Experiment-1:

1. **Creation of a datawarehouse.**

A. Build Data Warehouse/Data Mart (using open source tools like Pentaho Data Integration Tool, Pentaho Business Analytics; or other data warehouse tools like Microsoft-SSIS,Informatica,Business Objects,etc.,)

A.(i) Identify source tables and populate sample data.

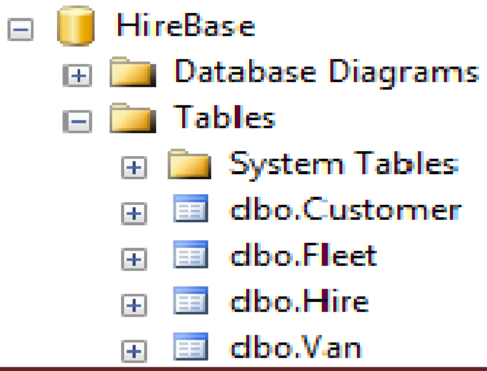
The data warehouse contains 4 tables:

1. Date dimension: contains every single date from 2006 to 2016.
2. Customer dimension: contains 100 customers. To be simple we'll make it type 1 so we don't create a new row f**o**r each change.
3. Van dimension: contains 20 vans. To be simple we'll make it type 1 so **w**e don't create a new row for each change.
4. Hire fact table: contains **1**000 hire transactions since 1st Jan 2011. It is a **d**aily snapshot fact table so that every day w**e** insert 1000 rows into this fact table. So over t**i**me we can track the changes of total bill, **v**an charges, satnav income, etc.

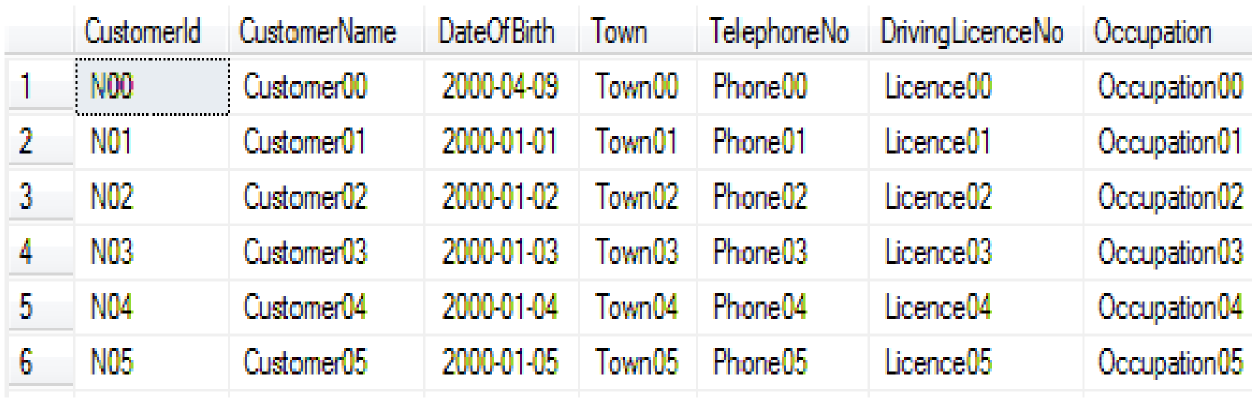
Create the source tables and populate them

So now we are going to create t**h**e 3 tables in HireBase database: Customer, Va**n, a**nd Hire. Then we populate them.

First I'll show you how it l**o**oks when it's done:

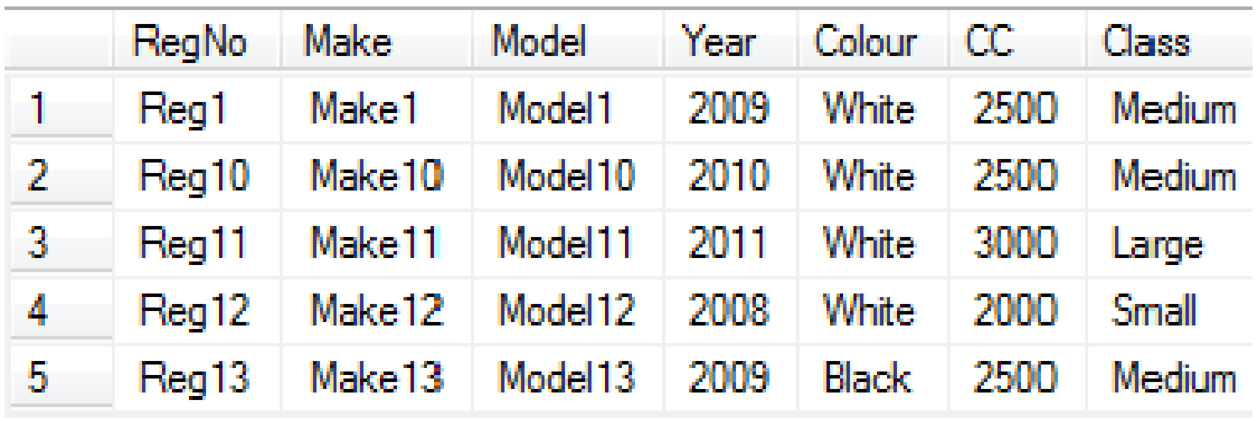


Customer table:



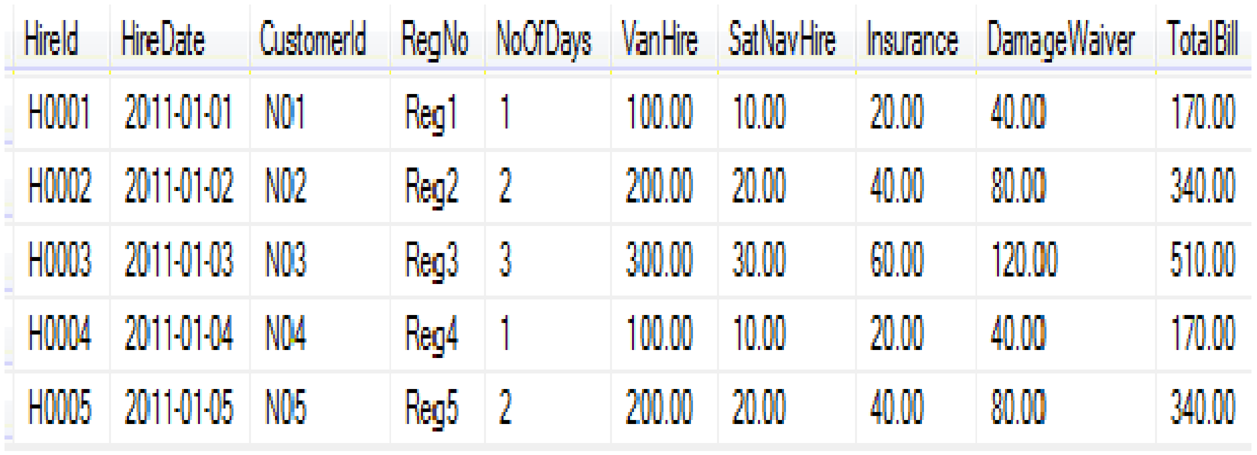
	CustomerId	CustomerName	DateOfBirth	Town	TelephoneNo	DrivingLicenceNo	Occupation
1	N00	Customer00	2000-04-09	Town00	Phone00	Licence00	Occupation00
2	N01	Customer01	2000-01-01	Town01	Phone01	Licence01	Occupation01
3	N02	Customer02	2000-01-02	Town02	Phone02	Licence02	Occupation02
4	N03	Customer03	2000-01-03	Town03	Phone03	Licence03	Occupation03
5	N04	Customer04	2000-01-04	Town04	Phone04	Licence04	Occupation04
6	N05	Customer05	2000-01-05	Town05	Phone05	Licence05	Occupation05

Van table:



	RegNo	Make	Model	Year	Colour	CC	Class
1	Reg1	Make1	Model1	2009	White	2500	Medium
2	Reg10	Make10	Model10	2010	White	2500	Medium
3	Reg11	Make11	Model11	2011	White	3000	Large
4	Reg12	Make12	Model12	2008	White	2000	Small
5	Reg13	Make13	Model13	2009	Black	2500	Medium

Hire table:



HireId	HireDate	CustomerId	RegNo	NoOfDays	VanHire	SatNavHire	Insurance	DamageWaiver	TotalBill
H0001	2011-01-01	N01	Reg1	1	100.00	10.00	20.00	40.00	170.00
H0002	2011-01-02	N02	Reg2	2	200.00	20.00	40.00	80.00	340.00
H0003	2011-01-03	N03	Reg3	3	300.00	30.00	60.00	120.00	510.00
H0004	2011-01-04	N04	Reg4	1	100.00	10.00	20.00	40.00	170.00
H0005	2011-01-05	N05	Reg5	2	200.00	20.00	40.00	80.00	340.00

And here is the script to create and populate them:

```
-- Create database
create database HireBase
go
use HireBase
go

-- Create customer table
if exists (select * from sys.tables where name = 'Customer')
drop table Customer
go

create table Customer
( CustomerId varchar(20) not null primary key,
 CustomerName varchar(30), DateOfBirth date, Town varchar(50),
 TelephoneNo varchar(30), DrivingLicenceNo varchar(30), Occupation varchar(30)
)
go

-- Populate Customer
truncate table Customer
go

declare @i int, @si varchar(10), @startdate date
set @i = 1
while @i <= 100
begin
 set @si = right('0'+CONVERT(varchar(10), @i),2)
 insert into Customer
 ( CustomerId, CustomerName, DateOfBirth, Town, TelephoneNo, DrivingLicenceNo,
Occupation)
 values
 ( 'N'+@si, 'Customer'+@si, DATEADD(d,@i-1,'2000-01-01'), 'Town'+@si, 'Phone'+@si,
'Licence'+@si, 'Occupation'+@si)
 set @i = @i + 1
end
go

select * from Customer

-- Create Van table
if exists (select * from sys.tables where name = 'Van')
drop table Van
go
```

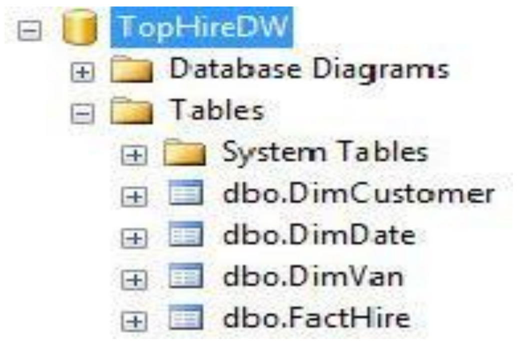
```
create table Van
( RegNo varchar(10) not null primary key,
 Make varchar(30), Model varchar(30), [Year] varchar(4),
 Colour varchar(20), CC int, Class varchar(10)
)
go

-- Populate Van table
truncate table Van
go

declare @i int, @si varchar(10)
set @i = 1
while @i <= 20
begin
 set @si = convert(varchar, @i)
 insert into Van
 ( RegNo, Make, Model, [Year], Colour, CC, Class)
 values
 ( 'Reg'+@si, 'Make'+@si, 'Model'+@si,
 case @i%4 when 0 then 2008 when 1 then 2009 when 2 then 2010 when 3 then 2011 end,
 case when @i%5<3 then 'White' else 'Black' end,
 case @i%3 when 0 then 2000 when 1 then 2500 when 2 then 3000 end,
 case @i%3 when 0 then 'Small' when 1 then 'Medium' when 2 then 'Large' end)
 set @i = @i + 1
end
go

select * from Van

-- Create Hire table
if exists (select * from sys.tables where name = 'Hire')
drop table Hire
go

create table Hire
( HireId varchar(10) not null primary key,
 HireDate date not null,
 CustomerId varchar(20) not null,
 RegNo varchar(10), NoOfDays int, VanHire money, SatNavHire money,
 Insurance money, DamageWaiver money, TotalBill money
)
go
```

```
-- Populate Hire table
truncate table Hire
go

declare @i int, @si varchar(10), @DaysFrom1stJan int, @CustomerId int, @RegNo int, @mi int
set @i = 1
while @i <= 1000
begin
 set @si = right('000'+convert(varchar(10), @i),4) -- string of i
 set @DaysFrom1stJan = (@i-1)%200 --The Hire Date is derived from i modulo 200
 set @CustomerId = (@i-1)%100+1 --The CustomerId is derived from i modulo 100
 set @RegNo = (@i-1)%20+1 --The Van RegNo is derived from i modulo 20
 set @mi = (@i-1)%3+1 --i modulo 3
 insert into Hire (HireId, HireDate, CustomerId, RegNo, NoOfDays, VanHire, SatNavHire,
Insurance, DamageWaiver, TotalBill)
 values ('H'+@si, DateAdd(d, @DaysFrom1stJan, '2011-01-01'),
 left('N0'+CONVERT(varchar(10),@CustomerId),3), 'Reg'+CONVERT(varchar(10), @RegNo),
 @mi, @mi*100, @mi*10, @mi*20, @mi*40, @mi*170)
 set @i += 1
end
go

select * from Hire
```

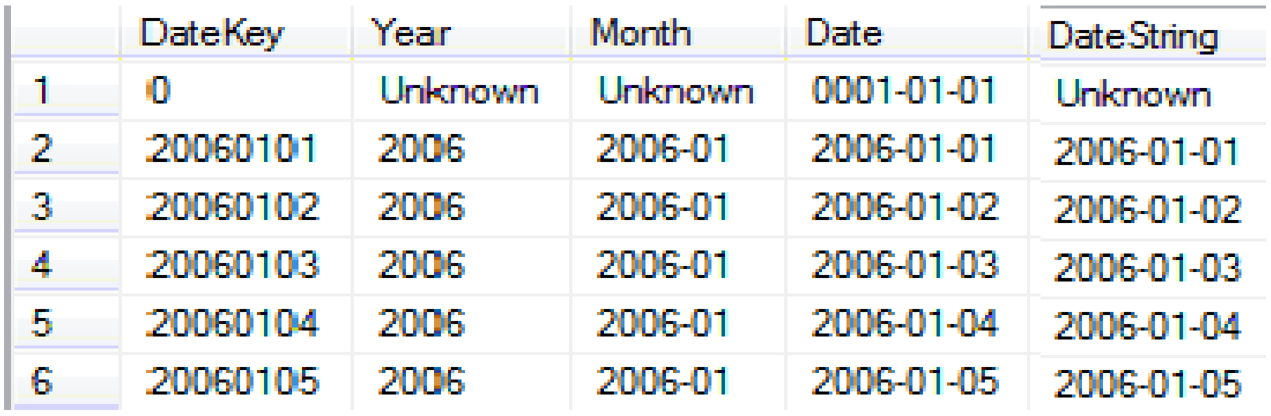
Create the Data Warehouse

So now we are going to create the 3 dimension tables and 1 fact table in the data warehouse: DimDate, DimCustomer, DimVan and FactHire. We are going to populate the 3 dimensions but we'll leave the fact table empty. The purpose of this article is to show how to populate the fact table using SSIS.

First I'll show you how it looks when it's done:

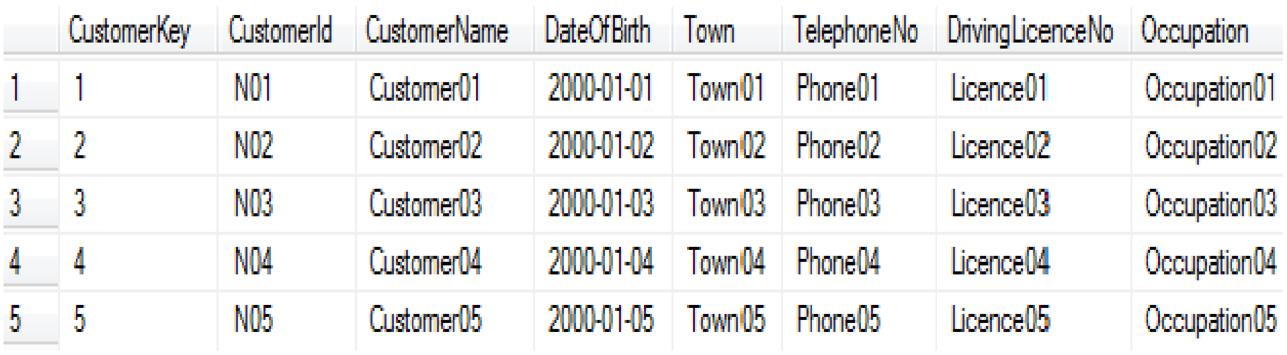


Date Dimension:



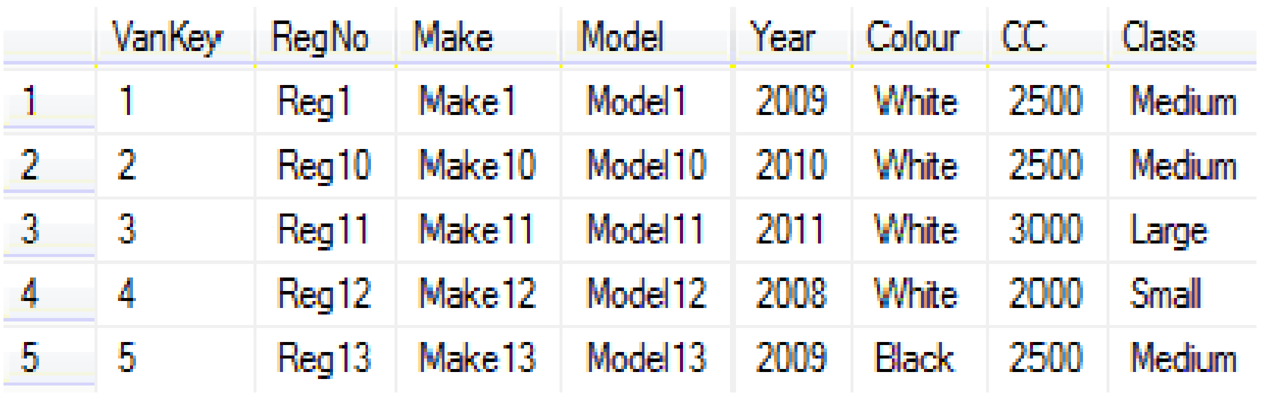
	DateKey	Year	Month	Date	DateString
1	0	Unknown	Unknown	0001-01-01	Unknown
2	20060101	2006	2006-01	2006-01-01	2006-01-01
3	20060102	2006	2006-01	2006-01-02	2006-01-02
4	20060103	2006	2006-01	2006-01-03	2006-01-03
5	20060104	2006	2006-01	2006-01-04	2006-01-04
6	20060105	2006	2006-01	2006-01-05	2006-01-05

Customer Dimension:



	CustomerKey	CustomerId	CustomerName	DateOfBirth	Town	TelephoneNo	DrivingLicenceNo	Occupation
1	1	N01	Customer01	2000-01-01	Town01	Phone01	Licence01	Occupation01
2	2	N02	Customer02	2000-01-02	Town02	Phone02	Licence02	Occupation02
3	3	N03	Customer03	2000-01-03	Town03	Phone03	Licence03	Occupation03
4	4	N04	Customer04	2000-01-04	Town04	Phone04	Licence04	Occupation04
5	5	N05	Customer05	2000-01-05	Town05	Phone05	Licence05	Occupation05

Van Dimension:



	VanKey	RegNo	Make	Model	Year	Colour	CC	Class
1	1	Reg1	Make1	Model1	2009	White	2500	Medium
2	2	Reg10	Make10	Model10	2010	White	2500	Medium
3	3	Reg11	Make11	Model11	2011	White	3000	Large
4	4	Reg12	Make12	Model12	2008	White	2000	Small
5	5	Reg13	Make13	Model13	2009	Black	2500	Medium

And then we do it. This is the script to create and populate those dim and fact tables:

```
-- Create the data warehouse
create database TopHireDW
go
use TopHireDW
go

-- Create Date Dimension
if exists (select * from sys.tables where name = 'DimDate')
drop table DimDate
go

create table DimDate
( DateKey int not null primary key,
 [Year] varchar(7), [Month] varchar(7), [Date] date, DateString varchar(10))
go

-- Populate Date Dimension
truncate table DimDate
go

declare @i int, @Date date, @StartDate date, @EndDate date, @DateKey int,
 @DateString varchar(10), @Year varchar(4),
 @Month varchar(7), @Date1 varchar(20)
set @StartDate = '2006-01-01'
set @EndDate = '2016-12-31'
set @Date = @StartDate

insert into DimDate (DateKey, [Year], [Month], [Date], DateString)
 values (0, 'Unknown', 'Unknown', '0001-01-01', 'Unknown') --The unknown row

while @Date <= @EndDate
begin
 set @DateString = convert(varchar(10), @Date, 20)
 set @DateKey = convert(int, replace(@DateString,'-',''))
 set @Year = left(@DateString,4)
 set @Month = left(@DateString, 7)
 insert into DimDate (DateKey, [Year], [Month], [Date], DateString)
 values (@DateKey, @Year, @Month, @Date, @DateString)
 set @Date = dateadd(d, 1, @Date)
end
go
```

```
select * from DimDate
```

```
-- Create Customer dimension
if exists (select * from sys.tables where name = 'DimCustomer')
drop table DimCustomer
go

create table DimCustomer
( CustomerKey int not null identity(1,1) primary key,
 CustomerId varchar(20) not null,
 CustomerName varchar(30), DateOfBirth date, Town varchar(50),
 TelephoneNo varchar(30), DrivingLicenceNo varchar(30), Occupation varchar(30)
)
go

insert into DimCustomer (CustomerId, CustomerName, DateOfBirth, Town, TelephoneNo,
 DrivingLicenceNo, Occupation)
select * from HireBase.dbo.Customer

select * from DimCustomer

-- Create Van dimension
if exists (select * from sys.tables where name = 'DimVan')
drop table DimVan
go

create table DimVan
( VanKey int not null identity(1,1) primary key,
 RegNo varchar(10) not null,
 Make varchar(30), Model varchar(30), [Year] varchar(4),
 Colour varchar(20), CC int, Class varchar(10)
)
go

insert into DimVan (RegNo, Make, Model, [Year], Colour, CC, Class)
select * from HireBase.dbo.Van
go

select * from DimVan

-- Create Hire fact table
if exists (select * from sys.tables where name = 'FactHire')
drop table FactHire
go
```

```
create table FactHire
( SnapshotDateKey int not null, --Daily periodic snapshot fact table
 HireDateKey int not null, CustomerKey int not null, VanKey int not null, --Dimension Keys
 HireId varchar(10) not null, --Degenerate Dimension
 NoOfDays int, VanHire money, SatNavHire money,
 Insurance money, DamageWaiver money, TotalBill money
)
go

select * from FactHire
```

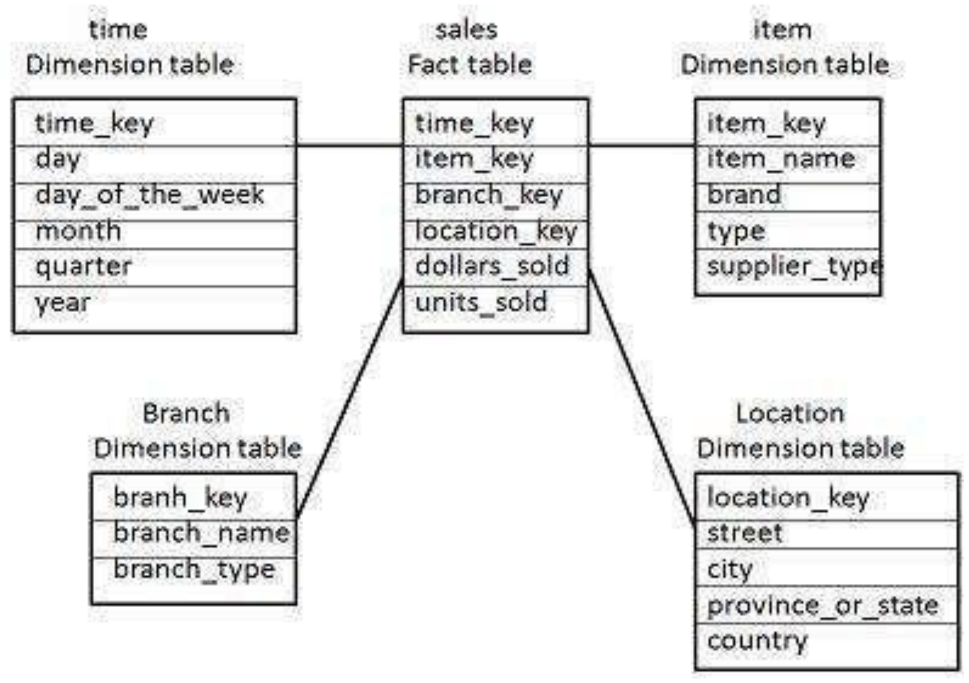
A.(ii). Design multi-demesional data models namely Star, Snowflake and Fact Constellation schemas for any one enterprise (ex. Banking,Insurance, Finance, Healthcare, manufacturing, Automobiles,sales etc).

Ans: **SchemaDefinition**

Multidimensional schema is defined using Data Mining Query Language (DMQL). The two primitives, cube definition and dimension definition, can be used for defining the data warehouses and data marts.

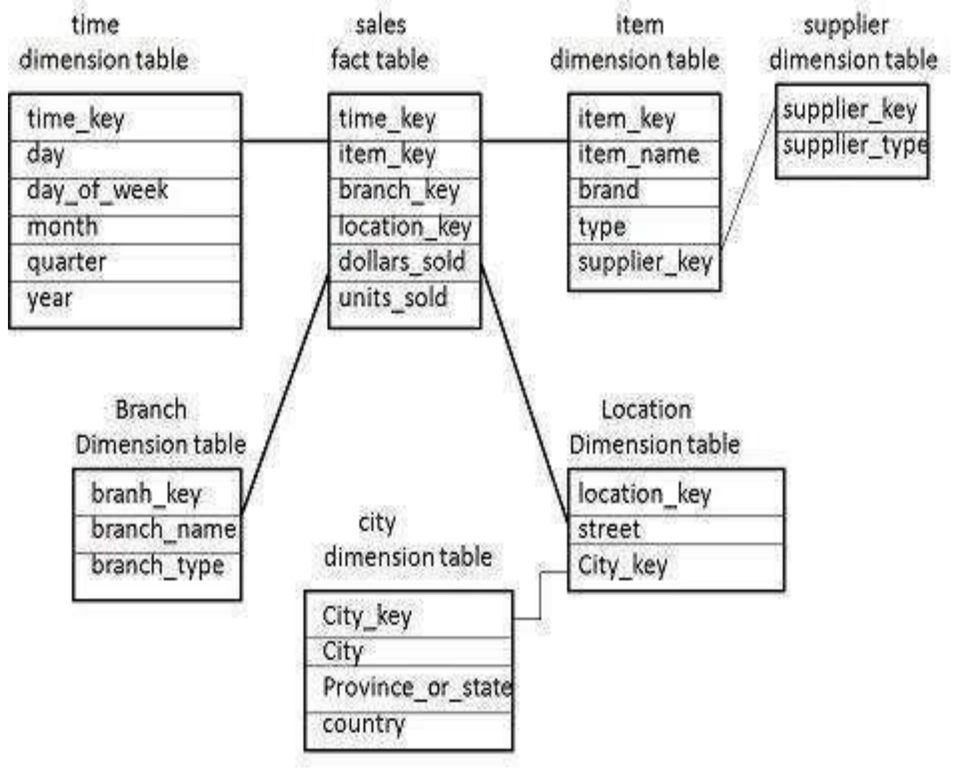
StarSchema

- Each dimension in a star schema is represented with only one-dimension table.
- This dimension table contains the set of attributes.
- The following diagram shows the sales data of a company with respect to the four dimensions, namely time, item, branch, and location.
- There is a fact table at the center. It contains the keys to each of four dimensions.
- The fact table also contains the attributes, namely dollars sold and units sold.



SnowflakeSchema

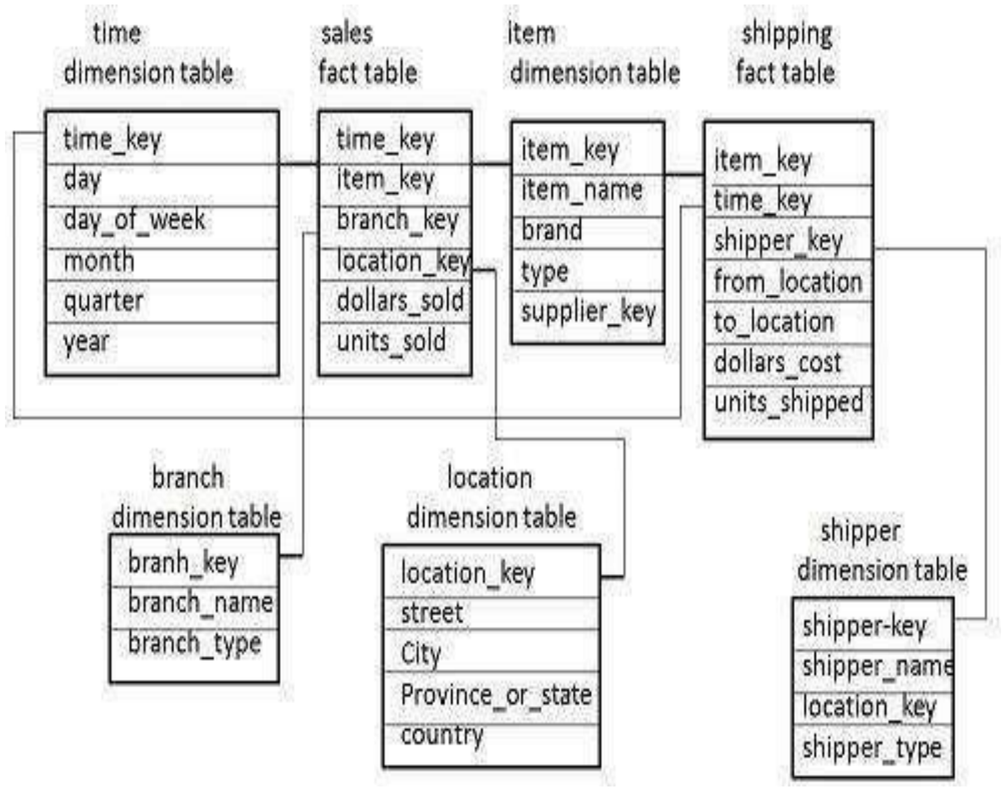
- Some dimension tables in the Snowflake schema are normalized.
- The normalization splits up the data into additional tables.
- Unlike Star schema, the dimensions table in a snowflake schema is normalized. For example, the item dimension table in star schema is normalized and split into two dimension tables, namely item and supplier table.
- Now the item dimension table contains the attributes item_key, item_name, type, brand, and supplier-key.
- The supplier key is linked to the supplier dimension table. The supplier dimension table contains the attributes supplier_key and supplier_type.



FactConstellationSchema

s

- A fact constellation has multiple fact tables. It is also known as galaxy schema.
- The following diagram shows two fact tables, namely sales and shipping.
- The sales fact table is same as that in the star schema.
- The shipping fact table has the five dimensions, namely item_key, time_key, shipper_key, from_location, to_location.
- The shipping fact table also contains two measures, namely dollars sold and units sold.
- It is also possible to share dimension tables between fact tables. For example, time, item, and location dimension tables are shared between the sales and shipping fact table.



A.(iii) Write ETL scripts and implement using data warehouse tools.

Ans:

ETL comes from Data Warehousing and stands for Extract-Transform-Load. ETL covers a process of how the data are loaded from the source system to the data warehouse. Extraction–transformation–loading (ETL) tools are pieces of software responsible for the extraction of data from several sources, its cleansing, customization, reformatting, integration, and insertion into a data warehouse.

Building the ETL process is potentially one of the biggest tasks of building a warehouse; it is complex, time consuming, and consumes most of data warehouse project’s implementation efforts, costs, and resources.

Building a data warehouse requires focusing closely on understanding three main areas:

1. Source Area- The source area has standard models such as entity relationship diagram.

2. Destination Area- The destination area has standard models such as star schema.

3. Mapping Area- But the mapping area has not a standard model till now.

Abbreviations

- ETL-extraction–transformation–loading
- DW-data warehouse
- DM- data mart
- OLAP- on-line analytical processing
- DS-data sources
- ODS- operational data store
- DSA- data staging area
- DBMS- database management system
- OLTP-on-line transaction processing
- CDC-change data capture
- SCD-slowly changing dimension
- FCME- first-class modeling elements
- EMD-entity mapping diagram
- DSA-data storage area

ETL Process:

Extract

The Extract step covers the data extraction from the source system and makes it accessible for further processing. The main objective of the extract step is to retrieve all the required data from the source system with as little resources as possible. The extract step should be designed in a way that it does not negatively affect the source system in terms or performance, response time or any kind of locking.

There are several ways to perform the extract:

- Update notification - if the source system is able to provide a notification that a record has been changed and describe the change, this is the easiest way to get the data.
- Incremental extract - some systems may not be able to provide notification that an update has occurred, but they are able to identify which records have been modified and provide an extract of such records. During further ETL steps, the system needs to identify changes and propagate it down. Note, that by using daily extract, we may not be able to handle deleted records properly.
- Full extract - some systems are not able to identify which data has been changed at all, so a full extract is the only way one can get the data out of the system. The full extract requires keeping a copy of the last extract in the same format in order to be able to identify changes. Full extract handles deletions as well.

Transform

The transform step applies a set of rules to transform the data from the source to the target. This includes converting any measured data to the same dimension (i.e. conformed dimension) using the same units so that they can later be joined. The transformation step also requires joining data from several sources, generating aggregates, generating surrogate keys, sorting, deriving new calculated values, and applying advanced validation rules.

Load

During the load step, it is necessary to ensure that the load is performed correctly and with as little resources as possible. The target of the Load process is often a database. In order to make the load process efficient, it is helpful to disable any constraints and indexes before the load and enable them back only after the load completes. The referential integrity needs to be maintained by ETL tool to ensure consistency.

ETL method – nothin' but SQL

ETL as scripts that can just be run on the database.These scripts must be re-runnable: they should be able to be run without modification to pick up any changes in the legacy data, and automatically work out how to merge the changes into the new schema.

In order to meet the requirements, my scripts must:

1. INSERT rows in the new tables based on any data in the source that hasn't already been created in the destination
2. UPDATE rows in the new tables based on any data in the source that has already been inserted in the destination
3. DELETE rows in the new tables where the source data has been deleted

Now, instead of writing a whole lot of INSERT, UPDATE and DELETE statements, I thought "surely MERGE would be both faster and better" – and in fact, that has turned out to be the case. By writing all the transformations as MERGE statements, I've satisfied all the criteria, while also making my code very easily modified, updated, fixed and rerun. If I discover a bug or a change in requirements, I simply change the way the column is transformed in the MERGE statement, and re-run the statement. It then takes care of working out whether to insert, update or delete each row.

My next step was to design the architecture for my custom ETL solution. I went to the dba with the following design, which was approved and created for me:

1. create two new schemas on the new 11g database: LEGACY and MIGRATE
2. take a snapshot of all data in the legacy database, and load it as tables in the LEGACY schema
3. grant read-only on all tables in LEGACY to MIGRATE
4. grant CRUD on all tables in the target schema to MIGRATE.

For example, in the legacy database we have a table:

```
LEGACY.BMS_PARTIES(
 par_id            NUMBER       PRIMARY KEY,
 par_domain        VARCHAR2(10)  NOT NULL,
 par_first_name    VARCHAR2(100) ,
 par_last_name     VARCHAR2(100),
 par_dob           DATE,
 par_business_name  VARCHAR2(250),
 created_by        VARCHAR2(30)  NOT NULL,
 creation_date     DATE         NOT NULL,
 last_updated_by   VARCHAR2(30),
 last_update_date  DATE)
```

In the new model, we have a new table that represents the same kind of information:

```
NEW.TBMS_PARTY(
 party_id          NUMBER(9)     PRIMARY KEY,
 party_type_code   VARCHAR2(10)  NOT NULL,
 first_name        VARCHAR2(50),
 surname           VARCHAR2(100),
date_of_birth      DATE,

business_name      VARCHAR2(300),

db_created_by      VARCHAR2(50)  NOT NULL,

db_created_on      DATE          DEFAULT SYSDATE NOT NULL,

db_modified_by     VARCHAR2(50),

db_modified_on     DATE,
```

```
version_id         NUMBER(12)    DEFAULT 1 NOT NULL)
```

This was the simplest transformation you could possibly think of – the mapping from one to the other is 1:1, and the columns almost mean the same thing.

The solution scripts start by creating an intermediary table:

```
MIGRATE.TBMS_PARTY(

 old_par_id         NUMBER        PRIMARY KEY,

 party_id           NUMBER(9)     NOT NULL,

 party_type_code    VARCHAR2(10)  NOT NULL,

 first_name         VARCHAR2(50),

 surname            VARCHAR2(100),

 date_of_birth      DATE,

 business_name      VARCHAR2(300),

 db_created_by      VARCHAR2(50),
 db_created_on      DATE,

 db_modified_by     VARCHAR2(50),

 db_modified_on     DATE,

 deleted            CHAR(1))
```

The second step is the E and T parts of “ETL”: I query the legacy table, transform the data right there in the query, and insert it into the intermediary table. However, since I want to be able to re• run this script as often as I want, I wrote this as a MERGE statement:

```
MERGE INTO MIGRATE.TBMS_PARTY dest

USING (

  SELECT par_id            AS old_par_id,

         par_id            AS party_id,

         CASE par_domain

           WHEN 'P' THEN 'PE' /*Person*/

           WHEN 'O' THEN 'BU' /*Business*/

         END               AS party_type_code,

         par_first_name    AS first_name,

         par_last_name     AS surname,

         par_dob           AS date_of_birth,

         par_business_name AS business_name,
        created_by        AS db_created_by,

        creation_date     AS db_created_on,

        last_updated_by  AS db_modified_by,

        last_update_date  AS db_modified_on

  FROM LEGACY.BMS_PARTIES s

  WHERE NOT EXISTS (

    SELECT null

    FROM  MIGRATE.TBMS_PARTY d

    WHERE  d.old_par_id = s.par_id

    AND    (d.db_modified_on = s.last_update_date

           OR (d.db_modified_on IS NULL

              AND s.last_update_date IS NULL))

    )

  ) src

ON (src.OLD_PAR_ID = dest.OLD_PAR_ID)

WHEN MATCHED THEN UPDATE SET

  party_id        = src.party_id        ,

  party_type_code = src.party_type_code ,

  first_name      = src.first_name      ,
  surname        = src.surname        ,

  date_of_birth = src.date_of_birth ,

  business_name = src.business_name ,

  db_created_by = src.db_created_by ,

  db_created_on  = src.db_created_on  ,

  db_modified_by  = src.db_modified_by  ,
```

A.(iv) Perform Various OLAP operations such slice, dice, roll up, drill up and pivot.

Ans: **OLAPOPERATIONS**

Online Analytical Processing Server (OLAP) is based on the multidimensional data model. It allows managers, and analysts to get an insight of the information through fast, consistent, and interactive access to information.

OLAP operations in multidimensional data.

Here is the list of OLAP operations:

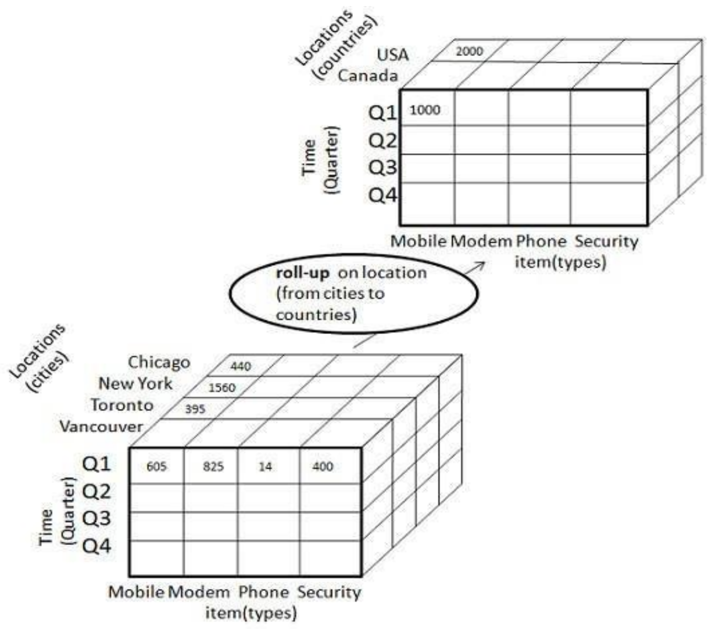
- Roll-up
- Drill-down
- Slice and dice
- Pivot (rotate)

Roll-up

Roll-up performs aggregation on a data cube in any of the following ways:

- By climbing up a concept hierarchy for a dimension
- By dimension reduction

The following diagram illustrates how roll-up works.



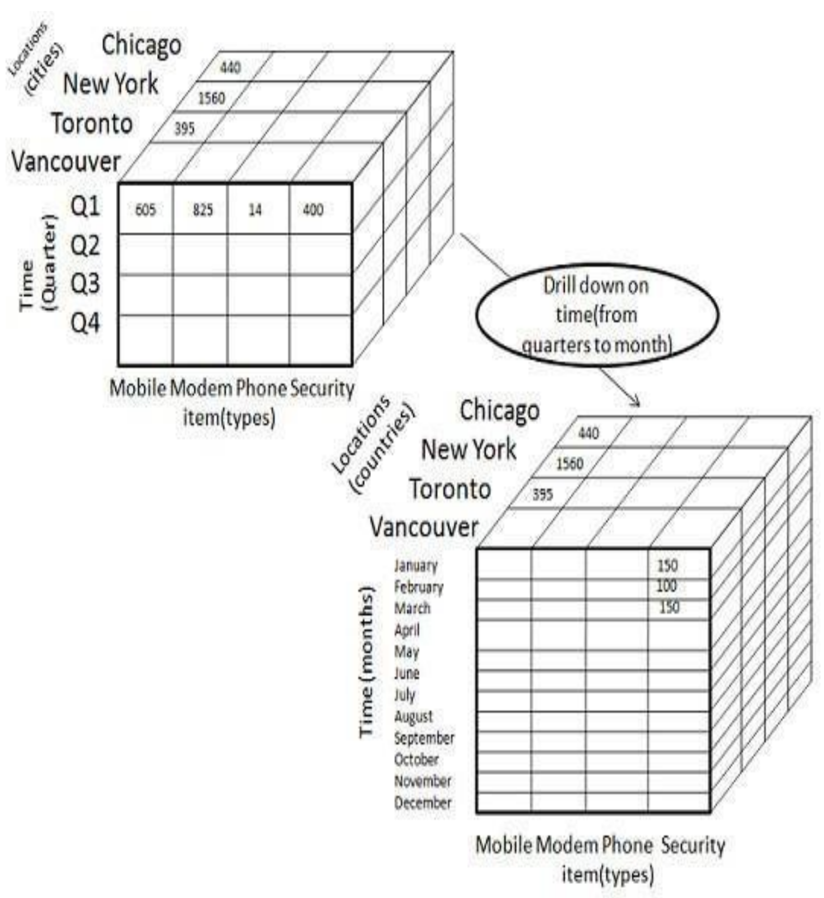
- Roll-up is performed by climbing up a concept hierarchy for the dimension location.
- Initially the concept hierarchy was "street < city < province < country".
- On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
- The data is grouped into cities rather than countries.
- When roll-up is performed, one or more dimensions from the data cube are removed.

Drill-down

Drill-down is the reverse operation of roll-up. It is performed by either of the following ways:

- By stepping down a concept hierarchy for a dimension
- By introducing a new dimension.

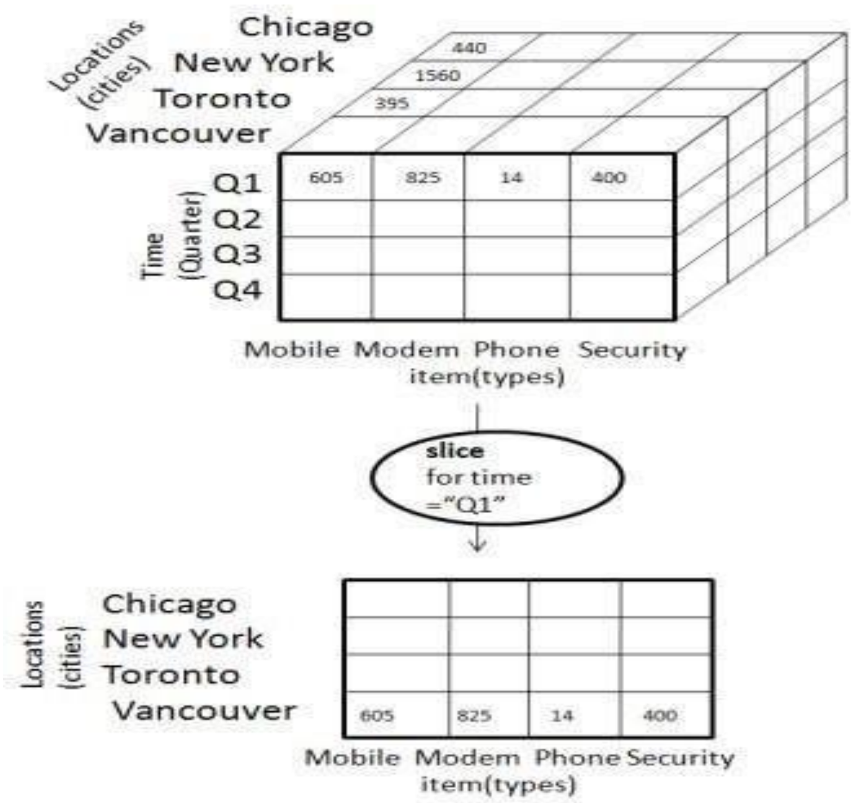
The following diagram illustrates how drill-down works:



- Drill-down is performed by stepping down a concept hierarchy for the dimension time.
- Initially the concept hierarchy was "day < month < quarter < year."
- On drilling down, the time dimension is descended from the level of quarter to the level of month.
- When drill-down is performed, one or more dimensions from the data cube are added.
- It navigates the data from less detailed data to highly detailed data.

Slice

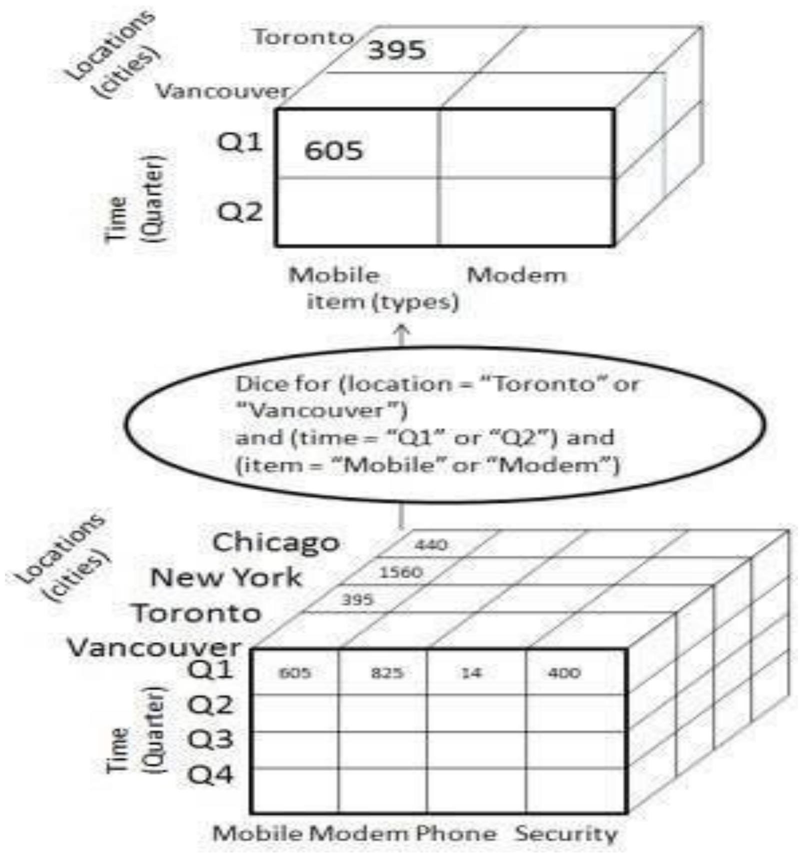
The slice operation selects one particular dimension from a given cube and provides a new sub-cube. Consider the following diagram that shows how slice works.



- Here Slice is performed for the dimension "time" using the criterion time = "Q1".
- It will form a new sub-cube by selecting one or more dimensions.

Dice

Dice selects two or more dimensions from a given cube and provides a new sub-cube. Consider the following diagram that shows the dice operation.

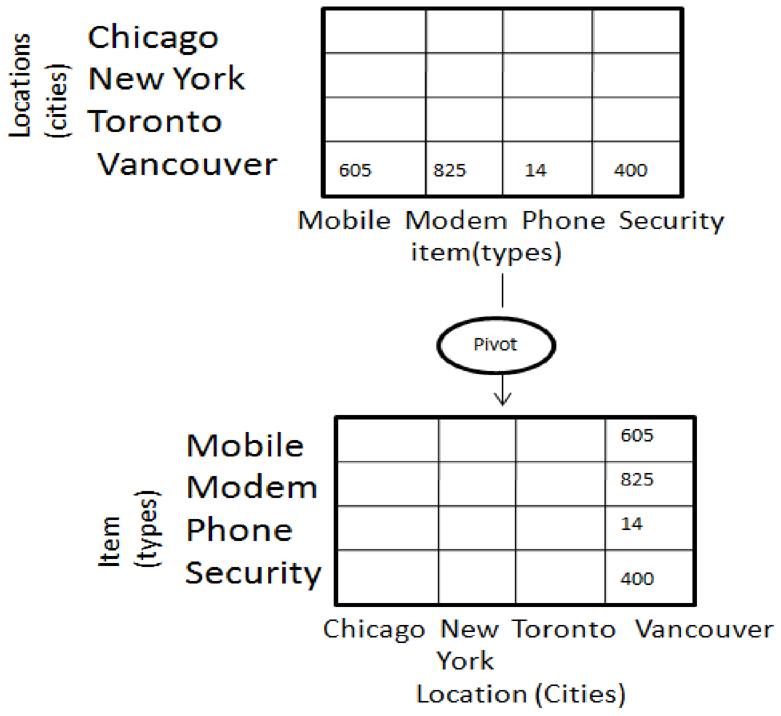


The dice operation on the cube based on the following selection criteria involves three dimensions.

- (location = "Toronto" or "Vancouver")
- (time = "Q1" or "Q2")
- (item =" Mobile" or "Modem")

Pivot

The pivot operation is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data. Consider the following diagram that shows the pivot operation.


Locations (cities)
Chicago
New York
Toronto
Vancouver
605
825
14
400
Mobile Modem Phone Security
item(types)
Pivot
Item (types)
Mobile
Modem
Phone
Security
605
825
14
400
Chicago New York Toronto Vancouver
Location (Cities)

Experiment-2:

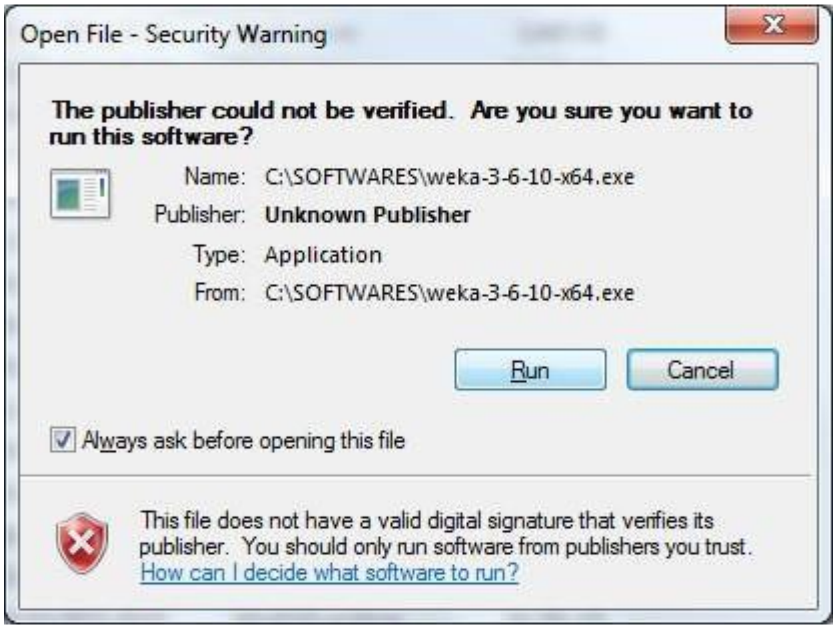
2. Explore machine learning tool "WEKA"

A.Explore WEKA Data Mining/Machine Learning Toolkit

B.(i) Downloading and/or installation of WEKA data mining toolkit.

Ans: **<u>Install Steps for WEKA a Data Mining Tool</u>**

1. Download the software as your requirements from the below given link. http://www.cs.waikato.ac.nz/ml/weka/downloading.html
2. The Java is mandatory for installation of WEKA so if you have already Java on your machine then download only WEKA else download the software with JVM.
3. Then open the file location and double click on the file



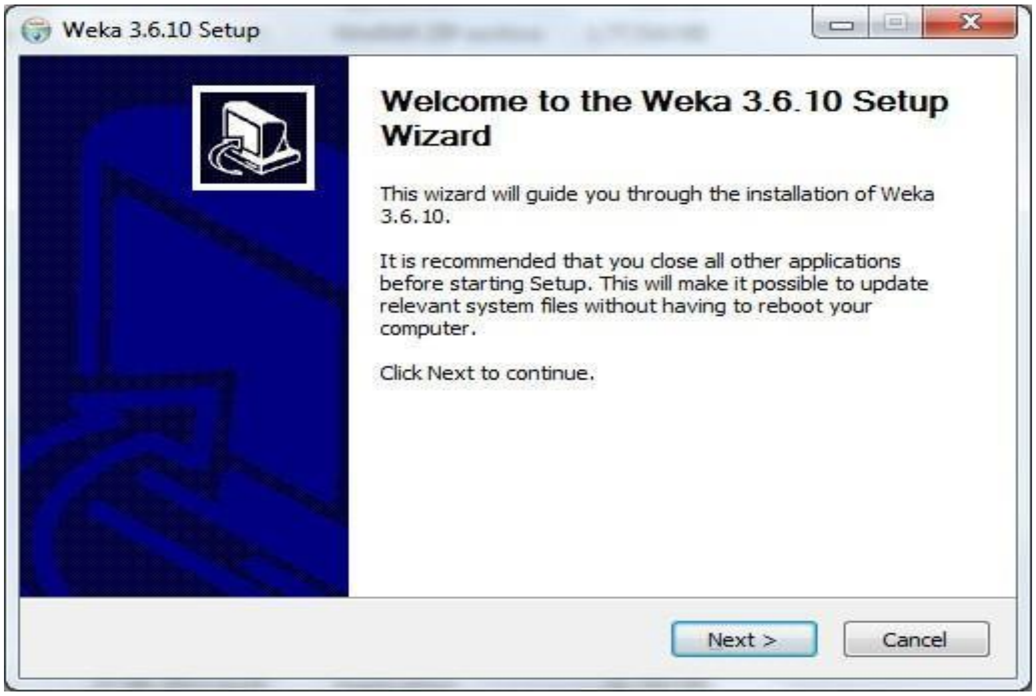
Open File - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

Name: C:\SOFTWARES\weka-3-6-10-x64.exe
Publisher: Unknown Publisher
Type: Application
From: C:\SOFTWARES\weka-3-6-10-x64.exe

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust.
How can I decide what software to run?

4. Click Next



Weka 3.6.10 Setup

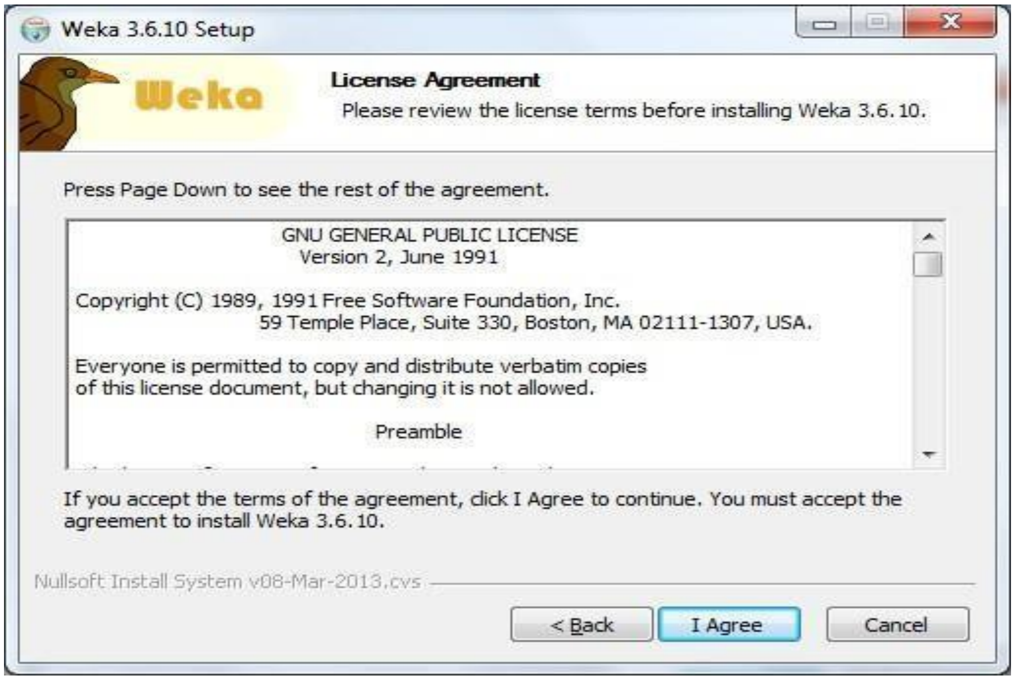
Welcome to the Weka 3.6.10 Setup Wizard

This wizard will guide you through the installation of Weka 3.6.10.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

5. Click I Agree.



Weka 3.6.10 Setup

License Agreement

Please review the license terms before installing Weka 3.6.10.

Press Page Down to see the rest of the agreement.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

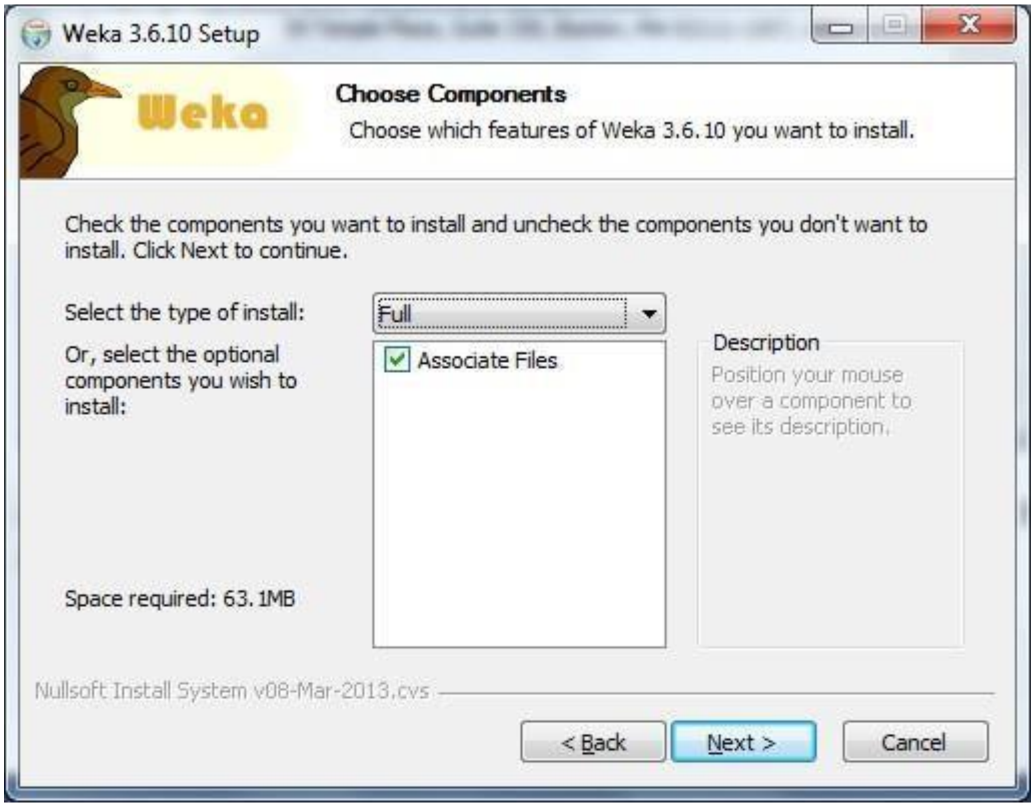
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Weka 3.6.10.

Nullsoft Install System v08-Mar-2013.cvs

6. As your requirement do the necessary changes of settings and click Next. Full and Associate files are the recommended settings.



Weka 3.6.10 Setup

Choose Components

Choose which features of Weka 3.6.10 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Full

Or, select the optional components you wish to install: [✓ Associate Files]

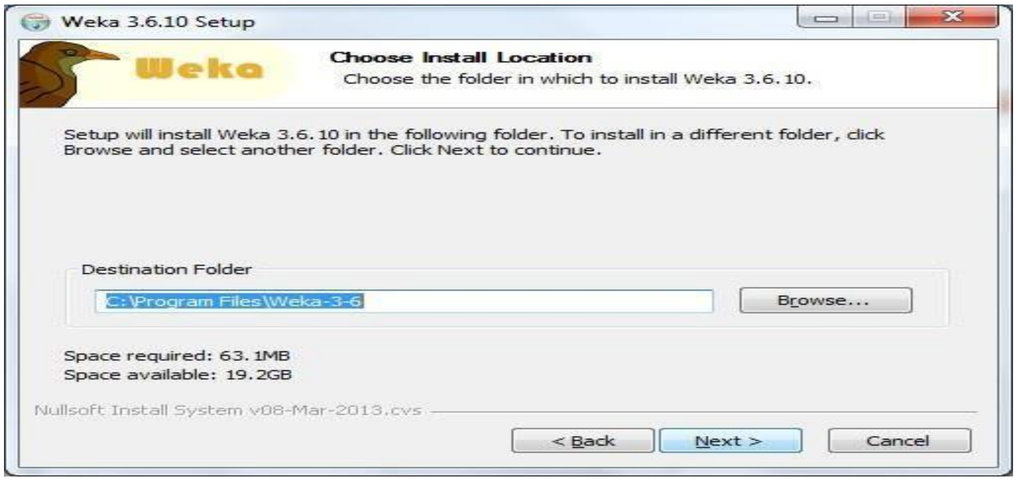
Description

Position your mouse over a component to see its description.

Space required: 63.1MB

Nullsoft Install System v08-Mar-2013.cvs

7. Change to your desire installation location.



Weka 3.6.10 Setup

Choose Install Location

Choose the folder in which to install Weka 3.6.10.

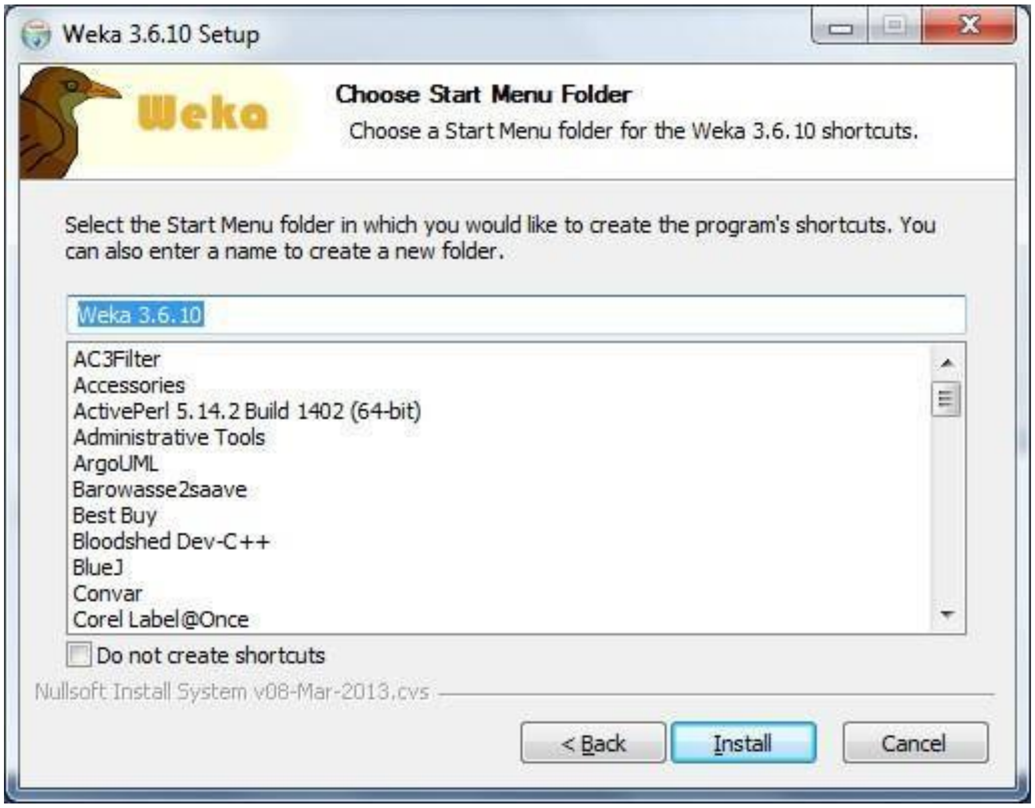
Setup will install Weka 3.6.10 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\Weka-3-6

Space required: 63.1MB
Space available: 19.2GB

Nullsoft Install System v08-Mar-2013.cvs

8. If you want a shortcut then check the box and click Install.


Weka 3.6.10 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the Weka 3.6.10 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

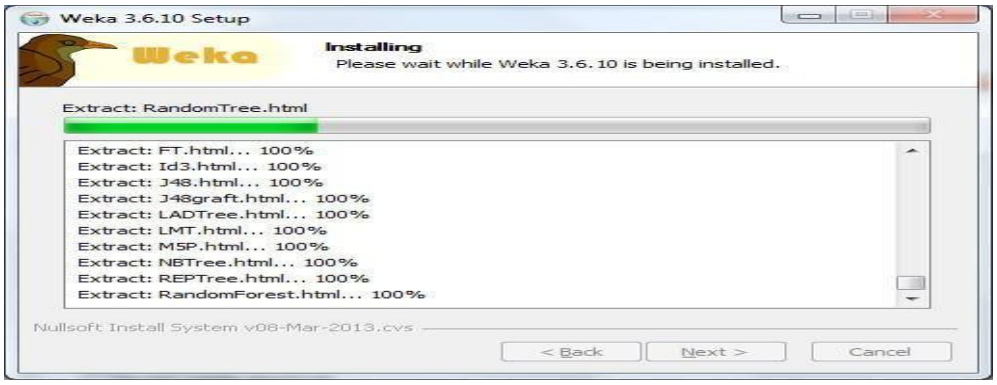
Weka 3.6.10

AC3Filter
Accessories
ActivePerl 5.14.2 Build 1402 (64-bit)
Administrative Tools
ArgoUML
Barowasse2saave
Best Buy
Bloodshed Dev-C++
BlueJ
Convar
Corel Label@Once

☐ Do not create shortcuts

Nullsoft Install System v08-Mar-2013.cvs

9. The Installation will start wait for a while it will finish within a minute.


Weka 3.6.10 Setup

Installing

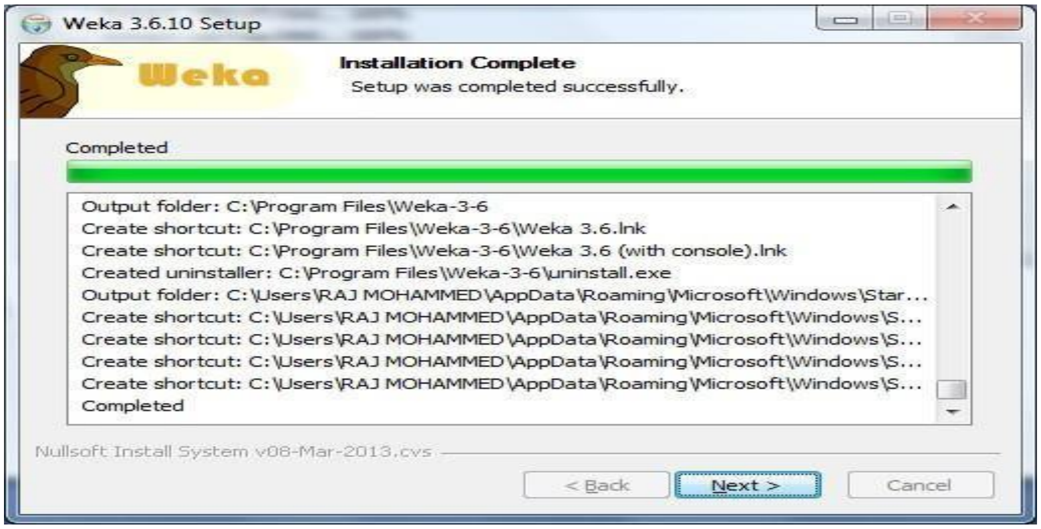
Please wait while Weka 3.6.10 is being installed.

```
Extract: RandomTree.html
```

```
Extract: FT.html... 100%
Extract: Id3.html... 100%
Extract: J48.html... 100%
Extract: J48graft.html... 100%
Extract: LADTree.html... 100%
Extract: LMT.html... 100%
Extract: M5P.html... 100%
Extract: NBTree.html... 100%
Extract: REPTree.html... 100%
Extract: RandomForest.html... 100%
```

Nullsoft Install System v08-Mar-2013.cvs

10. After complete installation click on Next.



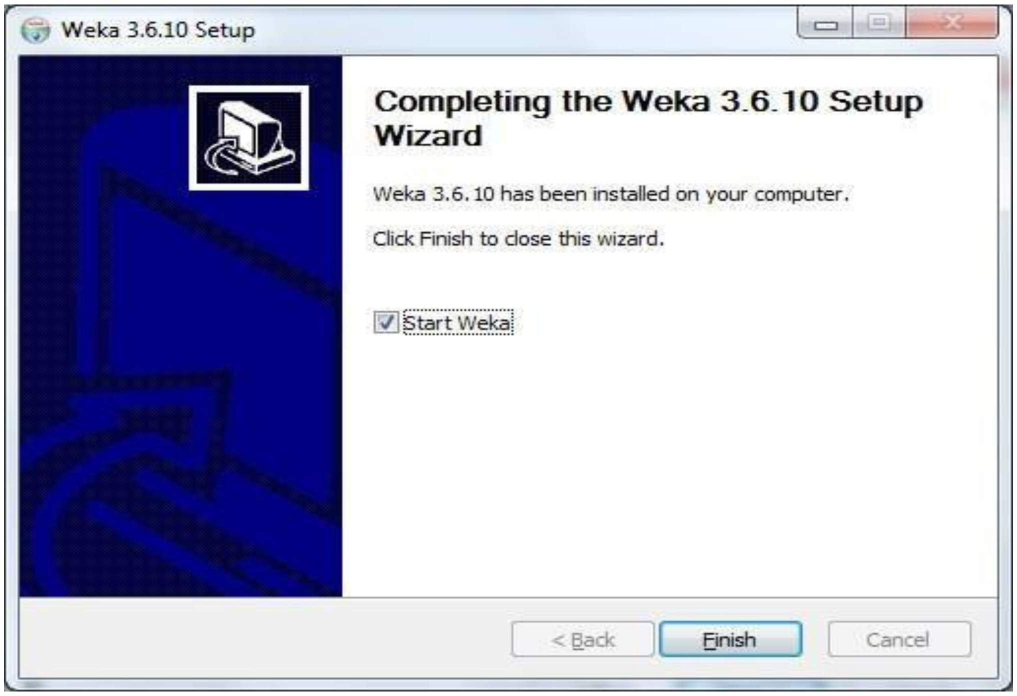
Installation Complete
Setup was completed successfully.

Completed

```
Output folder: C:\Program Files\Weka-3-6
Create shortcut: C:\Program Files\Weka-3-6\Weka 3.6.lnk
Create shortcut: C:\Program Files\Weka-3-6\Weka 3.6 (with console).lnk
Created uninstaller: C:\Program Files\Weka-3-6\uninstall.exe
Output folder: C:\Users\RAJ MOHAMMED\AppData\Roaming\Microsoft\Windows\Star...
Create shortcut: C:\Users\RAJ MOHAMMED\AppData\Roaming\Microsoft\Windows\S...
Create shortcut: C:\Users\RAJ MOHAMMED\AppData\Roaming\Microsoft\Windows\S...
Create shortcut: C:\Users\RAJ MOHAMMED\AppData\Roaming\Microsoft\Windows\S...
Create shortcut: C:\Users\RAJ MOHAMMED\AppData\Roaming\Microsoft\Windows\S...
Completed
```

Nullsoft Install System v08-Mar-2013.cvs

11. Hurray !!!!!!! That's all click on the Finish and take a shovel and start Mining. Best of Luck.

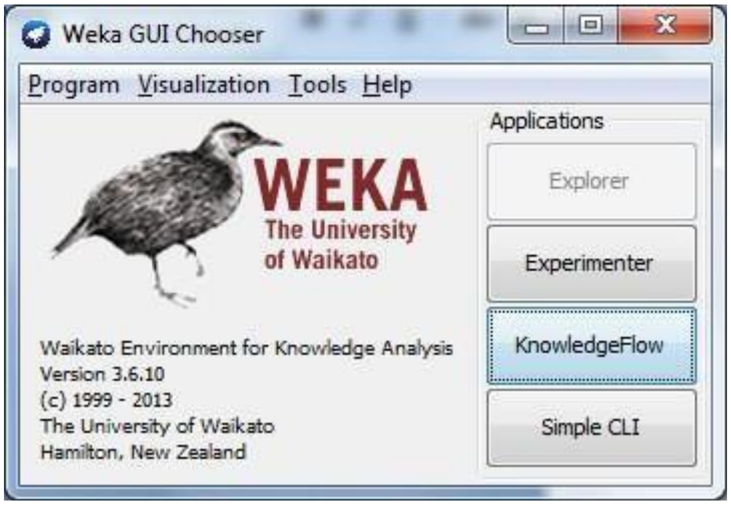


Completing the Weka 3.6.10 Setup Wizard

Weka 3.6.10 has been installed on your computer.

Click Finish to close this wizard.

[✓] Start Weka



Waikato Environment for Knowledge Analysis
Version 3.6.10
(c) 1999 - 2013
The University of Waikato
Hamilton, New Zealand

This is the GUI you get when started. You have 4 options Explorer, Experimenter, KnowledgeFlow and Simple CLI.

B.(ii)Understand the features of WEKA tool kit such as Explorer, Knowledge flow interface, Experimenter, command-line interface.

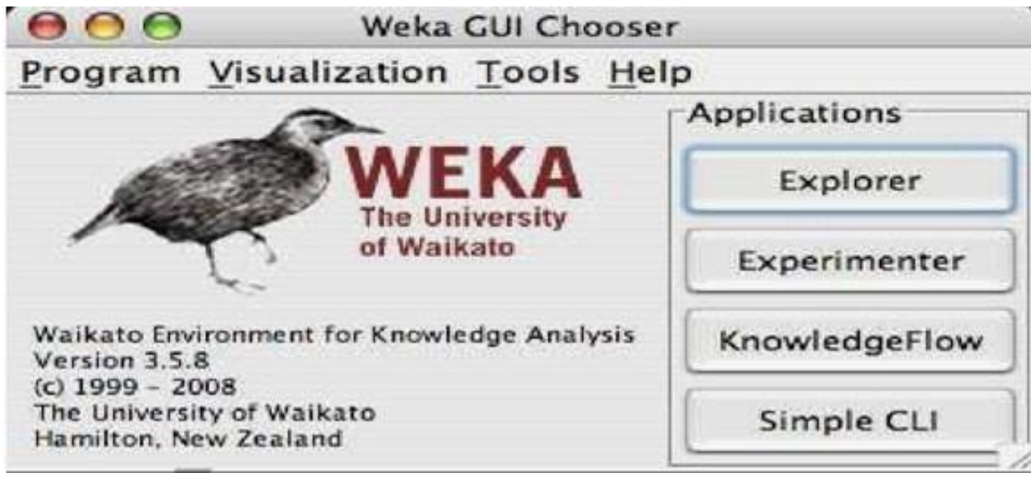
Ans: **<u>WEKA</u>**

Weka is created by researchers at the university WIKATO in New Zealand. University of Waikato, Hamilton, New Zealand Alex Seewald (original Command-line primer) David Scuse (original Experimenter tutorial)

- It is java based application.
- It is collection often source, Machine Learning Algorithm.
- The routines (functions) are implemented as classes and logically arranged in packages.
- It comes with an extensive GUI Interface.
- Weka routines can be used standalone via the command line interface.

The Graphical User Interface;-

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



Waikato Environment for Knowledge Analysis
Version 3.5.8
(c) 1999 - 2008
The University of Waikato
Hamilton, New Zealand

The buttons can be used to start the following applications:

- **Explorer An environment** for exploring data with WEKA (the rest of this Documentation deals with this application in more detail).
- **Experimenter** An environment for performing experiments and conducting statistical tests between learning schemes.

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

- **SimpleCLI Provides** a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface.

1. Explorer

The Graphical user interface

Section Tabs

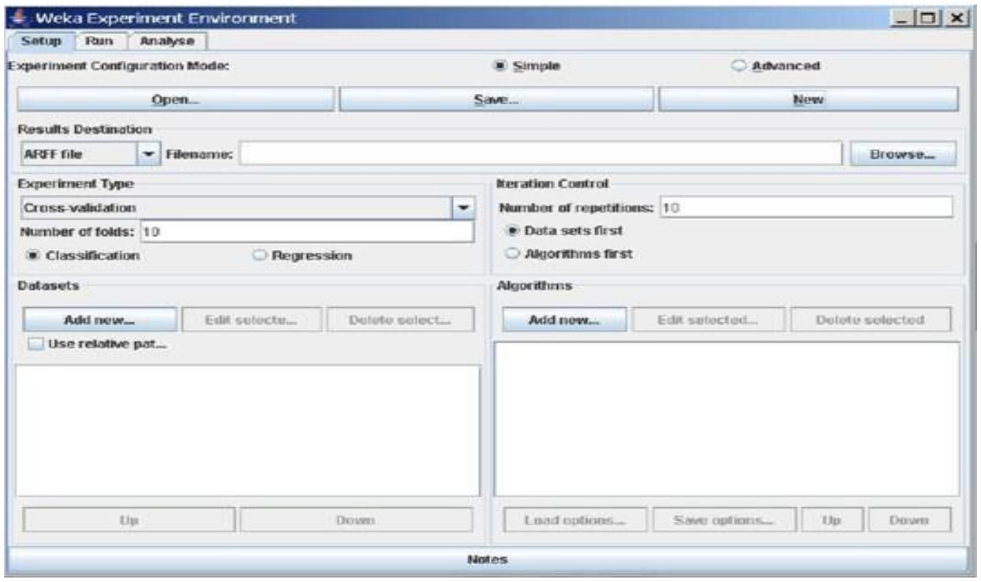
At the very top of the window, just below the title bar, is a row of tabs. When the Explorer is first started only the first tab is active; the others are grayed out. This is because it is necessary to open (and potentially pre-process) a data set before starting to explore the data. The tabs are as follows:

1. **Preprocess.** Choose and modify the data being acted on.
2. **Classify.** Train & test learning schemes that classify or perform regression
3. **Cluster.** Learn clusters for the data.
4. **Associate.** Learn association rules for the data.
5. **Select attributes.** Select the most relevant attributes in the data.
6. **Visualize.** View an interactive 2D plot of the data.

Once the tabs are active, clicking on them flicks between different screens, on which the respective actions can be performed. The bottom area of the window (including the status box, the log button, and the Weka bird) stays visible regardless of which section you are in. The Explorer can be easily extended with custom tabs. The Wiki article **"Adding tabs in the Explorer" explains this in detail.**

2. Weka Experimenter:-

The Weka Experiment Environment enables the user to create, run, modify, and analyze experiments in a more convenient manner than is possible when processing the schemes individually. For example, the user can create an experiment that runs several schemes against a series of datasets and then analyze the results to determine if one of the schemes is (statistically) better than the other schemes.



The Experiment Environment can be run from the command line using the Simple CLI. For example, the following commands could be typed into the CLI to run the OneR scheme on the Iris dataset using a basic train and test process. (Note that the commands would be typed on one line into the CLI.) While commands can be typed directly into the CLI, this technique is not particularly convenient and the experiments are not easy to modify. The Experimenter comes in two flavors', either with a simple interface that provides most of the functionality one needs for experiments, or with an interface **with full access to the Experimenter's capabilities. You can** choose between those two with the Experiment Configuration Mode radio buttons:

- Simple
- Advanced

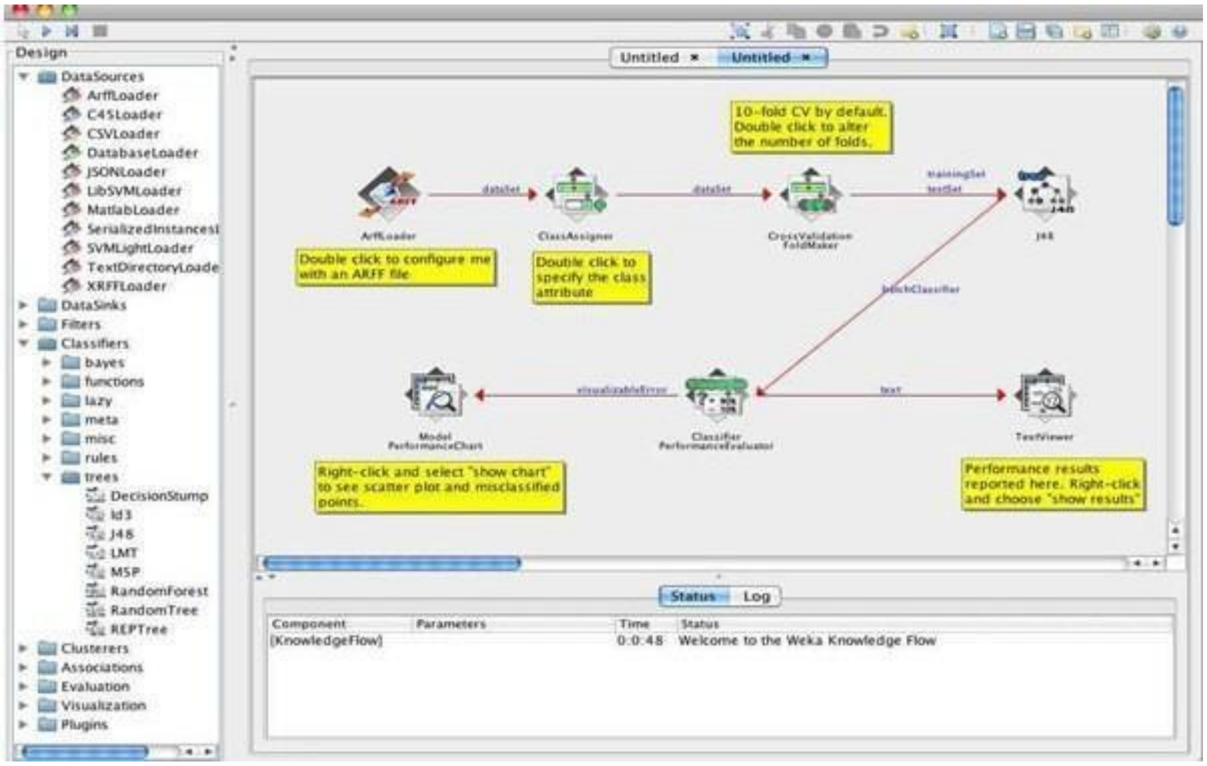
Both setups allow you to setup standard experiments, that are run locally on a single machine, or remote experiments, which are distributed between several hosts. The distribution of experiments cuts down the time the experiments will take until completion, but on the other hand the setup takes more time. The next section covers the standard experiments (both, simple and advanced), followed by the remote experiments and finally the analyzing of the results.

3. <u>**Knowledge Flow**</u>

Introduction

The Knowledge Flow provides an alternative to the Explorer as a graphical front end to **WEKA's core algorithms.**

The Knowledge Flow presents a data-flow inspired interface to WEKA. The user can select WEKA components from a palette, place them on a layout canvas and connect them together in order to form a knowledge flow for processing and analyzing data. At present, all of **WEKA's classifiers, filters, clusterers, associators, loaders and savers a**re available in the Knowledge Flow along with some extra tools.



```
Component        Parameters    Time    Status
[KnowledgeFlow]                0:0:48  Welcome to the Weka Knowledge Flow
```

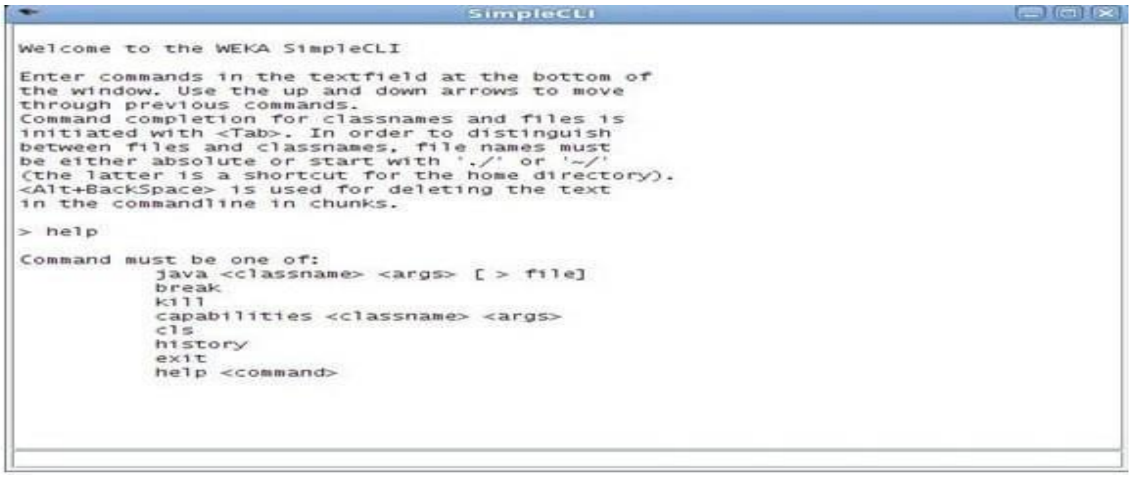
The Knowledge Flow can handle data either incrementally or in batches (the Explorer handles batch data only). Of course learning from data incremen- tally requires a classifier that can be updated on an instance by instance basis. Currently in WEKA there are ten classifiers that can handle data incrementally.

The Knowledge Flow offers the following features:

- **Intuitive** data flow style layout.
- **Process** data in batches or incrementally.
- **Process multiple batches** or streams in parallel (each separate flow executes in its own thread) .
- **Process multiple streams sequentially** via a user-specified order of execution.
- **Chain filters** together.
- **View models** produced by classifiers for each fold in a cross validation.
- **Visualize performance** of incremental classifiers during processing (scrolling plots of classification accuracy, RMS error, predictions etc.).
- **Plugin "perspectives" that add major new functionality (e.g. 3D data** visualization, time series forecasting environment etc.).

4. Simple CLI

The Simple CLI provides full access to all Weka classes, i.e., classifiers, filters, clusterers, etc., but without the hassle of the CLASSPATH (it facilitates the one, with which Weka was started). It offers a simple Weka shell with separated command line and output.



```
Welcome to the WEKA SimpleCLI

Enter commands in the textfield at the bottom of
the window. Use the up and down arrows to move
through previous commands.
Command completion for classnames and files is
initiated with <Tab>. In order to distinguish
between files and classnames, file names must
be either absolute or start with './' or '~/'
(the latter is a shortcut for the home directory).
<Alt+BackSpace> is used for deleting the text
in the commandline in chunks.

> help

Command must be one of:
          java <classname> <args> [ > file]
          break
          kill
          capabilities <classname> <args>
          cls
          history
          exit
          help <command>
```

Commands

The following commands are available in the Simple CLI:

- Java <classname> [<args>]

  Invokes a java class with the given arguments (if any).

- Break

  Stops the current thread, e.g., a running classifier, in a friendly manner kill stops the current thread in an unfriendly fashion.

- Cls
  Clears the output area

- Capabilities <classname> [<args>]

  Lists the capabilities of the specified class, e.g., for a classifier with its.

- option:

  Capabilities weka.classifiers.meta.Bagging -W weka.classifiers.trees.Id3

- exit

  Exits the Simple CLI

- help [<command>]

  Provides an overview of the available commands if without a command name as argument, otherwise more help on the specified command

Invocation

In order to invoke a Weka class, one has only to prefix the class with "java". This command tells the Simple CLI to load a class and execute it with any given parameters. E.g., the J48 classifier can be invoked on the iris dataset with the following command:

```
java weka.classifiers.trees.J48 -t c:/temp/iris.arff
```

This results in the following output:

Command redirection

Starting with this version of Weka one can perform a basic redirection: java weka.classifiers.trees.J48 -t test.arff > j48.txt

Note: the > must be preceded and followed by a space, otherwise it is not recognized as redirection, but part of another parameter.

Command completion

Commands starting with java support completion for classnames and filenames via Tab (Alt+BackSpace deletes parts of the command again). In case that there are several matches, Weka lists all possible matches.

- Package Name Completion java weka.cl<Tab>

Results in the following output of possible matches of

package names: Possible matches:

```
weka.classifiers
weka.clusterers
```

- Classname completion

java weka.classifiers.meta.A<Tab> lists the following classes

```
Possible matches:
        weka.classifiers.meta.AdaBoostM1
        weka.classifiers.meta.AdditiveRegression
        weka.classifiers.meta.AttributeSelectedClassifier
```

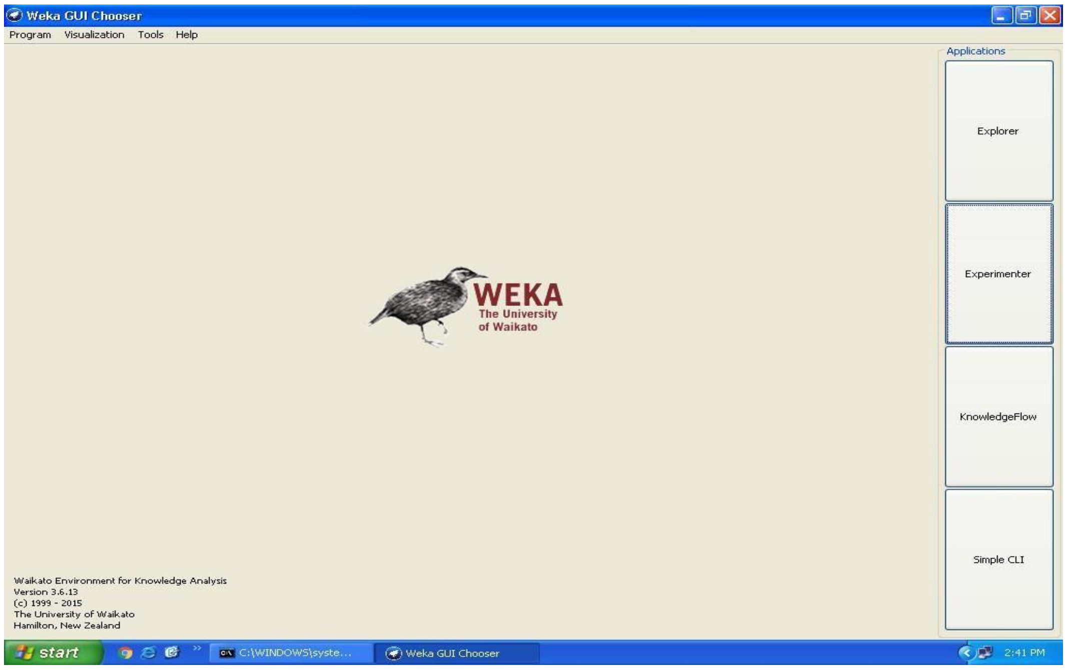
- Filename Completion

In order for Weka to determine whether a the string under the cursor is a classname or a filename, filenames need to be absolute (Unix/Linx: /some/path/file;Windows: C:\Some\Path\file) or relative and starting with a dot (Unix/Linux:./some/other/path/file; Windows: .\Some\Other\Path\file).

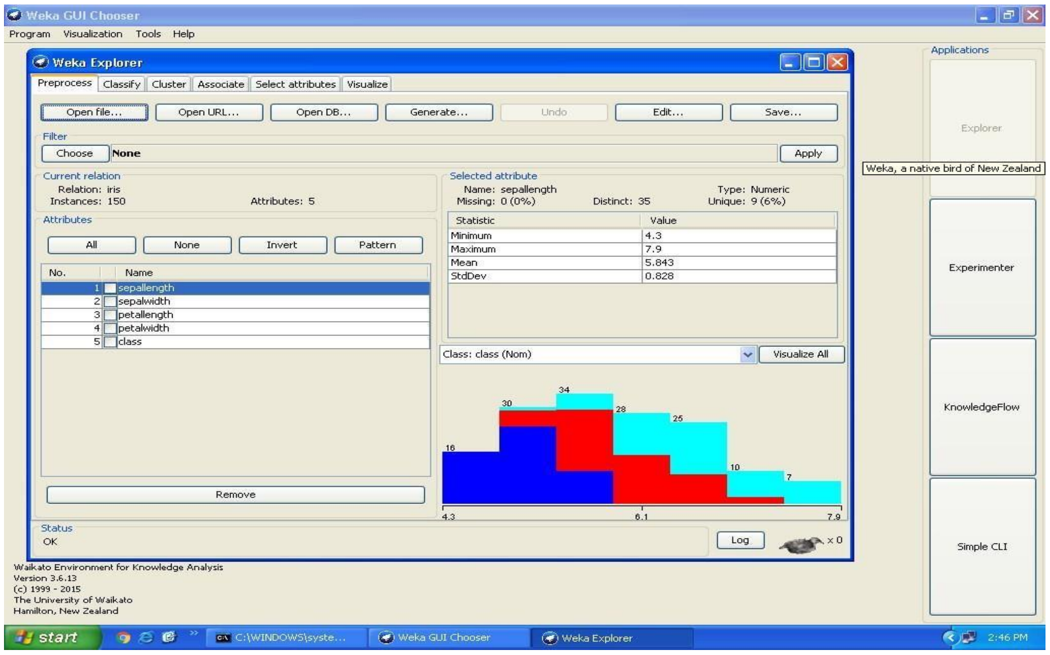
B.(iii)Navigate the options available in the WEKA(ex.select attributes panel,preprocess panel,classify panel,cluster panel,associate panel and visualize)

Ans: Steps for identify options in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose iris data set and open file.
8. All tabs available in WEKA home page.



Waikato Environment for Knowledge Analysis
Version 3.6.13
(c) 1999 - 2015
The University of Waikato
Hamilton, New Zealand



Filter: None

Current relation
Relation: iris
Instances: 150 Attributes: 5

Attributes

No.	Name
1	sepallength
2	sepalwidth
3	petallength
4	petalwidth
5	class

Selected attribute
Name: sepallength Type: Numeric
Missing: 0 (0%) Distinct: 35 Unique: 9 (6%)

Statistic	Value
Minimum	4.3
Maximum	7.9
Mean	5.843
StdDev	0.828

Class: class (Nom)

Status
OK

x 0

Waikato Environment for Knowledge Analysis
Version 3.6.13
(c) 1999 - 2015
The University of Waikato
Hamilton, New Zealand

A. (iv) Study the ARFF file format

Ans: **ARFF File Format**

An ARFF (= Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes.

ARFF files are not the only format one can load, but all files that can be converted with **Weka's "core converters". The following formats are currently** supported:

- ARFF (+ compressed)
- C4.5
- CSV
- libsvm
- binary serialized instances
- XRFF (+ compressed)

Overview

ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information. The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types.

An example header on the standard IRIS dataset looks like this:

1. Title: Iris Plants Database

2. Sources:

(a) Creator: R.A. Fisher
(b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
(c) Date: July, 1988

@RELATION iris
@ATTRIBUTE sepal length NUMERIC
@ATTRIBUTE sepal width NUMERIC
@ATTRIBUTE petal length NUMERIC
@ATTRIBUTE petal width NUMERIC
@ATTRIBUTE class {Iris-setosa, Iris-versicolor, Iris-irginica} The Data of the ARFF file looks like the following:

@DATA

5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa

Lines that begin with a % are comments.
The @RELATION, @ATTRIBUTE and @DATA declarations are case insensitive.

The ARFF Header Section

The ARFF Header section of the file contains the relation declaration and at•
tribute declarations.

The @relation Declaration

The relation name is defined as the first line in the ARFF file. The format is: @relation <relation-name>

where <relation-name> is a string. The string must be quoted if the name includes spaces.

The @attribute Declarations

Attribute declarations take the form of an ordered sequence of @attribute statements. Each attribute in the data set has its own @attribute statement which uniquely defines the name **of that attribute and it's data type. The order** the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found in the third comma delimited column.

The format for the @attribute statement is:

@attribute <attribute-name> <datatype>

where the <attribute-name> must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted.

The <datatype> can be any of the four types supported by Weka:

- numeric
- integer is treated as numeric
- real is treated as numeric
- <nominal-specification>
- string
- date [<date-format>]
- relational for multi-instance data (for future use)

where <nominal-specification> and <date-format> are defined below. The keywords numeric, real, integer, string and date are case insensitive.

Numeric attributes

Numeric attributes can be real or integer numbers.

Nominal attributes

Nominal values are defined by providing an <nominal-specification> listing the possible values: <nominal-name1>, <nominal-name2>, <nominal-name3>,
For example, the class value of the Iris dataset can be defined as follows: @ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica} Values that contain spaces must be quoted.

String attributes

String attributes allow us to create attributes containing arbitrary textual values. This is very useful in text-mining applications, as we can create datasets with string attributes, then write Weka Filters to manipulate strings (like String- ToWordVectorFilter). String attributes are declared as follows:

@ATTRIBUTE LCC string

Date attributes

Date attribute declarations take the form: @attribute <name> date [<date-format>] where <name> is the name for the attribute and <date-format> is an optional string specifying how date values should be parsed and printed (this is the same format used by SimpleDateFormat). The default format string accepts the ISO-8601 combined date and time format: yyyy-MM-**dd'T'HH:mm:ss.** Dates must be specified in the data section as the corresponding string representations of the date/time (see example below).

Relational attributes

Relational attribute declarations take the form: @attribute <name> relational
<further attribute definitions> @end <name>
For the multi-instance dataset MUSK1 the definition would look like thi**s ("...")** denotes an omission):
@attribute molecule_name {MUSK-jf78,...,NON-MUSK-199} @attribute bag relational
@attribute f1 numeric
...
@attribute f166 numeric @end bag

```
@attribute class {0,1}
```

The ARFF Data Section

The ARFF Data section of the file contains the data declaration line and the actual instance lines.

The @data Declaration

The @data declaration is a single line denoting the start of the data segment in the file. The format is:

@data

The instance data

Each instance is represented on a single line, with carriage returns denoting the end of the instance. A percent sign (%) introduces a comment, which continues to the end of the line.

Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth @attribute declaration is always the nth field of the attribute).

Missing values are represented by a single question mark, as in:

@data 4.4,?,1.5,?,Iris-setosa

Values of string and nominal attributes are case sensitive, and any that contain space or the comment-delimiter character % must be quoted. (The code suggests that double-quotes are acceptable and that a backslash will escape individual characters.)

An example follows: @relation LCCvsLCSH @attribute LCC string @attribute LCSH string

```
@data
```

AG5, ’Encyclopedias and dictionaries.;Twentieth century.’ AS262, ’Science -- Soviet Union -- **History.’**
AE5, ’Encyclopedias and dictionaries.’
AS281, ’Astronomy, Assyro-Babylonian.;Moon -- **Phases.’**
AS28**1, ’Astronomy, Assyro**-Babylonian.;Moon -- **Tables.’**

Dates must be specified in the data section using the string representation specified in the attribute declaration.

For example:
@RELATION Timestamps
@ATTRIBUTE timestamp DATE "yyyy-MM-dd HH:mm:ss" @DATA

"2001-04-03 12:12:12"
"2001-05-03 12:59:55"

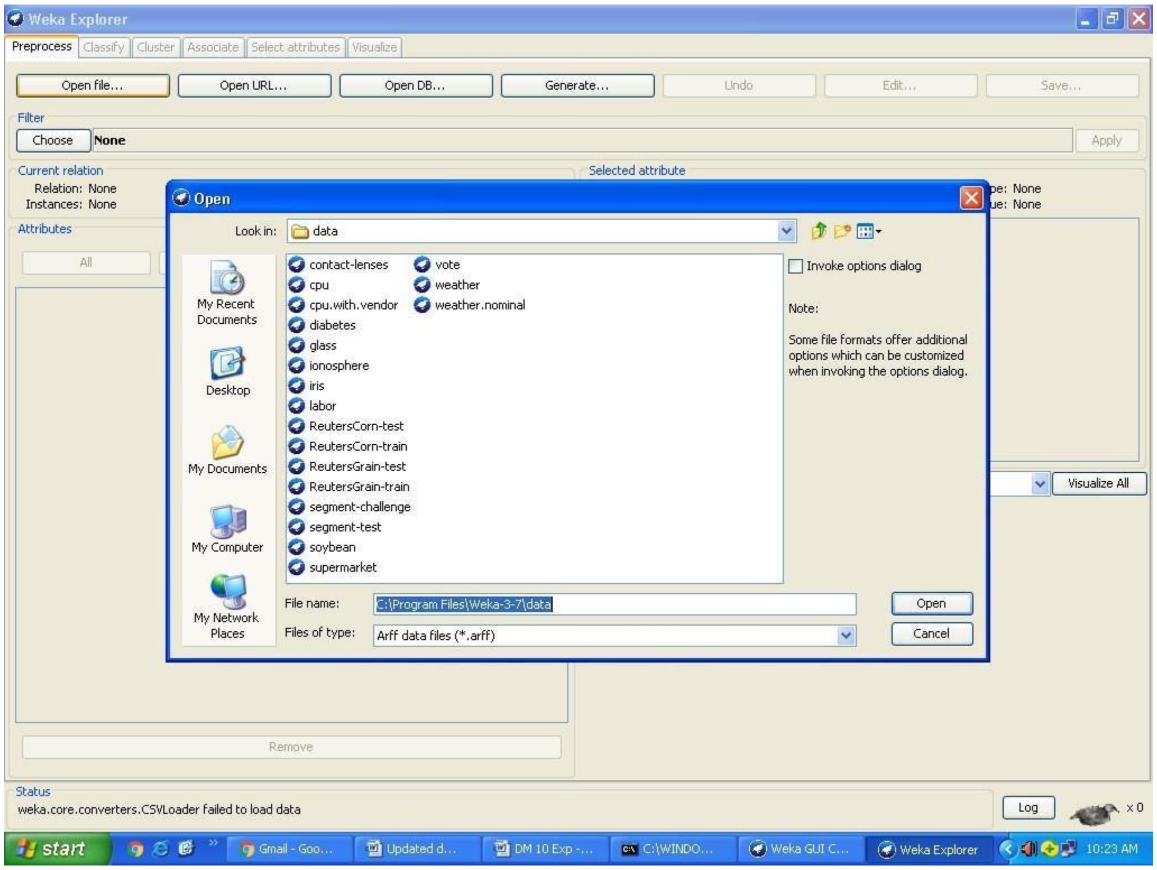
Relational data must be enclosed within double quotes ”. For example an instance of the MUSK1 **dataset (”...” denotes an omission):**

MUSK-188,"42,...,30",1

B.(v) Explore the available data sets in WEKA.

Ans: Steps for identifying data sets in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on open file button.
4. Choose WEKA folder in C drive.
5. Select and Click on data option button.



.

Sample Weka Data Sets

Below are some sample WEKA data sets, in arff format.

- contact-lens.arff
- cpu.arff
- cpu.with-vendor.arff
- diabetes.arff
- glass.arff
- ionospehre.arff
- iris.arff
- labor.arff
- ReutersCorn-train.arff

- ReutersCorn-test.arff
- ReutersGrain-train.arff
- ReutersGrain-test.arff
- segment-challenge.arff
- segment-test.arff
- soybean.arff
- supermarket.arff
- vote.arff
- weather.arff
- weather.nominal.arff

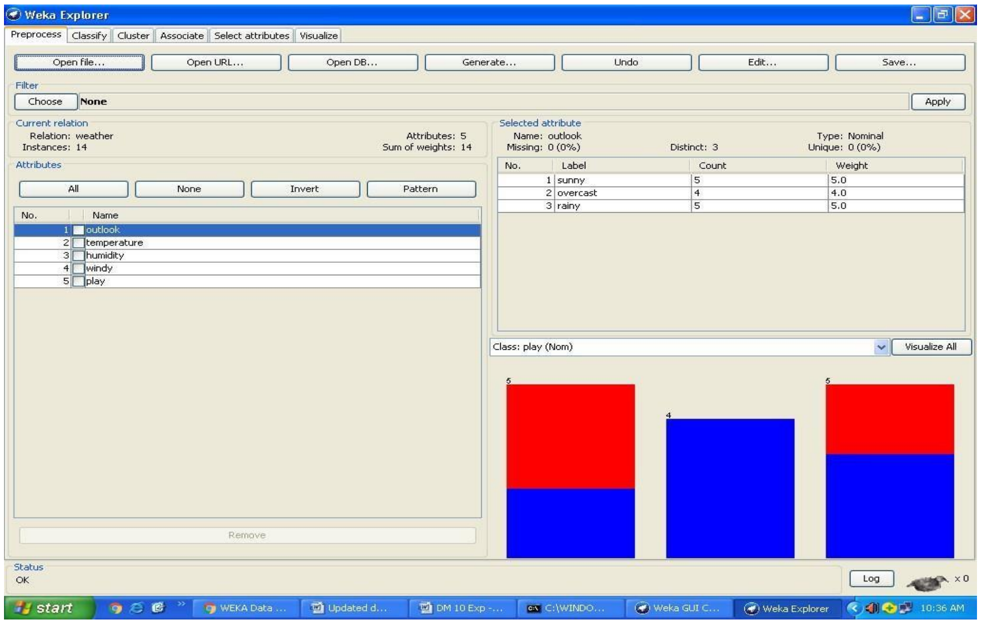
B. (vi) Load a data set (ex.Weather dataset,Iris dataset,etc.)

Ans: Steps for load the Weather data set.

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on open file button.
4. Choose WEKA folder in C drive.
5. Select and Click on data option button.
6. Choose Weather.arff file and Open the file.

Steps for load the Iris data set.

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on open file button.
4. Choose WEKA folder in C drive.
5. Select and Click on data option button.
6. Choose Iris.arff file and Open the file.



Filter: None

Current relation
Relation: weather
Instances: 14
Attributes: 5
Sum of weights: 14

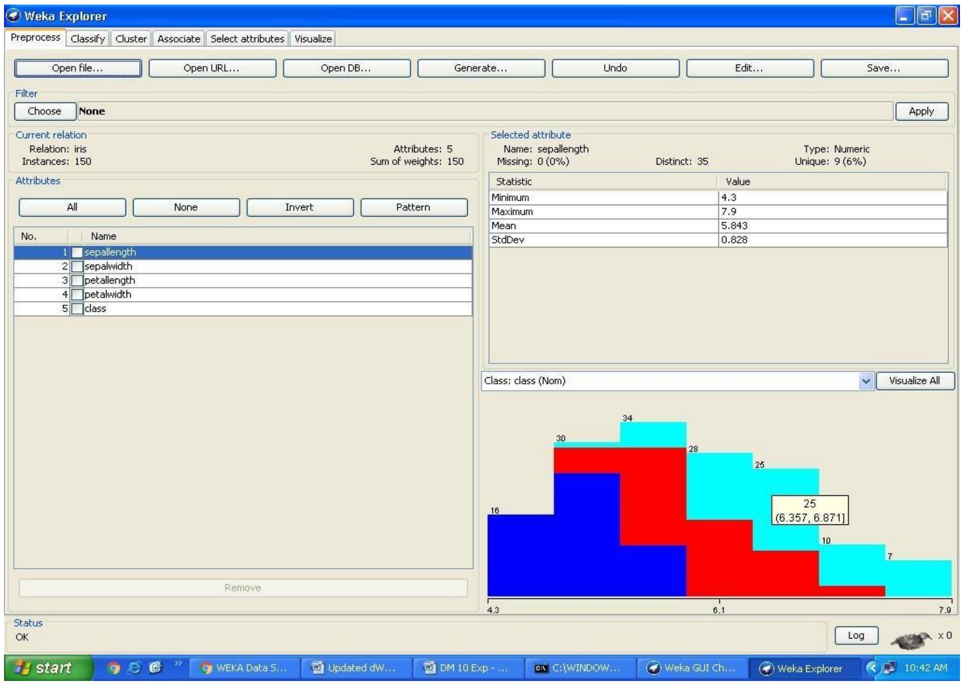
Selected attribute
Name: outlook
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	sunny	5	5.0
2	overcast	4	4.0
3	rainy	5	5.0

No.	Name
1	outlook
2	temperature
3	humidity
4	windy
5	play

Class: play (Nom)

Status
OK



Filter: None

Current relation
Relation: iris
Instances: 150
Attributes: 5
Sum of weights: 150

Selected attribute
Name: sepallength
Missing: 0 (0%)
Distinct: 35
Type: Numeric
Unique: 9 (6%)

Statistic	Value
Minimum	4.3
Maximum	7.9
Mean	5.843
StdDev	0.828

No.	Name
1	sepallength
2	sepalwidth
3	petallength
4	petalwidth
5	class

Class: class (Nom)

Status
OK

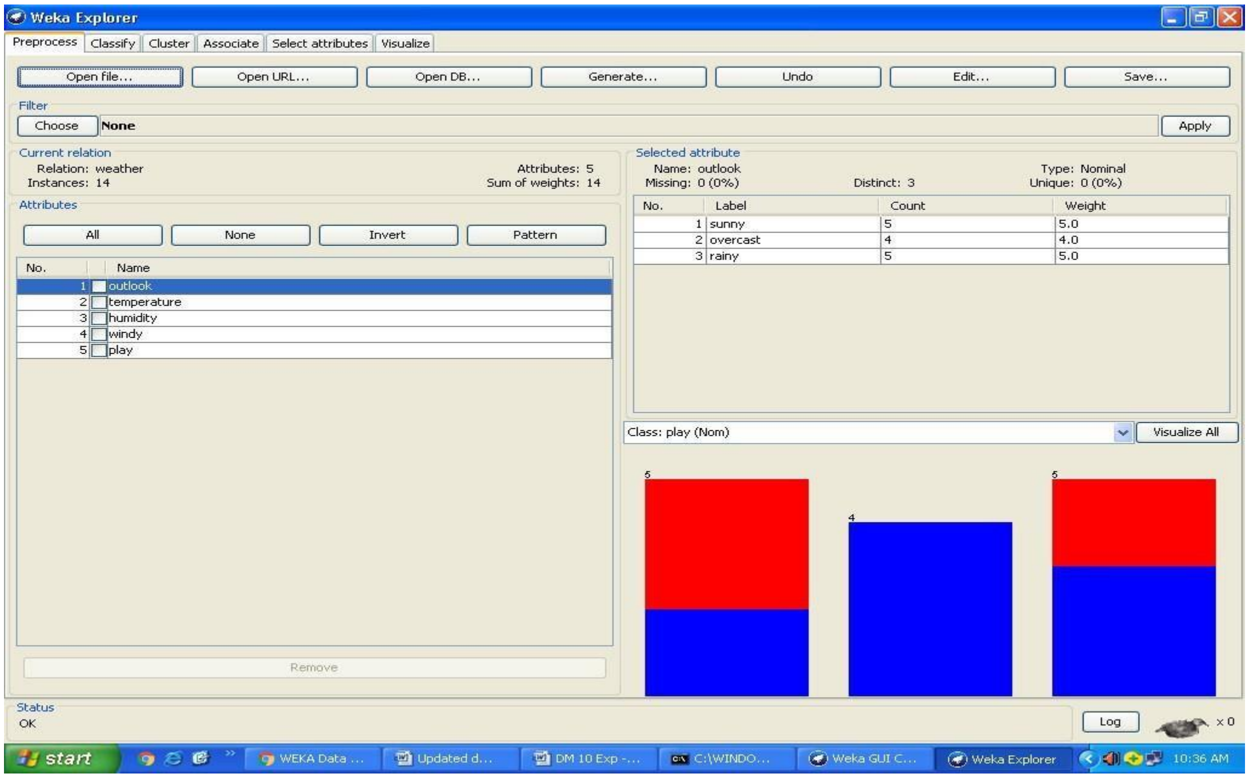
B. (vii) Load each dataset and observe the following:

B. (vii.i) List attribute names and they types

Ans: Example dataset-Weather.arff

List out the attribute names:

1. outlook
2. temperature
3. humidity
4. windy
5. play



Filter: None

Current relation
Relation: weather
Instances: 14
Attributes: 5
Sum of weights: 14

Attributes

No.	Name
1	outlook
2	temperature
3	humidity
4	windy
5	play

Selected attribute
Name: outlook
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	sunny	5	5.0
2	overcast	4	4.0
3	rainy	5	5.0

Class: play (Nom)

Status
OK

B. (vii.ii) Number of records in each dataset.

Ans: @relation weather.symbolic

```
@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}
@data
sunny,hot,high,FALSE,no
sunny,hot,high,TRUE,no
overcast,hot,high,FALSE,yes
rainy,mild,high,FALSE,yes
rainy,cool,normal,FALSE,yes
rainy,cool,normal,TRUE,no
overcast,cool,normal,TRUE,yes
sunny,mild,high,FALSE,no
sunny,cool,normal,FALSE,yes
rainy,mild,normal,FALSE,yes
sunny,mild,normal,TRUE,yes
overcast,mild,high,TRUE,yes
overcast,hot,normal,FALSE,yes
rainy,mild,high,TRUE,no
```

B. (vii.iii) Identify the class attribute (if any)

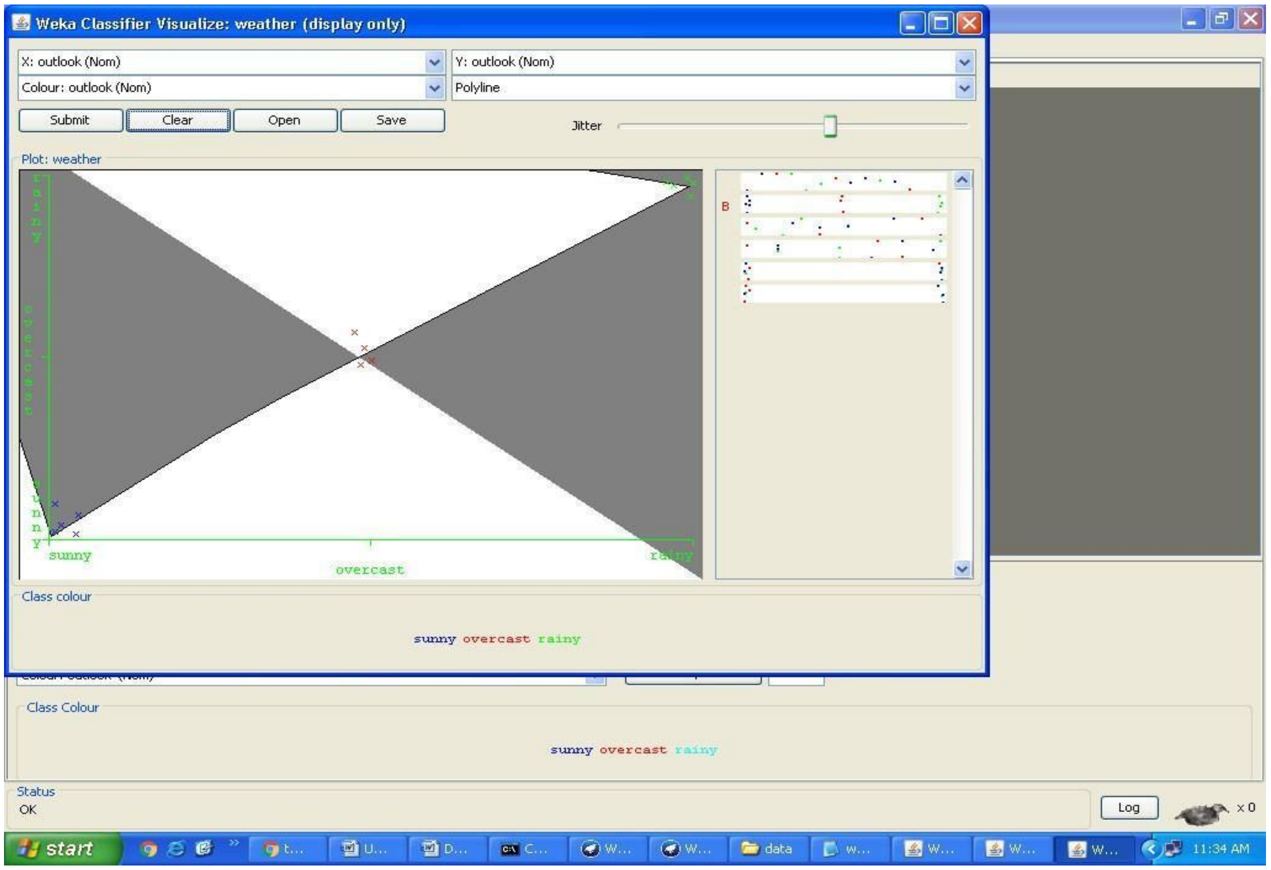
Ans: class attributes

1. sunny
2. overcast
3. rainy

B. (vii.iv) Plot Histogram

Ans: Steps for identify the plot histogram

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Visualize button.
4. Click on right click button.
5. Select and Click on polyline option button.



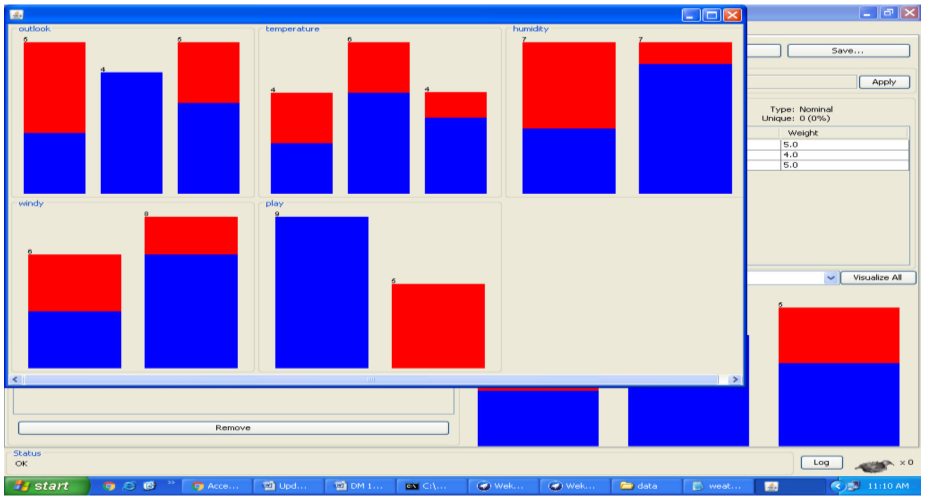
B. (vii.v) Determine the number of records for each class

Ans: @relation weather.symbolic
@data

sunny,hot,high,FALSE,no
sunny,hot,high,TRUE,no
overcast,hot,high,FALSE,yes
rainy,mild,high,FALSE,yes
rainy,cool,normal,FALSE,yes
rainy,cool,normal,TRUE,no
overcast,cool,normal,TRUE,yes
sunny,mild,high,FALSE,no
sunny,cool,normal,FALSE,yes
rainy,mild,normal,FALSE,yes
sunny,mild,normal,TRUE,yes
overcast,mild,high,TRUE,yes
overcast,hot,normal,FALSE,yes
rainy,mild,high,TRUE,no

B. (vii.vi) Visualize the data in various dimensions

Click on Visualize All button in WEKA Explorer.



Type: Nominal
Unique: 0 (0%)

Weight
5.0
4.0
5.0

Status
OK

Experiment-3:

3. Perform data preprocessing tasks and Demonstrate performing association rule mining on data sets

A. Explore various options in Weka for Preprocessing data and apply (like Discretization Filters, Resample filter, etc.) n each dataset.

Ans:

Preprocess Tab

1. Loading Data

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

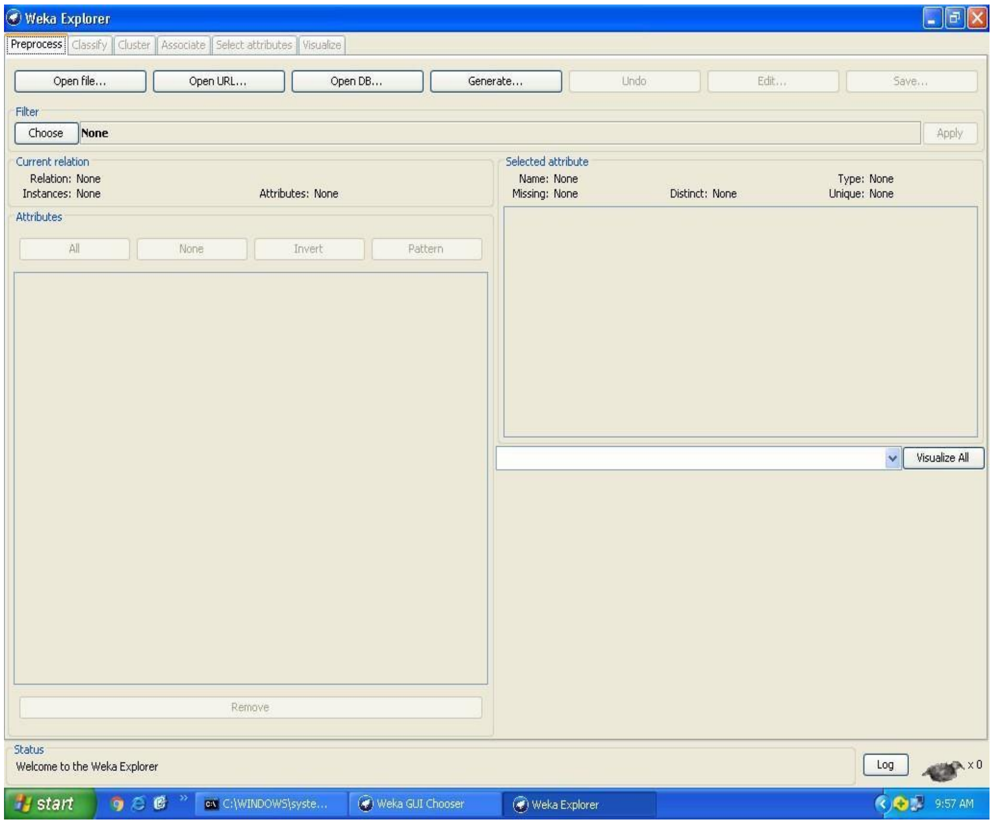
1. Open file.... Brings up a dialog box allowing you to browse for the data file on the local file system.

2. Open URL.... Asks for a Uniform Resource Locator address for where the data is stored.

3. Open DB.... Reads data from a database. (Note that to make this work you might have to edit the file in weka/experiment/DatabaseUtils.props.)

4. Generate.... Enables you to generate artificial data from a variety of Data Generators. Using the Open file... button you can read files in a variety of formats: **WEKA's ARFF format, CSV**

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



Current Relation: Once some data has been loaded, the Preprocess panel shows a variety of in**formation. The Current relation box (the "current relation" is the** currently loaded data, which can be interpreted as a single relational table in database terminology) has three entries:

1. Relation. The name of the relation, as given in the file it was loaded from. Filters (described below) modify the name of a relation.

2. Instances. The number of instances (data points/records) in the data.

3. Attributes. The number of attributes (features) in the data.

Working With Attributes

Below the Current relation box is a box titled Attributes. There are four buttons, and beneath them is a list of the attributes in the current relation.

The list has three columns:

1. **No..** A number that identifies the attribute in the order they are specified in the data file.

2. **Selection tick boxes**. These allow you select which attributes are present in the relation.
3. **Name.** The name of the attribute, as it was declared in the data file. When you click on different rows in the list of attributes, the fields change in the box to the right titled Selected attribute.

This box displays the characteristics of the currently highlighted attribute in the list:

1. **Name.** The name of the attribute, the same as that given in the attribute list.

2. **Type.** The type of attribute, most commonly Nominal or Numeric.

3. **Missing.** The number (and percentage) of instances in the data for which this attribute is missing (unspecified).
4. **Distinct.** The number of different values that the data contains for this attribute.

5. **Unique.** The number (and percentage) of instances in the data having a value for this attribute that no other instances have.

Below these statistics is a list showing more information about the values stored in this attribute, which differ depending on its type. If the attribute is nominal, the list consists of each possible value for the attribute along with the number of instances that have that value. If the attribute is numeric, the list gives four statistics describing the distribution of values in the data—the minimum, maximum, mean and standard deviation. And below these statistics there is a coloured histogram, colour-coded according to the attribute chosen as the Class using the box above the histogram. (This box will bring up a drop-down list of available selections when clicked.) Note that only nominal Class attributes will result in a colour-coding. Finally, after pressing the Visualize All button, histograms for all the attributes in the data are shown in a separate window.

Returning to the attribute list, to begin with all the tick boxes are unticked.

They can be toggled on/off by clicking on them individually. The four buttons above can also be used to change the selection:

PREPROCESSING

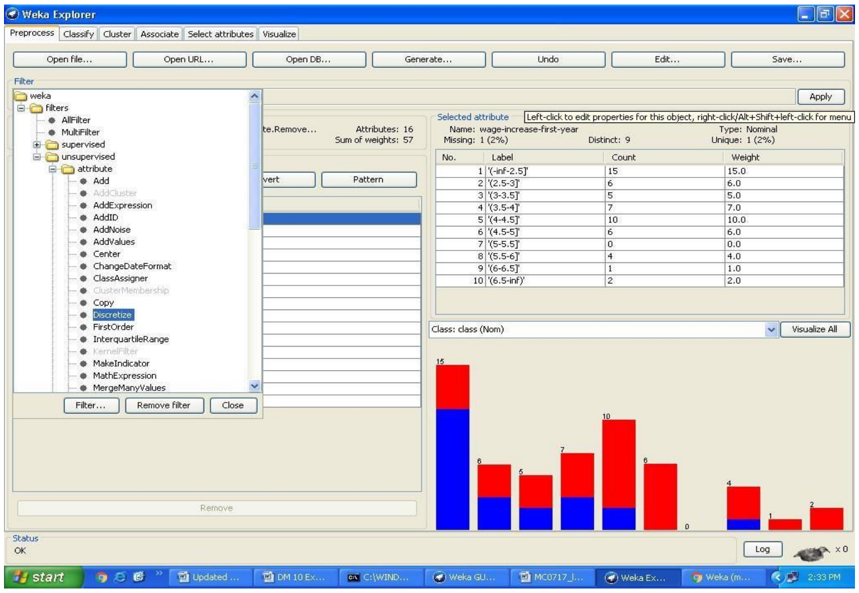
1. **All.** All boxes are ticked.
2. **None.** All boxes are cleared (unticked).
3. **Invert.** Boxes that are ticked become unticked and vice versa.

4. **Pattern.** Enables the user to select attributes based on a Perl 5 Regular Expression. E.g., .* id selects all attributes which name ends with id.

Once the desired attributes have been selected, they can be removed by clicking the Remove button below the list of attributes. Note that this can be undone by clicking the Undo button, which is located next to the Edit button in the top-right corner of the Preprocess panel.

Working with Filters:-

The preprocess section allows filters to be defined that transform the data in various ways. The Filter box is used to set up the filters that are required. At the left of the Filter box is a Choose button. By clicking this button it is possible to select one of the filters in WEKA. Once a filter has been selected, its name and options are shown in the field next to the Choose button. Clicking on this box with the left mouse button brings up a GenericObjectEditor dialog box. A click with the right mouse button (or Alt+Shift+left click) brings up a menu where you can choose, either to display the properties in a GenericObjectEditor dialog box, or to copy the current setup string to the clipboard.



Selected attribute
Left-click to edit properties for this object, right-click/Alt+Shift+left-click for menu

Name: wage-increase-first-year Type: Nominal
Missing: 1 (2%) Distinct: 9 Unique: 1 (2%)

No.	Label	Count	Weight
1	'(-inf-2.5]'	15	15.0
2	'(2.5-3]'	6	6.0
3	'(3-3.5]'	5	5.0
4	'(3.5-4]'	7	7.0
5	'(4-4.5]'	10	10.0
6	'(4.5-5]'	6	6.0
7	'(5-5.5]'	0	0.0
8	'(5.5-6]'	4	4.0
9	'(6-6.5]'	1	1.0
10	'(6.5-inf)'	2	2.0

Class: class (Nom)

The GenericObjectEditor Dialog Box

The GenericObjectEditor dialog box lets you configure a filter. The same kind of dialog box is used to configure other objects, such as classifiers and clusterers

(see below). The fields in the window reflect the available options.

Right-clicking (or Alt+Shift+Left-Click) on such a field will bring up a popup menu, listing the following options:

1. Show properties... has the same effect as left-clicking on the field, i.e., a dialog appears allowing you to alter the settings.

2. Copy configuration to clipboard copies the currently displayed configuration string to the **system's clipboar**d and therefore can be used anywhere else in WEKA or in the console. This is rather handy if you have to setup complicated, nested schemes.

3. Enter configuration... is the "receiving" end for configurations that got copied to the clipboard earlier on. In this dialog you can enter a class name followed by options (if the class supports these). This also allows you to transfer a filter setting from the Preprocess panel to a Filtered Classifier used in the Classify panel.

Left-Clicking on any of these gives an opportunity to alter the filters settings. For example, the setting may take a text string, in which case you type the string into the text field provided. Or it may give a drop-down box listing several states to choose from. Or it may do something else, depending on the information required. Information on the options is provided in a tool tip if you let the mouse pointer hover of the corresponding field. More information on the filter and its options can be obtained by clicking on the More button in the About panel at the top of the GenericObjectEditor window.

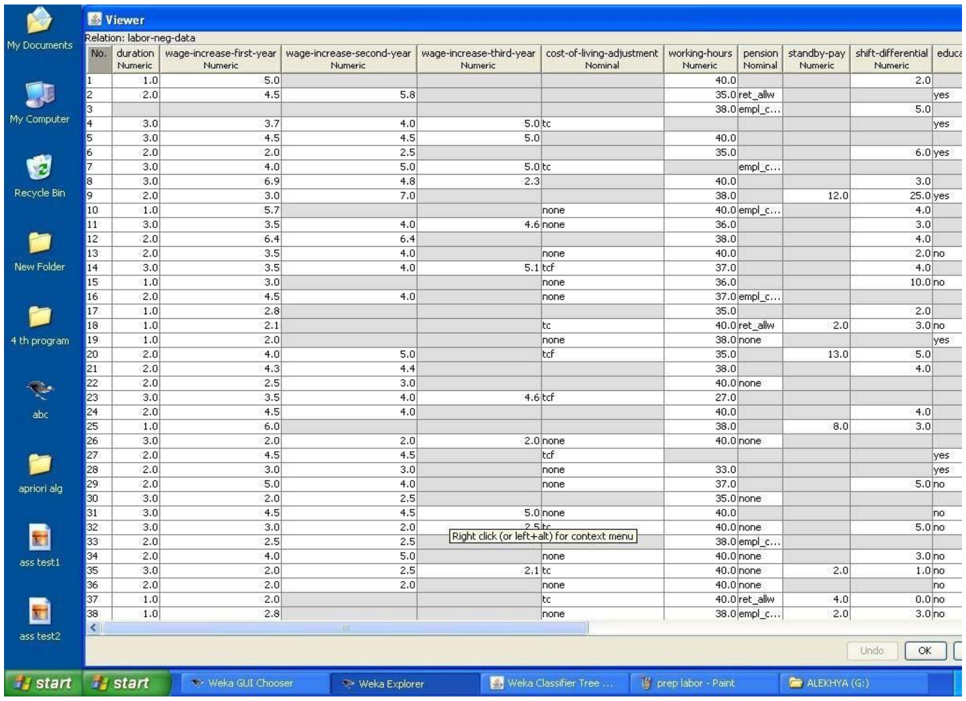
Applying Filters

Once you have selected and configured a filter, you can apply it to the data by pressing the Apply button at the right end of the Filter panel in the Preprocess panel. The Preprocess panel will then show the transformed data. The change can be undone by pressing the Undo button. You can also use the Edit...button to modify your data manually in a dataset editor. Finally, the Save... button at the top right of the Preprocess panel saves the current version of the relation in file formats that can represent the relation, allowing it to be kept for future use.

- Steps for run preprocessing tab in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose labor data set and open file.
8. Choose filter button and select the Unsupervised-Discritize option and apply

Dataset labor.arff

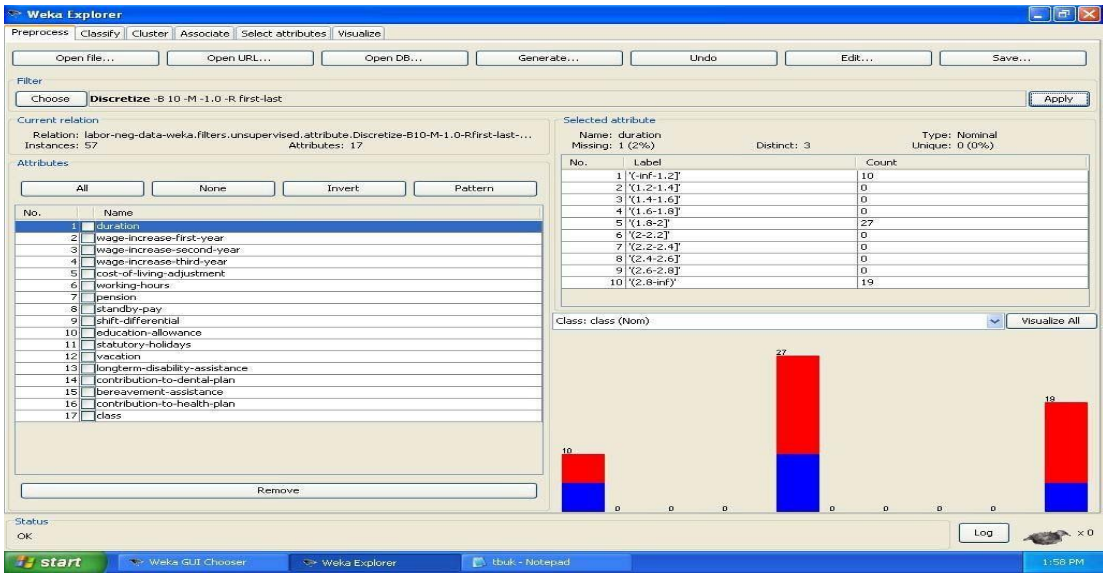


Viewer

Relation: labor-neg-data

No.	duration Numeric	wage-increase-first-year Numeric	wage-increase-second-year Numeric	wage-increase-third-year Numeric	cost-of-living-adjustment Nominal	working-hours Numeric	pension Nominal	standby-pay Numeric	shift-differential Numeric	educa
1	1.0	5.0				40.0			2.0	
2	2.0	4.5	5.8			35.0	ret_allw			yes
3						38.0	empl_c...		5.0	
4	3.0	3.7	4.0	5.0	tc					yes
5	3.0	4.5	4.5	5.0		40.0				
6	2.0	2.0	2.5			35.0			6.0	yes
7	3.0	4.0	5.0	5.0	tc		empl_c...			
8	3.0	6.9	4.8	2.3		40.0			3.0	
9	2.0	3.0	7.0			38.0		12.0	25.0	yes
10	1.0	5.7			none	40.0	empl_c...		4.0	
11	3.0	3.5	4.0	4.6	none	36.0			3.0	
12	2.0	6.4	6.4			38.0			4.0	
13	2.0	3.5	4.0		none	40.0			2.0	no
14	3.0	3.5	4.0	5.1	tcf	37.0			4.0	
15	1.0	3.0			none	36.0			10.0	no
16	2.0	4.5	4.0		none	37.0	empl_c...			
17	1.0	2.8				35.0			2.0	
18	1.0	2.1			tc	40.0	ret_allw	2.0	3.0	no
19	1.0	2.0			none	38.0	none			yes
20	2.0	4.0	5.0		tcf	35.0		13.0	5.0	
21	2.0	4.3	4.4			38.0			4.0	
22	2.0	2.5	3.0			40.0	none			
23	3.0	3.5	4.0	4.6	tcf	27.0				
24	2.0	4.5	4.0			40.0			4.0	
25	1.0	6.0				38.0		8.0	3.0	
26	3.0	2.0	2.0	2.0	none	40.0	none			
27	2.0	4.5	4.5		tcf					yes
28	2.0	3.0	3.0		none	33.0				yes
29	2.0	5.0	4.0		none	37.0			5.0	no
30	3.0	2.0	2.5			35.0	none			
31	3.0	4.5	4.5	5.0	none	40.0				no
32	3.0	3.0	2.0	2.5	tc	40.0	none		5.0	no
33	2.0	2.5	2.5			38.0	empl_c...			
34	2.0	4.0	5.0		none	40.0	none		3.0	no
35	3.0	2.0	2.5	2.1	tc	40.0	none	2.0	1.0	no
36	2.0	2.0	2.0		none	40.0	none			no
37	1.0	2.0			tc	40.0	ret_allw	4.0	0.0	no
38	1.0	2.8			none	38.0	empl_c...	2.0	3.0	no

The following screenshot shows the effect of discretization



Weka Explorer

Filter: Discretize -B 10 -M -1.0 -R first-last

Current relation
Relation: labor-neg-data-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-...
Instances: 57 Attributes: 17

Attributes

No.	Name
1	duration
2	wage-increase-first-year
3	wage-increase-second-year
4	wage-increase-third-year
5	cost-of-living-adjustment
6	working-hours
7	pension
8	standby-pay
9	shift-differential
10	education-allowance
11	statutory-holidays
12	vacation
13	longterm-disability-assistance
14	contribution-to-dental-plan
15	bereavement-assistance
16	contribution-to-health-plan
17	class

Selected attribute
Name: duration Type: Nominal
Missing: 1 (2%) Distinct: 3 Unique: 0 (0%)

No.	Label	Count
1	'(-inf-1.2]'	10
2	'(1.2-1.4]'	0
3	'(1.4-1.6]'	0
4	'(1.6-1.8]'	0
5	'(1.8-2]'	27
6	'(2-2.2]'	0
7	'(2.2-2.4]'	0
8	'(2.4-2.6]'	0
9	'(2.6-2.8]'	0
10	'(2.8-inf)'	19

Class: class (Nom)

Status
OK

B. Load each dataset into Weka and run Aprior algorithm with different support and confidence values. Study the rules generated.

Ans:

Steps for run Aprior algorithm in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose Weather data set and open file.
8. Click on Associate tab and Choose Aprior algorithm
9. Click on start button.

Output : === Run information ===

```
Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c •
1
Relation:     weather.symbolic
Instances:    14
Attributes: 5
outlook
temperature
humidity
windy
play
=== Associator model (full training set) ===
Apriori
=======

Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17
```

Generated sets of large itemsets:

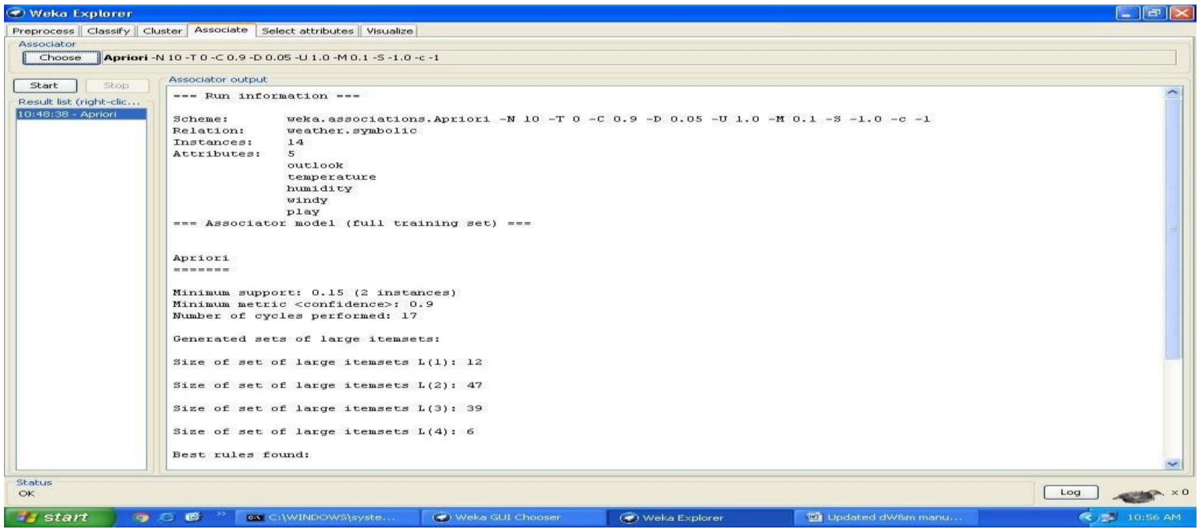
Size of set of large itemsets L(1): 12

Size of set of large itemsets L(2): 47
Size of set of large itemsets L(3): 39

Size of set of large itemsets L(4): 6

Best rules found:

1. outlook=overcast 4 ==> play=yes 4 conf:(1)
2. temperature=cool 4 ==> humidity=normal 4 conf:(1)
3. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1)
4. outlook=sunny play=no 3 ==> humidity=high 3 conf:(1)
5. outlook=sunny humidity=high 3 ==> play=no 3 conf:(1)
6. outlook=rainy play=yes 3 ==> windy=FALSE 3 conf:(1)
7. outlook=rainy windy=FALSE 3 ==> play=yes 3 conf:(1)
8. temperature=cool play=yes 3 ==> humidity=normal 3 conf:(1)
9. outlook=sunny temperature=hot 2 ==> humidity=high 2 conf:(1)
10. temperature=hot play=no 2 ==> outlook=sunny 2 conf:(1)



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

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     weather.symbolic
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

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

Size of set of large itemsets L(2): 47

Size of set of large itemsets L(3): 39

Size of set of large itemsets L(4): 6

Best rules found:
```

Association Rule:

An association rule has two parts, an antecedent (if) and a consequent (then). An antecedent is an item found in the data. A consequent is an item that is found in combination with the antecedent.

Association rules are created by analyzing data for frequent if/then patterns and using the criteriasupport and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true.

In data mining, association rules are useful for analyzing and predicting customer behavior. They play an important part in shopping basket data analysis, product clustering, catalog design and store layout.

Support and Confidence values:

- Support count: The support count of an itemset *X*, denoted by *X.count*, in a data set *T* is the number of transactions in *T* that contain *X*. Assume *T* has *n* transactions.
- Then,

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

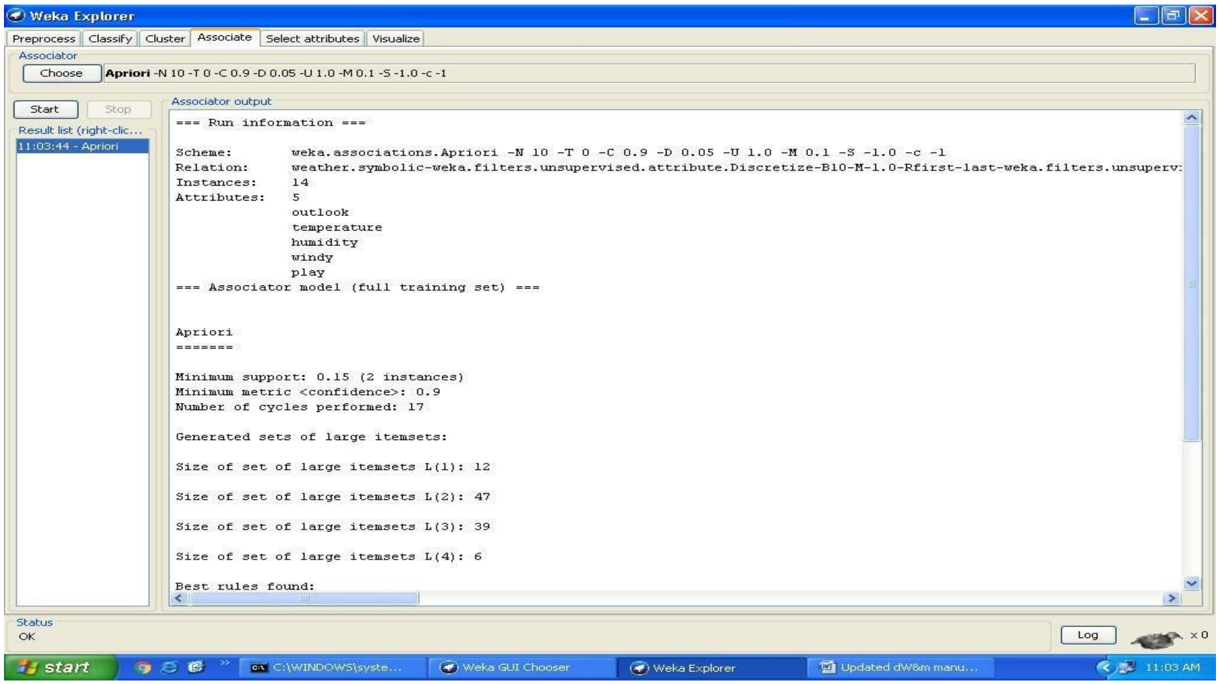
support = support({*A* U *C*})

confidence = support({*A* U *C*})/support({*A*})

C. Apply different discretization filters on numerical attributes and run the Aprior association rule algorithm. Study the rules generated. Derive interesting insights and observe the effect of discretization in the rule generation process.

Ans: Steps for run Aprior algorithm in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose Weather data set and open file.
8. Choose filter button and select the Unsupervised-Discritize option and apply
9. Click on Associate tab and Choose Aprior algorithm
10. Click on start button.



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

```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation:     weather.symbolic-weka.filters.unsupervised.attribute.Discretize-B10-M-1.0-Rfirst-last-weka.filters.unsuperv:
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

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

Size of set of large itemsets L(2): 47

Size of set of large itemsets L(3): 39

Size of set of large itemsets L(4): 6

Best rules found:
```

Output :
```
=== Run information ===

Scheme:       weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c •
1
Relation:     weather.symbolic
Instances:    14
Attributes: 5
outlook
temperature
humidity
windy
play
=== Associator model (full training set) ===
Apriori
=======
Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

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

Size of set of large itemsets L(2): 47
Size of set of large itemsets L(3): 39

Size of set of large itemsets L(4): 6

Best rules found:

1. outlook=overcast 4 ==> play=yes 4    conf:(1)
2. temperature=cool 4 ==> humidity=normal 4    conf:(1)
3. humidity=normal windy=FALSE 4 ==> play=yes 4    conf:(1)
4. outlook=sunny play=no 3 ==> humidity=high 3    conf:(1)
5. outlook=sunny humidity=high 3 ==> play=no 3    conf:(1)
6. outlook=rainy play=yes 3 ==> windy=FALSE 3    conf:(1)
7. outlook=rainy windy=FALSE 3 ==> play=yes 3    conf:(1)
```

Experiment-4:

4. Demonstrate performing classification on data sets.

Classification Tab

Selecting a Classifier

At the top of the classify section is the Classifier box. This box has a text fieldthat gives the name of the currently selected classifier, and its options. Clicking on the text box with the left mouse button brings up a GenericObjectEditor dialog box, just the same as for filters, that you can use to configure the options of the current classifier. With a right click (or Alt+Shift+left click) you can once again copy the setup string to the clipboard or display the properties in a GenericObjectEditor dialog box. The Choose button allows you to choose one of the classifiers that are available in WEKA.

Test Options

The result of applying the chosen classifier will be tested according to the options that are set by clicking in the Test options box. There are four test modes:

1. Use training set. The classifier is evaluated on how well it predicts the class of the instances it was trained on.

2. Supplied test set. The classifier is evaluated on how well it predicts the class of a set of instances loaded from a file. Clicking the Set... button brings up a dialog allowing you to choose the file to test on.

3. Cross-validation. The classifier is evaluated by cross-validation, using the number of folds that are entered in the Folds text field.
4. Percentage split. The classifier is evaluated on how well it predicts a certain percentage of the data which is held out for testing. The amount of data held out depends on the value entered in the % field.

Classifier Evaluation Options:

1. Output model. The classification model on the full training set is output so that it can be viewed, visualized, etc. This option is selected by default.

2. Output per-class stats. The precision/recall and true/false statistics for each class are output. This option is also selected by default.

3. Output entropy evaluation measures. Entropy evaluation measures are included in the output. This option is not selected by default.
4. Output confusion matrix. The confusion matrix of the classifier's predictions is included in the output. This option is selected by default.

5. Store predictions for visualization. Th**e classifier's predictions are** remembered so that they can be visualized. This option is selected by default.

6. Output predictions. The predictions on the evaluation data are output.

Note that in the case of a cross-validation the instance numbers do not correspond to the location in the data!

7. Output additional attributes. If additional attributes need to be output alongside the

predictions, e.g., an ID attribute for tracking misclassifications, then the index of this attribute can be specified here. The usual **Weka ranges are supported,"first" and "last" are therefore valid** indices as **well (example: "first**-3,6,8,12-**last").**

8. Cost-sensitive evaluation. The errors is evaluated with respect to a cost matrix. The Set... button allows you to specify the cost matrix used.

9. Random seed for xval / % Split. This specifies the random seed used when randomizing the data before it is divided up for evaluation purposes.

10. Preserve order for % Split. This suppresses the randomization of the data before splitting into train and test set.

11. Output source code. If the classifier can output the built model as Java source code, you can specify the class name here. The code will be printed **in the "Classifier output" area.**

The Class Attribute

The classifiers in **WEKA are designed to be trained to predict a single 'class'** attribute, which is the target for prediction. Some classifiers can only learn nominal classes; others can only learn numeric classes (regression problems) still others can learn both.

By default, the class is taken to be the last attribute in the data. If you want

to train a classifier to predict a different attribute, click on the box below the Test options box to bring up a drop-down list of attributes to choose from.

Training a Classifier

Once the classifier, test options and class have all been set, the learning process is started by clicking on the Start button. While the classifier is busy being trained, the little bird moves around. You can stop the training process at any time by clicking on the Stop button. When training is complete, several things happen. The Classifier output area to the right of the display is filled with text describing the results of training and testing. A new entry appears in the Result list box. We look at the result list below; but first we investigate the text that has been output.

A. Load each dataset into Weka and run id3, j48 classification algorithm, study the classifier output. Compute entropy values, Kappa ststistic.

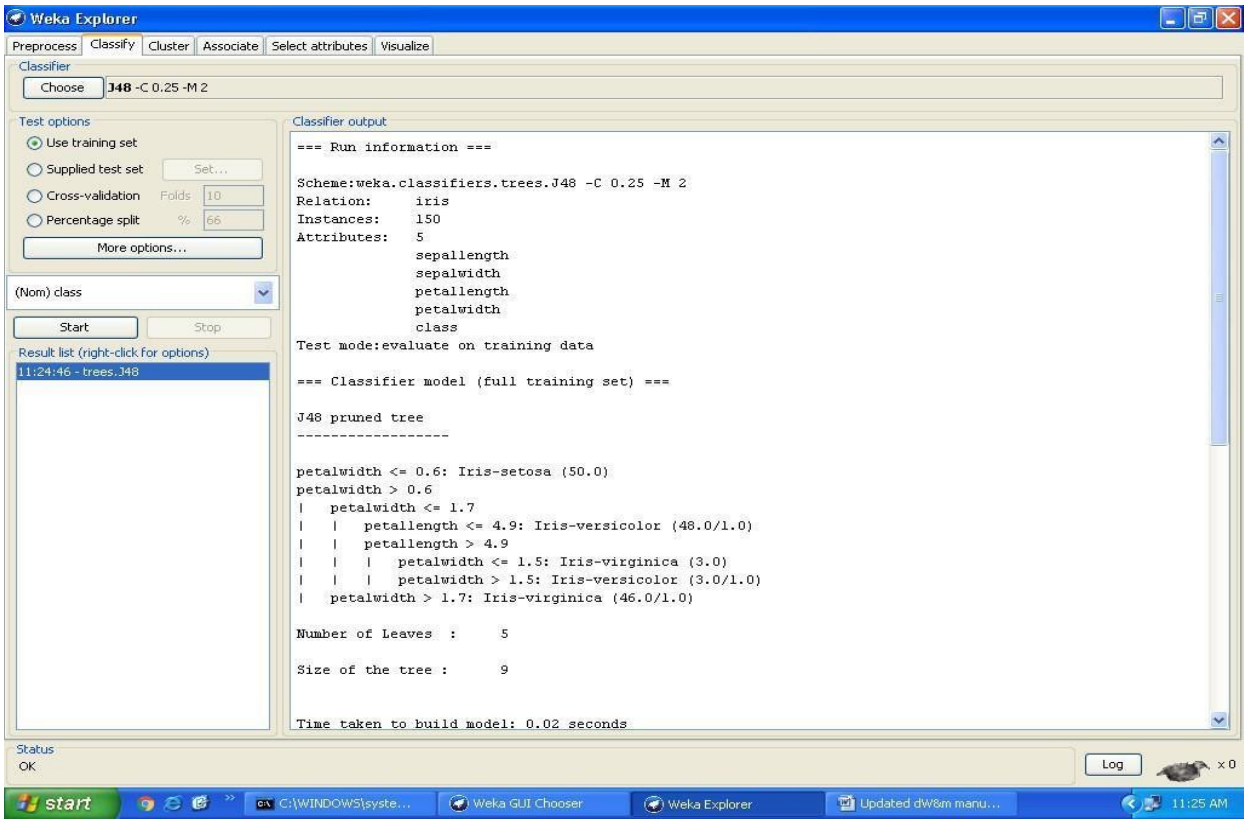
Ans:

- Steps for run ID3 and J48 Classification algorithms in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose iris data set and open file.
8. Click on classify tab and Choose J48 algorithm and select use training set test option.
9. Click on start button.
10. Click on classify tab and Choose ID3 algorithm and select use training set test option.
11. Click on start button.

Output:

```
=== Run information ===

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

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

J48 pruned tree
------------------

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

Number of Leaves  : 5

Size of the tree :          9


Time taken to build model: 0 seconds

=== Evaluation on training set ===
=== Summary ===
Correctly Classified Instances         147               98      %
Incorrectly Classified Instances         3                2      %
Kappa statistic                      0.97
K&B Relative Info Score              14376.1925 %
K&B Information Score                  227.8573 bits      1.519 bits/instance
Class complexity | order 0           237.7444 bits      1.585 bits/instance
Class complexity | scheme              16.7179 bits      0.1115 bits/instance
Complexity improvement     (Sf)        221.0265 bits      1.4735 bits/instance
Mean absolute error                   0.0233
Root mean squared error                 0.108
Relative absolute error               5.2482 %
Root relative squared error             22.9089 %
Total Number of Instances               150

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision   Recall  F-Measure   ROC Area  Class
1         0          1         1         1          1       Iris-setosa
 0.98      0.02       0.961     0.98      0.97       0.99     Iris-versicolor
 0.96      0.01       0.98      0.96      0.97       0.99    Iris-virginica
Weighted Avg.    0.98      0.01       0.98      0.98      0.98       0.993

=== Confusion Matrix ===

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



J48 -C 0.25 -M 2

```
=== Run information ===

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

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

J48 pruned tree
------------------

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

Number of Leaves  :     5

Size of the tree :      9


Time taken to build model: 0.02 seconds
```

The Classifier Output Text

The text in the Classifier output area has scroll bars allowing you to browse the results. Clicking with the left mouse button into the text area, while holding Alt and Shift, brings up a dialog that enables you to save the displayed output

in a variety of formats (currently, BMP, EPS, JPEG and PNG). Of course, you can also resize the Explorer window to get a larger display area.

The output is

Split into several sections:

1. Run information. A list of information giving the learning scheme options, relation name, instances, attributes and test mode that were involved in the process.

2. Classifier model (full training set). A textual representation of the classification model that was produced on the full training data.

3. The results of the chosen test mode are broken down thus.

4. Summary. A list of statistics summarizing how accurately the classifier was able to predict the true class of the instances under the chosen test mode.

5. Detailed Accuracy By Class. A more detailed per-class break down **of the classifier's** prediction accuracy.

6. Confusion Matrix. Shows how many instances have been assigned to each class. Elements show the number of test examples whose actual class is the row and whose predicted class is the column.

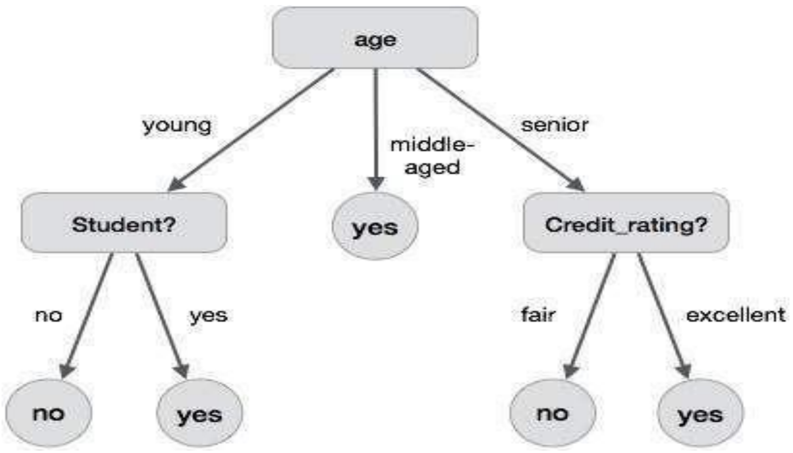
7. Source code (optional). This section lists the Java source code if one **chose "Output source code" in the "More options" dialog.**

B.Extract if-then rues from decision tree gentrated by classifier, Observe the confusion matrix and derive Accuracy, F- measure, TPrate, FPrate , Precision and recall values. Apply cross-validation strategy with various fold levels and compare the accuracy results.

Ans:

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

The following decision tree is for the concept buy_computer that indicates whether a customer at a company is likely to buy a computer or not. Each internal node represents a test on an attribute. Each leaf node represents a class.



The benefits of having a decision tree are as follows −

- It does not require any domain knowledge.
- It is easy to comprehend.
- The learning and classification steps of a decision tree are simple and fast.

IF-THEN Rules:

Rule-based classifier makes use of a set of IF-THEN rules for classification. We can express a rule in the following from −

IF condition THEN conclusion
Let us consider a rule R1,

```
R1: IF age=youth AND student=yes

THEN buy_computer=yes
```

Points to remember −

- The IF part of the rule is called **rule antecedent** or**precondition**.
- The THEN part of the rule is called **rule consequent**.
- The antecedent part the condition consist of one or more attribute tests and these tests are logically ANDed.
- The consequent part consists of class prediction.

Note − We can also write rule R1 as follows:

```
R1: (age = youth) ^ (student = yes))(buys computer = yes)
```

If the condition holds true for a given tuple, then the antecedent is satisfied.

RuleExtraction

Here we will learn how to build a rule-based classifier by extracting IF-THEN rules from a decision tree.

Points to remember −

- One rule is created for each path from the root to the leaf node.
- To form a rule antecedent, each splitting criterion is logically ANDed.
- The leaf node holds the class prediction, forming the rule consequent.

RuleInductionUsingSequentialCovering Algorithm

Sequential Covering Algorithm can be used to extract IF-THEN rules form the training data. We do not require to generate a decision tree first. In this algorithm, each rule for a given class covers many of the tuples of that class.

Some of the sequential Covering Algorithms are AQ, CN2, and RIPPER. As per the general strategy the rules are learned one at a time. For each time rules are learned, a tuple covered by the rule is removed and the process continues for the rest of the tuples. This is because the path to each leaf in a decision tree corresponds to a rule.

Note − The Decision tree induction can be considered as learning a set of rules simultaneously.

The Following is the sequential learning Algorithm where rules are learned for one class at a time. When learning a rule from a class Ci, we want the rule to cover all the tuples from class C only and no tuple form any other class.

```
Algorithm: Sequential Covering

Input:
D, a data set class-labeled tuples,
Att_vals, the set of all attributes and their possible values.
```

```
Output:  A Set of IF-THEN rules.
Method:
Rule_set={ }; // initial set of rules learned is empty

for each class c do

repeat
Rule = Learn_One_Rule(D, Att_valls, c);
remove tuples covered by Rule form D;
until termination condition;

Rule_set=Rule_set+Rule; // add a new rule to rule-set
end for
return Rule_Set;
```

Rule Pruning

The rule is pruned is due to the following reason −

- The Assessment of quality is made on the original set of training data. The rule may perform well on training data but less well on subsequent data. That's why the rule pruning is required.
- The rule is pruned by removing conjunct. The rule R is pruned, if pruned version of R has greater quality than what was assessed on an independent set of tuples.

FOIL is one of the simple and effective method for rule pruning. For a given rule R,

FOIL_Prune = pos - neg / pos + neg
where pos and neg is the number of positive tuples covered by R, respectively.

Note − This value will increase with the accuracy of R on the pruning set. Hence, if the FOIL_Prune value is higher for the pruned version of R, then we prune R.

- Steps for run decision tree algorithms in WEKA

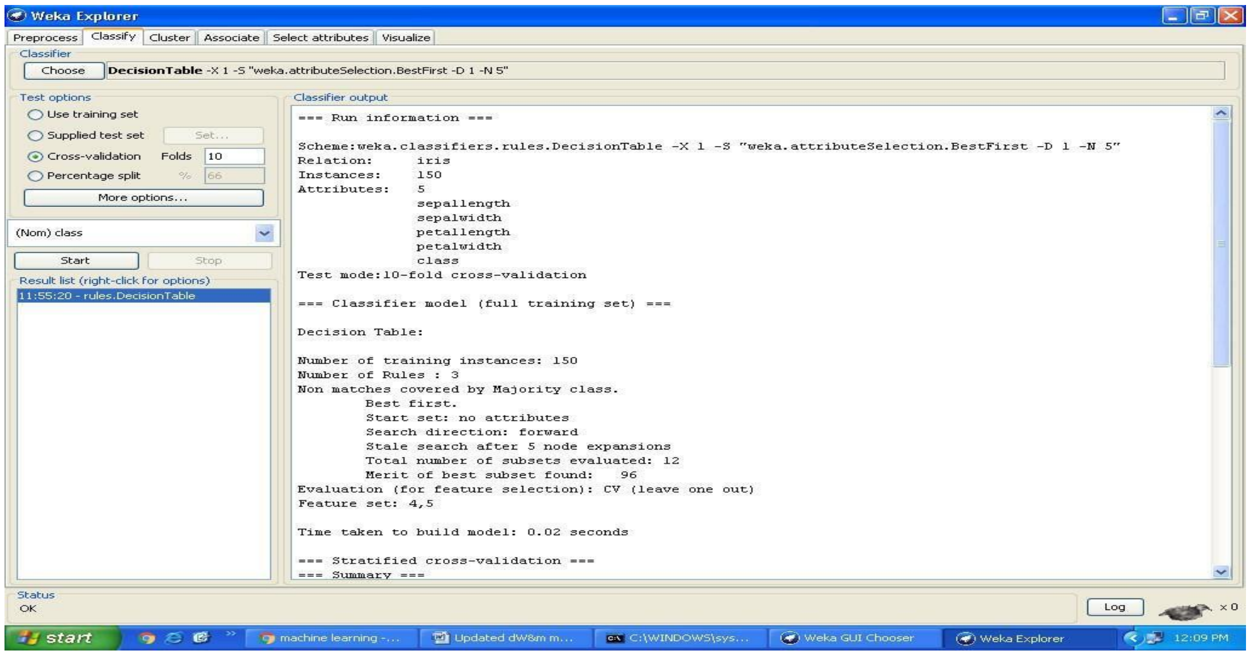
  1. Open WEKA Tool.
  2. Click on WEKA Explorer.
  3. Click on Preprocessing tab button.

4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose iris data set and open file.
8. Click on classify tab and Choose decision table algorithm and select cross-validation folds value-10 test option.
9. Click on start button.

Output:

```
=== Run information ===
Scheme:weka.classifiers.rules.DecisionTable -X 1 -S "weka.attributeSelection.BestFirst -D
1 -N 5"
Relation:     iris
Instances:    150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode:10-fold cross-validation

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

Decision Table:

Number of training instances: 150
Number of Rules : 3
Non matches covered by Majority class.
Best first.
Start set: no attributes
Search direction: forward
Stale search after 5 node expansions
Total number of subsets evaluated: 12
Merit of best subset found:  96
Evaluation (for feature selection): CV (leave one out)
Feature set: 4,5
```

```
Time taken to build model: 0.02 seconds

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

Correctly Classified Instances         139               92.6667 %
Incorrectly Classified Instances        11                7.3333 %
Kappa statistic                  0.89
Mean absolute error                  0.092
Root mean squared error                0.2087
Relative absolute error             20.6978 %
Root relative squared error           44.2707 %
Total Number of Instances             150

=== Detailed Accuracy By Class ===

TP Rate  FP Rate   Precision   Recall  F-Measure   ROC Area  Class
1            0            1           1            1         1            Iris-setosa
0.88      0.05         0.898       0.88        0.889     0.946       Iris-versicolor
0.9       0.06         0.882       0.9         0.891     0.947       Iris-virginica
Weighted Avg.    0.927     0.037        0.927        0.927         0.927      0.964

=== Confusion Matrix ===

a  b  c  <-- classified as
50  0  0 |  a = Iris-setosa
0 44  6 |  b = Iris-versicolor
0  5 45 |  c = Iris-virginica
```



Classifier: DecisionTable -X 1 -S "weka.attributeSelection.BestFirst -D 1 -N 5"

Test options: Use training set ( ) Supplied test set ( ) [✓ Cross-validation] Folds 10 Percentage split ( ) % 66

(Nom) class

Result list (right-click for options): 11:55:20 - rules.DecisionTable

Status: OK

```
=== Run information ===

Scheme:weka.classifiers.rules.DecisionTable -X 1 -S "weka.attributeSelection.BestFirst -D 1 -N 5"
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:10-fold cross-validation

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

Decision Table:

Number of training instances: 150
Number of Rules : 3
Non matches covered by Majority class.
        Best first.
        Start set: no attributes
        Search direction: forward
        Stale search after 5 node expansions
        Total number of subsets evaluated: 12
        Merit of best subset found:   96
Evaluation (for feature selection): CV (leave one out)
Feature set: 4,5

Time taken to build model: 0.02 seconds

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

C. Load each dataset into Weka and perform Naïve-bayes classification and k-Nearest Neighbor classification, Interpret the results obtained.

Ans:

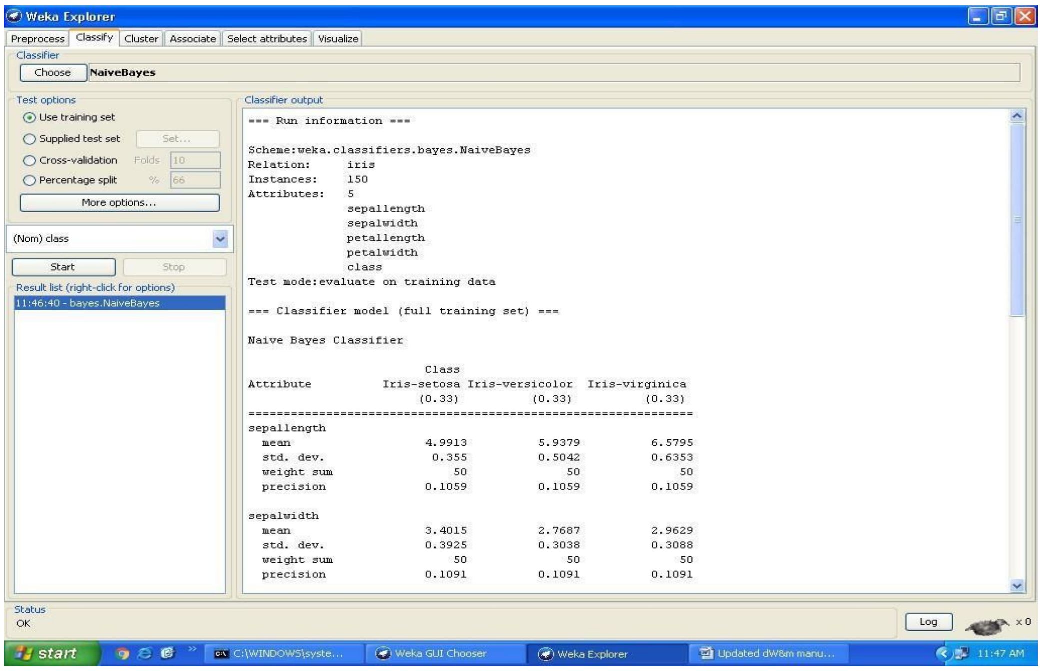
- Steps for run Naïve-bayes and k-nearest neighbor Classification algorithms in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose iris data set and open file.
8. Click on classify tab and Choose Naïve-bayes algorithm and select use training set test option.
9. Click on start button.
10. Click on classify tab and Choose k-nearest neighbor and select use training set test option.
11. Click on start button.

Output: Naïve Bayes

```
=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayes
Relation:     iris
Instances:    150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode:evaluate on training data


=== 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 training set ===


=== Summary ===
Correctly Classified Instances        144            96    %
Incorrectly Classified Instances        6             4    %
Kappa statistic                      0.94
Mean absolute error                     0.0324
Root mean squared error                   0.1495
Relative absolute error                 7.2883 %
Root relative squared error              31.7089 %
Total Number of Instances                 150

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision   Recall  F-Measure   ROC Area  Class
1        0        1        1        1        1       Iris-setosa
0.96     0.04     0.923    0.96     0.941    0.993   Iris-versicolor
0.92     0.02     0.958    0.92     0.939    0.993   Iris-virginica
Weighted Avg.   0.96    0.02     0.96     0.96     0.96     0.995

=== Confusion Matrix ===
a b c  <-- classified as
50  0  0 |  a = Iris-setosa
0 48  2 |  b = Iris-versicolor
0  4 46 |  c = Iris-virginica.
```



Classifier: NaiveBayes

Test options: [✓ Use training set] Folds 10, % 66

(Nom) class

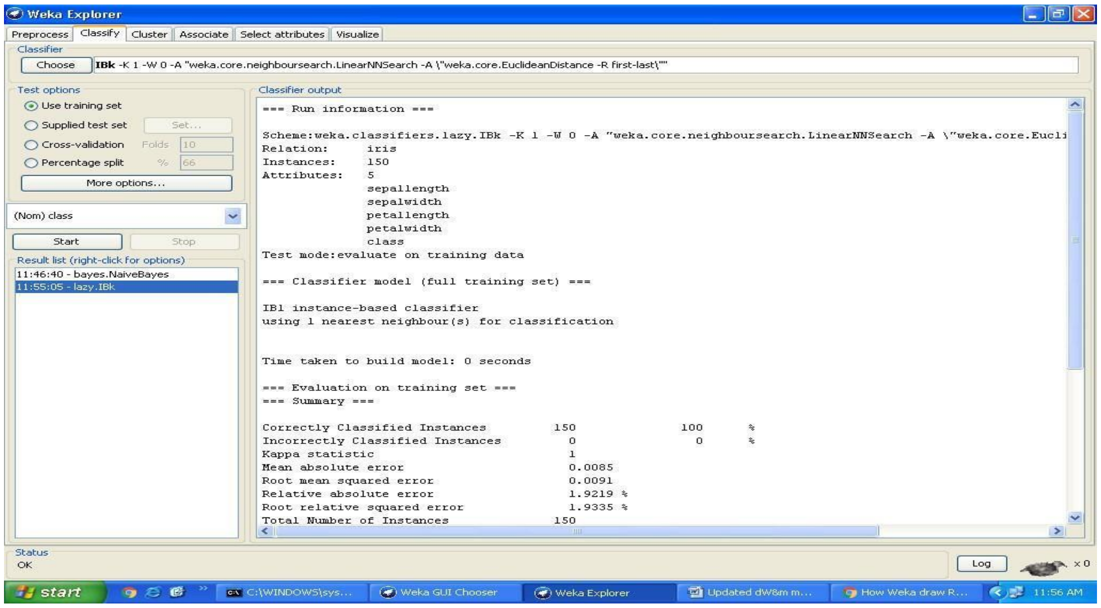
Result list: 11:46:40 - bayes.NaiveBayes

```
=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayes
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:evaluate on training data

=== 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
```

Output: KNN (IBK)

```
=== Run information ===

Scheme:weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.EuclideanDistance -R first-last\""
Relation:     iris
Instances:    150
Attributes: 5
sepallength
sepalwidth
petallength
```

```
petalwidth
class
Test mode:evaluate on training data

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

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances        150              100      %
Incorrectly Classified Instances        0                0      %
Kappa statistic                         1
Mean absolute error                     0.0085
Root mean squared error                 0.0091
Relative absolute error                 1.9219 %
Root relative squared error             1.9335 %
Total Number of Instances             150

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision   Recall  F-Measure   ROC Area  Class
1        0        1           1       1           1         Iris-setosa
1        0        1           1       1           1         Iris-versicolor
1        0        1           1       1           1         Iris-virginica
Weighted Avg.    1        0        1        1        1        1

=== Confusion Matrix ===

a  b  c  <-- classified as
50  0  0 |  a = Iris-setosa
0 50  0 |  b = Iris-versicolor
0  0 50 |  c = Iris-virginica
```



IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""

[✓ Use training set] Supplied test set; Cross-validation Folds 10; Percentage split % 66

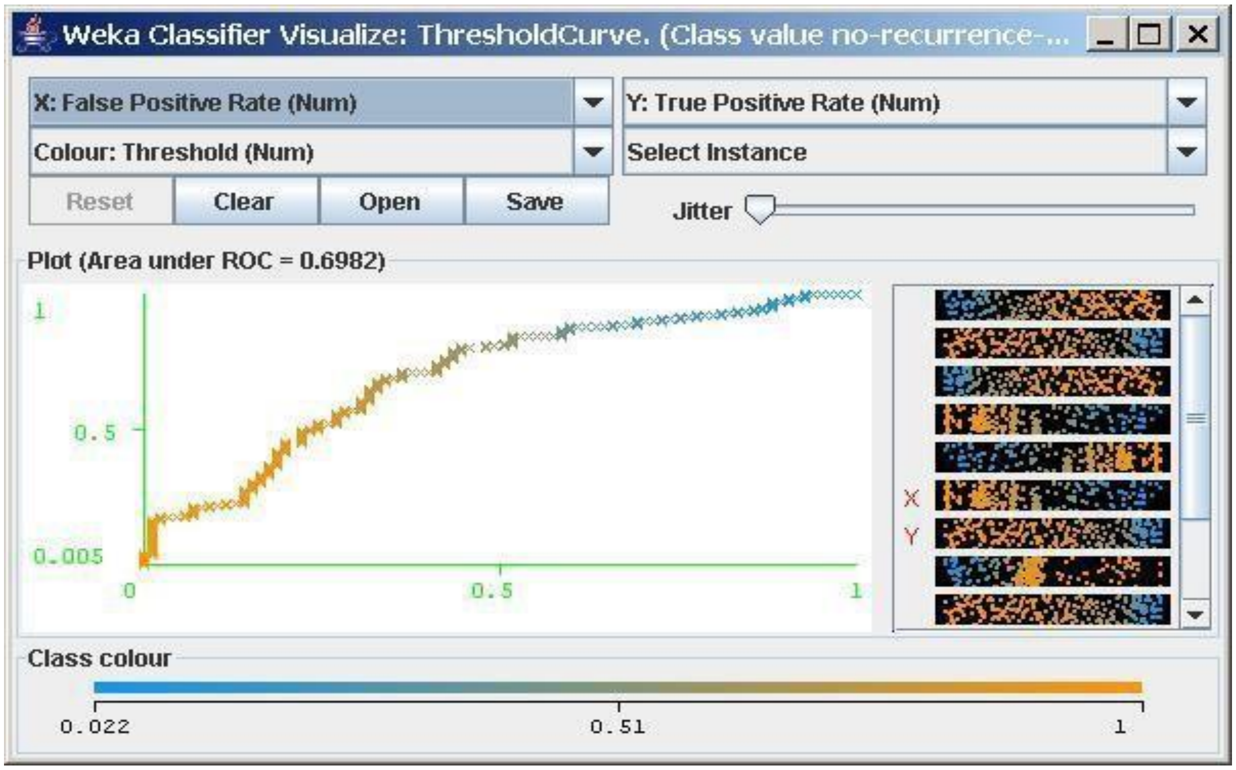
11:46:40 - bayes.NaiveBayes
11:55:05 - lazy.IBk

```
=== Run information ===

Scheme:weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.Eucli
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:evaluate on training data

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

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances         150               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0085
Root mean squared error                  0.0091
Relative absolute error                  1.9219 %
Root relative squared error              1.9335 %
Total Number of Instances              150
```

D. Plot RoC Curves.

Ans: Steps for identify the plot RoC Curves.

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Visualize button.
4. Click on right click button.
5. Select and Click on polyline option button.


Weka Classifier Visualize: ThresholdCurve. (Class value no-recurrence-...
X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot (Area under ROC = 0.6982)
1
0.5
0.005
0
0.5
1
X
Y
Class colour
0.022
0.51
1

E. Compare classification results of ID3,J48, Naïve-Bayes and k-NN classifiers for each dataset , and reduce which classifier is performing best and poor for each dataset and justify.

Ans:

- Steps for run ID3 and J48 Classification algorithms in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose iris data set and open file.
8. Click on classify tab and Choose J48 algorithm and select use training set test option.

9. Click on start button.
10. Click on classify tab and Choose ID3 algorithm and select use training set test option.
11. Click on start button.
12. Click on classify tab and Choose Naïve-bayes algorithm and select use training set test option.
13. Click on start button.
14. Click on classify tab and Choose k-nearest neighbor and select use training set test option.
15. Click on start button.

J48:

```
=== Run information ===

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

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

Number of Leaves  : 5
```

```
Size of the tree :          9

Time taken to build model: 0 seconds

=== Evaluation on training set ===
=== Summary ===

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

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision   Recall  F-Measure   ROC Area  Class
1         0         1         1         1         1        Iris-setosa
0.98      0.02      0.961     0.98      0.97       0.99     Iris-versicolor
0.96      0.01      0.98      0.96      0.97       0.99     Iris-virginica
Weighted Avg.    0.98      0.01       0.98      0.98      0.98       0.993

=== Confusion Matrix ===

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

Naïve-bayes:

```
=== Run information ===

Scheme:weka.classifiers.bayes.NaiveBayes
Relation:     iris
Instances:    150
Attributes:   5
sepallength
```

```
sepalwidth
petallength
petalwidth
class
Test mode:evaluate on training data
=== 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 training set ===
=== Summary ===
Correctly Classified Instances         144               96      %
Incorrectly Classified Instances         6                4      %
Kappa statistic                        0.94
Mean absolute error                    0.0324
Root mean squared error                0.1495
Relative absolute error                7.2883 %
Root relative squared error           31.7089 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===
TP Rate  FP Rate  Precision   Recall  F-Measure   ROC Area  Class
1         0          1         1         1         1       Iris-setosa
0.96      0.04      0.923     0.96      0.941     0.993    Iris-versicolor
0.92      0.02      0.958     0.92      0.939     0.993    Iris-virginica
Weighted Avg.    0.96      0.02       0.96      0.96      0.96       0.995

=== Confusion Matrix ===
a  b  c  <-- classified as
50  0  0 |  a = Iris-setosa
0 48  2 |  b = Iris-versicolor
0  4 46 |  c = Iris-virginica
```

K-Nearest Neighbor (IBK):

```
=== Run information ===
Scheme:weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.EuclideanDistance -R first-last\""
Relation:     iris
Instances:    150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode:evaluate on training data

=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds
=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances        150              100     %
Incorrectly Classified Instances        0                0     %
Kappa statistic                     1
Mean absolute error                    0.0085
Root mean squared error                  0.0091
Relative absolute error                1.9219 %
Root relative squared error              1.9335 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision   Recall  F-Measure   ROC Area  Class
1        0         1        1        1        1       Iris-setosa
1        0         1        1        1        1       Iris-versicolor
1        0         1        1        1        1       Iris-virginica
Weighted Avg.    1        0        1        1        1        1

=== Confusion Matrix ===

a  b  c  <-- classified as
50  0  0 |  a = Iris-setosa
0 50  0 |  b = Iris-versicolor
0  0 50 |  c = Iris-virginica
```

Experiment:5

5. Demonstrate performing clustering on data sets Clustering Tab

Selecting a Clusterer

By now you will be familiar with the process of selecting and configuring objects. Clicking on the clustering scheme listed in the Clusterer box at the top of the

window brings up a GenericObjectEditor dialog with which to choose a new clustering scheme.

Cluster Modes

The Cluster mode box is used to choose what to cluster and how to evaluate

the results. The first three options are the same as for classification: Use training set, Supplied test set and Percentage split (Section 5.3.1)—except that now the data is assigned to clusters instead of trying to predict a specific class. The fourth mode, Classes to clusters evaluation, compares how well the chosen clusters match up with a pre-assigned class in the data. The drop-down box below this option selects the class, just as in the Classify panel.

An additional option in the Cluster mode box, the Store clusters for visualization tick box, determines whether or not it will be possible to visualize the clusters once training is complete. When dealing with datasets that are so large that memory becomes a problem it may be helpful to disable this option.

Ignoring Attributes

Often, some attributes in the data should be ignored when clustering. The Ignore attributes button brings up a small window that allows you to select which attributes are ignored. Clicking on an attribute in the window highlights it, holding down the SHIFT key selects a range

of consecutive attributes, and holding down CTRL toggles individual attributes on and off. To cancel the selection, back out with the Cancel button. To activate it, click the Select button. The next time clustering is invoked, the selected attributes are ignored.

Working with Filters

The Filtered Clusterer meta-clusterer offers the user the possibility to apply filters directly before the clusterer is learned. This approach eliminates the manual application of a filter in the Preprocess panel, since the data gets processed on the fly. Useful if one needs to try out different filter setups.

Learning Clusters

The Cluster section, like the Classify section, has Start/Stop buttons, a result text area and a result list. These all behave just like their classification counterparts. Right-clicking an entry in the result list brings up a similar menu, except that it shows only two visualization options: Visualize cluster assignments and Visualize tree. The latter is grayed out when it is not applicable.

A.Load each dataset into Weka and run simple k-means clustering algorithm with different values of k(number of desired clusters). Study the clusters formed. Observe the sum of squared errors and centroids, and derive insights.

Ans:

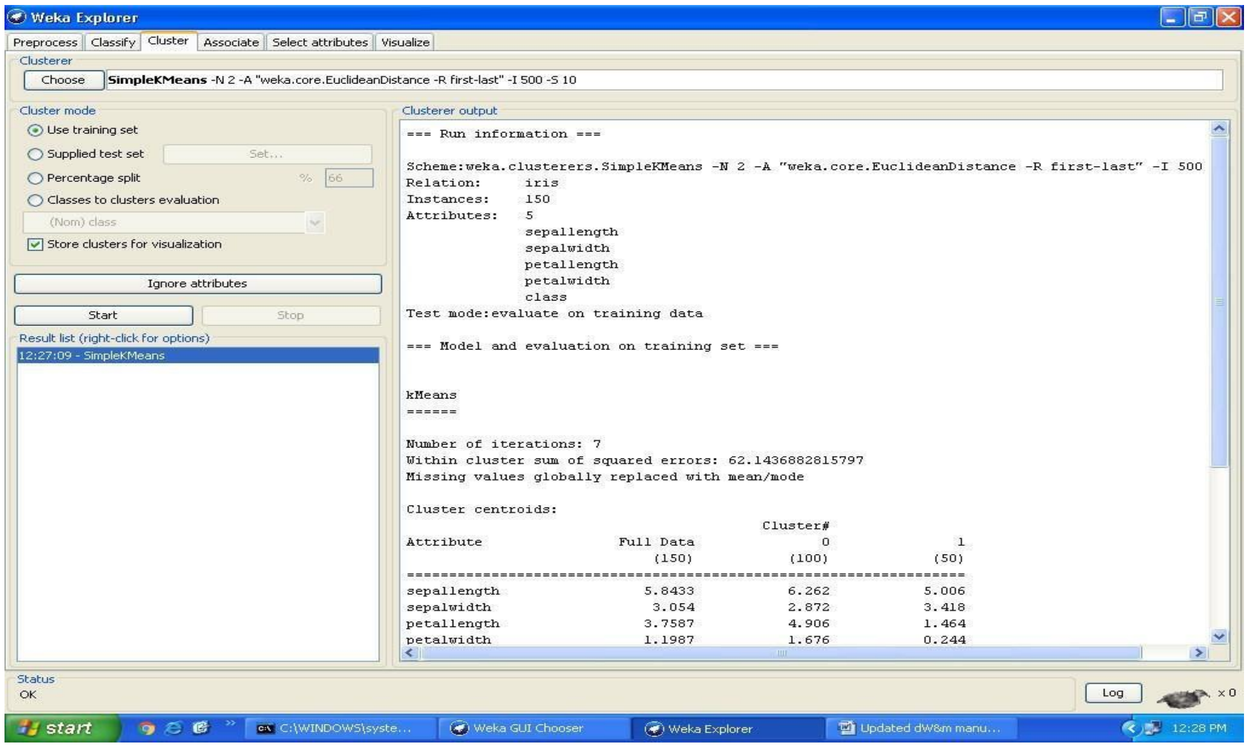
- Steps for run K-mean Clustering algorithms in WEKA

1. Open WEKA Tool.
2. Click on WEKA Explorer.
3. Click on Preprocessing tab button.
4. Click on open file button.
5. Choose WEKA folder in C drive.
6. Select and Click on data option button.
7. Choose iris data set and open file.
8. Click on cluster tab and Choose k-mean and select use training set test option.
9. Click on start button.

Output:

```
=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500
-S 10
Relation:     iris
```

```
Instances:    150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode:evaluate on training data

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

kMeans
======
Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797
Missing values globally replaced with mean/mode

Cluster centroids:
Cluster#
Attribute              Full Data           0            1
(150)          (100)           (50)
==================================================================
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 seconds

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

Clustered Instances

0      100 ( 67%)
1       50 ( 33%)
```



Clusterer: SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Cluster mode: [✓ Use training set] Supplied test set Percentage split (% 66) Classes to clusters evaluation ((Nom) class) [✓ Store clusters for visualization]

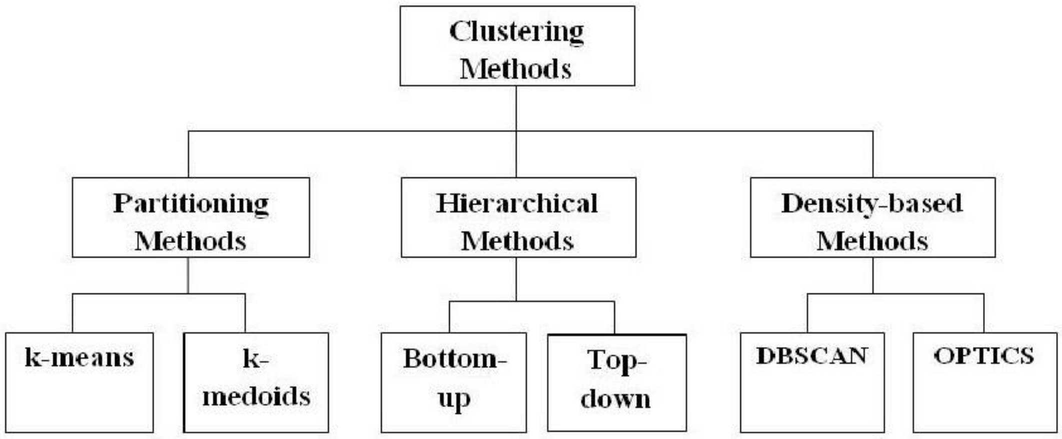
Result list: 12:27:09 - SimpleKMeans

Status: OK

```
=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:evaluate on training data

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


kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797
Missing values globally replaced with mean/mode

Cluster centroids:
                                          Cluster#
Attribute                 Full Data              0                 1
                              (150)          (100)              (50)
==================================================================
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
```

B.Explore other clustering techniques available in Weka.

Ans: Clustering Algorithms And Techniques in WEKA, They are



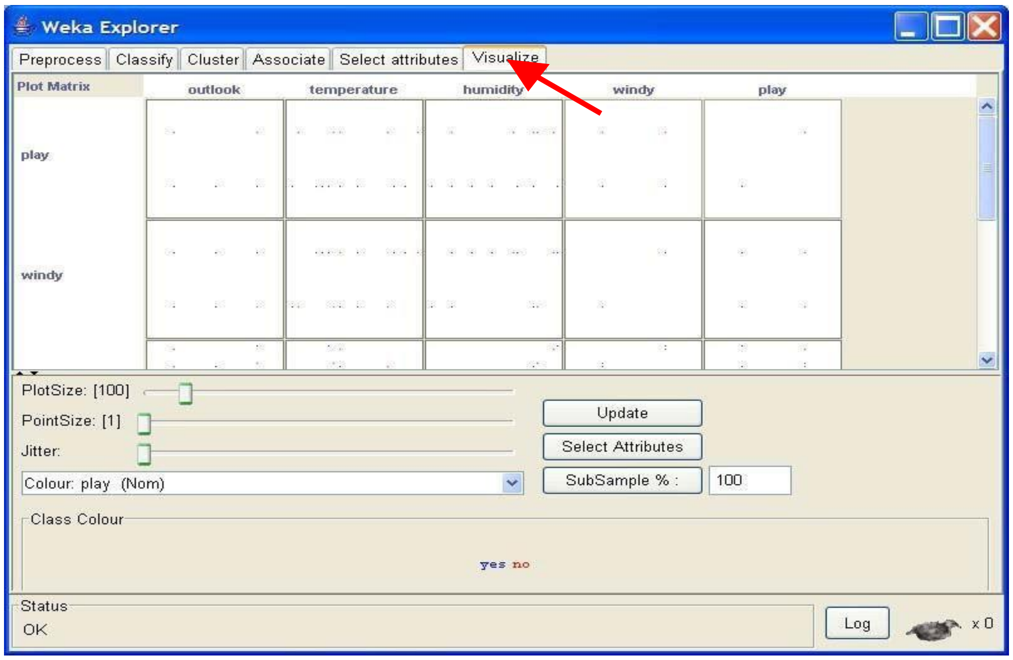
C.Explore visualization features of weka to visualize the clusters. Derive interesting insights and explain.

Ans: <u>Visualize Features</u>

WEKA's visualization allows you to visualize a 2-D plot of the current working relation. Visualization is very useful in practice, it helps to determine difficulty of the learning problem. WEKA can visualize single attributes (1-d) and pairs of attributes (2-d), rotate 3-d visualizations (Xgobi-style). WEKA has "Jitter" option to deal with nominal attributes and to detect "hidden" data points.

Access To **Visualization** From The *Classifier*, *Cluster* And *Attribute Selection* Panel Is Available From A Popup Menu. Click The Right Mouse Button Over An Entry In The Result List To Bring Up The Menu. You Will Be Presented With Options For Viewing Or Saving The Text Output And --- Depending On The Scheme --- Further Options For Visualizing Errors, Clusters, Trees Etc.

To open Visualization screen, click 'Visualize' tab.



PlotSize: [100]
PointSize: [1]
Jitter:
Colour: play (Nom)
SubSample % : 100
Status
OK

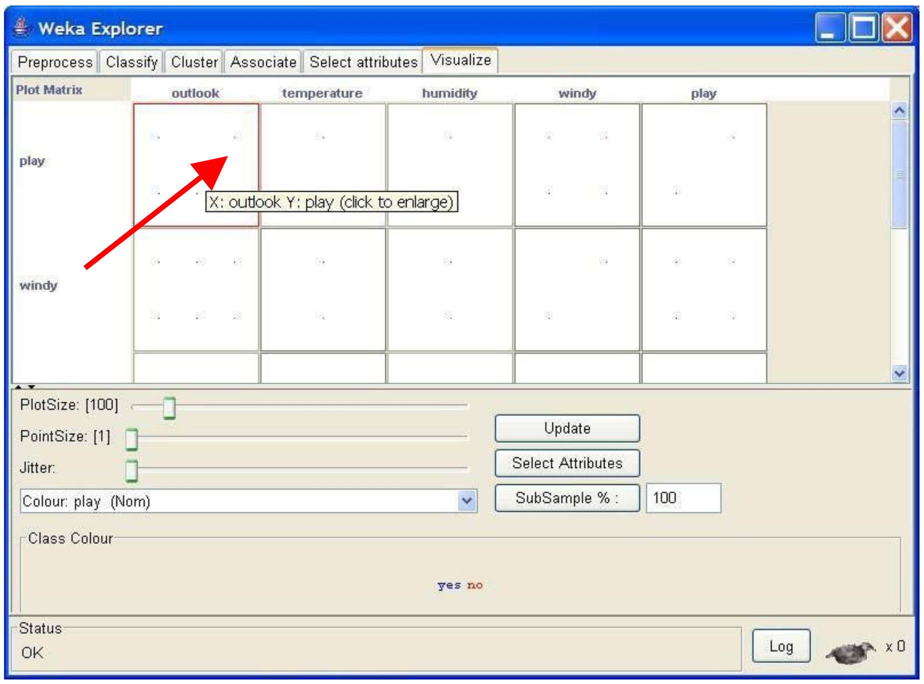
Select a square that corresponds to the attributes you would like to visualize. For example, let's choose 'outlook' for X – axis and 'play' for Y – axis. Click anywhere inside the square that corresponds to 'play o

Changing the View:

In the visualization window, beneath the X-axis selector there is a drop-down list,

'Colour', for choosing the color scheme. This allows you to choose the color of points based on the attribute selected. Below the plot area, there is a legend that describes what values the colors correspond to. In your example, red represents 'no', while blue represents 'yes'. For better visibility you should change the color of label 'yes'. Left-click on 'yes' in the 'Class colour' box and select lighter color from the color palette.

n the left and 'outlook' at the top.



PlotSize: [100]
PointSize: [1]
Jitter:
Colour: play (Nom)
SubSample % : 100
Class Colour
yes no
Status
OK

Experiment-6:

6.Write a java program to prepare a simulated data set with unique instances

Creating new dataset java

```
Dataset dt = new DefaultDataset (); // creation syntax for the dataset
for (b=0, b<8, b++) // condition setting
{
Instnc inst_1 = Instnc.randomInstnc(12); // defining the instance for the dataset
Dt.add(inst_1); //adding the instance for the dataset
}
```

This program is used for creating and iterating the entire dataset representing the car name and car characteristic when getting a sql query to be performed over it.

```
public class Cars_dtset {
public String car_name;
public String car_description;
public int car_no;
}
interface Actual_Query extends Bs_Query {
@Select("select car_name, car_description, car_no from Cars_dtset")
DataSet<Cars_dtset> getAllCars_dtset();
}
Actual_Query mq_0 = con.createQueryObject(Actual_Query.class);
DataSet rows = mq_0.getAllCars_dtset();
for (Cars_dtset mq_0: rows) {
System.out.println("CarName = " + mq_0.car_name);
System.out.println("CarDescription = " + mq_0.car_description);
}
```

Experiment-7:

7.Write a python program to generate frequent item sets/association rules using apriori algorithm.

Step 1: Data preprocessing

- **Installing the required package**

```
!pip install apyori
```

- **Importing the libraries**

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

- **Importing the dataset**

```
Data = pd.read_csv('/content/drive/MyDrive/Market_Basket_Optimisation.csv', header =
None)
```

- **Transforming our pandas dataset into a list dataset**

```
# Intializing the list
transacts = []
# populating a list of transactions
for i in range(0, 7501):
  transacts.append([str(Data.values[i,j]) for j in range(0, 20)])
```

Step 2: Training apriori model

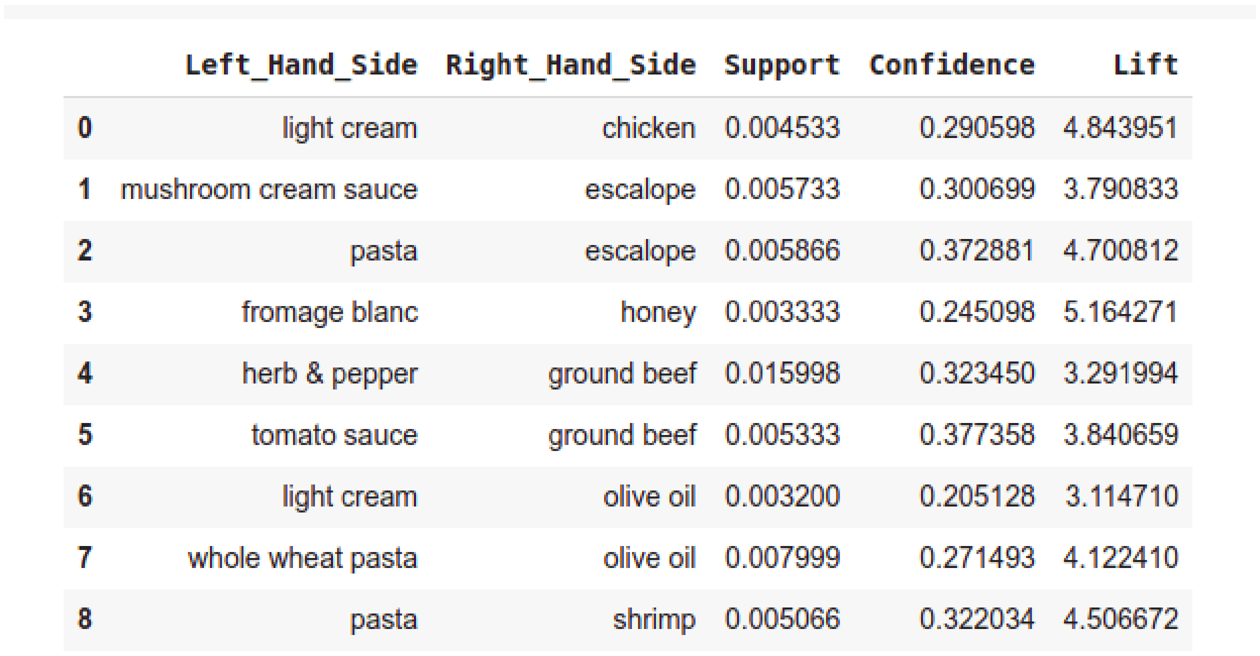
```
from apyori import apriori
rule = apriori(transactions = transacts, min_support = 0.003, min_confidence = 0.2,
min_lift = 3, min_length = 2, max_length = 2)
```

Step 3: Visualising the results

```
output = list(rule) # returns a non-tabular output
# putting output into a pandas dataframe
def inspect(output):
    lhs         = [tuple(result[2][0][0])[0] for result in output]
    rhs         = [tuple(result[2][0][1])[0] for result in output]
    support    = [result[1] for result in output]
    confidence = [result[2][0][2] for result in output]
    lift       = [result[2][0][3] for result in output]
    return list(zip(lhs, rhs, support, confidence, lift))
output_DataFrame = pd.DataFrame(inspect(results), columns = ['Left_Hand_Side',
'Right_Hand_Side', 'Support', 'Confidence', 'Lift'])
```

```
output_DataFrame
```

Output:



	Left_Hand_Side	Right_Hand_Side	Support	Confidence	Lift
0	light cream	chicken	0.004533	0.290598	4.843951
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
2	pasta	escalope	0.005866	0.372881	4.700812
3	fromage blanc	honey	0.003333	0.245098	5.164271
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
6	light cream	olive oil	0.003200	0.205128	3.114710
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
8	pasta	shrimp	0.005066	0.322034	4.506672

Experiment-8:
8. Write a program to calculate chi-square value using python. Report your observation.

We need to compare the obtained p-value with alpha value of 0.05.

```
from scipy.stats import chi2_contingency



# defining the table

data = [[207, 282, 241], [234, 242, 232]]

stat, p, dof, expected = chi2_contingency(data)



# interpret p-value

alpha = 0.05

print("p value is " + str(p))

if p <= alpha:

    print('Dependent (reject H0)')

else:

    print('Independent (H0 holds true)')
```

Output :

```
p value is 0.1031971404730939
Independent (H0 holds true)
```

Experiment-9:

9. Write a program of Naïve Bayesian classification using python programming language

Code:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred)
```

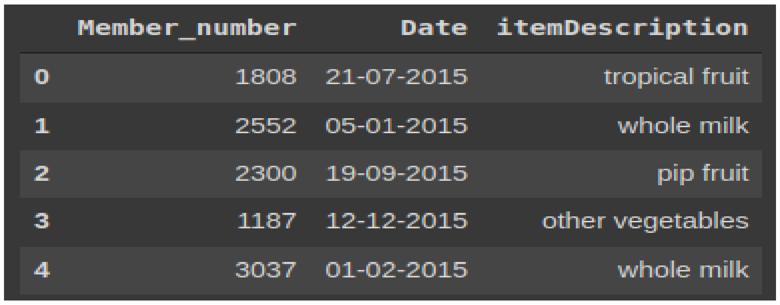
Experiment-10:

10. Implement a java program to perform Apriori algorithm.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder
from mpl_toolkits.mplot3d import Axes3D
import networkx as nx

basket = pd.read_csv("Groceries_dataset.csv")
display(basket.head())
```

out put:

:

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk

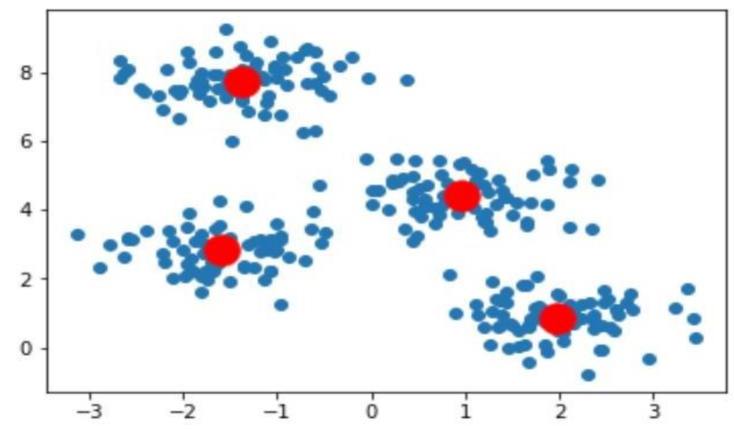
Experiment-11:

11. Write a program of cluster analysis using simple k-means algorithm Python Programming language.

Code:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60,
random_state=0)plt.scatter(X[:,0], X[:,1])
wcss = []for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0)
pred_y = kmeans.fit_predict(X)plt.scatter(X[:,0], X[:,1])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.show()
```

OUTPUT:



Experiment-12:

12. Write a program to compute/display dissimilarity matrix using python.

a) Demonstrate the following Similarity and Dissimilarity Measures using python FOR Cosine Similarity.

we calculate the **Cosine Similarity** between the two non-zero vectors. A vector is a single dimesingle-dimensional signal **NumPy array**. Cosine similarity is a measure of similarity, often used to measure document similarity in text analysis. We use the below formula to compute the cosine similarity.

Similarity = (A.B) / (||A||.||B||)

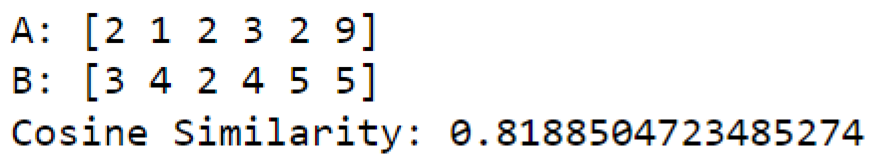
where A and B are vectors:

- A.B is dot product of A and B: It is computed as sum of element-wise product of A and B.
- ||A|| is L2 norm of A: It is computed as square root of the sum of squares of elements of the vector A.

Example 1:

```
# import required libraries
import numpy as np
from numpy.linalg import norm

# define two lists or array
A = np.array([2,1,2,3,2,9])
B = np.array([3,4,2,4,5,5])

print("A:", A)
print("B:", B)

# compute cosine similarity
cosine = np.dot(A,B)/(norm(A)*norm(B))
print("Cosine Similarity:", cosine)
```

Output:



```
A: [2 1 2 3 2 9]
B: [3 4 2 4 5 5]
Cosine Similarity: 0.8188504723485274
```

b) Demonstrate the following Similarity and Dissimilarity Measures using python FOR Jaccard Similarity

The Jaccard similarity (also known as Jaccard similarity coefficient, or Jaccard index) is a statistic used to measure similarities between two sets.

Its use is further extended to measure similarities between two objects, for example two text files. In Python programming, Jaccard similarity is mainly used to measure similarities between two sets or between two asymmetric binary vectors.

Mathematically, the calculation of Jaccard similarity is simply taking the ratio of set intersection over set union.

Consider two sets **A** and **B**:

Then their Jaccard similarity (or Jaccard index) is given by:

$$J=|A\cap B||AUB|=|A\cap B||A|+|B|-|AUB|$$

Let’s break down this formula into two components:

1. Nominator

The nominator is effectively the set intersection between **A** and **B**, shown by the yellow area in the infographic below:

2. Denominator

The denominator is effectively the set union of **A** and **B**, shown by the yellow area in the infographic below:

Using the formula of Jaccard similarity, we can see that the similarity statistic is simply the ratio of the above two visualizations, where:

- If both sets are identical, for example A=1,2,3 and B=1,2,3, then their Jaccard similarity = 1.
- If sets A and B don't have common elements, for example, say A=1,2,3 and B=4,5,6, then their Jaccard similarity = 0.
- If sets sets A and B have some common elements, for example, A=1,2,3 and B=3,4,5, then their Jaccard similarity is some value on the interval: $0 \leq J(A,B) \leq 1$.

Calculate Jaccard similarity
Consider two sets:

- **A** = {1, 2, 3, 5, 7}
- **B** = {1, 2, 4, 8, 9}

Or visually:

Step 1:
As the first step, we will need to find the set intersection between **A** and **B**:

In this case:

$$A \cap B = \{1,2\}$$

Step 2:
The second step is to find the set union of **A** and **B**:

In this case:

$$A \cup B = \{1,2,3,5,7,4,8,9\}$$

Step 3:

And the final step is to take the ratio of sizes of intersection and union:

$$J=|A∩B||AUB|=28=0.25$$

What is Jaccard distance

Unlike the Jaccard similarity (Jaccard index), the Jaccard distance is a measure of dissimilarity between two sets.

Mathematically, the calculation of Jaccard distance is the ratio of difference between set union and set intersection over set union.

Consider two sets **A** and **B**:

Then their Jaccard distance is given by:

$$dJ=|AUB|–|A∩B||AUB|=1–J(A,B)$$

Let's break down this formula into two components:

---

1. Nominator

The nominator can be also written as:

$$|AUB|–|A∩B|=(A\setminus B)U(B\setminus A)=AoB$$

which is effectively the set symmetric difference between **A** and **B**, shown by the yellow area in the infographic below:

2. Denominator

The denominator is effectively the set union of **A** and **B**, shown by the yellow area in the infographic below:

Using the formula of Jaccard distance, we can see that the dissimilarity statistic is simply the ratio of the above two visualizations, where:

- If both sets are identical, for example A=1,2,3 and B=1,2,3, then their Jaccard distance = 0.
- If sets A and B don't have common elements, for example, say A=1,2,3 and B=4,5,6, then their Jaccard distance = 1.
- If sets sets A and B have some common elements, for example, A=1,2,3 and B=3,4,5, then their Jaccard distance is some value on the interval: $0≤dJ(A,B)≤1$.

Calculate Jaccard distance
Consider two sets:

- **A** = {1, 2, 3, 5, 7}
- **B** = {1, 2, 4, 8, 9}

Or visually:

Step 1:
As the first step, we will need to find the set symmetric difference between **A** and **B**:

In this case:

$$|AUB|-|A\cap B|=(A\setminus B)U(B\setminus A)=AoB=\{3,7,5,4,8,9\}$$

Step 2:
The second step is to find the set union of **A** and **B**:

In this case:

$$AUB=\{1,2,3,5,7,4,8,9\}$$

Step 3:
And the final step is to take the ratio of sizes of symmetric difference and union:

$$dJ=|AUB|-|A\cap B||AUB|=68=0.75$$

Similarity and distance of asymmetric binary attributes
In this section we will look into a more specific application of Jaccard similarity and Jaccard distance. More specifically, their application to asymmetric binary attributes.

From the naming of it, we can already guess what a binary attribute is. It's an attribute that has only two states, and those two states are:

- 0, meaning an attribute is not present
- 1, meaning an attribute is present

The asymmetry comes from the point that if both attributes are present (both equal to 1), it is considered more important, than if both attributes weren't present (both equal to 0).

Suppose we have two vectors, **A** and **B**, each with n binary attributes.
In this case, the Jaccard similarity (index) can be calculated as:

$$J=M11M01+M10+M11$$

and Jaccard distance can be calculated as:

$$dJ=M01+M10M01+M10+M11=1-J$$

where:

- M11 is the total numbers of attributes, for which both **A** and **B** have 1
- M01 is the total numbers of attributes, for which **A** has 0 and **B** has 1
- M10 is the total numbers of attributes, for which **A** has 1 and **B** has 0
- M00 is the total numbers of attributes, for which both **A** and **B** have 0

and:

$$M11+M01+M10+M00=n$$

Example
To explain this in more simple terms, consider the example that can be used for market basket analysis.
You operate a store that has 6 products (attributes) and 2 customers (objects), and also keep track of which customer bought which item. You know that:

- Customer A bought: apple, milk coffee
- Customer B bought: eggs, milk, coffee

As you can already imagine, we can construct the following matrix:

	Apple	Tomato	Eggs	Milk	Coffee	Sugar
A	1	0	0	1	1	1
B	0	0	1	1	1	0

Where the binary attribute for each customer is indicating if customer purchased (1) or didn't purchase (0) a particular product.

The question is to find the Jaccard similarity and Jaccard distance for these two customers.

Step 1:
We will first need to find the total number for attributes for each M:

	Count	Explanation
M11	2	Both customers bought coffee and milk
M01	1	Customer A didn't buy eggs, whereas Customer B bought eggs
M10	2	Customer B didn't buy apple and sugar, whereas Customer 1 bought apple and sugar
M00	1	Neither of customers bought tomato

We can validate the groups by summing up the counts. it should be equal to 6 which is the n number of attributes (products):

M11+M01+M10+M00=2+1+2+1=6

Step 2:
Since we have all the required inputs, we can now calculate the Jaccard similarity:

J=M11M01+M10+M11=21+2+2=25=0.4

And Jaccard distance:

dJ=M01+M10M01+M10+M11=1+21+2+2=35=0.6

Calculate Jaccard similarity in Python
In this section we will use the same sets as we defined in the one of the first sections:

- **A** = {1, 2, 3, 5, 7}
- **B** = {1, 2, 4, 8, 9}

We begin by defining them in Python:

```
A = {1, 2, 3, 5, 7}
B = {1, 2, 4, 8, 9}
```

Python
Copy
As the next step we will construct a function that takes set **A** and set **B** as parameters and then calculates the Jaccard similarity using set operations and returns it:

```
def jaccard_similarity(A, B):
    #Find intersection of two sets
    nominator = A.intersection(B)

    #Find union of two sets
    denominator = A.union(B)

    #Take the ratio of sizes
    similarity = len(nominator)/len(denominator)

    return similarity
```

Python
Copy
Then test our function:

```
similarity = jaccard_similarity(A, B)

print(similarity)
```

Python
Copy
And you should get:

```
0.25
```

which is exactly the same as the statistic we calculated manually.

Calculate Jaccard distance in Python
In this section we continue working with the same sets (**A** and **B**) as in the previous section:

- **A** = {1, 2, 3, 5, 7}
- **B** = {1, 2, 4, 8, 9}

We begin by defining them in Python:

```
A = {1, 2, 3, 5, 7}
B = {1, 2, 4, 8, 9}
```

Python
Copy
As the next step we will construct a function that takes set **A** and set **B** as parameters and then calculates the Jaccard similarity using set operations and returns it:

```
def jaccard_distance(A, B):
    #Find symmetric difference of two sets
    nominator = A.symmetric_difference(B)

    #Find union of two sets
    denominator = A.union(B)

    #Take the ratio of sizes
    distance = len(nominator)/len(denominator)

    return distance

distance = jaccard_distance(A, B)
```

Python
Copy
Then test our function:

```
distance = jaccard_distance(A, B)

print(distance)
```

Python
Copy
And you should get:

```
0.75
```

which is exactly the same as the statistic we calculated manually

Calculate similarity and distance of asymmetric binary attributes in Python

We begin by importing the required dependencies:

```
import numpy as np
from scipy.spatial.distance import jaccard
from sklearn.metrics import jaccard_score
```

Python

Copy

Using the table we used in the theory section:

	Apple	Tomato	Eggs	Milk	Coffee	Sugar
A	1	0	0	1	1	1
B	0	0	1	1	1	0

we can create the required binary vectors:

```
A = np.array([1,0,0,1,1,1])
B = np.array([0,0,1,1,1,0])
```

Python

Copy

and then use the libraries' function to calculate the Jaccard similarity and Jaccard distance:

```
similarity = jaccard_score(A, B)
distance = jaccard(A, B)

print(f'Jaccard similarity is equal to: {similarity}')
print(f'Jaccard distance is equal to: {distance}')
```

Python

Copy

And you should get:

```
Jaccard similarity is equal to: 0.4
Jaccard distance is equal to: 0.6
```

which is exactly the same as the statistic we calculated manually.

---

c) Demonstrate the following Similarity and Dissimilarity Measures using python for EUCLIDEAN DISTANCE.

```python
# Python code to find Euclidean distance

# using linalg.norm()



# Import NumPy Library

import numpy as np



# initializing points in

# numpy arrays

point1 = np.array((4, 4, 2))

point2 = np.array((1, 2, 1))
```

```
00# calculate Euclidean distance

# using linalg.norm() method

dist = np.linalg.norm(point1 - point2)



# printing Euclidean distance

print(dist)
```

Output

```
3.7416573867739413
```

d) Demonstrate the following Similarity and Dissimilarity Measures using python for Manhattan Distance.

```
from math import sqrt

#create function to calculate Manhattan distance
def manhattan(a, b):
    return sum(abs(val1-val2) for val1, val2 in zip(a,b))

#define vectors
A = [2, 4, 4, 6]
B = [5, 5, 7, 8]

#calculate Manhattan distance between vectors
manhattan(A, B)
```

```
9
```

The Manhattan distance between these two vectors turns out to be **9**.

We can confirm this is correct by quickly calculating the Manhattan distance by hand:

$\Sigma|A_i – B_i|$ = |2-5| + |4-5| + |4-7| + |6-8| = 3 + 1 + 3 + 2 = **9**.

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)
    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__== "_main_":
    main()
```

output:

```
Estimated coefficients:
b_0 = -0.0586206896552
b_1 = 1.45747126437
```

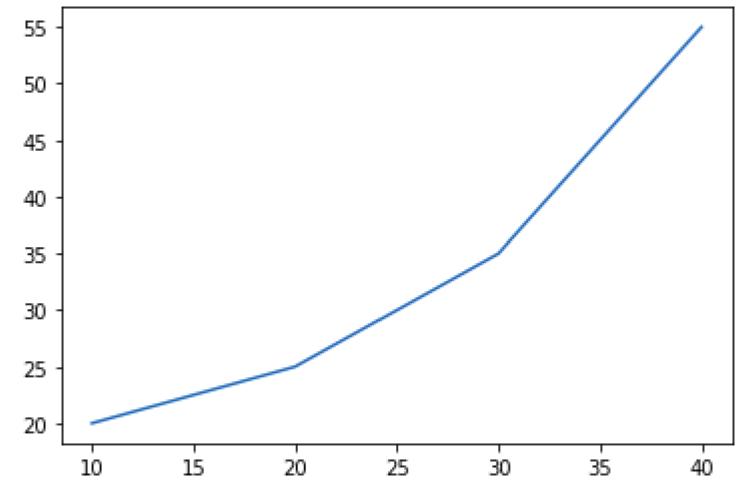
Experiment-13:

13.Visualize the data sets using matplotlib in python.(Histogram, Box Plot, Bar chart, Pie-Chart)

Code for pie chart:

```
import matplotlib.pyplot as plt

 # initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

 # plotting the data

plt.plot(x, y)

 plt.show()
```

Output:



Steps to plot a histogram in Python using Matplotlib

Step 1: Install the Matplotlib package

Install the Matplotlib package using the following command (under Windows):

```
pip install matplotlib
```

Step 2: Collect the data for the histogram

For example, let's say that you have the following data about the age of 100 individuals:

```
Age
1,1,2,3,3,5,7,8,9,10,
10,11,11,13,13,15,16,17,18,18,
18,19,20,21,21,23,24,24,25,25,
25,25,26,26,26,27,27,27,27,27,
29,30,30,31,33,34,34,34,35,36,
36,37,37,38,38,39,40,41,41,42,
43,44,45,45,46,47,48,48,49,50,
51,52,53,54,55,55,56,57,58,60,
61,63,64,65,66,68,70,71,72,74,
75,77,81,83,84,87,89,90,90,91
```

Step 3: Determine the number of bins

set the number of bins to 10. At the end of this guide, I'll show you another way to derive the bins.

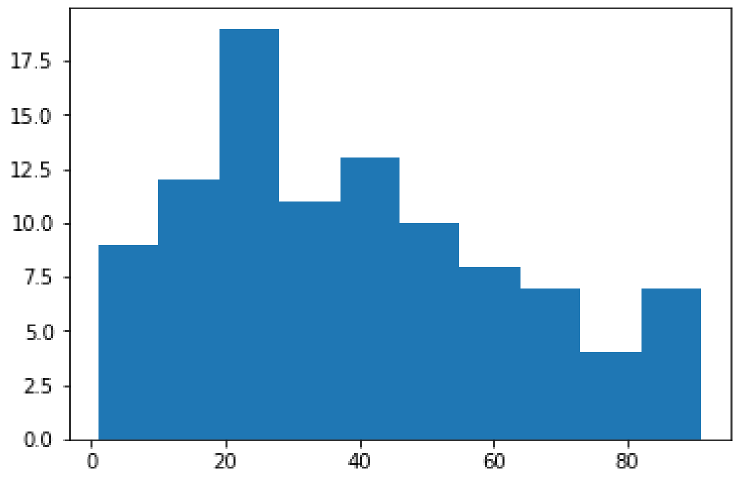
Step 4: Plot the histogram in Python using matplotlib

to plot the histogram based on the template that you saw at the beginning of this guide:

```
import matplotlib.pyplot as plt



x = [value1, value2, value3, ... ]

plt.hist(x, bins = number of bins)

plt.show()
```

And for our example, this is the complete Python code after applying the above template:

```
import matplotlib.pyplot as plt

 x = [1,1,2,3,3,5,7,8,9,10,

     10,11,11,13,13,15,16,17,18,18,

     18,19,20,21,21,23,24,24,25,25,

     25,25,26,26,26,27,27,27,27,27,

     29,30,30,31,33,34,34,34,35,36,

     36,37,37,38,38,39,40,41,41,42,

     43,44,45,45,46,47,48,48,49,50,

     51,52,53,54,55,55,56,57,58,60,

     61,63,64,65,66,68,70,71,72,74,

     75,77,81,83,84,87,89,90,90,91

     ]

plt.hist(x, bins=10)

plt.show()
```


17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20
40
60
80