CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

FACULTY OF TECHNOLOGY & ENGINEERING

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY AND RESEARCH

Subject: CE350 Data Warehousing & Data Mining

[DWDM (CE350)]

Semester: 6th [BTECH CE]

Academic year: Jan 2020 - April 2021

PRACTICAL - 1

AIM: Data Preprocessing using Pandas (Handling Missing Value, Data Wrangling, Dimension Reduction)

THEORY:

Python

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- Why Python?
  - Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.
    - Python is Interpreted
    - Python is Interactive
    - Python is Object-Oriented
    - Python is a Beginner's Language
  - Characteristics of Python
    - It supports functional and structured programming methods as well as OOP.
    - It can be used as a scripting language or can be compiled to byte-code for building large applications.
    - It provides very high-level dynamic data types and supports dynamic type checking.
    - It supports automatic garbage collection.
    - It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
  - Applications of Python
    - Easy-to-learn
    - Easy-to-read
    - Easy-to-maintain
    - A broad standard library
    - Interactive Mode
    - Portable
    - Extendable
    - Databases
    - GUI Programming
    - Scalable

Pandas

Pandas is the most popular python library that is used for data analysis. It provides highly optimized performance with back-end source code is purely written in C or Python.

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from Data Frame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, Data Frame, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPydata structures into Data Frame objects.
- Intelligent label-based slicing, fancy indexing, and sub setting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving/loading data from the ultrafast HDF5 format
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

NumPy

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication

- Why use NumPy?
  - NumPy is memory efficiency, meaning it can handle the vast amount of data more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, Tensor Flow and Scikit learn to use NumPy array to compute the matrix multiplication in the back end.
- Characteristics :
  - NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
  - The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of

(Python, including NumPy) objects, thereby allowing for arrays of different sized elements.

- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to
- NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python- based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

Working with Pandas

Data:

```
In [1]: import pandas as pd
        df=pd.read_csv("weather.csv")
        df
```

Out[1]:

	outlook	temperature	humidity	windy	play	temp in Celcius	wind speed	Date
0	overcast	hot	high	False	yes	43	4	12/9/2012
1	overcast	cool	normal	True	yes	15	5	10/3/2000
2	overcast	mild	high	True	yes	30	3	24/09/2001
3	overcast	hot	normal	False	yes	40	4	21/06/1999
4	rainy	mild	high	False	yes	30	7	28/01/2014
5	rainy	cool	normal	False	yes	14	5	19/04/2012
6	rainy	cool	normal	True	no	28	9	14/9/2012
7	rainy	mild	normal	False	yes	25	8	11/7/2012
8	rainy	mild	high	True	no	25	7	4/4/2012
9	sunny	hot	high	False	no	38	5	12/9/2012
10	sunny	hot	high	True	no	38	3	28/01/2014
11	sunny	mild	high	False	no	33	2	12/9/2012
12	sunny	cool	normal	False	yes	32	9	10/3/2000
13	sunny	mild	normal	True	yes	33	4	28/01/2014

Missing Data:

In [21]:
```
import pandas as pd
df=pd.read_csv("weather.csv")
df.fillna(0,inplace=True)
df
```

Out[21]:

	outlook	temperature	humidity	windy	play	temp	wind speed	Date
0	overcast	hot	high	False	yes	43	4.0	12/9/2012
1	overcast	cool	normal	True	yes	15	5.0	10/3/2000
2	overcast	mild	high	True	yes	30	0.0	24/09/2001
3	overcast	hot	normal	False	yes	40	0.0	21/06/1999
4	rainy	mild	high	False	yes	30	7.0	28/01/2014
5	rainy	cool	normal	False	yes	14	5.0	19/04/2012
6	rainy	cool	normal	True	no	28	9.0	14/9/2012
7	rainy	mild	normal	False	yes	25	8.0	11/7/2012
8	rainy	mild	high	True	no	25	0.0	4/4/2012
9	sunny	hot	high	False	no	38	5.0	12/9/2012
10	sunny	hot	high	True	no	38	3.0	28/01/2014

Operations:

In [9]:
```
df.head(7)
```

Out[9]:

	outlook	temperature	humidity	windy	play	temp in Celcius	wind speed	Date
0	overcast	hot	high	False	yes	43	4	12/9/2012
1	overcast	cool	normal	True	yes	15	5	10/3/2000
2	overcast	mild	high	True	yes	30	3	24/09/2001
3	overcast	hot	normal	False	yes	40	4	21/06/1999
4	rainy	mild	high	False	yes	30	7	28/01/2014
5	rainy	cool	normal	False	yes	14	5	19/04/2012
6	rainy	cool	normal	True	no	28	9	14/9/2012

In [10]:
```
df['temp in Celcius'].std()
```

Out[10]:
```
8.561349385272353
```

In [11]:
```
df['temp in Celcius'].mean()
```

Out[11]:
```
30.285714285714285
```

In [12]:
```
df['wind speed'].mean()
```

Out[12]:
```
5.357142857142857
```

In [13]:
```
df['wind speed'].mean()
```

Out[13]:
```
5.357142857142857
```

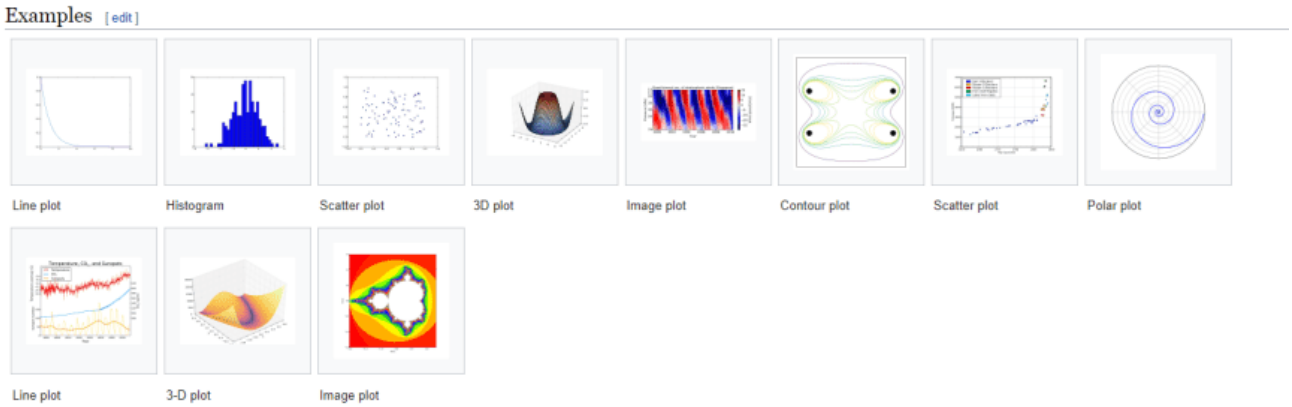
PRACTICAL - 2

Aim: Vehicle data analysis using matplotlib and seaborn

Theory:

Matplotlib: It is a plotting library for the Python programming language and its numerical mathematics extension NumPy. Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.



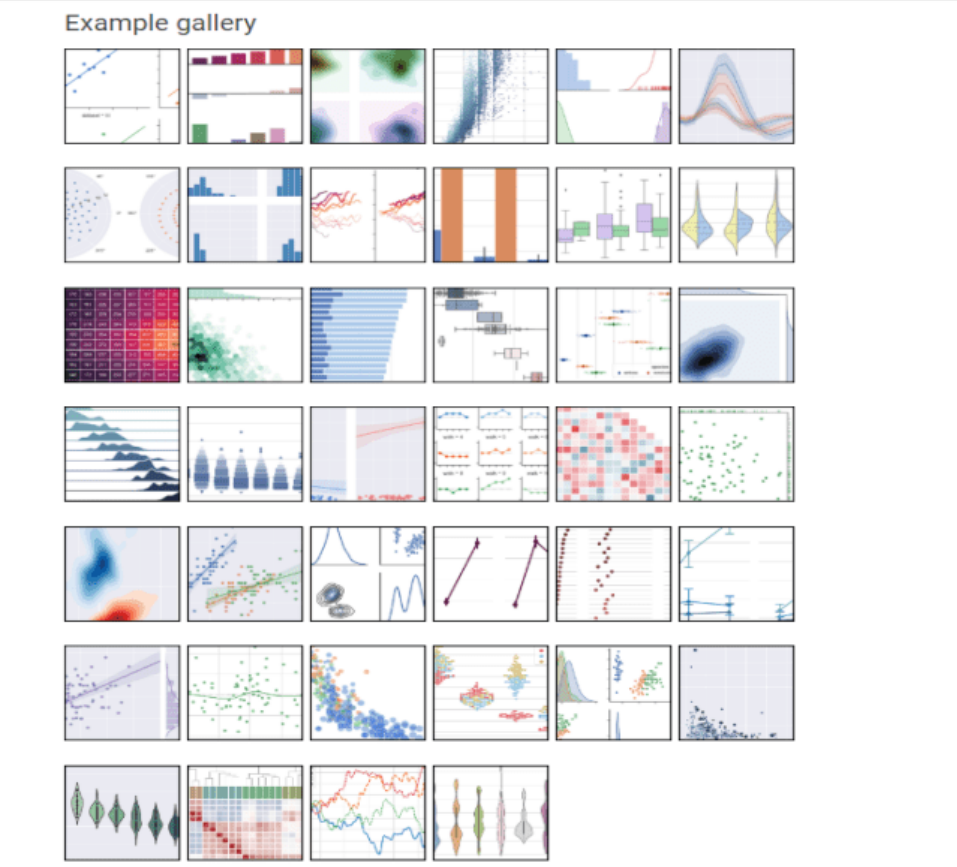
Seaborn:

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

Here is some of the functionality that seaborn offers:

- A dataset-oriented API for examining relationships between multiple variables
- Specialized support for using categorical variables to show observations or aggregate statistics
- Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data
- Automatic estimation and plotting of linear regression models for different kinds dependent variables
- Convenient views onto the overall structure of complex datasets
- High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations
- Concise control over matplotlib figure styling with several built-in themes
- Tools for choosing color palettes that faithfully reveal patterns in your data

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.



Working:

Code:

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

Output:

Out[54]:

	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.60	...	130	mpfi	3.47	2.68	9.00	111	5000	21	27	13495
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	110	5500	19	25	15250

5 rows × 26 columns

Code:

```
headers = ["symboling", "normalized-losses", "make",
           "fuel-type", "aspiration","num-of-doors",
           "body-style","drive-wheels", "engine-location",
           "wheel-base","length", "width","height", "curb-weight",
           "engine-type","num-of-cylinders", "engine-size",
           "fuel-system","bore","stroke", "compression-ratio",
           "horsepower", "peak-rpm","city-mpg","highway-mpg","price"]
df.columns=headers
df.head()
```

Output:

Out[28]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepowe
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	11
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	15
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	10
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	11
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	11

5 rows × 26 columns

Code:

```
data = df
data.isna().any()
data.isnull().any()
```

Output:

Out[56]:

```
3              False
?              False
alfa-romero    False
gas            False
std            False
two            False
convertible    False
rwd            False
front          False
88.60          False
168.80         False
64.10          False
48.80          False
2548           False
dohc           False
four           False
130            False
mpfi           False
3.47           False
2.68           False
9.00           False
111            False
5000           False
21             False
27             False
13495          False
dtype: bool
```

Code:

```
data['city-mpg'] = 235 / df['city-mpg']
        data.rename(columns = {'city_mpg': "city-L / 100km"}, inplace = True)
        print(data.columns)
        data.dtypes
```

Output:

```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price'],
      dtype='object')
```

Out[30]:

```
symboling              int64
normalized-losses     object
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg             float64
highway-mpg            int64
price                 object
dtype: object
```

Code:

```
        data.price.unique()
        data = data[data.price != '?']
        data.dtypes
```

Output:

Out[70]:

```
array([16500, 13950, 17450, 15250, 17710, 18920, 23875, 16430, 16925,
       20970, 21105, 24565, 30760, 41315, 36880,  5151,  6295,  6575,
        5572,  6377,  7957,  6229,  6692,  7609,  8558,  8921, 12964,
        6479,  6855,  5399,  6529,  7129,  7295,  7895,  9095,  8845,
       10295, 12945, 10345,  6785, 11048, 32250, 35550, 36000,  5195,
        6095,  6795,  6695,  7395, 10945, 11845, 13645, 15645,  8495,
       10595, 10245, 10795, 11245, 18280, 18344, 25552, 28248, 28176,
       31600, 34184, 35056, 40960, 45400, 16503,  5389,  6189,  6669,
        7689,  9959,  8499, 12629, 14869, 14489,  6989,  8189,  9279,
        5499,  7099,  6649,  6849,  7349,  7299,  7799,  7499,  7999,
        8249,  8949,  9549, 13499, 14399, 17199, 19699, 18399, 11900,
       13200, 12440, 13860, 15580, 16900, 16695, 17075, 16630, 17950,
       18150, 12764, 22018, 32528, 34028, 37028,  9295,  9895, 11850,
       12170, 15040, 15510, 18620,  5118,  7053,  7603,  7126,  7775,
        9960,  9233, 11259,  7463, 10198,  8013, 11694,  5348,  6338,
        6488,  6918,  7898,  8778,  6938,  7198,  7788,  7738,  8358,
        9258,  8058,  8238,  9298,  9538,  8449,  9639,  9989, 11199,
       11549, 17669,  8948, 10698,  9988, 10898, 11248, 16558, 15998,
       15690, 15750,  7975,  7995,  8195,  9495,  9995, 11595,  9980,
       13295, 13845, 12290, 12940, 13415, 15985, 16515, 18420, 18950,
       16845, 19045, 21485, 22470, 22625], dtype=int64)
```

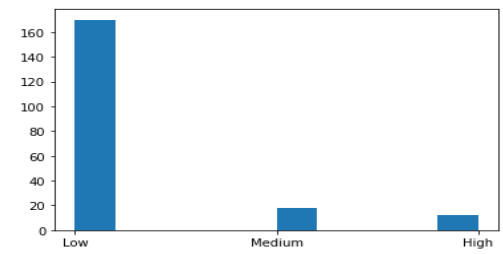
```
Out[71]: symboling              int64
         normalized-losses     object
         make                  object
         fuel-type             object
         aspiration            object
         num-of-doors          object
         body-style            object
         drive-wheels          object
         engine-location       object
         wheel-base           float64
         length               float64
         width                float64
         height               float64
         curb-weight            int64
         engine-type           object
         num-of-cylinders      object
         engine-size            int64
         fuel-system           object
         bore                  object
         stroke                object
         compression-ratio    float64
         horsepower            object
         peak-rpm              object
         city-mpg             float64
         highway-mpg            int64
         price                  int32
         price-binned        category
         dtype: object
```

Code:

```
data['length'] = data['length']/data['length'].max()
data['width'] = data['width']/data['width'].max()
data['height'] = data['height']/data['height'].max()
 bins = np.linspace(min(data['price']), max(data['price']), 4)
group_names = ['Low', 'Medium', 'High']
data['price-binned'] = pd.cut(data['price'], bins,
                              labels = group_names,
                              include_lowest = True)
print(data['price-binned'])
plt.hist(data['price-binned'])
plt.show()
```

Output:

```
0          Low
1          Low
2          Low
3          Low
4          Low
         ...
199        Low
200     Medium
201     Medium
202     Medium
203     Medium
Name: price-binned, Length: 200, dtype: category
Categories (3, object): [Low < Medium < High]
```



Code:

```
pd.get_dummies(data['fuel-type']).head()
data.describe()
```

Output:

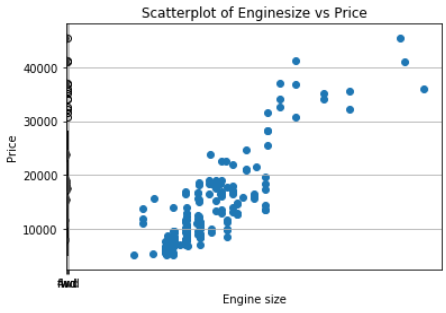
Out[46]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg	price
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000
mean	0.830000	98.848000	0.837232	0.915250	0.899523	2555.705000	126.860000	10.170100	9.937914	30.705000	13205.690000
std	1.248557	6.038261	0.059333	0.029207	0.040610	518.594552	41.650501	4.014163	2.539415	6.827227	7966.982558
min	-2.000000	86.600000	0.678039	0.837500	0.799331	1488.000000	61.000000	7.000000	4.795918	16.000000	5118.000000
25%	0.000000	94.500000	0.800937	0.891319	0.869565	2163.000000	97.750000	8.575000	7.833333	25.000000	7775.000000
50%	1.000000	97.000000	0.832292	0.909722	0.904682	2414.000000	119.500000	9.000000	9.791667	30.000000	10270.000000
75%	2.000000	102.400000	0.881788	0.926042	0.928512	2928.250000	142.000000	9.400000	12.368421	34.000000	16500.750000
max	3.000000	120.900000	1.000000	1.000000	1.000000	4066.000000	326.000000	23.000000	18.076923	54.000000	45400.000000

Code:

```
plt.boxplot(data['price'])
sns.boxplot(x ='drive-wheels', y ='price', data = data)
plt.scatter(data['engine-size'], data['price'])
plt.title('Scatterplot of Enginesize vs Price')
plt.xlabel('Engine size')
plt.ylabel('Price')
plt.grid()
plt.show()
```

Output:



Code:

```
test = data[['drive-wheels', 'body-style', 'price']]
data_grp = test.groupby(['drive-wheels', 'body-style'], as_index = False).mean()
data_grp
```

Output:

Out[49]:

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	26563.250000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

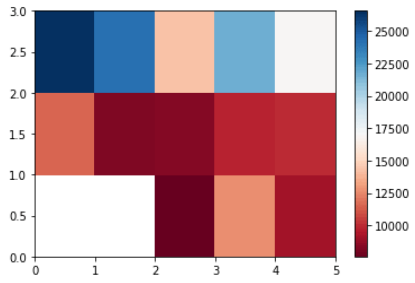
Code:

```
data_pivot = data_grp.pivot(index = 'drive-wheels', columns = 'body-style')
data_pivot
plt.pcolor(data_pivot, cmap ='RdBu')
plt.colorbar()
plt.show()
```

Output:

Out[50]:

	price				
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	NaN	NaN	7603.000000	12647.333333	9095.750000
fwd	11595.00	8249.000000	8396.387755	9811.800000	9997.333333
rwd	26563.25	24202.714286	14337.777778	21711.833333	16994.222222

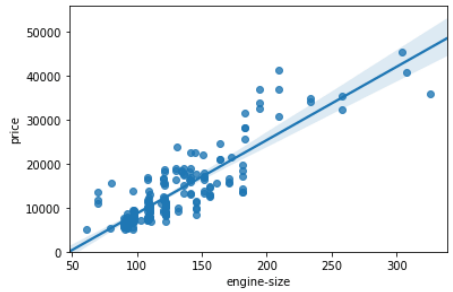


Code:

```
data_annova = data[['make', 'price']]
grouped_annova = data_annova.groupby(['make'])
annova_results_l = sp.stats.f_oneway( grouped_annova.get_group('honda')['price'],
grouped_annova.get_group('subaru')['price'] )
print(annova_results_l)
sns.regplot(x ='engine-size', y ='price', data = data)
plt.ylim(0, )
```

Output:

```
Out[53]: (0, 55927.89182129007)
```



PRACTICAL - 3

Aim: Introduction to R programming: R-GUI, RStudio – Basic working & Commands

Theory:

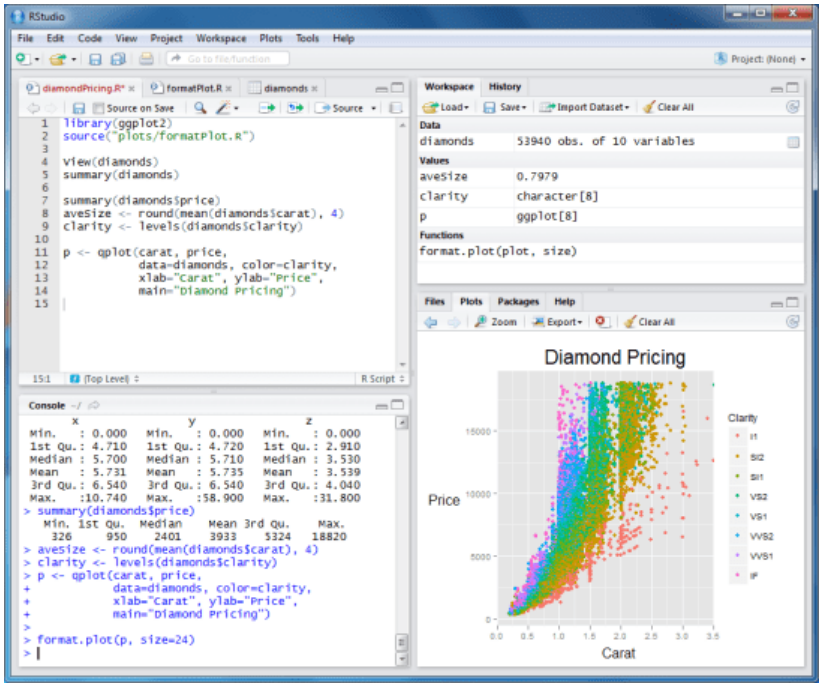
R Programming: R is a programming language and software environment used for statistical analysis, data modeling, and graphical representation and reporting. R is best tool for software programmers, statisticians and data miners who looking forward for to easily manipulate and present data in compelling ways. R is an interpreted language; users typically access it through a command-line interpreter.

R-GUI:

As part of the process of downloading and installing R, you get the standard graphical user interface (GUI), called RGui. RGui gives you some tools to manage your R environment — most important, a console window. The console is where you type instructions, or scripts, and generally get R to do useful things for you.

RStudio :

RStudio is an integrated development environment (IDE) for R, a programming language for statistical computing and graphics. It is available in two formats: RStudio Desktop is a regular desktop application while RStudio Server runs on a remote server and allows accessing RStudio using a web browser.



Difference:

It is important to note the differences between R and RStudio. R is a programming language used for statistical computing while RStudio uses the R language to develop statistical programs. In R, you can write a program and run the code independently of any other computer program. RStudio however, must be used alongside R in order to properly function. Often referred to as an IDE, or integrated development environment, RStudio allows users to develop and edit programs in R by supporting a large number of statistical packages, higher quality graphics, and the ability to manage your workspace.

R and RStudio are not separate versions of the same program, and cannot be substituted for one another. R may be used without RStudio, but RStudio may not be used without R.

R Command Prompt

LOOPS

- **repeat loop :**Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

```
> v <- c("Hello","loop")
> cnt <- 2
>
> repeat {
+    print(v)
+    cnt <- cnt+1
+
+    if(cnt > 5) {
+       break
+    }
+ }
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
>
```

- **While loop: Repeats** a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
> v <- c("Hello","while loop")
> cnt <- 2
>
> while (cnt < 7) {
+    print(v)
+    cnt = cnt + 1
+ }
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
>
```

- **For loop: Like** a while statement, except that it tests the condition at the end of the loop body.

```
> v <- LETTERS[1:4]
> for ( i in v) {
+    print(i)
+ }
[1] "A"
[1] "B"
[1] "C"
[1] "D"
>
>
```

- **Break statement: Terminates** the **loop** statement and transfers execution to the statement immediately following the loop.

```
> v <- c("Hello","loop")
> cnt <- 2
>
> repeat {
+    print(v)
+    cnt <- cnt + 1
+
+    if(cnt > 5) {
+       break
+    }
+ }
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
>
```

- **Next statement: The next** statement simulates the behavior of R switch.

```
> v <- LETTERS[1:6]
> for ( i in v) {
+
+    if (i == "D") {
+       next
+    }
+    print(i)
+ }
[1] "A"
[1] "B"
[1] "C"
[1] "E"
[1] "F"
>
```

FUNCTION

- **Built-in Function:**
  Simple examples of in-built functions are **seq()**, **mean()**, **max()**, **sum(x)** and **paste(...)** etc. They are directly called by user written programs. You can refer most widely used Rfunctions.

```
> # Create a sequence of numbers from 32 to 44.
> print(seq(32,44))
 [1] 32 33 34 35 36 37 38 39 40 41 42 43 44
>
> # Find mean of numbers from 25 to 82.
> print(mean(25:82))
[1] 53.5
>
> # Find sum of numbers frm 41 to 68.
> print(sum(41:68))
[1] 1526
>
```

- **User-defined Function**
  We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
> # Create a function to print squares of numbers in sequence.
> new.function <- function(a) {
+    for(i in 1:a) {
+       b <- i^2
+       print(b)
+    }
+ }
> # Call the function new.function supplying 6 as an argument.
> new.function(6)
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
>
```

STRING

- **Concatenating Strings - paste() function**
  Many strings in R are combined using the **paste()** function. It can take any number of arguments to be combined together.

```
> a <- "Hello"
> b <- 'How'
> c <- "are you? "
>
> print(paste(a,b,c))
[1] "Hello How are you? "
>
> print(paste(a,b,c, sep = "-"))
[1] "Hello-How-are you? "
>
> print(paste(a,b,c, sep = "", collapse = ""))
[1] "HelloHoware you? "
>
```

- **Formatting numbers & strings - format() function**
  Numbers and strings can be formatted to a specific style using **format()** function.

```
> # Total number of digits displayed. Last digit rounded off.
> result <- format(23.123456789, digits = 9)
> print(result)
[1] "23.1234568"
>
> # Display numbers in scientific notation.
> result <- format(c(6, 13.14521), scientific = TRUE)
> print(result)
[1] "6.000000e+00" "1.314521e+01"
>
> # The minimum number of digits to the right of the decimal point.
> result <- format(23.47, nsmall = 5)
> print(result)
[1] "23.47000"
>
> # Format treats everything as a string.
> result <- format(6)
> print(result)
[1] "6"
>
> # Numbers are padded with blank in the beginning for width.
> result <- format(13.7, width = 6)
> print(result)
[1] "  13.7"
>
> # Left justify strings.
> result <- format("Hello", width = 8, justify = "l")
> print(result)
[1] "Hello   "
>
> # Justfy string with center.
> result <- format("Hello", width = 8, justify = "c")
> print(result)
[1] " Hello  "
>
```

- **Counting number of characters in a string - nchar() function**
  This function counts the number of characters including spaces in a string.

```
> result <- nchar("Count the number of characters")
> print(result)
[1] 30
>
```

- **Changing the case - toupper() &tolower() functions**
  These functions change the case of characters of a string.

```
> # Changing to Upper case.
> result <- toupper("Changing To Upper")
> print(result)
[1] "CHANGING TO UPPER"
>
> # Changing to lower case.
> result <- tolower("Changing To Lower")
> print(result)
[1] "changing to lower"
>
```

- **Extracting parts of a string - substring() function**
  This function extracts parts of a String.

```
> # Extract characters from 5th to 7th position.
> result <- substring("Extract", 5, 7)
> print(result)
[1] "act"
>
```

VECTORS

- **Single Element Vector**

  Even when you write just one value in R, it becomes a vector of length 1 and belongs to one of the above vector types.

```
> # Atomic vector of type double.
> print(12.5)
[1] 12.5
>
> # Atomic vector of type integer.
> print(63L)
[1] 63
>
> # Atomic vector of type logical.
> print(TRUE)
[1] TRUE
>
> # Atomic vector of type complex.
> print(2+3i)
[1] 2+3i
>
> # Atomic vector of type raw.
> print(charToRaw('hello'))
[1] 68 65 6c 6c 6f
> # Creating a sequence from 5 to 13.
> v <- 5:13
> print(v)
[1]  5  6  7  8  9 10 11 12 13
>
> # Creating a sequence from 6.6 to 12.6.
> v <- 6.6:12.6
> print(v)
[1]  6.6  7.6  8.6  9.6 10.6 11.6 12.6
>
> # If the final element specified does not belong to the sequence then it is discarded.
> v <- 3.8:11.4
> print(v)
[1]  3.8  4.8  5.8  6.8  7.8  8.8  9.8 10.8
>
```

- **Accessing Vector Elements**

  Elements of a Vector are accessed using indexing. The **[ ] brackets** are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result.**TRUE**, **FALSE** or **0** and **1** can also be used for indexing.

```
> # Accessing vector elements using position.
> t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
> u <- t[c(2,3,6)]
> print(u)
[1] "Mon" "Tue" "Fri"
>
> # Accessing vector elements using logical indexing.
> v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
> print(v)
[1] "Sun" "Fri"
>
> # Accessing vector elements using negative indexing.
> x <- t[c(-2,-5)]
> print(x)
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
>
> # Accessing vector elements using 0/1 indexing.
> y <- t[c(0,0,0,0,0,0,1)]
> print(y)
[1] "Sun"
>
```

- **Vector arithmetic**
  Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
> # Create two vectors.
> v1 <- c(3,8,4,5,0,11)
> v2 <- c(4,11,0,8,1,2)
>
> # Vector addition.
> add.result <- v1+v2
> print(add.result)
[1]  7 19  4 13  1 13
>
> # Vector subtraction.
> sub.result <- v1-v2
> print(sub.result)
[1] -1 -3  4 -3 -1  9
>
> # Vector multiplication.
> multi.result <- v1*v2
> print(multi.result)
[1] 12 88  0 40  0 22
>
> # Vector division.
> divi.result <- v1/v2
> print(divi.result)
[1] 0.7500000 0.7272727       Inf 0.6250000 0.0000000 5.5000000
>
```

- **Vector Element Sorting**
  Elements in a vector can be sorted using the **sort()** function.

```
> # Sort the elements of the vector.
> sort.result <- sort(v)
> print(sort.result)
[1]  -9   0   3   4   5   8  11 304
>
> # Sort the elements in the reverse order.
> revsort.result <- sort(v, decreasing = TRUE)
> print(revsort.result)
[1] 304  11   8   5   4   3   0  -9
>
> # Sorting character vectors.
> v <- c("Red","Blue","yellow","violet")
> sort.result <- sort(v)
> print(sort.result)
[1] "Blue"   "Red"    "violet" "yellow"
>
> # Sorting character vectors in reverse order.
> revsort.result <- sort(v, decreasing = TRUE)
> print(revsort.result)
[1] "yellow" "violet" "Red"    "Blue"
>
```

LIST

- **Creating a List**
  Following is an example to create a list containing strings, numbers, vectors and a logical values.

```
> # Create a list containing strings, numbers, vectors and a logical
> # values.
> list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
> print(list_data)
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE

[[5]]
[1] 51.23

[[6]]
[1] 119.1
```

- **Naming List Elements**
  The list elements can be given names and they can be accessed using these names.

```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
>
> # Show the list.
> print(list_data)
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8

$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

$`A Inner list`[[2]]
[1] 12.3
```

- **Accessing List Elements**
  Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
>
> # Access the first element of the list.
> print(list_data[1])
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

>
> # Access the thrid element. As it is also a list, all its elements will be printed.
> print(list_data[3])
$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

$`A Inner list`[[2]]
[1] 12.3


>
> # Access the list element using the name of the element.
> print(list_data$A_Matrix)
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
>
```

- **Manipulating List Elements**
  We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
>
> # Add element at the end of the list.
> list_data[4] <- "New element"
> print(list_data[4])
[[1]]
[1] "New element"

>
> # Remove the last element.
> list_data[4] <- NULL
>
> # Print the 4th Element.
> print(list_data[4])
$<NA>
NULL

>
> # Update the 3rd Element.
> list_data[3] <- "updated element"
> print(list_data[3])
$`A Inner list`
[1] "updated element"
```

- **Merging Lists**
  You can merge many lists into one list by placing all the lists inside one list() function.

```
> # Create two lists.
> list1 <- list(1,2,3)
> list2 <- list("Sun","Mon","Tue")
>
> # Merge the two lists.
> merged.list <- c(list1,list2)
>
> # Print the merged list.
> print(merged.list)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"

[[5]]
[1] "Mon"

[[6]]
[1] "Tue"
```

- **Converting List to Vector**

  A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the **unlist()** function. It takes the list as input and produces a vector.

```
> # Create lists.
> list1 <- list(1:5)
> print(list1)
[[1]]
[1] 1 2 3 4 5

>
> list2 <-list(10:14)
> print(list2)
[[1]]
[1] 10 11 12 13 14

>
> # Convert the lists to vectors.
> v1 <- unlist(list1)
> v2 <- unlist(list2)
>
> print(v1)
[1] 1 2 3 4 5
> print(v2)
[1] 10 11 12 13 14
>
> # Now add the vectors
> result <- v1+v2
> print(result)
[1] 11 13 15 17 19
```

ARRAY

- **Naming Columns and Rows**
  We can give names to the rows, columns and matrices in the array by using the **dimnames** parameter.

```
> # Create two vectors of different lengths.
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> column.names <- c("COL1","COL2","COL3")
> row.names <- c("ROW1","ROW2","ROW3")
> matrix.names <- c("Matrix1","Matrix2")
>
> # Take these vectors as input to the array.
> result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,column.names,
+    matrix.names))
> print(result)
, , Matrix1

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15

, , Matrix2

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15
```

Check Available R Packages

- Get library locations containing R packages :libPaths()

```
> .libPaths()
[1] "C:/Program Files/R/R-3.5.3/library"
```

- Get the list of all the packages installed :library()

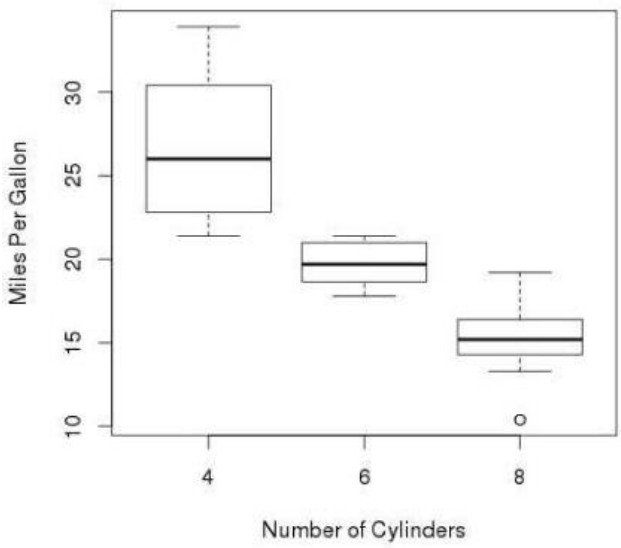
```
Packages in library 'C:/Program Files/R/R-3.5.3/library':

base                    The R Base Package
boot                    Bootstrap Functions (Originally by Angelo Canty for S)
class                   Functions for Classification
cluster                 "Finding Groups in Data": Cluster Analysis Extended
                        Rousseeuw et al.
codetools               Code Analysis Tools for R
compiler                The R Compiler Package
datasets                The R Datasets Package
foreign                 Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS',
                        'Stata', 'Systat', 'Weka', 'dBase', ...
graphics                The R Graphics Package
grDevices               The R Graphics Devices and Support for Colours and
                        Fonts
grid                    The Grid Graphics Package
KernSmooth              Functions for Kernel Smoothing Supporting Wand & Jones
                        (1995)
lattice                 Trellis Graphics for R
MASS                    Support Functions and Datasets for Venables and
                        Ripley's MASS
Matrix                  Sparse and Dense Matrix Classes and Methods
methods                 Formal Methods and Classes
mgcv                    Mixed GAM Computation Vehicle with Automatic Smoothness
                        Estimation
nlme                    Linear and Nonlinear Mixed Effects Models
nnet                    Feed-Forward Neural Networks and Multinomial Log-Linear
                        Models
parallel                Support for Parallel computation in R
rpart                   Recursive Partitioning and Regression Trees
spatial                 Functions for Kriging and Point Pattern Analysis
splines                 Regression Spline Functions and Classes
stats                   The R Stats Package
stats4                  Statistical Functions using S4 Classes
survival                Survival Analysis
tcltk                   Tcl/Tk Interface
tools                   Tools for Package Development
translations            The R Translations Package
utils                   The R Utils Package
```

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc.

R CHARTS & GRAPHS

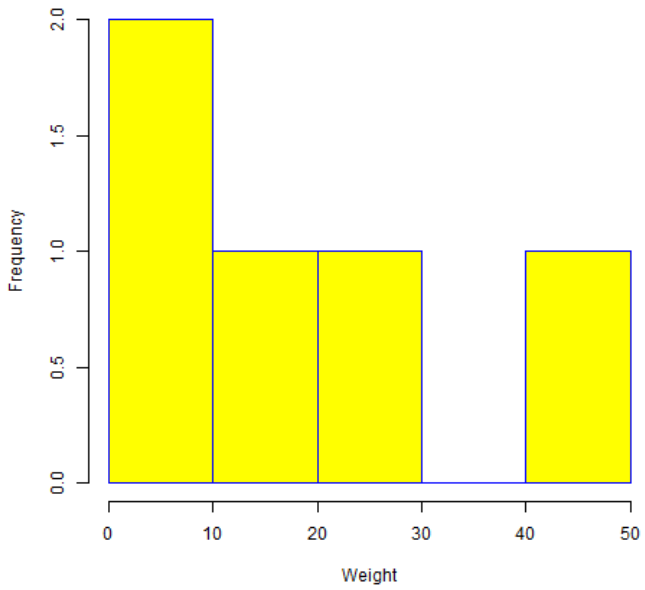
- **Boxplots** are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.
  Boxplots are created in R by using the **boxplot()** function.

```
> input <- mtcars[,c('mpg','cyl')]
> print(head(input))
                   mpg cyl
Mazda RX4         21.0   6
Mazda RX4 Wag     21.0   6
Datsun 710        22.8   4
Hornet 4 Drive    21.4   6
Hornet Sportabout 18.7   8
Valiant           18.1   6
> # Give the chart file a name.
> png(file = "boxplot.png")
>
> # Plot the chart.
> boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
+    ylab = "Miles Per Gallon", main = "Mileage Data")
>
> # Save the file.
> dev.off()
```



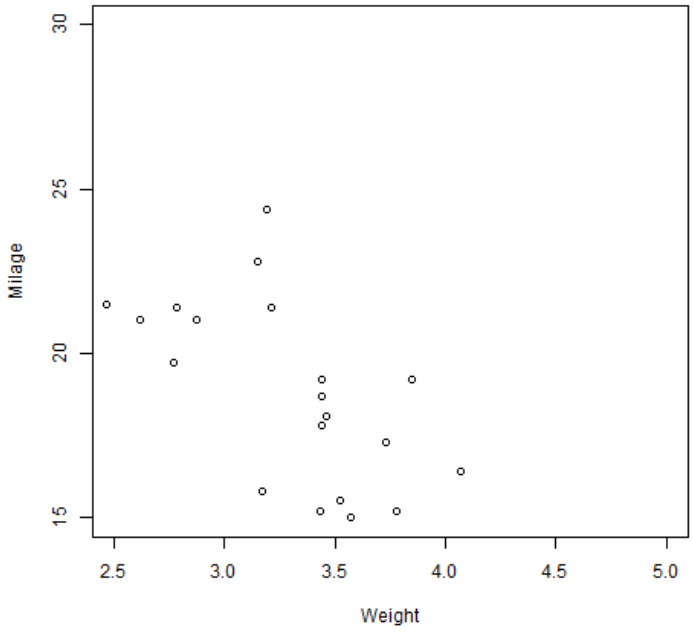
- A **histogram** represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.
  R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.

```
> # Create data for the graph.
> v <-  c(9,13,21,8,36,22,12,41,31,33,19)
>
> # Give the chart file a name.
> png(file = "histogram.png")
>
> # Create the histogram.
> hist(v,xlab = "Weight",col = "yellow",border = "blue")
>
> # Save the file.
> dev.off()
```



- **Scatterplots** show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.
  The simple scatterplot is created using the **plot()** function.

```
> input <- mtcars[,c('wt','mpg')]
> print(head(input))
                   wt  mpg
Mazda RX4         2.620 21.0
Mazda RX4 Wag     2.875 21.0
Datsun 710        2.320 22.8
Hornet 4 Drive    3.215 21.4
Hornet Sportabout 3.440 18.7
Valiant           3.460 18.1
> # Get the input values.
> input <- mtcars[,c('wt','mpg')]
>
> # Give the chart file a name.
> png(file = "scatterplot.png")
>
> # Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
> plot(x = input$wt,y = input$mpg,
+    xlab = "Weight",
+    ylab = "Milage",
+    xlim = c(2.5,5),
+    ylim = c(15,30),
+    main = "Weight vs Milage"
+ )
>
> # Save the file.
> dev.off()
```


Milage
30
25
20
15
2.5
3.0
3.5
4.0
4.5
5.0
Weight

PRACTICAL - 4

Aim: Performing Association Rule Mining using R Programming.

Theory:

Association Rule Mining (also called as Association Rule Learning) is a common technique used to find associations between many variables. It is often used by grocery stores, e-commerce websites, and anyone with large transactional databases. A most common example that we encounter in our daily lives — Amazon knows what else you want to buy when you order something on their site. The same idea extends to Spotify too — They know what song you want to listen to next. All of these incorporate, at some level, data mining concepts and association rule mining algorithms.

Market Basket Analysis is similar to ARM. Market Basket Analysis is a modelling technique based upon the theory that if you buy a certain group of items, you are more (or less) likely to buy another group of items. For example, if you are in an English pub and you buy a pint of beer and don't buy a bar meal, you are more likely to buy crisps at the same time than somebody who didn't buy beer.

There are three parameters controlling the number of rules to be generated *viz.* **Support and Confidence**. Another parameter **Lift** is generated using Support and Confidence and is one of the major parameters to filter the generated rules.

- ***Support*** is an indication of how frequently the itemset appears in the dataset. Consider only the two transactions from the above output. The support of the item *citrus fruit* is 1/2 as it appears in only 1 out of the two transactions.
- ***Confidence*** is an indication of how often the rule has been found to be true. We will discuss more about confidence after generating the rules.

The apriori() generates the most relevent set of rules from a given transaction data. It also shows the *support*, *confidence* and *lift* of those rules. These three measure can be used to decide the relative strength of the rules.?

- Lets consider the rule **A => B** in order to compute these metrics.

$$Support = \frac{Number\ of\ transactions\ with\ both\ A\ and\ B}{Total\ number\ of\ transactions} = P(A \cap B)$$

$$Confidence = \frac{Number\ of\ transactions\ with\ both\ A\ and\ B}{Total\ number\ of\ transactions\ with\ A} = \frac{P(A \cap B)}{P(A)}$$

$$ExpectedConfidence = \frac{Number\ of\ transactions\ with\ B}{Total\ number\ of\ transactions} = P(B)$$

$$Lift = \frac{Confidence}{Expected\ Confidence} = \frac{P(A \cap B)}{P(A).P(B)}$$

Example

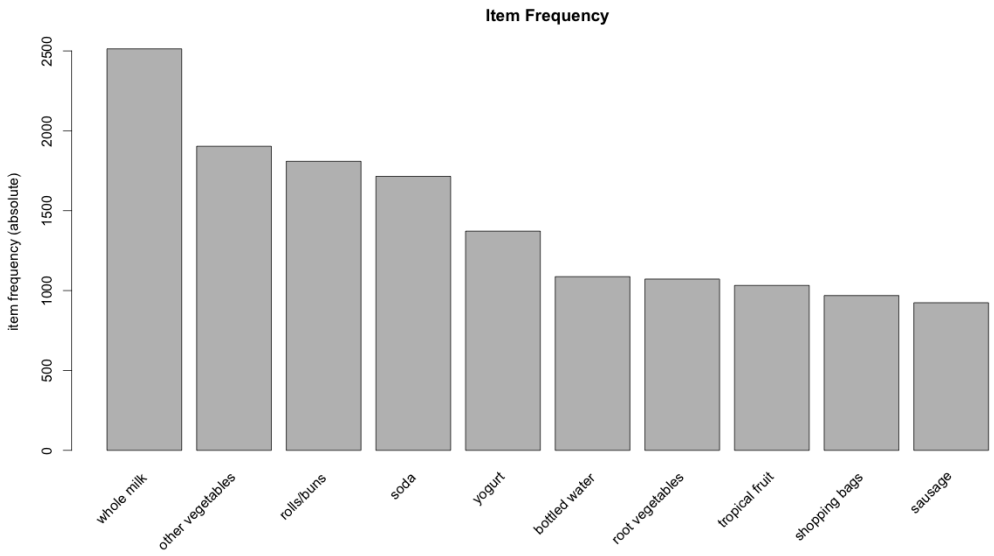
Transactions data

Lets play with the Groceries data that comes with the arules pkg. Unlike dataframe, using head(Groceries) does not display the transaction items in the data. To view the transactions, use the inspect() function instead.

Since association mining deals with transactions, the data has to be converted to one of class transactions, made available in R through the arules pkg. This is a necessary step because the apriori() function accepts transactions data of class transactions only.

```
library(arules)
class(Groceries)
#> [1] "transactions"
#>attr(,"package")
#> [1] "arules"
inspect(head(Groceries, 3))
#> items
#> 1 {citrus fruit,
#>    semi-finished bread,
#>    margarine,
#>    ready soups}
#> 2 {tropical fruit,
#>    yogurt,
#> coffee}
#> 3 {whole milk}
tdata<- read.transactions("transactions_data.txt", sep="\t")
tData<- as (myDataFrame, "transactions") # convert to 'transactions' class
size(head(Groceries)) # number of items in each observation
#> [1] 4 3 1 4 4 5
LIST(head(Groceries, 3)) # convert 'transactions' to a list, note the LIST in CAPS
#> [[1]]
#> [1] "citrus fruit"       "semi-finished bread" "margarine"
```

```
#> [4] "ready soups"
#>
#> [[2]]
#> [1] "tropical fruit" "yogurt"        "coffee"
#>
#> [[3]]
#> [1] "whole milk"
frequentItems<- eclat (Groceries, parameter = list(supp = 0.07, maxlen = 15)) # calculates support for frequent items
inspect(frequentItems)
#>    items                    support
#>1  {other vegetables,whole milk} 0.07483477
#>2  {whole milk}              0.25551601
#>3  {other vegetables}        0.19349263
#>4  {rolls/buns}              0.18393493
#>5  {yogurt}                  0.13950178
#>6  {soda}                    0.17437722
itemFrequencyPlot(Groceries, topN=10, type="absolute", main="Item Frequency") # plot frequent items
```



```
rules <- apriori (Groceries, parameter = list(supp = 0.001, conf = 0.5)) # Min Support as 0.001, confidence as 0.8.

rules_conf<- sort (rules, by="confidence", decreasing=TRUE) # 'high-confidence' rules.

inspect(head(rules_conf)) # show the support, lift and confidence for all rules

#>lhsrhs support confidence lift

#>113 {rice,sugar} => {whole milk} 0.001220132 1 3.913649

#>258 {canned fish,hygiene articles} => {whole milk} 0.001118454 1 3.913649

#> 1487 {root vegetables,butter,rice} => {whole milk} 0.001016777 1 3.913649

#> 1646 {root vegetables,whipped/sour cream,flour} => {whole milk} 0.001728521 1 3.913649

#> 1670 {butter,softcheese,domestic eggs} => {whole milk} 0.001016777 1 3.913649

#> 1699 {citrus fruit,rootvegetables,soft cheese} => {other vegetables} 0.001016777 1 5.168156

rules_lift<- sort (rules, by="lift", decreasing=TRUE) # 'high-lift' rules.

inspect(head(rules_lift)) # show the support, lift and confidence for all rules

#>lhsrhssupport confidence lift

#> 53 {Instant food products,soda} => {hamburger meat} 0.001220 0.6315789 18.995

#> 37 {soda,popcorn} => {salty snack} 0.001220 0.6315789 16.697

#>444 {flour,baking powder} => {sugar} 0.001016 0.5555556 16.408

#>327 {ham,processed cheese} => {white bread} 0.001931 0.6333333 15.045

#> 55 {whole milk,Instant food products} => {hamburger meat} 0.001525 0.5000000 15.038

#> 4807 {other vegetables,curd,yogurt,whipped/sour cream} => {cream cheese } 0.001016 0.5882353 14.834
```

The rules with confidence of 1 (see rules_conf above) imply that, whenever the LHS item was purchased, the RHS item was also purchased 100% of the time.

A rule with a lift of 18 (see rules_lift above) imply that, the items in LHS and RHS are 18 times more likely to be purchased together compared to the purchases when they are assumed to be unrelated.

Adjust the maxlen, supp and conf arguments in the apriori function to control the number of rules generated. You will have to adjust this based on the sparesness of you data.

```
rules <- apriori(Groceries, parameter = list (supp = 0.001, conf = 0.5, maxlen=3)) # maxlen = 3 limits the elements in a rule to 3
```

1. To get **'strong'** rules, increase the value of *'conf'* parameter.
2. To get **'longer'** rules, increase *'maxlen'*.

Sometimes it is desirable to remove the rules that are subset of larger rules. To do so, use the below code to filter the redundant rules.

```
subsetRules<- which(colSums(is.subset(rules, rules)) > 1) # get subset rules in vector
length(subsetRules)  #> 3913
rules <- rules[-subsetRules] # remove subset rules.
```

This can be achieved by modifying the appearance parameter in the apriori() function. For example,

To find what factors influenced purchase of product X

To find out what customers had purchased before buying 'Whole Milk'. This will help you understand the patterns that led to the purchase of 'whole milk'.

```
rules <- apriori (data=Groceries, parameter=list (supp=0.001,conf = 0.08), appearance = list (default="lhs",rhs="whole milk"), control = list (verbose=F)) # get rules that lead to buying 'whole milk'

rules_conf<- sort (rules, by="confidence", decreasing=TRUE) # 'high-confidence' rules.

inspect(head(rules_conf))

#>lhsrhs        support     confidence lift

#>196  {rice,sugar}                    => {whole milk} 0.001220132 1          3.913649

#>323  {canned fish,hygiene articles}          => {whole milk} 0.001118454 1          3.91
3649

#> 1643 {root vegetables,butter,rice}          => {whole milk} 0.001016777 1          3.91
3649
```

```
#> 1705 {root vegetables,whipped/sour cream,flour} => {whole milk} 0.001728521 1 3.913649
#> 1716 {butter,softcheese,domestic eggs}          => {whole milk} 0.001016777 1          3. 913649
#> 1985 {pip fruit,butter,hygiene articles}        => {whole milk} 0.001016777 1          3.9 13649
```

To find out what products were purchased after/along with product X

The is a case to find out the Customers who bought ‘Whole Milk’ also bought . . In the equation, ‘whole milk’ is in LHS (left hand side).

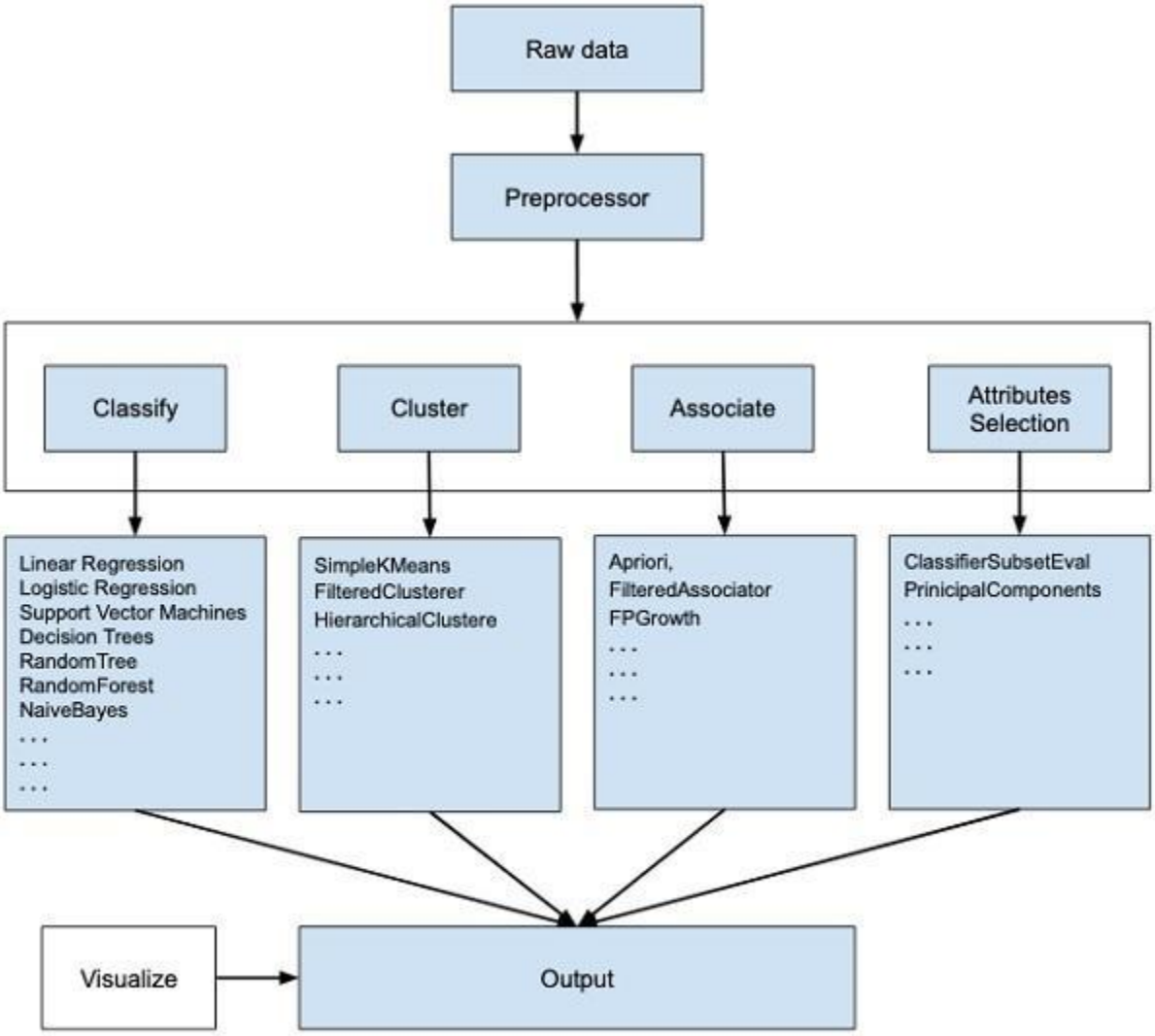
```
rules <- apriori (data=Groceries, parameter=list (supp=0.001,conf = 0.15,minlen=2), app earance = list(default="rhs",lhs="whole milk"), control = list (verbose=F)) # those who b ought 'milk' also bought..

rules_conf<- sort (rules, by="confidence", decreasing=TRUE) # 'high-confidence' rules.

inspect(head(rules_conf))

#>lhsrhs              support    confidence lift

#> 6 {whole milk} => {other vegetables} 0.07483477 0.2928770  1.5136341

#> 5 {whole milk} => {rolls/buns}       0.05663447 0.2216474  1.2050318

#> 4 {whole milk} => {yogurt}           0.05602440 0.2192598  1.5717351

#> 2 {whole milk} => {root vegetables}  0.04890696 0.1914047  1.7560310

#> 1 {whole milk} => {tropical fruit}   0.04229792 0.1655392  1.5775950

#> 3 {whole milk} => {soda}             0.04006101 0.1567847  0.8991124
```

PRACTICAL - 5

Aim: Perform Different Data Mining Activities using Weka Explorer Tool (Open Source Data Mining Tool) & Experimental Tool (Open Source Data Mining Tool).

Theory:

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



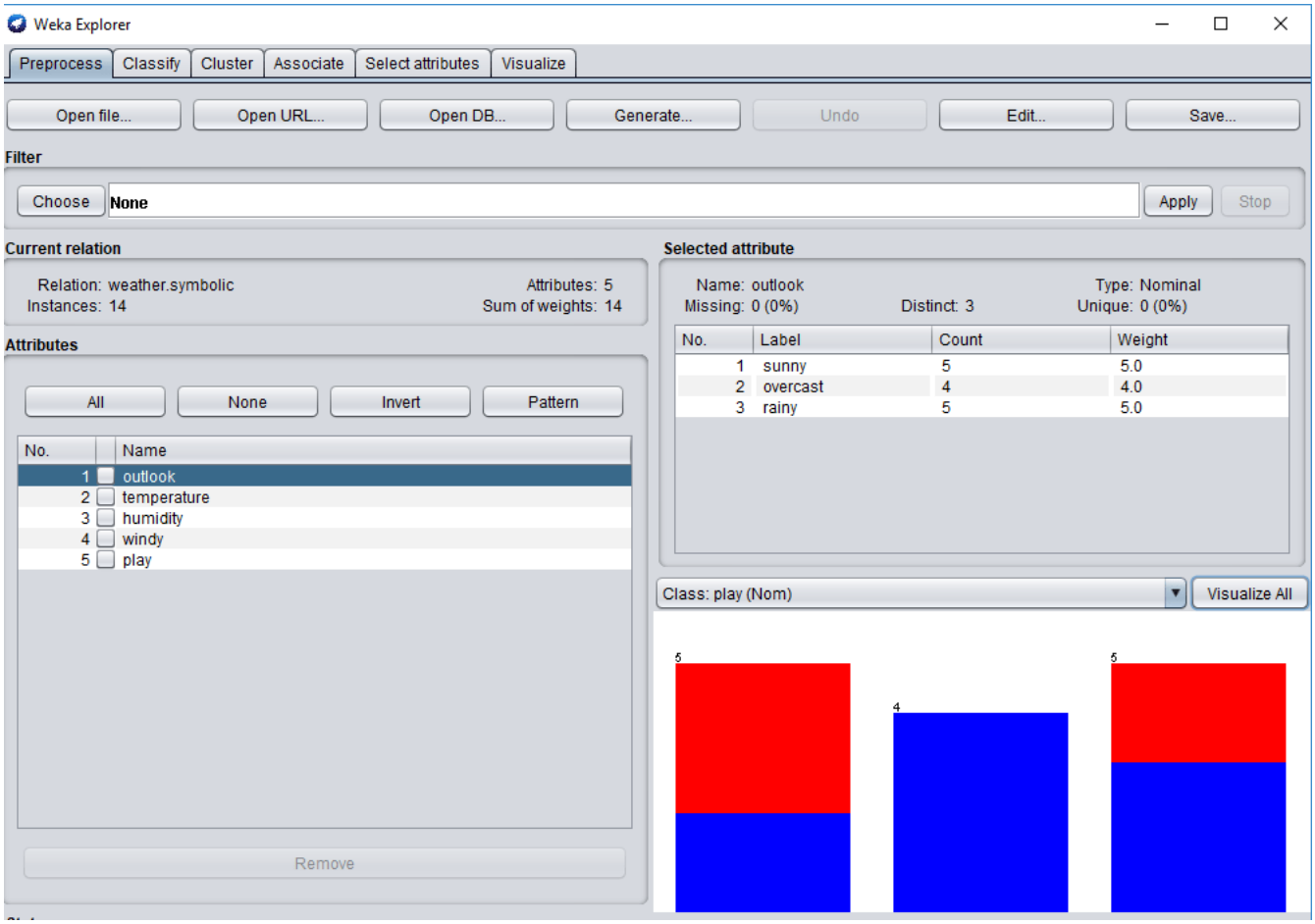
- If you observe the beginning of the flow of the image, you will understand that there are many stages in dealing with Big Data to make it suitable for machine learning −
- First, you will start with the raw data collected from the field. This data may contain several null values and irrelevant fields. You use the data preprocessing tools provided in WEKA to cleanse the data.

- Then, you would save the preprocessed data in your local storage for applying ML algorithms.
- Next, depending on the kind of ML model that you are trying to develop you would select one of the options such as **Classify, Cluster**, or **Associate**. The **Attributes Selection** allows the automatic selection of features to create a reduced dataset.
- Note that under each category, WEKA provides the implementation of several algorithms. You would select an algorithm of your choice, set the desired parameters and run it on the dataset.
- Then, WEKA would give you the statistical output of the model processing. It provides you a visualization tool to inspect the data.
- The various models can be applied on the same dataset. You can then compare the outputs of different models and select the best that meets your purpose.
- Thus, the use of WEKA results in a quicker development of machine learning models on the whole.
-

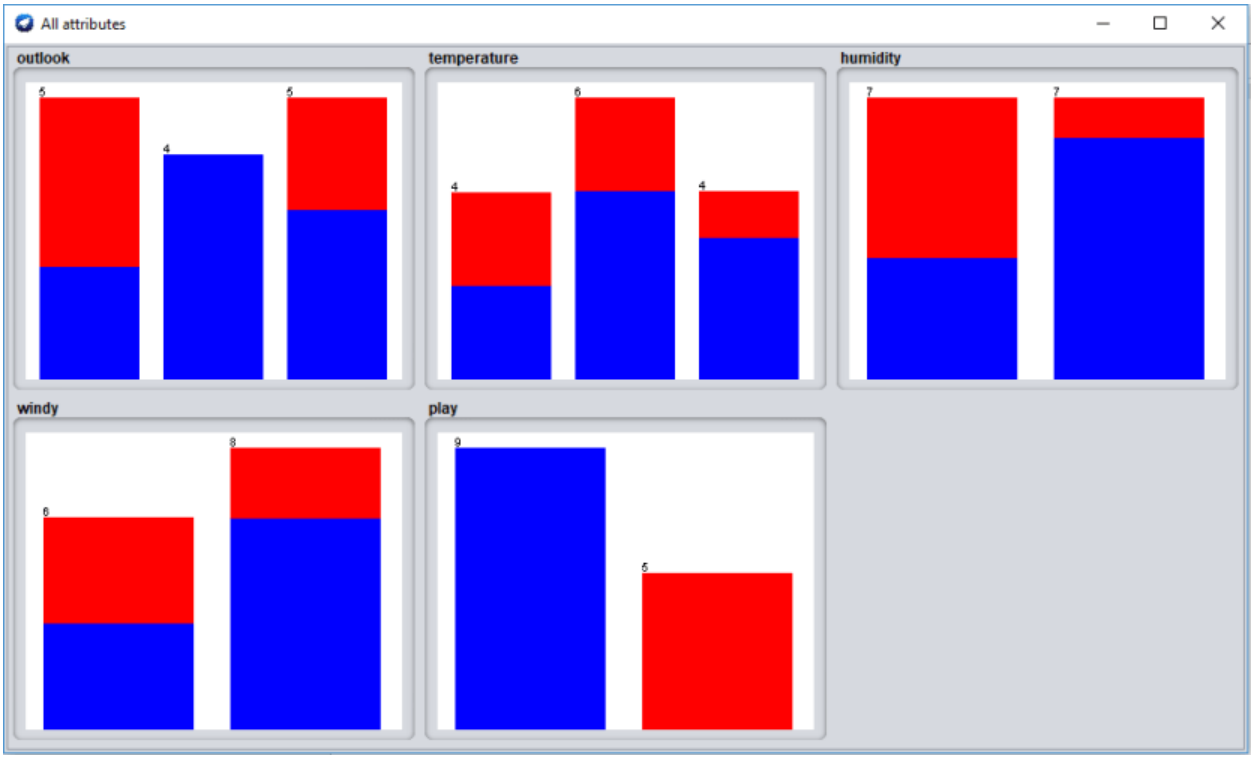
Output/Working:

Data: weather_nominal.arff

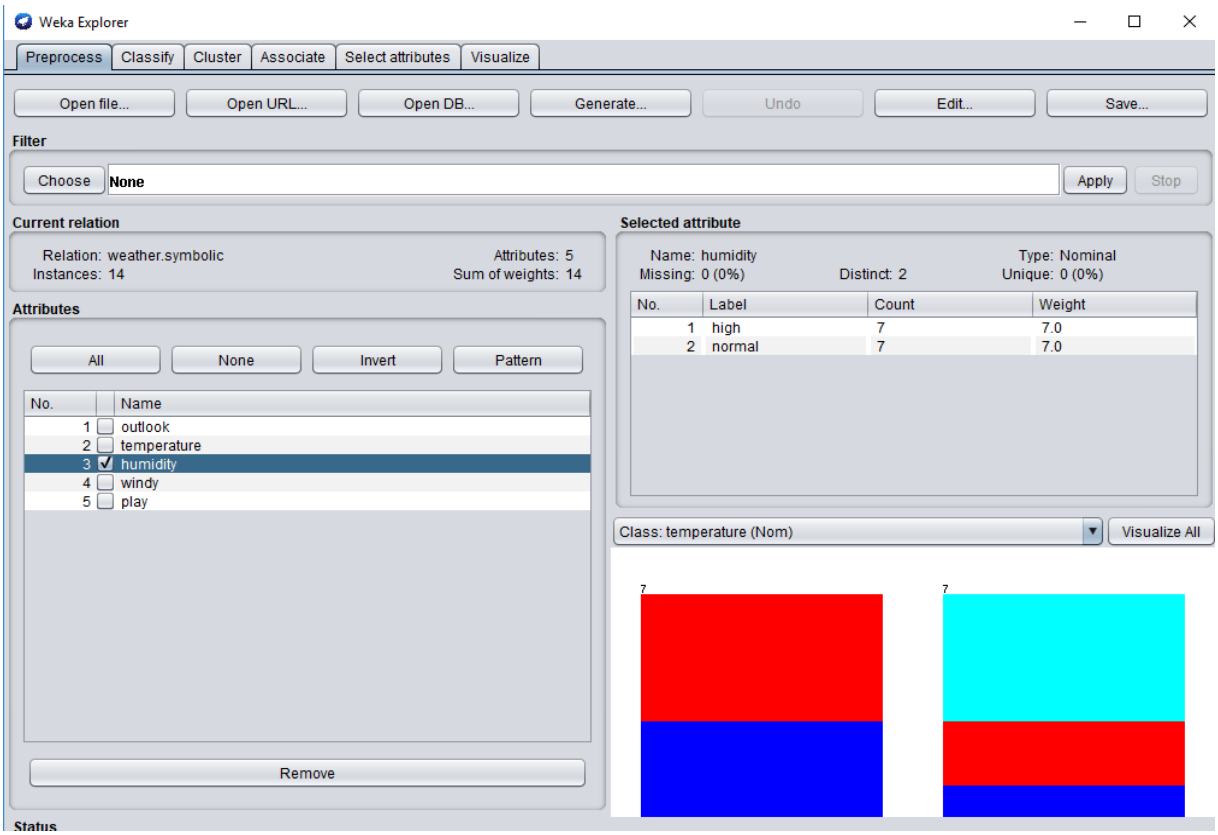
Preprocessing

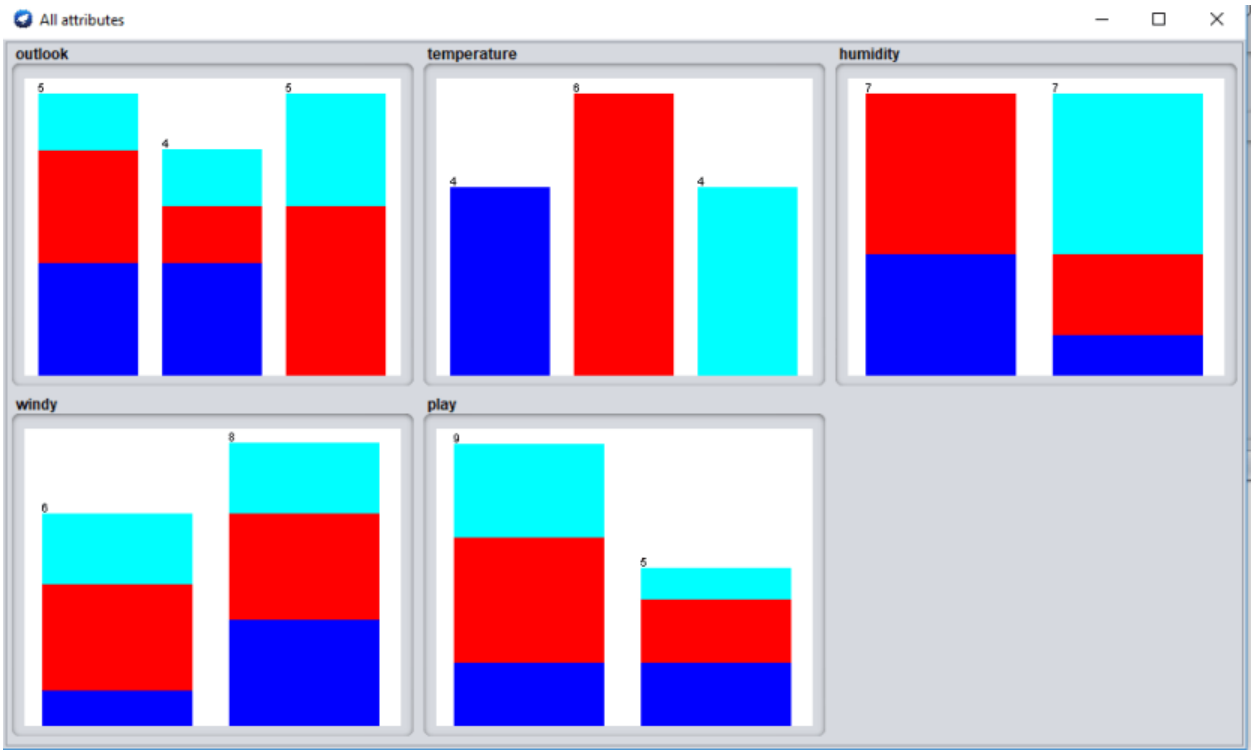


Statistics:



Attribute Selection:


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: weather.symbolic
Instances: 14
Attributes: 5
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
Remove
Selected attribute
Name: humidity
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 high 7 7.0
2 normal 7 7.0
Class: temperature (Nom)
Visualize All
Status


All attributes
outlook
temperature
humidity
windy
play

PRACTICAL - 6

Aim: Predicting Sports Winners with Decision Trees.

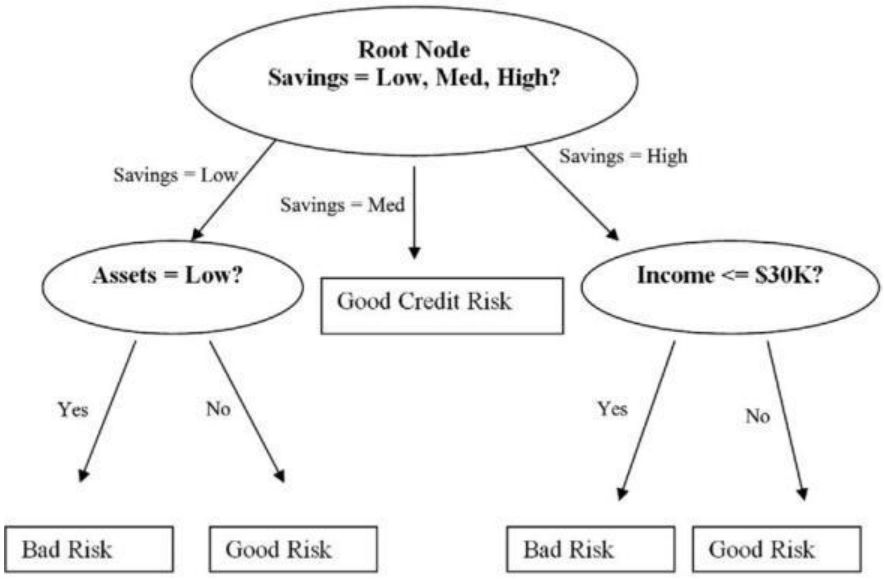
Theory:

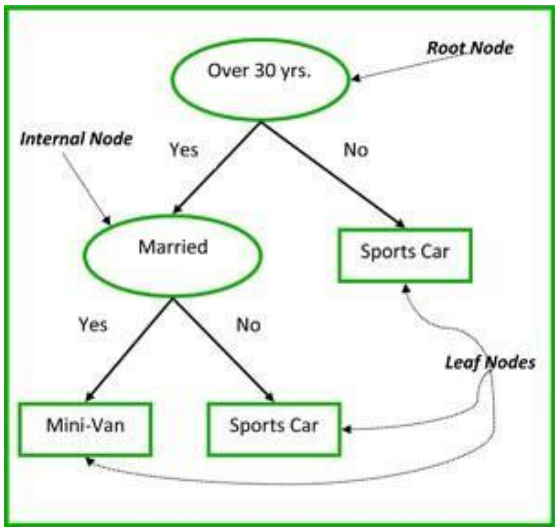
Introduction to Decision Trees

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Tree based learning algorithms are considered to be one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression). Decision Tree algorithms are referred to as CART (Classification and Regression Trees).

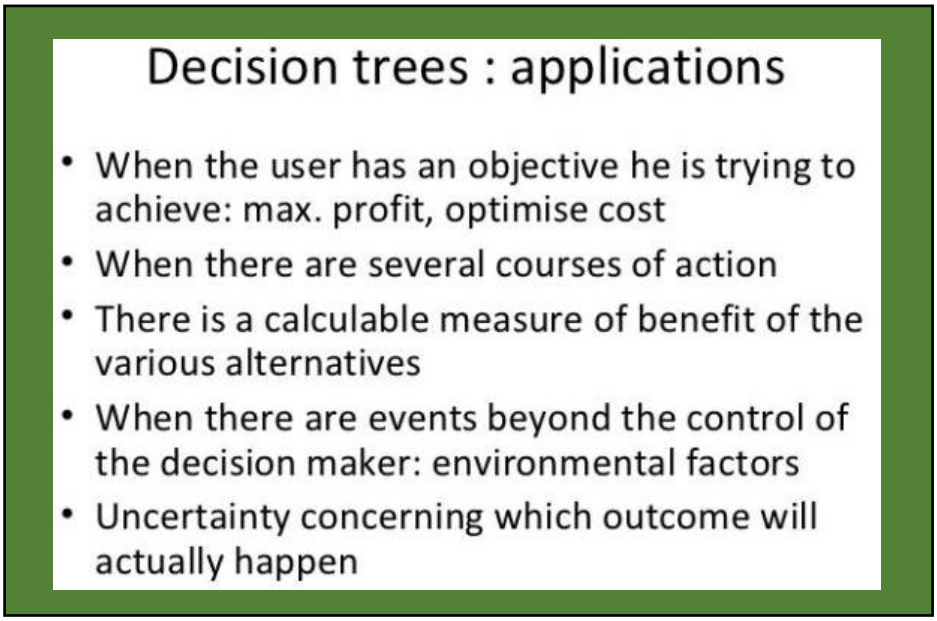
Methods like decision trees, random forest, gradient boosting are being popularly used in all kinds of data science problems.





Common terms used with Decision trees:

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning**: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.



As a result, the decision making tree is one of the more popular classification algorithms being used in Data Mining and Machine Learning. Example applications include:

· Evaluation of brand expansion opportunities for a business using historical sales data

· Determination of likely buyers of a product using demographic data to enable targeting of limited advertisement budget

· Prediction of likelihood of default for applicant borrowers using predictive models generated from historical data

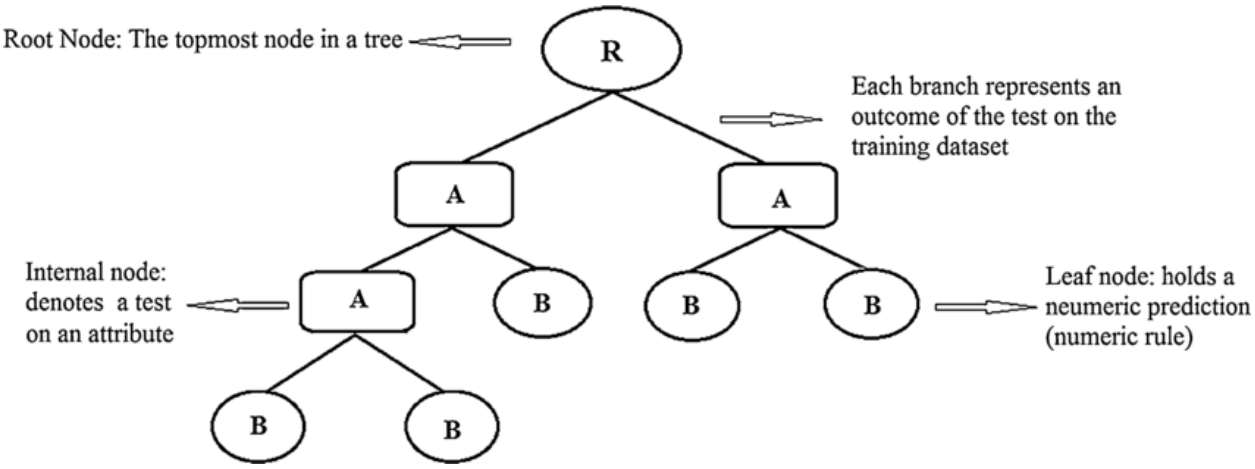
· Help with prioritization of emergency room patient treatment using a predictive model based on factors such as age, blood pressure, gender, location and severity of pain, and other measurements

· Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal.

Because of their simplicity, tree diagrams have been used in a broad range of industries and disciplines including civil planning, energy, financial, engineering, healthcare, pharmaceutical, education, law, and business.

How does Decision Tree works ?

Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.



Types of Decision Trees

1. **Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree. E.g.:- In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.
2. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

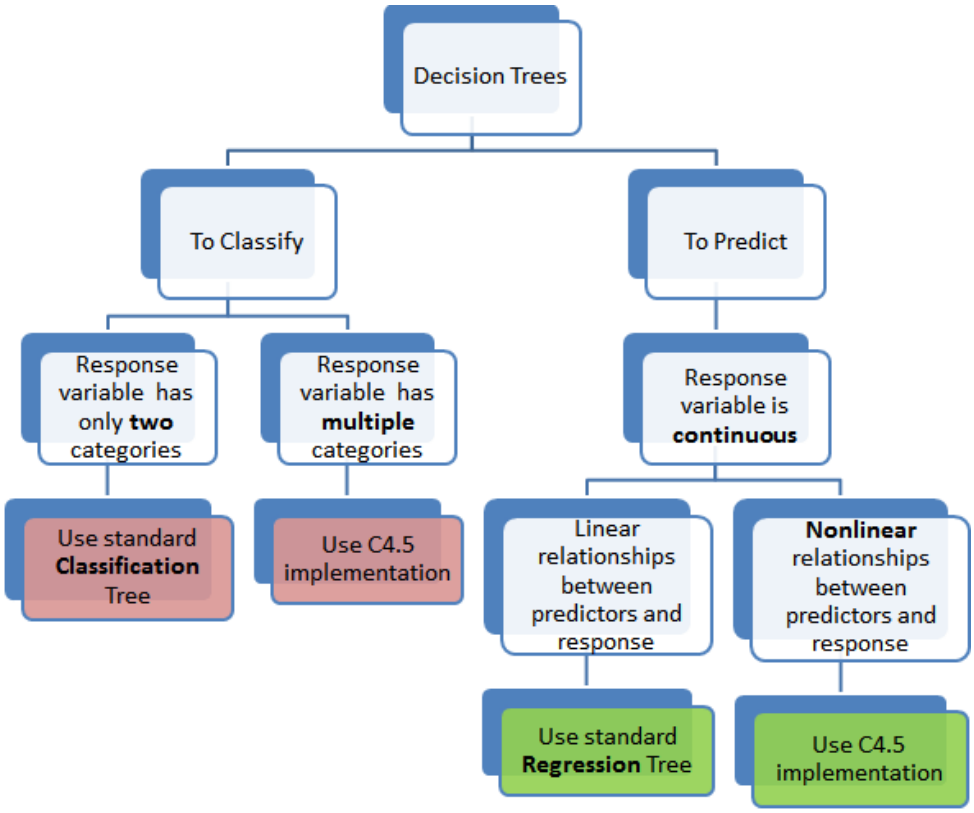
The decision tree algorithm tries to solve the problem, by using tree representation. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label.
Place the best attribute of the dataset at the root of the tree.
Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

Advantages:

1. Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
2. A decision tree does not require normalization of data.
3. A decision tree does not require scaling of data as well.
4. Missing values in the data also does NOT affect the process of building decision tree to any considerable extent.
5. A Decision trees model is very intuitive and easy to explain to technical teams as well as stakeholders.

Disadvantage:

1. A small change in the data can cause a large change in the structure of the decision tree causing instability.
2. For a Decision tree sometimes calculation can go far more complex compared to other algorithms.
3. Decision tree often involves higher time to train the model.
4. Decision tree training is relatively expensive as complexity and time taken is more.
5. Decision Tree algorithm is inadequate for applying regression and predicting continuous values.



Here in this practical we will look at predicting the winner of games of the National Basketball Association (NBA) using a different type of classification algorithm—decision trees.

Collecting the data:

The data we will be using is the match history data for the NBA, for the 2013-2014 season. The Basketball-Reference.com website contains a significant number of resources and statistics collected from the NBA and other leagues. Perform the following steps to download the dataset:

1. Navigate to http://www.basketball-reference.com/leagues/NBA_2014_games.html in your web browser.
2. Click on the Export button next to the Regular Season heading.
3. Download the file to your data folder (and make a note of the path).

We will load the file with the pandas library, which is an incredibly useful library for manipulating data. Python also contains a built-in library called *csv* that supports reading and writing CSV files. We will use pandas instead as it provides more powerful functions to work with datasets.

Working:

In [1]:
```
%matplotlib inline

import numpy as np
import pandas as pd

from collections import defaultdict

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier


from scipy.sparse import hstack # np.vstack/hstack does not work.
```

In [2]:
```
df = pd.read_csv('leagues_nba_2014_games.txt',
                 parse_dates=['Date']) # We have a column called "Date".
df.iloc[:3]
```

Out[2]:

	Date	Start (ET)	Visitor/Neutral	PTS	Home/Neutral	PTS.1	Unnamed: 6	Unnamed: 7	Notes
0	2013-10-29	7:00 pm	Orlando Magic	87	Indiana Pacers	97	Box Score	NaN	NaN
1	2013-10-29	10:30 pm	Los Angeles Clippers	103	Los Angeles Lakers	116	Box Score	NaN	NaN
2	2013-10-29	8:00 pm	Chicago Bulls	95	Miami Heat	107	Box Score	NaN	NaN

In [3]:
```
original_columns = df.columns.tolist()
original_columns
```

Out[3]:
```
['Date',
 'Start (ET)',
 'Visitor/Neutral',
 'PTS',
 'Home/Neutral',
 'PTS.1',
 'Unnamed: 6',
 'Unnamed: 7',
 'Notes']
```

In [4]:
```
renamed_columns = ['Date',
                   'Start (ET)',
                   'Visitor Team',
                   'VisitorPts',
                   'Home Team',
                   'HomePts',
                   'Score Type',
                   'OT?',
                   'Notes']
df = df.rename(columns=dict(zip(original_columns, renamed_columns)))
df.head()
```

Out[4]:

	Date	Start (ET)	Visitor Team	VisitorPts	Home Team	HomePts	Score Type	OT?	Notes
0	2013-10-29	7:00 pm	Orlando Magic	87	Indiana Pacers	97	Box Score	NaN	NaN
1	2013-10-29	10:30 pm	Los Angeles Clippers	103	Los Angeles Lakers	116	Box Score	NaN	NaN
2	2013-10-29	8:00 pm	Chicago Bulls	95	Miami Heat	107	Box Score	NaN	NaN
3	2013-10-30	7:00 pm	Brooklyn Nets	94	Cleveland Cavaliers	98	Box Score	NaN	NaN
4	2013-10-30	8:30 pm	Atlanta Hawks	109	Dallas Mavericks	118	Box Score	NaN	NaN

In [5]:
```
# Adding new features.
df['HomeWin'] = df['HomePts'] > df['VisitorPts']
```

In [6]:
```
y_true = df['HomeWin'].values
y_true
```

Out[6]:
```
array([ True,  True,  True, ..., False, False,  True])
```

In [7]:
```
won_last = defaultdict(int)

for index, row in df.iterrows():
    home, visitor = row['Home Team'], row['Visitor Team']
    row['HomeLastWin'] = won_last[home]
    row['VisitorLastWin'] = won_last[visitor]
    df.loc[index] = row

    won_last[home] = row['HomeWin']
    won_last[visitor] = not row['HomeWin']

True
```

Out[7]:
```
True
```

In [8]:
```
df.iloc[20:25]
```

Out[8]:

	Date	Start (ET)	Visitor Team	VisitorPts	Home Team	HomePts	Score Type	OT?	Notes	HomeWin
20	2013-11-01	7:30 pm	Milwaukee Bucks	105	Boston Celtics	98	Box Score	NaN	NaN	False
21	2013-11-01	8:00 pm	Miami Heat	100	Brooklyn Nets	101	Box Score	NaN	NaN	True
22	2013-11-01	7:00 pm	Cleveland Cavaliers	84	Charlotte Bobcats	90	Box Score	NaN	NaN	True
23	2013-11-01	9:00 pm	Portland Trail Blazers	113	Denver Nuggets	98	Box Score	NaN	NaN	False
24	2013-11-01	8:00 pm	Dallas Mavericks	105	Houston Rockets	113	Box Score	NaN	NaN	True

In [10]:
```
df_standings = pd.read_csv('leagues_nba_2013_standings.txt', skiprows=[0])
df_standings.head()
```

Out[10]:

	Rk	Team	Overall	Home	Road	E	W	A	C	SE	...	Post	=3	=10	Oct	Nov	De
0	1	Miami Heat	66-16	37-4	29-12	41-11	25-5	14-4	12-6	15-1	...	30-2	9-3	39-8	1-0	10-3	1
1	2	Oklahoma City Thunder	60-22	34-7	26-15	21-9	39-13	7-3	8-2	6-4	...	21-8	3-6	44-6	NaN	13-4	1
2	3	San Antonio Spurs	58-24	35-6	23-18	25-5	33-19	8-2	9-1	8-2	...	16-12	9-5	31-10	1-0	12-4	1
3	4	Denver Nuggets	57-25	38-3	19-22	19-11	38-14	5-5	10-0	4-6	...	24-4	11-7	28-8	0-1	8-8	9
4	5	Los Angeles Clippers	56-26	32-9	24-17	21-9	35-17	7-3	8-2	6-4	...	17-9	3-5	38-12	1-0	8-6	1

5 rows × 24 columns

In [11]:
```
df['HomeTeamRanksHigher'] = 0
for index, row in df.iterrows():
    home, visitor = row['Home Team'], row['Visitor Team']

    # The team was renamed between the 2013 and 2014 seasons!
    # But it was still the same team.
    if home == 'New Orleans Pelicans':
        home = 'New Orleans Hornets'
    elif visitor == 'New Orleans Pelicans':
        visitor = 'New Orleans Hornets'

    home_rank = df_standings[df_standings['Team'] == home]['Rk'].values[0]
    visitor_rank = df_standings[df_standings['Team'] == visitor]['Rk'].values[0]
    row['HomeTeamRanksHigher'] = int(home_rank > visitor_rank)
    df.loc[index] = row
True
```

Out[11]: True

In [14]:
```
last_match_winner = defaultdict(int)
df['HomeTeamLastWon'] = 0
```

In [15]:
```
for index, row in df.iterrows():
    home, visitor = row['Home Team'], row['Visitor Team']
    teams = tuple(sorted([home, visitor]))

    row['HomeTeamLastWon'] = 1 if last_match_winner[teams] == row['Home Team'] el
    df.loc[index] = row
    winner = row['Home Team'] if row['HomeWin'] else row['Visitor Team']
    last_match_winner[teams] = winner
```

In [16]:
```
X_last_winner = df[['HomeTeamRanksHigher', 'HomeTeamLastWon']].values
```

In [17]:
```
clf = DecisionTreeClassifier(random_state=14)
scores = cross_val_score(clf, X_last_winner, y_true, cv=5)
print(f'Accuracy: {np.mean(scores) * 100:.2f}%')
```

```
Accuracy: 59.89%
```

In [19]:
```
encoder = LabelEncoder()
encoder.fit(df['Home Team'].values)
home_teams = encoder.transform(df['Home Team'].values)
visitor_teams = encoder.transform(df['Visitor Team'].values)
X_teams = np.vstack([home_teams, visitor_teams]).T
```

In [21]:
```
clf = RandomForestClassifier(random_state=14, n_estimators=100)
scores = cross_val_score(clf, X_teams, y_true, scoring='accuracy', cv=5)
print(f'Accuracy: {np.mean(scores) * 100:.2f}%')
```

```
Accuracy: 58.60%
```

In [1]:
```
import pandas as pd
from collections import defaultdict

import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, train_test_split, GridSearch
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
input_file = 'leagues_nba_2014_games.txt' # Source: https://www.basketball-refere
```

In [3]:
```
# Load data
df = pd.read_csv(input_file, parse_dates = ['Date'])

# Rename existing columns
df.rename(columns = {'Visitor/Neutral': 'Visitor Team',
                     'Home/Neutral': 'Home Team',
                     'Unnamed: 6': 'Score Type',
                     'Unnamed: 7': 'OT?',
                     'PTS': 'VisitorPts',
                     'PTS.1': 'HomePts'}, inplace = True)
df.head()
```

Out[3]:

	Date	Start (ET)	Visitor Team	VisitorPts	Home Team	HomePts	Score Type	OT?	Notes
0	2013-10-29	7:00 pm	Orlando Magic	87	Indiana Pacers	97	Box Score	NaN	NaN
1	2013-10-29	10:30 pm	Los Angeles Clippers	103	Los Angeles Lakers	116	Box Score	NaN	NaN
2	2013-10-29	8:00 pm	Chicago Bulls	95	Miami Heat	107	Box Score	NaN	NaN
3	2013-10-30	7:00 pm	Brooklyn Nets	94	Cleveland Cavaliers	98	Box Score	NaN	NaN
4	2013-10-30	8:30 pm	Atlanta Hawks	109	Dallas Mavericks	118	Box Score	NaN	NaN

In [4]:
```
df.describe()
```

Out[4]:

	VisitorPts	HomePts
count	1319.000000	1319.000000
mean	99.603487	102.216831
std	11.684056	11.888199
min	66.000000	63.000000
25%	91.000000	94.000000
50%	99.000000	102.000000
75%	107.000000	110.000000
max	145.000000	143.000000

In [5]:
```
# Create a new column `HomeWin` which is boolean that indicates if the Home is th
# team or Visitor
df['HomeWin'] = df['HomePts'] > df['VisitorPts']
df.head()
```

Out[5]:

	Date	Start (ET)	Visitor Team	VisitorPts	Home Team	HomePts	Score Type	OT?	Notes	HomeWin
0	2013-10-29	7:00 pm	Orlando Magic	87	Indiana Pacers	97	Box Score	NaN	NaN	True
1	2013-10-29	10:30 pm	Los Angeles Clippers	103	Los Angeles Lakers	116	Box Score	NaN	NaN	True
2	2013-10-29	8:00 pm	Chicago Bulls	95	Miami Heat	107	Box Score	NaN	NaN	True
3	2013-10-30	7:00 pm	Brooklyn Nets	94	Cleveland Cavaliers	98	Box Score	NaN	NaN	True
4	2013-10-30	8:30 pm	Atlanta Hawks	109	Dallas Mavericks	118	Box Score	NaN	NaN	True

In [7]:
```
# Create a new classifier
clf = DecisionTreeClassifier(random_state=4)
```

In [8]:
```
X_previous_wins = df[['HomeLastWin', 'VisitorLastWin']].values
y_true = df['HomeWin'].values
```

In [9]:
```
scores = cross_val_score(clf, X_previous_wins, y_true, scoring='accuracy')
print('Accuracy: {:.1f}%'.format(np.mean(scores) * 100))
```

```
Accuracy: 57.9%
```

In [10]:
```
# Load the standings data
standings_filename = 'leagues_nba_2013_standings.txt'
```

In [11]:
```
standings = pd.read_csv(standings_filename, skiprows=[0])
standings.head()
```

Out[11]:

	Rk	Team	Overall	Home	Road	E	W	A	C	SE	...	Post	=3	=10	Oct	Nov	De
0	1	Miami Heat	66-16	37-4	29-12	41-11	25-5	14-4	12-6	15-1	...	30-2	9-3	39-8	1-0	10-3	1(
1	2	Oklahoma City Thunder	60-22	34-7	26-15	21-9	39-13	7-3	8-2	6-4	...	21-8	3-6	44-6	NaN	13-4	1
2	3	San Antonio Spurs	58-24	35-6	23-18	25-5	33-19	8-2	9-1	8-2	...	16-12	9-5	31-10	1-0	12-4	1:
3	4	Denver Nuggets	57-25	38-3	19-22	19-11	38-14	5-5	10-0	4-6	...	24-4	11-7	28-8	0-1	8-8	9
4	5	Los Angeles Clippers	56-26	32-9	24-17	21-9	35-17	7-3	8-2	6-4	...	17-9	3-5	38-12	1-0	8-6	1(

5 rows × 24 columns

In [12]:
```
df['HomeTeamRanksHigher'] = 0

# Note that iterrows create a copy, so you can update the column with it
for index, row in df.iterrows():
    home_team = row['Home Team']
    visitor_team = row['Visitor Team']

    # Correct the 2013 team name that has been renamed in 2014
    if home_team == 'New Orleans Pelicans':
        home_team = 'New Orleans Hornets'
    elif visitor_team == 'New Orleans Pelicans':
        visitor_team = 'New Orleans Hornets'

    home_rank = standings[standings['Team'] == home_team]['Rk'].values[0]
    visitor_rank = standings[standings['Team'] == visitor_team]['Rk'].values[0]

    # Overwrite the values for the column
    df.loc[index, 'HomeTeamRanksHigher'] = int(home_rank > visitor_rank)
print('done')
```

```
done
```

In [13]:
```
df.head()
```

Out[13]:

	Date	Start (ET)	Visitor Team	VisitorPts	Home Team	HomePts	Score Type	OT?	Notes	HomeWin	HomeLas
0	2013-10-29	7:00 pm	Orlando Magic	87	Indiana Pacers	97	Box Score	NaN	NaN	True	F
1	2013-10-29	10:30 pm	Los Angeles Clippers	103	Los Angeles Lakers	116	Box Score	NaN	NaN	True	F
2	2013-10-29	8:00 pm	Chicago Bulls	95	Miami Heat	107	Box Score	NaN	NaN	True	F
3	2013-10-30	7:00 pm	Brooklyn Nets	94	Cleveland Cavaliers	98	Box Score	NaN	NaN	True	F
4	2013-10-30	8:30 pm	Atlanta Hawks	109	Dallas Mavericks	118	Box Score	NaN	NaN	True	F

In [14]:
```
X_home_higher = df[['HomeLastWin', 'VisitorLastWin', 'HomeTeamRanksHigher']].valu
print(X_home_higher)
```

```
[[False False 0]
 [False False 1]
 [False False 0]
 ...
 [False False 0]
 [False False 0]
 [False False 1]]
```

In [15]:
```
# Create classifier
clf = DecisionTreeClassifier(random_state=14)
scores = cross_val_score(clf, X_home_higher, y_true, scoring='accuracy')
print('Accuracy is: {:.1f}%'.format(np.mean(scores) * 100))
```

```
Accuracy is: 57.4%
```

In [16]:
```
# Create a dictionary that store the winner of the past game and
# create a new feature in our data frame
last_match_winner = defaultdict(int)
df['HomeTeamLastWon'] = 0

for index, row in df.iterrows():
    home_team = row['Home Team']
    visitor_team = row['Visitor Team']

    teams = tuple(sorted([home_team, visitor_team]))
    row['HomeTeamWonLast'] = 1 if last_match_winner[teams] == row['Home Team'] el

    # Update the column
    df.loc[index, 'HomeTeamWonLast'] = row['HomeTeamWonLast']

    winner = row['Home Team'] if row['HomeWin'] else row['Visitor Team']
    last_match_winner[teams] = winner
print('done')
```

```
done
```

In [17]:
```
X_last_winner = df[['HomeTeamRanksHigher', 'HomeTeamWonLast']].values
clf = DecisionTreeClassifier(random_state=14)
scores = cross_val_score(clf, X_last_winner, y_true, scoring='accuracy')
print('Accuracy: {:.1f}%'.format(np.mean(scores) * 100))
```

```
Accuracy: 57.8%
```

In [18]:
```
# Create a new label encoder to convert between string-based team names into inte
encoding = LabelEncoder()

# Fit the transformer to the home teams
encoding.fit(df['Home Team'].values)
```

Out[18]:
```
LabelEncoder()
```

In [19]:
```
home_teams = encoding.transform(df['Home Team'].values)
visitor_teams = encoding.transform(df['Visitor Team'].values)
X_teams = np.vstack([home_teams, visitor_teams]).T
```

In [20]:
```
# Create one hot encoder to encode integers into numbers of integer values
onehot = OneHotEncoder()
```

In [21]:
```
X_teams_expanded = onehot.fit_transform(X_teams).todense()
```

In [22]:
```
clf = DecisionTreeClassifier(random_state=14)

scores = cross_val_score(clf, X_teams_expanded, y_true, scoring='accuracy')
print('Accuracy is {:.1f}%'.format(np.mean(scores) * 100))
```

```
Accuracy is 59.5%
```

In [23]:
```
clf = RandomForestClassifier(random_state=14)

scores = cross_val_score(clf, X_teams, y_true, scoring='accuracy')
print('Accuracy is {:.1f}%'.format(np.mean(scores) * 100))
```

```
Accuracy is 58.4%
```

In [24]:
```
X_all = np.hstack([X_home_higher, X_teams])

clf = RandomForestClassifier(random_state=14)

scores = cross_val_score(clf, X_all, y_true, scoring='accuracy')
print('Accuracy is {:.1f}%'.format(np.mean(scores) * 100))
```

```
Accuracy is 58.8%
```

In [25]:
```
# Perform grid search
parameter_space = {
    'max_features': [2, 10, 'auto'],
    'n_estimators': [100,],
    'criterion': ['gini', 'entropy'],
    'min_samples_leaf': [2, 4, 6]
}
clf = RandomForestClassifier(random_state=14)
grid = GridSearchCV(clf, parameter_space)
grid.fit(X_teams_expanded, y_true)
print('Accuracy of grid search: {0:.1f}%'.format(grid.best_score_ * 100))
```

```
Accuracy of grid search: 65.0%
```

In [26]:
```
print(grid.best_estimator_)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features=2, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=4, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=14, verbose=0, warm_start=False)
```

PRACTICAL - 7

Aim: Text Classification using Naïve Bayes, SVM and Evaluating Classification Models.

Theory:

Principle of Naive Bayes Classifier:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

Bayes Theorem:

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

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Types of Naive Bayes Classifier:

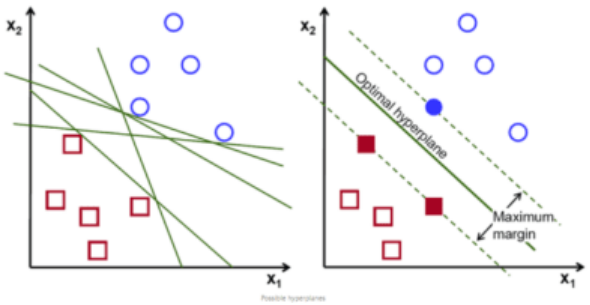
1. **Multinomial Naive Bayes:**
   This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

2. **Bernoulli Naive Bayes:**
   This is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

3. **Gaussian Naive Bayes:**
   When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

Support Vector Machine

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



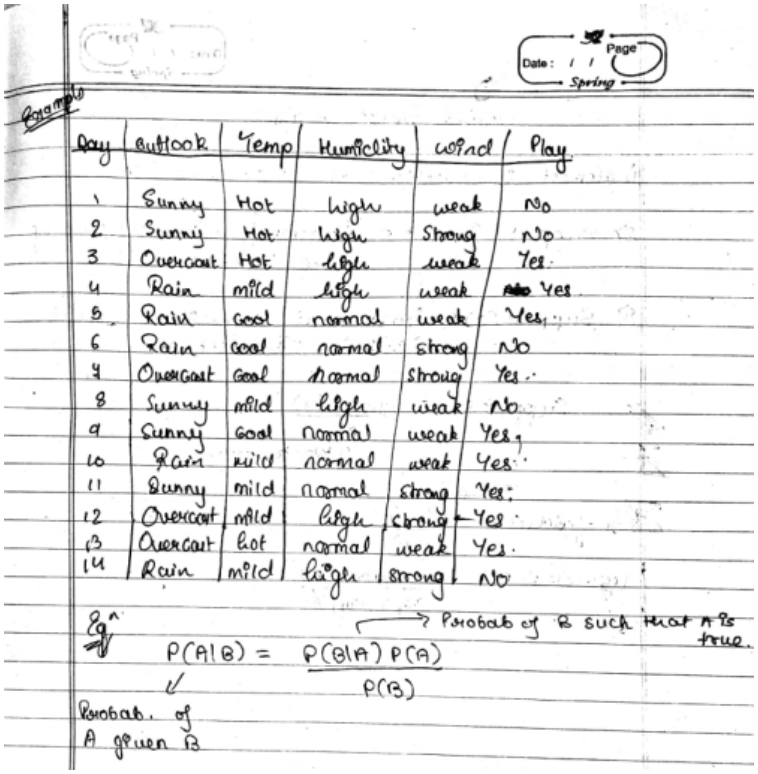
SVM Implementation in Python

The dataset we will be using to implement our SVM algorithm is the Iris dataset.

```
import pandas as pd
df = pd.read_csv('/Users/rohith/Documents/Datasets/Iris_dataset/iris.csv')
df = df.drop(['Id'],axis=1)
target = df['Species']
s = set()
for val in target:
s.add(val)
s = list(s)
rows = list(range(100,150))
df = df.drop(df.index[rows])
```

Program Working:

Dataset :



Example

Day	Outlook	Temp	Humidity	Wind	Play
1	Sunny	Hot	High	weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	weak	Yes
4	Rain	mild	High	weak	Yes
5	Rain	Cool	normal	weak	Yes
6	Rain	Cool	normal	strong	No
7	Overcast	Cool	normal	Strong	Yes
8	Sunny	mild	High	weak	No
9	Sunny	Cool	normal	weak	Yes
10	Rain	mild	normal	weak	Yes
11	Sunny	mild	normal	Strong	Yes
12	Overcast	mild	High	strong	Yes
13	Overcast	hot	normal	weak	Yes
14	Rain	mild	High	Strong	No

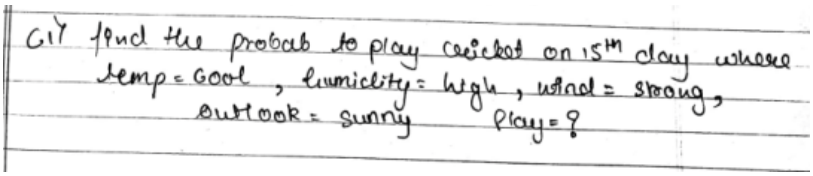
Eq^n

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

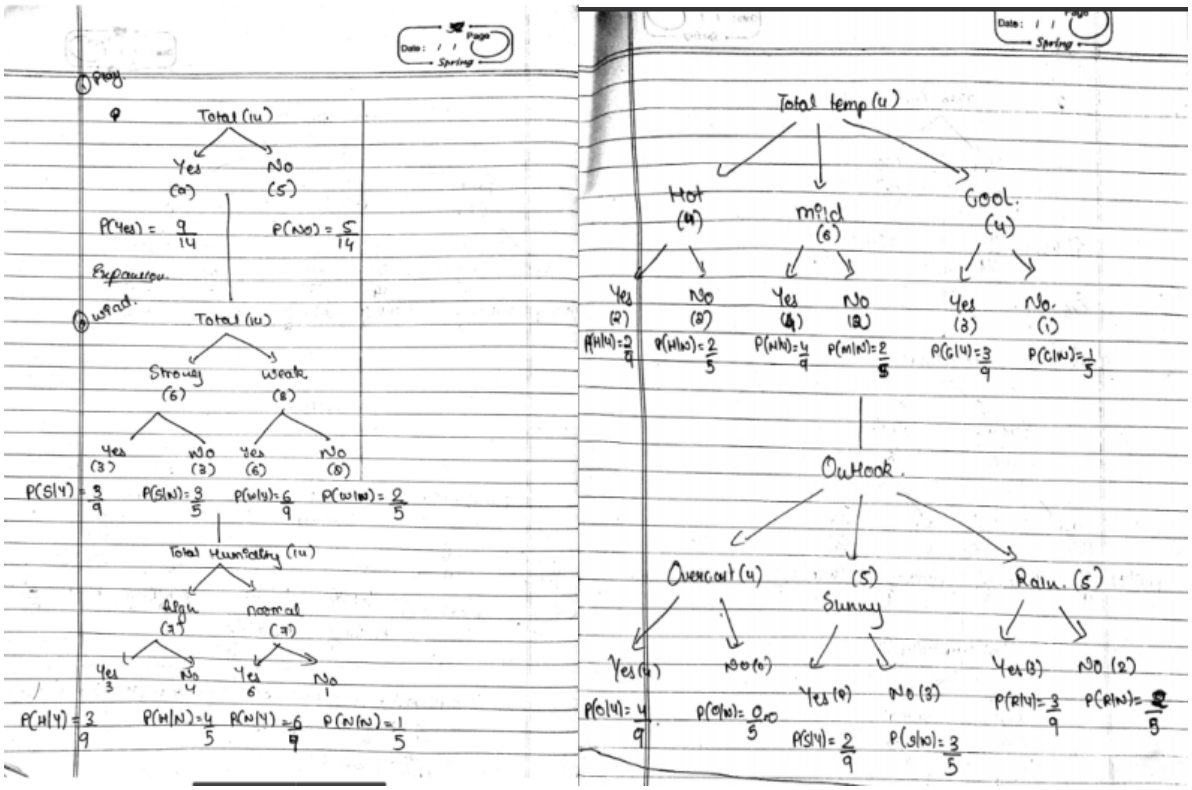
Probab. of A given B

Probab of B such that A is true.

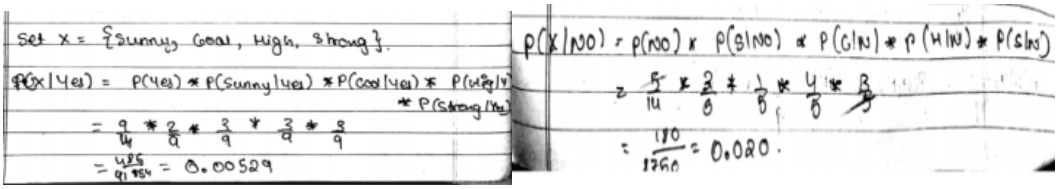
Question:


(iv) find the probab to play cricket on 15th day where
temp = cool , humicility = high , wind = strong,
outlook = sunny play = ?

Solution:


Play
Total (14)
Yes (9)
No (5)
P(Yes) = 9/14
P(No) = 5/14
Expansion.
wind.
Total (14)
Strong (6)
Weak (8)
Yes (3)
No (3)
Yes (6)
No (2)
P(S|Y) = 3/9
P(S|N) = 3/5
P(W|Y) = 6/9
P(W|N) = 2/5
Total Humidity (14)
High (7)
normal (7)
Yes 3
No 4
Yes 6
No 1
P(H|Y) = 3/9
P(H|N) = 4/5
P(N|Y) = 6/9
P(N|N) = 1/5
Total temp (14)
Hot (4)
mild (6)
Cool. (4)
Yes (2)
No (2)
Yes (4)
No (2)
Yes (3)
No. (1)
P(H|Y) = 2/9
P(H|N) = 2/5
P(M|Y) = 4/9
P(M|N) = 2/5
P(C|Y) = 3/9
P(C|N) = 1/5
Outlook.
Overcast (4)
(5)
Sunny
Rain. (5)
Yes (4)
No (0)
Yes (2)
No (3)
Yes (3)
No (2)
P(O|Y) = 4/9
P(O|N) = 0/5
P(S|Y) = 2/9
P(S|N) = 3/5
P(R|Y) = 3/9
P(R|N) = 2/5

Calculation:



Using Pandas:

```
from sklearn import preprocessing

#Generating the Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

# Assign features and encoding labels
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','S
'Rainy','Sunny','Overcast','Overcast','Rainy']
humidity=['High','High','High','Medium','Low','Low','Low','Medium','Low','Medium

batfirst=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Y

# Creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
hum_encoded=le.fit_transform(humidity)
label=le.fit_transform(batfirst)
print(weather_encoded,hum_encoded,label)


#Combining weather and humidity in a single tuple as features
features=list(zip(weather_encoded,hum_encoded))

#Create a Gaussian Classifier
model = GaussianNB()
model.fit(features,label) #Train the model using training set.

print("Enter Weather and Humidtity conditions : ")
w,h=map(int, input().split())

#Predict Output
predicted= model.predict([[w,h]]) # ''' For Weather : 0:Overcast, 2:Sunny , 1:Ra

print(predicted) # --> [1] that means yes, the player should bat first and [0] t
```

Amazon Data Set

AMAZON ALEXA REVIEW RATINGS ANALYSIS

In [1]:
```
# import libraries
import pandas as pd # Import Pandas for data manipulation using dataframes
import numpy as np # Import Numpy for data statistical analysis
import matplotlib.pyplot as plt # Import matplotlib for data visualisation
import seaborn as sns # Statistical data visualization
# %matplotlib inline
```

In [2]:
```
df_alexa = pd.read_csv('amazon_alexa.tsv', sep='\t')
```

In [3]:
```
df_alexa.head()
```

Out[3]:

	rating	date	variation	verified_reviews	feedback
0	5	31-Jul-18	Charcoal Fabric	Love my Echo!	1
1	5	31-Jul-18	Charcoal Fabric	Loved it!	1
2	4	31-Jul-18	Walnut Finish	Sometimes while playing a game, you can answer...	1
3	5	31-Jul-18	Charcoal Fabric	I have had a lot of fun with this thing. My 4 ...	1
4	5	31-Jul-18	Charcoal Fabric	Music	1

In [4]:
```
df_alexa.keys()
```

Out[4]:
```
Index(['rating', 'date', 'variation', 'verified_reviews', 'feedback'], dtype='o
bject')
```

In [5]:
```
df_alexa.tail()
```

Out[5]:

	rating	date	variation	verified_reviews	feedback
3145	5	30-Jul-18	Black Dot	Perfect for kids, adults and everyone in betwe...	1
3146	5	30-Jul-18	Black Dot	Listening to music, searching locations, check...	1
3147	5	30-Jul-18	Black Dot	I do love these things, i have them running my...	1
3148	5	30-Jul-18	White Dot	Only complaint I have is that the sound qualit...	1
3149	4	29-Jul-18	Black Dot	Good	1

```
In [6]: df_alexa['verified_reviews']
```

```
Out[6]: 0                                          Love my Echo!
        1                                               Loved it!
        2       Sometimes while playing a game, you can answer...
        3       I have had a lot of fun with this thing. My 4 ...
        4                                                   Music
        5       I received the echo as a gift. I needed anothe...
        6       Without having a cellphone, I cannot use many ...
        7       I think this is the 5th one I've purchased. I'...
        8                                             looks great
        9       Love it! I've listened to songs I haven't hear...
        10      I sent it to my 85 year old Dad, and he talks ...
        11      I love it! Learning knew things with it eveyda...
        12      I purchased this for my mother who is having k...
        13                                    Love, Love, Love!!
        14                              Just what I expected....
        15                             I love it, wife hates it.
        16      Really happy with this purchase.  Great speake...
        17      We have only been using Alexa for a couple of ...
        18      We love the size of the 2nd generation echo. S...
        19      I liked the original Echo. This is the same bu...
        20      Love the Echo and how good the music sounds pl...
        21      We love Alexa! We use her to play music, play ...
        22      Have only had it set up for a few days. Still ...
        23      I love it. It plays my sleep sounds immediatel...
        24      I got a second unit for the bedroom, I was exp...
        25                                        Amazing product
        26      I love my Echo. It's easy to operate, loads of...
        27                              Sounds great!! Love them!
        28      Fun item to play with and get used to using.  ...
        29                                Just like the other one
                                       ...
        3120
        3121    I like the hands free operation vs the Tap. We...
        3122    I dislike that it confuses my requests all the...
        3123
        3124    Love my Alexa! Actually have 3 throughout the ...
        3125    This product is easy to use and very entertain...
        3126
        3127    works great but speaker is not the good for mu...
        3128      Outstanding product - easy to use.  works great
        3129    We have six of these throughout our home and t...
```

VISUALIZING THE DATA

```
In [7]: positive = df_alexa[df_alexa['feedback']==1]
```

```
In [8]: negative = df_alexa[df_alexa['feedback']==0]
```

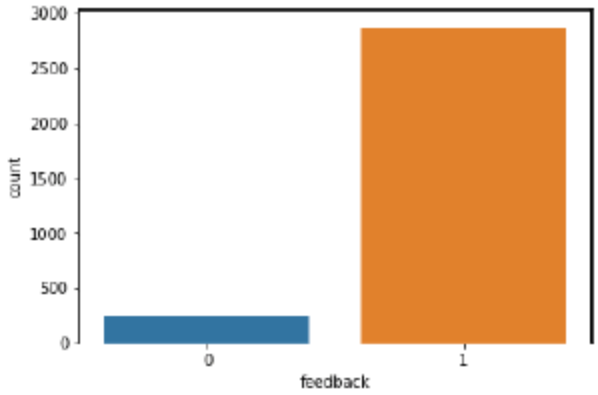
```
In [9]: negative
```

Out[9]:

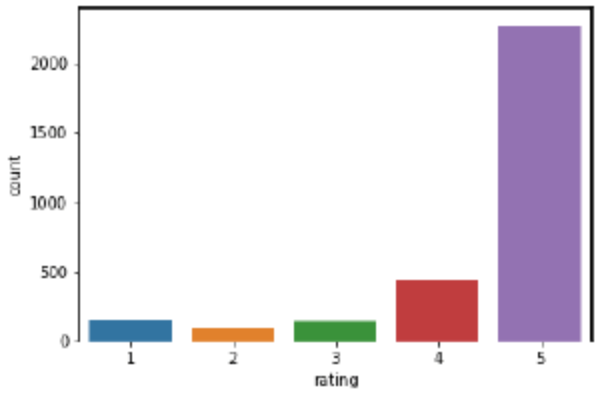
	rating	date	variation	verified_reviews	feedback
46	2	30-Jul-18	Charcoal Fabric	It's like Siri, in fact, Siri answers more acc...	0
111	2	30-Jul-18	Charcoal Fabric	Sound is terrible if u want good music too get...	0
141	1	30-Jul-18	Charcoal Fabric	Not much features.	0
162	1	30-Jul-18	Sandstone Fabric	Stopped working after 2 weeks ,didn't follow c...	0
176	2	30-Jul-18	Heather Gray Fabric	Sad joke. Worthless.	0
187	2	29-Jul-18	Charcoal Fabric	Really disappointed Alexa has to be plug-in to...	0
205	2	29-Jul-18	Sandstone Fabric	It's got great sound and bass but it doesn't w...	0
233	2	29-Jul-18	Sandstone Fabric	I am not super impressed with Alexa. When my P...	0
299	2	29-Jul-18	Charcoal Fabric	Too difficult to set up. It keeps timing out ...	0
341	1	28-Jul-18	Charcoal Fabric	Alexa hardly came on..	0
350	1	31-Jul-18	Black	Item no longer works after just 5 months of us...	0
361	1	29-Jul-18	Black	This thing barely works. You have to select 3r...	0
368	1	28-Jul-18	Black	I returned 2 Echo Dots & am only getting refun...	0
369	1	28-Jul-18	Black	not working	0
373	1	27-Jul-18	Black	I'm an Echo fan but this one did not work	0
374	1	26-Jul-18	Black		0
376	2	26-Jul-18	Black	Doesn't always respond when spoken to with pro...	0
381	1	25-Jul-18	White	It worked for a month or so then it stopped. I...	0
382	2	25-Jul-18	Black	Poor quality. Gave it away.	0
388	1	24-Jul-18	Black	Never could get it to work. A techie friend lo...	0
394	2	22-Jul-18	White	Initially, this echo dot worked very well. Ove...	0
396	1	20-Jul-18	Black	I bought an echo dot that had been refurbished...	0

…. So on

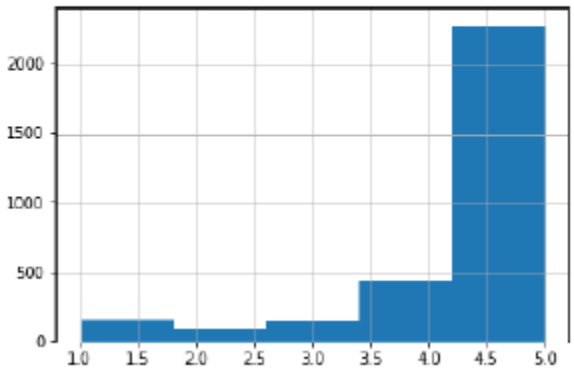
```
In [10]: sns.countplot(df_alexa['feedback'], label = "Count")
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x201427a4128>
```




```
In [11]: sns.countplot(x = 'rating', data = df_alexa)
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x20142773d30>
```

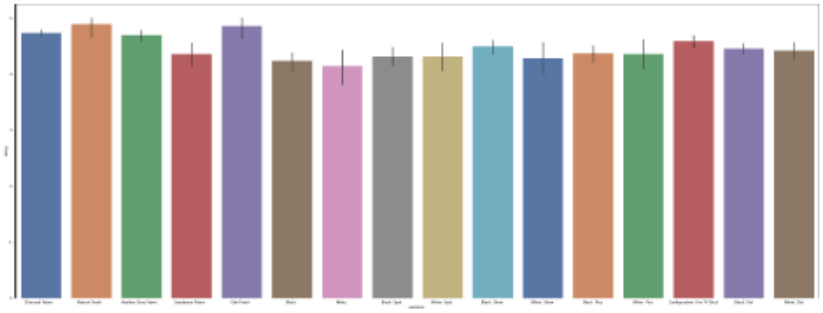



```
In [12]: df_alexa['rating'].hist(bins = 5)
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x20142965710>
```



In [13]:
```
plt.figure(figsize = (40,15))
sns.barplot(x = 'variation', y='rating', data=df_alexa, palette = 'deep')
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x201428d7390>



In [120]:
```
words_as_one_string =" ".join(words)
```

In [121]:
```
words_as_one_string
```

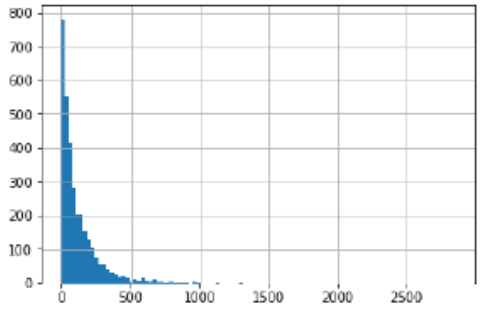
Out[121]:
```
'Love my Echo! Loved it! Sometimes while playing a game, you can answer a que
stion correctly but Alexa says you got it wrong and answers the same as you.
I like being able to turn lights on and off while away from home. I have had
a lot of fun with this thing. My 4 yr old learns about dinosaurs, i control t
he lights and play games like categories. Has nice sound when playing music a
s well. Music I received the echo as a gift. I needed another Bluetooth or so
mething to play music easily accessible, and found this smart speaker. Can't
wait to see what else it can do. Without having a cellphone, I cannot use man
y of her features. I have an iPad but do not see that of any use.  It IS a gr
eat alarm.  If u r almost deaf, you can hear her alarm in the bedroom from ou
t in the living room, so that is reason enough to keep her.It is fun to ask r
andom questions to hear her response.  She does not seem to be very smartbon
politics yet. I think this is the 5th one I\'ve purchased. I\'m working on ge
tting one in every room of my house. I really like what features they offer s
pecifily playing music on all Echos and controlling the lights throughout my
house. looks great Love it! I've listened to songs I haven't heard since chil
dhood! I get the news, weather, information! It's great! I sent it to my 85 y
ear old Dad, and he talks to it constantly. I love it! Learning knew things w
ith it eveyday! Still figuring out how everything works but so far it\'s been
```

In [122]:
```
len(words_as_one_string)
```

Out[122]: 419105

In [77]:
```
df_alexa['length'].hist(bins=100)
```

Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x2012601ce80>



In [87]:
```
min_char = df_alexa['length'].min()
df_alexa[df_alexa['length'] == min_char]['verified_reviews'].iloc[0]
```

Out[87]: '😍'

PRACTICAL - 8

AIM: Perform the linear regression in python by takingthe data set of employees of any company. Data set should contain Employee ID, Name, Salary, Department, Age, gender etc.

PROGRAM CODE AND OUTPUT:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [3]:
```
# Importing the dataset
dataset = pd.read_csv('Desktop\salary_data.csv')
X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column
y = dataset.iloc[:, 1].values #get array of dataset in column 1st
```

In [4]:
```
# 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=1/3, random_state=0)
```

In [5]:
```
# Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [6]:
```
#Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[6]:
```
LinearRegression()
```

In [7]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

In [9]:
```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

In [8]:
```
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
```

PRATICAL - 9

AIM: Implement the following clustering algorithms using WEKA Tool

1. K-means
2. Agglomerative
3. Divisive.

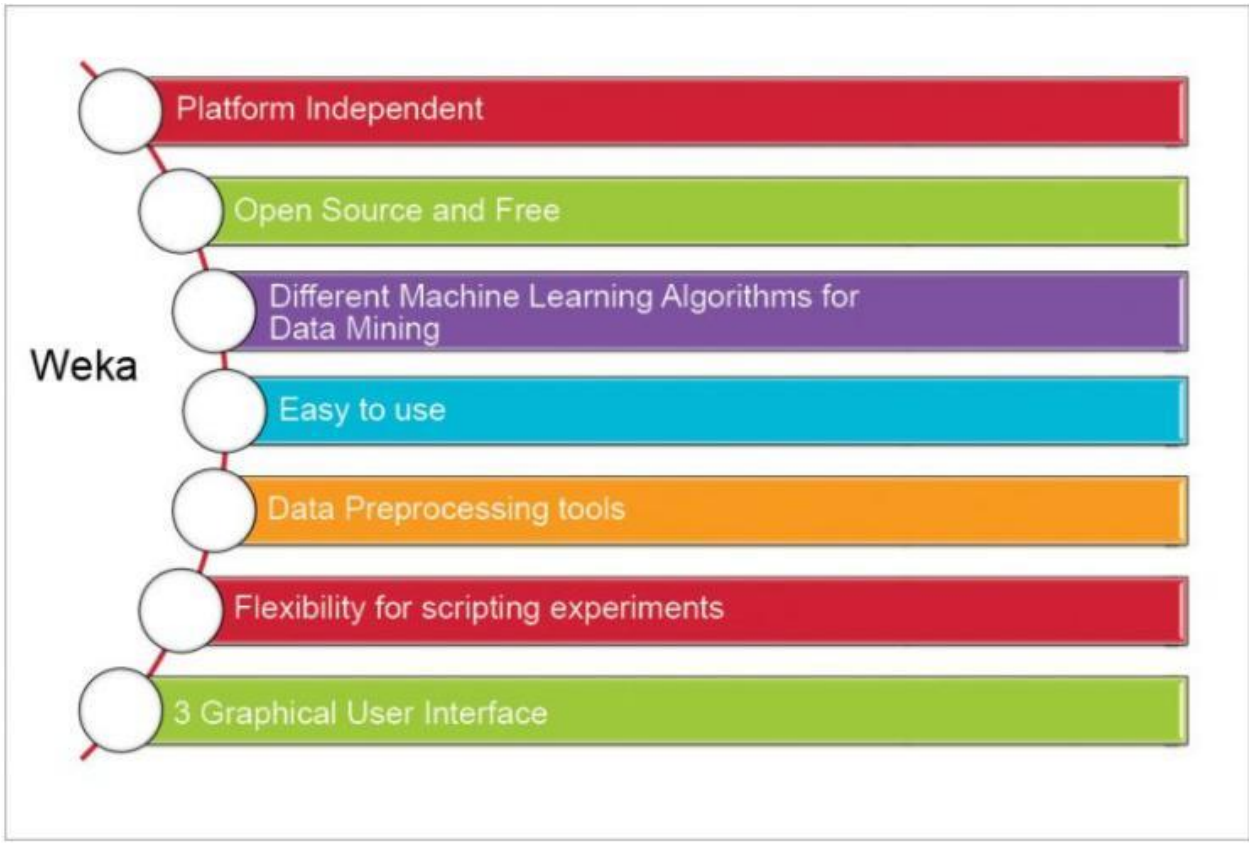
THEORY:

Weka is a collection of **machine learning** algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. **Weka** contains **tools** for data pre-processing, classification, regression, clustering, association rules, and visualization.

Weka is a collection of tools for:

- Regression
- Clustering
- Association
- Data pre-processing
- Classification
- Visualization

The features of Weka are shown:



Kmeans Algorithm

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

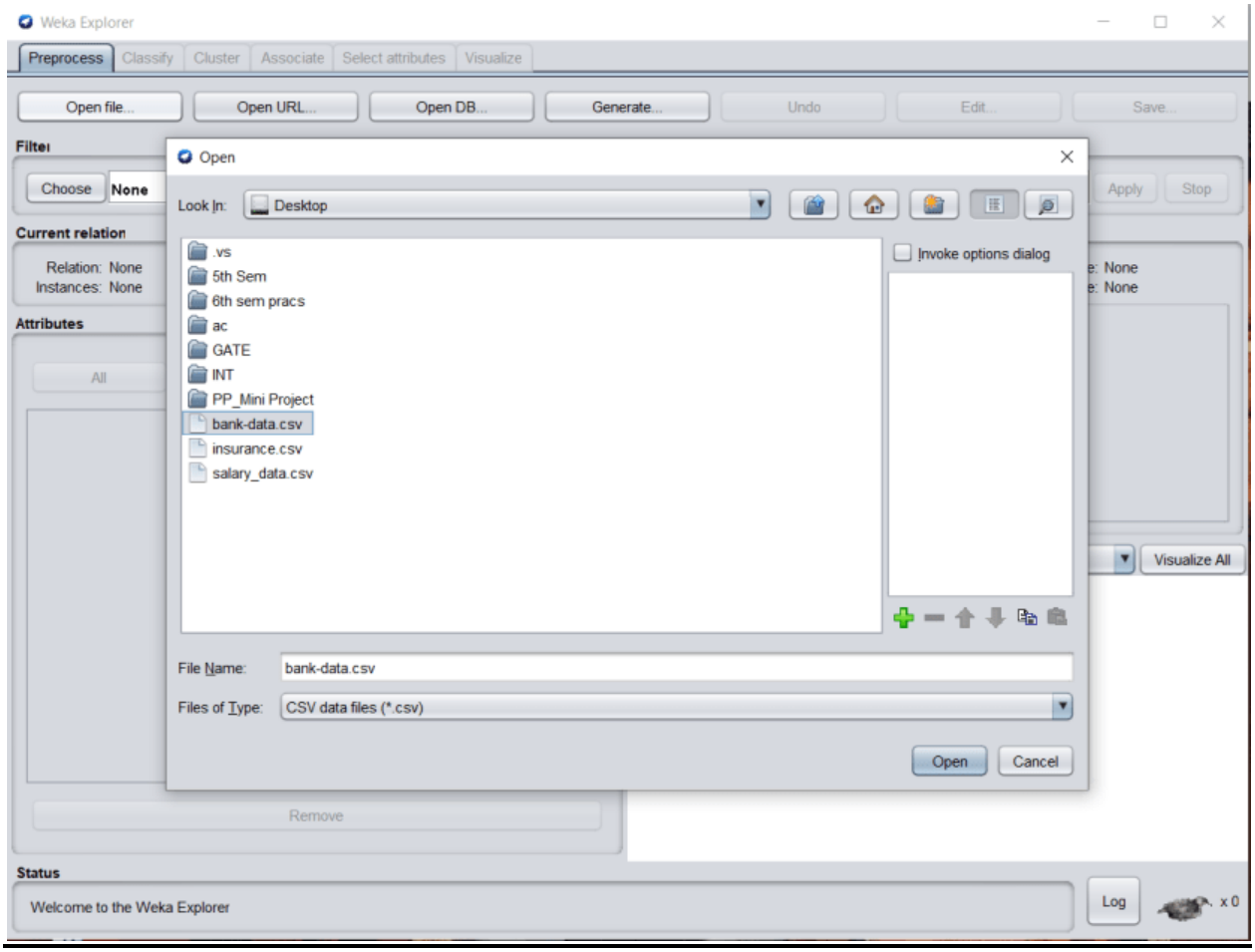
The way kmeans algorithm works is as follows:

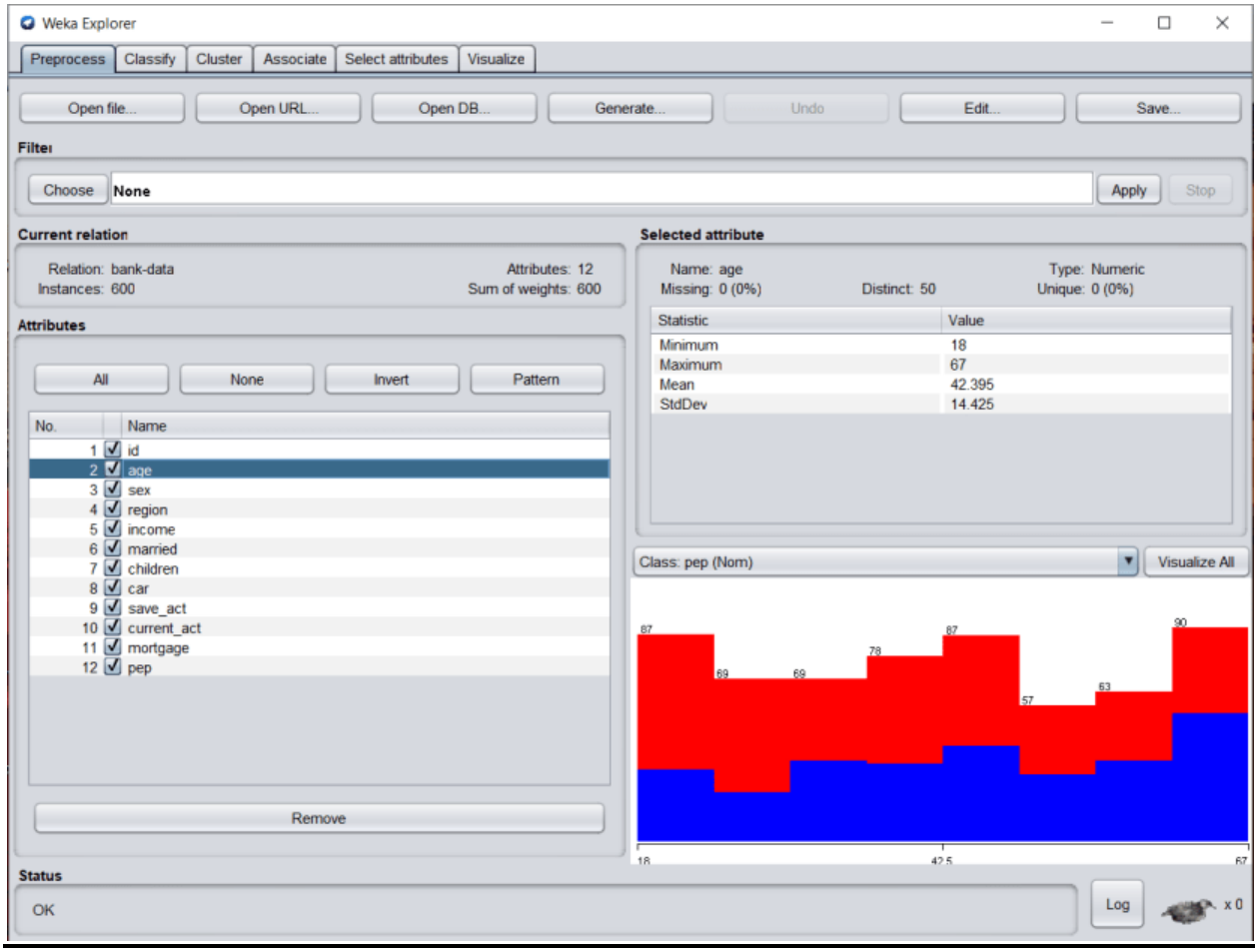
1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

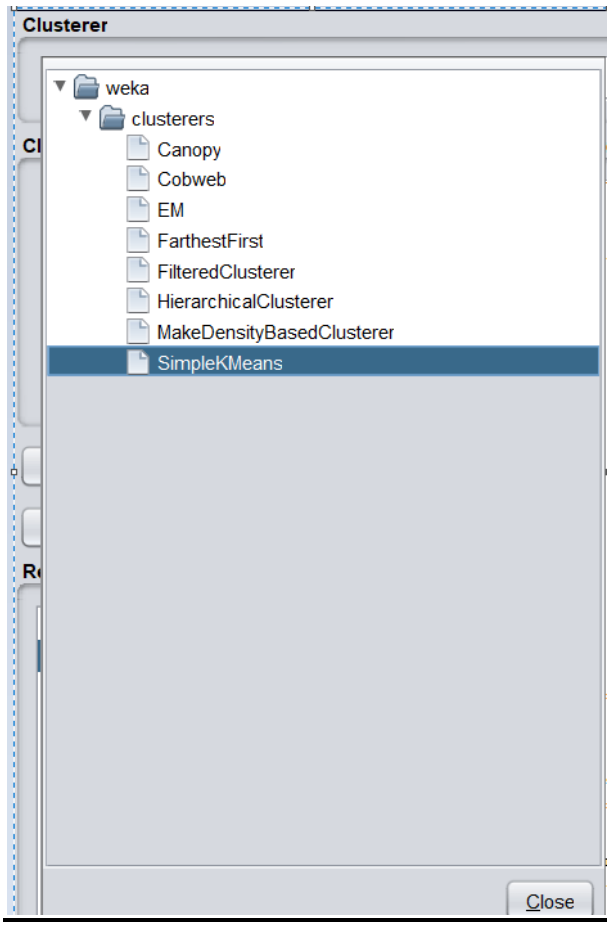
- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).
- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

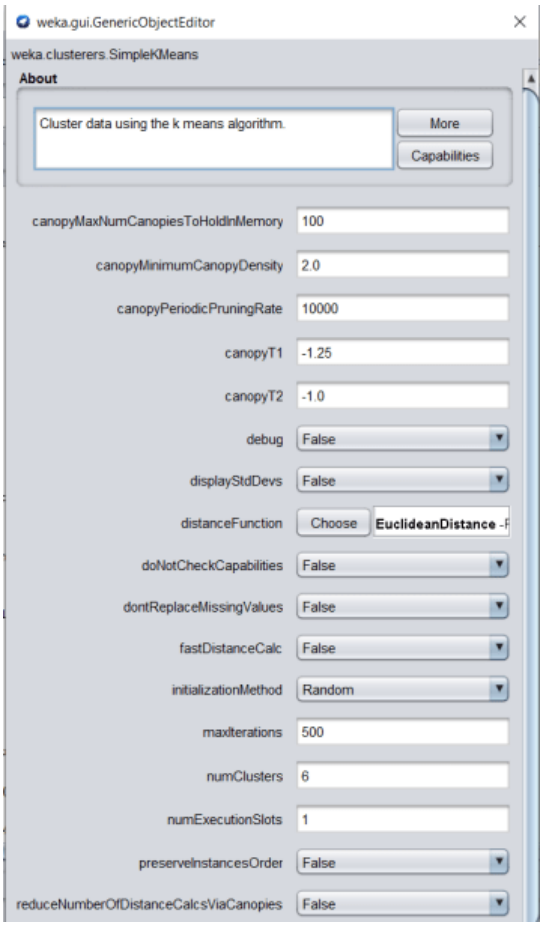
IMPLEMNTATION:

Importing dataset



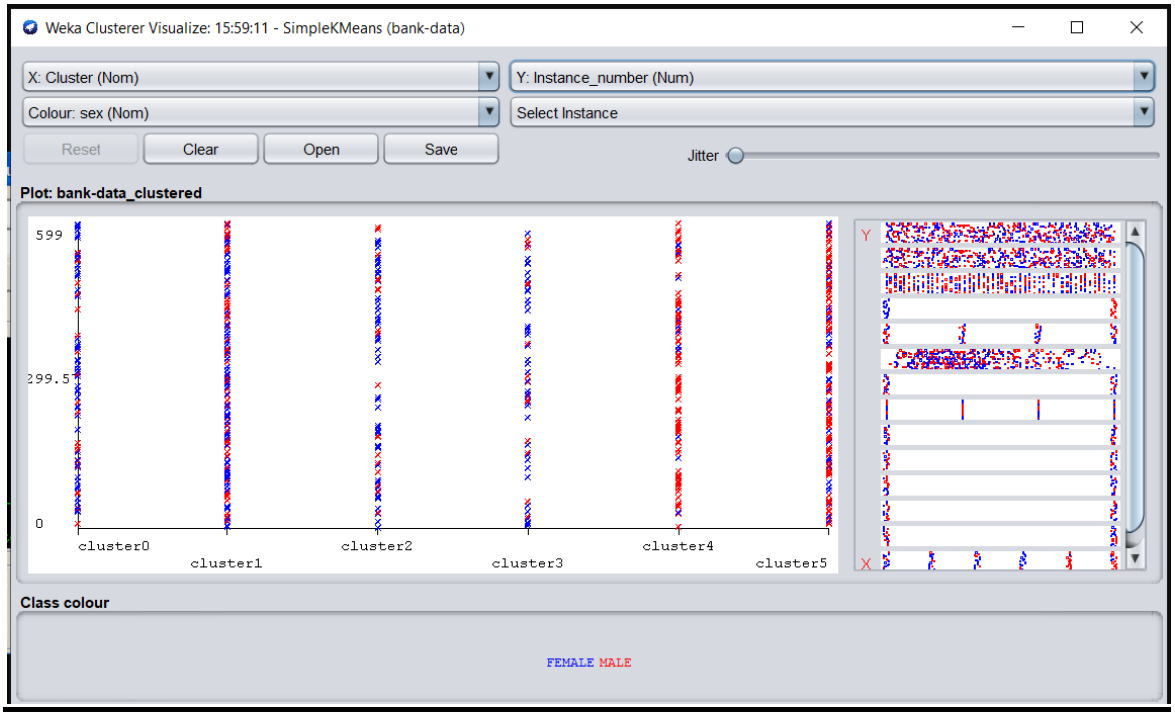

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: bank-data
Instances: 600
Attributes: 12
Sum of weights: 600
Attributes
All
None
Invert
Pattern
No.
Name
1 id
2 age
3 sex
4 region
5 income
6 married
7 children
8 car
9 save_act
10 current_act
11 mortgage
12 pep
Remove
Selected attribute
Name: age
Missing: 0 (0%)
Distinct: 50
Type: Numeric
Unique: 0 (0%)
Statistic
Value
Minimum 18
Maximum 67
Mean 42.395
StdDev 14.425
Class: pep (Nom)
Visualize All
87
69
69
78
87
57
63
90
18
42.5
67
Status
OK
Log
x 0

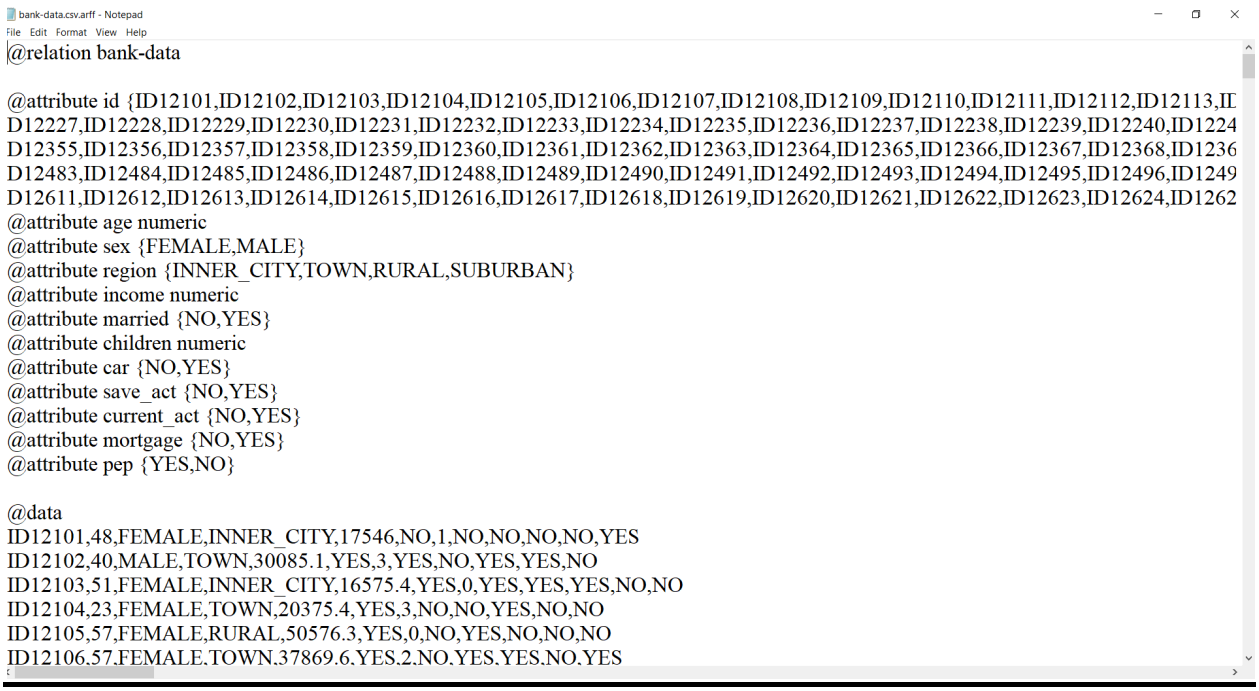

Clusterer
weka
clusterers
Canopy
Cobweb
EM
FarthestFirst
FilteredClusterer
HierarchicalClusterer
MakeDensityBasedClusterer
SimpleKMeans
Close


weka.gui.GenericObjectEditor
weka.clusterers.SimpleKMeans
About
Cluster data using the k means algorithm.
More
Capabilities
canopyMaxNumCanopiesToHoldInMemory 100
canopyMinimumCanopyDensity 2.0
canopyPeriodicPruningRate 10000
canopyT1 -1.25
canopyT2 -1.0
debug False
displayStdDevs False
distanceFunction Choose EuclideanDistance -
doNotCheckCapabilities False
dontReplaceMissingValues False
fastDistanceCalc False
initializationMethod Random
maxIterations 500
numClusters 6
numExecutionSlots 1
preserveInstancesOrder False
reduceNumberOfDistanceCalcsViaCanopies False

15:54:19 - SimpleKMeans

```
kMeans
======

Number of iterations: 18
Within cluster sum of squared errors: 1955.4146634784236

Initial starting points (random):

Cluster 0: ID12614,25,FEMALE,RURAL,14505.3,NO,3,NO,YES,YES,NO,NO
Cluster 1: ID12131,61,FEMALE,RURAL,22942.9,YES,2,NO,YES,YES,NO,NO
Cluster 2: ID12190,54,FEMALE,INNER_CITY,31095.6,YES,2,NO,NO,YES,NO,YES
Cluster 3: ID12485,36,FEMALE,TOWN,26920.8,YES,0,NO,NO,YES,NO,NO
Cluster 4: ID12203,42,MALE,INNER_CITY,15499.9,YES,0,YES,NO,YES,YES,YES
Cluster 5: ID12597,50,MALE,TOWN,40972.9,NO,2,YES,YES,YES,YES,YES

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute      Full Data          0          1          2          3          4          5
                 (600.0)     (74.0)    (164.0)     (71.0)     (58.0)     (99.0)    (134.0)
=============================================================================================
id               ID12101    ID12107    ID12103    ID12101    ID12104    ID12102    ID12108
age               42.395    42.9324    43.7744    39.0282    37.3103     38.404    47.3433
sex               FEMALE     FEMALE     FEMALE     FEMALE     FEMALE       MALE       MALE
region        INNER_CITY      RURAL INNER_CITY INNER_CITY      TOWN INNER_CITY       TOWN
income       27524.0312 28838.7605 28586.4063 20463.1273 20600.8528  25720.037 33568.3929
married             YES         NO        YES        YES        YES        YES         NO
children         1.0117      1.973      0.628     0.6901     1.6207      0.899     0.9403
car                 NO         NO         NO         NO         NO        YES        YES
save_act           YES        YES        YES         NO         NO         NO        YES
current_act        YES        YES        YES        YES        YES        YES        YES
```



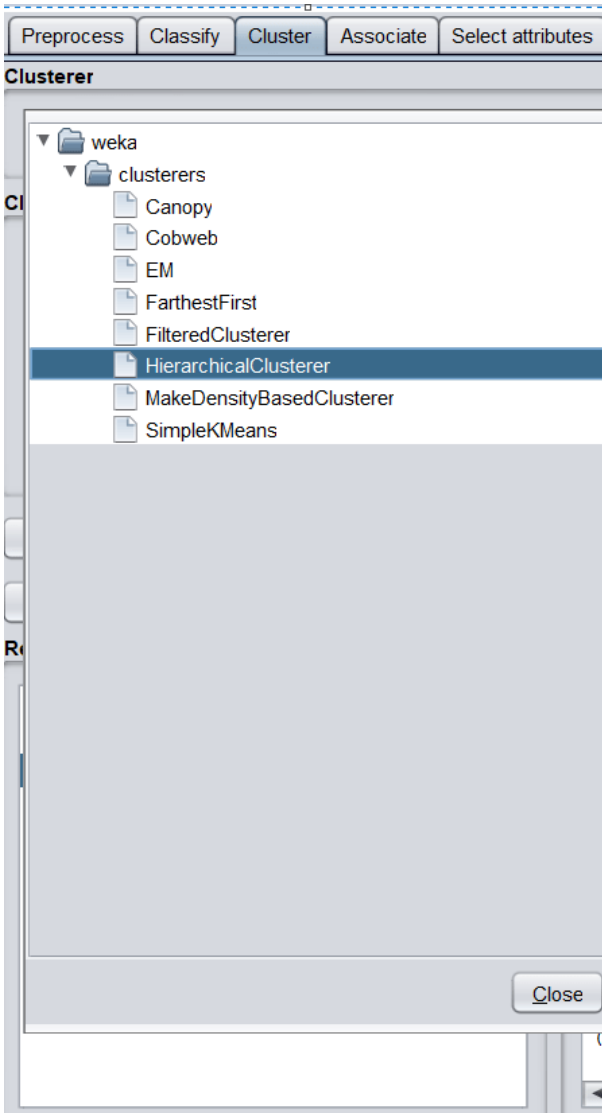


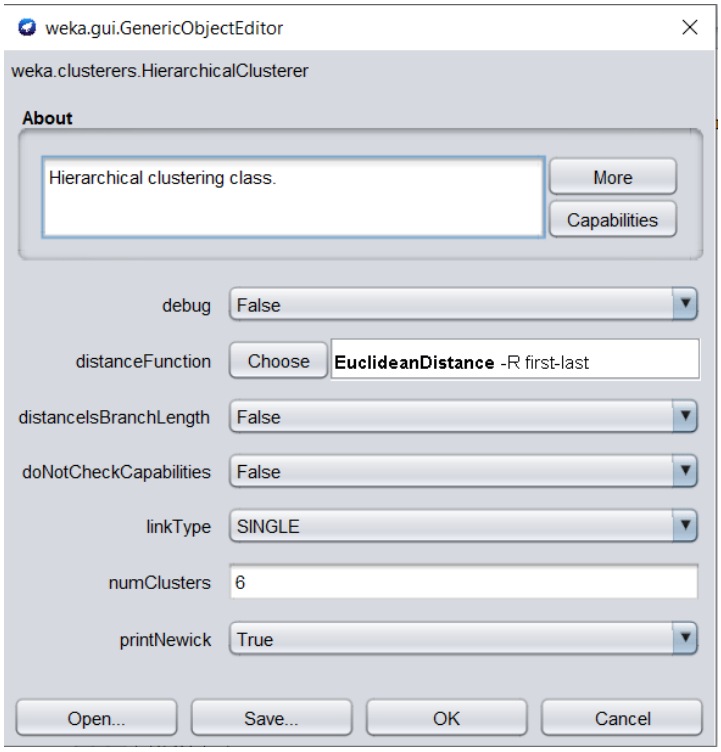
bank-data.csv.arff - Notepad

File Edit Format View Help

```
@relation bank-data

@attribute id {ID12101,ID12102,ID12103,ID12104,ID12105,ID12106,ID12107,ID12108,ID12109,ID12110,ID12111,ID12112,ID12113,ID
D12227,ID12228,ID12229,ID12230,ID12231,ID12232,ID12233,ID12234,ID12235,ID12236,ID12237,ID12238,ID12239,ID12240,ID1224
D12355,ID12356,ID12357,ID12358,ID12359,ID12360,ID12361,ID12362,ID12363,ID12364,ID12365,ID12366,ID12367,ID12368,ID1236
D12483,ID12484,ID12485,ID12486,ID12487,ID12488,ID12489,ID12490,ID12491,ID12492,ID12493,ID12494,ID12495,ID12496,ID1249
D12611,ID12612,ID12613,ID12614,ID12615,ID12616,ID12617,ID12618,ID12619,ID12620,ID12621,ID12622,ID12623,ID12624,ID1262
@attribute age numeric
@attribute sex {FEMALE,MALE}
@attribute region {INNER_CITY,TOWN,RURAL,SUBURBAN}
@attribute income numeric
@attribute married {NO,YES}
@attribute children numeric
@attribute car {NO,YES}
@attribute save_act {NO,YES}
@attribute current_act {NO,YES}
@attribute mortgage {NO,YES}
@attribute pep {YES,NO}

@data
ID12101,48,FEMALE,INNER_CITY,17546,NO,1,NO,NO,NO,NO,YES
ID12102,40,MALE,TOWN,30085.1,YES,3,YES,NO,YES,YES,NO
ID12103,51,FEMALE,INNER_CITY,16575.4,YES,0,YES,YES,YES,NO,NO
ID12104,23,FEMALE,TOWN,20375.4,YES,3,NO,NO,YES,NO,NO
ID12105,57,FEMALE,RURAL,50576.3,YES,0,NO,YES,NO,NO,NO
ID12106,57,FEMALE,TOWN,37869.6,YES,2,NO,YES,YES,NO,YES
```

Hierarchal Clustering: Two types Agglomerative and Divisive





16:44:08 - HierarchicalClusterer

```
              car
              save_act
              current_act
              mortgage
              pep
Test mode:    evaluate on training data


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

Cluster 0
((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((0.0:1.17447,0.0:1.17447):0.2

Cluster 1
(1.0:1.4863,1.0:1.4863)



Time taken to build model (full training data) : 1.56 seconds

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

Clustered Instances

0      594 ( 99%)
1        2 (  0%)
2        1 (  0%)
3        1 (  0%)
4        1 (  0%)
5        1 (  0%)
```

PRACTICAL - 10

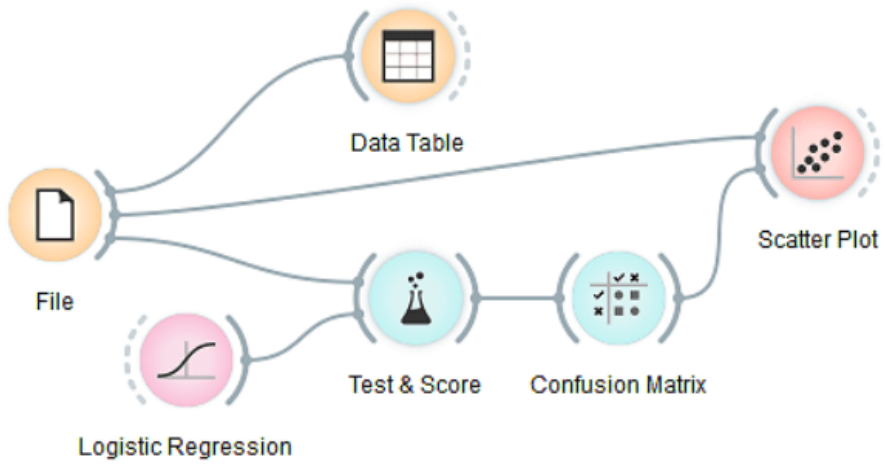
AIM: Perform the Data Preprocessing, Data Visualization, Model, Evaluate, Unsupervised learning and Text mining modules using Orange Tool.

THEORY:

Orange is a great data mining tool for beginners as well as for expert data scientists. Thanks to its user interface users can focus on data analysis instead on laborious coding, making a construction of complex data analytics pipelines simple.

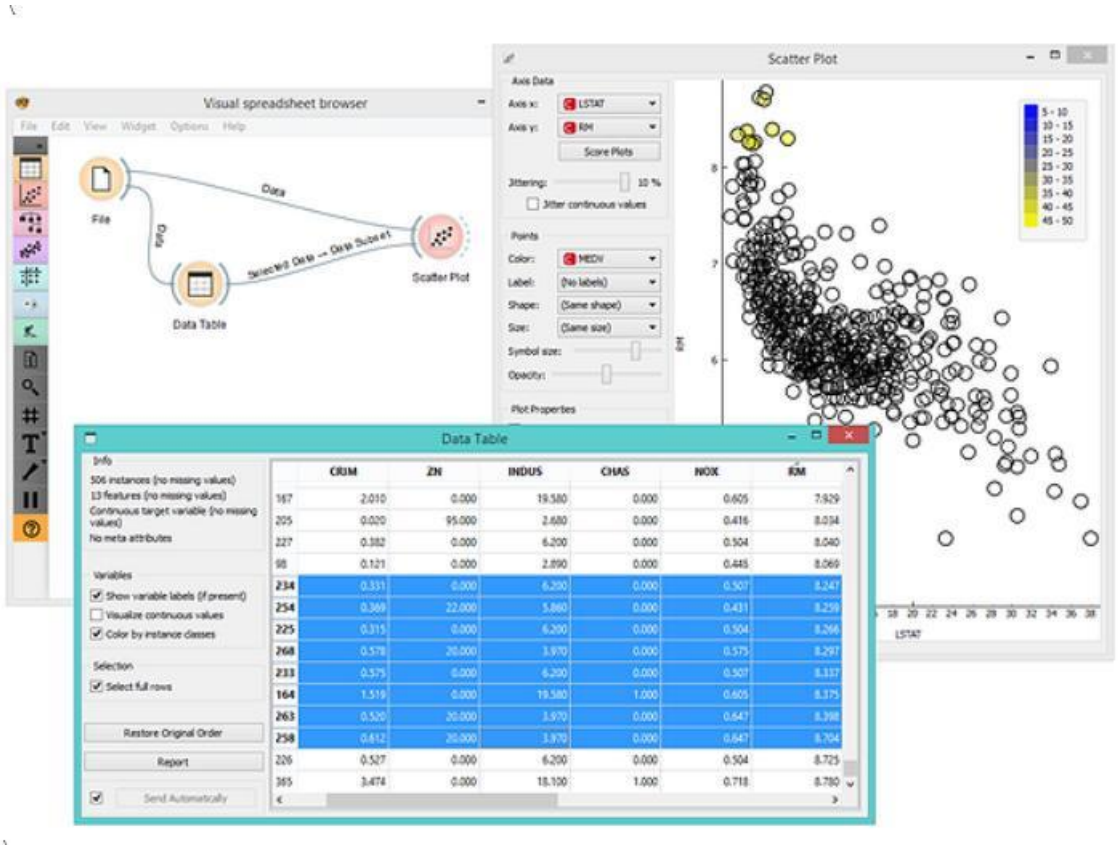
Component-Based Data Mining

In Orange, data analysis is done by stacking components into workflows. Each component, called a widget, embeds some data retrieval, preprocessing, visualization, modeling or evaluation task. Combining different widgets in a workflow enables you to build comprehensive data analysis schemas as you go. With a large library of widgets you won't be short for choice. Additional widgets are available through add-ons and allow for a more focused and topic-oriented research.



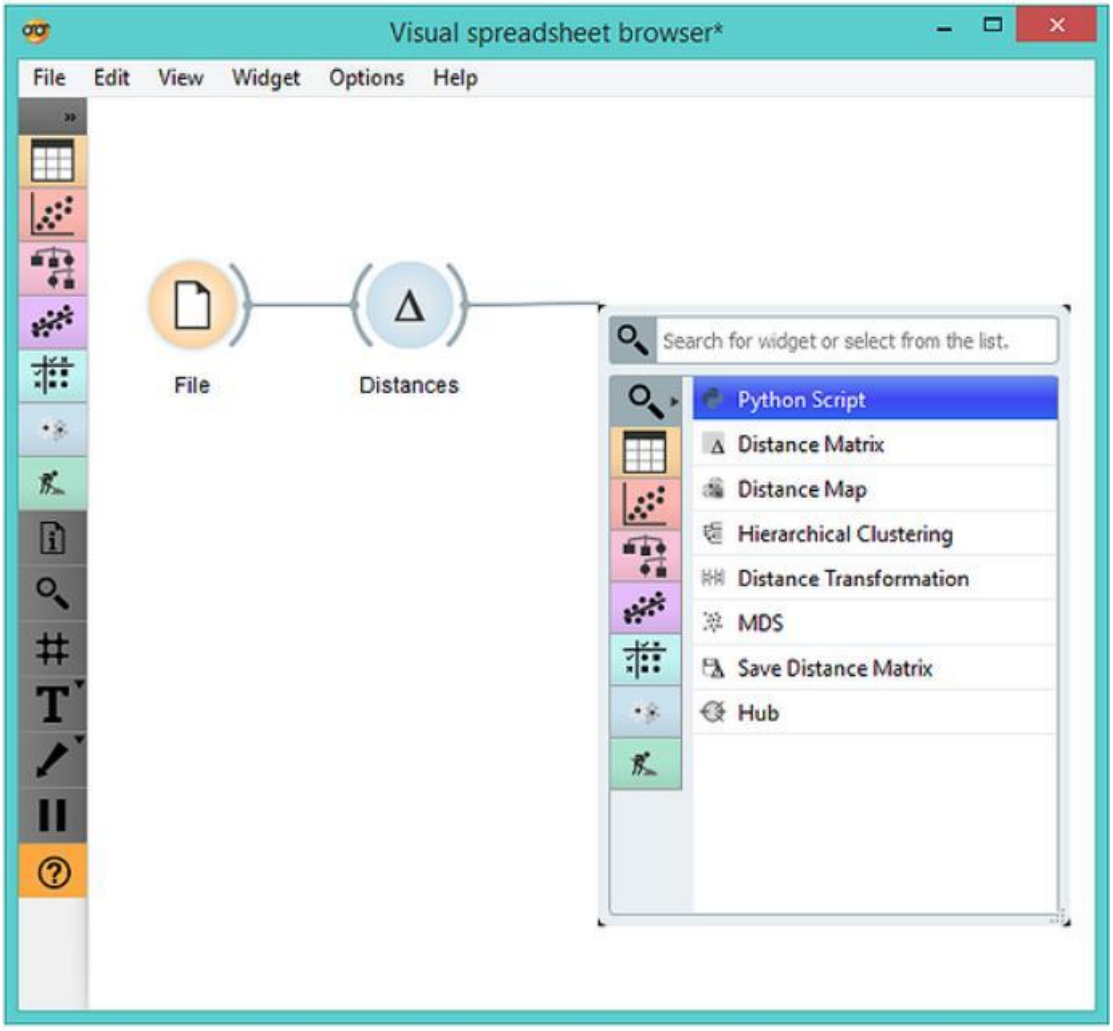
Interactive Data Exploration

Orange widgets communicate with each other. They receive data on the input and send out filtered or processed data, models, or anything the widget does on the output. Say, start with a File widget that reads the data and connect its output to another widget, say, a Data Table, and you have a functioning workflow. Alter any change in one widget, the changes are instantaneously propagated through the downstream workflow. Changing a data file in the File widget will trigger the response in all downstream widgets. This is especially fun if the widgets are open and when you can immediately see the results of any changes in that data, parameters of the methods or selections in interactive visualizations. For example, in a simple workflow below, where selection of the data in the spreadsheet propagates to a scatter plot, which marks the selected data instances.

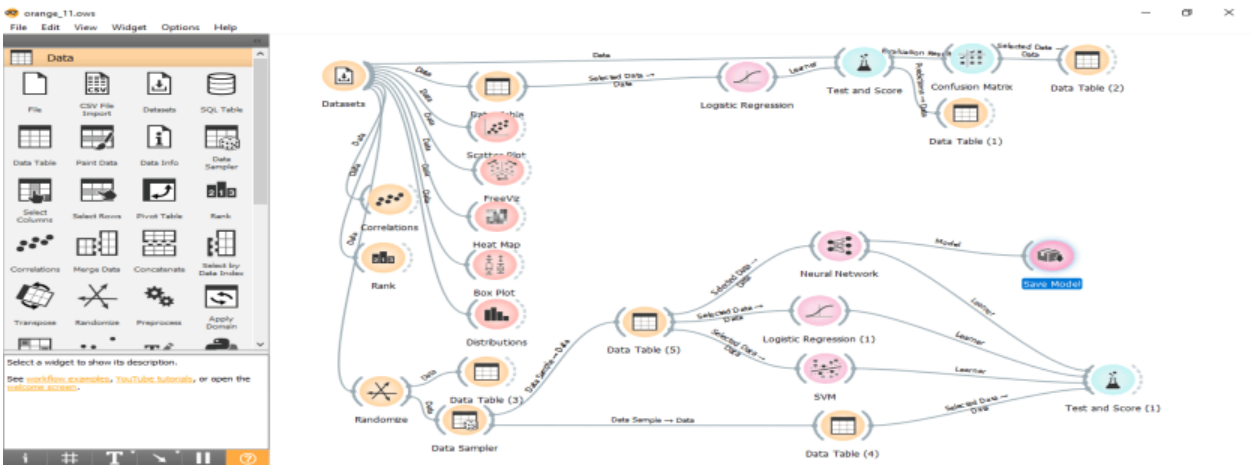


Clever Workflow Design Interface

Orange is easy to use even for complete novices. Start with the File widget and Orange will automatically suggest the next widgets that can be connected to it. For example, Orange knows you are likely to want Hierarchical Clustering after you've set up your Distances widget. All other defaults in the widgets are also set in a way that enables a simple analysis even without knowing a whole lot about statistics, machine learning, or exploratory data mining in general.



OUTPUT:



Data Table

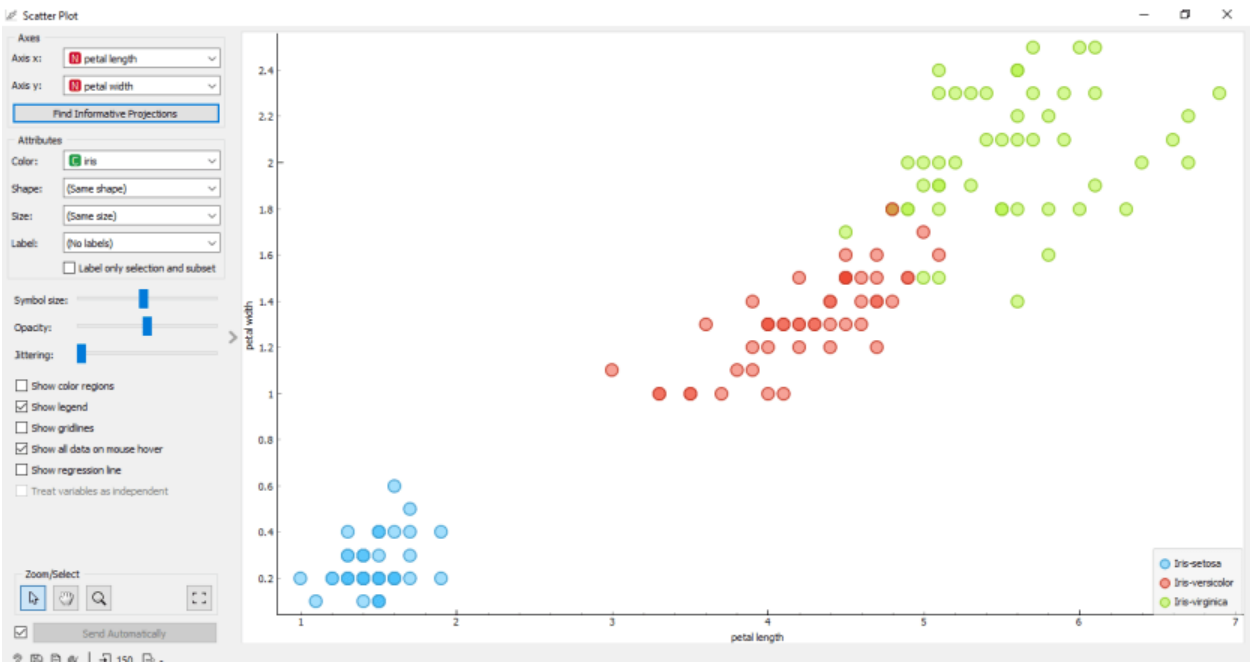
Info
150 instances (no missing data)
4 features
Target with 3 values
No meta attributes

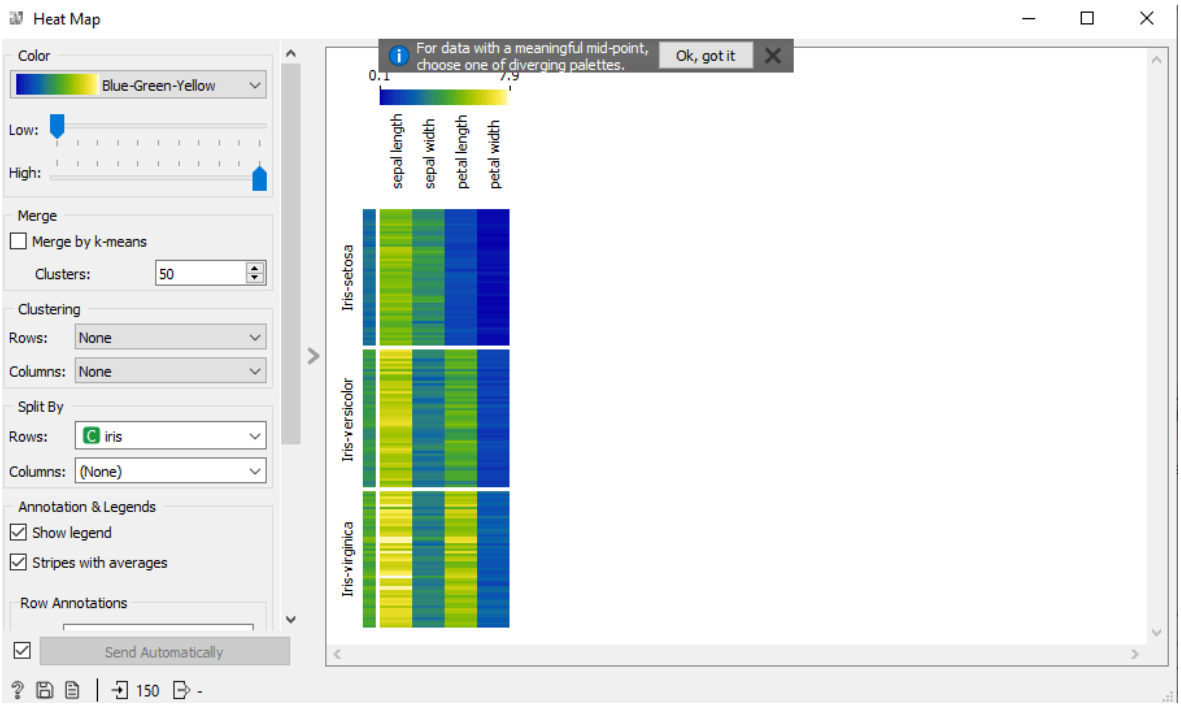
Variables
- Show variable labels (if present)
- Visualize numeric values
- Color by instance classes

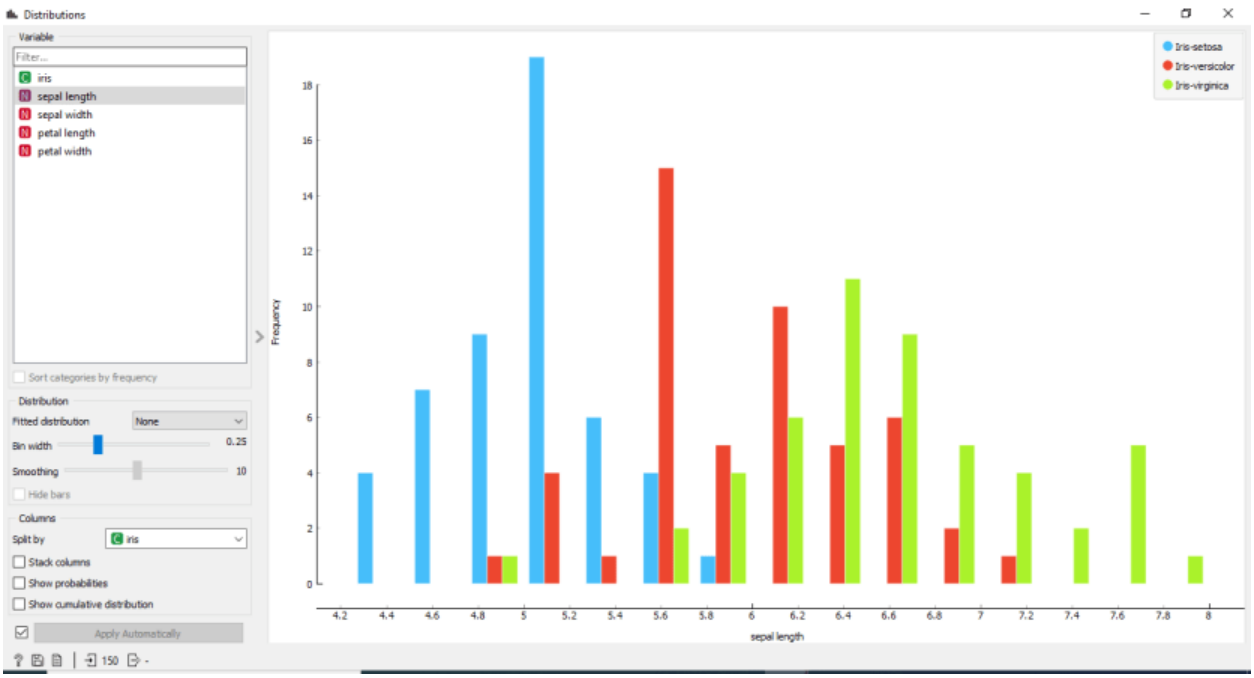
Selection
- Select full rows

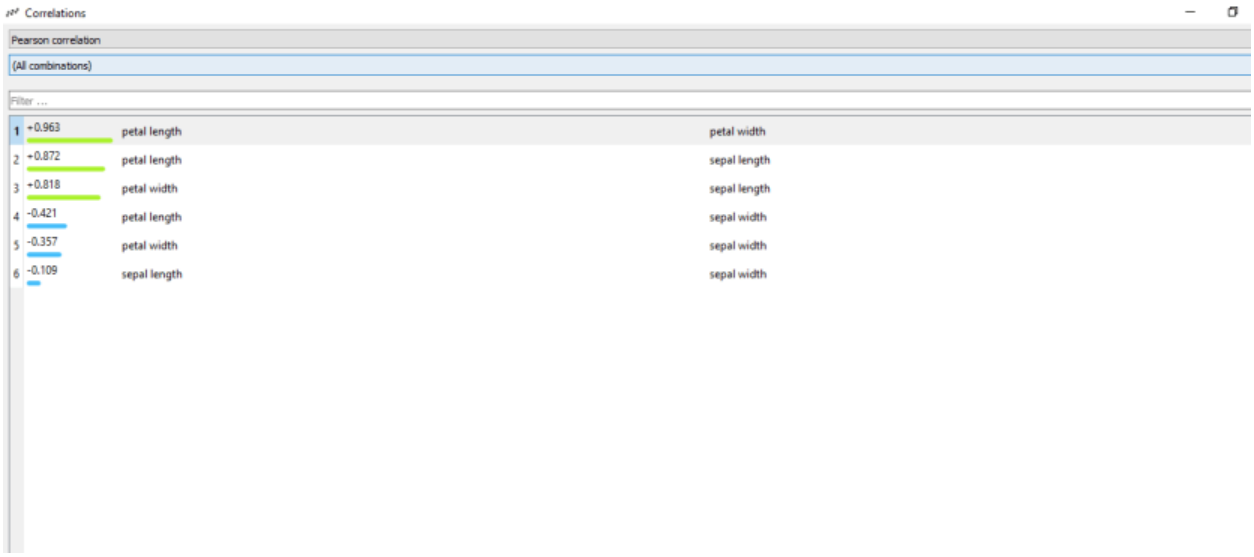
Restore Original Order
Send Automatically

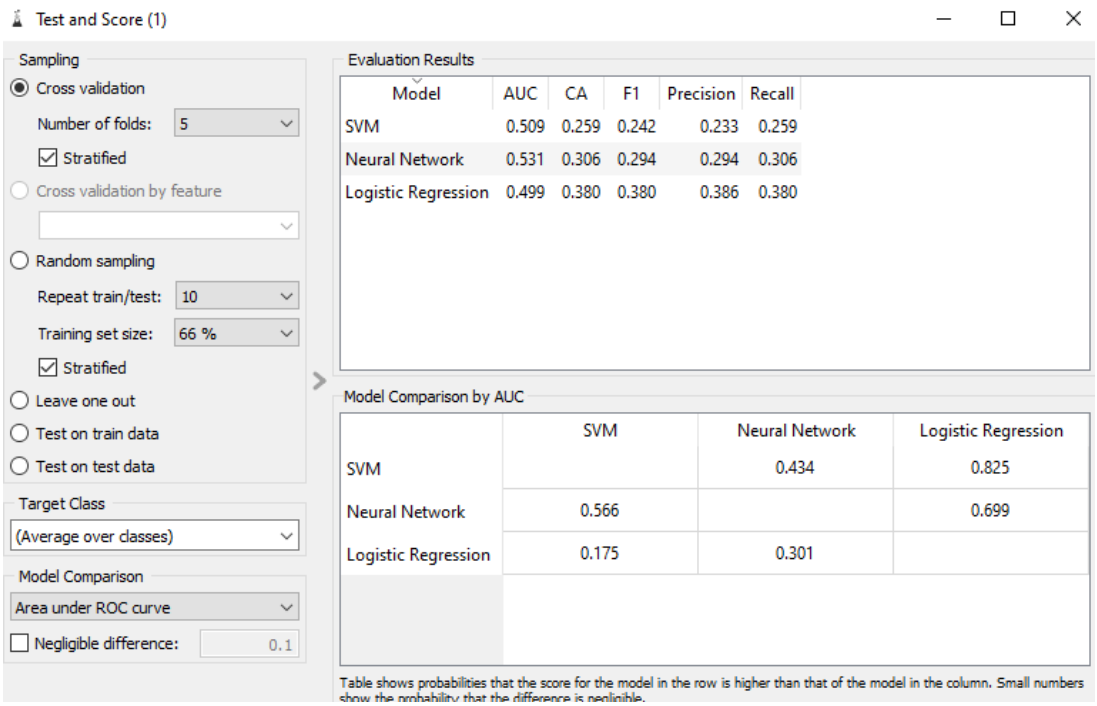
	iris	sepal length	sepal width	petal length	petal width
1	Iris-setosa	5.1	3.5	1.4	0.2
2	Iris-setosa	4.9	3.0	1.4	0.2
3	Iris-setosa	4.7	3.2	1.3	0.2
4	Iris-setosa	4.6	3.1	1.5	0.2
5	Iris-setosa	5.0	3.6	1.4	0.2
6	Iris-setosa	5.4	3.9	1.7	0.4
7	Iris-setosa	4.6	3.4	1.4	0.3
8	Iris-setosa	5.0	3.4	1.5	0.2
9	Iris-setosa	4.4	2.9	1.4	0.2
10	Iris-setosa	4.9	3.1	1.5	0.1
11	Iris-setosa	5.4	3.7	1.5	0.2
12	Iris-setosa	4.8	3.4	1.6	0.2
13	Iris-setosa	4.8	3.0	1.4	0.1
14	Iris-setosa	4.3	3.0	1.1	0.1
15	Iris-setosa	5.8	4.0	1.2	0.2
16	Iris-setosa	5.7	4.4	1.5	0.4
17	Iris-setosa	5.4	3.9	1.3	0.4
18	Iris-setosa	5.1	3.5	1.4	0.3
19	Iris-setosa	5.7	3.8	1.7	0.3
20	Iris-setosa	5.1	3.8	1.5	0.3
21	Iris-setosa	5.4	3.4	1.7	0.2
22	Iris-setosa	5.1	3.7	1.5	0.4
23	Iris-setosa	4.6	3.6	1.0	0.2
24	Iris-setosa	5.1	3.3	1.7	0.5
25	Iris-setosa	4.8	3.4	1.9	0.2
26	Iris-setosa	5.0	3.0	1.6	0.2
27	Iris-setosa	5.0	3.4	1.6	0.4
28	Iris-setosa	5.2	3.5	1.5	0.2
29	Iris-setosa	5.2	3.4	1.4	0.2
30	Iris-setosa	4.7	3.2	1.6	0.2
31	Iris-setosa	4.8	3.1	1.6	0.2


Scatter Plot
Axes
Axis x: petal length
Axis y: petal width
Find Informative Projections
Attributes
Color: iris
Shape: (Same shape)
Size: (Same size)
Label: (No labels)
Label only selection and subset
Symbol size:
Opacity:
Jittering:
Show color regions
Show legend
Show gridlines
Show all data on mouse hover
Show regression line
Treat variables as independent
Zoom/Select
Send Automatically
petal width
petal length
Iris-setosa
Iris-versicolor
Iris-virginica


Heat Map
Color
Blue-Green-Yellow
Low:
High:
Merge
Merge by k-means
Clusters: 50
Clustering
Rows: None
Columns: None
Split By
Rows: iris
Columns: (None)
Annotation & Legends
Show legend
Stripes with averages
Row Annotations
Send Automatically
150
For data with a meaningful mid-point, choose one of diverging palettes.
Ok, got it
0.1
7.9
sepal length
sepal width
petal length
petal width
Iris-setosa
Iris-versicolor
Iris-virginica


Distributions
Variable
Filter...
iris
sepal length
sepal width
petal length
petal width
Sort categories by frequency
Distribution
Fitted distribution
None
Bin width
0.25
Smoothing
10
Hide bars
Columns
Split by
iris
Stack columns
Show probabilities
Show cumulative distribution
Apply Automatically
Iris-setosa
Iris-versicolor
Iris-virginica
Frequency
sepal length


Correlations
Pearson correlation
(All combinations)
Filter ...
1 +0.963 petal length petal width
2 +0.872 petal length sepal length
3 +0.818 petal width sepal length
4 -0.421 petal length sepal width
5 -0.357 petal width sepal width
6 -0.109 sepal length sepal width


Test and Score (1)
Sampling
Cross validation
Number of folds: 5
Stratified
Cross validation by feature
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Model Comparison
Area under ROC curve
Negligible difference: 0.1
Evaluation Results
Model AUC CA F1 Precision Recall
SVM 0.509 0.259 0.242 0.233 0.259
Neural Network 0.531 0.306 0.294 0.294 0.306
Logistic Regression 0.499 0.380 0.380 0.386 0.380
Model Comparison by AUC
SVM Neural Network Logistic Regression
SVM 0.434 0.825
Neural Network 0.566 0.699
Logistic Regression 0.175 0.301
Table shows probabilities that the score for the model in the row is higher than that of the model in the column. Small numbers show the probability that the difference is negligible.

PRACTICAL - 11

AIM: Perform Association, Classification and clustering from Data Cube created by OLAP Manager using DBMiner Tool.

THEORY AND OUTPUT:

1. Mining Associations from a Data Cube
   - **i. About Association**

Association mining on a set of data looks for values in different dimensions (attributes) that commonly occur together, suggesting an association between them.

In DBMiner, three kinds of associations could be possibly mined:

Inter-dimensional association. Associations among or across two or more dimensions.

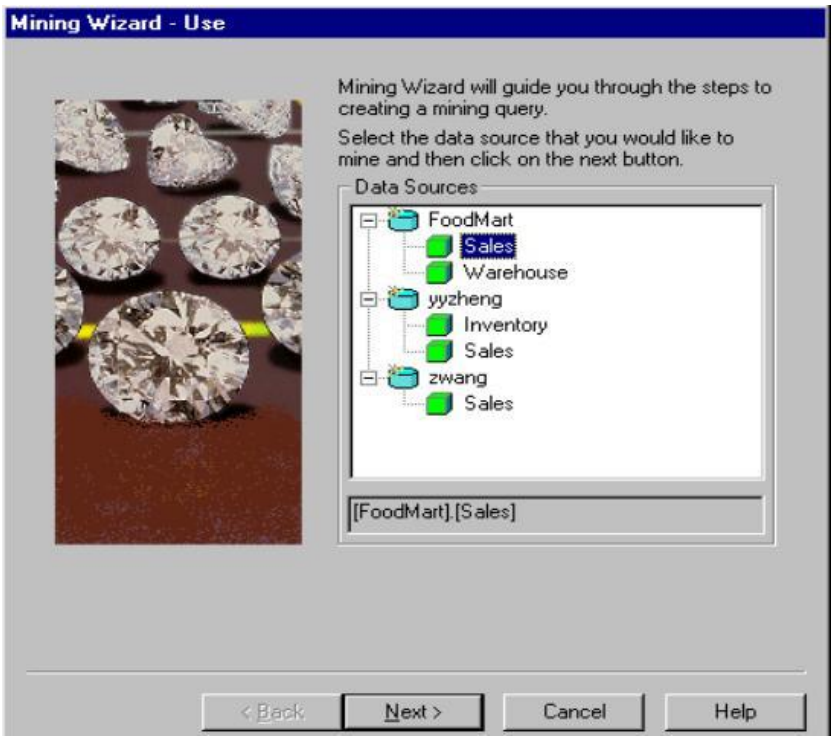
Customer-Country("Canada") => Product-Sub Category("Coffee")
i.e. Canadian customers are likely to buy coffee.

Intra-dimensional association. Associations present within one dimension grouped by another one or several dimensions. For example, if you want to find out which products customers in Canada are likely to purchase together:

Within Customer-Country("Canada"):
Product-ProductName("CarryBags") => Product-ProductName("Tents")
i.e. Customers in Canada, who buy carry-bags, are also likely to buy tents.

Hybrid association. Associations combining elements of both inter- and intra-dimensional association mining. For example,

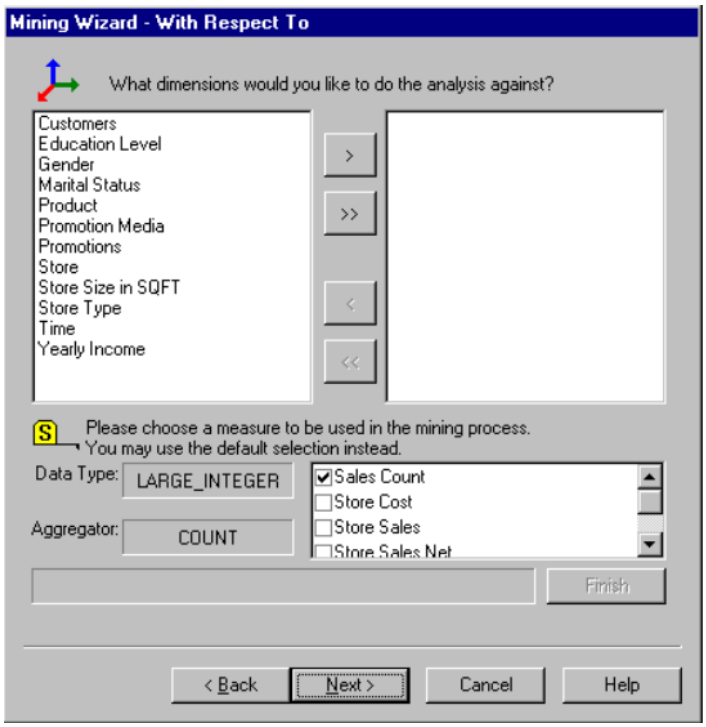
Step (a):

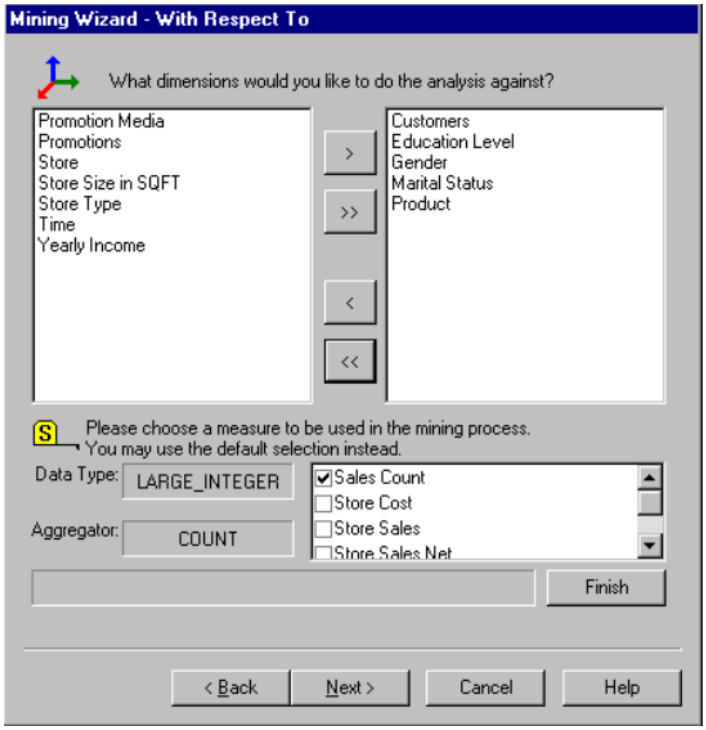

Mining Wizard - Use
Mining Wizard will guide you through the steps to creating a mining query.
Select the data source that you would like to mine and then click on the next button.
Data Sources
FoodMart
Sales
Warehouse
yyzheng
Inventory
Sales
zwang
Sales
[FoodMart].[Sales]
< Back
Next >
Cancel
Help

Step (b):

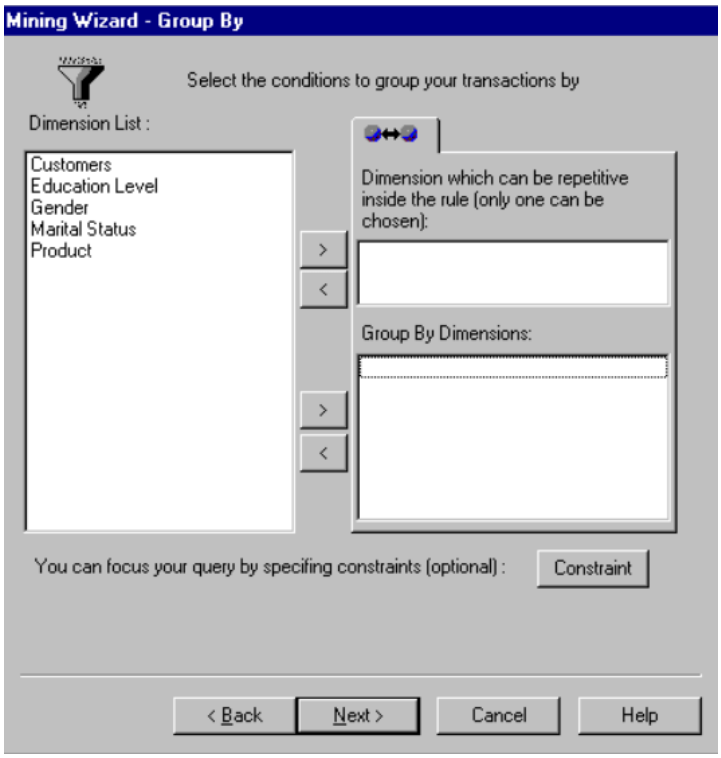

Mining Wizard - Mine
What kind of knowledge would you like DBMiner to discover?
Olap Browser
3D Cube Explorer
Association
Classification
Clustering
Finish
< Back
Next >
Cancel
Help

Step (c):

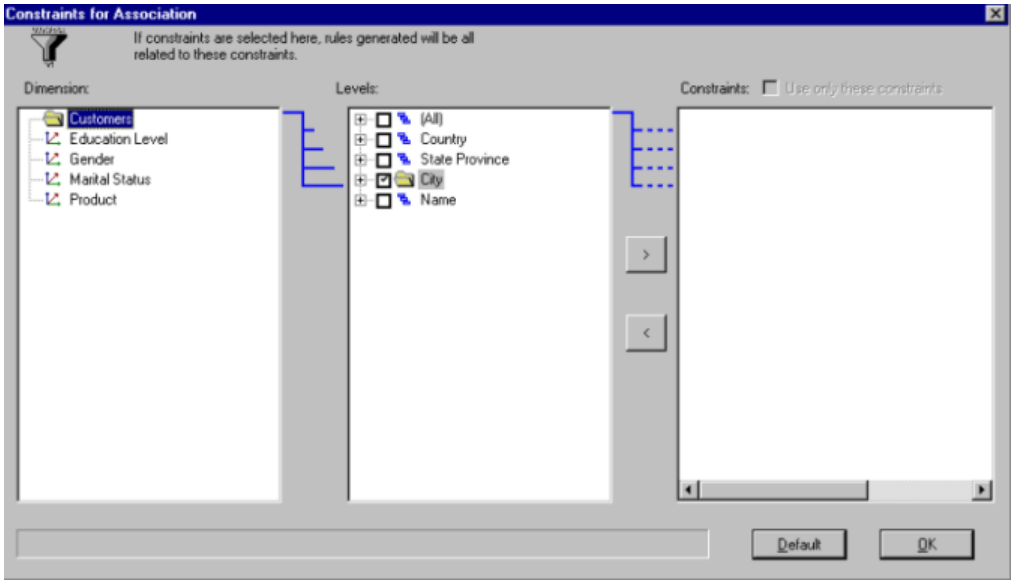

Mining Wizard - With Respect To
What dimensions would you like to do the analysis against?
Customers
Education Level
Gender
Marital Status
Product
Promotion Media
Promotions
Store
Store Size in SQFT
Store Type
Time
Yearly Income
>
>>
<
<<
Please choose a measure to be used in the mining process.
You may use the default selection instead.
Data Type:
LARGE_INTEGER
Aggregator:
COUNT
Sales Count
Store Cost
Store Sales
Store Sales Net
Finish
< Back
Next >
Cancel
Help


Mining Wizard - With Respect To
What dimensions would you like to do the analysis against?
Promotion Media
Promotions
Store
Store Size in SQFT
Store Type
Time
Yearly Income
Customers
Education Level
Gender
Marital Status
Product
>
>>
<
<<
Please choose a measure to be used in the mining process.
You may use the default selection instead.
Data Type:
LARGE_INTEGER
Aggregator:
COUNT
Sales Count
Store Cost
Store Sales
Store Sales Net
Finish
< Back
Next >
Cancel
Help

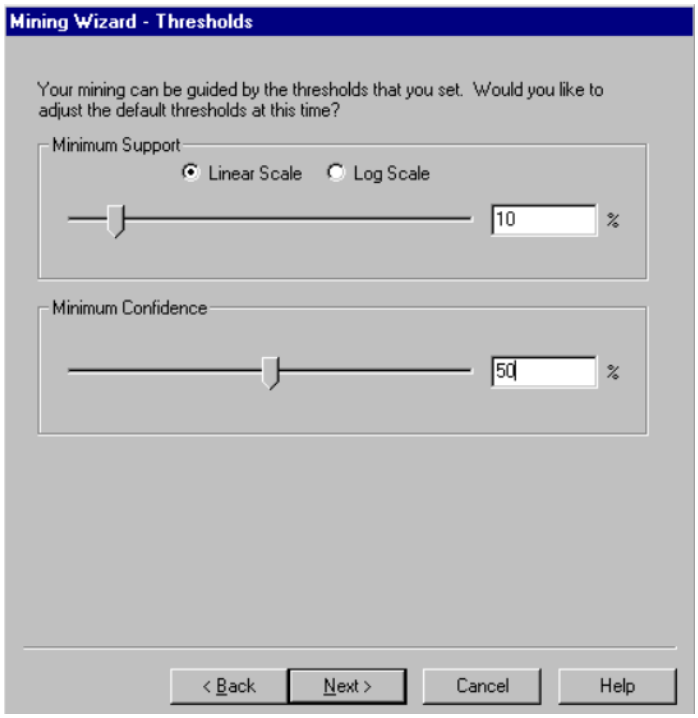
Step (d):


Mining Wizard - Group By
Select the conditions to group your transactions by
Dimension List :
Customers
Education Level
Gender
Marital Status
Product
Dimension which can be repetitive inside the rule (only one can be chosen):
Group By Dimensions:
You can focus your query by specifing constraints (optional) :
Constraint
< Back
Next >
Cancel
Help

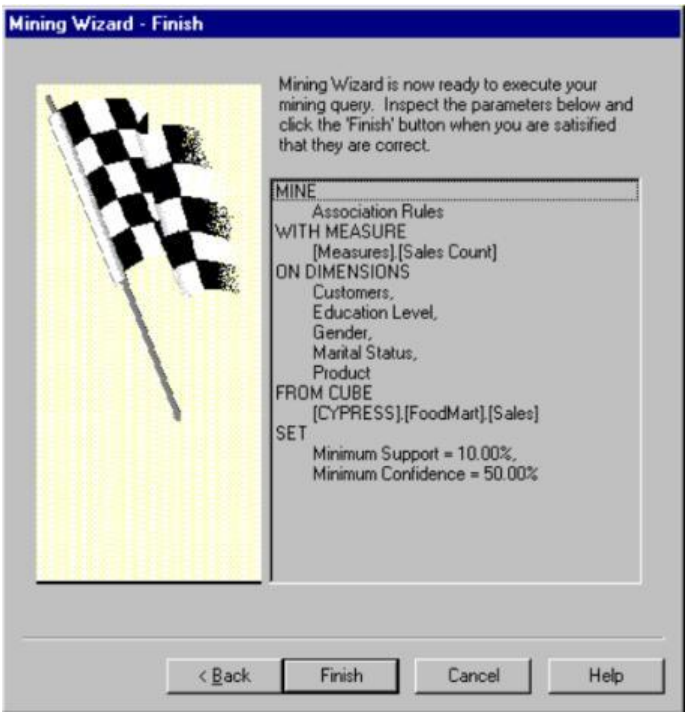
Step (e):


Constraints for Association
If constraints are selected here, rules generated will be all related to these constraints.
Dimension:
Levels:
Constraints:
Use only these constraints
Customers
Education Level
Gender
Marital Status
Product
(All)
Country
State Province
City
Name
Default
OK

Step (e):


Mining Wizard - Thresholds
Your mining can be guided by the thresholds that you set. Would you like to adjust the default thresholds at this time?
Minimum Support
Linear Scale
Log Scale
10
%
Minimum Confidence
50
%
< Back
Next >
Cancel
Help

Step (f):


Mining Wizard - Finish
Mining Wizard is now ready to execute your mining query. Inspect the parameters below and click the 'Finish' button when you are satisfied that they are correct.
MINE
Association Rules
WITH MEASURE
[Measures].[Sales Count]
ON DIMENSIONS
Customers,
Education Level,
Gender,
Marital Status,
Product
FROM CUBE
[CYPRESS].[FoodMart].[Sales]
SET
Minimum Support = 10.00%,
Minimum Confidence = 50.00%
< Back
Finish
Cancel
Help

2. Mining Classifications from a Data Cube

i. About Classification

Classification mining analyzes a set of training data (i.e. a set of objects whose class labels are known) and constructs a model for each class based on the features in the data. A set of classification rules are generated by the classification process, and these can be used to classify future data, as well as develop a better understanding of each class in the database.

In the classification process, attribute relevance analysis is very important. It is performed according to the analysis of an ***uncertainty measurement***, which determines how relevant an attribute is to the chosen classification attribute. Only a few of the most relevant attributes are retained for the classification analysis and the weakly relevant or irrelevant ones are not further considered.

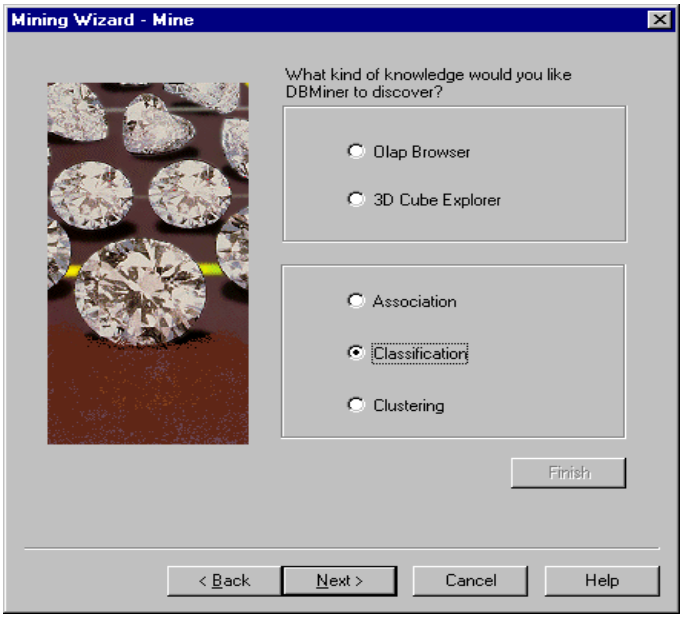
In DBMiner, three thresholds are used to tackle noise and exceptional data and facilitate statistical analysis.

- ***Classification threshold:*** Helps justify the classification of a particular subset of the data (found at a single node) when a significant portion of it belong to the same class.
- ***Noise threshold:*** Helps ignore a node if it contains only a negligible number of examples (i.e. noise).
- ***Training/testing set size:*** Sets the percentage of the whole data set to be used for training and testing. (i.e. 80% for training, 20% for testing).

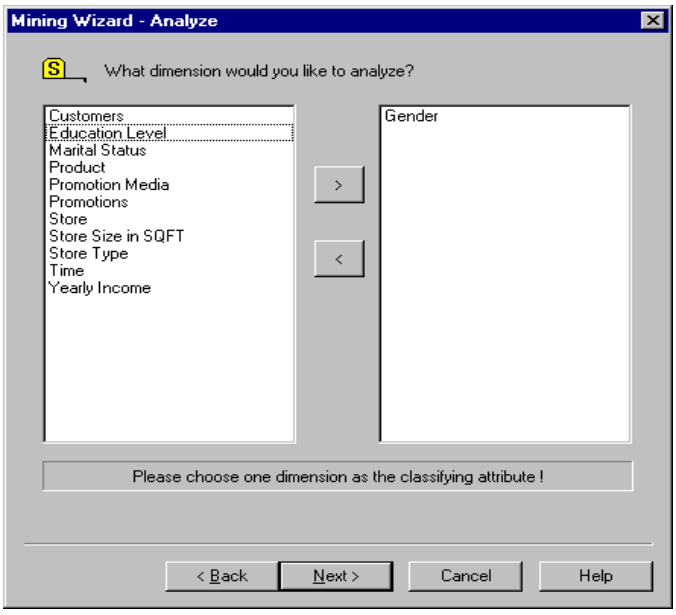
The following steps will take you through a classification task.

Step (a):

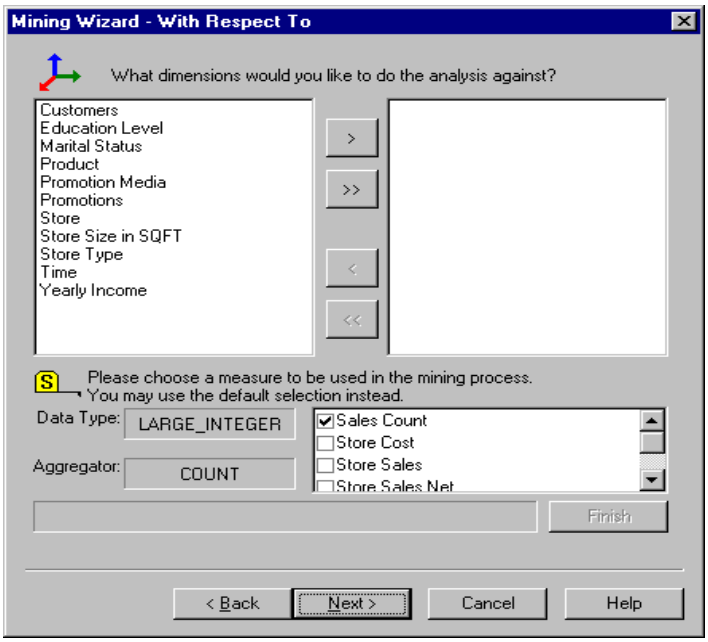
Step (b)

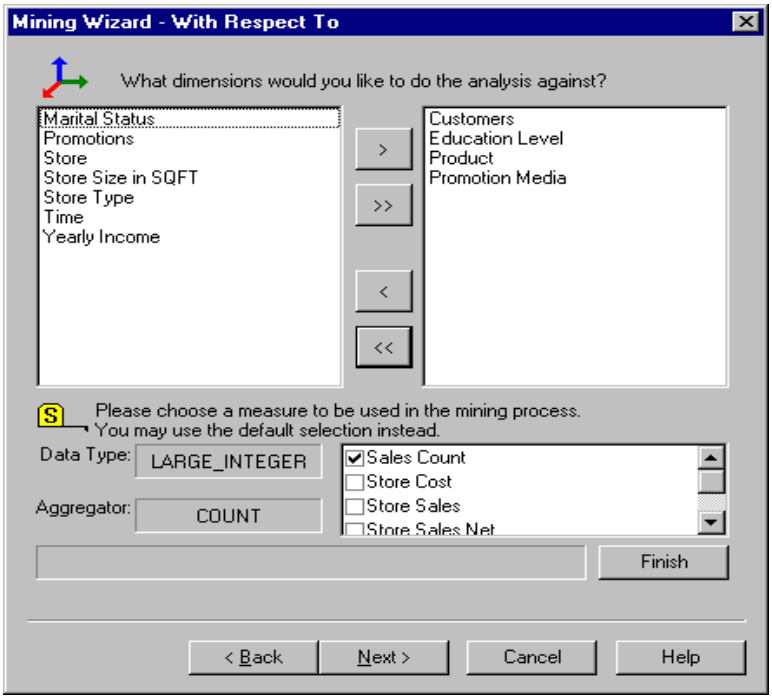


Step (c)

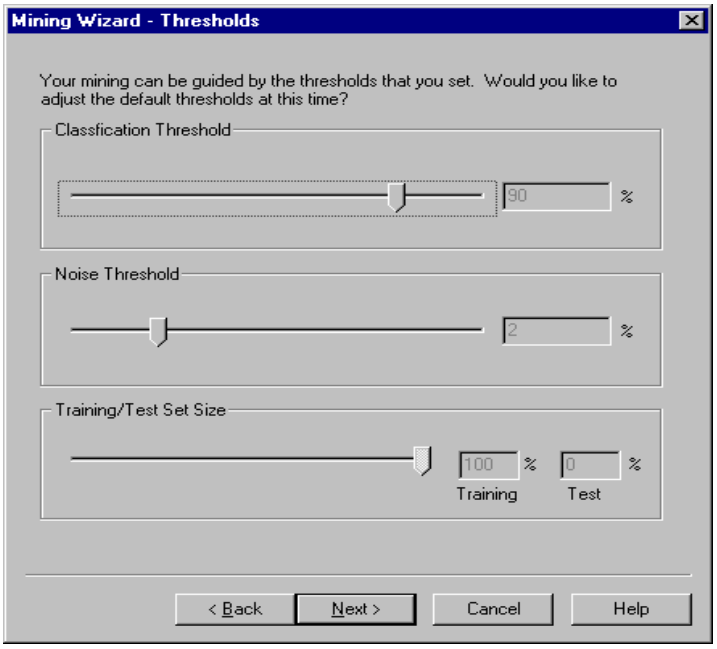

Mining Wizard - Analyze
What dimension would you like to analyze?
Customers
Education Level
Marital Status
Product
Promotion Media
Promotions
Store
Store Size in SQFT
Store Type
Time
Yearly Income
>
<
Gender
Please choose one dimension as the classifying attribute !
< Back
Next >
Cancel
Help

Step (d):

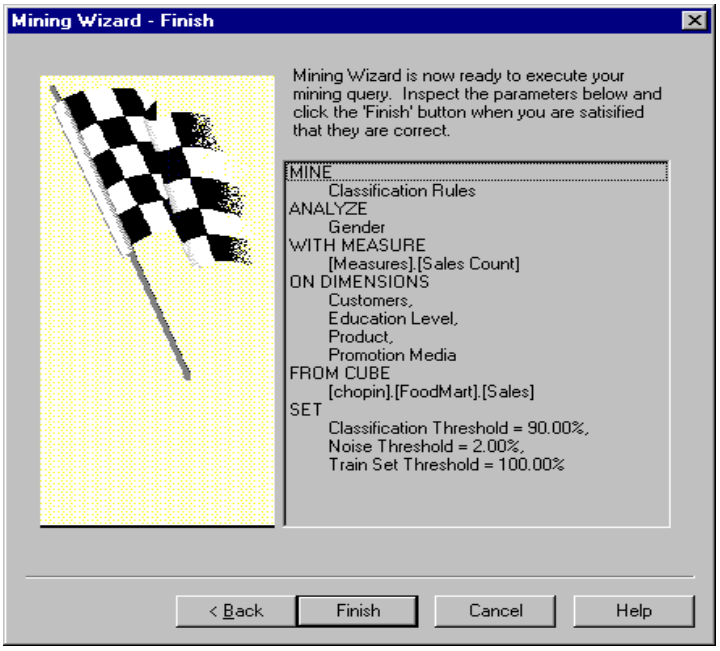

Mining Wizard - With Respect To
What dimensions would you like to do the analysis against?
Customers
Education Level
Marital Status
Product
Promotion Media
Promotions
Store
Store Size in SQFT
Store Type
Time
Yearly Income
>
>>
<
<<
Please choose a measure to be used in the mining process.
You may use the default selection instead.
Data Type:
LARGE_INTEGER
Aggregator:
COUNT
Sales Count
Store Cost
Store Sales
Store Sales Net
Finish
< Back
Next >
Cancel
Help


Mining Wizard - With Respect To
What dimensions would you like to do the analysis against?
Marital Status
Promotions
Store
Store Size in SQFT
Store Type
Time
Yearly Income
>
>>
<
<<
Customers
Education Level
Product
Promotion Media
Please choose a measure to be used in the mining process.
You may use the default selection instead.
Data Type:
LARGE_INTEGER
Aggregator:
COUNT
Sales Count
Store Cost
Store Sales
Store Sales Net
Finish
< Back
Next >
Cancel
Help

Step (e):


Mining Wizard - Thresholds
Your mining can be guided by the thresholds that you set. Would you like to adjust the default thresholds at this time?
Classfication Threshold
90
%
Noise Threshold
2
%
Training/Test Set Size
100
%
Training
0
%
Test
< Back
Next >
Cancel
Help

Step (f):


Mining Wizard - Finish
Mining Wizard is now ready to execute your mining query. Inspect the parameters below and click the 'Finish' button when you are satisified that they are correct.
MINE
Classification Rules
ANALYZE
Gender
WITH MEASURE
[Measures].[Sales Count]
ON DIMENSIONS
Customers,
Education Level,
Product,
Promotion Media
FROM CUBE
[chopin].[FoodMart].[Sales]
SET
Classification Threshold = 90.00%,
Noise Threshold = 2.00%,
Train Set Threshold = 100.00%
< Back
Finish
Cancel
Help

3. Mining Clusters from a Data Cube

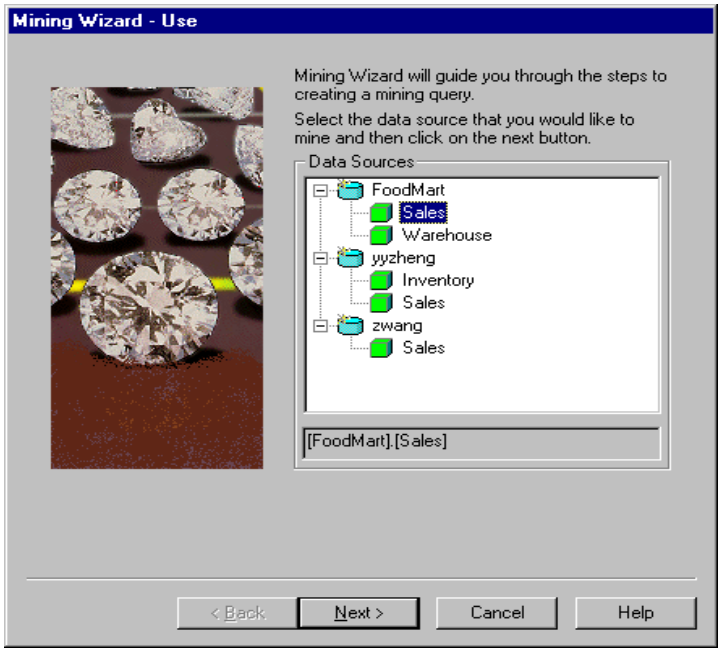
i. About Clustering

Clustering groups all the value of one dimension by the values of the second dimension. In DBMiner, only two cube dimensions can be chosen in a mining session since the clustering space is 2-dimensional. The underlying algorithm used in DBMiner is the *k*-means method. For detailed information about the *k*-means method, see the on-line documentation.

In DBMiner, there are several parameters that can be adjusted to tune the mining process.

- ***Number of clusters*** refers to *k* in the *k*-mean algorithm. If the number of clusters requested exceeds the number of points in the plane, an error message will be issued and the clustering process is stopped.
- ***Dimension weights*** refers to the coefficients for each dimension. The default value is 1.00 but can be decreased if you want a particular dimension to be relatively less influential with respect to the other dimension. Because this scaling reflects the relative different in influence, it is only calibrated between 0.01 - 1.00.
- ***Max clustering passes*** refers to the number of passes in the *k*-mean algorithm.
- ***Filter threshold*** ensures that cells in a data cube containing no records are not included in the clusters. This threshold can be raised to exclude more cells and thus reduce the number of points on a 2-dimensional plane.

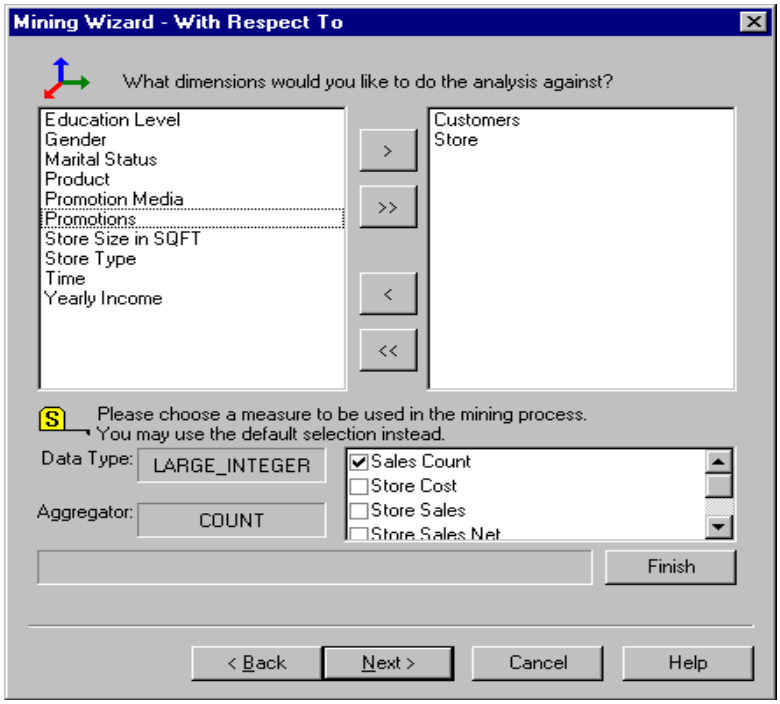
Step (a):


Mining Wizard - Use
Mining Wizard will guide you through the steps to creating a mining query.
Select the data source that you would like to mine and then click on the next button.
Data Sources
FoodMart
Sales
Warehouse
yyzheng
Inventory
Sales
zwang
Sales
[FoodMart].[Sales]
< Back
Next >
Cancel
Help

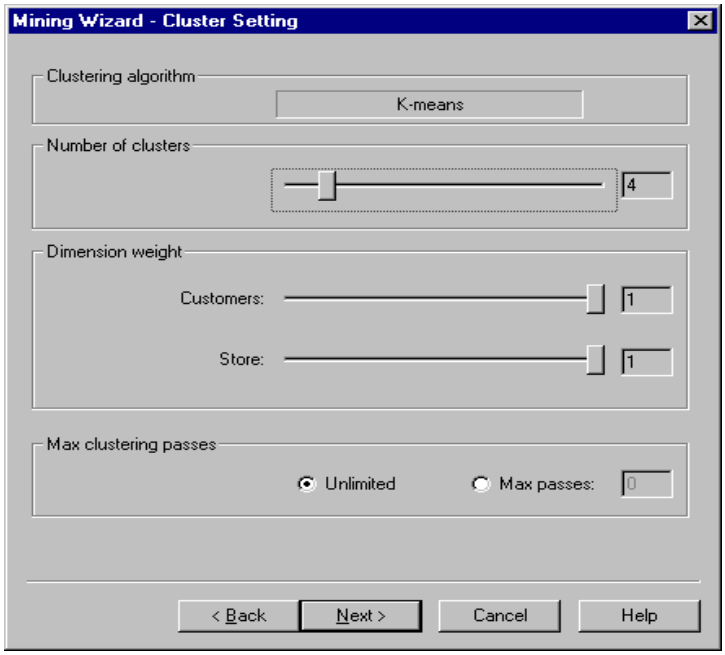
Step (b):


Mining Wizard - Mine
What kind of knowledge would you like DBMiner to discover?
Olap Browser
3D Cube Explorer
Association
Classification
Clustering
Finish
< Back
Next >
Cancel
Help

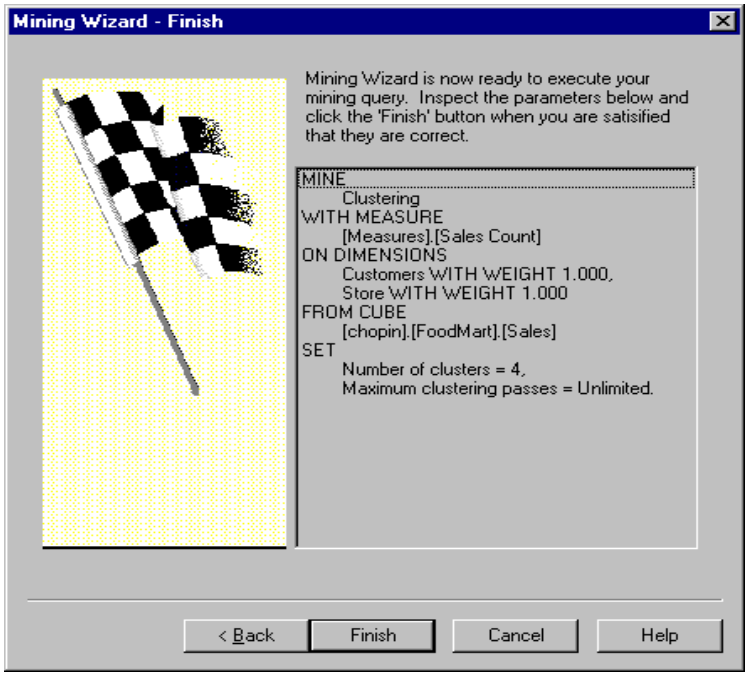
Step (c):


Mining Wizard - With Respect To
What dimensions would you like to do the analysis against?
Education Level
Gender
Marital Status
Product
Promotion Media
Promotions
Store Size in SQFT
Store Type
Time
Yearly Income
Customers
Store
>
>>
<
<<
Please choose a measure to be used in the mining process.
You may use the default selection instead.
Data Type:
LARGE_INTEGER
Aggregator:
COUNT
Sales Count
Store Cost
Store Sales
Store Sales Net
Finish
< Back
Next >
Cancel
Help

Step (d):


Mining Wizard - Cluster Setting
Clustering algorithm
K-means
Number of clusters
4
Dimension weight
Customers:
1
Store:
1
Max clustering passes
Unlimited
Max passes:
0
< Back
Next >
Cancel
Help

Step (e):


Mining Wizard - Finish
Mining Wizard is now ready to execute your mining query. Inspect the parameters below and click the 'Finish' button when you are satisified that they are correct.
MINE
Clustering
WITH MEASURE
[Measures].[Sales Count]
ON DIMENSIONS
Customers WITH WEIGHT 1.000,
Store WITH WEIGHT 1.000
FROM CUBE
[chopin].[FoodMart].[Sales]
SET
Number of clusters = 4,
Maximum clustering passes = Unlimited.
< Back
Finish
Cancel
Help

Practical - 12

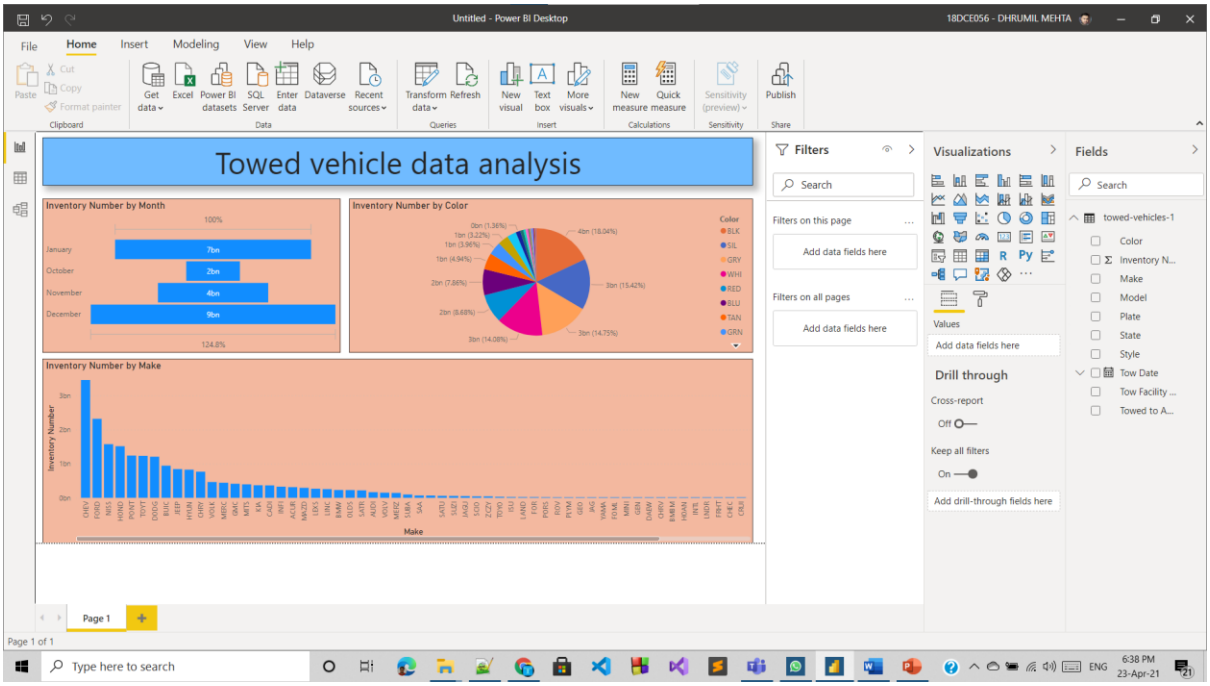
Aim: Implement Data Analysis and Data Visualization using Power BI and Tableau for vehicle dataset.

Power BI:

- Power BI is a Data Visualization and Business Intelligence tool that converts data from different data sources to interactive dashboards and BI reports. Power BI suite provides multiple software, connector, and services - Power BI desktop, Power BI service based on Saas, and mobile Power BI apps available for different platforms. These set of services are used by business users to consume data and build BI reports.
- Power BI desktop app is used to create reports, while Power BI Services (Software as a Service - SaaS) is used to publish the reports, and Power BI mobile app is used to view the reports and dashboards.

Implementation:

Vehicle towed dataset used



➢ **Conclusion:** With the use of PowerBI tool we can visualize any type data easily and it can easily convert different data to interactive dashboards and BI report.