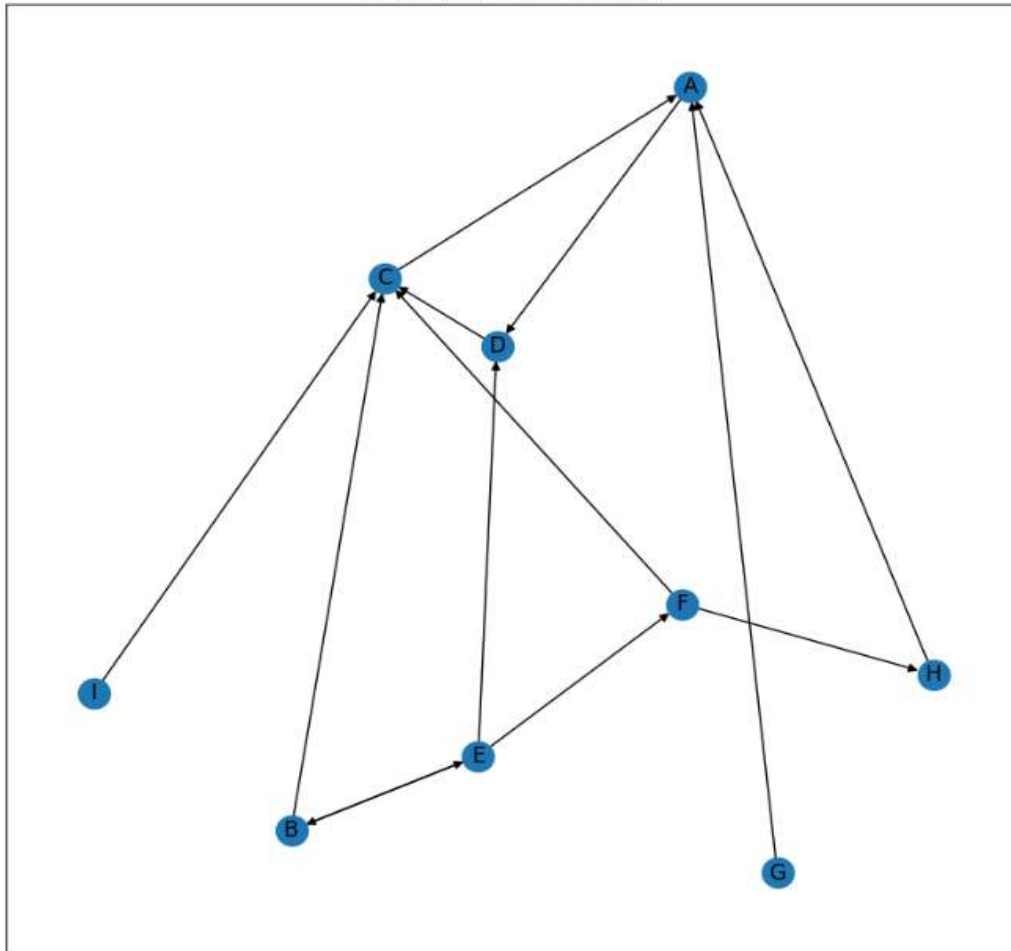
# Visualize the graph (optional)
plt.figure(figsize=(10, 10))
pos = nx.spring_layout(graph, seed=42)
nx.draw_networkx(graph, pos, with_labels=True)
plt.title("Property Relationship Graph")
plt.show()

# Calculate HITS scores
hubs, authorities = nx.hits(graph, max_iter=10, normalized=True)

# Print HITS scores
for node in graph.nodes:
    print(f"Node {node}:")
    print(f"Hub Score: {hubs[node]}")
    print(f"Authority Score: {authorities[node]}\n")
```



Property Relationship Graph



Node A:  
 Hub Score:  $-2.0458796793463883e-19$   
 Authority Score:  $0.0$

Node D:  
 Hub Score:  $0.21922359359558488$   
 Authority Score:  $-5.976351314820015e-19$

Node B:  
 Hub Score:  $0.28077640640441526$   
 Authority Score:  $-1.5299459365939238e-16$

Node C:  
 Hub Score:  $0.0$   
 Authority Score:  $0.6403882032022077$

Node E:  
 Hub Score:  $-1.311408874461035e-16$   
 Authority Score:  $0.17980589839889644$

Node F:  
 Hub Score:  $0.2807764064044152$   
 Authority Score:  $-2.294918904890886e-16$

Node H:  
 Hub Score:  $0.0$   
 Authority Score:  $0.17980589839889624$

Node G:  
 Hub Score:  $0.0$   
 Authority Score:  $1.5299459365939238e-16$

Node I:  
 Hub Score:  $0.21922359359558488$   
 Authority Score:  $-1.147459452445443e-16$

S

## DWM Assignment - 01

Q.) What is metadata? Explain data warehouse metadata with example

→ Metadata is simply defined as about data. The data that is used to represent other data is known as metadata.

Eg :- The index of book serves as metadata for content in book. In other words, we can say that metadata is summarized data that leads to detailed data.

- In terms of data warehouse, we create metadata for dataranges and definition of given data. It acts like a roadmap to the data warehouse and defines the warehouse objects. It also acts as a directory which helps in decision support system to locate the contents of data warehouse.

- Metadata can be categorized into 3 categories :-

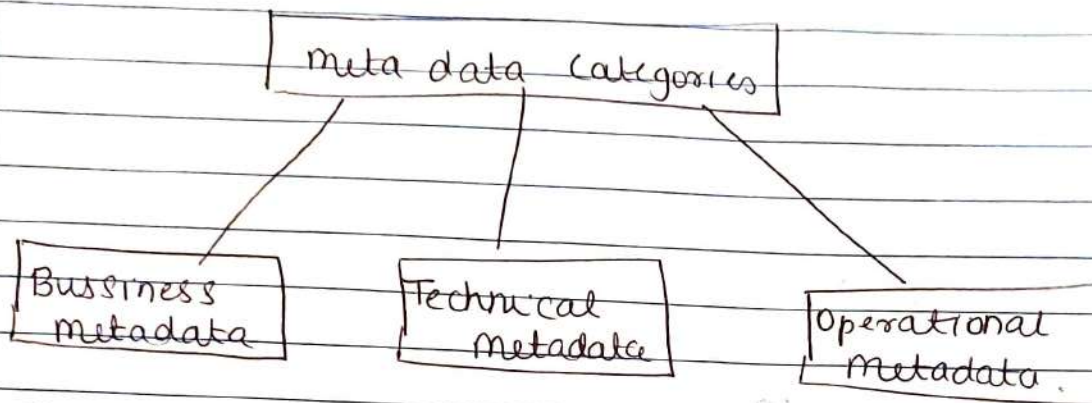
⇒ Business metadata :- It has data ownership information, business definition and changing policies.

⇒ Technical metadata :- It includes database system names, table and column names and size, data type and allowed values.

Technical metadata also includes structural info like primary, foreign key, attributes and indexes.



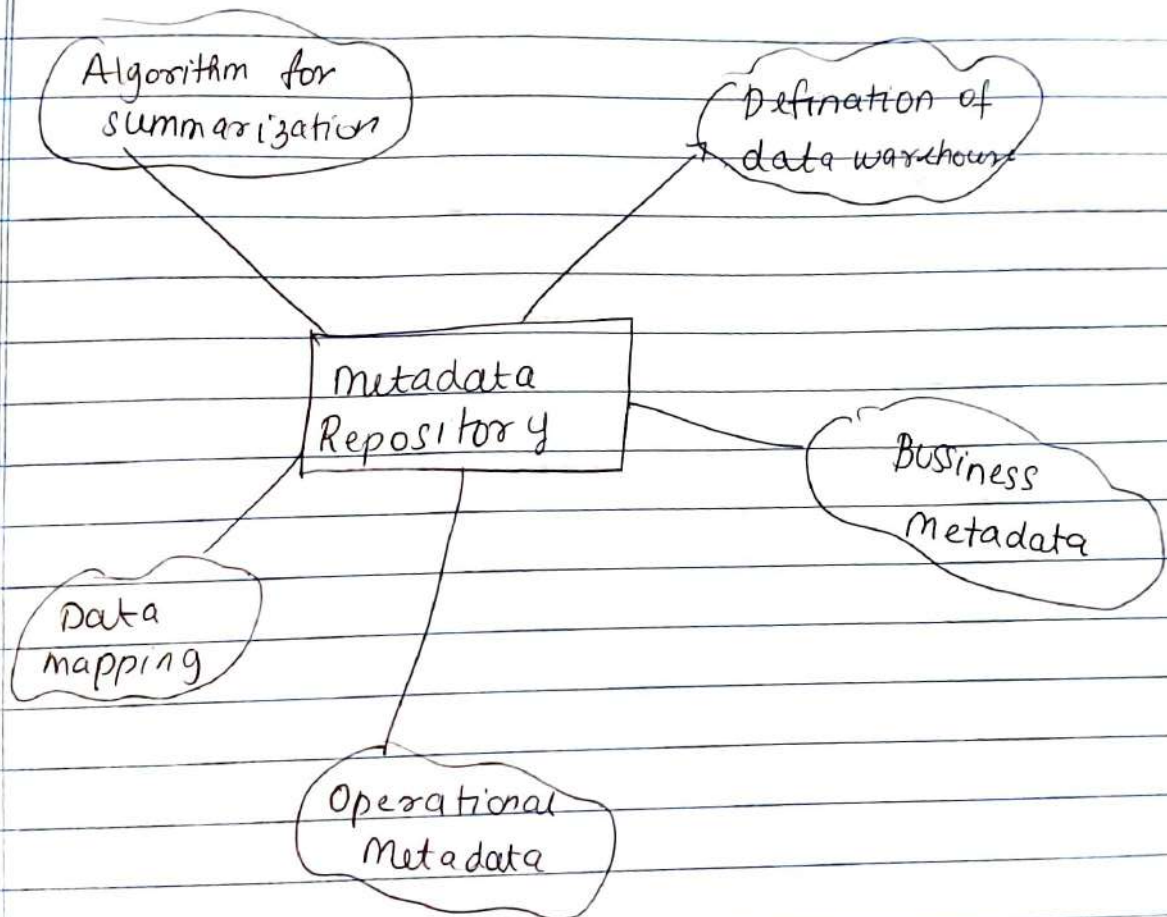
⇒ Operational Metadata ⇒ Includes currency of data & data lineage. Currency of data means whether data is active, archived or purged.



• Roles of Metadata: -

- It act as directory
- directory helps decision support system to locate content of data warehouse.
- metadata help in decision support system for mapping of data when data is transformed from operational environment to data warehouse environment.
- Meta data help in summarization between highly detailed data & highly summarized data.
- Meta data is used for query tools.
- Metadata is used in extraction and cleaning tools
- Meta data is used for transformation tools
- 'Plays an important role in loading functions'

Meta data Repository is integral part of datawarehouse system. It has following ~~dat~~ metadata



### Example

Meta data for a 'customer' entity :-

Defination :- A person / org purchasing goods or services from the company.

Remarks :- Customer entity includes regular, current and past customer



Source Systems :- Finished goods order, maintenance controls online calls.

Create Date : January 15, 2000

Last Update Date :- January 21, 2002

Update Cycle :- Weekly

Last full refresh data :- January 21, 2002

Planned ~~up dat~~ archival :- Six months.

Responsible user :- John Daddy

Last Depuplication : January 10, 2002

Data quality reviewed :- January 25, 2001

Full refresh cycle :- Every six months.

~~21/10/02~~

## ASSIGNMENT NO-2

Write short Notes on -

a.) Multilevel Association Rules & multidimensional allocation rules.

→ Multilevel association rules:-

Association rules created from mining information at different degree of reflection are called various level or staggered association rules.

These rules can be mined effectively utilizing idea progression under help certaining system.

Rules at a high level may add to good judgement while rules at low level may not be valuable consistently.

- At the point when uniform least help edge is utilised the system is rearranged.

- The technique is likewise straightforward, in that clients are needed to indicate just a client atleast help edge.

- A similar least help edge is utilized when mining at each degree of deliberation.

for example for mining from "PC" down to "PC". Both "PC" & "PC" discovered to be important, while "PC" not.

Needs of multilevel rule:-



- Sometimes at low data level, data does not show any significant pattern but there is useful interaction hiding behind it.
- The aim is to find hidden information in or between level of abstraction.

Approaches used in multi-level association rule mining

- Uniform support:** A point when a uniform least help edge is used the search methodology is simplified. The technique is likewise basic in that client are needed to determine just a single least help threshold. An advancement technique can be adopted, based on information that a progenitor is a superset of its descendant.
- Reduce support:** - It has other types like level by level independence, level cross separating by single thing, level cross separating by K-itemset, level by level independence, and other that use reduced minimum support at lower level.
- Group based support:** - The group wise threshold value for support & confidence is input by user or expert. The group is selected based on input or expert. The group is selected based on product item set because often expert has insight as to which group are important.



## Multidimensional association rule:

- In this type the quantities can be absolute or quantitative
- Quantitative characteristics are numeric & consolidate order.
- Numeric trait should be discretized.

### 2.) Using powerful discretization of quantitative traits.

- Known as mining Quantitative Association Rules
- Numeric properties are progressively discretized.

### 3.) Guide for tuples.

Using distance based discretization with bunching

- Perform bunching to discover the time period included

- Get affiliation rules via looking for pathways of groups that happen together.

### b.) Web usage mining: -

Web usage mining is used to derive useful data information knowledge from weblogs & help in identifying the users access logs for web pages.



In mining, the management of web resources. The individual is thinking about data of resources of visitors of a website that are composed as web server logs. While the content & mechanism of set of web pages follow intentions of authors of the pages, the single request show how the users view these pages. Web pages usage mining can disclose relationship that were not suggested by designer of webpages.

With available UPC, time IP address webpage content data a multidimensional view can be built on weblog database & multidimensional OLAP analysis can be implemented to discover the top users, top N accessed web server etc

Data mining can implemented on weblog records to discover association patterns, sequential pattern trend on web accessing

~~Qf~~  
24/1/23