SARASWATI Education Society's

SARASWATI College of Engineering

Learn Live Achieve and Contribute

Kharghar, Navi Mumbai - 410 210.

NAAC Accredited

Department of Computer Engineering

Name : Shaikh Adnan Shaukat Ali

Roll No : 68

Subject : Data Warehousing and Mining(DWM)

Class/Sem: T.E. / SEM-V

SARASWATI Education Society's
SARASWATI College of Engineering
Learn Live Achieve and Contribute
Kharghar, Navi Mumbai - 410 210.
NAAC Accredited

Department of Computer Engineering

Subject :Data Warehousing and Mining **Class/Sem: TE/V**

Name of the Laboratory: Data Warehousing and Mining Lab **Year: 2021-2022**

LIST OF EXPERIMENTS

Prof. Shatabdi Bhalerao Prof. Sujata Bhairnallykar

Subject In-charge **HOD**

Experiment No. 1

Aim: Case study on building Data warehouse/Data Mart.

Problem Statement: Case study on Electronic sales data warehouse and designing star and snow flake schema for it.

Theory:

What is an electronic sales data warehouse?

=>A data warehouse, in this context, is a cloud-based system for gathering, organizing, and storing information about your customers. The name has a brick-and-mortar feel, but it's a modern concept. "Warehouse" is an apt term. A data warehouse creates a single digital place for you to review your information. You can then use that warehouse to run analytics, reports, and measure what's going on throughout the entire company.

Few key benefits of data warehouse:

Faster time to insights

Data analysis always requires data gathering first. If you already have a system in place to collect and store all relevant data, you can run analysis whenever you like.

Reducing the silo effect

You may have incoming data from places like Shopify, Google Analytics, and Klaviyo. Data may still be sitting on paid advertising networks like Facebook, Google, and Taboola. The problem? You can't get a sense of the bigger picture, because that data is stuck within those systems.

Whenever you try to measure your data across multiple channels, things get messy. Your siloed data might be useful, but if you can only see a fraction of the big picture at once that data will by definition only be partly useful to you. A data warehouse reduces the pains of the silo effect and helps you visualize big-picture trends.

Full ownership of data

When you silo your data, you lack a single source of truth for business insights. Even worse: unless you warehouse your own data, you're at the mercy of the data retention policy of every platform you're on. If they decide to ditch your data and you don't already have it, you're out of luck.

With data warehousing, you can migrate that data into your own reference source. If you ever want to refer to it for predictive models (like for personalized product recommendations), all the historical data you need is ready and waiting.

What can you do with an electronic sales data warehouse?

So far, so good. Data warehouse management sounds great. But what do you do with the data? What kinds of returns should you expect on your investment? Let's explore the possibilities.

Attribution modelling

An attribution model means you "tag" your incoming revenue with its appropriate source. You set the rules here. You assign partial or full credit of a sale to individual touchpoints in your sales pipeline.

As a result, you'll have a clearer measurement of internal Return of investment (ROI). Who's making the sales? Which channels are providing the best results? For mostly offline brick-and-mortar retail, it's nearly impossible to pull this off. But in an ecommerce data warehouse environment, these insights are invaluable.

Predictive analytics

In electronic sales, your predictive analytics aren't just for guessing at next-quarter sales. They help you build real, practical product and content recommendations for your customer segments.

A 2015 Forrester study found that predictive "lead scoring" was one of the top use cases here. With lead scoring, you can leverage data to predict which leads are most likely to convert into customers. This creates immediate leverage in marketing: you know who to market to, where to put your money, and what kind of ROI to expect.

Or, take another example: Netflix. When the streaming company created "House of Cards," it wasn't throwing darts at the wall. It used predictive analytics via historical data to determine the kind of show customers had already demonstrated they wanted. Netflix then simply went about creating that show.

Customer segmentation

It's a simple fact of economics, as defined by the Pareto principle: a small portion of your customers are likely to have the greatest impact on your bottom line. Customer segmentation is all about identifying that impact and using it to your advantage.

Typically, customer segmentation has focused on traditional variables, like customer demographics. But an ecommerce data warehouse opens all sorts of possibilities. You can identify and differentiate customers by products purchased, how likely they are to open your emails, and their behavior upon a previous visit. Some ecommerce outlets even offer weather-specific recommendations based on geolocation.

Optimizing paid ads and marketing spend on campaigns

Once you have a more accurate view of your customer segments, you'll have more precise targets for your paid ads. And with your pipelines transferring data into your data warehouse, every new advertising campaign doubles as a fresh learning experience.

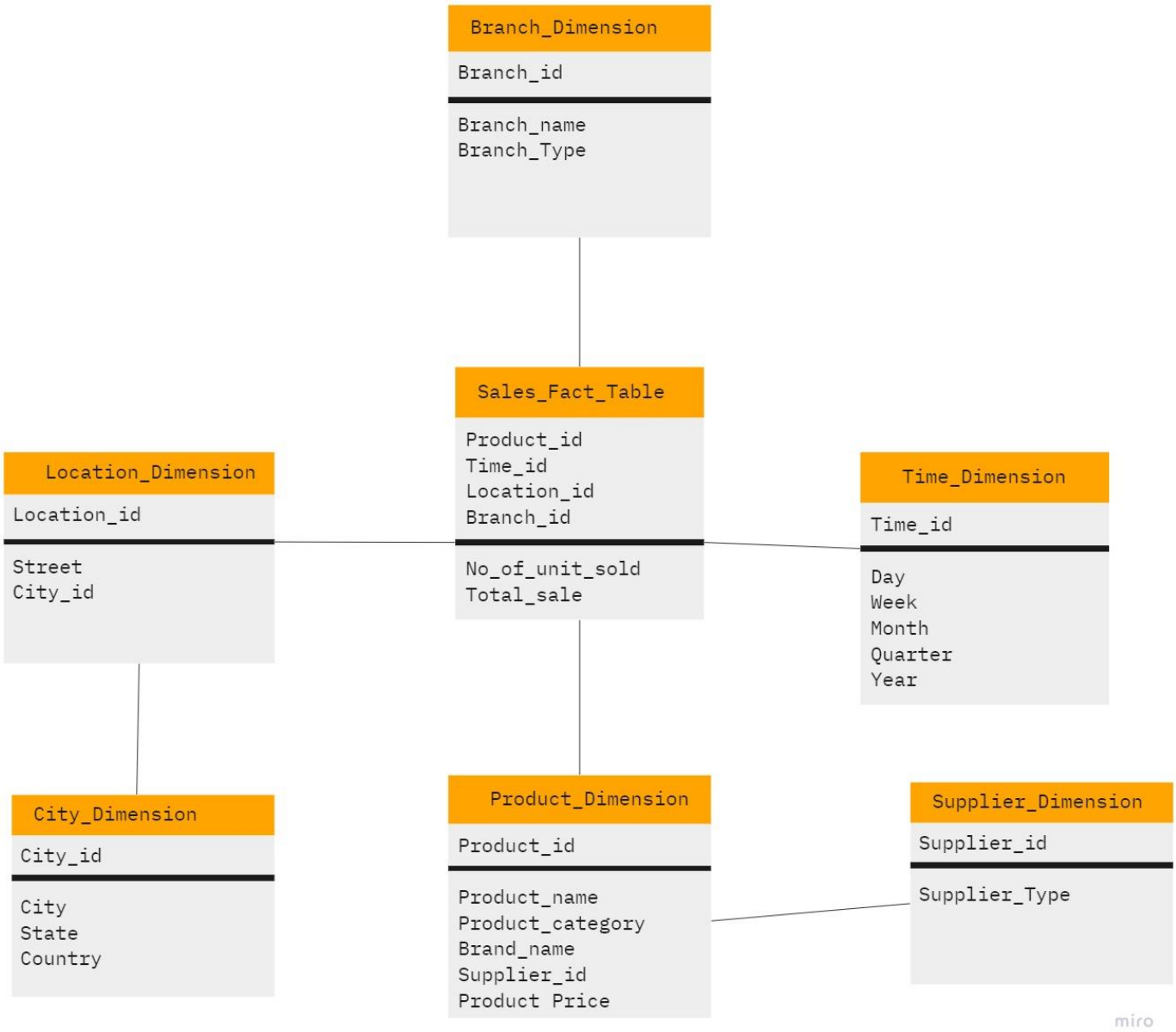
For example, A/B split testing lets you target different variables in your campaigns. This includes ad channel selection, high-level messaging, audience targeting, and even the specific copy you use in the ads. Properly channelled into your data warehouse, you'll have the results of every campaign ready for comparison. What works and what doesn't work? Now, you'll know.

Design Dimensional Modelling:

Star Schema: Star Schema in data warehouse, in which the centre of the star can have one fact table and a number of associated dimension tables. It is known as star schema as its structure resembles a star. The Star Schema data model is the simplest type of Data Warehouse schema. It is also known as Star Join Schema and is optimized for querying large data sets.

Star Schema of Electronic Sales

Snow Flake Schema: Snowflake Schema in data warehouse is a logical arrangement of tables in a multidimensional database such that the ER diagram resembles a snowflake shape. A Snowflake Schema is an extension of a Star Schema, and it adds additional dimensions. The dimension tables are normalized which splits data into additional tables.



Snow Flake Schema of Electronic Sales

Conclusion: We have successfully done case study on Electronic sales data warehouse and designed star and snow flake schema for it.

Experiment No. 2

Aim: To implement Dimension and Fact table of Electronic Sales.

Requirement: Windows/Linux/Mac OS, MYSQL/Oracle SQL database app.

Theory:

Dimension Table:

A Dimension Table is present in the star or snowflake schema. Dimension tables' help to describe dimensions i.e. dimension values, attributes and keys. It is generally small in size. Size can range from several to thousand rows. It describes the objects present in the fact table. Dimension Table refers to the collection or group of information related to any measurable event. They form a core for dimensional modelling. It contains a column that can be considered as a primary key column which helps to uniquely identify every dimension row or record. It is being joined with the fact tables through this key. When it is created a key called surrogate key that is system generated is used to uniquely identify the rows in the dimension.

Fact Table:

A fact table or a fact entity is a table or entity in a star or snowflake schema that stores measures that measure the business, such as sales, cost of goods, or profit.

Fact tables and entities aggregate measures, or the numerical data of a business. To measure data in a fact table or entity, all of the measures in a fact table or entity must be of the same grain.

To obtain the most useful data in a fact table or entity, you should use measures that are both numeric and additive. Using these measures guarantees that data can be retrieved and aggregated, so that the business can make use of the wealth of business data in the database.

Fact tables and entities also contain foreign keys to the dimension tables. These foreign keys relate each row of data in the fact table to its corresponding dimensions and levels.

Product Dimension Table:

```
mysql> CREATE TABLE product_dw(prod_id INTEGER AUTO_INCREMENT, prod_name VARCHAR(60) NOT NULL, prod_categroy VARCHAR(256) NOT NULL, Brand_na
OAT DEFAULT 0.0, PRIMARY KEY(prod_id));
Query OK, 0 rows affected (0.04 sec)

mysql> DESCRIBE product_dw;
+--------------+-------------+------+-----+---------+----------------+
| Field        | Type        | Null | Key | Default | Extra          |
+--------------+-------------+------+-----+---------+----------------+
| prod_id      | int         | NO   | PRI | NULL    | auto_increment |
| prod_name    | varchar(60) | NO   |     | NULL    |                |
| prod_categroy | varchar(256) | NO   |     | NULL    |                |
| Brand_name   | varchar(256) | NO   |     | NULL    |                |
| suppl_name   | varchar(256) | NO   |     | NULL    |                |
| prod_price   | float       | YES  |     | 0       |                |
+--------------+-------------+------+-----+---------+----------------+
6 rows in set (0.02 sec)
```

```
mysql> INSERT INTO product_dw(prod_name,prod_category,Brand_name,suppl_name,prod_price)
    -> VALUES
    -> ("Rice","Grocery","Dawat","Ramesh",140),
    -> ("Sugar","Grocery","Dawat","Ramesh",50),
    -> ("Kurta","Cloth","Max","Lila",500),
    -> ("Jacket","Cloth","Max","Lila",700);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from product_dw;
+---------+-----------+---------------+------------+------------+------------+
| prod_id | prod_name | prod_category | Brand_name | suppl_name | prod_price |
+---------+-----------+---------------+------------+------------+------------+
|       1 | Rice      | Grocery       | Dawat      | Ramesh     |        140 |
|       2 | Sugar     | Grocery       | Dawat      | Ramesh     |         50 |
|       3 | Kurta     | Cloth         | Max        | Lila       |        500 |
|       4 | Jacket    | Cloth         | Max        | Lila       |        700 |
+---------+-----------+---------------+------------+------------+------------+
4 rows in set (0.01 sec)
```

Location Dimension Table:

```
mysql> CREATE TABLE location_dw(loc_id INTEGER AUTO_INCREMENT,street VARCHAR(60) NOT NULL,city VARCHAR(25
Query OK, 0 rows affected (0.03 sec)

mysql> DESCRIBE location_dw;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| loc_id  | int         | NO   | PRI | NULL    | auto_increment |
| street  | varchar(60) | NO   |     | NULL    |                |
| city    | varchar(256)| NO   |     | NULL    |                |
| state   | varchar(256)| NO   |     | NULL    |                |
| country | varchar(256)| NO   |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

```
mysql> INSERT INTO location_dw(loc_id,street,city,state,country)
    -> VALUES
    -> (201,"ML ROAD","MUMBAI","MAHARASHTRA","INDIA"),
    -> (202,"AI ROAD","MUMBAI","MAHARASHTRA","INDIA"),
    -> (203,"BI ROAD","KOLKATA","WEST BENGAL","INDIA"),
    -> (204,"DB ROAD","KOLKATA","WEST BENGAL","INDIA");
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM time_dw;
+---------+------------+----------+----+------+
| time_id | day        | month    | qt | yr   |
+---------+------------+----------+----+------+
|     101 | 2021-01-17 | January  | Q1 | 2021 |
|     102 | 2021-02-14 | February | Q1 | 2021 |
|     103 | 2021-05-21 | May      | Q2 | 2021 |
|     104 | 2021-06-26 | June     | Q2 | 2021 |
+---------+------------+----------+----+------+
4 rows in set (0.00 sec)
```

Time Dimension Table:

```
mysql> CREATE TABLE time_dw(time_id INTEGER AUTO_INCREMENT,day DATE NOT NULL,month VARCHAR(256) NOT NULL,qt VAR
Query OK, 0 rows affected (0.07 sec)

mysql> DESCRIBE time_dw;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| time_id | int         | NO   | PRI | NULL    | auto_increment |
| day     | date        | NO   |     | NULL    |                |
| month   | varchar(256) | NO   |     | NULL    |                |
| qt      | varchar(256) | NO   |     | NULL    |                |
| yr      | varchar(256) | NO   |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

```
mysql> INSERT INTO time_dw(time_id,day,month,qt,y
    -> VALUES
    -> (101,"2021-1-17","January","Q1","2021"),
    -> (102,"2021-2-14","February","Q1","2021"),
    -> (103,"2021-5-21","May","Q2","2021"),
    -> (104,"2021-6-26","June","Q2","2021");
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM time_dw;
+---------+------------+----------+----+------+
| time_id | day        | month    | qt | yr   |
+---------+------------+----------+----+------+
|     101 | 2021-01-17 | January  | Q1 | 2021 |
|     102 | 2021-02-14 | February | Q1 | 2021 |
|     103 | 2021-05-21 | May      | Q2 | 2021 |
|     104 | 2021-06-26 | June     | Q2 | 2021 |
+---------+------------+----------+----+------+
4 rows in set (0.01 sec)
```

Sales Fact Table:

```
mysql> CREATE TABLE factsales(prod_id INTEGER,time_id INTEGER,loc_id INTEGER,no_of_unit_sold
d) REFERENCES time_dw(time_id),FOREIGN KEY(loc_id) REFERENCES location_dw(loc_id));
Query OK, 0 rows affected (0.06 sec)

mysql> DESCRIBE factsales;
+----------------+------+------+-----+---------+-------+
| Field          | Type | Null | Key | Default | Extra |
+----------------+------+------+-----+---------+-------+
| prod_id        | int  | YES  | MUL | NULL    |       |
| time_id        | int  | YES  | MUL | NULL    |       |
| loc_id         | int  | YES  | MUL | NULL    |       |
| no_of_unit_sold | int  | NO   |     | NULL    |       |
| total_sales    | int  | NO   |     | NULL    |       |
+----------------+------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

```
mysql> INSERT INTO factsales(prod_id,time_id,loc_id,no_of_unit_sold,total_sales)
    -> VALUES
    -> (1,101,201,400,80000),
    -> (1,102,201,400,90000),
    -> (1,103,201,400,70000),
    -> (1,104,201,400,90000),
    -> (1,101,202,400,90000),
    -> (1,102,202,400,90000),
    -> (1,103,202,400,90000),
    -> (1,104,202,400,90000),
    -> (1,101,203,400,90000),
    -> (1,102,203,400,90000),
    -> (1,103,203,400,90000),
    -> (1,104,203,400,90000);
Query OK, 12 rows affected (0.02 sec)
Records: 12  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM factsales;
+---------+---------+--------+-----------------+-------------+
| prod_id | time_id | loc_id | no_of_unit_sold | total_sales |
+---------+---------+--------+-----------------+-------------+
|       1 |     101 |    201 |             400 |       80000 |
|       1 |     102 |    201 |             400 |       90000 |
|       1 |     103 |    201 |             400 |       70000 |
|       1 |     104 |    201 |             400 |       90000 |
|       1 |     101 |    202 |             400 |       90000 |
|       1 |     102 |    202 |             400 |       90000 |
|       1 |     103 |    202 |             400 |       90000 |
|       1 |     104 |    202 |             400 |       90000 |
|       1 |     101 |    203 |             400 |       90000 |
|       1 |     102 |    203 |             400 |       90000 |
|       1 |     103 |    203 |             400 |       90000 |
|       1 |     104 |    203 |             400 |       90000 |
+---------+---------+--------+-----------------+-------------+
12 rows in set (0.00 sec)
```

Conclusion: We have successfully implemented Electronic Sales Dimension and Fact Table using MySQL.

Experiment No. 3

Aim: To implement OLAP operations: Slice, Dice, Rollup, Drilldown and Pivot.

Requirements: Windows/MAC/Linux O.S and MYSQL/Oracle SQL.

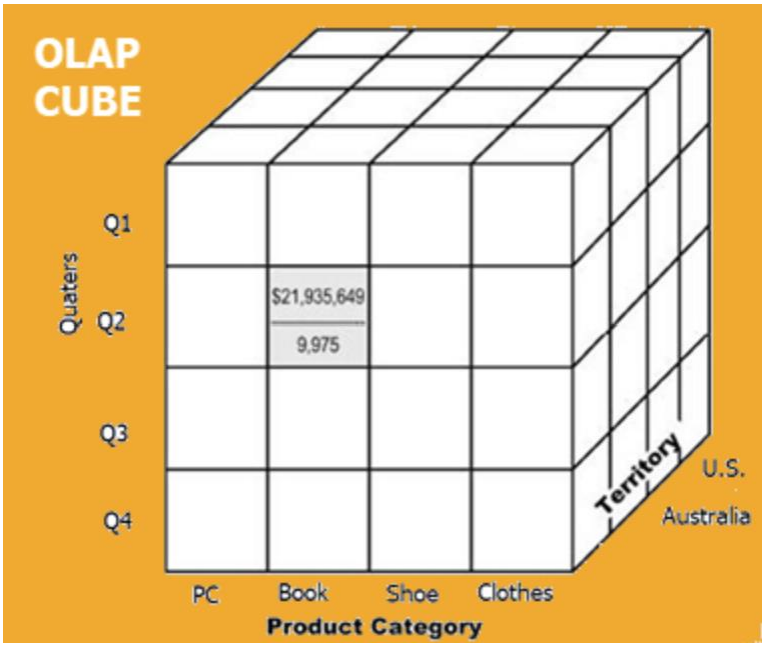
Problem statement: To implement OLAP operations: Slice, Dice, Rollup, Drilldown and Pivot on Electronic sales data set created in experiment 2.

Theory:

Online Analytical Processing (OLAP) is a category of software that allows users to analyse information from multiple database systems at the same time. It is a technology that enables analysts to extract and view business data from different points of view.

Analysts frequently need to group, aggregate and join data. These OLAP operations in data mining are resource intensive. With OLAP data can be pre-calculated and pre-aggregated, making analysis faster.

OLAP cube:



OLAP Cube

At the core of the OLAP concept, is an OLAP Cube. The OLAP cube is a data structure optimized for very quick data analysis.

The OLAP Cube consists of numeric facts called measures which are categorized by dimensions. OLAP Cube is also called the **hypercube**.

Basic analytical operations of OLAP

Four types of analytical OLAP operations are:

1. Roll-up
2. Drill-down

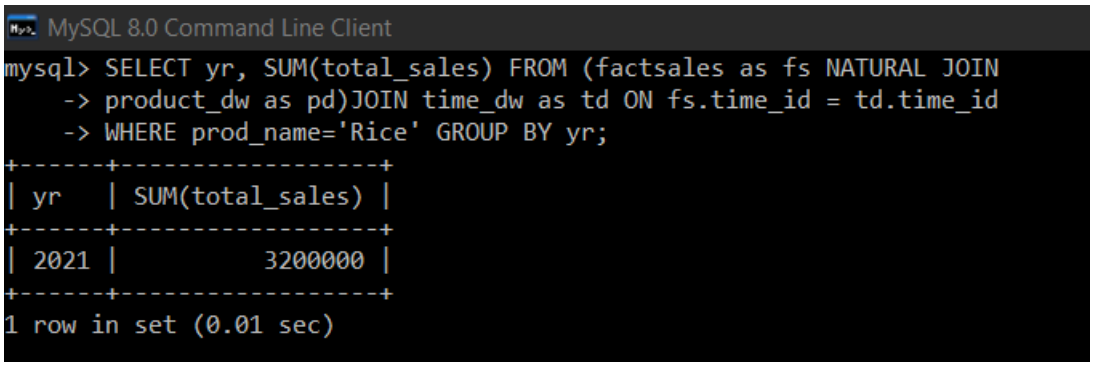
3. Slice and dice
4. Pivot (rotate)

1) Roll-up:

Roll-up is also known as “consolidation” or “aggregation.” The Roll-up operation can be performed in 2 ways

1. Reducing dimensions
2. Climbing up concept hierarchy. Concept hierarchy is a system of grouping things based on their order or level.

Roll-up on factsales, product_dw and time_dw:

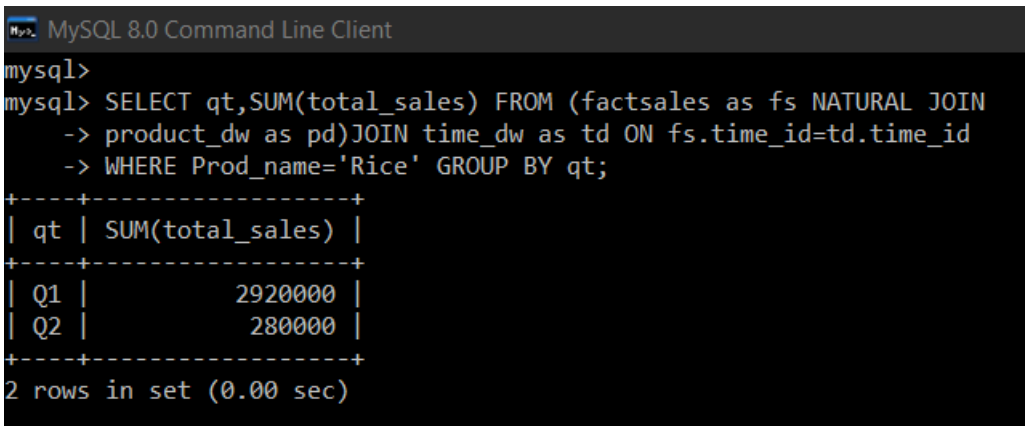


2) Drill-down

In drill-down data is fragmented into smaller parts. It is the opposite of the rollup process. It can be done via

- Moving down the concept hierarchy
- Increasing a dimension

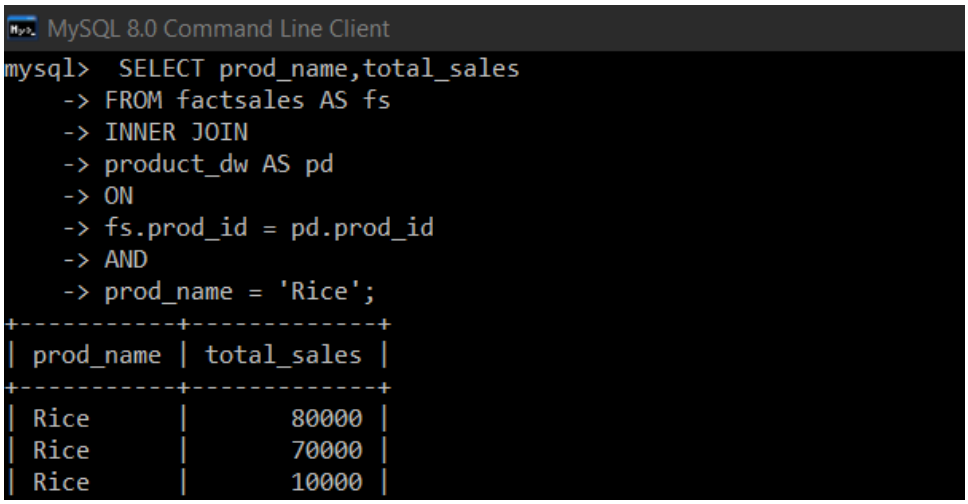
Drill-down on factsales, product_dw and time_dw:

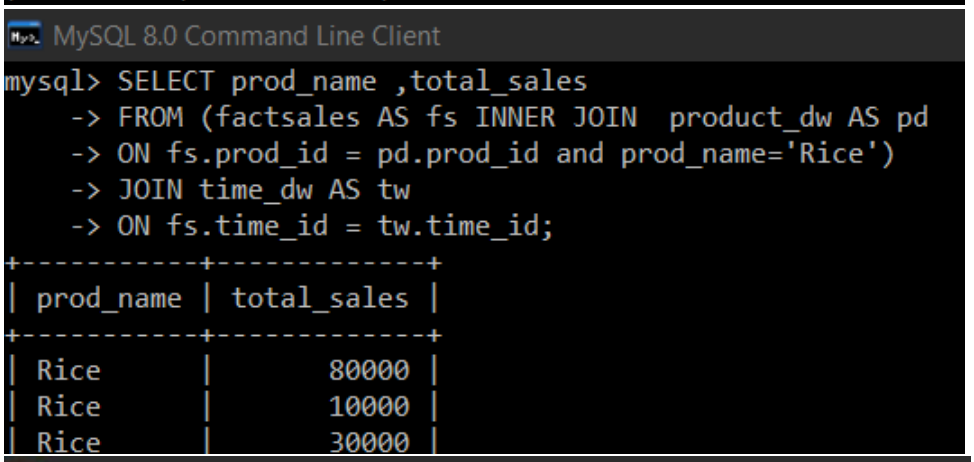


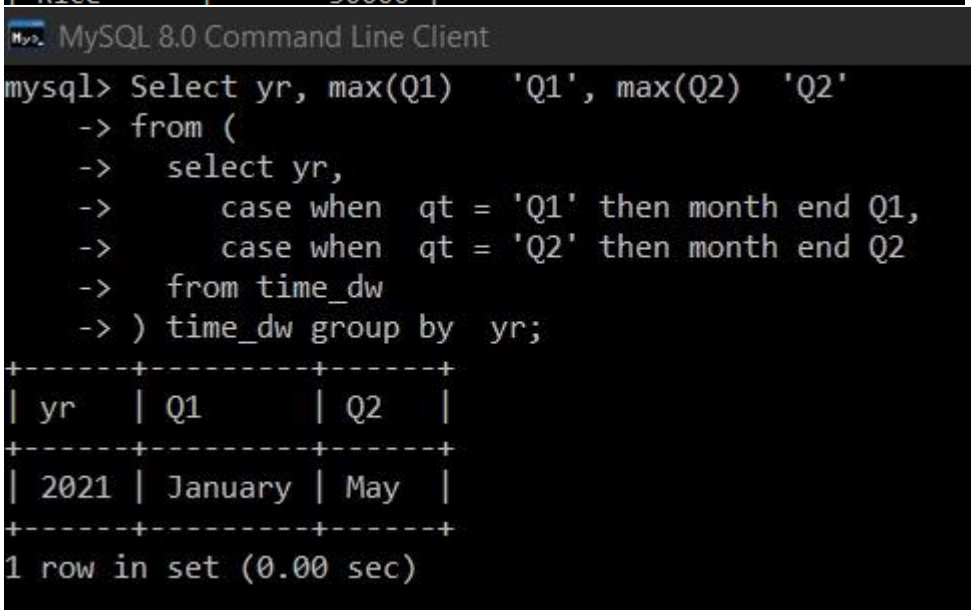
3) Slice:

Here, one dimension is selected, and a new sub-cube is created.

Slice on factsales, product_dw and time_dw:







4) Pivot

In Pivot, you rotate the data axes to provide a substitute presentation of data.

Pivot on time_dw:

```
mysql> SELECT yr,
    ->  MAX(IF(qt = "Q1",  month,NULL)) AS "Q1",
    ->  MAX(IF(qt = "Q2",  month,NULL)) AS "Q2"
    -> FROM time_dw group by yr;
+------+---------+------+
| yr   | Q1      | Q2   |
+------+---------+------+
| 2021 | January | May  |
+------+---------+------+
1 row in set (0.00 sec)

mysql>
mysql>
mysql> SELECT yr,
    ->  MAX(IF(qt = "Q1",  month,NULL)) AS "Q1",
    ->  MAX(IF(qt = "Q2",  month,NULL)) AS "Q2"
    -> FROM time_dw;
+------+---------+------+
| yr   | Q1      | Q2   |
+------+---------+------+
| 2021 | January | May  |
+------+---------+------+
1 row in set (0.00 sec)
```

Conclusion: We have successfully implemented OLAP operations on Electronic sales data set using MYSQL.

Experiment No. 4

Aim: To implement Bayesian algorithm.

Requirements: Windows O.S, Weka tool, Python and Python libraries: Pandas, Numpy, Sklearn and Matplot.

Problem Statement: To implement Naïve Bayes algorithm on iris data set using Weka tool and Python.

Theory:

Naive Bayes: Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x1 through xn, :

$$P(y|x1,\ldots,xn) = P(y)P(x1,\ldots,xn|y)/P(x1,\ldots,xn)$$

Using the naive conditional independence assumption that

$$P(xi|y,x1,\ldots,xi-1,xi+1,\ldots,xn) = P(xi|y),$$

for all i, this relationship is simplified to

$$P(y|x1,\ldots,xn) = P(y)\prod i=1nP(xi|y)/P(x1,\ldots,xn)$$

Since P(x1,…,xn) is constant given the input, we can use the following classification rule:

$$P(y|x1,\ldots,xn) \propto P(y)\prod i=1nP(xi|y)\Downarrow$$

$$y\hat{} = \text{argmax}(y)(P(y)\prod i=1nP(xi|y)),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate P(y) and P(xi|y); the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi|y).

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

Gaussian Naive Bayes: When working with continuous data, an assumption often taken is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. The likelihood of the features is assumed to be-

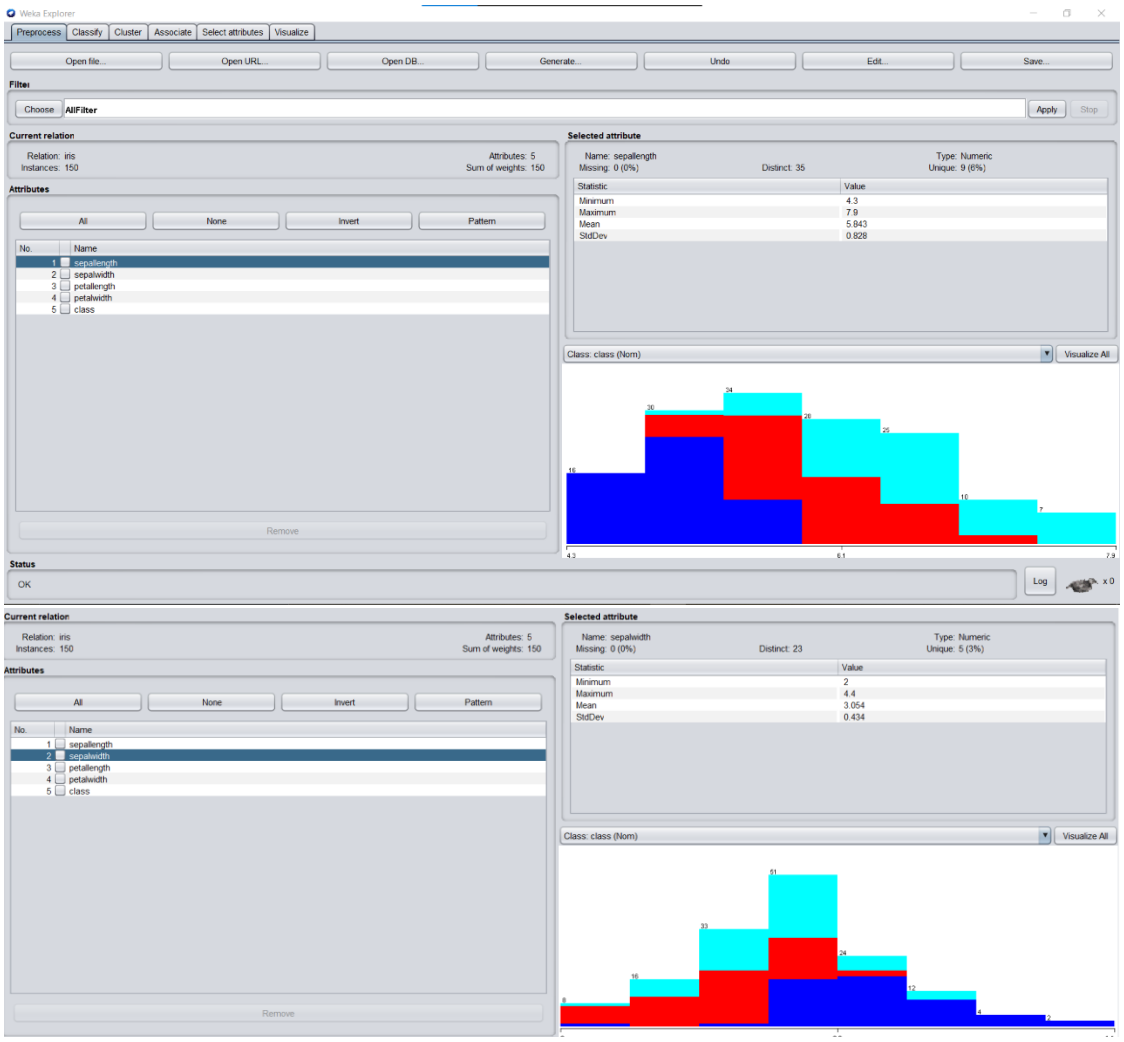
$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

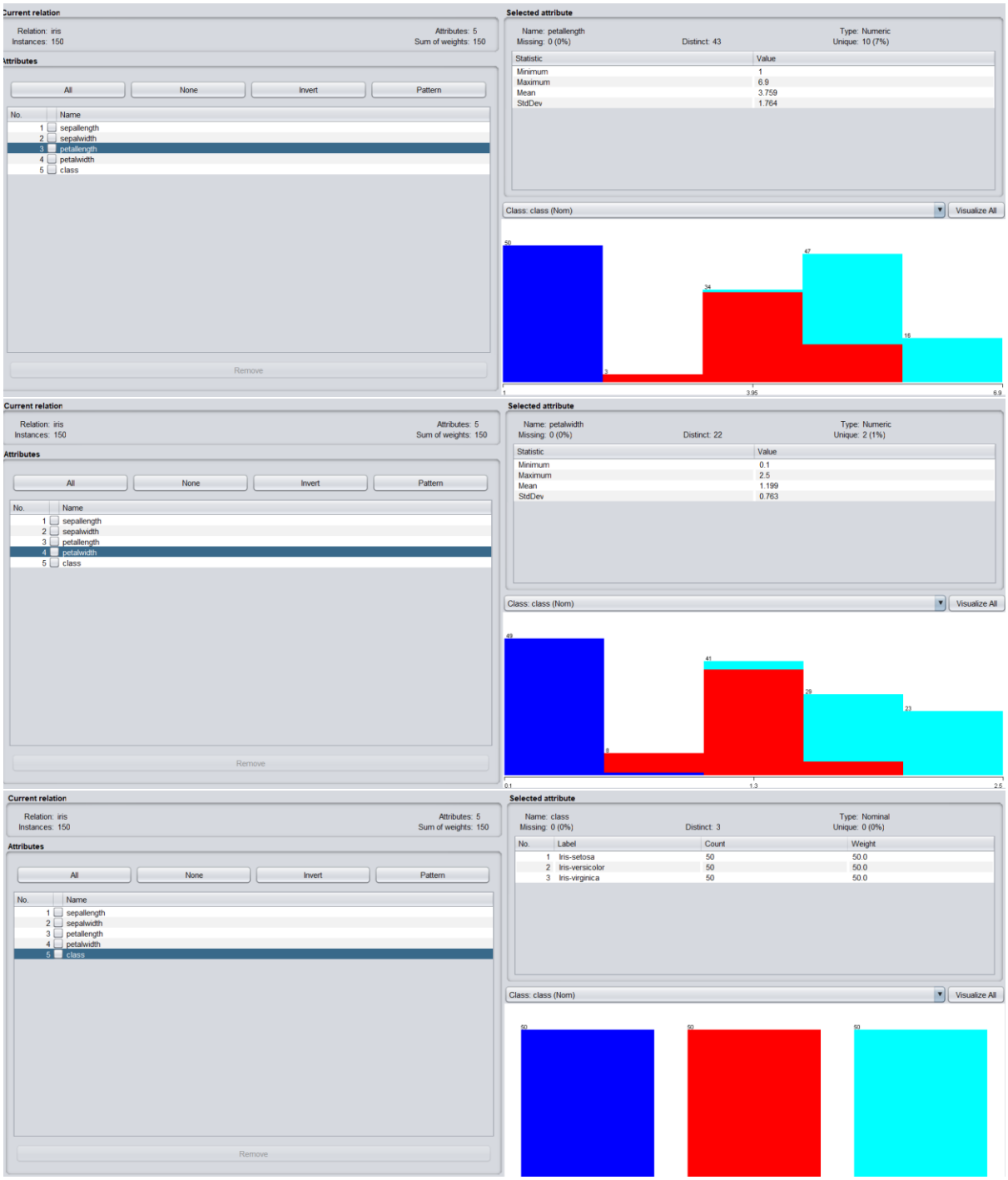
Sometimes assume variance

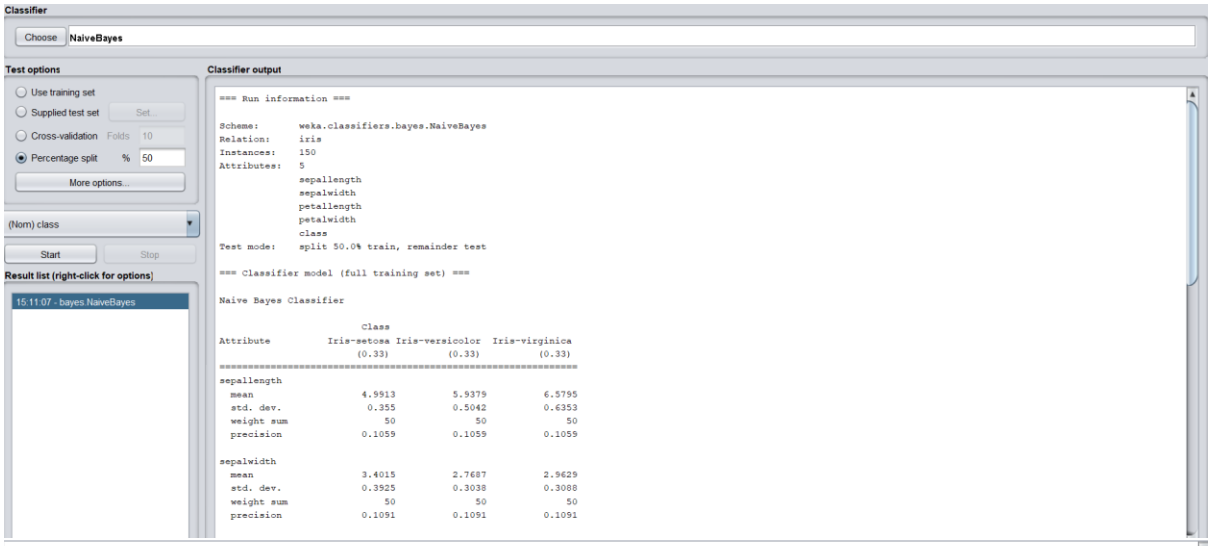
- is independent of Y (i.e., σi),
- or independent of Xi (i.e., σk)
- or both (i.e., σ)

Gaussian Naive Bayes supports continuous valued features and models each as conforming to a Gaussian (normal) distribution.

Weka tool output:




Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150
Attributes
All
None
Invert
Pattern
No.
Name
1 sepallength
2 sepalwidth
3 petallength
4 petalwidth
5 class
Remove
Selected attribute
Name: petallength
Missing: 0 (0%)
Distinct: 43
Type: Numeric
Unique: 10 (7%)
Statistic
Value
Minimum 1
Maximum 6.9
Mean 3.759
StdDev 1.764
Class: class (Nom)
Visualize All
50
3
24
47
18
1
3.95
6.9
Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150
Attributes
All
None
Invert
Pattern
No.
Name
1 sepallength
2 sepalwidth
3 petallength
4 petalwidth
5 class
Remove
Selected attribute
Name: petalwidth
Missing: 0 (0%)
Distinct: 22
Type: Numeric
Unique: 2 (1%)
Statistic
Value
Minimum 0.1
Maximum 2.5
Mean 1.199
StdDev 0.763
Class: class (Nom)
Visualize All
49
8
41
29
23
0.1
1.3
2.5
Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150
Attributes
All
None
Invert
Pattern
No.
Name
1 sepallength
2 sepalwidth
3 petallength
4 petalwidth
5 class
Remove
Selected attribute
Name: class
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 Iris-setosa 50 50.0
2 Iris-versicolor 50 50.0
3 Iris-virginica 50 50.0
Class: class (Nom)
Visualize All
50
50
50




```
=== Run information ===

Scheme:       weka.classifiers.bayes.NaiveBayes
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    split 50.0% train, remainder test

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

Naive Bayes Classifier

                          Class
Attribute          Iris-setosa Iris-versicolor  Iris-virginica
                        (0.33)          (0.33)          (0.33)
============================================================
sepallength
  mean                 4.9913          5.9379          6.5795
  std. dev.             0.355          0.5042          0.6353
  weight sum               50              50              50
  precision            0.1059          0.1059          0.1059

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

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

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



Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances          72               96      %
Incorrectly Classified Instances         3                4      %
Kappa statistic                          0.9398
Mean absolute error                      0.0336
Root mean squared error                  0.1606
Relative absolute error                  7.5238 %
Root relative squared error             33.8795 %
Total Number of Instances               75

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
                 1.000    0.061    0.897      1.000    0.945      0.917    0.987     0.975     Iris-versicolor
                 0.889    0.000    1.000      0.889    0.941      0.915    0.988     0.982     Iris-virginica
Weighted Avg.    0.960    0.021    0.964      0.960    0.960      0.941    0.991     0.985

=== Confusion Matrix ===

  a  b  c   <-- classified as
 22  0  0 |  a = Iris-setosa
  0 26  0 |  b = Iris-versicolor
  0  3 24 |  c = Iris-virginica
```

Python output:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
```

In [2]:
```
data = load_iris()
X,Y = load_iris(return_X_y=True)
```

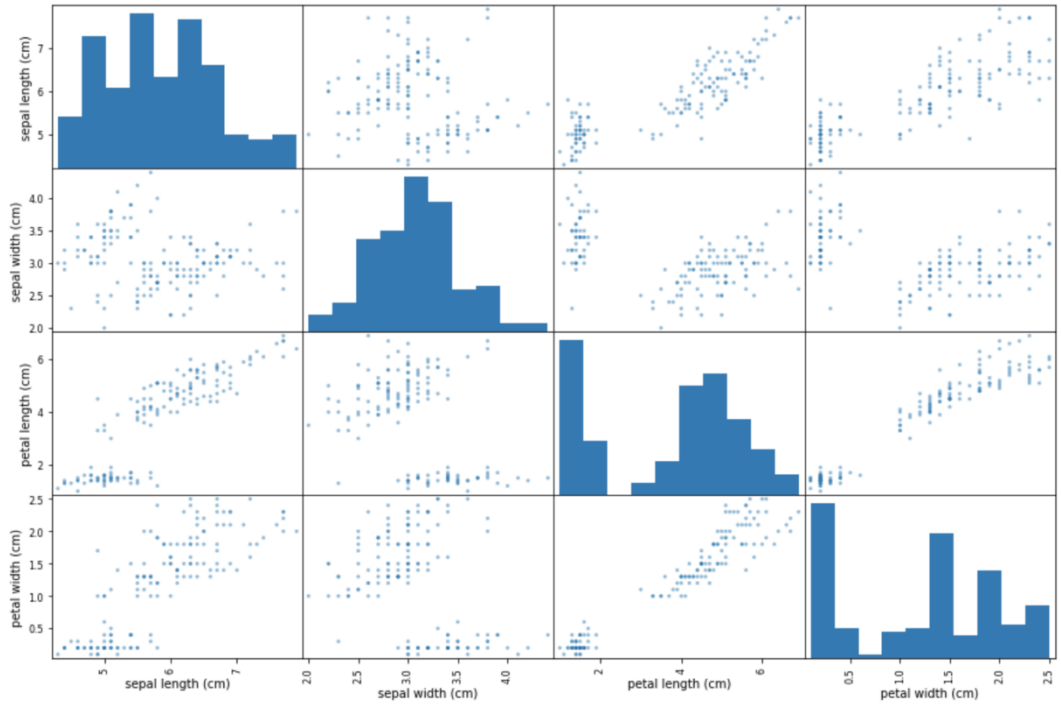
In [3]:
```
iris = pd.DataFrame(data = np.column_stack((data.data,data.target)),columns = [*data.feature_names,"Class"])
iris.Class.replace([0,1,2],['setosa', 'versicolor', 'virginica'],inplace = True)
iris
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Class
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [4]:
```
pd.plotting.scatter_matrix(iris,figsize=(15,10),diagonal="hist")
plt.show()
```



In [6]:
```
iris.groupby("Class").describe()
```

Out[6]:

	sepal length (cm)								sepal width (cm)		...	petal length (cm)		petal width (cm)						
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%	max
Class																					
setosa	50.0	5.006	0.352490	4.3	4.800	5.0	5.2	5.8	50.0	3.428	...	1.575	1.9	50.0	0.246	0.105386	0.1	0.2	0.2	0.3	0.6
versicolor	50.0	5.936	0.516171	4.9	5.600	5.9	6.3	7.0	50.0	2.770	...	4.600	5.1	50.0	1.326	0.197753	1.0	1.2	1.3	1.5	1.8
virginica	50.0	6.588	0.635880	4.9	6.225	6.5	6.9	7.9	50.0	2.974	...	5.875	6.9	50.0	2.026	0.274650	1.4	1.8	2.0	2.3	2.5

3 rows × 32 columns

In [7]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,train_size=0.5,random_state=0)
```

In [8]:
```
nav_bay = GaussianNB()
model = nav_bay.fit(X_train,Y_train)
y_pred = model.predict(X_test)
```

In [9]:
```
prediction = pd.DataFrame(np.column_stack((X_test,Y_test,y_pred)),
                          columns = [*data.feature_names,"Original_Class","Predicted_Class"])
prediction.Original_Class.replace([0,1,2],['setosa', 'versicolor', 'virginica'],inplace = True)
prediction.Predicted_Class.replace([0,1,2],['setosa', 'versicolor', 'virginica'],inplace = True)
prediction
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Original_Class	Predicted_Class
0	5.8	2.8	5.1	2.4	virginica	virginica
1	6.0	2.2	4.0	1.0	versicolor	versicolor
2	5.5	4.2	1.4	0.2	setosa	setosa
3	7.3	2.9	6.3	1.8	virginica	virginica
4	5.0	3.4	1.5	0.2	setosa	setosa
...	...	...	...	...	...	...
70	6.4	2.7	5.3	1.9	virginica	virginica
71	5.7	3.0	4.2	1.2	versicolor	versicolor
72	5.4	3.4	1.7	0.2	setosa	setosa
73	5.7	4.4	1.5	0.4	setosa	setosa
74	6.9	3.1	4.9	1.5	versicolor	versicolor

In [10]:
```
accuracy = prediction.loc[prediction["Original_Class"] == prediction["Predicted_Class"]].count()[0]/prediction.count()[0]
print(f"Accuracy of model = {accuracy*100}%")
```

```
Accuracy of model = 94.66666666666667%
```

In [12]:
```
conf_mat = pd.DataFrame(confusion_matrix(prediction["Original_Class"],prediction["Predicted_Class"]),
                        index = ['setosa', 'versicolor', 'virginica'],columns = ['setosa', 'versicolor', 'virginica'])
conf_mat.index.name = "Original Class"
conf_mat
```

Out[12]:

Original Class	setosa	versicolor	virginica
setosa	21	0	0
versicolor	0	30	0
virginica	0	4	20

Conclusion: We have successfully implemented Bayes Algorithm on iris data set using Weka tool and Python libraries.

EXPERIMENT NO. 5

Aim: To implement Data Discretization and Visualization.

Requirement: Windows OS and Weka Tool.

Theory:

Data Discretization: Data discretization refers to a method of converting a huge number of data values into smaller ones so that the evaluation and management of data become easy. In other words, data discretization is a method of converting attributes values of continuous data into a finite set of intervals with minimum data loss. There are two forms of data discretization first is supervised discretization, and the second is unsupervised discretization. Supervised discretization refers to a method in which the class data is used. Unsupervised discretization refers to a method depending upon the way which operation proceeds. It means it works on the top-down splitting strategy and bottom-up merging strategy.

Now, we can understand this concept with the help of an example

Suppose we have an attribute of Age with the given values

Age	1,5,9,4,7,11,14,17,13,18, 19,31,33,36,42,44,46,70,74,78,77

Table before Discretization

Attribute	Age	Age	Age	Age
	1,5,4,9,7	11,14,17,13,18,19	31,33,36,42,44,46	70,74,77,78
After Discretization	Child	Young	Mature	Old

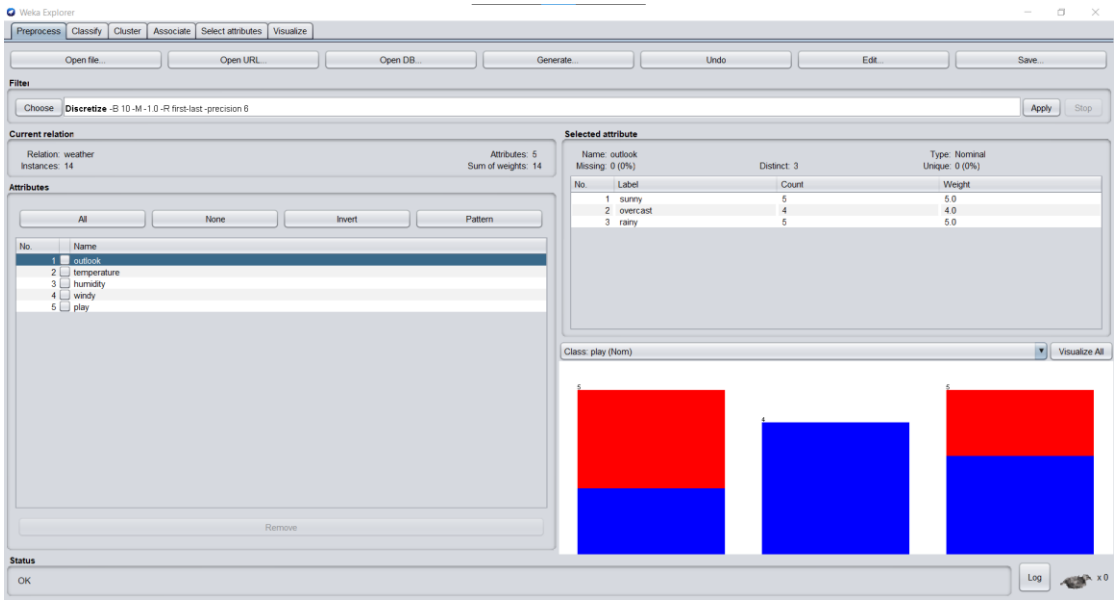
Table after Discretization

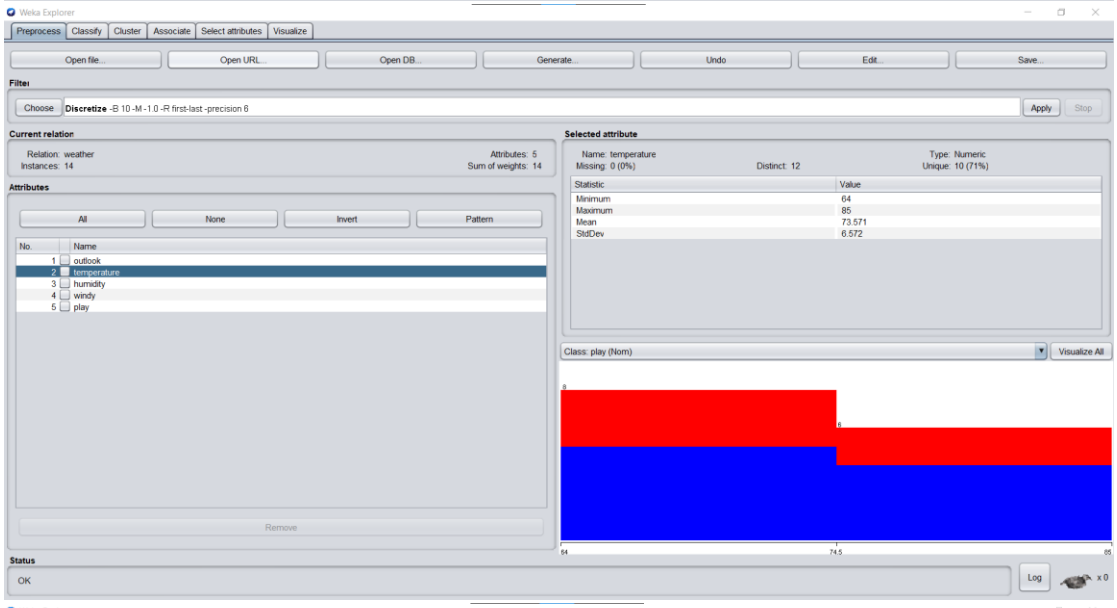
Data Visualization: Data visualization is the process of translating large data sets and metrics into charts, graphs and other visuals. The resulting visual representation of data makes it easier to identify and share real-time trends, outliers, and new insights about the information represented in the data.

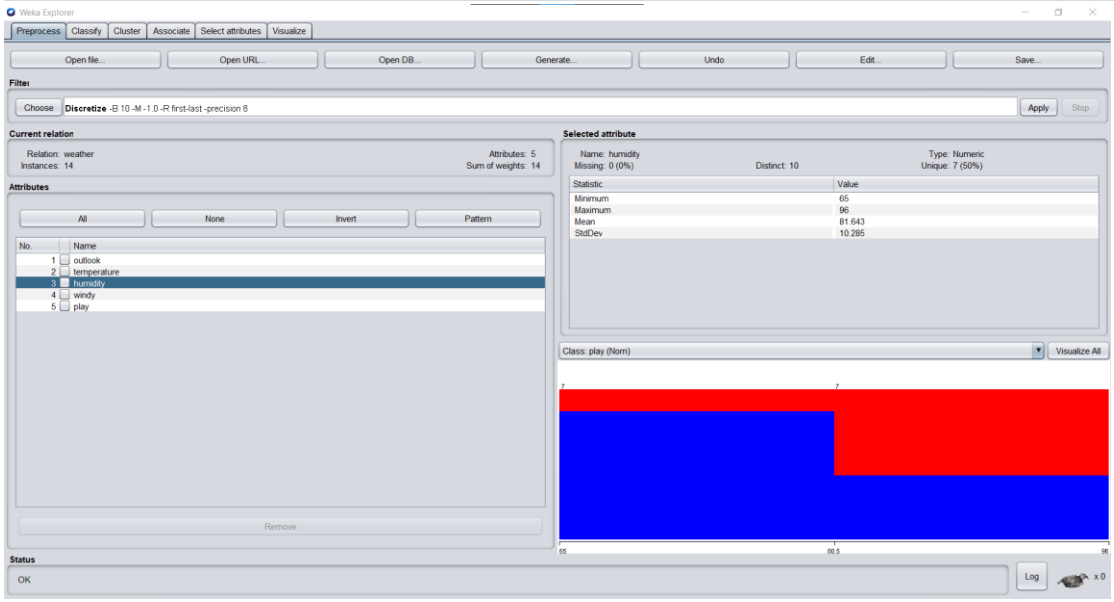
A dashboard is an information visualization tool. It helps you monitor events or activities at a glance by providing insights on one or more pages or screens. Unlike an infographic, which presents a static graphical representation, a dashboard conveys real-time information by pulling complex data points directly from large data sets. An interactive dashboard makes it easy to sort, filter, or drill into different types of data as needed. Data science techniques can be used to identify what is happening, why it's happening, and what will happen next at speed.

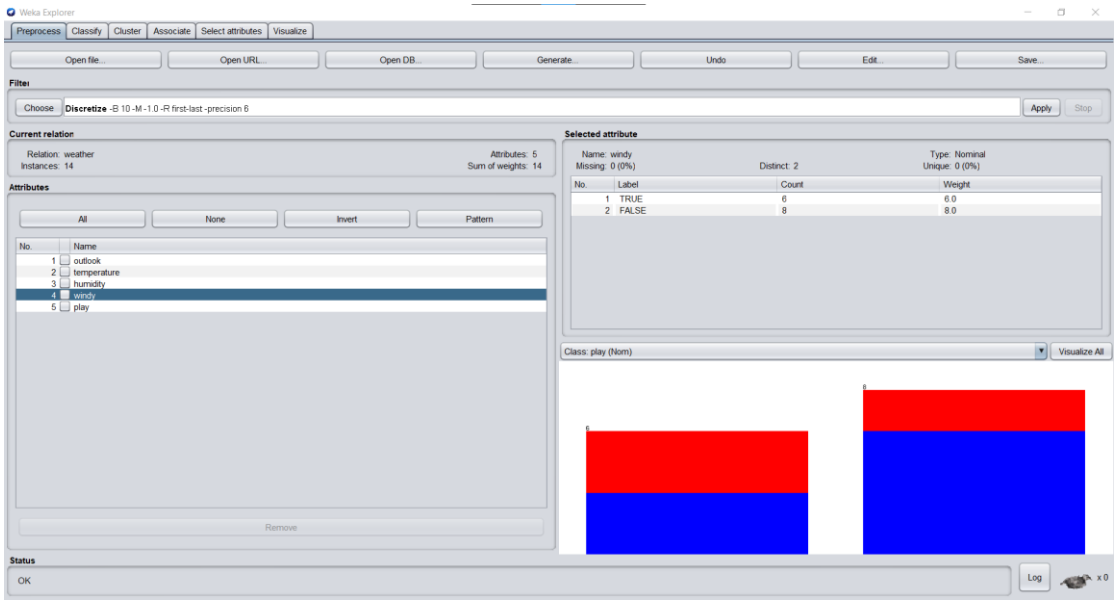
As the amount of big data increases, more people are using data visualization tools to access insights on their computer and on mobile devices. Dashboards are used by business people, data analysts, and data scientists to make data-driven business decisions.

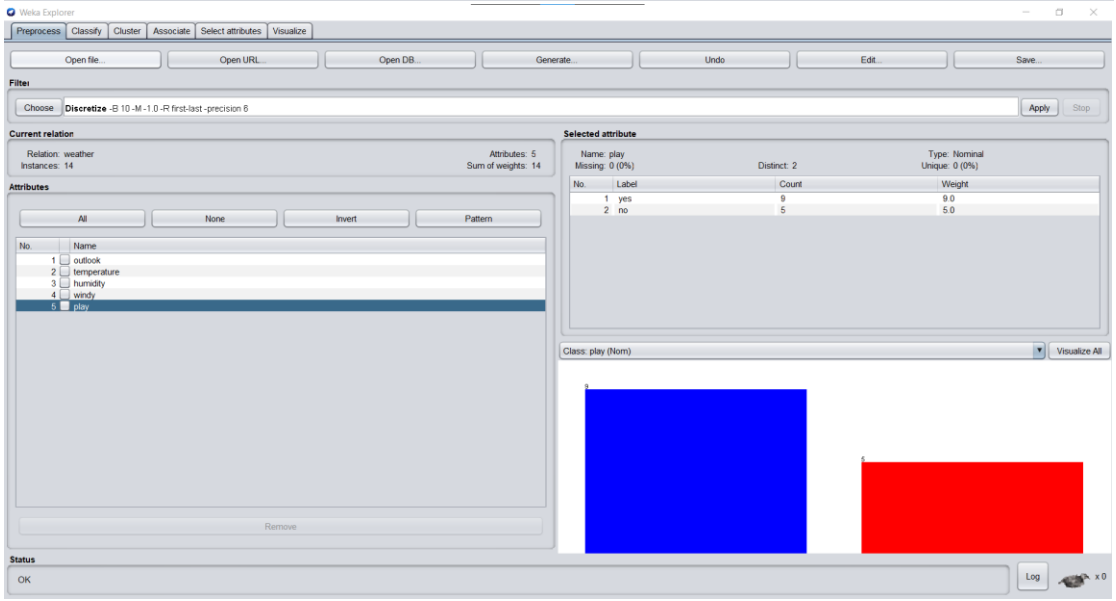
Weather Data before Discretization:

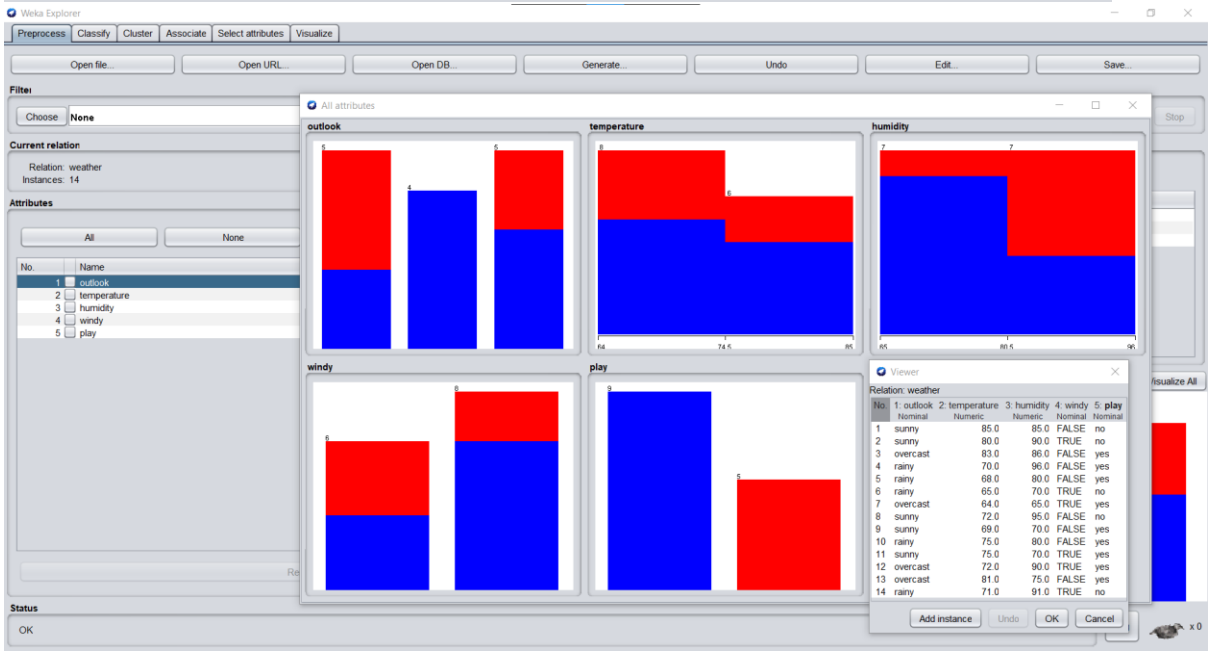




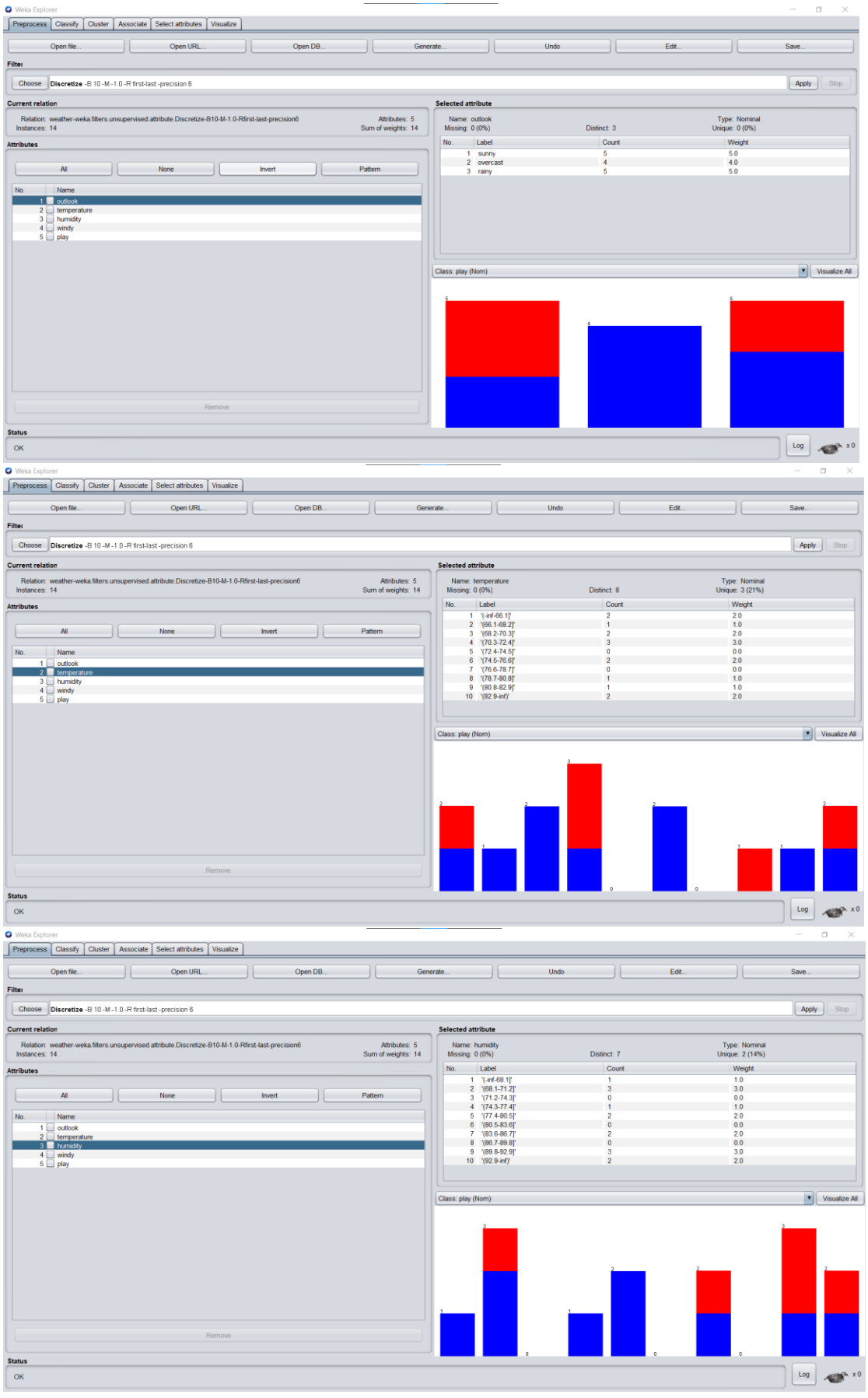


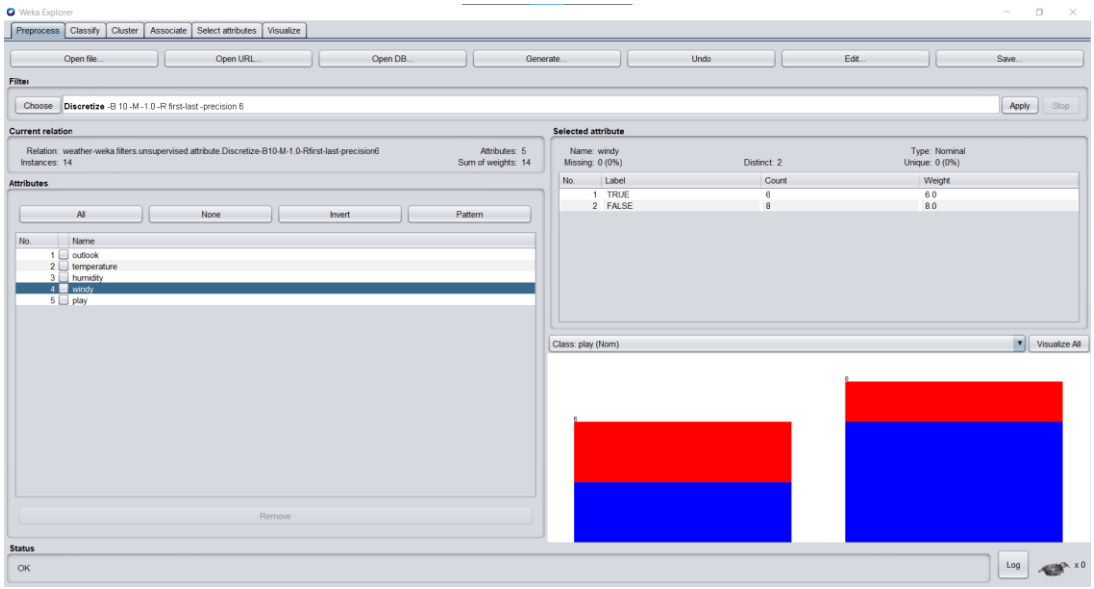

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
Discretize -B 10 -M -1.0 -R first-last -precision 6
Apply
Stop
Current relation
Relation: weather
Instances: 14
Attributes: 5
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
Remove
Selected attribute
Name: windy
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 TRUE 6 6.0
2 FALSE 8 8.0
Class: play (Nom)
Visualize All
Status
OK
Log
x 0

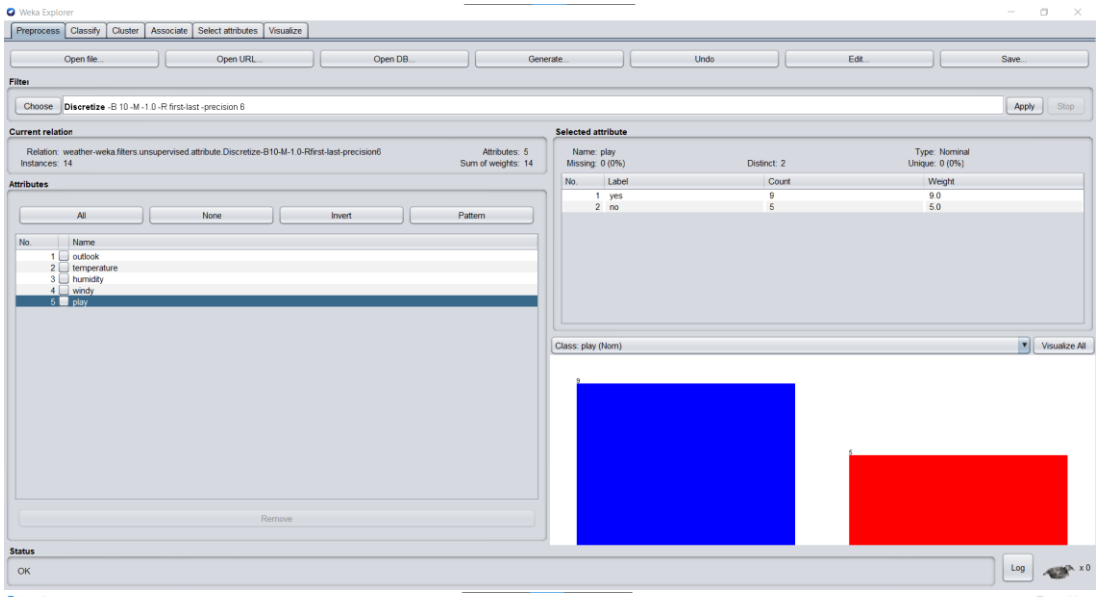

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
Discretize -B 10 -M -1.0 -R first-last -precision 6
Apply
Stop
Current relation
Relation: weather
Instances: 14
Attributes: 5
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
Remove
Selected attribute
Name: play
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 yes 9 9.0
2 no 5 5.0
Class: play (Nom)
Visualize All
Status
OK
Log
x 0

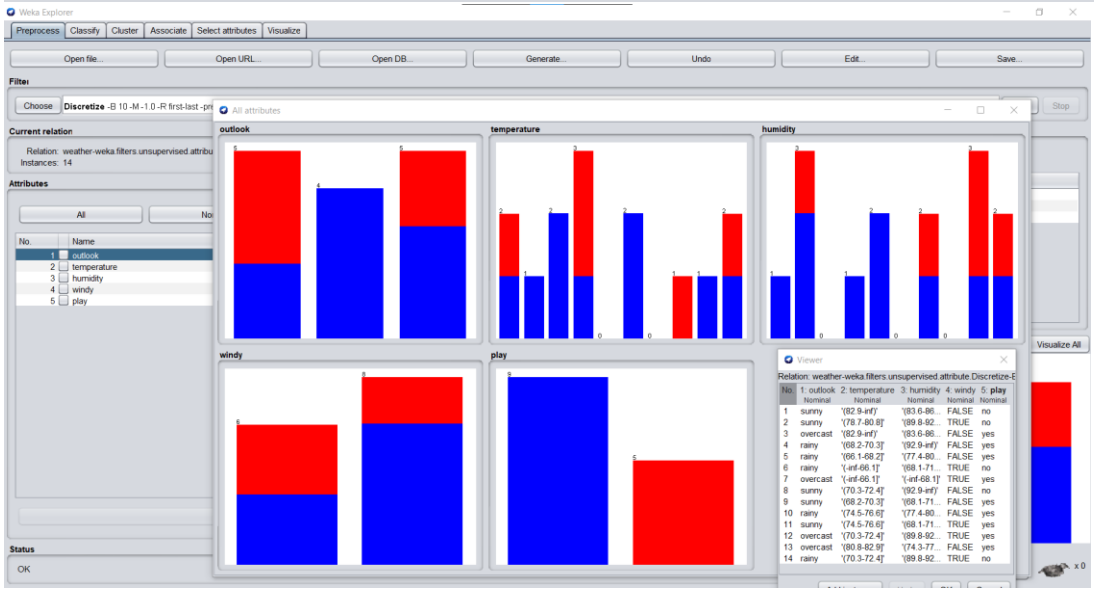

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Current relation
Relation: weather
Instances: 14
Attributes
All
None
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
All attributes
outlook
temperature
humidity
windy
play
Viewer
Relation: weather
No. 1: outlook Nominal 2: temperature Numeric 3: humidity Numeric 4: windy Nominal 5: play Nominal
1 sunny 85.0 85.0 FALSE no
2 sunny 80.0 90.0 TRUE no
3 overcast 83.0 86.0 FALSE yes
4 rainy 70.0 96.0 FALSE yes
5 rainy 68.0 80.0 FALSE yes
6 rainy 65.0 70.0 TRUE no
7 overcast 64.0 65.0 TRUE yes
8 sunny 72.0 95.0 FALSE no
9 sunny 69.0 70.0 FALSE yes
10 rainy 75.0 80.0 FALSE yes
11 sunny 75.0 70.0 TRUE yes
12 overcast 72.0 90.0 TRUE yes
13 overcast 81.0 75.0 FALSE yes
14 rainy 71.0 91.0 TRUE no
Add instance
Undo
OK
Cancel
Status
OK
x 0

Weather Data after Discretization:

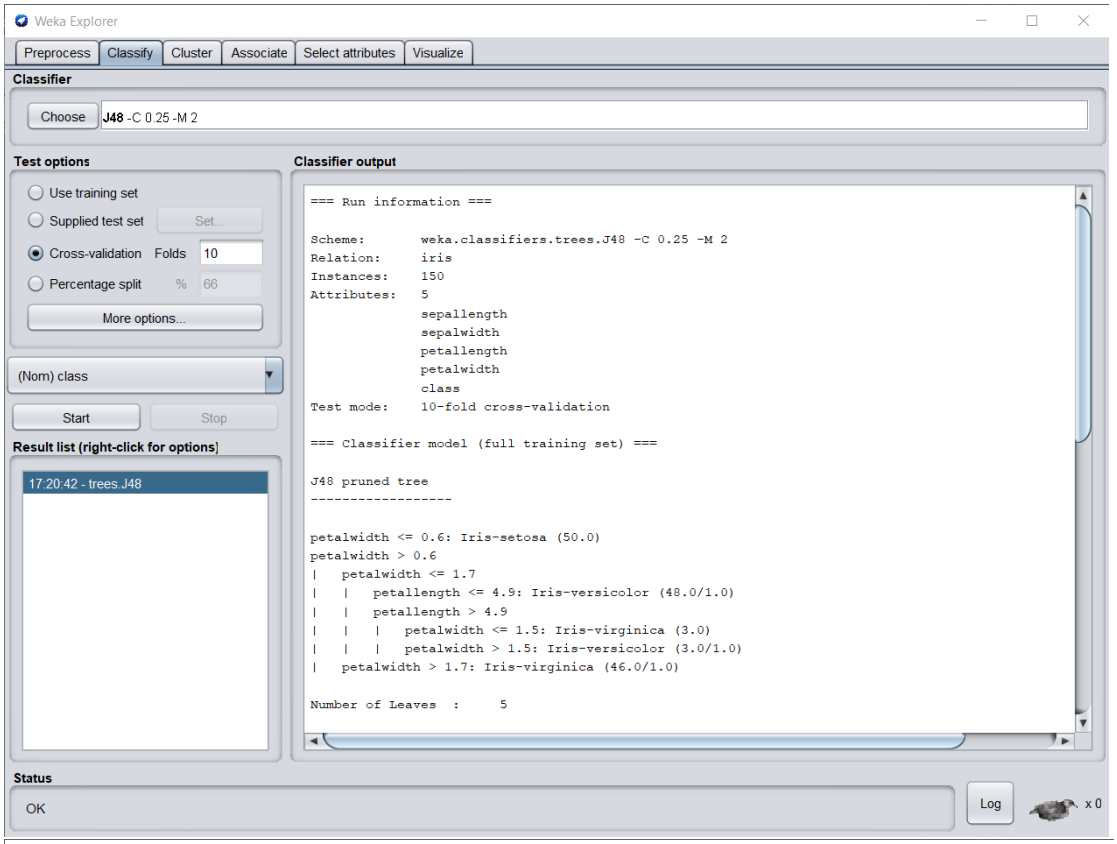


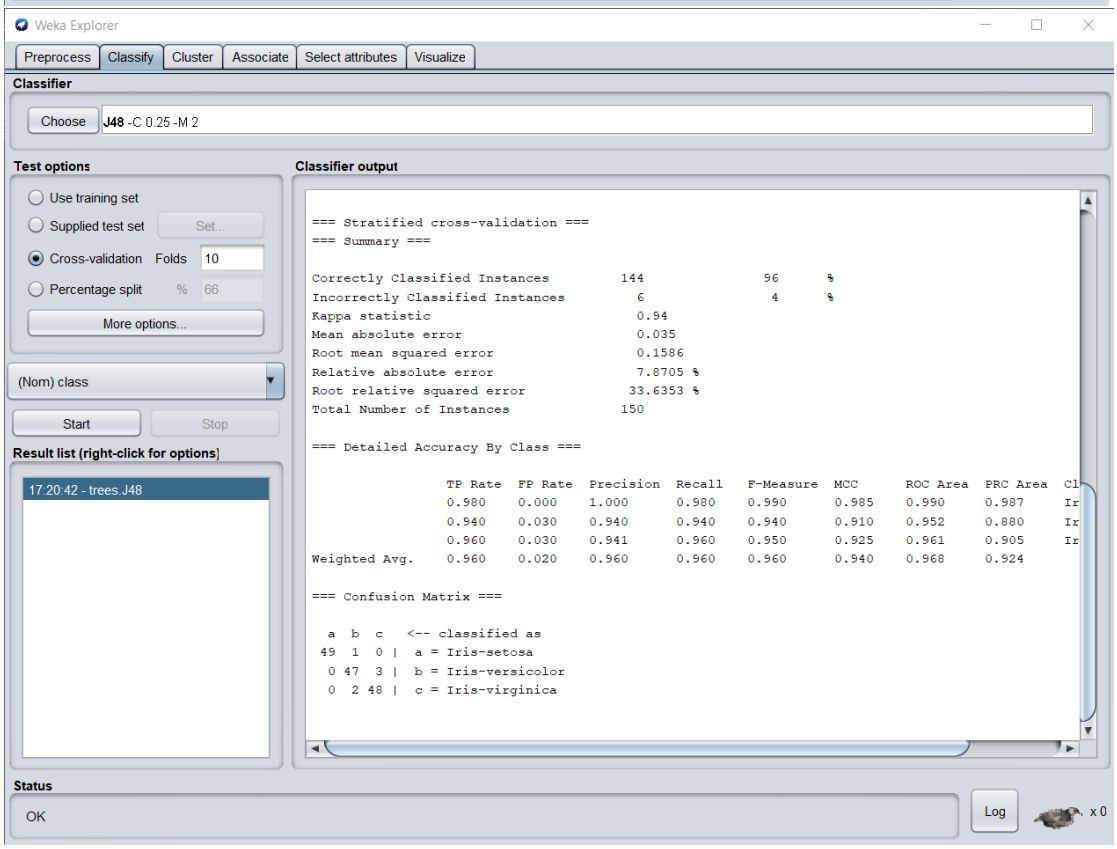

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
Discretize -B 10 -M -1.0 -R first-last -precision 6
Apply
Stop
Current relation
Relation: weather-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-precision6
Instances: 14
Attributes: 5
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
Remove
Selected attribute
Name: windy
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 TRUE 6 6.0
2 FALSE 8 8.0
Class: play (Nom)
Visualize All
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
Discretize -B 10 -M -1.0 -R first-last -precision 6
Apply
Stop
Current relation
Relation: weather-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-precision6
Instances: 14
Attributes: 5
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
Remove
Selected attribute
Name: play
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 yes 9 9.0
2 no 5 5.0
Class: play (Nom)
Visualize All
Status
OK
Log
x 0

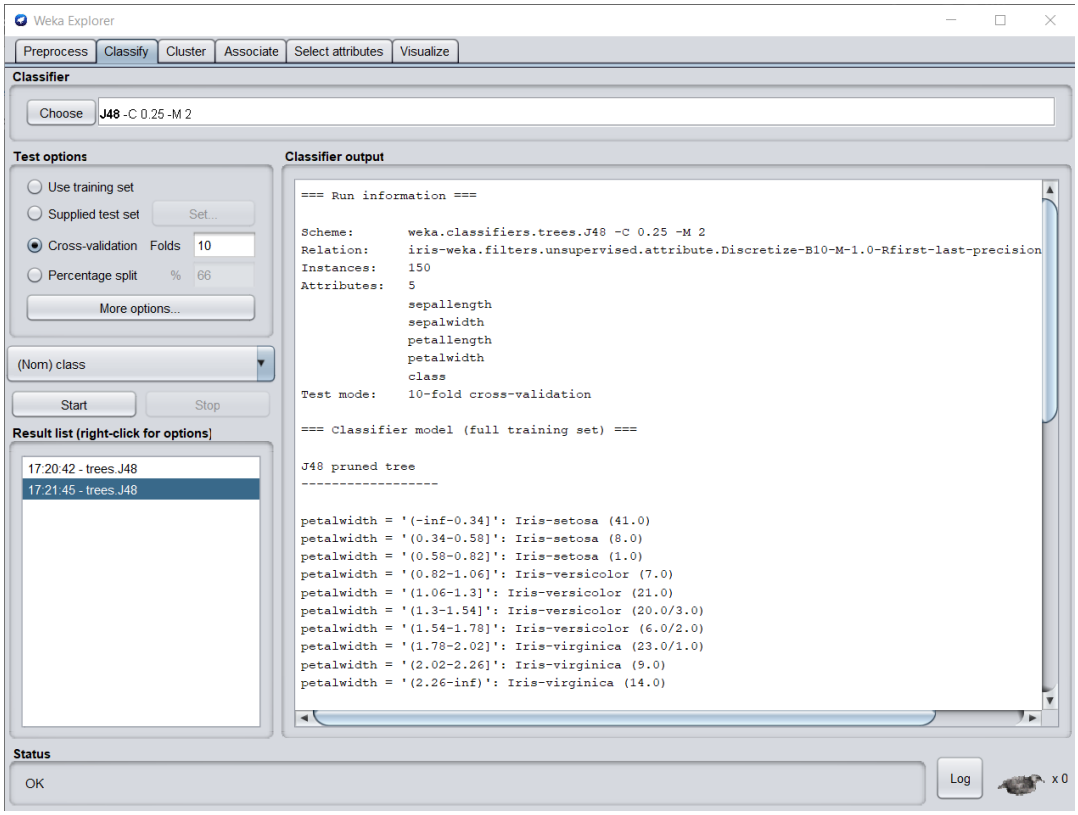

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Current relation
Attributes
All
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
Visualize All
Status
OK
x 0
All attributes
outlook
temperature
humidity
windy
play
Viewer
No. 1: outlook Nominal
2: temperature Nominal
3: humidity Nominal
4: windy Nominal
5: play Nominal

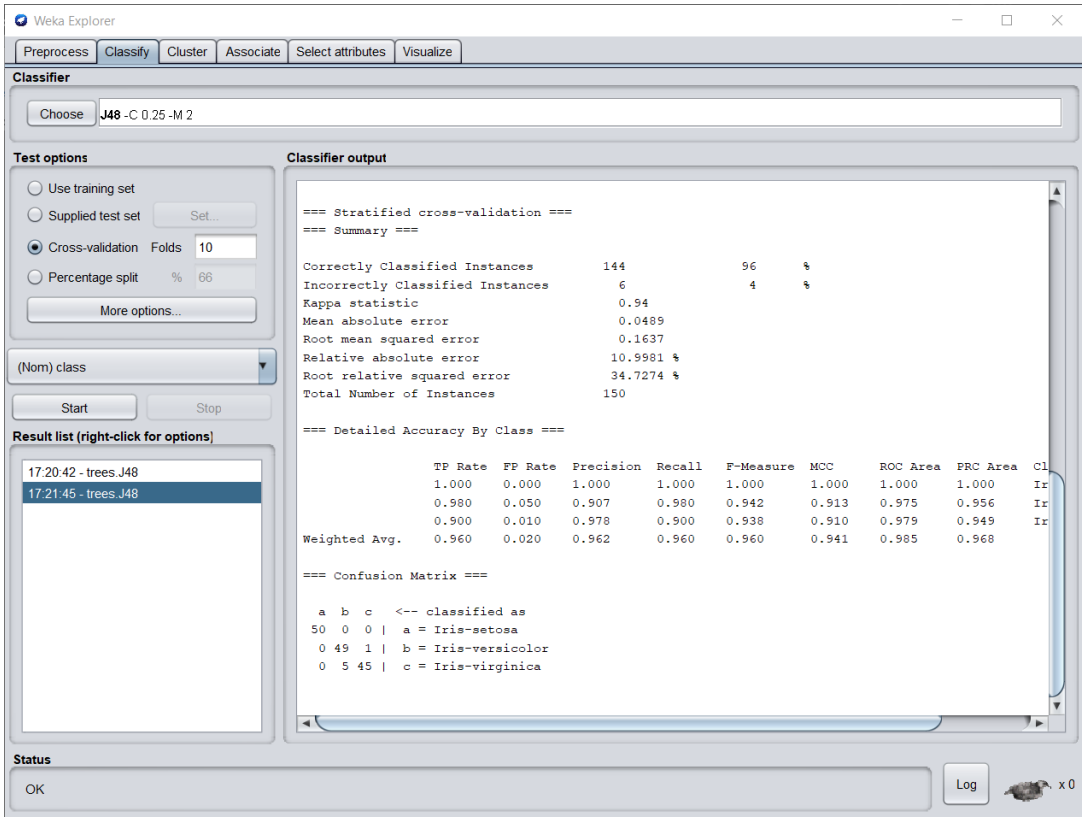
J48 Tree classification on iris data set before discretization:





J48 Tree classification on iris data set after discretization:





Conclusion: We have successfully implemented Data Discretization and Visualization on weather Data Set using Weka Tool.

EXPERIMENT NO. 6

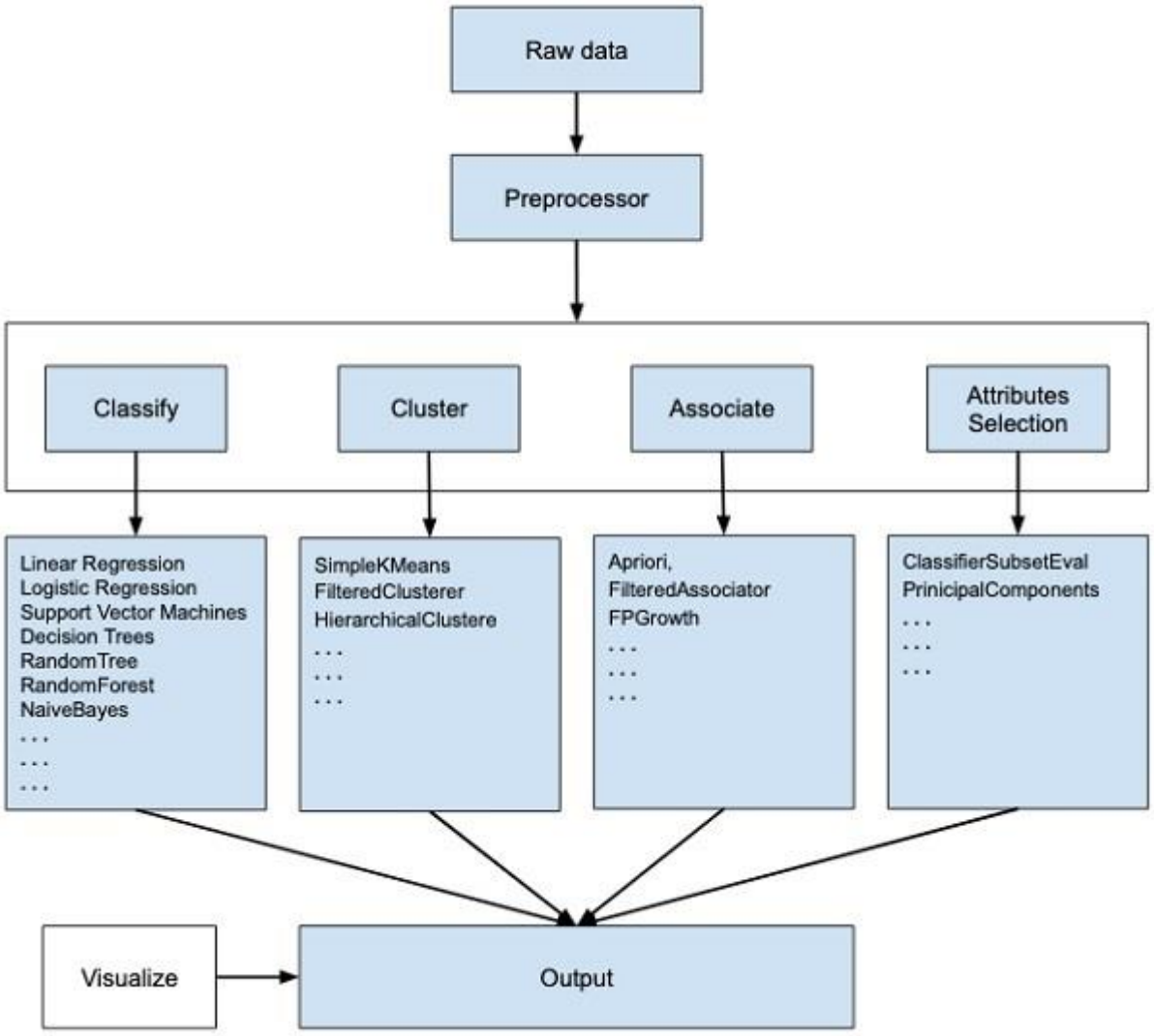
Aim: To perform data Pre-processing and implement Classification, Clustering and Association algorithms on data set.

Requirement: Windows O.S and Weka Tool.

Theory:

Weka Tool:

WEKA - an open source software provides tools for data pre-processing, 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 −



If you observe the beginning of the flow of the image, you will understand that there are many stages in dealing with Big Data to make it suitable for machine learning −

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 pre-processing tools provided in WEKA to cleanse the data.

Then, you would save the pre-processed 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.

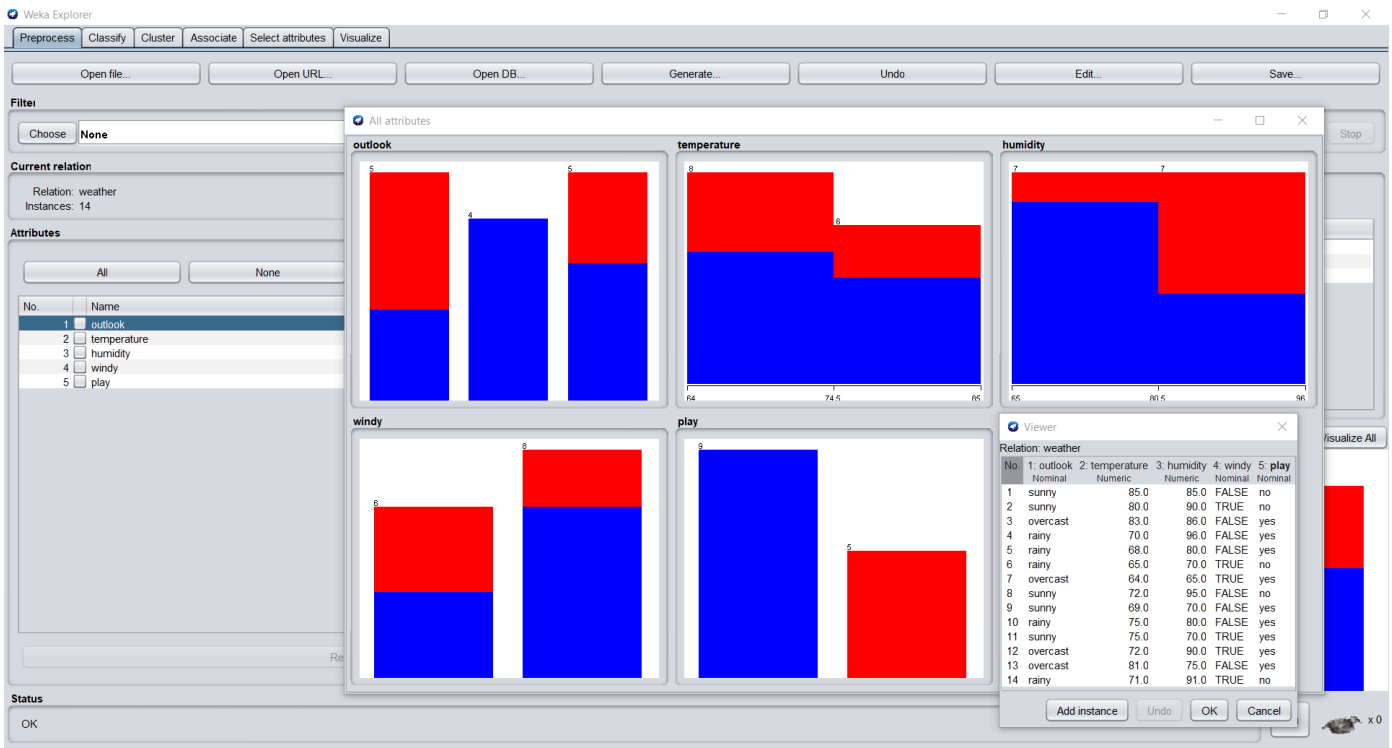
Then, WEKA would give you the statistical output of the model processing. It provides you a visualization tool to inspect the data.

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.

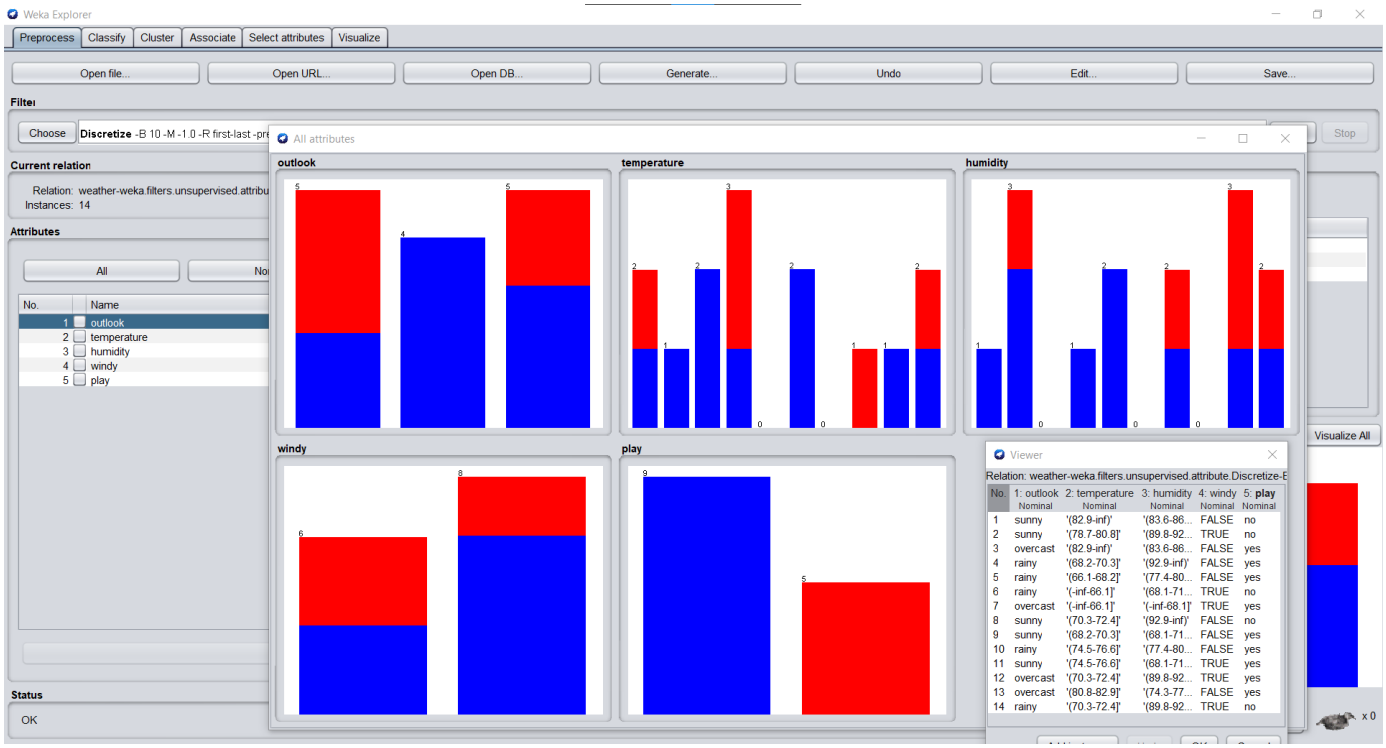
Thus, the use of WEKA results in a quicker development of machine learning models on the whole.

Output:

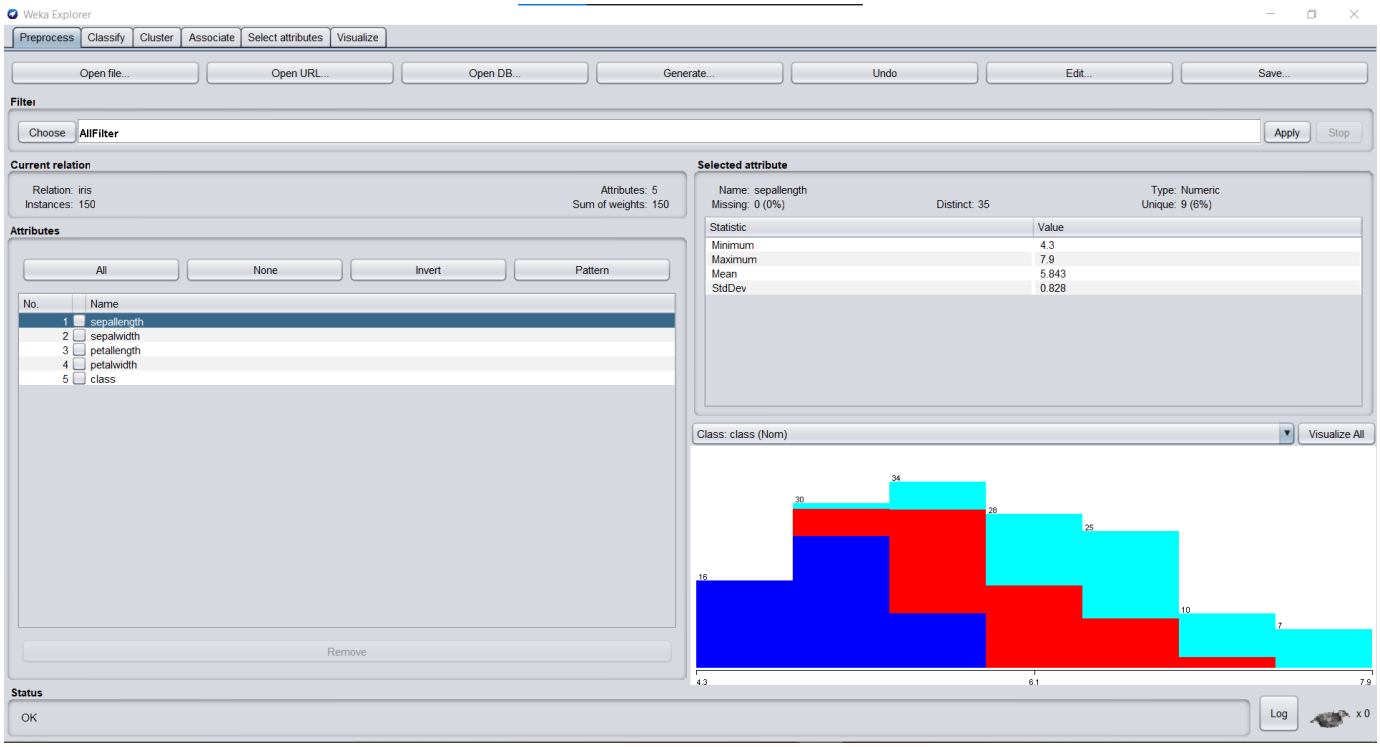
Data Set before Data Pre-processing (Data Discretization):



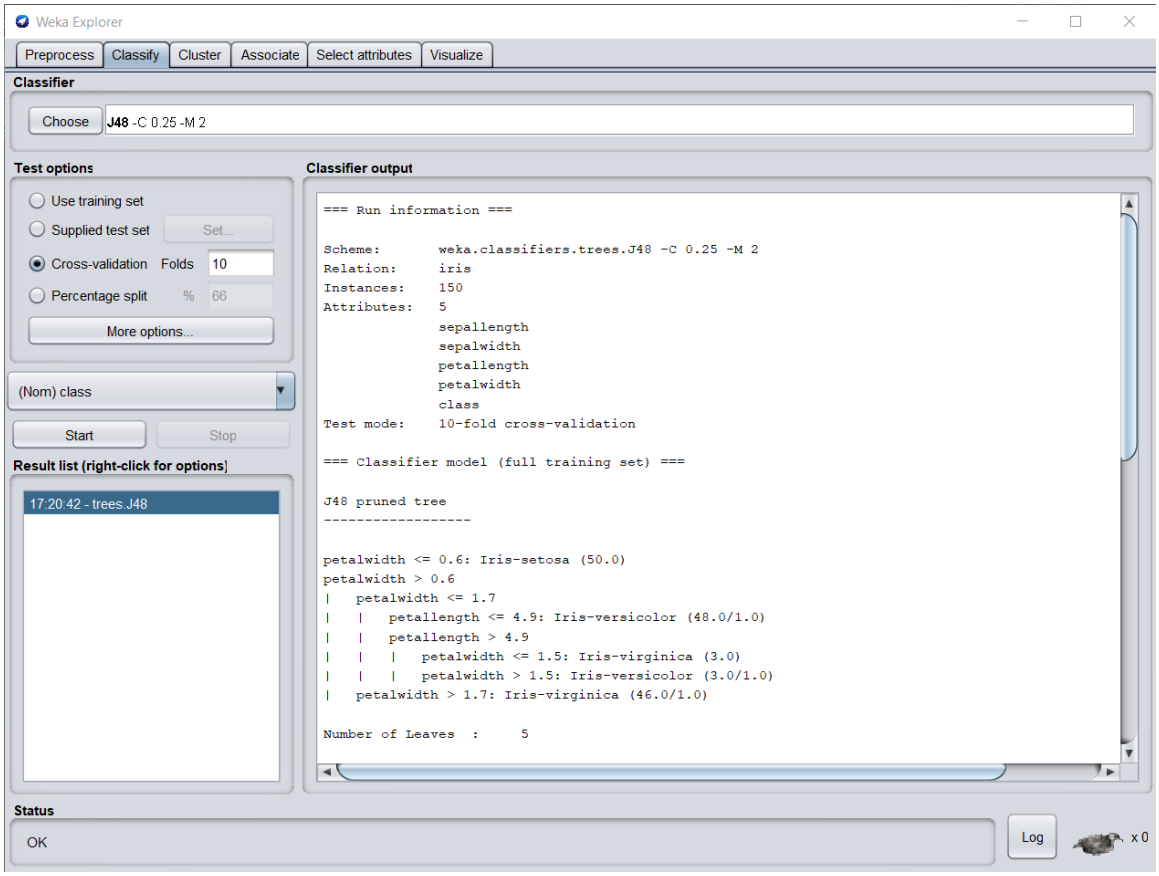
Data Set after Data Pre-processing (Discretization):

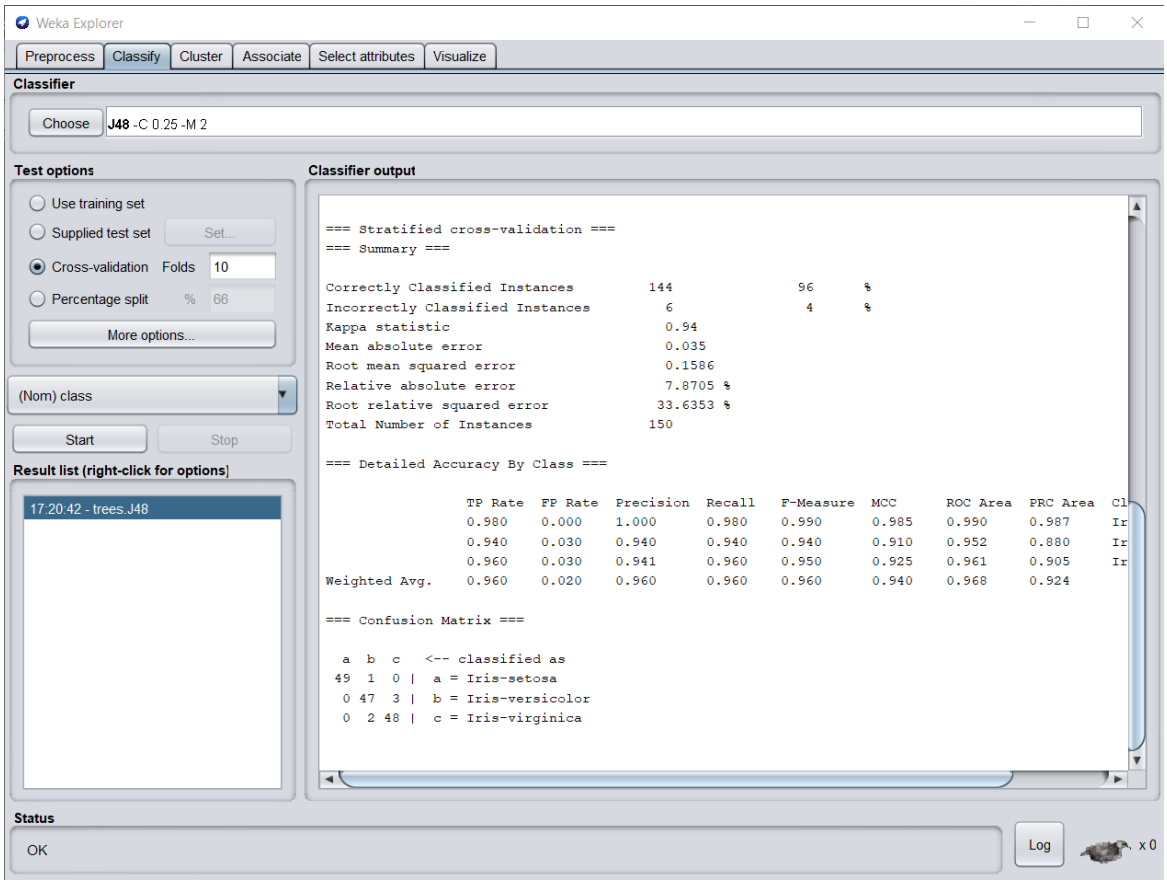


Iris Data Set:

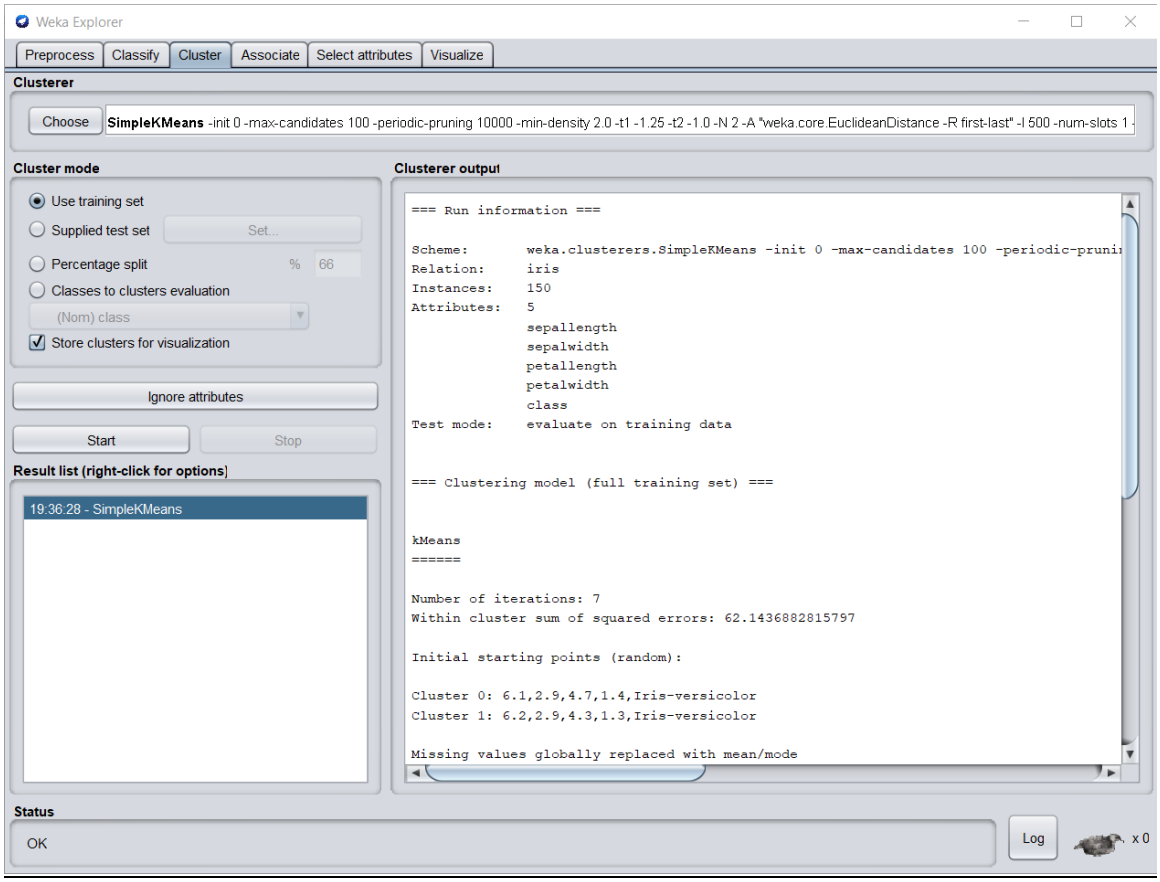


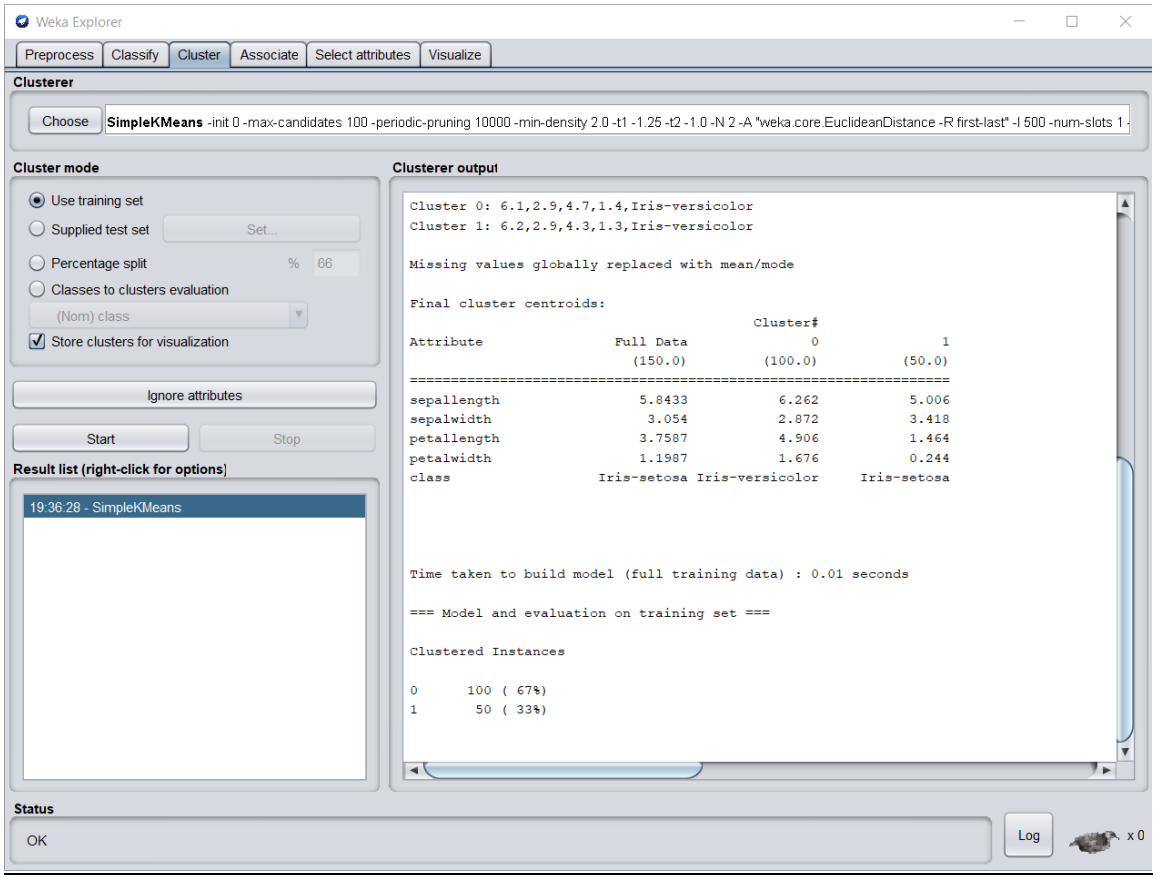
J48 Tree classification on iris data set:

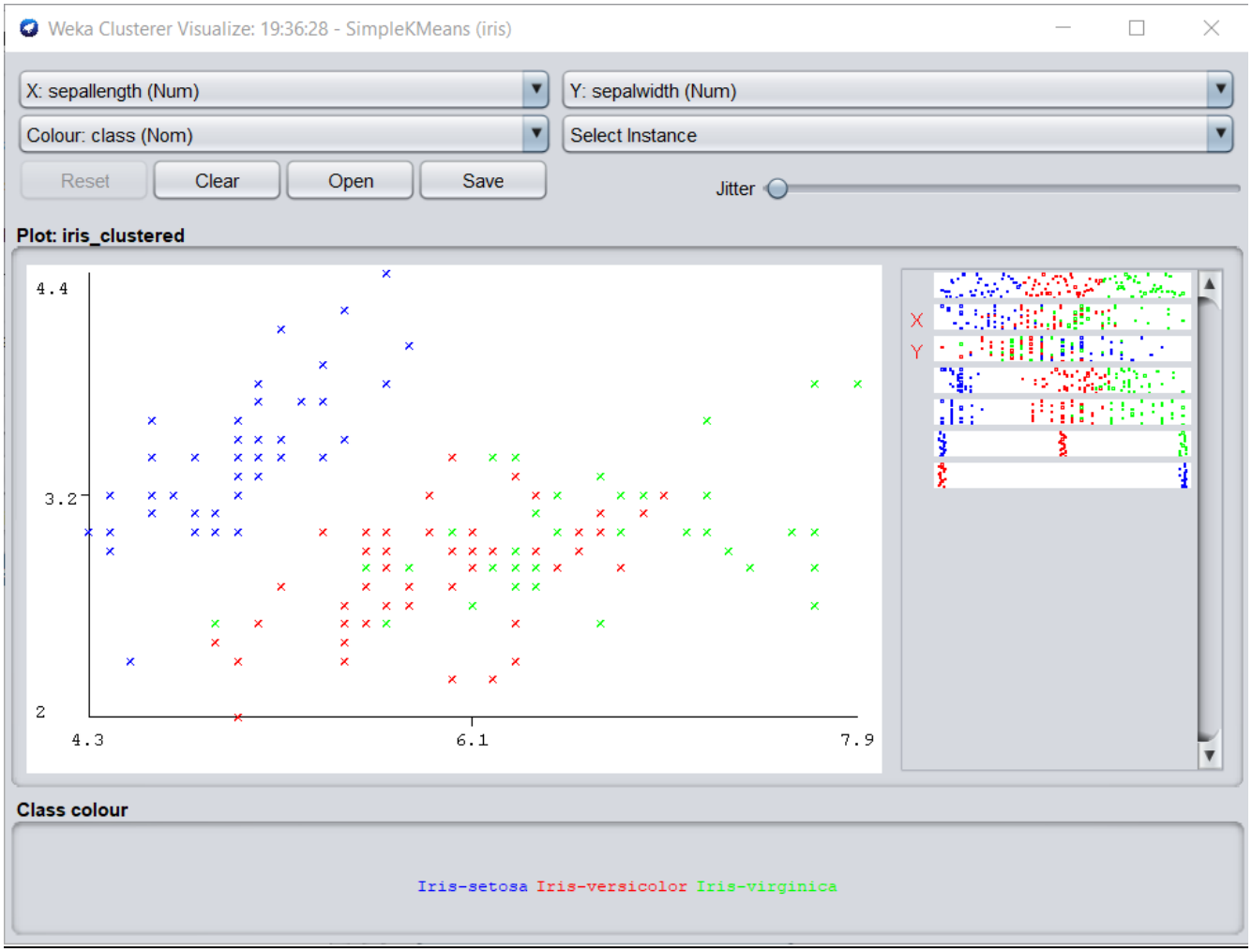




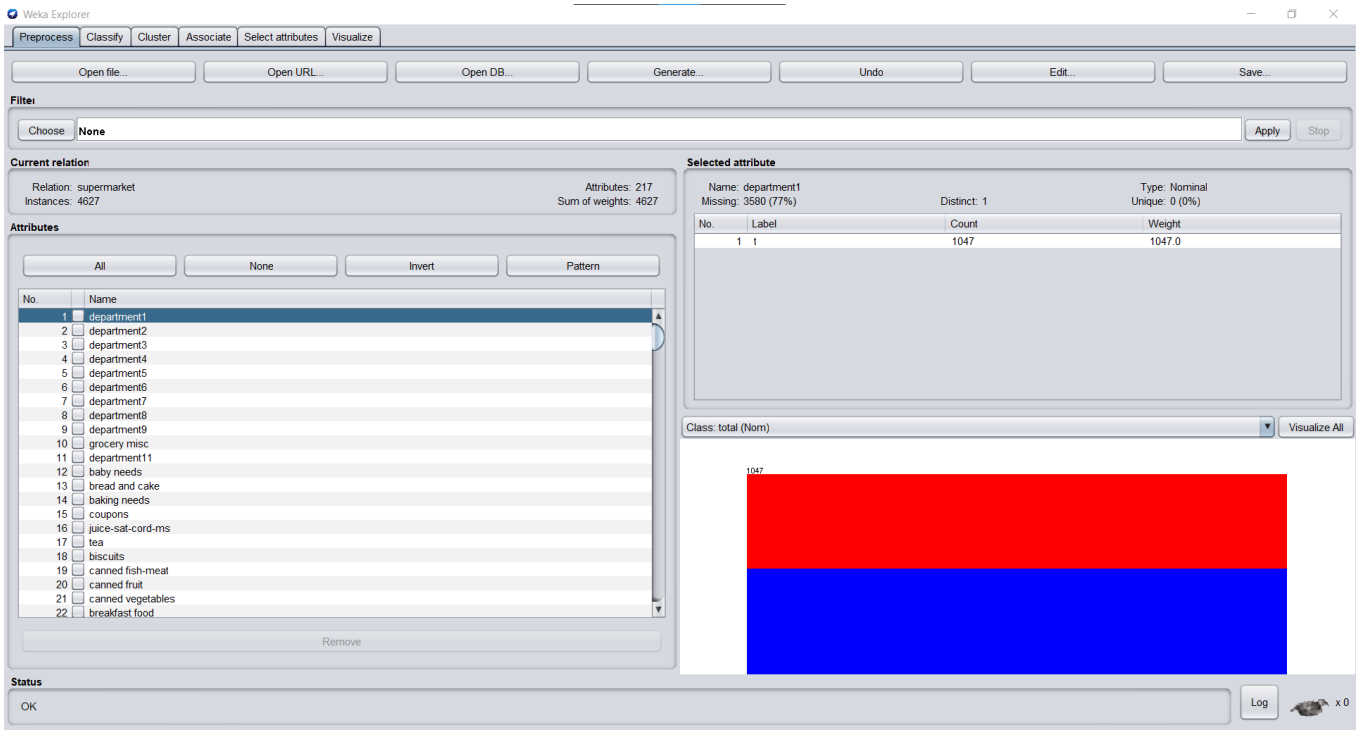
Data Clustering (K-Means/ K-Medoids) on iris Data Set:

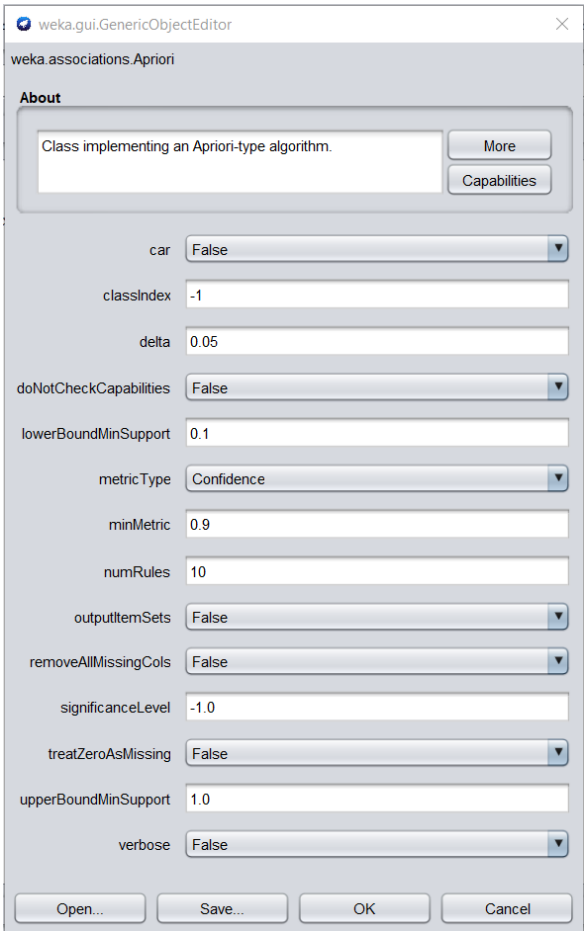


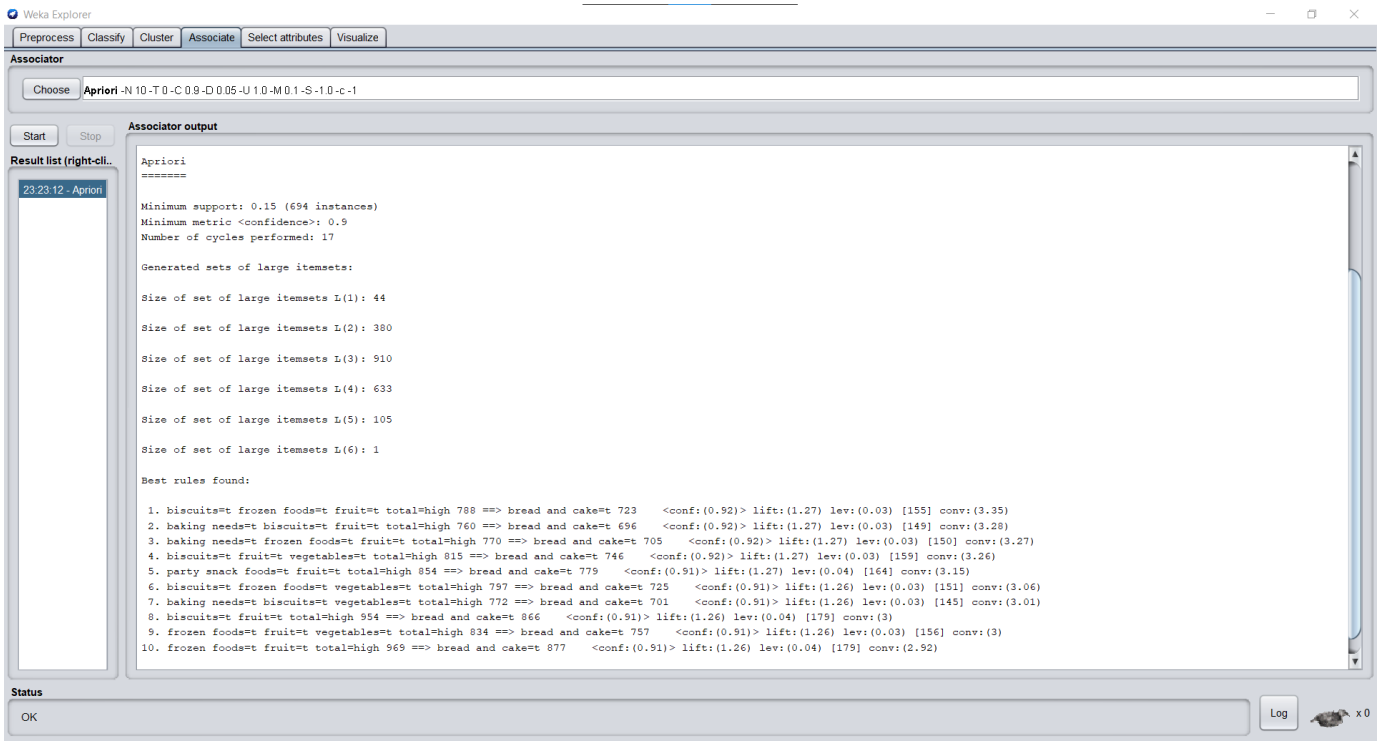




Association algorithm (Apriori) on Supermarket Data Set:







Conclusion: We have successfully Pre-processed data and implemented Classification, Clustering and Association algorithms on data sets using Weka Tool.

EXPERIMENT NO. 7

Aim: To implement Clustering algorithms (K-means/K-medoids).

Requirements: Windows O.S, Weka tool, Python and Python libraries: Pandas, Numpy, Sklearn and Matplot.

Problem Statement: To implement Clustering algorithms (K-means/K-medoids) on iris data set.

Theory:

K-Means Clustering:

K-Means algorithm is a centroid based clustering technique. This technique cluster the dataset to k different cluster having an almost equal number of points. Each cluster is k-means clustering algorithm is represented by a centroid point.

What is a centroid point?

The centroid point is the point that represents its cluster. Centroid point is the average of all the points in the set and will change in each step and will be computed by:

$$C_i = \frac{1}{||S_i||} \sum_{x_j \epsilon S_i} x_j$$

The idea of the K-Means algorithm is to find k-centroid points and every point in the dataset will belong either of k-sets having minimum Euclidean distance.

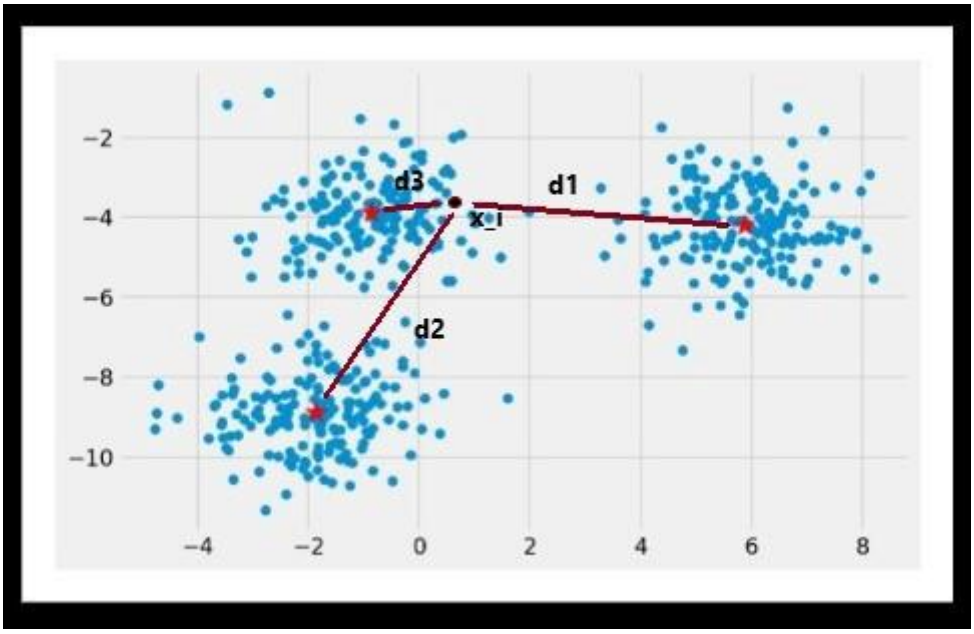


Image a

From the image above (Image a), the distance of point x_i from all three centroids are d1, d2, d3, x_i point is nearest to centroid_3 with distance d3, so the point x_i will belong to the cluster of centroid_3 and this process will continue for all the points in the dataset.

Cost Function of K-Means:

$$C_1, C_2, \cdots, C_k = argmin \sum_{i=1}^{k} \sum_{x \epsilon S_i} ||x - C_i||^2$$

The idea of the K-Means algorithm is to find k centroid points (C_1, C_2, . . . C_k) by minimizing the sum over each cluster of the sum of the square of the distance between the point and its centroid.

Iterative implementation of the K-Means algorithm:

Steps #1: Initialization:

The initial k-centroids are randomly picked from the dataset of points.

Steps #2: Assignment:

For each point in the dataset, find the Euclidean distance between the point and all centroids. The point will be assigned to the cluster with the nearest centroid.

Steps #3: Updation of Centroid:

Update the value of the centroid with the new mean value.

Steps #4: Repeat:

Repeat steps 2 and 3 unless convergence is achieved. If convergence is achieved then break the loop. Convergence refers to the condition where the previous value of centroids is equal to the updated value.

K-Medoids clustering:

It is a clustering algorithm resembling the K-Means clustering technique. It falls under the category of unsupervised machine learning. It majorly differs from the K-Means algorithm in terms of the way it selects the clusters' centres. The former selects the average of a cluster's points as its centre (which may or may not be one of the data points) while the latter always picks the actual data points from the clusters as their centres (also known as '**exemplars**' or '**medoids**'). K-Medoids also differs in this respect from the K-Medians algorithm which is the same as K-means, except that it chooses the medians (instead of means) of the clusters as centres.

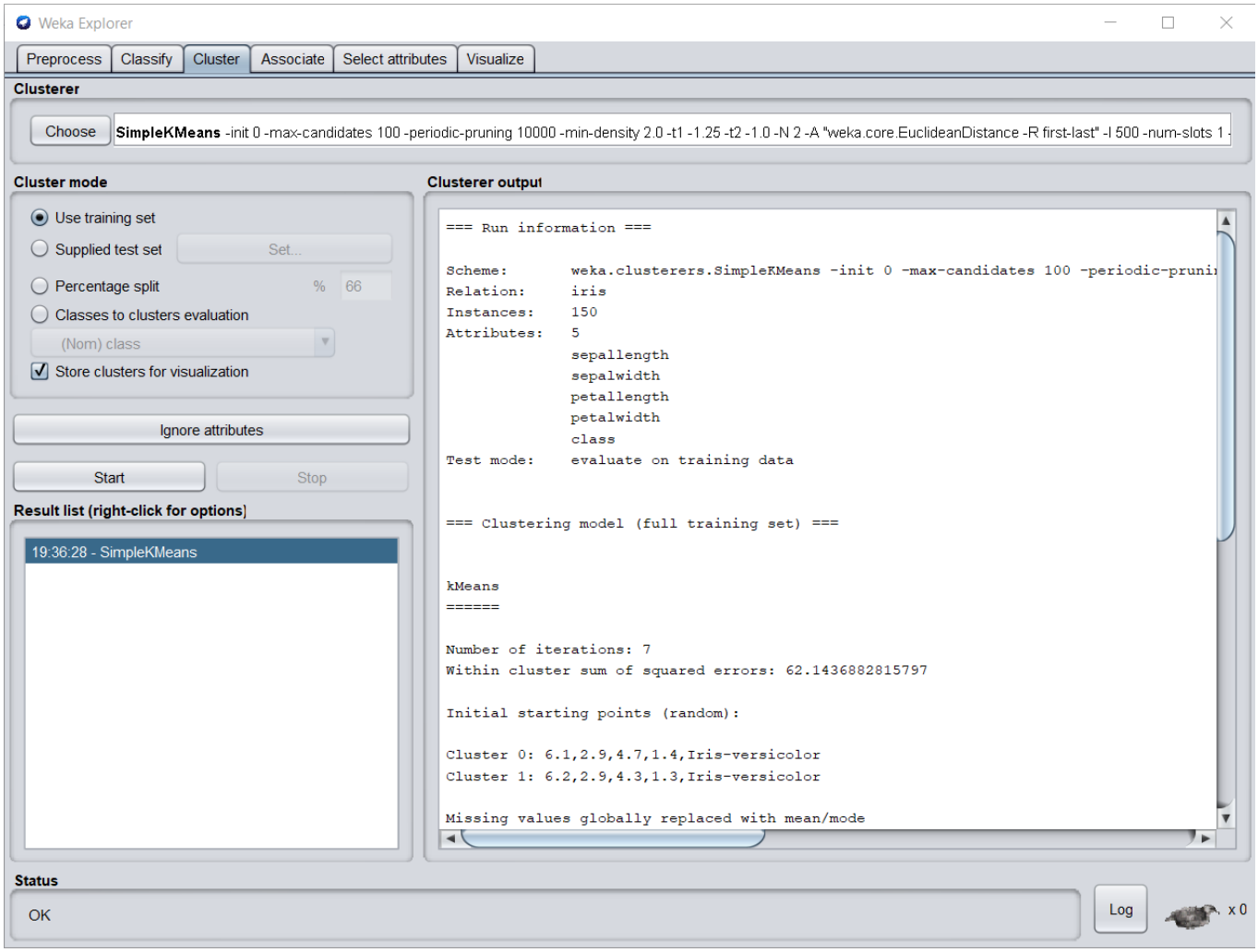
Working of the K-Medoids approach:

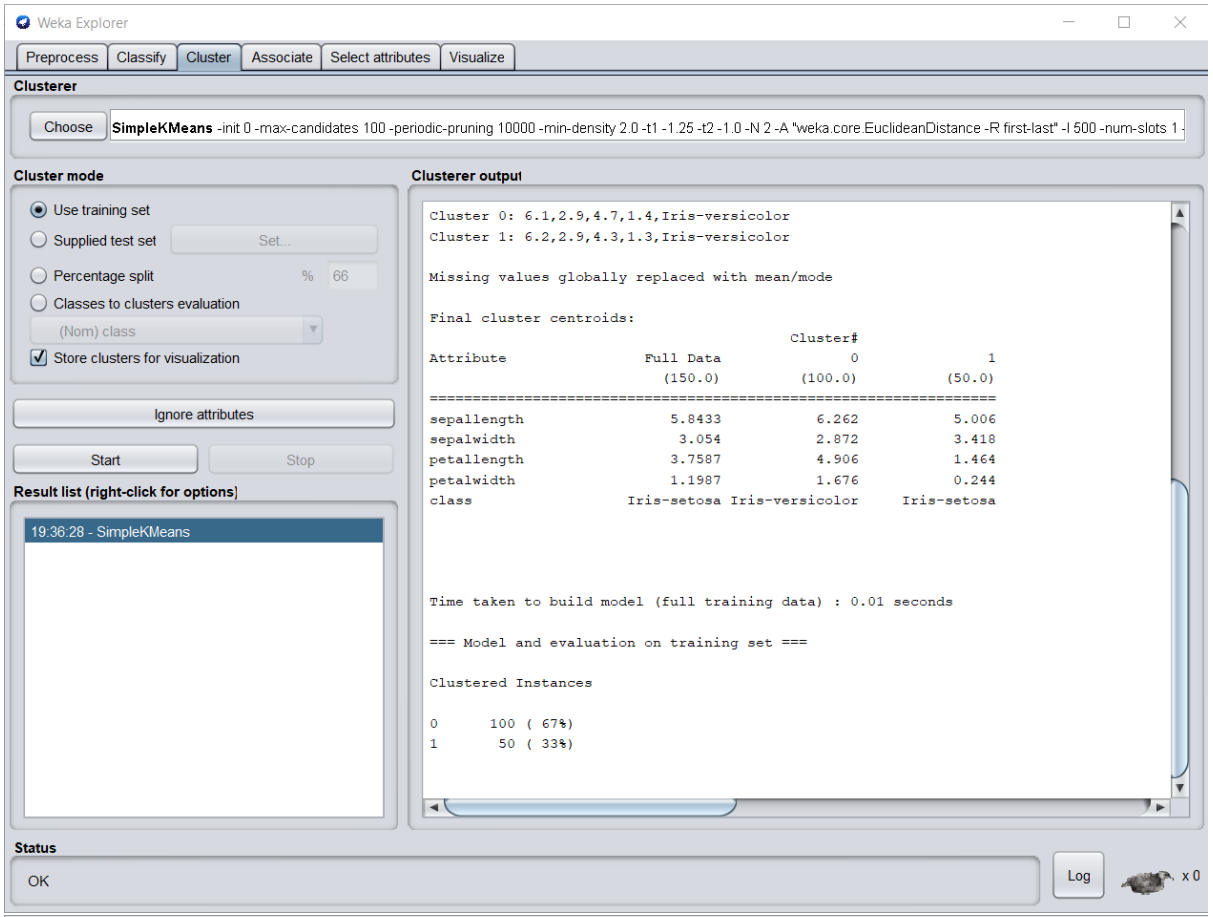
The steps followed by the K-Medoids algorithm for clustering are as follows:

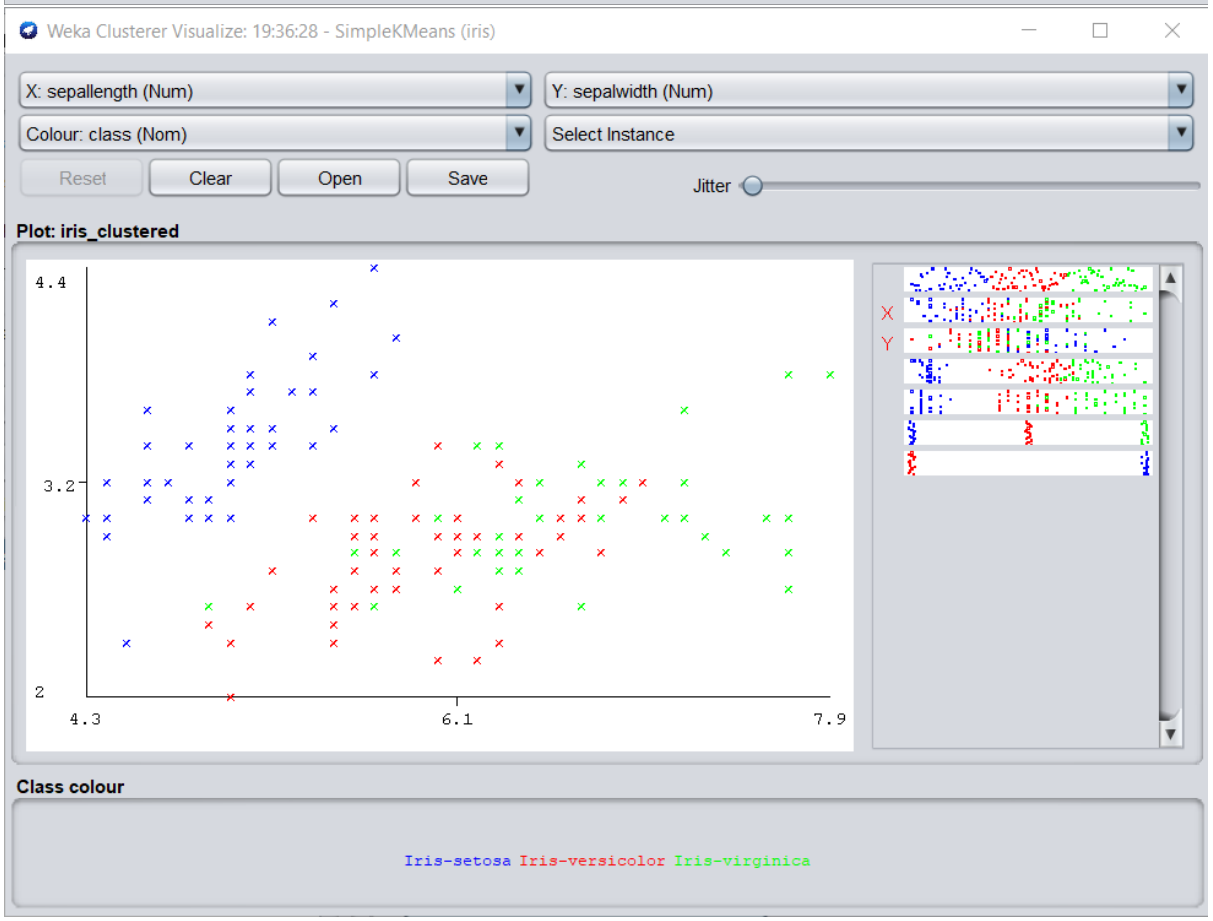
1. Randomly choose 'k' points from the input data ('k' is the number of clusters to be formed). The correctness of the choice of k's value can be assessed using methods such as silhouette method.

2. Each data point gets assigned to the cluster to which its nearest medoid belongs.
3. For each data point of cluster i, its distance from all other data points is computed and added. The point of ith cluster for which the computed sum of distances from other points is minimal is assigned as the medoid for that cluster.
4. Steps (2) and (3) are repeated until convergence is reached i.e. the medoids stop moving.

Weka K-means Output:




Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Nom) class
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
19:36:28 - SimpleKMeans
Clusterer output
Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor
Missing values globally replaced with mean/mode
Final cluster centroids:
Cluster#
Attribute Full Data 0 1
(150.0) (100.0) (50.0)
sepallength 5.8433 6.262 5.006
sepalwidth 3.054 2.872 3.418
petallength 3.7587 4.906 1.464
petalwidth 1.1987 1.676 0.244
class Iris-setosa Iris-versicolor Iris-setosa
Time taken to build model (full training data) : 0.01 seconds
=== Model and evaluation on training set ===
Clustered Instances
0 100 (67%)
1 50 (33%)
Status
OK
Log
x 0


Weka Clusterer Visualize: 19:36:28 - SimpleKMeans (iris)
X: sepallength (Num)
Y: sepalwidth (Num)
Colour: class (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: iris_clustered
4.4
3.2
2
4.3
6.1
7.9
X
Y
Class colour
Iris-setosa Iris-versicolor Iris-virginica

Python K-means and K-medoids Output:

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
import warnings
warnings.filterwarnings("ignore")
```

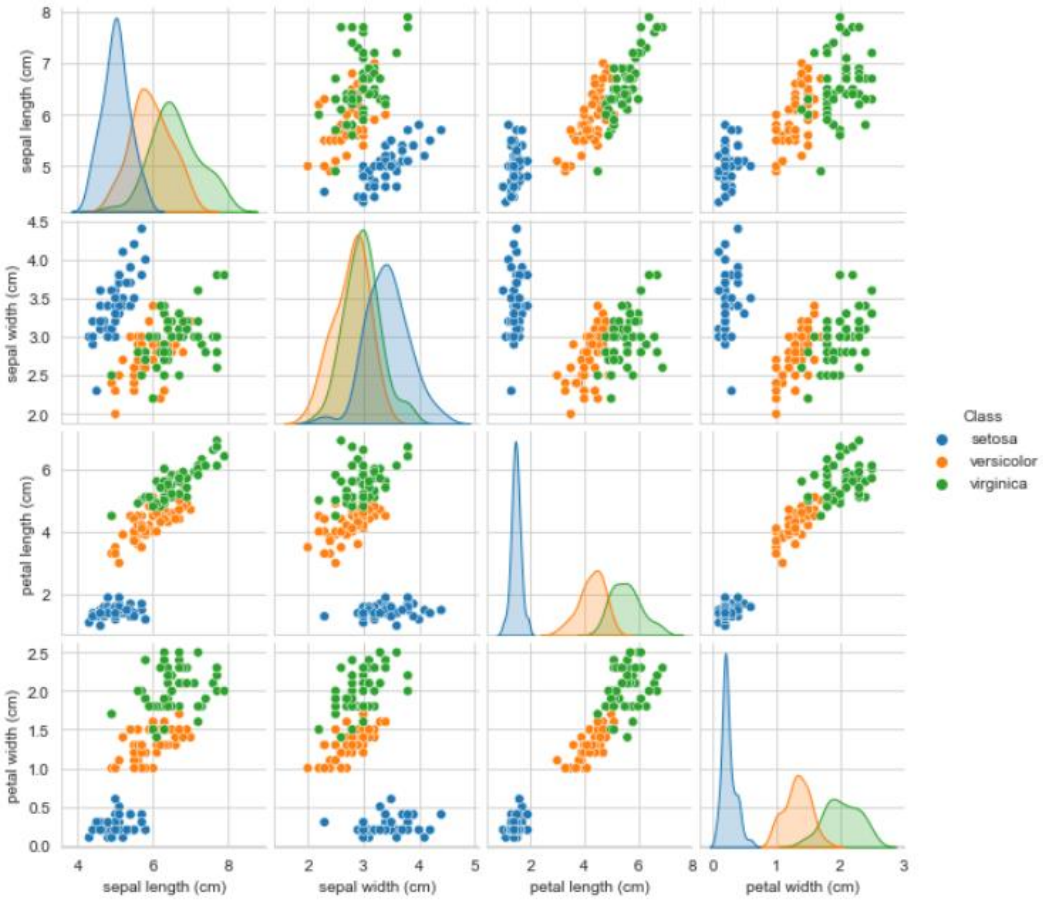
In [2]:
```
dt = load_iris()
X, y = load_iris(return_X_y=True)
```

In [3]:
```
df = pd.DataFrame(np.column_stack((X,y)),columns = [*dt.feature_names,"Class"])
df["Class"].replace([0,1,2],['setosa', 'versicolor', 'virginica'],inplace=True)
```

In [4]:
```
df["Class"].value_counts()
```

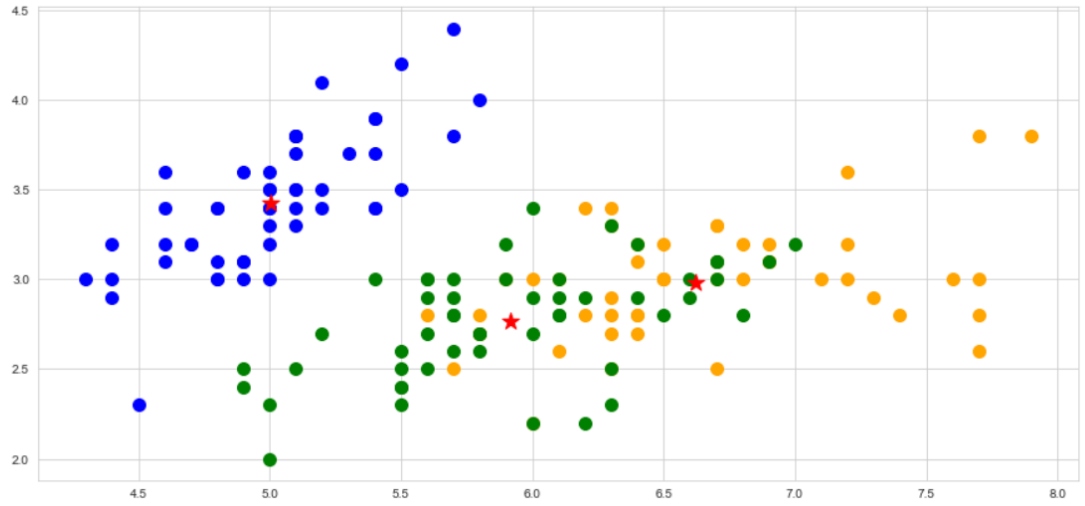
Out[4]:
```
versicolor    50
virginica     50
setosa        50
Name: Class, dtype: int64
```

In [12]:
```
sns.set_style("whitegrid")
sns.pairplot(df,hue="Class",size=2)
plt.show()
```

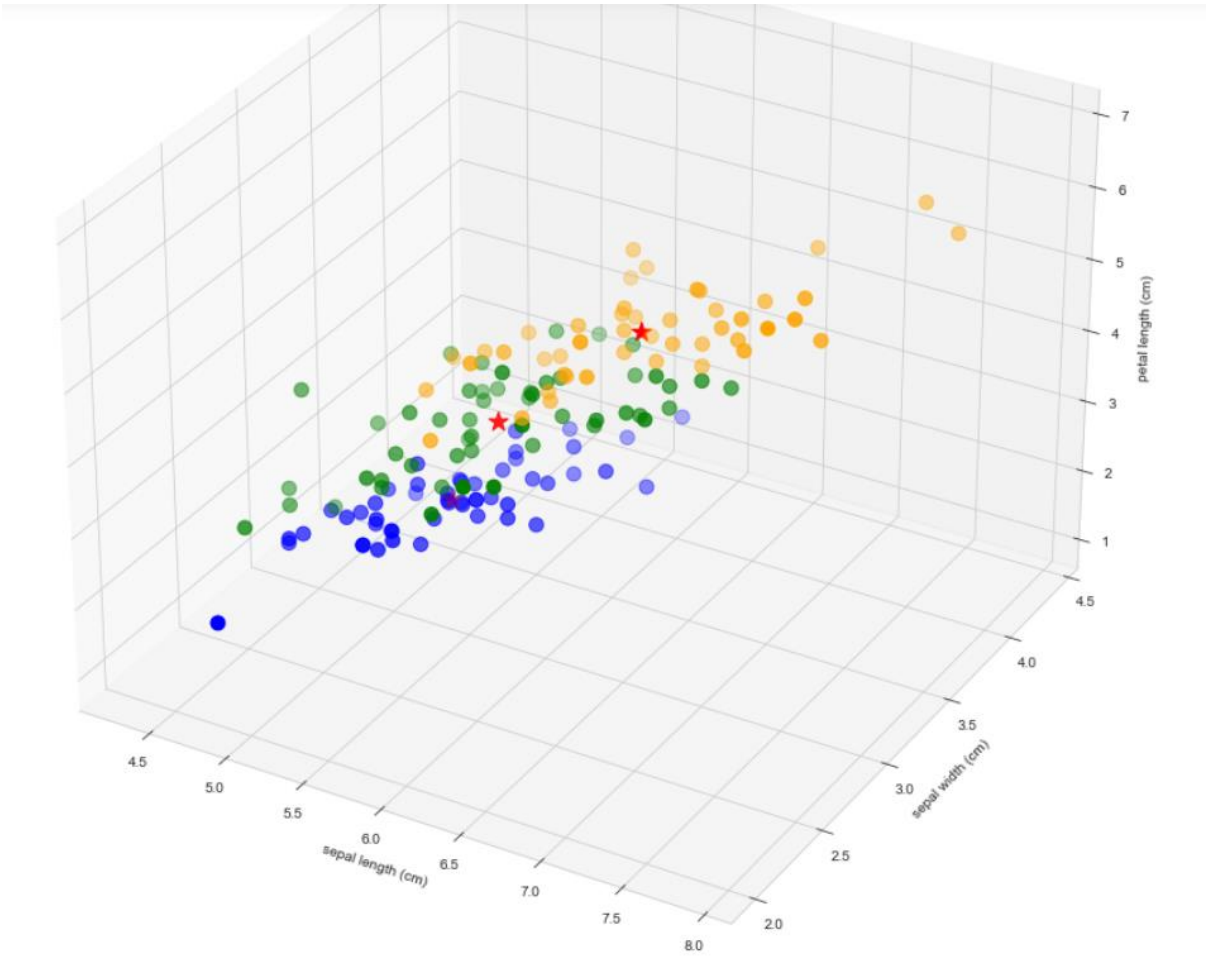


In [6]:
```
kmeans = KMeans(n_clusters=3,init = "random",random_state=77,algorithm="auto")
temp_kmeans = kmeans.fit_predict(df.replace(['setosa', 'versicolor', 'virginica'],[0,1,2]))
```

In [7]:
```
plt.subplots(figsize=(15,7))
plt.scatter(df.iloc[temp_kmeans == 0,0],df.iloc[temp_kmeans == 0,1],label="Setosa",color="blue",s=100)
plt.scatter(df.iloc[temp_kmeans == 1,0],df.iloc[temp_kmeans == 1,1],label="Versicolor",color="orange",s=100)
plt.scatter(df.iloc[temp_kmeans == 2,0],df.iloc[temp_kmeans == 2,1],label="Virginica",color="green",s=100)
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=200,marker="*",color="red")
plt.show()
```

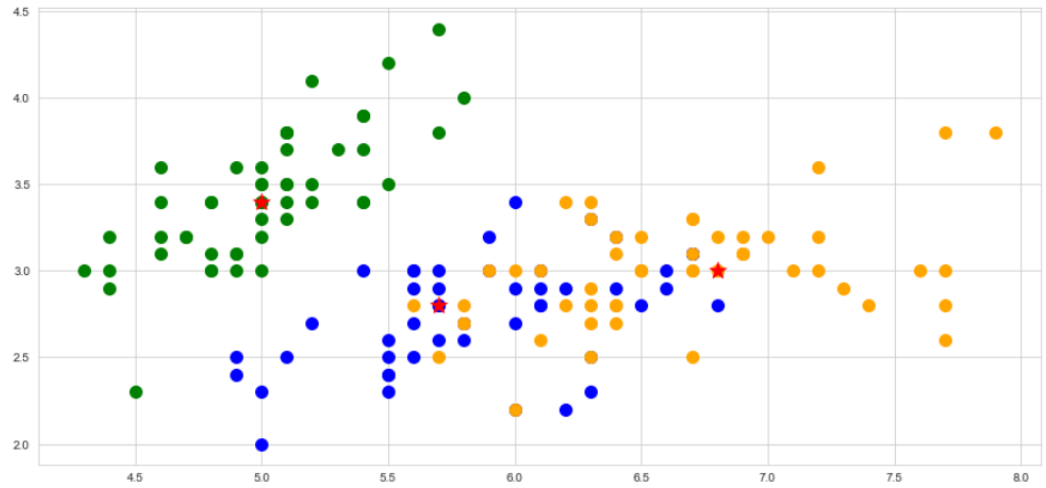


In [8]:
```
fig = plt.figure(figsize=(15,15))
ax = fig.add_subplot(projection="3d")
ax.scatter(df.iloc[temp_kmeans == 0,0],df.iloc[temp_kmeans == 0,1],df.iloc[temp_kmeans == 0,2],label="Setosa",color="blue",s=100
ax.scatter(df.iloc[temp_kmeans == 1,0],df.iloc[temp_kmeans == 1,1],df.iloc[temp_kmeans == 1,2],label="Versicolor",color="orange"
ax.scatter(df.iloc[temp_kmeans == 2,0],df.iloc[temp_kmeans == 2,1],df.iloc[temp_kmeans == 2,2],label="Virginica",color="green",s
ax.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],kmeans.cluster_centers_[:,2],s=200,marker="*",color="red")
ax.set_xlabel(df.columns[0])
ax.set_ylabel(df.columns[1])
ax.set_zlabel(df.columns[2])
plt.show()
```

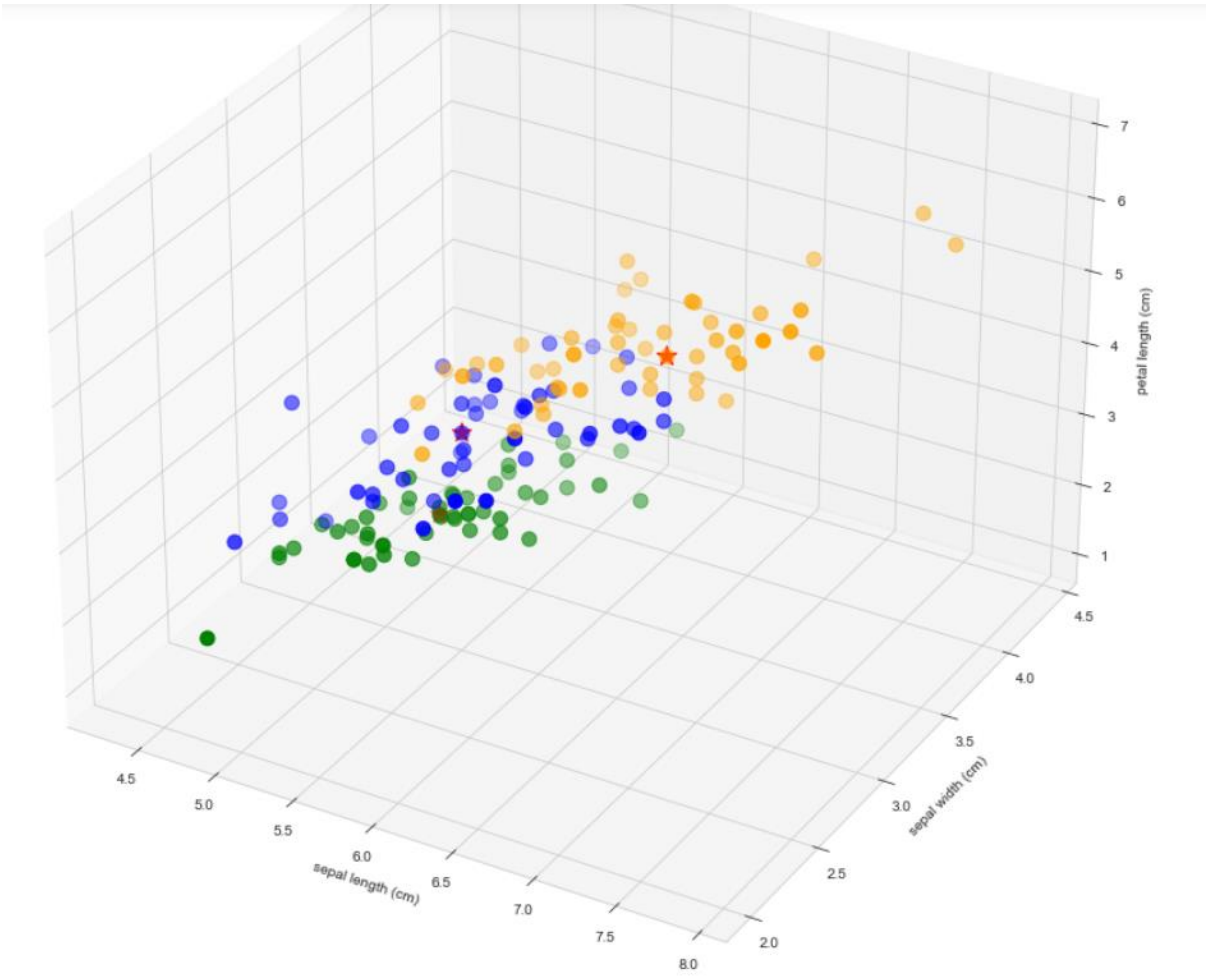


In [9]:
```
kmedoids = KMedoids(n_clusters=3,init = "random",random_state=77)
temp_kmedoids = kmedoids.fit_predict(df.replace(['setosa', 'versicolor', 'virginica'],[0,1,2]))
```

In [10]:
```
plt.subplots(figsize=(15,7))
plt.scatter(df.iloc[temp_kmedoids== 0,0],df.iloc[temp_kmedoids == 0,1],label="Setosa",color="blue",s=100)
plt.scatter(df.iloc[temp_kmedoids == 1,0],df.iloc[temp_kmedoids == 1,1],label="Versicolor",color="orange",s=100)
plt.scatter(df.iloc[temp_kmedoids == 2,0],df.iloc[temp_kmedoids == 2,1],label="Virginica",color="green",s=100)
plt.scatter(kmedoids.cluster_centers_[:,0],kmedoids.cluster_centers_[:,1],s=200,marker="*",color="red")
plt.show()
```



In [11]:
```
fig = plt.figure(figsize=(15,15))
ax = fig.add_subplot(projection="3d")
ax.scatter(df.iloc[temp_kmedoids == 0,0],df.iloc[temp_kmedoids == 0,1],df.iloc[temp_kmedoids == 0,2],label="Setosa",color="blue",
ax.scatter(df.iloc[temp_kmedoids == 1,0],df.iloc[temp_kmedoids == 1,1],df.iloc[temp_kmedoids == 1,2],label="Versicolor",color="or
ax.scatter(df.iloc[temp_kmedoids == 2,0],df.iloc[temp_kmedoids == 2,1],df.iloc[temp_kmedoids == 2,2],label="Virginica",color="gr
ax.scatter(kmedoids.cluster_centers_[:,0],kmedoids.cluster_centers_[:,1],kmedoids.cluster_centers_[:,2],s=200,marker="*",color="
ax.set_xlabel(df.columns[0])
ax.set_ylabel(df.columns[1])
ax.set_zlabel(df.columns[2])
plt.show()
```



Conclusion: We have successfully implemented K-means and K-medoids algorithms on iris data set using Weka Tool and Python.

EXPERIMENT NO. 8

Aim: To implement Hierarchical Clustering method.

Requirements: Windows O.S and Weka tool.

Problem Statement: To implement Hierarchical Clustering (Single Link, Complete Link and Average Link) on iris data set.

Theory:

Hierarchical Clustering:

Hierarchical clustering, also known as *hierarchical cluster analysis*, is an algorithm that groups similar objects into groups called *clusters*. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

If you want to do your own hierarchical cluster analysis, use the template below - just add your data!

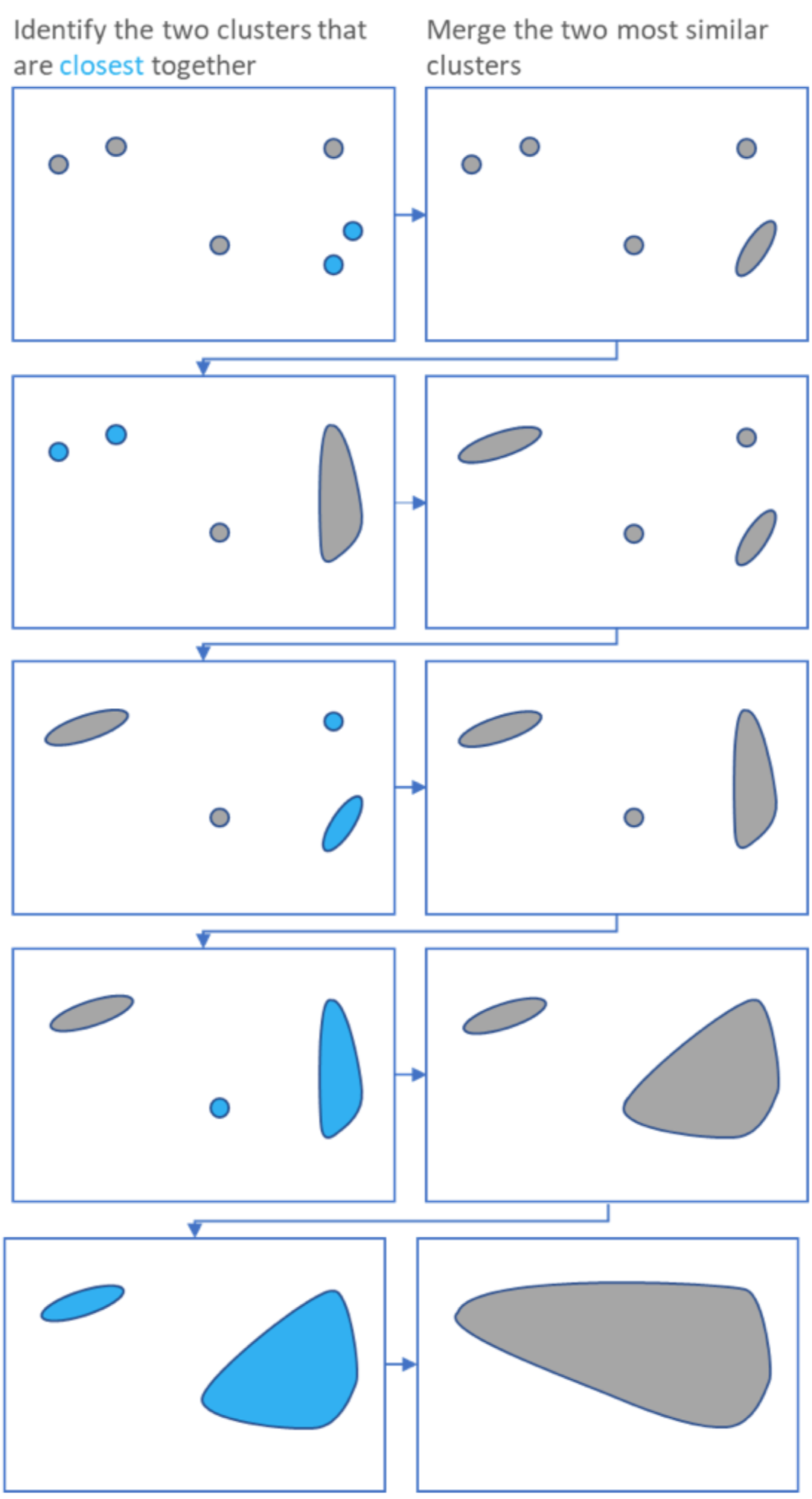
Required data

Hierarchical clustering can be performed with either a *distance matrix* or *raw data.* When raw data is provided, the software will automatically compute a distance matrix in the background. The distance matrix below shows the distance between six objects.

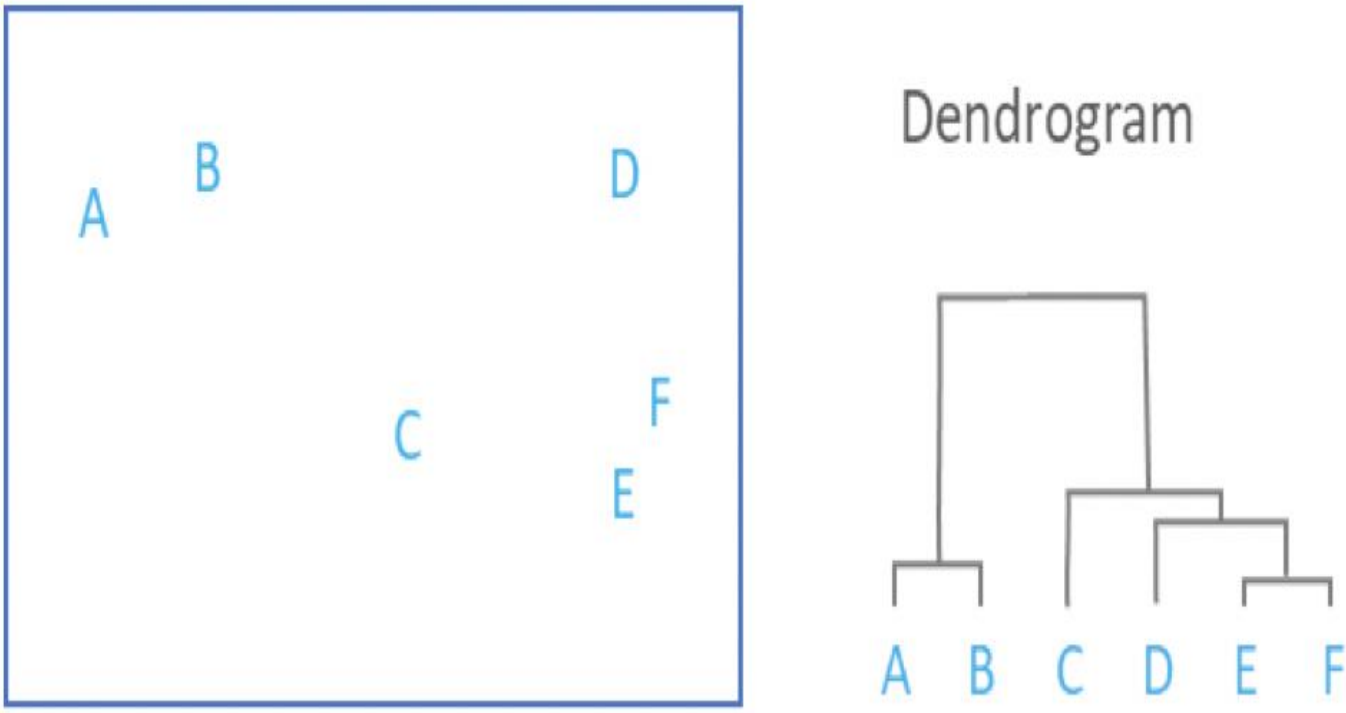
B	16				
C	47	37			
D	72	57	40		
E	77	65	30	31	
F	79	66	35	23	10
	A	B	C	D	E

How hierarchical clustering works

Hierarchical clustering starts by treating each observation as a separate cluster. Then, it repeatedly executes the following two steps: (1) identify the two clusters that are closest together, and (2) merge the two most similar clusters. This iterative process continues until all the clusters are merged together. This is illustrated in the diagrams below.


Identify the two clusters that are closest together
Merge the two most similar clusters

The main output of Hierarchical Clustering is a *dendrogram*, which shows the hierarchical relationship between the clusters:



Measures of distance (similarity)

In the example above, the *distance* between two clusters has been computed based on the length of the straight line drawn from one cluster to another. This is commonly referred to as the *Euclidean distance.* Many other *distance metrics* have been developed.

The choice of distance metric should be made based on theoretical concerns from the domain of study. That is, a distance metric needs to define similarity in a way that is sensible for the field of study. For example, if clustering crime sites in a city, city block distance may be appropriate. Or, better yet, the time taken to travel between each location. Where there is no theoretical justification for an alternative, the Euclidean should generally be preferred, as it is usually the appropriate measure of distance in the physical world.

Linkage Criteria

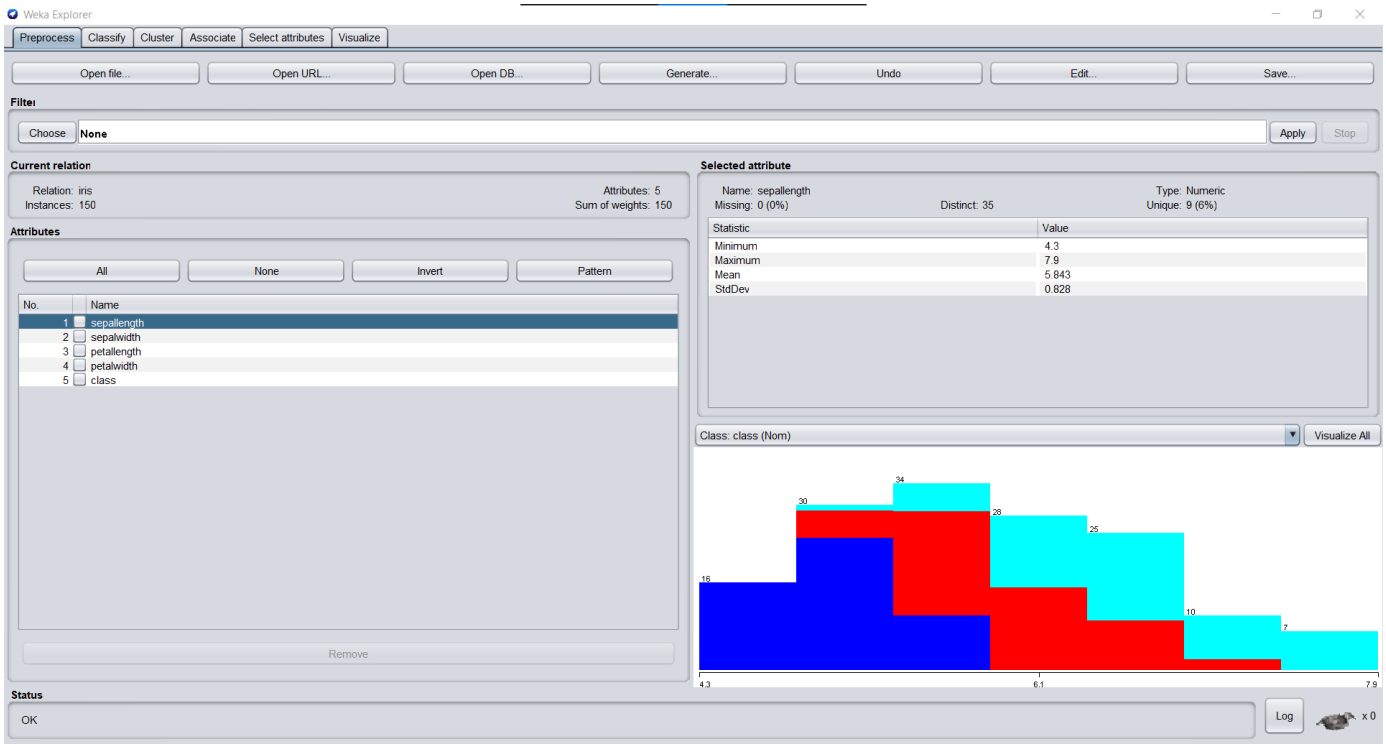
After selecting a distance metric, it is necessary to determine from where distance is computed. For example, it can be computed between the two most similar parts of a cluster (*single-linkage*), the two least similar bits of a cluster (*complete-linkage*), the center of the clusters (*mean* or *average-linkage*), or some other criterion. Many linkage criteria have been developed.

As with *distance metrics*, the choice of linkage criteria should be made based on theoretical considerations from the domain of application. A key theoretical issue is what causes variation. For example, in archeology, we expect variation to occur through innovation and natural

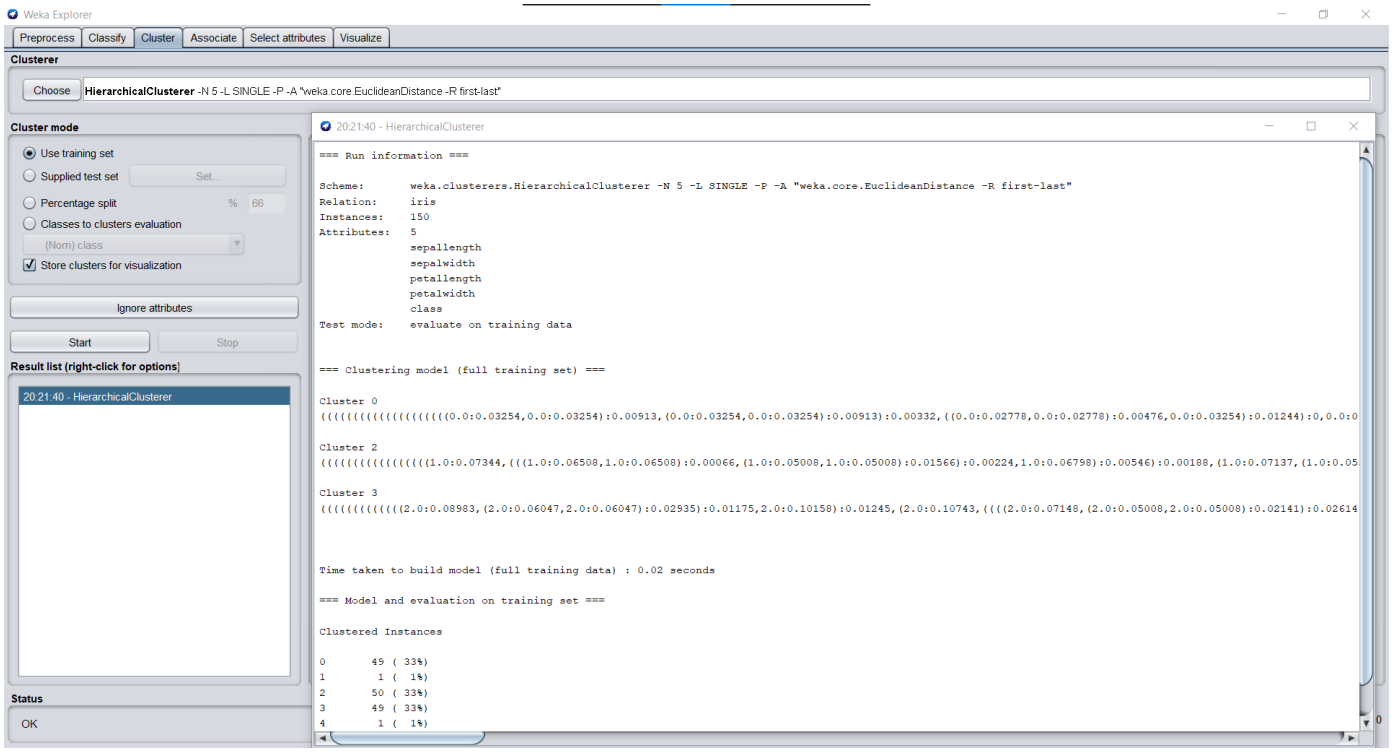
resources, so working out if two groups of artefacts are similar may make sense based on identifying the most similar members of the cluster.

Output:

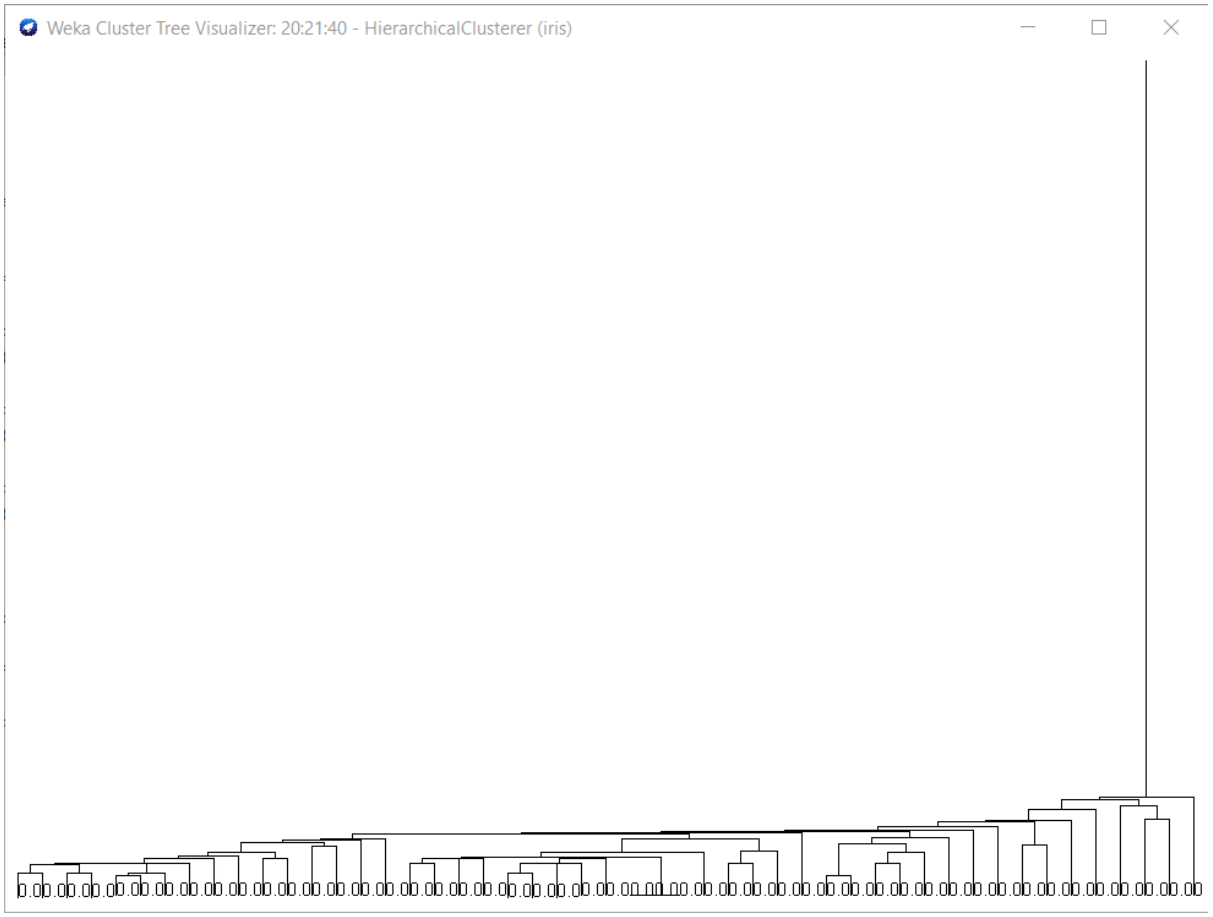
Iris Data set:



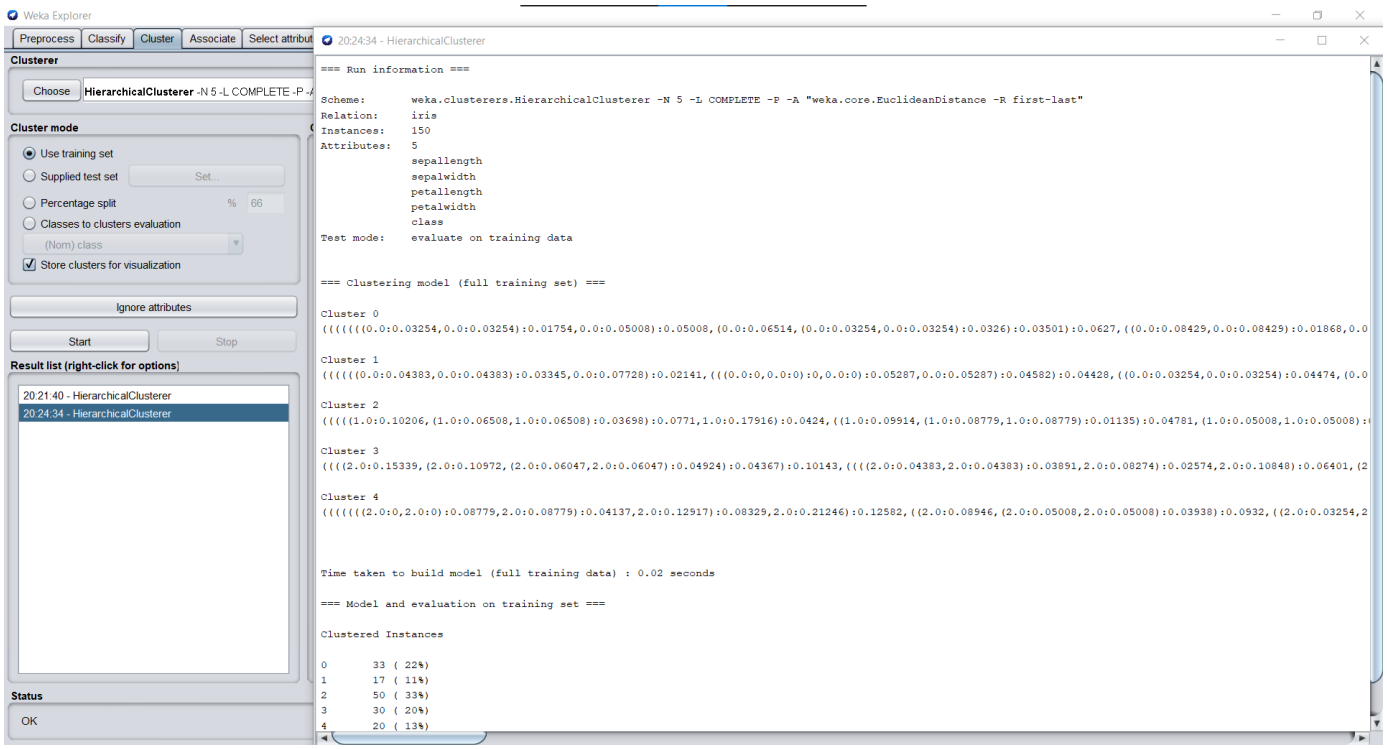
Single Link Clustering using 5 clusters and Euclidian Distance:



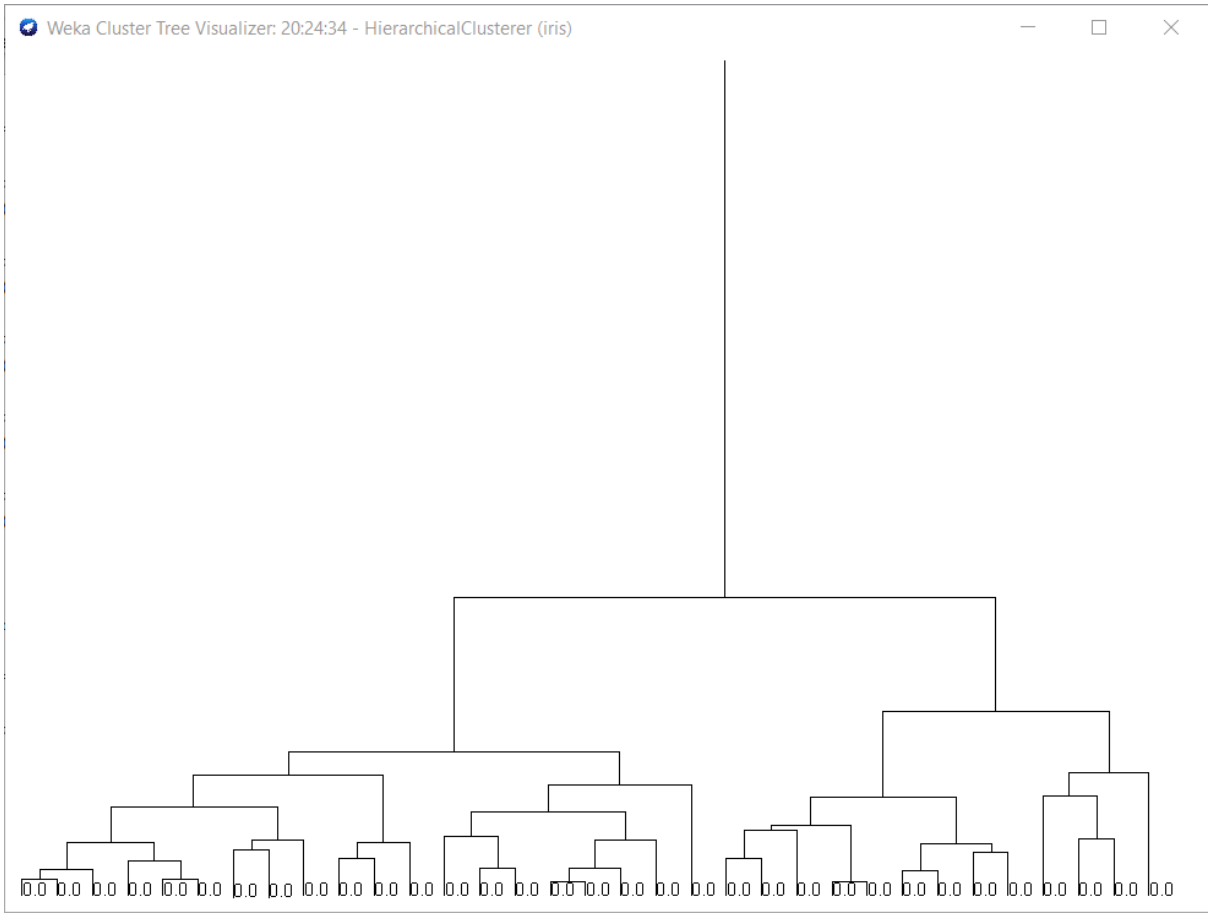
Single Link Dendrogram:



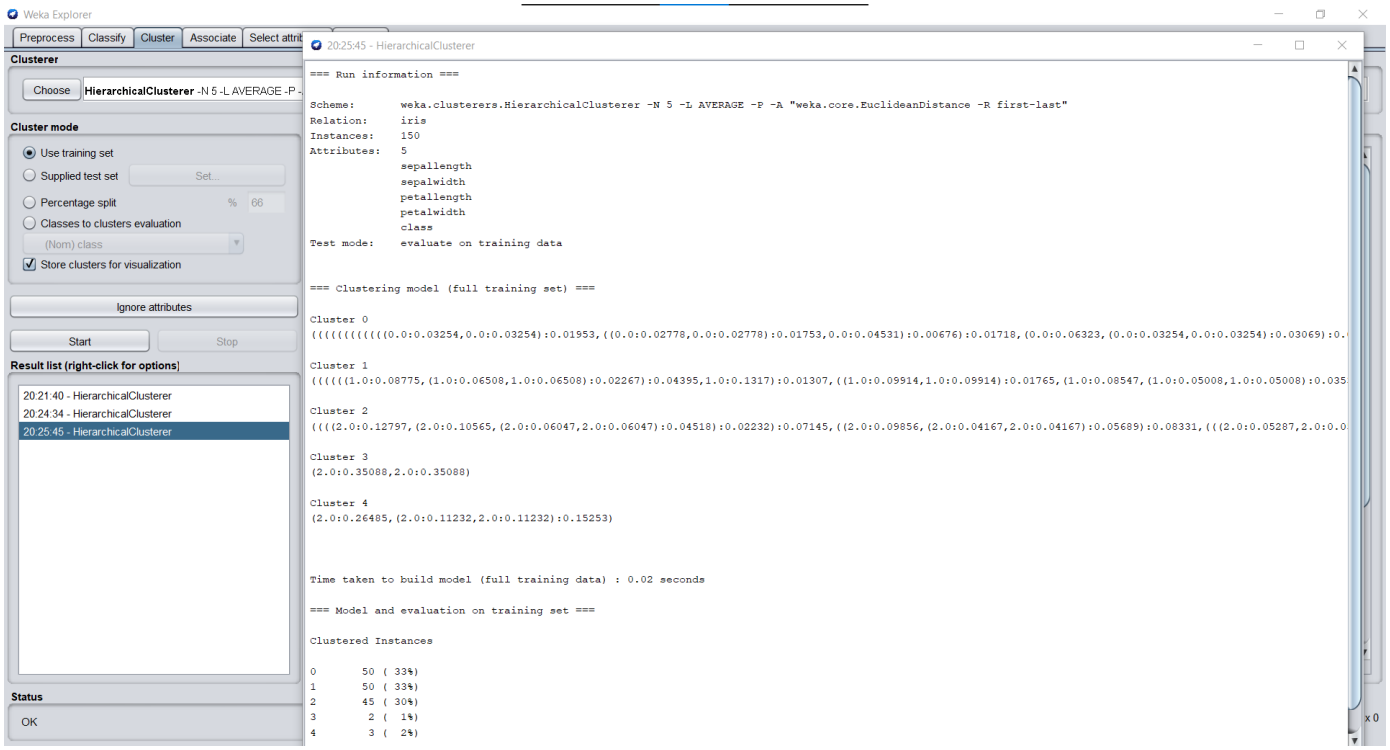
Complete Link Clustering using 5 clusters and Euclidian Distance:



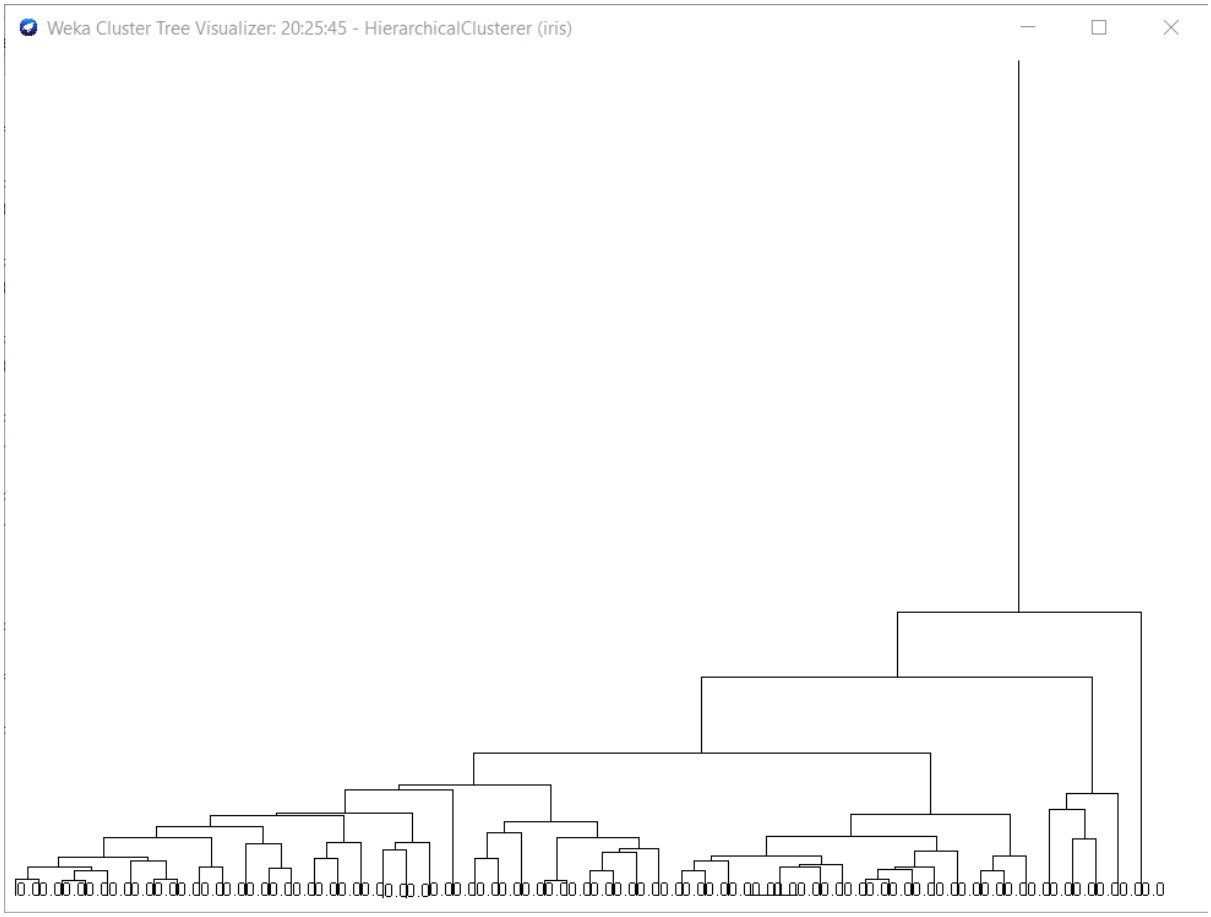
Complete Link Dendrogram:



Average Link Clustering using 5 clusters and Euclidian Distance:



Average Link Dendrogram:



Conclusions: We have successfully implemented Hierarchical Clustering (Single Link, Complete Link and Average Link) on iris data set using Weka Tool.

Experiment no. 9

Aim: To implement Association Rule Mining Algorithm (Apriori).

Requirements: Windows OS and Weka Tool.

Problem Statement: To implement Apriori algorithm on Supermarket data set using Weka Tool.

Theory:

Apriori Algorithm:

The Apriori algorithm uses frequent item sets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a breadth-first search and Hash Tree to calculate the item set associations efficiently. It is the iterative process for finding the frequent item sets from the large dataset.

This algorithm was given by the R. Agrawal and Srikant in the year 1994. It is mainly used for *market basket analysis* and helps to find those products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

What is Frequent Item set?

Frequent item sets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent item sets together, then individually A and B should also be the frequent item set.

Suppose there are the two transactions: A= {1,2,3,4,5}, and B= {2,3,7}, in these two transactions, 2 and 3 are the frequent item sets.

Steps for Apriori Algorithm:

Step-1: Determine the support of item sets in the transactional database, and select the minimum support and confidence.

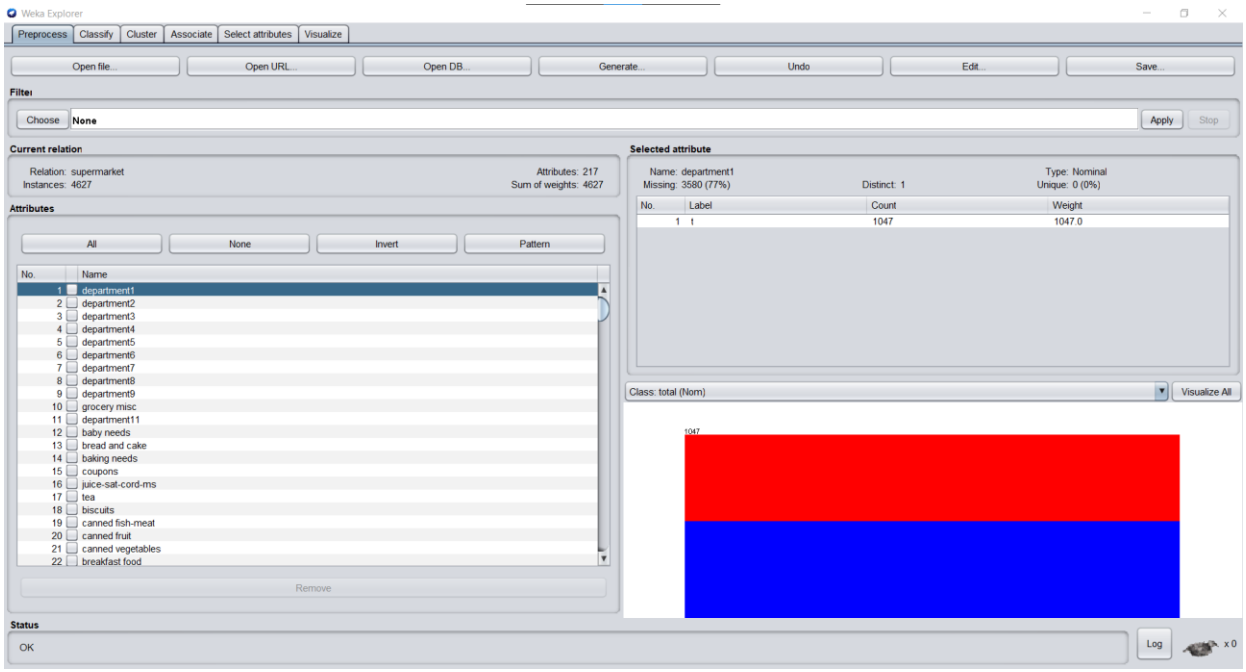
Step-2: Take all supports in the transaction with higher support value than the minimum or selected support value.

Step-3: Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.

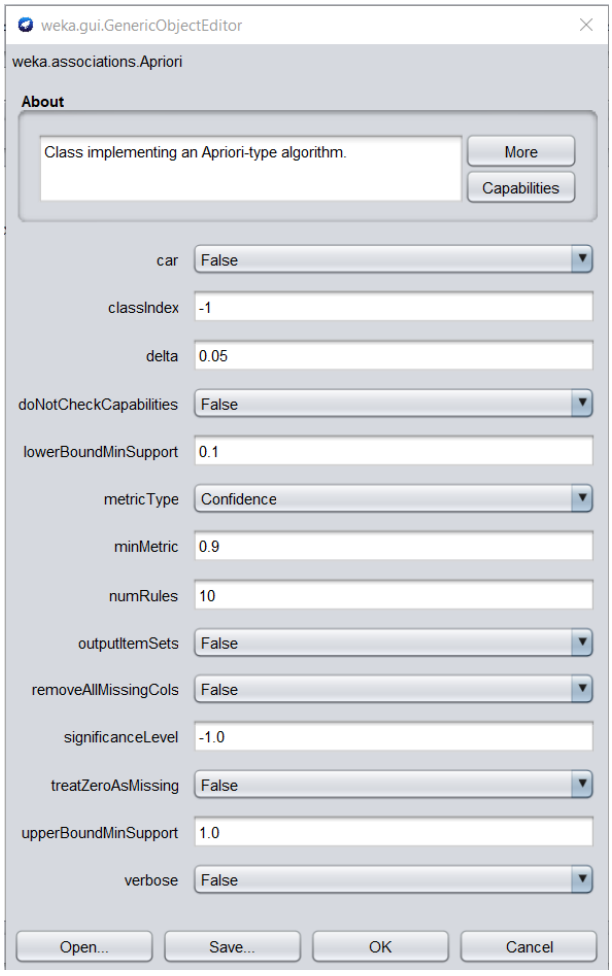
Step-4: Sort the rules as the decreasing order of lift.

Output**:**

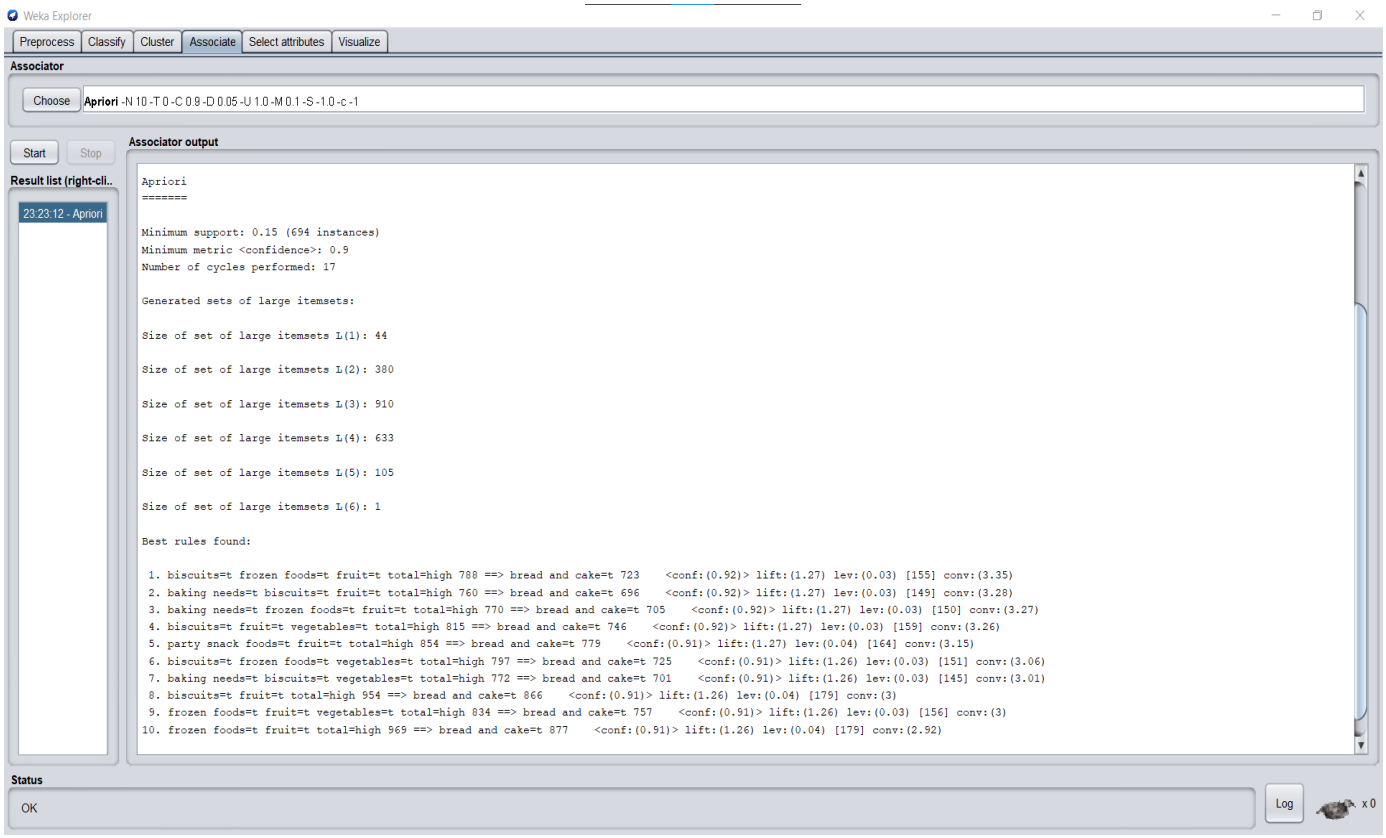
Supermarket data set:



Parameter of Apriori algorithm:



Result:



Conclusion: We have successfully implemented Apriori algorithm on supermarket data set using Weka Tool.

EXPERIMENT NO. 10

Aim: Implementation of Page rank/HITS algorithm

Requirements: Windows/MAC/Linux O.S, Compatible version of Python, Python libraries: Numpy, Matplotlib and Networkx.

Theory:

Page Rank:

The PageRank algorithm measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The underlying assumption roughly speaking is that a page is only as important as the pages that link to it.

PageRank is introduced in the original Google paper as a function that solves the following equation:

where,

- we assume that a page *A* has pages T_1 to T_n which point to it.
- *d* is a damping factor which can be set between 0 (inclusive) and 1 (exclusive). It is usually set to 0.85.
- *C(A)* is defined as the number of links going out of page *A*.

This equation is used to iteratively update a candidate solution and arrive at an approximate solution to the same equation.

Considerations:

There are some things to be aware of when using the PageRank algorithm:

- If there are no relationships from within a group of pages to outside the group, then the group is considered a spider trap.
- Rank sink can occur when a network of pages is forming an infinite cycle.
- Dead-ends occur when pages have no outgoing relationship.

Changing the damping factor can help with all the considerations above. It can be interpreted as a probability of a web surfer to sometimes jump to a random page and therefore not getting stuck in sinks.

Code:

```
import numpy as np

import operator

import networkx as nx

import matplotlib.pyplot as plt

import pylab
```

```
trusted_pages_ratio = 0.4
trusted_pages = []
maxer = 0
nodes_dict = {}
nodes = []
count = 0
beta = 0.85
# Reading form file and then saving the data to a dictionary
# of the form {NODE: [Set of nodes being pointed to by the "NODE"]}
with open("data.txt", "r") as data_file:
        for line in data_file:
                line_values = line.split("\t")
                a = int(line_values[0])
                b = int(line_values[1])
                if a > maxer:
                        maxer = a
                if b > maxer:
                        maxer = b
                if a not in nodes:
                        nodes.append(a)
                if b not in nodes:
                        nodes.append(b)
                if a not in nodes_dict:
                        nodes_dict[a] = [b]
                else:
                        nodes_dict[a].append(b)

# M is the Transition Matrix
# v is the matrix that defines the probability of the random surfer of being at any paricular node
M = np.zeros((maxer+1, maxer+1))
v = np.zeros(maxer + 1)
# Defining the Transition matrix
```

```
for from_node in nodes_dict:
        length = len(nodes_dict[from_node])
        fraction = 1/length
        for to_node in nodes_dict[from_node]:
                M[to_node][from_node] = fraction
# Defining initial v matrix
no_of_nodes = len(nodes)
fraction = 1 / no_of_nodes
for i in range(1, maxer + 1):
        if i in nodes:
                v[i] = fraction
# Definining the teleport matrix which takes care of the Dead ends and Spider traps
teleport = (1 - beta) * v
M = beta * M
# Carrying out the iterations until matrix v stops changing
while(1):
        v1 = np.dot(M, v) + teleport
        if np.array_equal(v1,v):
                break
        else:
                v = v1
                count += 1


print("No. of iterations required without considering TrustRank: " + str(count))
# Sorting nodes with respect to the final ranks of the nodes
page_rank_score = []
for i in range(1, len(v)):
        if v[i] != 0:
                page_rank_score.append([i, v[i]])
sorted_page_rank_score = sorted(page_rank_score, key = operator.itemgetter(1), reverse = True)
# Incorporating the concept of Trust rank
```

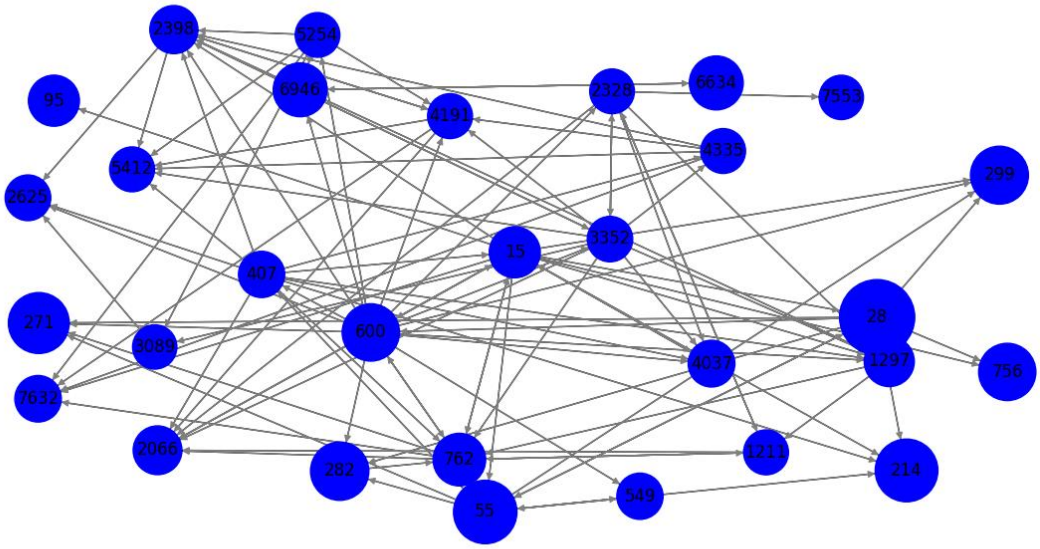
```
no_of_trusted_pages = int(trusted_pages_ratio * len(sorted_page_rank_score))
trusted_pages = [page_info[0] for i, page_info in zip(range(0, no_of_trusted_pages), sorted_page_rank_score)]
fraction = 1 / no_of_trusted_pages
# Defining initial v matrix
v = np.zeros(maxer + 1)
for i in range(1, maxer + 1):
        if i in trusted_pages:
                v[i] = fraction
# Defining teleport set
teleport = (1 - beta) * v
count = 0
# Carrying out the iterations until matrix v stops changing
while(1):
        v1 = np.dot(M, v) + teleport
        if np.array_equal(v1,v):
                break
        else:
                v = v1
                count += 1
print("No. of iterations required after considering TrustRank: " + str(count))
# Sorting nodes with respect to the final ranks of the nodes
page_rank_score_after_trustrank = []
for i in range(1, len(v)):
        if v[i] != 0:
                page_rank_score_after_trustrank.append([i, v[i]])

sorted_page_rank_score_after_trustrank = sorted(page_rank_score_after_trustrank, key = operator.itemgetter(1), reverse = True)
# Plotting the top 30 nodes
nodes_for_graph = []
edge_list_for_graph = []
G = nx.DiGraph()
```

```
for i, page_info in zip(range(30), sorted_page_rank_score_after_trustrank):
        nodes_for_graph.append(page_info[0])
        # print(page_info[1])
print(nodes_for_graph)
for node in nodes_for_graph:
        if node in nodes_dict:
                for page in nodes_dict[node]:
                        if page in nodes_for_graph:
                                edge_list_for_graph.append([node, page])
G.add_edges_from(edge_list_for_graph)
edge_colors = ['grey' for edge in G.edges]
final_node_size = [1950000 * page_info[1] for i, page_info in zip(range(30), sorted_page_rank_score_after_trustrank)]
pos  = nx.spring_layout(G, k = 1, iterations = 20)
nx.draw_networkx_edges(G,pos)
nx.draw(G, pos, node_size = final_node_size, node_color = 'Blue', edge_color = edge_colors,edge_cmap=plt.cm.Reds, with_labels = True)
pylab.show()
```

Output:

```
No. of iterations required without considering TrustRank: 82
No. of iterations required after considering TrustRank: 83
[28, 2625, 6634, 214, 271, 282, 15, 55, 299, 2398, 4037, 5412, 95, 1297, 4335, 2066, 407, 600, 762, 7553, 2328, 3089, 3352, 7632, 4191, 6946, 1211, 5254, 549, 756]
PS C:\Users\adnan\OneDrive\Desktop\College\sem5\DWM\Practical 10\PageRank-Implementation>
```



Conclusion: We have successfully understand the concept of page ranking algorithm and implemented in python.

Subject: D.W.M Sem: V

Q.» Explain different OLAP operations, with suitable example.

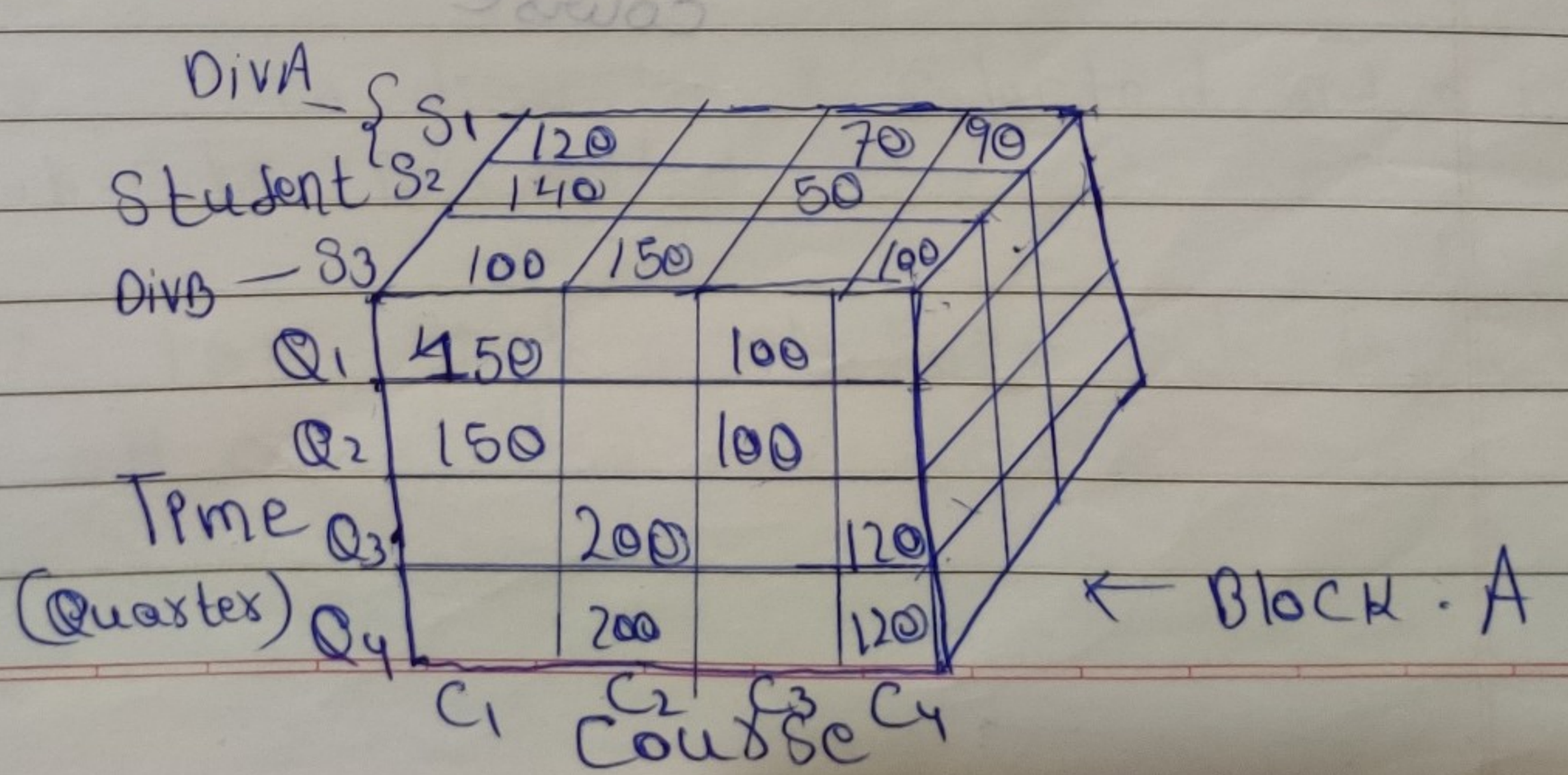
Ans) (i) Online Analytical Processing (OLAP) is a Category software that allows users to analyze information from multiple database systems at the same time. It is a technology that enables analysts to extract and view business data from different point of view.

(ii) The OLAP Cube consists of numeric facts called measures which are categorized by dimensions. OLAP cube is also called the hyper cube.

(iii) Four types of analytical OLAP operations are: 1. Roll-up 2. Drill down 3. Slice and dice 4. Pivot.

(iv) Let, us understand these operations through example of college database having dimensions 1. Course 2. Student 3. Time and fact (measure): Aggregate marks.

(v) Hyper-Cube:-

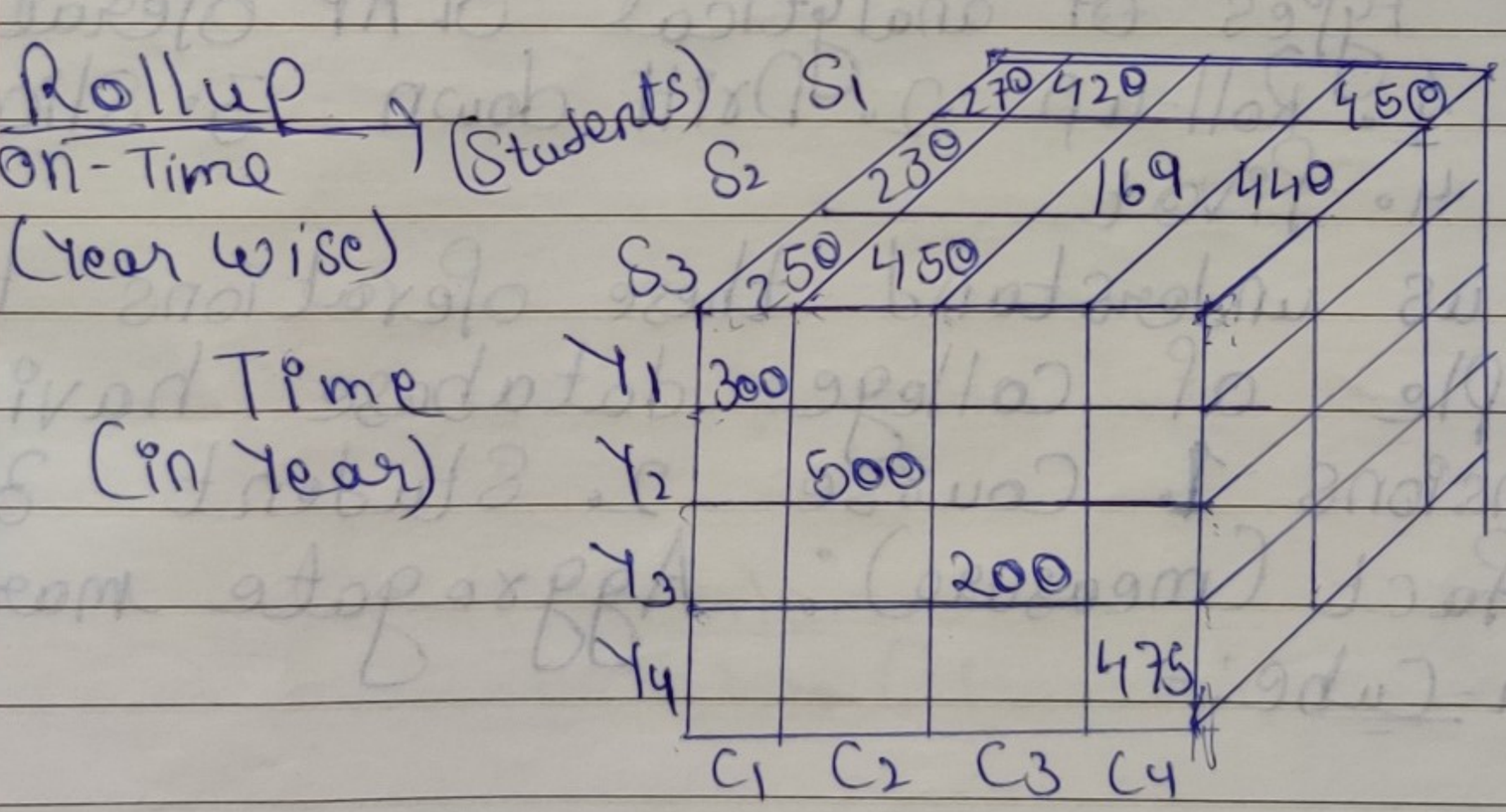


In Block A we have dimension Stundetnt at level individual Student, Time in Quarter and courses at deparment level. to columns and rows i.e each block fill with aggregate marks or NA values (Empty blocks).

1 Roll-up:- Roll-up is also known as "Consolidation" or "aggregation". The Roll-up operation can be performed in 2-ways:-

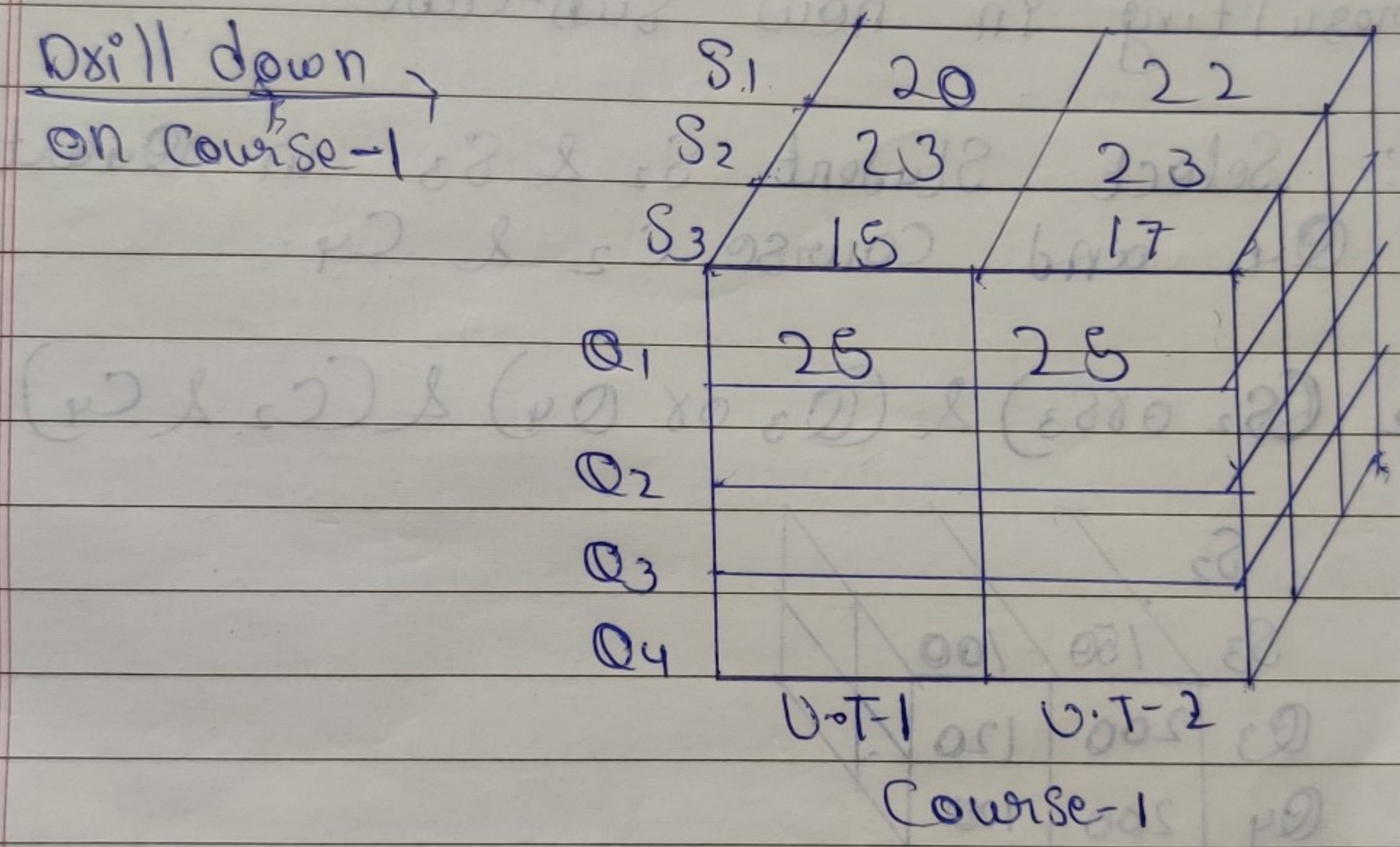
(a) Reducing dimensions (b) climbing up concept hierarchy

e.g:- Get the year wise marks for all Students is roll-up as Semester wise marks are added to get year wise marks & it's one level up hierarchy of dimension:- Time.



2) Drill-down:- In drill-down data is fragmented into smaller parts. It is the opposite of the rollup process. The drill-down operation can be performed in two ways:-
(a) Moving down the concept hierarchy.
(b) Increasing dimension.

e.g:- Drill down on courses to get unit test marks (Hatif Syllabus performance)



If we drill-down on each course, dimensions will be double of original dimension ~~of~~
So, we dice only Course-1.

3) (a) Slice:- One dimension is selected, and a new sub-cube is created.

e.g:- For Single students get the marks for all courses and all Semester.

Slice on Student S_1

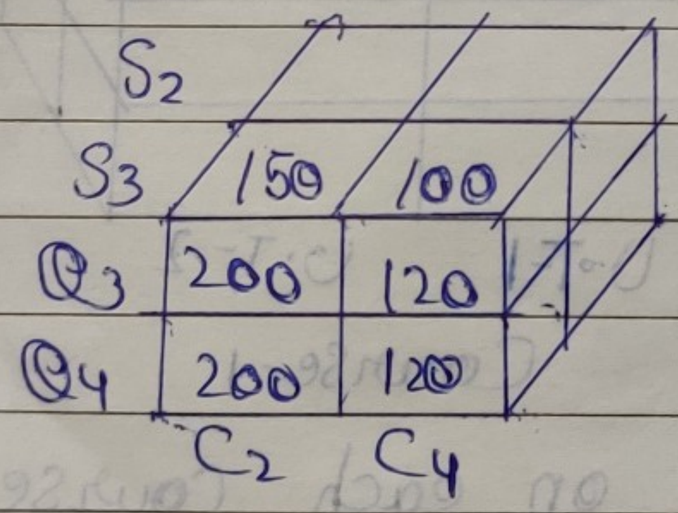
Q_1	120		70	
Q_2	120		~~70~~	
Q_3				90
Q_4				90
	C_1	C_2	C_3	C_4

C_2 is empty because S_1 didn't allote for it.

3) (b) Dice :- Two or more dimension is selected, resulting in new sub-cube.

e.g:- Select Student S_2 & S_3 in Quarter Q_3 & Q_4 and course C_2 & C_4.

Dice $(S_2$ or $S_3)$ & $(Q_3$ or $Q_4)$ & $(C_2$ & $C_4)$:-



4) Pivot :- In Pivot, you rotate the data axes to provide a substitue presentation of data. Analogous to matrix transpose.

e.g:- Pivoting result obtained in Slice.

C_1	120	120		
C_2				
C_3	70	70		
C_4			90	90
	Q_1	Q_2	Q_3	Q_4

Q.» Consider a data warehouse for a hospital where there are 3-dimensions namely
(a) Doctor (b) Patient (c) Time and
2 measures (i) Count (ii) Charge. Where,
Charge is the fee that the doctor
charges a patient for a visit.

(i) Draw Star and Snow-flake Schema.

(Ans) There are four tables:- 3-Dimensions tables
+ 1- fact table with 2 measures.

Dimension Table:-

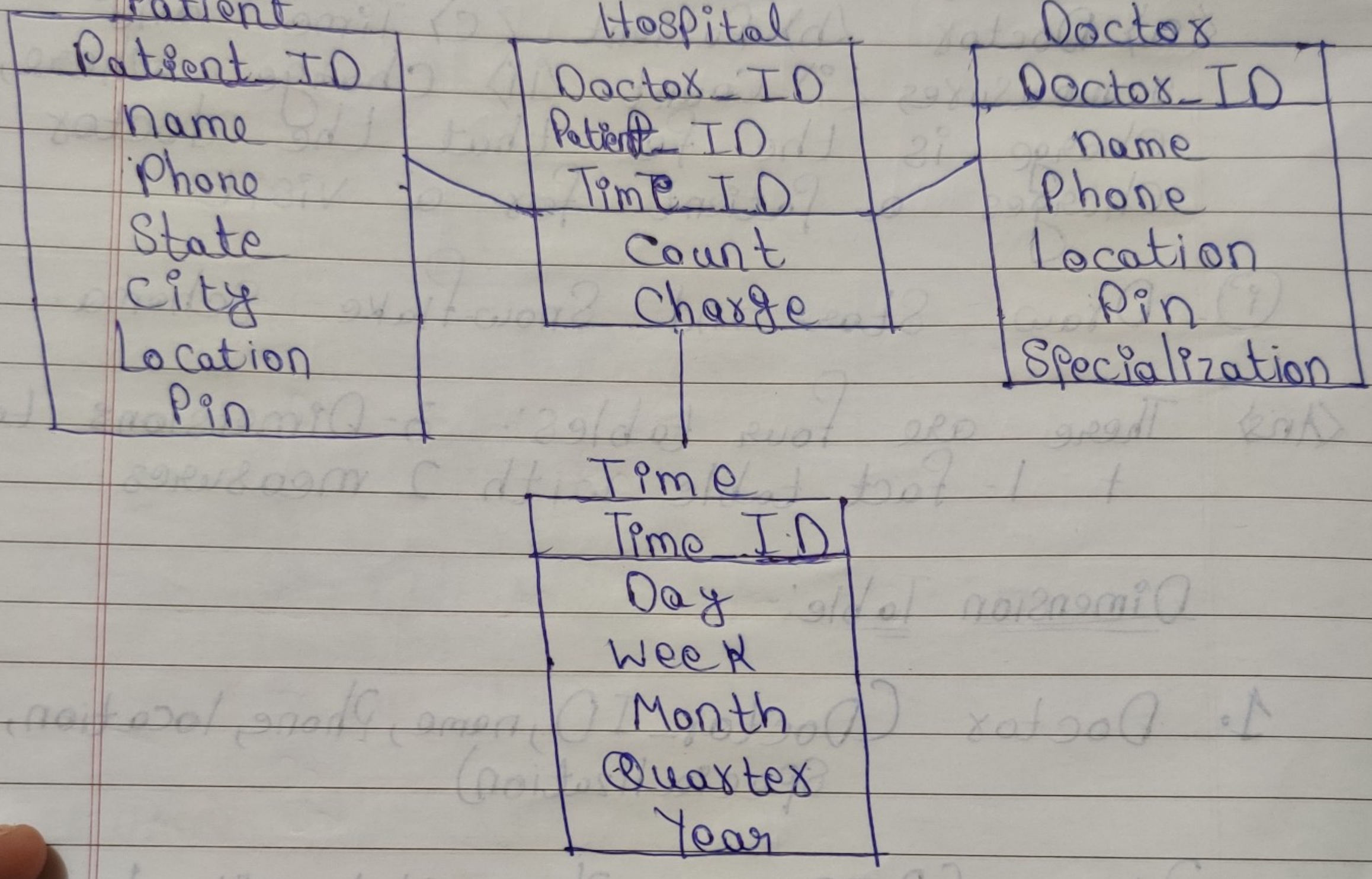
1. Doctor (Doctor_ID, name, Phone, location, Pin
Specialization)

2. Patient (P_ID, name, Phone, State, city, location,
Pin)

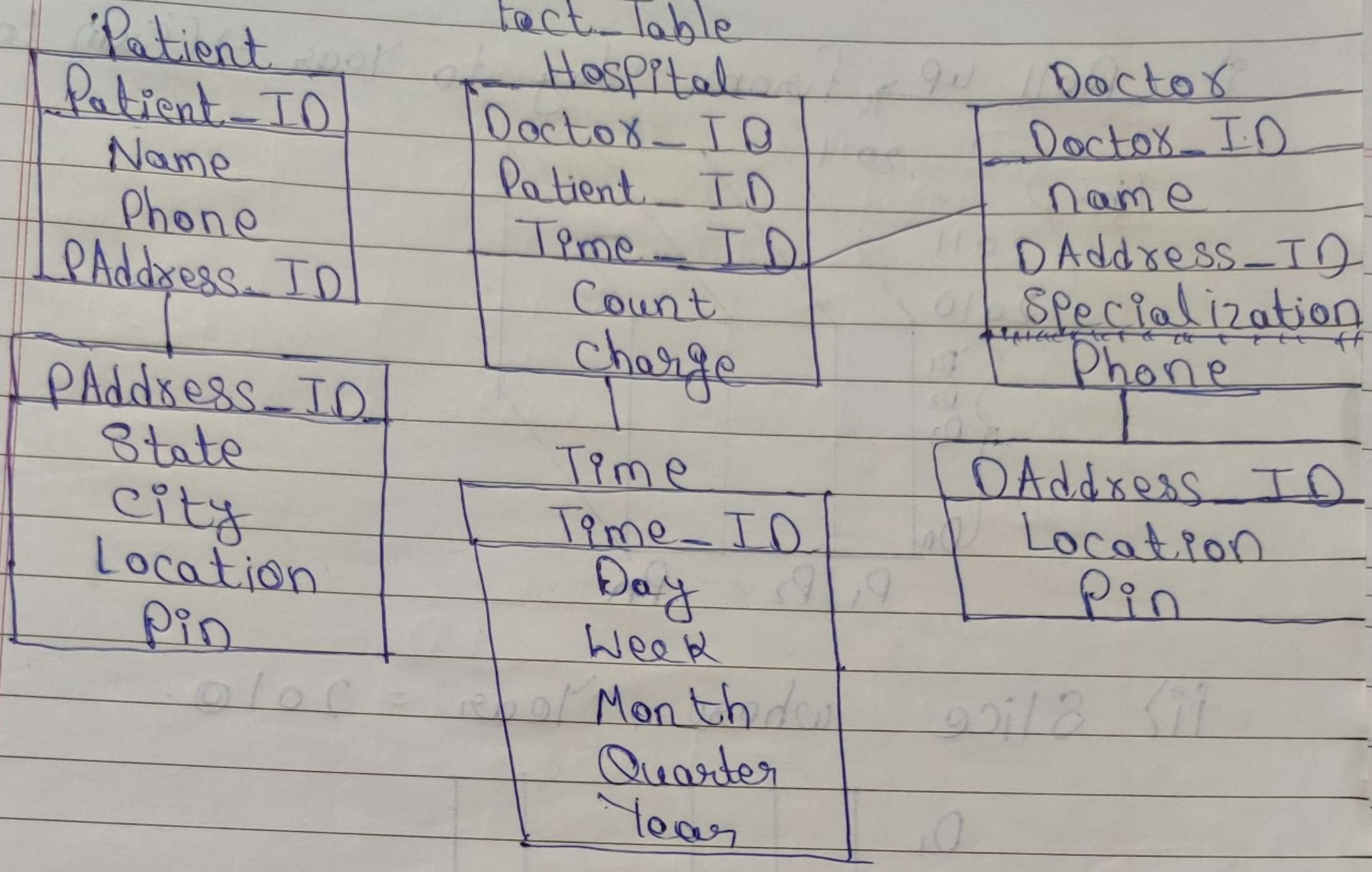
3. Time (T_ID, day, month, week, Quarter, Year)

Fact-Table:-

4. Fact_Table (Doctor_ID, P_ID, T_ID, Count, Charge)


Fact_Table
Hospital
Doctor_ID
Patient_ID
Time_ID
Count
Charge
Patient
Patient_ID
name
Phone
State
City
Location
Pin
Doctor
Doctor_ID
name
Phone
Location
Pin
Specialization
Time
Time_ID
Day
Week
Month
Quarter
Year

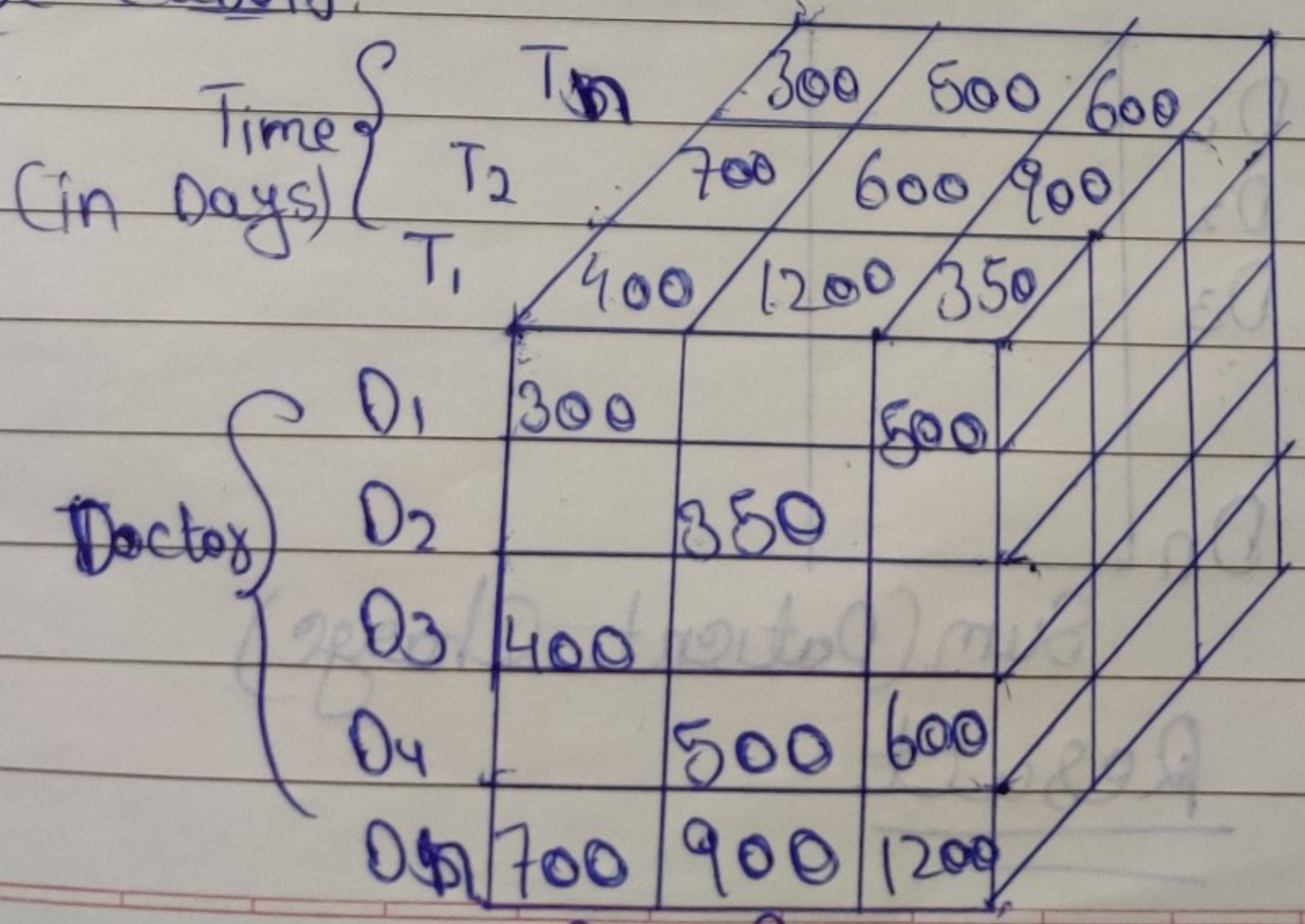
Star - Schema



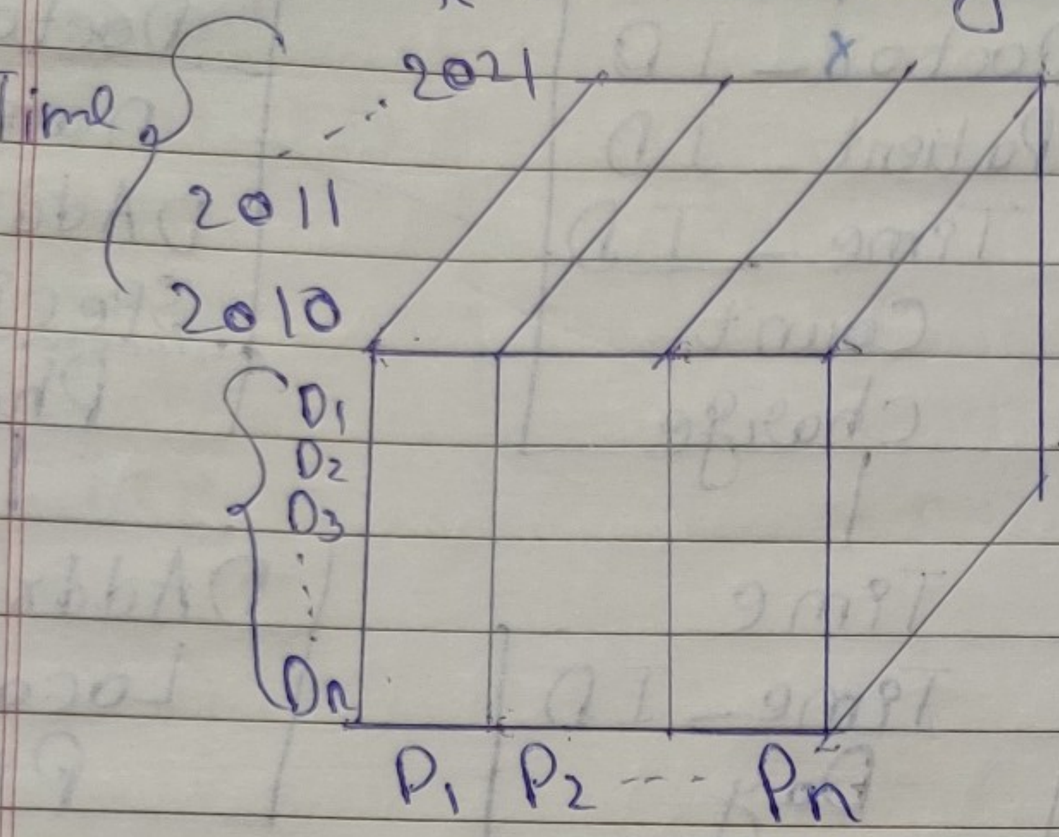
Snow-flake schema.

(ii) Starting with based cuboid [Day, Doctor, Patient], what specific OLAP operations should be performed in order to list the total fee collected by each doctor in 2010.

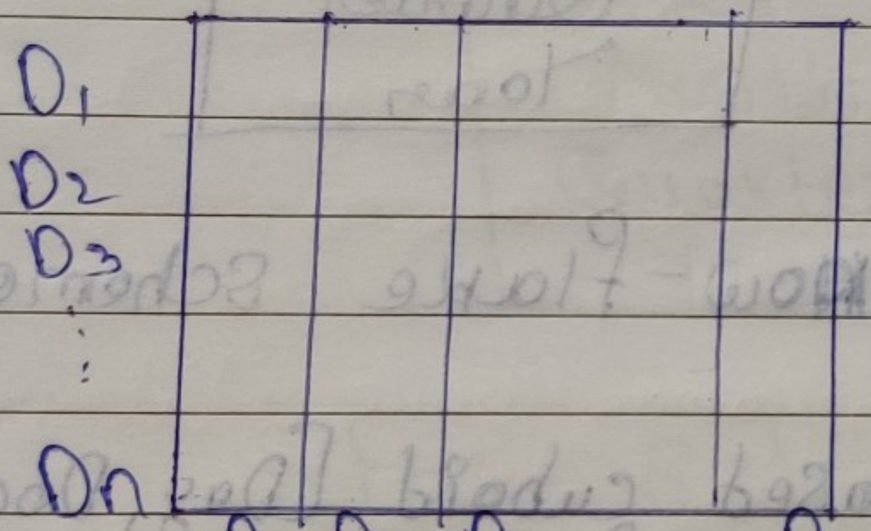
Ans) Base Cuboid:-



i) Roll up → from day to Year → (Sum to Year)

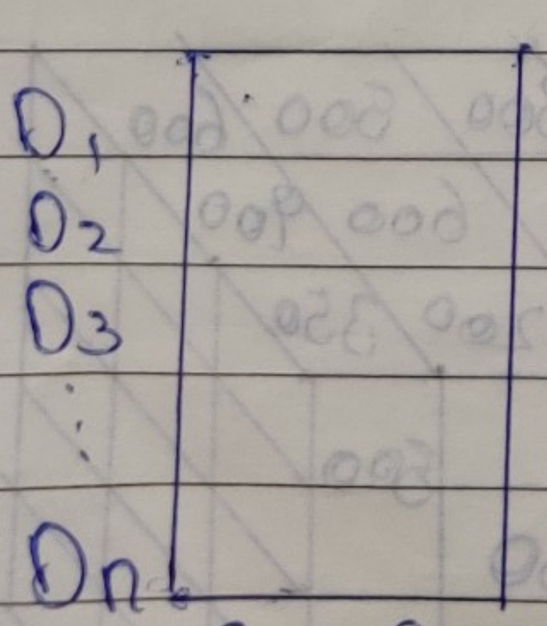


ii) Slice where Year = 2010.



Year = 2010

(iii) Roll-up on Patient group by doctor.



Sum(Patient_Charge)

Result

iii) To obtain the same list write an SQL query the data are stored in R.D.B with the schema Fee. [Day, Month, Year, Doctor, Hospital, Patient, Count, Charge]

Ans) SELECT Doctor, SUM(Charge) FROM Fee WHERE Year = 2010 GROUP BY Doctor.

Q.) Suppose that the data for analysis includes the attribute Age, the Age values for the data tuples are (in increasing order)
[13, 15, 16, 16, 19, 20, 23, 29, 33, 41, 44, 53, 62, 69, 72] use min_max normalization to transform the value 45 for ~~Charge~~ Age onto the range [0.0, 1.0]

Ans) $v' = \frac{(v - minage)}{maxage - minage} [new\_maxage - new\_min_{Age}] + new\_minage.$

$v = 45$, $minage = 13$, $maxage = 72$
$new\_maxage = 1.0$, $new\_minage = 0.0$

$\therefore v' = \frac{(45 - 13)}{72 - 13} [1.0 - 0.0] + 0.0$

$\therefore$ $v' = \frac{16}{31} = 0.51613$

Q.» Describe the steps involved in data mining when viewed as a process of Knowledge discovery.

Ans Steps involved in data mining when viewed as process of Knowledge discovery:-

1.) Data cleaning :- Data from multiple source (DataBase, flat-files, and different form) are extracted and clean to remove noise and inconsitent data.

2.) Data integration:- Since, data is extracted from multiple source they may be combined to provide useful information Domain Knowledge is necessary for proper integration. Generally, stored in R.D.B.

3.) Data Selection:- After cleaning & integration data relevant to the analysis task are retrived from the data ware-house. e.g:- Feature Sub-set Selection.

4.) Data transformation:- Data are transformed and consolidated into forms appropriate for mining by performing Data reduction, different transformation technique & aggregation.

5) Data-mining:- It is an essential process where Intelligent methods are applied, algorithms from various different fields such as Machine learning, Image processing and Natural language processing are used to extract novel data patterns.

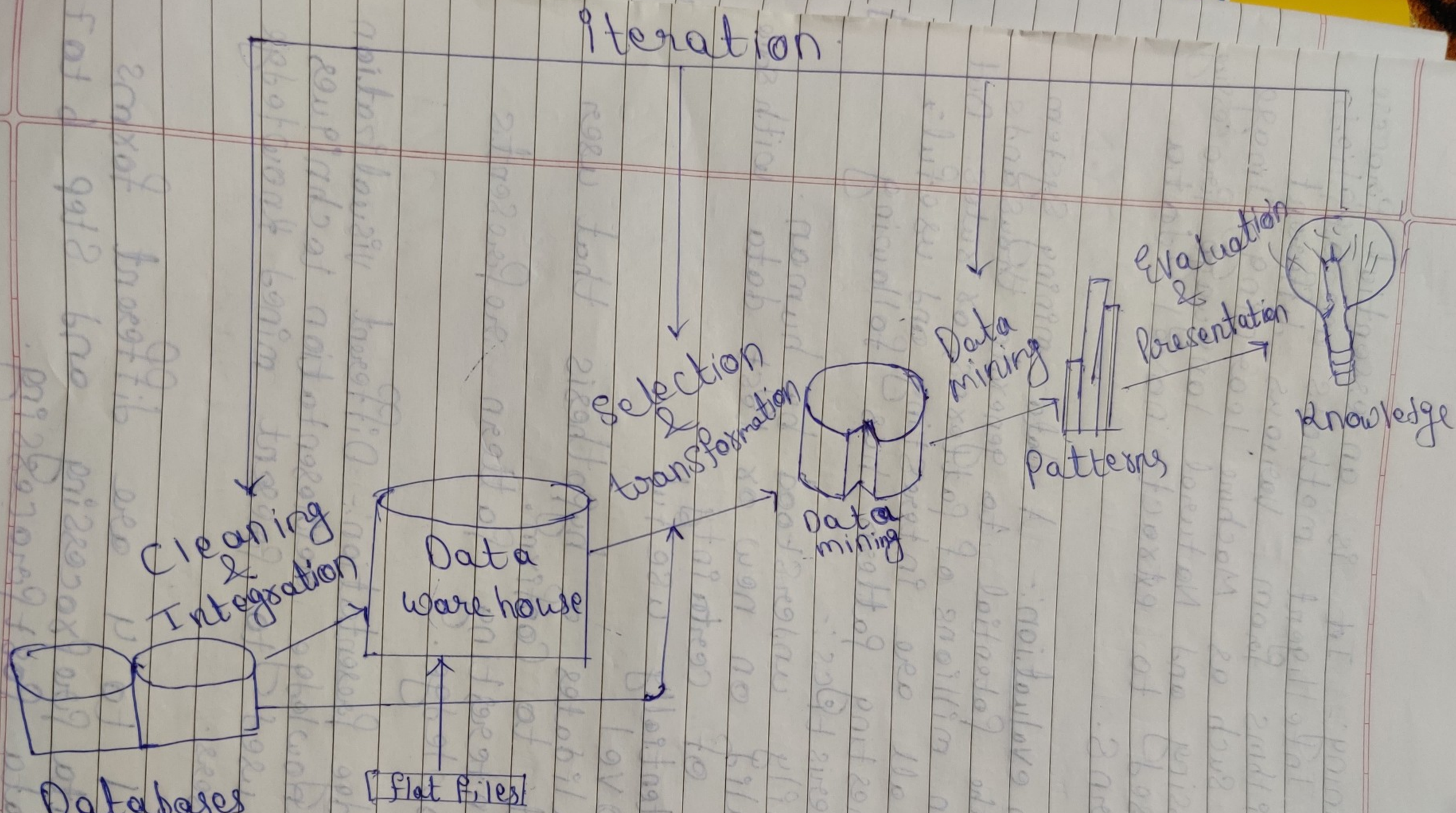
6) Pattern evaluation:- A data mining system has the potential to generate thousands or even millions of patterns, or rules. But Not all are interesting and useful; Interesting pattern has following Charateristics:-

(a) Easily understood by human.
(b) Valid on new or test data with some degree of certainty
(c) Potentially useful
(d) Novel
(e) Validates a hypothesis that user Sought to Confirm.

An interesting pattern represents Knowledge.

7) Knowledge presentation:- Different visualization and Knowledge representation techniques are used to present mined Knowledge to users.

Step 1 to 4 are different forms of data preprocessing and Step 6 to 7 is data postprocessing.



Data mining as a step in the process of knowledge discovery.

<<Q.>> (i) Why is tree pruning is useful in decision tree induction? What is a drawback of using a separate set of tuples to evaluate pruning. (ii) Given a decision tree you have the option of (a) Converting D.T to results & then pruning the resulting rules. (b) Pruning the D.T & then converting pruned tree to rules, what advantage does (a) have over (b)

Ans (1) (I) The decision tree built may overfit the training data. There could be too many branches some of which may reflect anomalies in training data due to noise or outliers.

(2) Tree pruning addresses this issue of overfitting the data by removing the least reliable branches (using statistical measures)

(3) This generally results in more compact and reliable decision tree that is faster and more accurate in its classification of data.

(4) The drawback of using separate set of tuples to evaluate pruning is that it may not be representative of training tuples used to create the original decision tree.

(5) If the separate set of tuples are skewed then using them to evaluate the pruned tree would not be a good indicator of the pruned tree's classification accuracy.

(6) Further more, using separate set of tuples to evaluate pruning means there are less tuples to use for creation and testing of the tree.

(ii) once a decision tree has been constructed, it is a simple matter to convert it into an equivalent set of rules.
Converting a decision tree to rules before has three main advantages [advantages of (a) over (b)]:
1. Converting to rules allows distinguishing among the different contexts in which a decision node is used.
(a) Since each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path.
(b) In constrast, if the tree itself were pruned, then only two choices would be: (1) Remove the decision node completely or (2) Retain it in its original form.
2. Converting to rule removes the distinction between attribute tests that occur near the root of the tree & those that occur near the leaves. We thus avoid messy book keeping issues such as how to reorganize the tree if the root node is pruned while retaining part of subtree below this test.
3. Converting to rules improves readability. Rules are often easier for human to understand.
To generate rules, trace each path in the decision tree, from root node to leaf node, recording

the test outcomes as antecedents and the leaf-node classification as the consequent.

Q) A database has 5 transactions, let min. Support = 60% & min. confidence = 80%

TID	Item_Bought
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I, E}

(a) Find all frequent item set using Apriori algorithm.

(b) List all the strong association rules matching the following, where 'x' is a variable representing Customers & item i denotes variable representing items {e.

$$\forall x \in \text{transaction}, buys(x, item_i) \wedge buys(x, item_j) \Rightarrow buy(x, item_k)$$

Soln: 1 frequent item set:

A → 1 → 1/5 < 60% X
C → 2 → 2/5 < 60% X
D → 1 → 1/5 < 60% X
E → 4 → 4/5 >= 60% ✓
K → 5 → 5/5 >= 60% ✓
I → 1 → 1/5 < 60% X
M → 3 → 3/5 >= 60% ✓
N → 2 → 2/5 < 60% X
O → 4 → 4/5 >= 60% ✓
U → 1 → 1/5 < 60% X

Y → 3 → 3/5 >= 60% ✓

1st frequent item-set = {E, K, M, O, Y}

2nd frequent item-set:

{E, K} → 4 → 4/5 ✓

{E, M} → 2 → 2/5 X

{E, O} → 3 → 3/5 ✓

{E, Y} → 2 → 2/5 X

{K, M} → 3 → 3/5 ✓

{K, O} → 3 → 3/5 ✓

{K, Y} → 3 → 3/5 ✓

{M, O} → 1 → 1/5 X

{M, Y} → 2 → 2/5 X

{O, Y} → 2 → 2/5 X

2nd frequent item-set => {{E, K}, {E, O}, {K, M}, {K, O}, {K, Y}, ~~{O, Y}~~}

3rd Frequent item-set:

{E, K, O} → 3 → 3/8 ✓

{E, K, M} → 1 → 1/8 ×

{E, K, Y} → 2 → 2/8 ×

{K, M, O} → 0 ×

{K, M, Y} → 2 → 2/8 ×

{K, O, Y} → 2 → 2/8 ×

3rd - Frequent item set = {E, K, O}

Association Rule	Support	Confidence
{E, K} → {O}	3	3/4 => 75% ×
{O} → {E, K}	3	3/4 ×
{E, O} → {K}	3	3/3 => 100% ✓
{K} → {E, O}	3	3/5 => 60% ×
{O, K} → {E}	3	3/3 => 100% ✓
{E} → {O, K}	3	3/4 => 75% ×

b) List of association rules that satisfies min. confidence =>

$\langle E \wedge O \rangle \longrightarrow K$

$\langle O \wedge K \rangle \longrightarrow E$