Week – 2

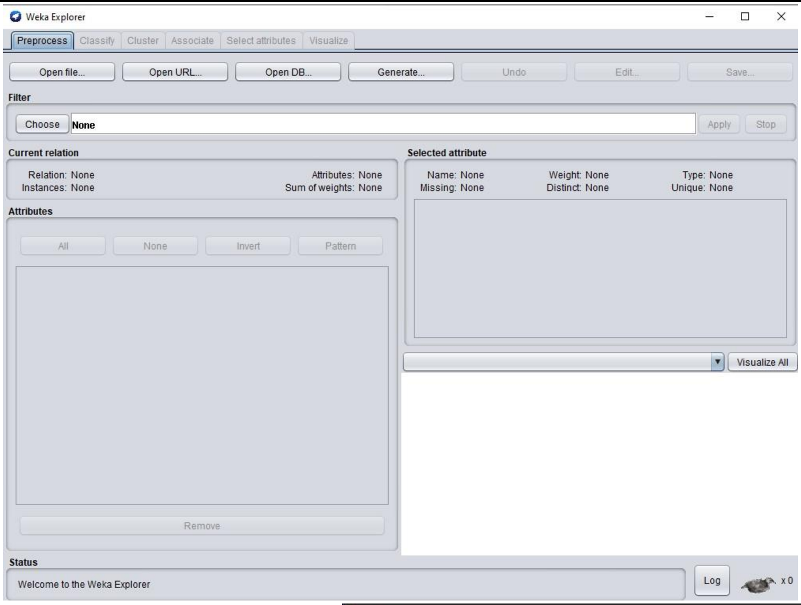
Explore machine learning tool "WEKA" Study the arff file format Explore the available data sets in WEKA. Load a data set (ex. Weather dataset, Iris dataset, etc.) Load each dataset and observe the following:

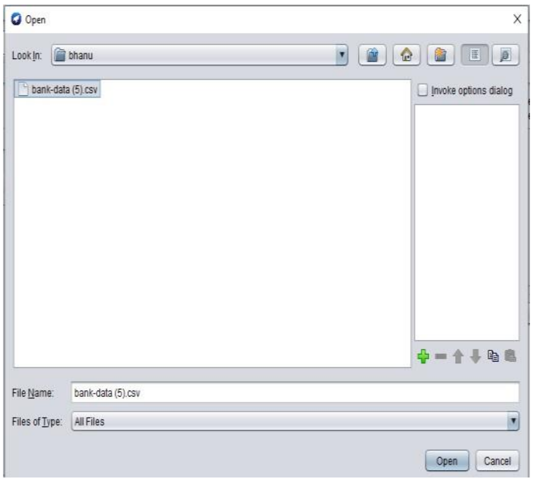
1. **List the attribute names and they types**
2. **Number of records in each dataset**
3. **Identify the class attribute (if any)**
4. **Plot Histogram**
5. **Determine the number of records for each class.**
6. **Visualize the data in various dimensions Introduction to WEKA**

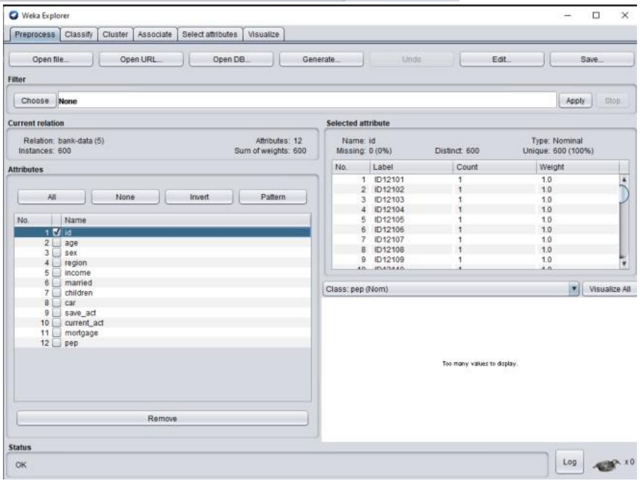
WEKA - an open source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualization tools so that you can develop machine learning techniques and apply them to real-world data mining problems.

features:

i) Preprocess
ii) Classify
iii) Cluster
iv) Associate
v) Select Attributes
vi) Visualise







Available data sets in weka:

1.airline
2.breast cancer
3.contact lenses
4.cpu
5.cpu with vendor
6.iris
7.weather.nominal
8.weather.numeric
9.diabetes
10.glass

WEATHER.ARFF:

@relation weather
@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}
@data
sunny,85,85,FALSE,
no
sunny,80,90,TRUE,n
o
overcast,83,86,FALS
E,yes
rainy,70,96,FALSE,y
es
rainy,68,80,FALSE,y
es
rainy,65,70,TRUE,no
overcast,64,65,TRUE
,yes
sunny,72,95,FALSE,
no
sunny,69,70,FALSE,
yes
rainy,75,80,FALSE,y
es
sunny,75,70,TRUE,y
es
overcast,72,90,TRUE
,yes
overcast,81,75,FALS
E,yes
rainy,71,91,TRUE,no

IRIS.ARFF:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall
(MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
% 3. Past Usage:
%    - Publications: too many to mention!!! Here are a few.
%    1. Fisher,R.A. "The use of multiple measurements
in taxonomic problems"
%       Annual Eugenics, 7, Part II, 179-188 (1936);
also in "Contributions
%       to Mathematical Statistics" (John Wiley, NY,
1950).
%    2. Duda,R.O., & Hart,P.E. (1973) Pattern
Classification and Scene Analysis.
%       (Q327.D83) John Wiley & Sons.  ISBN 0-471-
22361-1.  See page 218.
5. Number of Instances: 150 (50 in each of three classes)
%
% 6. Number of Attributes: 4 numeric, predictive
attributes and the class
%
% 7. Attribute Information:
%    1. sepal length in cm
%    2. sepal width in cm
%    3. Petal length in cm
%    4. Petal width in cm
%    5. class:
%       -- Iris Setosa
%       -- Iris Versicolour
%       -- Iris Virginica
%
% 8. Missing Attribute Values: None
%
% Summary Statistics:
%              Min  Max   Mean    SD   Class Correlation
%    sepal length: 4.3  7.9   5.84  0.83    0.7826
%     sepal width: 2.0  4.4   3.05  0.43   -0.4194
```

```
%    petal length: 1.0  6.9   3.76  1.76    0.9490  (high!)
%     petal width: 0.1  2.5   1.20  0.76    0.9565  (high!)
%
% 9. Class Distribution: 33.3% for each of 3 classes.
@RELATION iris
@ATTRIBUTE sepallength    REAL
@ATTRIBUTE sepalwidth     REAL
@ATTRIBUTE petallength    REAL
@ATTRIBUTE petalwidth     REAL
@ATTRIBUTE class  {Iris-setosa,Iris-versicolor,Iris-
virginica}
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.5,3.0,5.8,2.2,Iris-virginica
```

AIRLINE.ARFF:

```
%% Monthly totals of international airline passengers
(in thousands) for  %% 1949-1960.

@relation airline_passengers
@attribute passenger_numbers numeric
@attribute Date date 'yyyy-MM-dd'

@data
112,1949-01-01
118,1949-02-01
132,1949-03-01
129,1949-04-01
121,1949-05-01
135,1949-06-01
148,1949-07-01
148,1949-08-01
136,1949-09-01
119,1949-10-01
104,1949-11-01
118,1949-12-01
115,1950-01-01
126,1950-02-01
141,1950-03-01
135,1950-04-01
125,1950-05-01
149,1950-06-01
170,1950-07-01
170,1950-08-01
158,1950-09-01
133,1950-10-01
```

CPU.ARFF:

```
%
% As used by Kilpatrick, D. & Cameron-Jones, M.
(1998). Numeric prediction
% using instance-based learning with encoding length
selection. In Progress
% in Connectionist-Based Information Systems.
Singapore: Springer-Verlag.
%
% Deleted "vendor" attribute to make data
consistent with with what we % used in the data
mining book.
%
@relation 'cpu'
@attribute MYCT numeric
@attribute MMIN numeric
@attribute MMAX numeric
@attribute CACH numeric
@attribute CHMIN numeric
@attribute CHMAX numeric
@attribute class numeric
@data
125,256,6000,256,16,128,198
29,8000,32000,32,8,32,269
29,8000,32000,32,8,32,220
29,8000,32000,32,8,32,172
29,8000,16000,32,8,16,132
26,8000,32000,64,8,32,318
23,16000,32000,64,16,32,367
23,16000,32000,64,16,32,489
23,16000,64000,64,16,32,636
23,32000,64000,128,32,64,1144
400,1000,3000,0,1,2,38
```

CONTACT-LENSES.ARFF:

@relation contact-lenses

@attribute age	{young, pre-presbyopic, presbyopic}
@attribute spectacle-prescrip	{myope, hypermetrope}
@attribute astigmatism	{no, yes}
@attribute tear-prod-rate	{reduced, normal}
@attribute contact-lenses	{soft, hard, none}

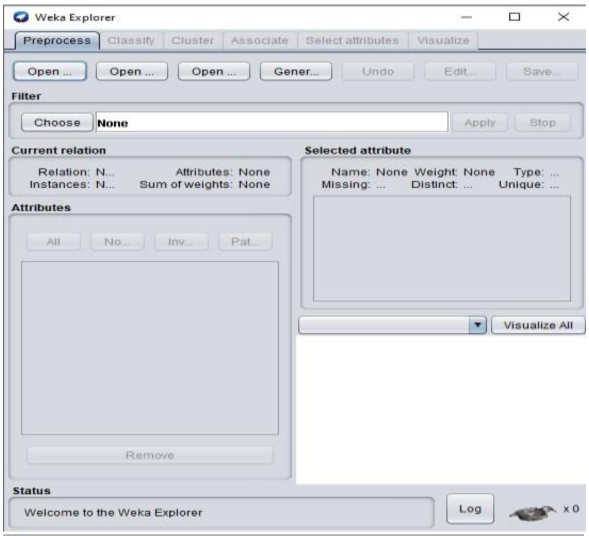
@data
%
% 24
instances %
young,myope,no,reduced,none
young,myope,no,normal,soft
young,myope,yes,reduced,none
young,myope,yes,normal,hard
young,hypermetrope,no,reduced,none
young,hypermetrope,no,normal,soft

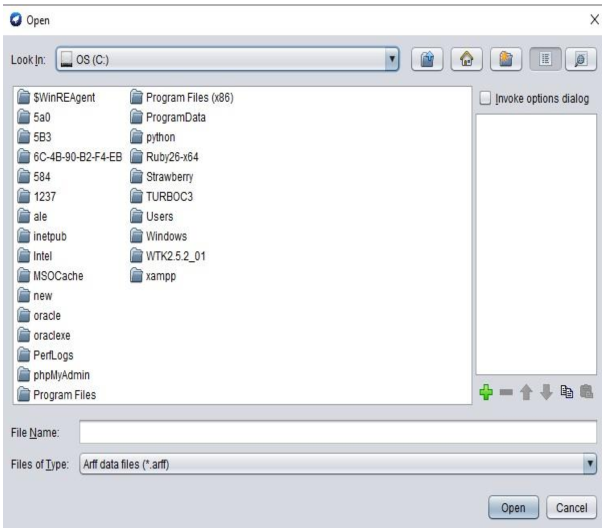
LOAD DATA SETS IN WEKA

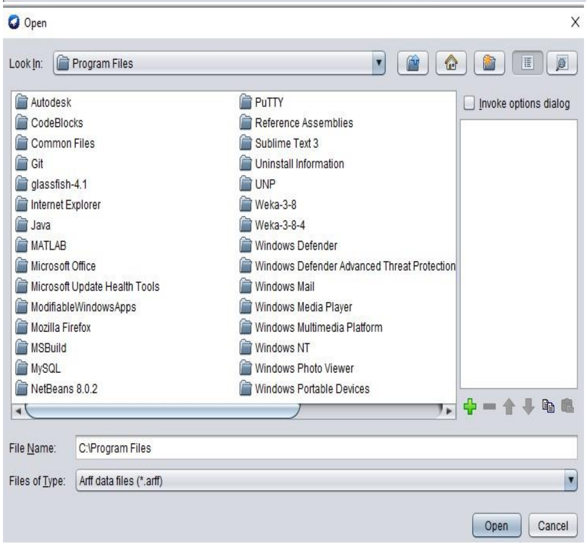
DESCRIPTION:

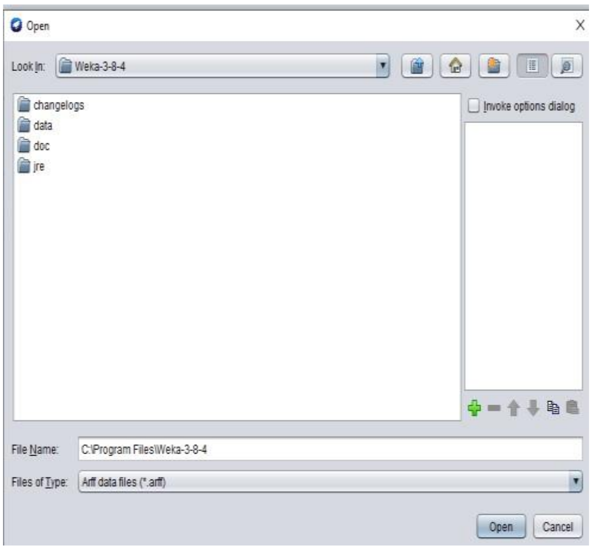
Step 1: open weka
Step 2:Go to file explorer
Step 3: Select open file under
preprocess
Step 4: Select the folder where the arff file is located
Step 5:Open the file
Step 6:observe attributes names, types, class attribute

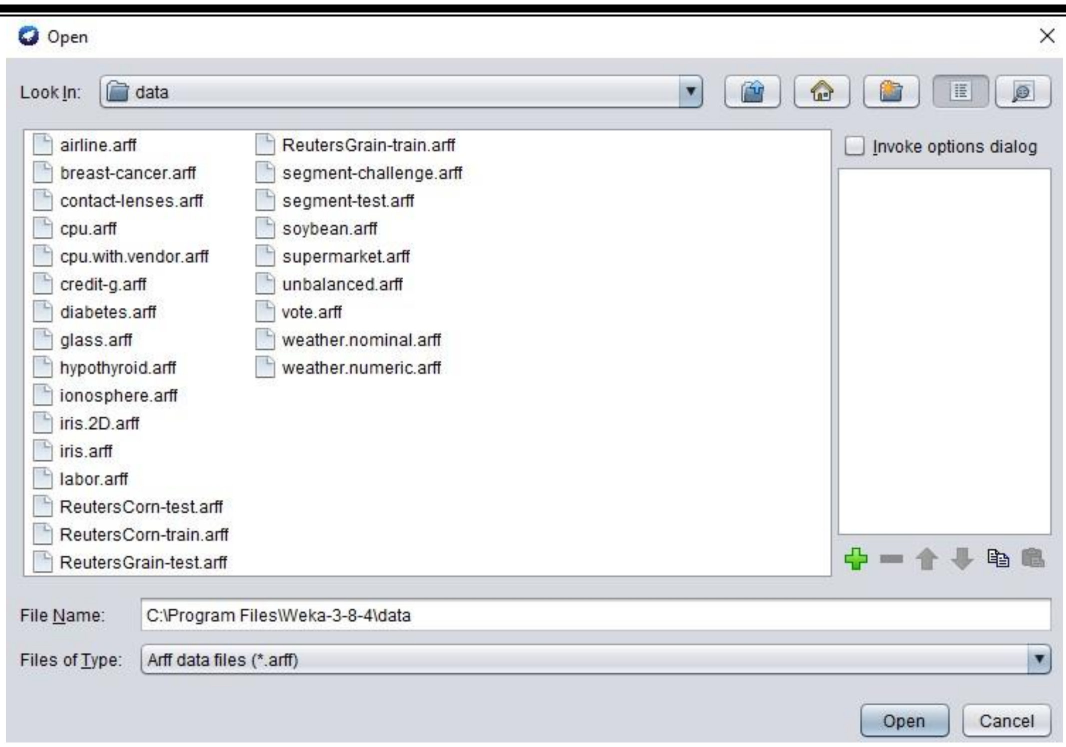
1.**WEATHER.ARFF:**



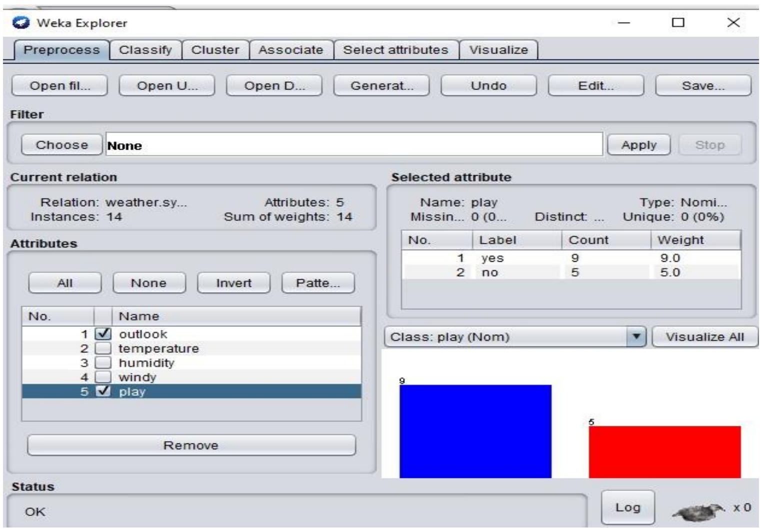








UPLOADING WEATHER.ARFF FILE:



1.List the attribute names and they types

Attributes are outlook, temperature, humidity, windy, play.

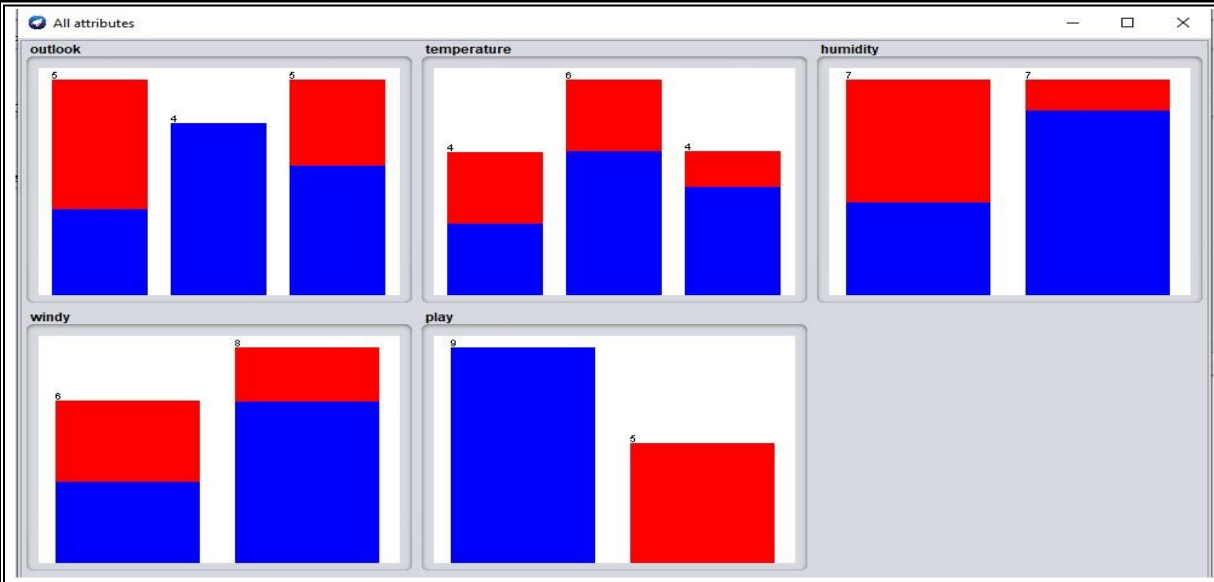
2.Number of records in each dataset

number of records are 14

3. Identify the class attribute (if any)

class attribute is play

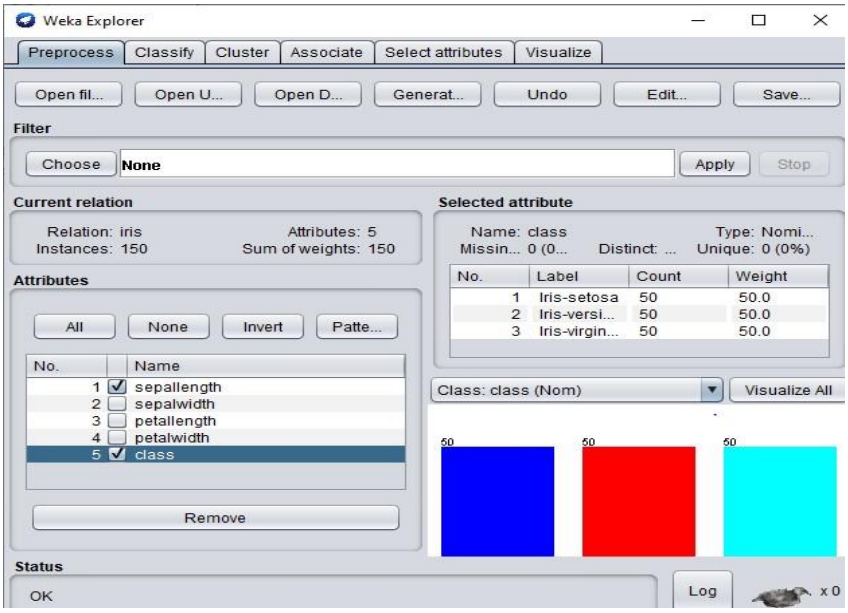
4.Histogram



Python Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data=pd.read_csv("/Iris.csv")
print(data.head(10)) data.info()
plt.figure(figsize=(10,7))
```

IRIS.ARFF:



1.List the attribute names and they types

sepallength
REAL
sepalwidth
REAL
petallength
REAL
petalwidth
REAL
class {Iris-setosa,Iris-versicolor,Iris-virginica}

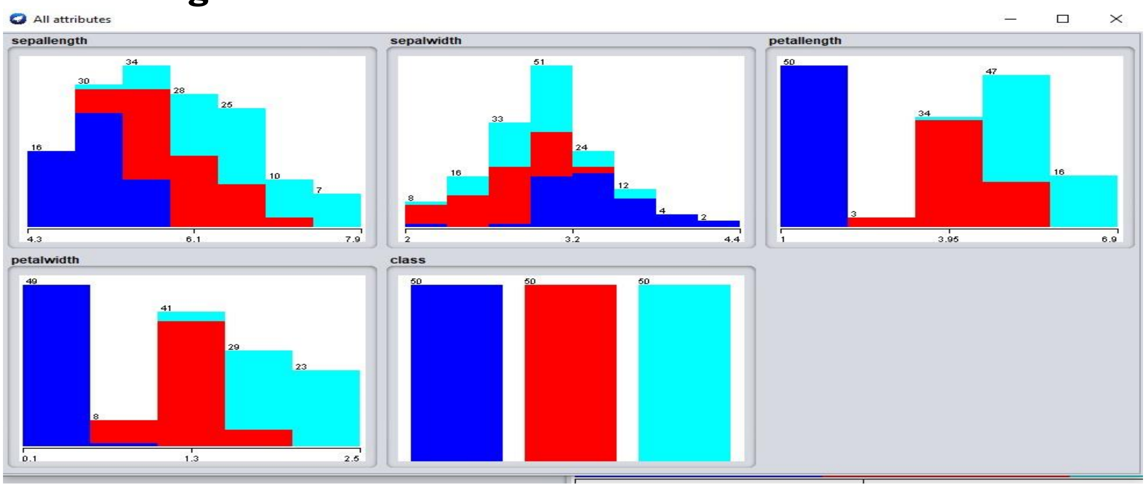
2.Number of records in each dataset

Number of records = 150

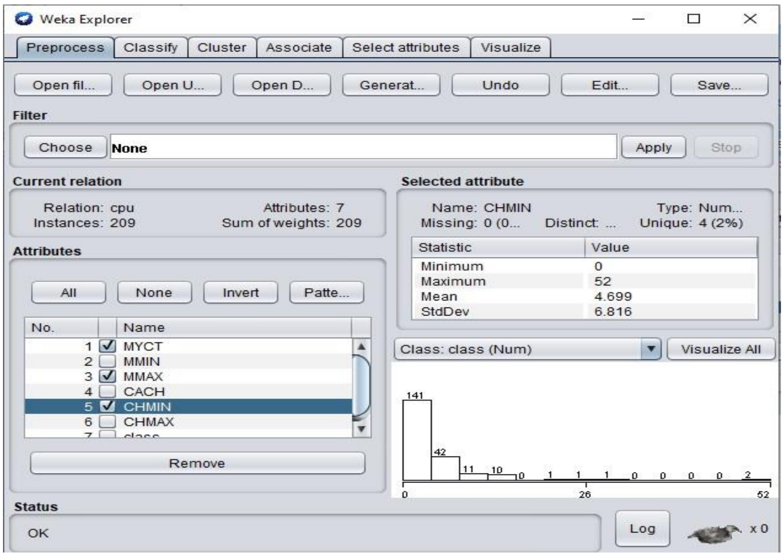
3.Identify the class attribute (if any)

Class {Iris-setosa, Iris-versicolor, Iris-virginica}

4.Plot Histogram



CPU.ARFF:



1.List the attribute names and they types

attribute MYCT
numeric MMIN
numeric
MMAX numeric
@attribute CACH numeric
@attribute CHMIN numeric
@attribute CHMAX numeric
@attribute class numeric

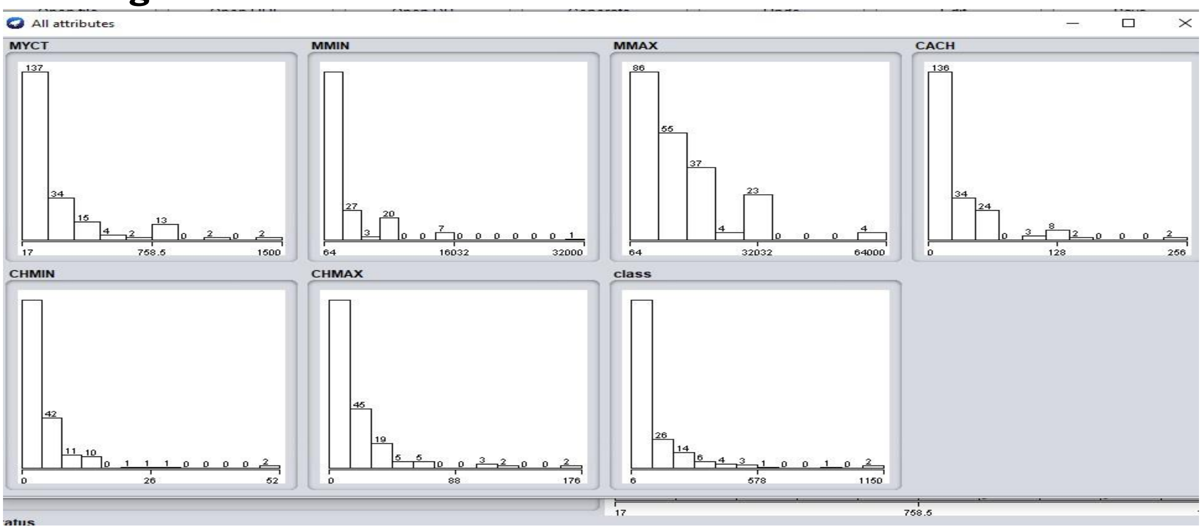
2.Number of records in each dataset

Number of records : 11

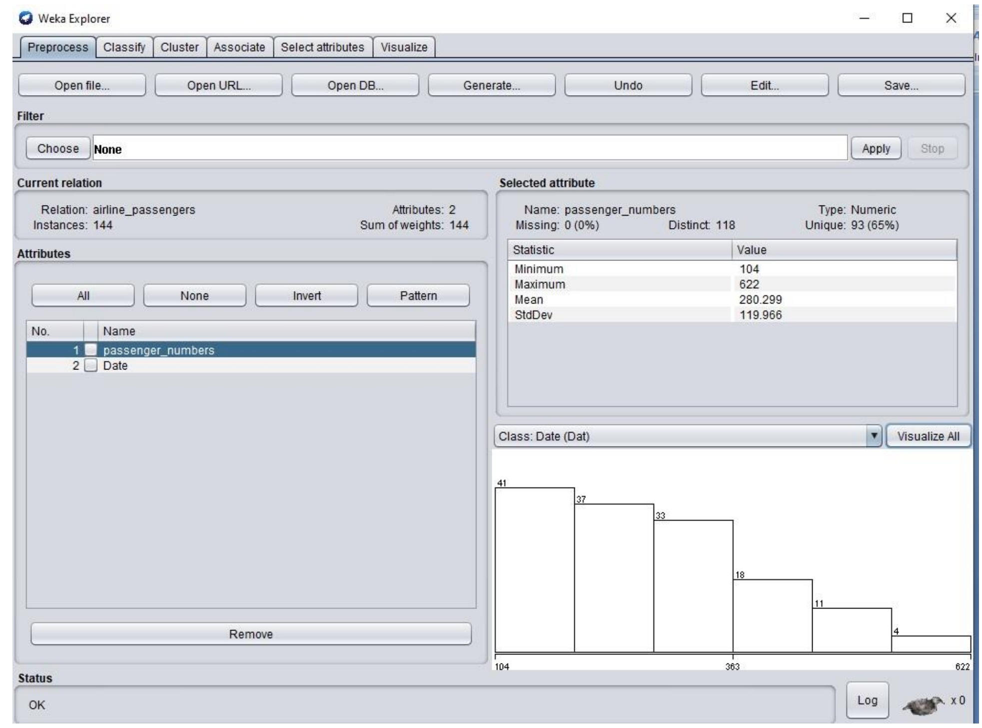
3.Identify the class attribute (if any)

class numeric

4.histogram:



AIRLINE.ARFF File:



1.List the attribute names and they types

Passengers number

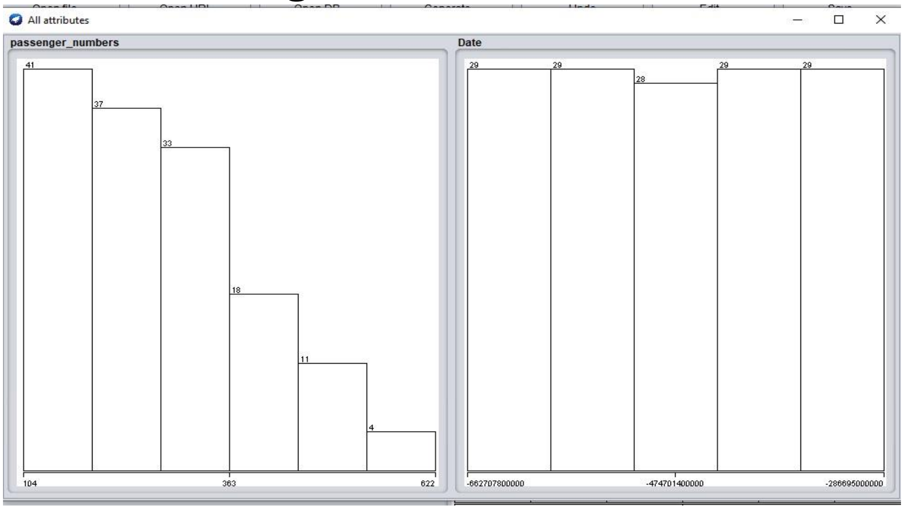
DATE

2.Number of data records: 22

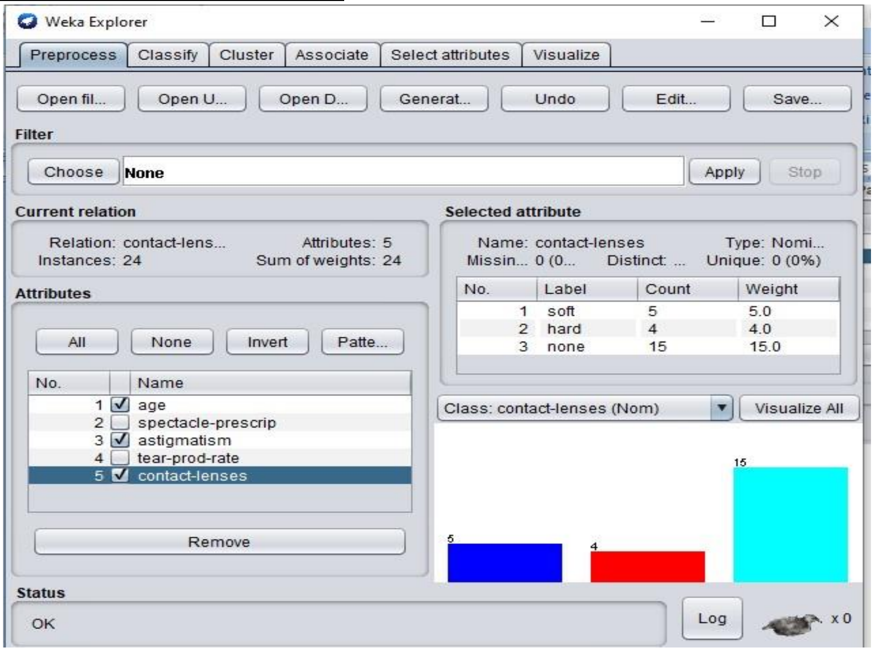
3.Identify the class attribute (if any)

No class attribute

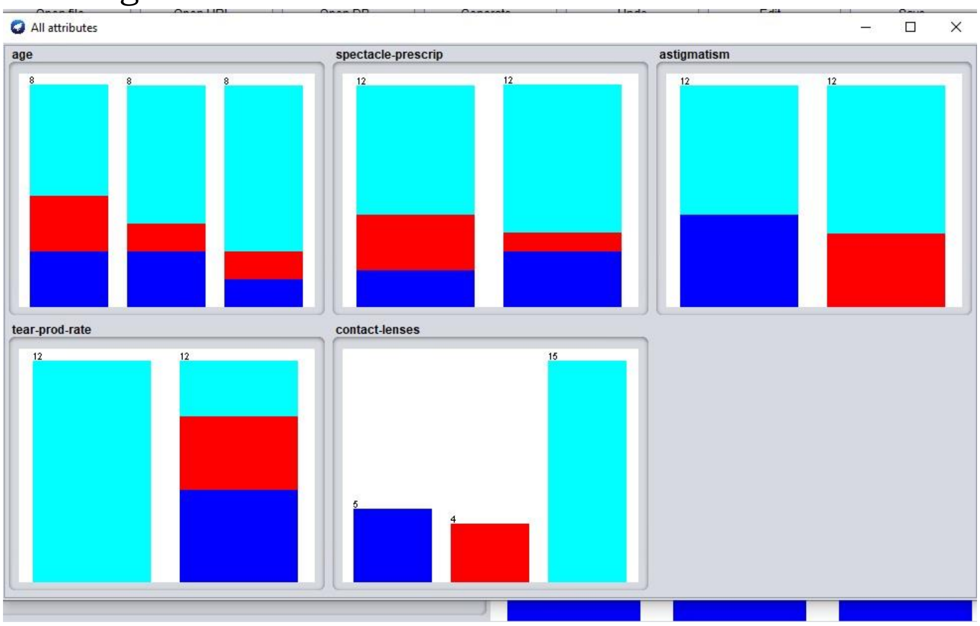
4.Plot the Histogram



Contact-lenses.arff:



4.histogram:



numpy:
NumPy is a Python library used for working with arrays.
It also has functions for working in domain of linear algebra, fourier transform, and matrices.
NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

pandas:
pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python

matplotlib:
Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications

Program:

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

```
[ ] data=pd.read_csv("/content/Iris (1).csv")
```

```
[ ] print(data.head(10))
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
5   6            5.4           3.9            1.7           0.4  Iris-setosa
6   7            4.6           3.4            1.4           0.3  Iris-setosa
7   8            5.0           3.4            1.5           0.2  Iris-setosa
8   9            4.4           2.9            1.4           0.2  Iris-setosa
9  10            4.9           3.1            1.5           0.1  Iris-setosa
```

```
[ ] data.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161

[ ]

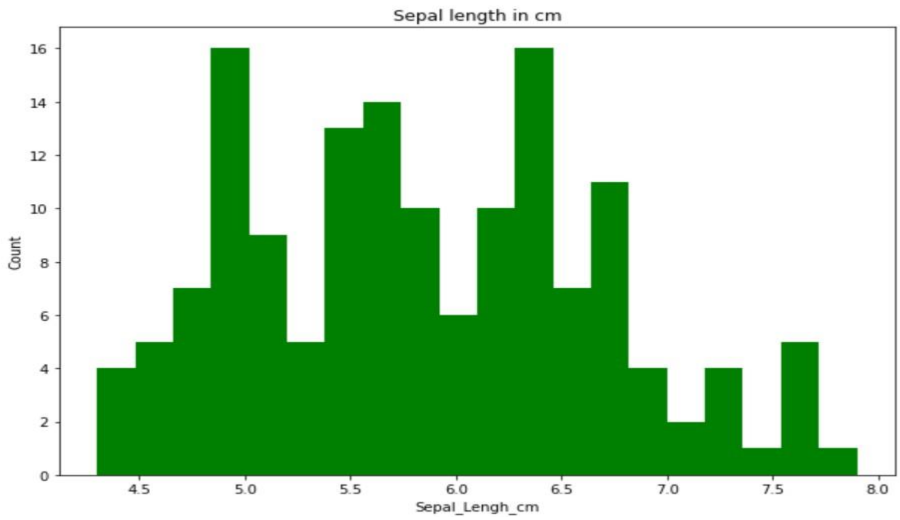
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

[ ]
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

[ ]
```
plt.figure(figsize=(10,7))
x=data["SepalLengthCm"]
plt.hist(x,bins=20,color="green")
plt.title("Sepal length in cm")
plt.xlabel("Sepal_Lengh_cm")
plt.ylabel("Count")
```

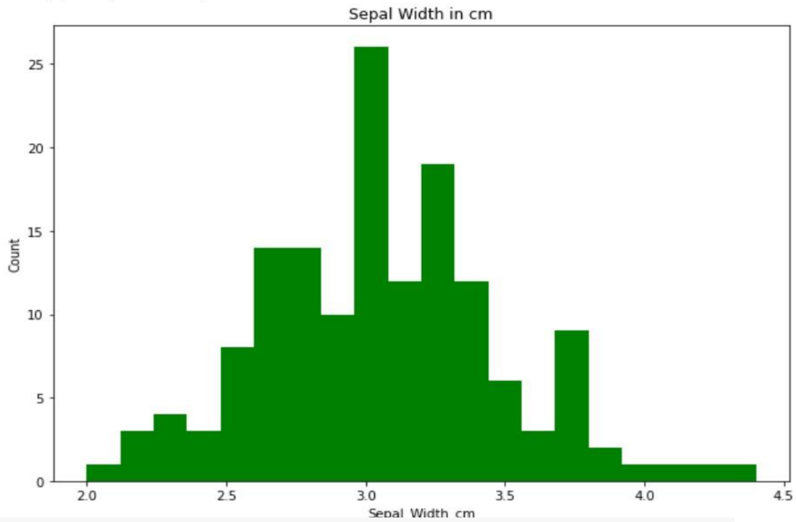
```
Text(0, 0.5, 'Count')
```



```
[ ] plt.figure(figsize=(10,7))
    x=data["SepalWidthCm"]
    plt.hist(x,bins=20,color="green")
    plt.title("Sepal Width in cm")
    plt.xlabel("Sepal_Width_cm")
    plt.ylabel("Count")
```

```
Text(0, 0.5, 'Count')
```

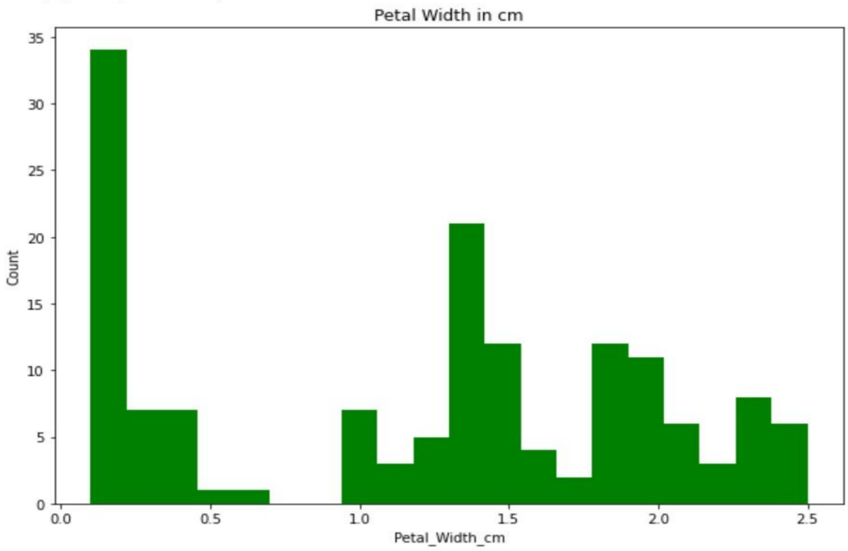
[ ]




```
[ ] plt.figure(figsize=(10,7))
    x=data["PetalWidthCm"]
    plt.hist(x,bins=20,color="green")
    plt.title("Petal Width in cm")
    plt.xlabel("Petal_Width_cm")
    plt.ylabel("Count")
```

```
Text(0, 0.5, 'Count')
```

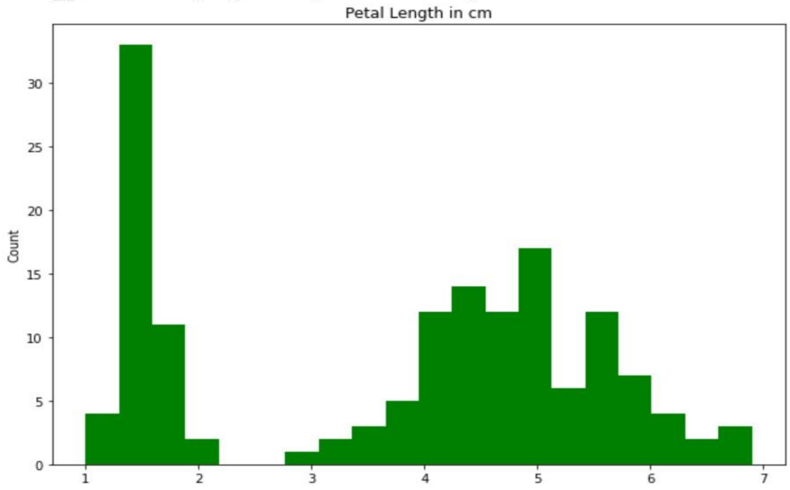
[ ]



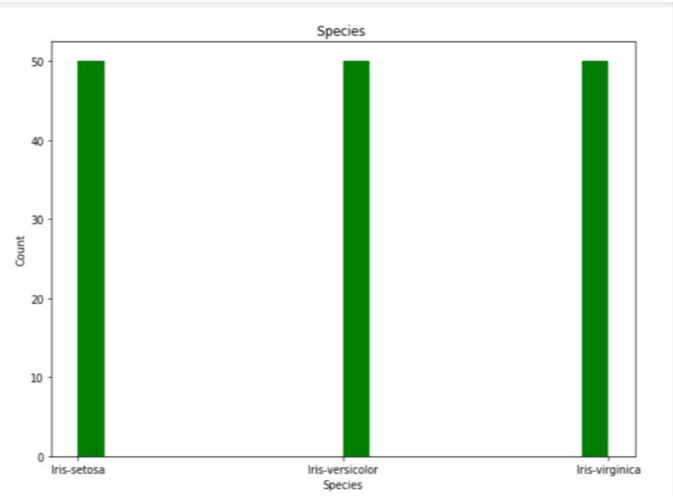

```
plt.figure(figsize=(10,7))
x=data["PetalLengthCm"]
plt.hist(x,bins=20,color="green")
plt.title("Petal Length in cm")
plt.xlabel("Petal_length_cm")
plt.ylabel("Count")
```

```
Text(0, 0.5, 'Count')
```

[ ]




```
[ ] plt.figure(figsize=(10,7))
    x=data.Species
    plt.hist(x,bins=20,color="green")
    plt.title("Species ")
    plt.xlabel("Species")
    plt.ylabel("Count")
    plt.show()
```



5.Determine the number of records for each class.

<u>Iris-data set</u>

Iris-setosa : 50 records

Iris-versicolor : 50 records

Iris virginica : 50 records

<u>Weather.nominal set</u>

9 records : yes

5 records : no

total 14 records

<u>Diabetes:</u>

500 records : tested_negative diabetes

268 records : tested_positive diabetes

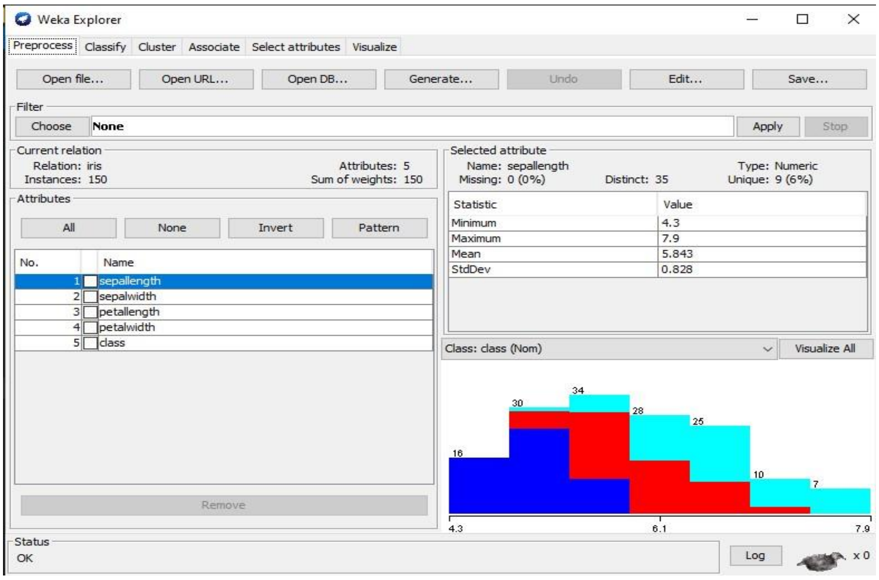
<u>Breast Cancer</u>

201 records : no recurrence events

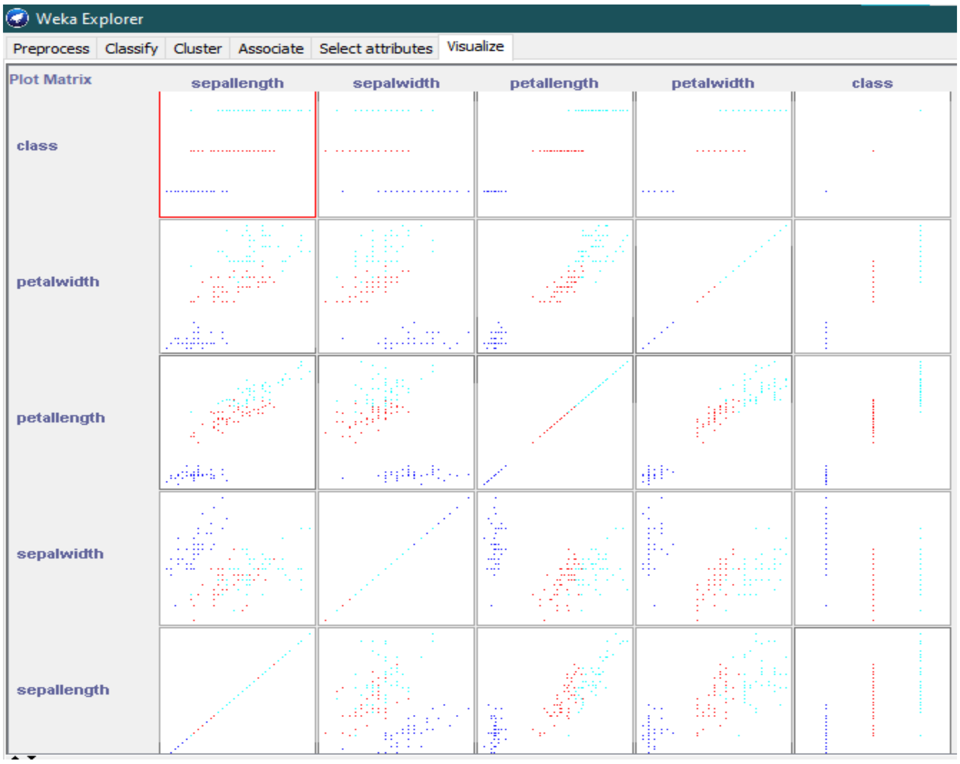
85 records : recurrence events

6.Visualize the data in various dimensions

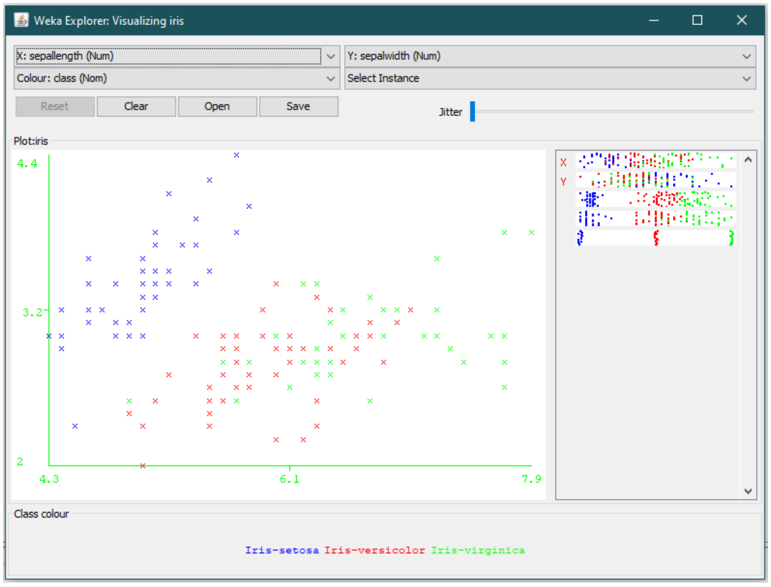
Load iris data set into weka



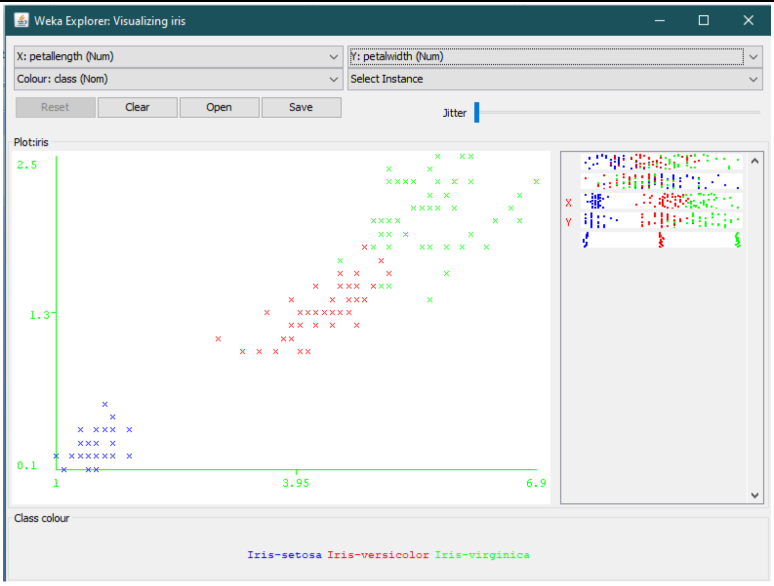
click on visualize tab (next to select attributes)



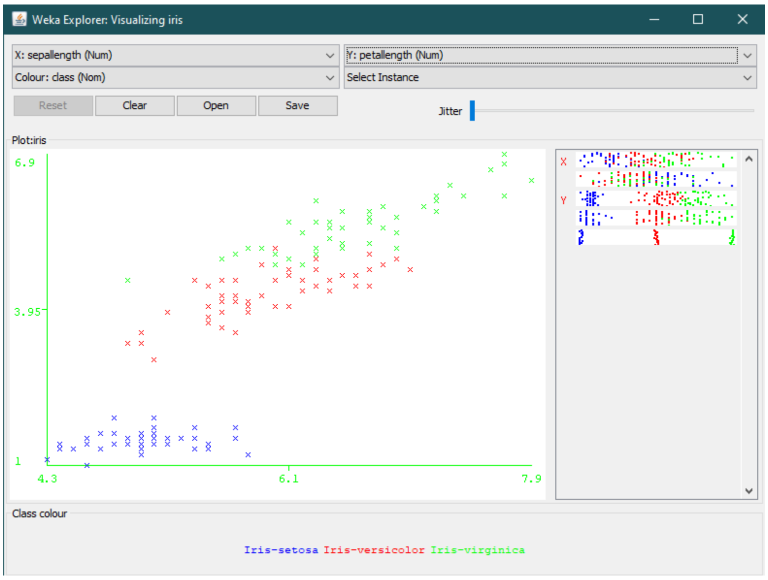
i) sepal_length vs sepal_width



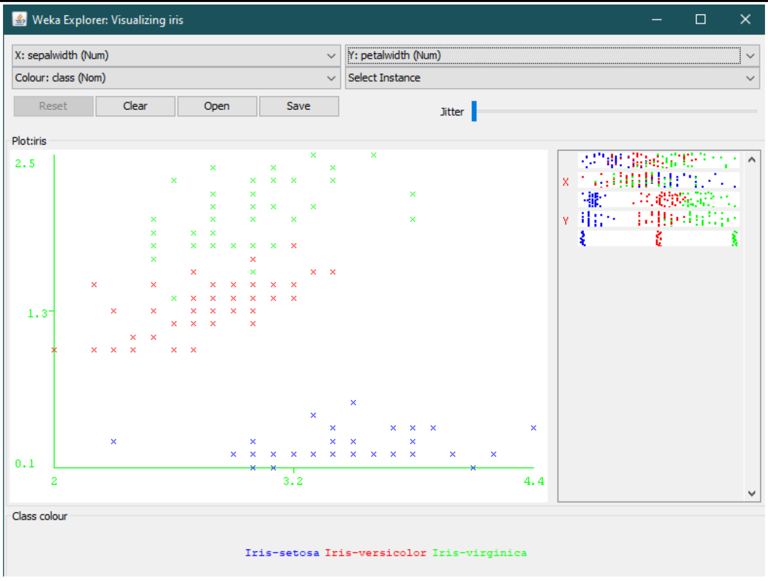
ii) petal_length vs petal_width



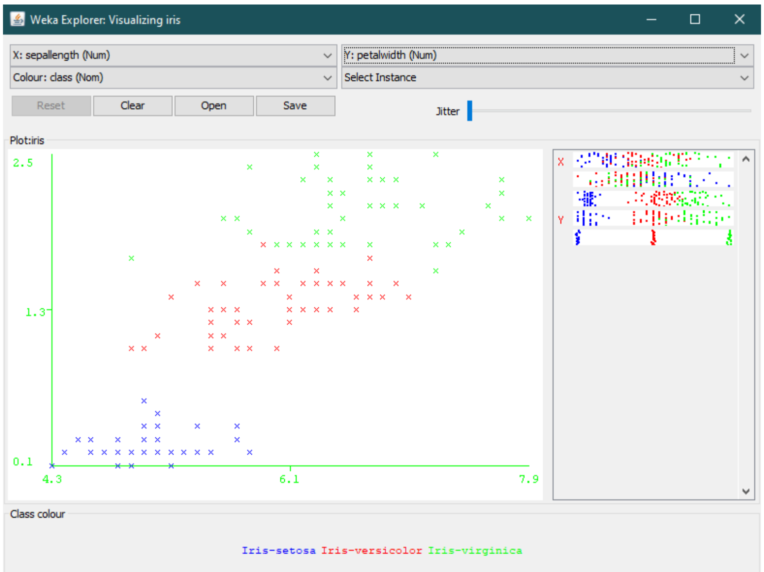
iii) sepal_length vs petal_length



iv) sepal_width vs petal_width



v) sepal_length vs petal_width



Week – 3

Perform following data preprocessing tasks using Python

i) Rescale Data ii) Binarize Data iii)Standardize Data

Aim:

To Perform following data preprocessing tasks using Python i) Rescale Data ii) Binarize Data iii)Standardize data

Normalization:

Normalization is used to scale the data of an attribute so that it falls in a smaller range, such as -1.0 to 1.0 or 0.0 to 1.0. It is generally useful for classification algorithms.

Min-Max Normalization :

In this technique of knowledge normalization, a linear transformation is performed on the first data. Minimum and maximum value from data is fetched and each value is replaced according to the following formula.

Min-Max Normalization preserves the relationships among the original data values. It will encounter an out-of-bounds error if a future input case for normalization falls outside the first data range for A. The formula is given below

$$V' = V - min(A)|max(A) - min(A)(new_max(A) - new_min(A)) + new_min(A)$$

Where A is the attribute data represent as follows.
Min(A) - It is the minimum absolute value A.
Max(A) - It is the maximum absolute value A.
v' - It is the new value of each attribute data.
v - It is the old value of each attribute data.
new_max(A), new_min(A) is the max and min value within the range
(i.e boundary value of range required) respectively.

Example :

Here, we will discuss an example as follows.
Normalize the following group of data –
1000,2000,3000,9000
using min-max normalization by setting min:0 and max:1
Solution –
As given in question
here,new_max(A)=1
max=1 new_min(A)=0,

min=0
max(A)=9000
as the maximum data among
1000,2000,3000,9000 is 9000
min(A)=1000
as the minimum data among
1000,2000,3000,9000 is 1000

Case-1:
normalizing 1000 –
v = 1000 ,
putting all values in the formula,we get

$$v' = \frac{(1000-1000) \times (1-0)}{9000-1000} + 0 = 0$$

Case-2:
normalizing 2000 –
v = 2000,
putting all values in the formula,we get

$$v' = \frac{(2000-1000) \times (1-0)}{9000-1000} + 0 = 0.125$$

Case-3:
normalizing 3000 –
v=3000,
putting all values in the formula,we get

$$v' = \frac{(3000-1000) \times (1-0)}{9000-1000} + 0 = 0.25$$

Case-4:
normalizing 9000 –
v=9000,
putting all values in the formula, we get

$$v' = \frac{(9000-1000) \times (1-0)}{9000-1000} + 0 = 1$$

Outcome :
Hence, the normalized values of 1000,2000,3000,9000 are 0, 0.125, .25, 1.

PROGRAM:

```
from numpy import asarray
from sklearn.preprocessing import MinMaxScaler

data = asarray([[100,150],[800,500],[500,750],[880,600],[400,100]])
print(data)
scaler = MinMaxScaler()
scaled = scaler.fit_transform(data)
print(scaled)
```

```
[[100 150]
 [800 500]
 [500 750]
 [880 600]
 [400 100]]
[[0.         0.07692308]
 [0.8974359  0.61538462]
 [0.51282051 1.        ]
 [1.         0.76923077]
 [0.38461538 0.        ]]
```

```
[3] from sklearn import datasets
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import MinMaxScaler
```

```
[4] iris = datasets.load_iris()
    X = iris.data
    Y = iris.target
```

```
[5] print(X)
```

```
 [5.8 2.6 4.  1.2]
 [5.  2.3 3.3 1. ]
 [5.6 2.7 4.2 1.3]
 [5.7 3.  4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
 [5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
 [6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
```

```
[7] print(Y)

    [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
     2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
     2 2]

[8] X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state = 1,stratify = Y)

[9] mmScaler = MinMaxScaler()
    X_train_norm = mmScaler.fit_transform(X_train)
    X_test_norm = mmScaler.transform(X_train)

[11] print(X_test_norm)
     [0.19444444 0.54545455 0.03389831 0.04166667]
     [0.66666667 0.5        0.77966102 0.95833333]
     [0.91666667 0.45454545 0.94915254 0.83333333]
     [0.41666667 0.90909091 0.03389831 0.04166667]
     [0.80555556 0.45454545 0.81355932 0.625     ]
     [0.63888889 0.40909091 0.61016949 0.5       ]
     [0.19444444 0.13636364 0.38983051 0.375     ]
     [0.25       0.31818182 0.49152542 0.54166667]
     [0.11111111 0.54545455 0.05084746 0.04166667]
     [0.5        0.36363636 0.62711864 0.45833333]
```

ii) Binarize data

sklearn.preprocessing
Binarizer() is a method which belongs to preprocessing module.It plays a key role in the discretization of continuous feature values

Example #1:

A continuous data of pixels values of an 8-bit grayscale image have values ranging between 0 (black) and 255 (white) and one needs it to be black and white.
So, using Binarizer() one can set a threshold converting pixel values from 0 – 127 to 0 and 128 – 255 as 1.

Syntax:

sklearn.preprocessing.Binarizier(th
reshold, copy)

Parameters :

threshold :[float, optional] Values less than or equal to threshold is mapped to 0, else to 1.
By default threshold value is 0.0.
copy :[boolean, optional] If set to False, it avoids a copy.
By default it is True.

PROGRAM

```
[1]  from sklearn.preprocessing import Binarizer
     import pandas
     import numpy as np
     url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"

[6]  col_name = ['preg','plasma','pres','skin','test','BMI','pedi','age','class']

[7]  print(col_name)

     ['preg', 'plasma', 'pres', 'skin', 'test', 'BMI', 'pedi', 'age', 'class']

[12] data = pandas.read_csv(url , names = col_name) #dataset is converted to data frame

[10] print(data)

          preg  plasma  pres  skin  test   BMI   pedi  age  class
     0       6     148    72    35     0  33.6  0.627   50      1
     1       1      85    66    29     0  26.6  0.351   31      0
     2       8     183    64     0     0  23.3  0.672   32      1
     3       1      89    66    23    94  28.1  0.167   21      0
     4       0     137    40    35   168  43.1  2.288   33      1
     ..    ...     ...   ...   ...   ...   ...    ...  ...    ...
     763    10     101    76    48   180  32.9  0.171   63      0
     764     2     122    70    27     0  36.8  0.340   27      0
     765     5     121    72    23   112  26.2  0.245   30      0
     766     1     126    60     0     0  30.1  0.349   47      1
     767     1      93    70    31     0  30.4  0.315   23      0

     [768 rows x 9 columns]

[11] array = data.values

[14] array

     array([[  6.   , 148.   ,  72.   , ...,   0.627,  50.   ,   1.   ],
            [  1.   ,  85.   ,  66.   , ...,   0.351,  31.   ,   0.   ],
            [  8.   , 183.   ,  64.   , ...,   0.672,  32.   ,   1.   ],
            ...,
            [  5.   , 121.   ,  72.   , ...,   0.245,  30.   ,   0.   ],
            [  1.   , 126.   ,  60.   , ...,   0.349,  47.   ,   1.   ],
            [  1.   ,  93.   ,  70.   , ...,   0.315,  23.   ,   0.   ]])
```

```
[15]  X = array[:,0:8]
      Y = array[:,8]
```

```
[16]  print(X)
```

```
[[  6.    148.     72.    ...  33.6     0.627  50.    ]
 [  1.     85.     66.    ...  26.6     0.351  31.    ]
 [  8.    183.     64.    ...  23.3     0.672  32.    ]
 ...
 [  5.    121.     72.    ...  26.2     0.245  30.    ]
 [  1.    126.     60.    ...  30.1     0.349  47.    ]
 [  1.     93.     70.    ...  30.4     0.315  23.    ]]
```

```
[18]  print(Y) #original outcome
```

```
[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1.
 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0.
 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0.
 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0.
 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0.
 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0.
 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1.
 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0.
 1. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0.
 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1.
 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
```

```
[19]  binarizer = Binarizer(threshold = 0.0).fit(X)
      binaryX = binarizer.transform(X)
```

```
[22]  print(binaryX[0:10,:]) #binarised data outcome converted to 0,1
```

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

iii) Standardise data

Data standardization is **the process of rescaling the attributes so that they have mean as 0 and variance as 1**. The ultimate goal to perform standardization is to bring down all the features to a common scale without distorting the differences in the range of the values.

```
[1]  from sklearn.preprocessing import StandardScaler
     import pandas
     import numpy as np
     url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
```

```
[2]  col_name = ['preg','plasma','pres','skin','test','BMI','pedi','age','class']
```

```
[3]  print(col_name)

     ['preg', 'plasma', 'pres', 'skin', 'test', 'BMI', 'pedi', 'age', 'class']
```

```
[4]  data = pandas.read_csv(url , names = col_name)
```

```
[5]  print(data)

         preg  plasma  pres  skin  test   BMI   pedi  age  class
     0      6     148    72    35     0  33.6  0.627   50      1
     1      1      85    66    29     0  26.6  0.351   31      0
     2      8     183    64     0     0  23.3  0.672   32      1
     3      1      89    66    23    94  28.1  0.167   21      0
     4      0     137    40    35   168  43.1  2.288   33      1
     ..   ...     ...   ...   ...   ...   ...    ...  ...    ...
     763   10     101    76    48   180  32.9  0.171   63      0
     764    2     122    70    27     0  36.8  0.340   27      0
     765    5     121    72    23   112  26.2  0.245   30      0
     766    1     126    60     0     0  30.1  0.349   47      1
     767    1      93    70    31     0  30.4  0.315   23      0

     [768 rows x 9 columns]
```

```
[6]  array = data.values
```

```
[7]  array

     array([[  6.   , 148.   ,  72.   , ...,   0.627,  50.   ,   1.   ],
            [  1.   ,  85.   ,  66.   , ...,   0.351,  31.   ,   0.   ],
            [  8.   , 183.   ,  64.   , ...,   0.672,  32.   ,   1.   ],
            ...,
            [  5.   , 121.   ,  72.   , ...,   0.245,  30.   ,   0.   ],
            [  1.   , 126.   ,  60.   , ...,   0.349,  47.   ,   1.   ],
            [  1.   ,  93.   ,  70.   , ...,   0.315,  23.   ,   0.   ]])
```

```
[8]  X = array[:,0:8]
     Y = array[:,8]
```

```
[9]  print(X)

     [[  6.    148.     72.    ...  33.6     0.627  50.   ]
      [  1.     85.     66.    ...  26.6     0.351  31.   ]
      [  8.    183.     64.    ...  23.3     0.672  32.   ]
      ...
      [  5.    121.     72.    ...  26.2     0.245  30.   ]
      [  1.    126.     60.    ...  30.1     0.349  47.   ]
      [  1.     93.     70.    ...  30.4     0.315  23.   ]]
```

```
print(Y)
```

```
[1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1.
 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0.
 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0.
 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 0.
 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0.
 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0.
 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1.
 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0.
 1. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0.
 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1.
 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 0.
 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 1. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0.
 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0.
 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 1.
```

```
[12] scaler = StandardScaler().fit(X)
     rescaledX = scaler.transform(X)
```

```
print(rescaledX[0:10,:])
```

```
[[ 0.63994726  0.84832379  0.14964075  0.90726993 -0.69289057  0.20401277
   0.46849198  1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575  0.53090156 -0.69289057 -0.68442195
  -0.36506078 -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 -1.28821221 -0.69289057 -1.10325546
   0.60439732 -0.10558415]
 [-0.84488505 -0.99820778 -0.16054575  0.15453319  0.12330164 -0.49404308
  -0.92076261 -1.04154944]
 [-1.14185152  0.5040552  -1.50468724  0.90726993  0.76583594  1.4097456
   5.4849091  -0.0204964 ]
 [ 0.3429808  -0.15318486  0.25303625 -1.28821221 -0.69289057 -0.81134119
  -0.81807858 -0.27575966]
 [-0.25095213 -1.34247638 -0.98770975  0.71908574  0.07120427 -0.12597727
  -0.676133   -0.61611067]
 [ 1.82781311 -0.184482   -3.57259724 -1.28821221 -0.69289057  0.41977549
  -1.02042653 -0.36084741]
 [-0.54791859  2.38188392  0.04624525  1.53455054  4.02192191 -0.18943689
  -0.94794368  1.68125866]
 [ 1.23388019  0.12848945  1.39038675 -1.28821221 -0.69289057 -4.06047387
  -0.7244549   1.76634642]]
```