ARYABHATTA COLLEGE

(UNIVERSITY OF DELHI)

DATA MINING

PRACTICAL FILE

SUBMITTED TO-
SONAL LINDA MA'AM
(*DEPARTMENT OF COMPUTER SCIENCE*)

SUBMITTED BY-
HARTIK SINGHAL
B.SC COMPUTER SCIENCE
ROLL NUMBER – CSC/20/61
EXAM ROLL NO.- 20059570048

Table of Contents

Q.1 Create a file “people.txt” with the following data: i) Read the data from the file “people.txt”. ii) Create a ruleset E that contain rules to check for the following conditions:

1. The age should be in the range 0-150.
2. The age should be greater than yearsmarried.
3. The status should be married or single or widowed.
4. If age is less than 18 the agegroup should be child, if age is between 18 and 65 the agegroup should be adult, if age is more than 65 the agegroup should be elderly. iii) Check whether ruleset E is violated by the data in the file people.txt. iv) Summarize the results obtained in part (iii) v) Visualize the results obtained in part (iii)

In [70]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import rulesetpracticall as rsl
```

In [66]:
```
#(i). Reading data from file 'people.txt'
df =pd.read_table('people.txt',delim_whitespace='True')
```

In [67]:
```
print(df)
```

```
   Age agegroup  height   status  yearsmarried
0   21    adult     6.0   single            -1
1    2    child     3.0  married             0
2   18    adult     5.7  married            20
3  221  elderly     5.0  widowed             2
4   34    child    -7.0  married             3
```

In [51]:
```
#(ii). Creating ruleset in separate file
```

In [71]:
```
E  = {'Rule1': rsl.age_range, 'Rule2': rsl.age_check, 'Rule3': rsl.status_check,'Rule4': rsl.age_group}
result =[]
for i in E.keys():
    result.append(E[i](df))
```

In [ ]:
```
# (iii). Checking violation of ruleset and storing in List named result
```

In [72]:
```
result
```

Out[72]:
```
[0     True
1      True
2      True
3     False
4      True
Name: age_range, dtype: bool,
0      True
1      True
2     False
3      True
4      True
Name: age_check, dtype: bool,
0     True
1     True
2     True
3     True
4     True
Name: status_check, dtype: bool,
0      True
1      True
2      True
3      True
4     False
Name: age_group, dtype: bool]
```

In [ ]:
```
#(iv). Summarizing the results obtained
```

In [48]:
```
result=pd.DataFrame(result)
result
```

Out[48]:

	0	1	2	3	4
Rules					
Rule1	True	True	True	False	True
Rule2	True	True	False	True	True
Rule3	True	True	True	True	True
Rule4	True	True	True	True	False

In [49]:
```
j=result.reset_index()
j
```

Out[49]:

Rules	0	1	2	3	4

	Rules	0	1	2	3	4
0	Rule1	True	True	True	False	True
1	Rule2	True	True	False	True	True
2	Rule3	True	True	True	True	True
3	Rule4	True	True	True	True	False

In [50]:
```
j.groupby(['Rules'J).sum().sum(axis=l)
# Rulel,2 and 4 are violated
```

Out[50]:
```
Rules
Rulel      4
Rule2      4
Rule3      5
Rule4      4
dtype: int64
```

In [51]:
```
result
```

Out[51]:

Rules	0	1	2	3	4
Rule1	True	True	True	False	True
Rule2	True	True	False	True	True
Rule3	True	True	True	True	True
Rule4	True	True	True	True	False

In [52]:
```
summarised=result.describe()
summarised
```

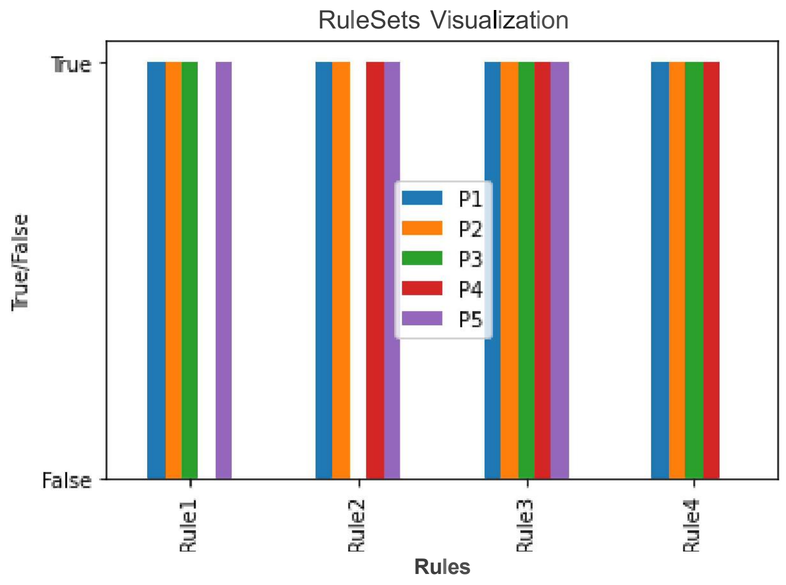
Out[52]:

	0	1	2	3	4
count	4	4	4	4	4
unique			2	2	2
top	True	True	True	True	True

	0	1	2	3	4
freq	4	4	3	3	3

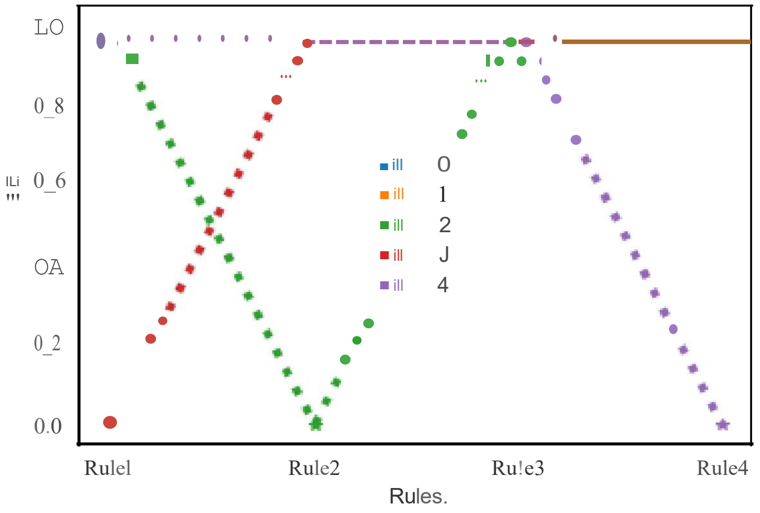
```
In [53]:  #(v). Visualizing the results obtained
           result.astype(int).plot(kind ='bar', title='RuleSets Visualization')
           plt.xlabel('Rules')
           plt.ylabel('True/False')
           plt.legend(['P1','P2','P3','P4','PS'],loc='center')
           plt.yticks([0,1],[False,True])
           #Bar is absent foe the person where value is false
```

```
out[53]: ([<matplotlib.axis.YTick at 0x13c43cd36d0>,
            <matplotlib.axis.YTick at 0x13c43cd32b0>],
           [Text(0, 0, 'False'), Text(0, 1, 'True')])
```



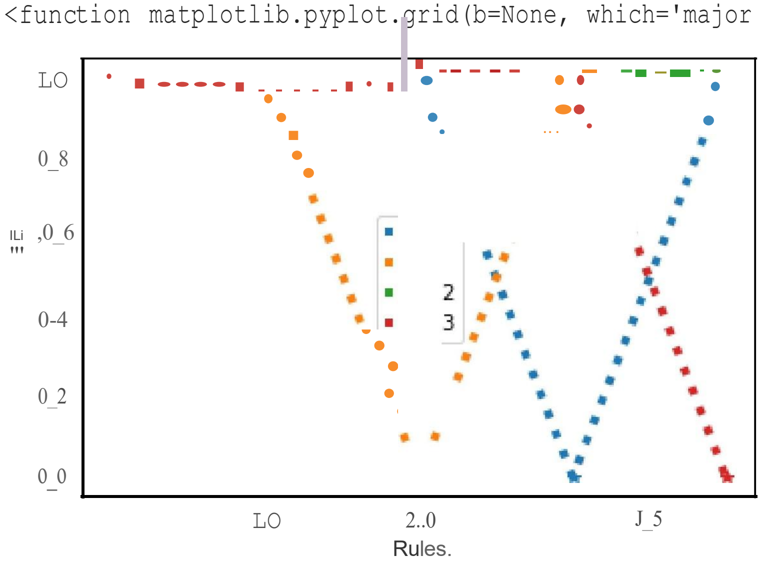
```
In [61]:  plt.plot(result,'*:',linewidth=4)
           plt.legend(('0','1','2','3','4'),loc='center')
           plt.xlabel('Rules')
           plt.ylabel('True/False')
           plt.grid
           #Line going down for false value i.e; violated result
```

```
out[61]: <function matplotlib.pyplot.grid(b=None, which='major', axis='both', **kwargs)>
```



```
In [62]:  plt.plot(result.T,'*:',linewidth=4)
          plt.legend(('0','1','2','3','4'),loc='center')
          plt.xlabel('Rules')
          plt.ylabel('True/False')
          plt.grid
```

```
out[62]: <function matplotlib.pyplot.grid(b=None, which='major ', axis='both', **kwargs)>
```



Ruleset-1

```
In [ ]:
#Rule 1
def age_range(x):
    rule1 = lambda y: yin range(151)
    return x['Age'].apply(rule1).rename('age_range')

#Rule 2
def age_check(x):
    rule2= lambda y: y[0] > y[1]
    return  x[['Age','yearsmarried']].apply(rule2,axis=1).rename('age_check')

#Rule3
def status_check(x):
    rule3= lambda y:y in ['single','married','widowed']
    return x['status'].apply(rule3).rename('status_check')

# Rule 4
def age_group(x):
    def rule4(y):
        if(y[0] in range(18) and y[1]=="child") or (y[0] in range(18,66) and y[1]=="adult") or (y[0]>65 and y[1]=="elderly"):
            return True
        else:
            return False
    return  (x[["Age","agegroup"]].apply(rule4,axis=1).rename('age_group'))
```

Q2. Perform the following preprocessing tasks on the dirty_iris datasetii. i) Calculate the number and percentage of observations that are complete. ii) Replace all the special values in data with NA. iii) Define these rules in a separate text file and read them. (Use editfile function in R (package editrules). Use similar function in Python). Print the resulting constraint object. - Species should be one of the following values: setosa, versicolor or virginica. - All measured numerical properties of an iris should be positive. - The petal length of an iris is at least 2 times its petal width. - The sepal length of an iris cannot exceed 30 cm. - The sepals of an iris are longer than its petals. iv) Determine how often each rule is broken (violatedEdits). Also summarize and plot the result. v) Find outliers in sepal length using boxplot and boxplot.stats

In [107...
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import rulesetpractical2 as rs2
```

In [89]:
```
df = pd.read_csv('dirty_iris.csv')
```

In [90]:
```
df.head(S)
```

Out[90]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.4	3.2	4.5	1.5	versicolor
1	6.3	3.3	6.0	2.5	virginica
2	6.2	NaN	5.4	2.3	virginica
3	5.0	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1.0	versicolor

i) Calculate the number and percentage of observations that are complete

In [91]:
```
null_values=df.isnull().sum()
 null values
 # Number of null values in each attribute is given below
```

Out[91]:
```
Sepal.Length    10
Sepal.Width     17
Petal.Length    19
Petal.Width   12
Species        0
dtype: int64
```

```
In [92]:  df.shape # there are 150 records(rows) in  total
```

```
Out[92]: (150, 5)
```

```
In [93]:  #Calculating percentage of complete values in each attribute
           a=0
           for i in null_values:
               print("Percentage of complete observations in",null_values.index[a],(150-i)/150 *100)
               a=a+1
```

```
Percentage of complete observations in Sepal.Length 93.33333333333333
Percentage of complete observations in Sepal.Width 88.66666666666667
Percentage of complete observations in Petal.Length 87.33333333333333
Percentage of complete observations in Petal.Width 92.0
Percentage of complete observations in Species 100.0
```

```
In [94]:  #counting null records(rows)
           df.isnull().sum(axis=1).sum()
           # Thus there are a total of 58 records with null values
```

```
Out[94]: 58
```

```
In [95]:   #percentage of complete observations
           (150-58)/150 *100
```

```
Out[95]: 61.33333333333333
```

ii) Replace all the special values in data with NA.

```
In [108...... na_values=['Inf']
           df.replace(na_values, 'NA')
```

Out[108...

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.4	3.2	4.5	1.5	versicolor
	6.3	3.3	6	2.5	v1rg1nica

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
2	6.2	NA	5.4	2.3	virginica
3	5	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1	versicolor
145	6.7	3.1	5.6	2.4	virg1nica
146	5.6	3	4.5	1.5	versicolor
147	5.2	3.5	1.5	0.2	setosa
148	6.4	3.1	NA	1.8	v1rg1nica
149	5.8	2.6	4	NA	versicolor

150 rows x 5 columns

iii) Define these rules in a separate text file and read them.

In [67]:
```
rules = {"species_check" : rs2.species_check, "all_positive" : rs2.positive_check,
         "check_petal_length" : rs2.check_petal, "sepal_length_check" : rs2.sepal_check,
         "sepal_petal_check" : rs2.sepal_petal_check}
```

iv)Determine how often each rule is broken (violatedEdits). Also summarize and plot the result.

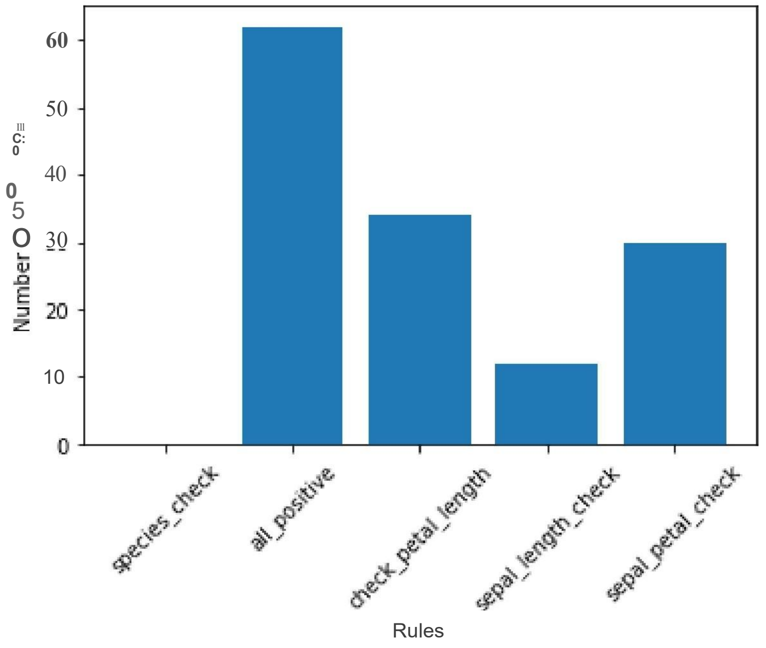
plt.bar(rules.keys(),result) plt.xticks(rotation = 45) plt.xlabel('Rules') plt.ylabel('Number of Violations') pit.show()

In [84]:
```
result=[]
for i in rules.keys()
  result.append(rules[i](df))
result=np.array(result)
print("Total no of violations       ",result.sum())
```

```
No Violation
```

```
Violation : Non-positive values present
62 violations
Violation : Petal Length is less than twice of Petal Width in some places
34 violations
Violation : Sepal Length is greater than 30 cm in some places
12 violations
Violation  Sepal length is greater than petal length in some places
30 violations
Total no of violations :   138
```

```
In [69]: plt.bar(rules.keys(),result)
         plt.xticks(rotation = 45)
         plt.xlabel('Rules')
         plt.ylabel('Number of Violations')
         pH.show()
```

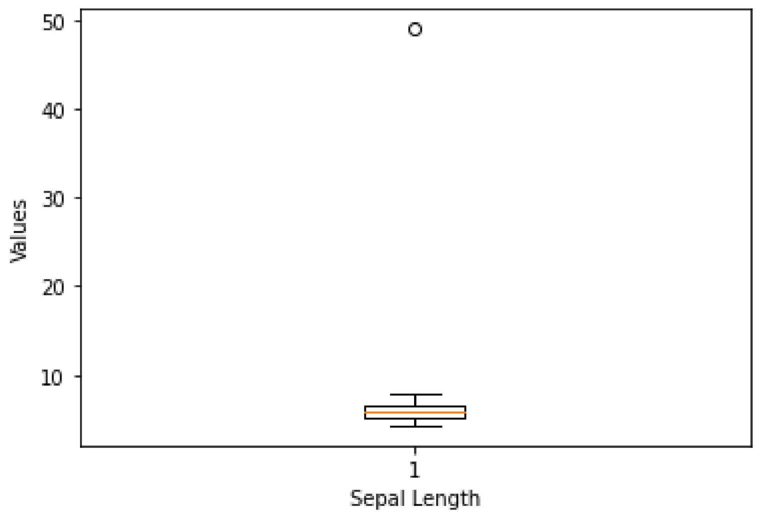


v) Find outliers in sepal length using boxplot and boxplot.stats

```
In [76]: temp=df.dropna()
```

```
In [79]: plt.boxplot(temp[['Sepal.Length']])
```

```
plt.xlabel('Sepal Length')
plt.ylabel('Values')
plt.show()
```



In [ ]:
```
# Outlier value is Se
```

Ruleset - 2

In [ ]:
```
import numpy as np

def species_check(df)
  species= set(["setosa","versicolor","virginica"J)
  func = lambda r : r in species
  x = np.array([func(xi) for xi in df["Species"]])
  if (False in x) :
    print("Violation  Invalid species name")
    print(str(len(x) - np.sum(x)) +" violations")
  else :
    print("No Violation")
  return (len(x) - np.sum(x))


def positive_check(df) :
  func = lambda r : r>0
  a= np.array([func(df[xi]) for  xi  in  df.columns[:-1]))
```

```
    a= a.reshape(a.shape[0]*a.shape[1])
    if (False in a) :
      print("Violation  Non-positive values present")
      print(str(len(a) - np.sum(a)) +" violations")
    else :
      print("No Violation")
    return (len(a) - np.sum(a))


def check_petal(df) :
    a= np.array(df["Petal.Length"]>(2*df["Petal.Width"]))
    if (False in a) :
      print("Violation   Petal Length is less than twice of Petal Width in some places")
      print(str(len(a) - np.sum(a)) +" violations")
    else :
      print("No Violation")
    return (len(a) - np.sum(a))

def sepal_check(df) :
    a = np.array(df["Sepal.Length"]<=30)
    if (False in a) :
      print("Violation   Sepal Length is greater than 30 cm in some places")
      print(str(len(a) - np.sum(a)) +" violations")
    else :
      print("No Violation")
    return (len(a) - np.sum(a))


def sepal_petal_check(df) :
    a= np.array(df["Sepal.Length"]>df["Petal.Length"])
    if (False in a) :
      print("Violation   Sepal length is greater than petal length in some places")
      print(str(len(a) - np.sum(a)) +" violations")
    else :
      print("No Violation")
    return (len(a) - np.sum(a))
```

Q3. Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
dfwine = pd.read_csv('wine.data')
```

In [4]:
```
dfwine.head()
```

Out [4]:

	1	14.23	1.71	2.43	15.6	127	2.8	3.06	.28	2.29	5.64	1.04	3.92	1065
0		13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
1		13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
2		14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
3		13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
4		14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450

In [5]:
```
dfwine.describe()
```

Out[5]:

	1	14.23	1.71	2.43	15.6	127	2.8	3.06	.28	2.29	5.64	1.04	
count	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	17
mean	1.943503	12.993672	2.339887	2.366158	19.516949	99.587571	2.292260	2.023446	0.362316	1.586949	5.054802	0.956983	
std	0.773991	0.808808	1.119314	0.275080	3.336071	14.174018	0.626465	0.998658	0.124653	0.571545	2.324446	0.229135	
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	
25%	1.000000	12.360000	1.600000	2.210000	17.200000	88.000000	1.740000	1.200000	0.270000	1.250000	3.210000	0.780000	
50%	2.000000	13.050000	1.870000	2.360000	19.500000	98.000000	2.350000	2.130000	0.340000	1.550000	4.680000	0.960000	
75%	3.000000	13.670000	3.100000	2.560000	21.500000	107.000000	2.800000	2.860000	0.440000	1.950000	6.200000	1.120000	
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	

In [6]:
```
dfiris = pd.read_csv('iris.data')
```

In [11]:
```
dfiris.head(5)
```

Out[11]:

	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

In [7]:
```
dfiris.describe()
```

Out[7]:

	5.1	3.5	1.4	0.2
count	149.000000	149.000000	149.000000	149.000000
mean	5.848322	3.051007	3.774497	1.205369
std	0.828594	0.433499	1.759651	0.761292
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [8]:
```
x = dfiris.iloc[:,:-1]
```

In [12]:
```
y = dfiris.iloc[:,-1]
```

In [15]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
x_train,x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
```

In [16]:
```
#Standardizing Iris dataframe
std_s = StandardScaler() #making object of standard scaler
std_s.fit(x_train)
x_train_std = std_s.transform(x_train)
x_test_std = std_s.transform(x_test)
```

In [18]:
```
print(x_train[0:5])
print(x_train_std[0:5])
print(x_test[0:5])
print(x_test_std[0:5])
```

```
     5.1  3.5  1.4  0.2
134  7.7  3.0  6.1  2.3
101  7.1  3.0  5.9  2.1
131  6.4  2.8  5.6  2.2
125  6.2  2.8  4.8  1.8
26   5.2  3.5  1.5  0.2
[[ 2.16839831 -0.07900966  1.29326088  1.42702129]
 [ 1.44559887 -0.07900966  1.17741742  1.15725873]
 [ 0.60233286 -0.5376511   1.00365222  1.29214001]
 [ 0.36139972 -0.5376511   0.54027835  0.75261488]
 [-0.84326601  1.06759395 -1.37113884 -1.40548563]]
     5.1  3.5  1.4  0.2
82   6.0  2.7  5.1  1.6
79   5.5  2.4  3.8  1.1
126  6.1  3.0  4.9  1.8
27   5.2  3.4  1.4  0.2
30   5.4  3.4  1.5  0.4
[[ 0.12046657 -0.76697182  0.71404355  0.48285232]
 [-0.48186629 -1.45493399 -0.03893898 -0.19155409]
 [ 0.24093315 -0.07900966  0.59820008  0.75261488]
 [-0.84326601  0.83827323 -1.42906058 -1.40548563]
 [-0.60233286  0.83827323 -1.37113884 -1.13572306]]
```

In [29]:
```
#Standardising all records of independent variable simultaneously
scaled dfiris= std_s.fit_transform(x)
scaled dfiris
```

Out[29]:
```
array([[-1.1483555 , -0.11805969, -1.35396443, -1.32506301],
       [-1.3905423 ,  0.34485856, -1.41098555, -1.32506301],
       [-1.51163569,  0.11339944, -1.29694332, -1.32506301],
       [-1.02726211,  1.27069504, -1.35396443, -1.32506301],
       [-0.54288852,  1.9650724 , -1.18290109, -1.0614657 ],
[-1.51163569,  0.8077768,  -1.35396443, -1.19326436],
[-1.02726211,  0.8077768,  -1.29694332, -1.32506301],
[-1.75382249, -0.34951881, -1.35396443, -1.32506301],
[-1.1483555 ,  0.11339944, -1.29694332, -1.45686167],
[-0.54288852,  1.50215416, -1.29694332, -1.32506301],
[-1.2694489,   0.8077768,  -1.23992221, -1.32506301],
[-1.2694489,  -0.11805969, -1.35396443, -1.45686167],
[-1.87491588, -0.11805969, -1.52502777, -1.45686167],
[-0.05851493,  2.19653152, -1.46800666, -1.32506301],
[-0.17960833,  3.122368   , -1.29694332, -1.0614657 ],
[-0.54288852,  1.9650724,  -1.41098555, -1.0614657 ],
[-0.90616871,  1.03923592, -1.35396443, -1.19326436],
[-0.17960833,  1.73361328, -1.18290109, -1.19326436],
[-0.90616871,  1.73361328, -1.29694332, -1.19326436],
[-0.54288852,  0.8077768,  -1.18290109, -1.32506301],
[-0.90616871,  1.50215416, -1.29694332, -1.0614657 ],
[-1.51163569,  1.27069504, -1.58204889, -1.32506301],
[-0.90616871,  0.57631768, -1.18290109, -0.92966704],
[-1.2694489,   0.8077768,  -1.06885886, -1.32506301],
[-1.02726211, -0.11805969, -1.23992221, -1.32506301],
[-1.02726211,  0.8077768,  -1.23992221, -1.0614657 ],
[-0.78507531,  1.03923592, -1.29694332, -1.32506301],
[-0.78507531,  0.8077768,  -1.35396443, -1.32506301],
[-1.3905423 ,  0.34485856, -1.23992221, -1.32506301],
[-1.2694489,   0.11339944, -1.23992221, -1.32506301],
[-0.54288852,  0.8077768,  -1.29694332, -1.0614657 ],
[-0.78507531,  2.42799064, -1.29694332, -1.45686167],
[-0.42179512,  2.65944976, -1.35396443, -1.32506301],
[-1.1483555 ,  0.11339944, -1.29694332, -1.45686167],
[-1.02726211,  0.34485856, -1.46800666, -1.32506301],
[-0.42179512,  1.03923592, -1.41098555, -1.32506301],
[-1.1483555 ,  0.11339944, -1.29694332, -1.45686167],
[-1.75382249, -0.11805969, -1.41098555, -1.32506301],
[-0.90616871,  0.8077768,  -1.29694332, -1.32506301],
[-1.02726211,  1.03923592, -1.41098555, -1.19326436],
[-1.63272909, -1.73827353, -1.41098555, -1.19326436],
[-1.75382249,  0.34485856, -1.41098555, -1.32506301],
[-1.02726211,  1.03923592, -1.23992221, -0.79786838],
[-0.90616871,  1.73361328, -1.06885886, -1.0614657 ],
[-1.2694489 , -0.11805969, -1.35396443, -1.19326436],
[-0.90616871,  1.73361328, -1.23992221, -1.32506301],
[-1.51163569,  0.34485856, -1.35396443, -1.32506301],
[-0.66398191,  1.50215416, -1.29694332, -1.32506301],
[-1.02726211,  0.57631768, -1.35396443, -1.32506301],
[ 1.39460583,  0.34485856,  0.52773232,  0.25652088],
[ 0.66804545,  0.34485856,  0.41369009,  0.38831953],
[ 1.27351244,  0.11339944,  0.64177455,  0.38831953],
[-0.42179512, -1.73827353,  0.12858453, 0.12472222],
[ 0.78913885, -0.58097793,  0.47071121, 0.38831953],
[-0.17960833, -0.58097793,  0.41369009, 0.12472222],
[ 0.54695205,  0.57631768,  0.52773232, 0.52011819],
[-1.1483555 , -1.50681441, -0.27056327, -0.27067375],
[ 0.91023225, -0.34951881,  0.47071121, 0.12472222],
[-0.78507531, -0.81243705,  0.07156341, 0.25652088],
[-1.02726211,  -2.43265089, -0.15652104, -0.27067375],
[ 0.06257847, -0.11805969,  0.24262675, 0.38831953],
[ 0.18367186, -1.96973265,  0.12858453, -0.27067375],
[ 0.30476526, -0.34951881,  0.52773232, 0.25652088],
[-0.30070172, -0.34951881, -0.09949993, 0.12472222],
[ 1.03132564,  0.11339944,  0.35666898, 0.25652088],
[-0.30070172, -0.11805969,  0.41369009, 0.38831953],
[-0.05851493, -0.81243705,  0.18560564, -0.27067375],
[ 0.42585866, -1.96973265,  0.41369009, 0.38831953],
[-0.30070172, -1.27535529,  0.07156341,  -0.1388751 ],
[ 0.06257847,  0.34485856,  0.58475344, 0.78371551],
[ 0.30476526, -0.58097793,  0.12858453, 0.12472222],
[ 0.54695205, -1.27535529,  0.64177455, 0.38831953],
[ 0.30476526, -0.58097793,  0.52773232, -0.00707644],
[ 0.66804545, -0.34951881,  0.29964787, 0.12472222],
[ 0.91023225, -0.11805969,  0.35666898, 0.25652088],
[ 1.15241904, -0.58097793,  0.58475344, 0.25652088],
[ 1.03132564, -0.11805969,  0.69879566, 0.65191685],
[ 0.18367186, -0.34951881,  0.41369009, 0.38831953],
[-0.17960833, -1.04389617, -0.15652104, -0.27067375],
[-0.42179512, -1.50681441,  0.0145423,  -0.1388751 ],
[-0.42179512, -1.50681441, -0.04247882, -0.27067375],
[-0.05851493, -0.81243705,  0.07156341, -0.00707644],
[ 0.18367186, -0.81243705,  0.75581678, 0.52011819],
[-0.54288852, -0.11805969,  0.41369009, 0.38831953],
[ 0.18367186,  0.8077768,  0.41369009, 0.52011819],
[ 1.03132564,  0.11339944,  0.52773232, 0.38831953],
[ 0.54695205, -1.73827353,  0.35666898, 0.12472222],
[-0.30070172, -0.11805969,  0.18560564, 0.12472222],
[-0.42179512, -1.27535529,  0.12858453, 0.12472222],
[-0.42179512, -1.04389617,  0.35666898, -0.00707644],
[ 0.30476526, -0.11805969,  0.47071121, 0.25652088],
[-0.05851493, -1.04389617,  0.12858453, -0.00707644],
[-1.02726211,  -1.73827353, -0.27056327, -0.27067375],
[-0.30070172, -0.81243705,  0.24262675, 0.12472222],
[-0.17960833, -0.11805969,  0.24262675, -0.00707644],
[-0.17960833, -0.34951881,  0.24262675, 0.12472222],
[ 0.42585866, -0.34951881,  0.29964787, 0.12472222],
[-0.90616871, -1.27535529, -0.44162661,  -0.1388751 ],
[-0.17960833, -0.58097793,  0.18560564, 0.12472222],
[ 0.54695205, 0.57631768,  1.2690068, 1.70630611],
[-0.05851493, -0.81243705, 0.75581678, 0.91551417],
[ 1.51569923, -0.11805969, 1.21198569, 1.17911148],
[ 0.54695205, -0.34951881, 1.04092235, 0.78371551],
[ 0.78913885, -0.11805969, 1.15496457, 1.31091014],
[ 2.12116622, -0.11805969, 1.61113348, 1.17911148],
[-1.1483555 , -1.27535529, 0.41369009, 0.65191685],
[ 1.75788602, -0.34951881, 1.44007014, 0.78371551],
[ 1.03132564, -1.27535529, 1.15496457, 0.78371551],
[ 1.63679263, 1.27069504, 1.32602791, 1.70630611],
[ 0.78913885, 0.34485856, 0.75581678, 1.04731282],
[ 0.66804545, -0.81243705, 0.869859  ,  0.91551417],
[ 1.15241904, -0.11805969, 0.98390123, 1.17911148],
[-0.17960833, -1.27535529, 0.69879566, 1.04731282],
[-0.05851493, -0.58097793, 0.75581678, 1.57450745],
[ 0.66804545, 0.34485856, 0.869859 ,  1.4427088 ],
[ 0.78913885, -0.11805969, 0.98390123, 0.78371551],
[ 2.24225961, 1.73361328, 1.6681546 , 1.31091014],
[ 2.24225961, -1.04389617, 1.78219682, 1.4427088 ],
[ 0.18367186, -1.96973265, 0.69879566, 0.38831953],
[ 1.27351244, 0.34485856, 1.09794346, 1.4427088 ],
[-0.30070172, -0.58097793, 0.64177455, 1.04731282],
[ 2.24225961, -0.58097793, 1.6681546 , 1.04731282],
[ 0.54695205, -0.81243705, 0.64177455, 0.78371551],
[ 1.03132564, 0.57631768, 1.09794346, 1.17911148],
[ 1.63679263, 0.34485856, 1.2690068, 0.78371551],
[ 0.42585866, -0.58097793, 0.58475344, 0.78371551],
[ 0.30476526, -0.11805969, 0.64177455, 0.78371551],
[ 0.66804545, -0.58097793, 1.04092235, 1.17911148],
[ 1.63679263, -0.11805969, 1.15496457, 0.52011819],
[ 1.87897942, -0.58097793, 1.32602791, 0.91551417],
[ 2.48444641, 1.73361328, 1.49709126, 1.04731282],
[ 0.66804545, -0.58097793, 1.04092235, 1.31091014],
[ 0.54695205, -0.58097793, 0.75581678, 0.38831953],
[ 0.30476526, -1.04389617, 1.04092235, 0.25652088],
[ 2.24225961, -0.11805969, 1.32602791, 1.4427088 ],
[ 0.54695205, 0.8077768, 1.04092235, 1.57450745],
[ 0.66804545, 0.11339944, 0.98390123, 0.78371551],
[ 0.18367186, -0.11805969, 0.58475344, 0.78371551],
[ 1.27351244, 0.11339944, 0.92688012, 1.17911148],
[ 1.03132564, 0.11339944, 1.04092235, 1.57450745],
[ 1.27351244, 0.11339944, 0.75581678, 1.4427088 ],
[-0.05851493, -0.81243705, 0.75581678, 0.91551417],
[ 1.15241904, 0.34485856, 1.21198569, 1.4427088 ],
[ 1.03132564, 0.57631768, 1.09794346, 1.70630611],
[ 1.03132564, -0.11805969, 0.81283789, 1.4427088 ],
[ 0.54695205, -1.27535529, 0.69879566, 0.91551417],
                0.78913885, -0.11805969,  0.81283789,  1.04731282],
                0.42585866,  0.8077768,   0.92688012,  1.4427088 ],
                0.06257847, -0.11805969,  0.75581678,  0.78371551]])
```

```
In [20]:  #Doing for wine DataFrame
          xwine= dfwine.iloc[:,:-1]
          ywine= dfwine.iloc[:,-1]
```

```
In [24]:  xwine.values
```

```
Out[24]: array([[ 1.  , 13.2 ,  1.78, ...,  4.38, 1.05,  3.4 ],
                [ 1.  , 13.16,  2.36, ...,  5.68, 1.03,  3.17],
                [ 1.  , 14.37,  1.95, ...,  7.8,  0.86,  3.45],
                ...,
                 3.   , 13.27,  4.28, ..., 10.2   0.59,  1.56],
                 3.   , 13.17,  2.59, ...,  9.3   0.6,   1.62],
                 3.   , 14.13,  4.1, ...,   9.2   0.61,  1.6 ]])
```

```
In [23]:  ywine.values
```

```
Out[23]:array([1050, 1185, 1480,  735, 1450, 1290, 1295, 1045, 1045, 1510, 1280,
               1320, 1150, 1547, 1310, 1280, 1130, 1680,  845,  780,  770, 1035,
               1015,  845,  830, 1195, 1285,  915, 1035, 1285, 1515,  990, 1235,
               1095,  920,  880, 1105, 1020,  760,  795, 1035, 1095,  680,  885,
               1080, 1065,  985, 1060, 1260, 1150, 1265, 1190, 1375, 1060, 1120,
                970, 1270, 1285,  520,  680,  450,  630,  420,  355,  678,  502,
                510,  750,  718,  870,  410,  472,  985,  886,  428,  392,  500,
                750,  463,  278,  714,  630,  515,  520,  450,  495,  562,  680,
                625,  480,  450,  495,  290,  345,  937,  625,  428,  660,  406,
                710,  562,  438,  415,  672,  315,  510,  488,  312,  680,  562,
                325,  607,  434,  385,  407,  495,  345,  372,  564,  625,  465,
                365,  380,  380,  378,  352,  466,  342,  580,  630,  530,  560,
                600,  650,  695,  720,  515,  580,  590,  600,  780,  520,  550,
                855,  830,  415,  625,  650,  550,  500,  480,  425,  675,  640,
                725,  480,  880,  660,  620,  520,  680,  570,  675,  615,  520,
                695,  685,  750,  630,  510,  470,  660,  740,  750,  835,  840,
                560], dtype=int64)
```

```
In [25]:  xwine_train,xwine_test, ywine_train, ywine_test  = train_test_split(x,y,test_size=0.2)
```

```
In [26]: # Standardising wine dataset
          obj2 = StandardScaler() #making object
          obj2.fit(xwine_train)
          xwine_train_std = obj2.transform(xwine_train)
          xwine_test_std = obj2.transform(xwine_test)
```

```
In [27]:   print(xwine_train[0:6])
           print(xwine_train_std[0:6])
           print(xwine_test[0:6])
           print(xwine_test_std[0:6])

                5.1  3.5  1.4  0.2
           26   5.2  3.5  1.5  0.2
           71   6.3  2.5  4.9  1.5
           51   6.9  3.1  4.9  1.5
           100  5.8  2.7  5.1  1.9
           40   4.5  2.3  1.3  0.3
           24   5.0  3.0  1.6  0.2
           [[-0.84212616 1.10757852 -1.36478869 -1.38704332]
            [ 0.50219475 -1.28446947 0.59821701  0.32545151]
            [ 1.2354607  0.15075933 0.59821701  0.32545151]
            [-0.10886021 -0.80605987 0.71368794  0.85237299]
            [-1.69760311 -1.76287906 -1.48025961 -1.25531295]
            [-1.08654815  -0.08844547 -1.30705323 -1.38704332]]
                5.1  3.5  1.4  0.2
           107  6.7  2.5  5.8  1.8
           87   5.6  3.0  4.1  1.3
           43   5.1  3.8  1.9  0.4
           129  7.4  2.8  6.1  1.9
           80   5.5  2.4  3.7  1.0
           14   5.7  4.4  1.5  0.4
           [[ 0.99103872 -1.28446947 1.11783617  0.72064262]
            [-0.35328219 -0.08844547 0.13633332  0.06199076]
            [-0.96433715  1.82519292  -1.13384684 -1.12358258]
            [ 1.84651566 -0.56685507 1.29104255  0.85237299]
            [-0.47549319 -1.52367427 -0.09460853 -0.33320035]
            [-0.2310712   3.26042171 -1.36478869 -1.12358258]]
```

```
In [30]:  #Standardising all records of independent variable simultaneously
           scaled_dfwine= obj2.fit_transform(xwine)
           scaled dfwine

Out[30]: array([[-1.22246766,  0.2558245, -0.50162433, ..., -0.29113022,
                  0.40709978,  1.13169801],
                 [-1.22246766,  0.20622873,  0.01802001, ...,  0.26972932,
                  0.31956674,  0.80457911],

                [-1.22246766,   1.70650069, -0.34931478, ...,  1.1843618,
                 -0.4244579,  1.20281081],
                ...,
                [ 1.36887097,  0.34261709,  1.73822194, ...,  2.2197948,
                 -1.60614514, -1.48525319],
                [ 1.36887097,  0.21862767,  0.22408586, ...,  1.83150742,
         -1.56237895, -1.39991783],
        [ 1.36887097,  1.40892609,  1.57695301, ...,  1.78836438,
         -1.51861275, -1.42836295]])
```

```
In [ ]:
```

Q4. Run Apriori algorithm to find frequent itemsets and association rules 1.1 Use minimum support as 50% and minimum confidence as 75% 1.2 Use minimum support as 60% and minimum confidence as 60 %

In [2]:
```
pip install mlxtend
```

```
Collecting mlxtend
  Downloading mlxtend-0.19.0-py2.py3-none-any.whl (1.3 MB)
Requirement already satisfied: setuptools in c:\users\chaud\anaconda3\lib\site-packages (from mlxtend) (50.3.1.post20201107)
Requirement already satisfied: joblib>=0.13.2 in c:\users\chaud\anaconda3\lib\site-packages (from mlxtend) (0.17.0)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\chaud\anaconda3\lib\site-packages (from mlxtend) (3.3.2)
Requirement already satisfied: scipy>=1.2.1 in c:\users\chaud\anaconda3\lib\site-packages (from mlxtend) (1.5.2)
Requirement already satisfied: scikit-learn>=0.20.3 in c:\users\chaud\anaconda3\lib\site-packages (from mlxtend) (0.23.2)
Requirement already satisfied: numpy>=1.16.2 in c:\users\chaud\anaconda3\lib\site-packages (from mlxtend) (1.19.2)
Requirement already satisfied: pandas>=0.24.2 in c:\users\chaud\anaconda3\lib\site-packages (from mlxtend) (1.1.3)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\chaud\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend)
(2020.6.20)
Requirement already satisfied: pillow>=6.2.0 in c:\users\chaud\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (8.0.
1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\chaud\anaconda3\lib\site-packages (from matplo
tlib>=3.0.0->mlxtend) (2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\chaud\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxten
d) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\chaud\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend)
(1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\chaud\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (0.10.
0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\chaud\anaconda3\lib\site-packages (from scikit-learn>=0.20.3->mlxt
end) (2.1.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\chaud\anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend) (2020.1)
Requirement already satisfied: six>=1.5 in c:\users\chaud\anaconda3\lib\site-packages (from python-dateutil>=2.1->matplotlib>=3.0.
0->mlxtend) (1.15.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.19.0
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
import pandas as pd
 from mlxtend.preprocessing import TransactionEncoder
 from mlxtend.frequent_patterns import apriori
```

In [4]:
```
 dataset  = [['Milk', 'Onion', 'Nutmeg', 'Kidney  Beans', 'Eggs', 'Yogurt'],
                 ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
                 ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
                 ['Milk', 'Unicorn', 'Corn', 'Kidney  Beans', 'Yogurt'],
                 ['Corn', 'Onion', 'Onion', 'Kidney  Beans', 'Ice  cream', 'Eggs']]
```

In [5]:
```
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

In [6]:
```
#4.1) • minsup = 50% , minconf = 75%

frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)
frequent_itemsets

from mlxtend.frequent_patterns import association_rules
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.75)
```

Out[6]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.8	1.00	0.00	1.0
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.0	1.00	0.00	inf
2	(Onion)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf
3	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
4	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
5	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
6	(Kidney Beans, Onion)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf
7	(Eggs, Onion)	(Kidney Beans)	0.6	1.0	0.6	1.0	1.00	0.00	inf
8	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf

In [7]:
```
# 4. 2) • minsup    60% , minconf = 60%

frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets

from mlxtend.frequent_patterns import association_rules
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)
```

Out[7]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.0

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf
2	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
3	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
4	(Kidney Beans)	(Milk)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
5	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
6	(Kidney Beans)	(Onion)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
7	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
8	(Kidney Beans)	(Yogurt)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
9	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
10	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
11	(Kidney Beans, Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
12	(Eggs, Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
13	(Kidney Beans)	(Eggs, Onion)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
14	(Eggs)	(Kidney Beans, Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
15	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf

Q5. Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations: 5.1 a) Training set = 75% Test set = 25% b) Training set = 66.6% (2/3rd of total), Test set = 33.3% 5.2 Training set is chosen by i) hold out method ii) Random subsampling iii) Cross-Validation. Compare the accuracy of the classifiers obtained. 5.3 Data is scaled to standard format.

In [ ]:
```
import numpy as np
import sklearn as skl
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

In [ ]:
```
df=pd.read_csv('wine.csv')
df.dropna(inplace=True)
```

In [ ]:
```
df=pd.read_csv('Iris.csv')
df.drop('Id',axis=1,inplace=True)
df.set_index('Species',inplace=True)
```

In [ ]:
```
print(df)
```

```
                SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
Iris-setosa               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-virginica            6.7           3.0            5.2           2.3
Iris-virginica            6.3           2.5            5.0           1.9
Iris-virginica            6.5           3.0            5.2           2.0
Iris-virginica            6.2           3.4            5.4           2.3
Iris-virginica            5.9           3.0            5.1           1.8

[150 rows x 4 columns]
```

In [ ]:
```
col=df.columns
```

```
In [ ]:  col=list(col)

In [ J:  col

Out[ ]: ['SepallengthCm', 'SepalWidthCm', 'PetallengthCm', 'PetalWidthCm']

In [ ]: df_mean=df[col].mean()

In [ ]:  df mean

Out[ ]: SepallengthCm  5.843333
        SepalWidthCm    3.054000
        PetallengthCm   3.758667
        PetalWidthCm    1.198667
        dtype: float64

In [ ]:  df_std=df[col].std()

In [ ]:  df_std

Out[ J: SepallengthCm    0.828066
        SepalWidthCm     0.433594
        PetallengthCm    1. 764420
        PetalWidthCm     0.763161
        dtype: float64

In [ ]: std_scaler=StandardScaler()
         std scaler

Out[ J: StandardScaler()

In [ ]: df_standarized=pd.DataFrame(std_scaler.fit_transform(df),columns=df.columns)

In [ J:  df_standarized

Out[ ]:
```

	SepallengthCm	SepalWidthCm	PetallengthCm	PetalWidthCm
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977

	SepallengthCm	SepalWidthCm	PetallengthCm	PetalWidthCm
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977
145	1.038005	-0.124958	0.819624	1.447956
146	0.553333	-1.281972	0.705893	0.922064
147	0.795669	-0.124958	0.819624	1.053537
148	0.432165	0.800654	0.933356	1.447956
149	0.068662	-0.124958	0.762759	0.790591

150 rows x 4 columns

```
In [ ]:  df_mean_standarized=df_standarized[col].mean()

In [ ]:  pd.to_numeric(df_mean_standarized,downcast='integer')

Out[ ]:  SepallengthCm   0
         SepalWidthCm    0
         PetallengthCm   0
         PetalWidthCm    0
         dtype: int8

In [ ]:  df_std_standarized=df_standarized[col].std()

In [ ]:  pd.to_numeric(df_std_standarized,downcast='integer')

out[ J:  SepallengthCm   1. 00335
         SepalWidthCm    1.00335
         PetallengthCm   1.00335
         PetalWidthCm    1.00335
         dtype: float64

In [ ]:  ds=pd.read_csv('Iris.csv')
          ds.shape

         (150,  6)
```

```
Out[ ]:

In [ ]:  X=ds.values[:,:-1]
         Y=ds.values[:,-1]
         print(X.shape)
         print(Y.shape)

        (150,  5)
        (150,)

In [ ]:  X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=3)

In [ ]:  DTclassifer=DecisionTreeClassifier()

         DTclassifer.fit(X_train,Y_train)

         predictions=DTclassifer.predict(X_test)
         predictions

out[ J:  array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
                'Iris-setosa', 'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
                'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
                'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
                'Iris-virginica', 'Iris-setosa', 'Iris-versicolor',
                'Iris-virginica', 'Iris-virginica', 'Iris-setosa',
                'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
                'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
                'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
                'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
                'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
                'Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
                'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
                'Iris-virginica'], dtype=object)

In [ ]:  #AccuracyScore

         accuracy_score(Y_test,predictions)

Out[ ]: 1. 0

In [ ]:  #ConfusionMatrix
         confusion_matrix(Y_test,predictions)

Out[ J: array([[17,  0,  0],
       [ 0, 14,  0],
       [ 0,  0, 14]])
```

```
In [ ]: df=pd.read_csv('wine.csv')# reading the dataset
```

```
In [ ]: data=df.values
        x=data[:,0:]#contains aLL info other than class Label
        y=data[:,0]#cLass Label
        print(y)
```

```
[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. 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. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.]
```

```
In [ ]: ###5.la)casel: test is 25%

        Val size=0.25#test size is how 25%
        random_seed = 0#randomLychosing
        X_train,X_test, Y_train,Y_test = train_test_split(x, y, test size= Val_size,random_state = random_seed)
        deciTree = DecisionTreeClassifier()
        deciTree.fit(X_train,Y_train)
        predictions= deciTree.predict(X_test)
        print("Accuracy on the TestData")
        print(accuracy_score(Y_test,predictions))
```

```
Accuracy on the TestData
1.0
```

```
In [ ]: ###5.lbcase2:testsetis  (2/3)rd

        Val_size=0.33#testsizeishowmuchrestistraining
        random_seed=3#randomLychosing
        X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=Val_size,random_state=random_seed)
        deciTree=DecisionTreeClassifier()
        deciTree.fit(X_train,Y_train)
        predictions=deciTree.predict(X_test)
        print("AccuracyontheTestData")
        print(accuracy_score(Y_test,predictions))
```

```
AccuracyontheTestData
```

```
1.0
```

In [ ]:
```
##5.2(a)choosingdatasetusingholdoutmethod

##assumetestdataas10%andrestastrainingdata

Val_size=0.10#testsizeishowmuchrestistraining

X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=Val_size,random_state=0)
deciTree=DecisionTreeClassifier()
deciTree.fit(X_train,Y_train)
predictions=deciTree.predict(X_test)
print("AccuracyontheTestData")
print(accuracy_score(V_test,predictions))
```

```
AccuracyontheTestData
1.0
```

In [ ]:
```
##5.2(b)choosingdatasetusingrandomsubsampling

##assumetestdataas10%andrestastrainingdata

Val_size=0.10          #testsizeishowmuchrestistraining
random_seed = 3
X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=Val_size,random_state = random_seed)
deciTree=DecisionTreeClassifier()
deciTree.fit(X_train,Y_train)
predictions=deciTree.predict(X_test)
print("AccuracyontheTestData")
print(accuracy_score(V_test,predictions))
```

```
AccuracyontheTestData
1.0
```

In [ ]:
```
###5.3scaling of data using minmaxscaler()
scaler=MinMaxScaler()
print(scaler.fit(df))
```

```
MinMaxScaler()
```

In [ ]:
```
df
```

Out[ ]:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
1		13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2		13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3		14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4		13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows x 14 columns

Q6. Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import hierarchical
from sklearn.cluster import DBSCAN
import pandas as pd

iris= pd.read_csv('iris.data')
```

```
C:\Users\chaud\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143: FutureWarning: The sklearn.cluster.hierarchical modul
e is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes/ functions should instead be impo
rted from sklearn.cluster. Anything that cannot be imported from sklearn.cluster is now part of the private API.
  warnings.warn(message, FutureWarning)
```

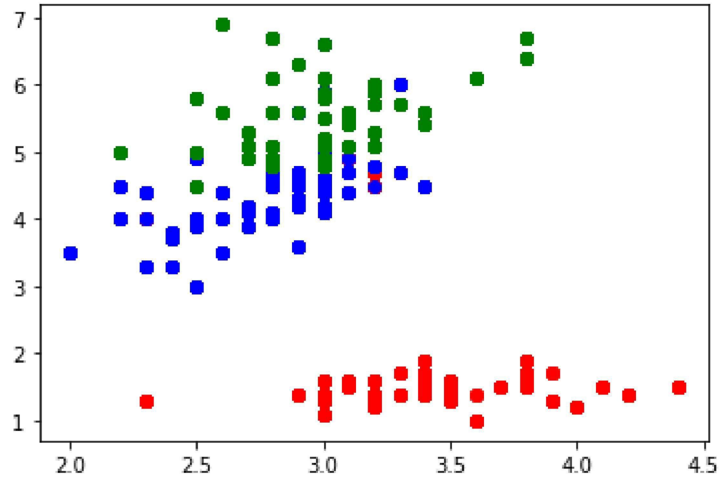
In [2]:
```
iris
```

Out[2]:

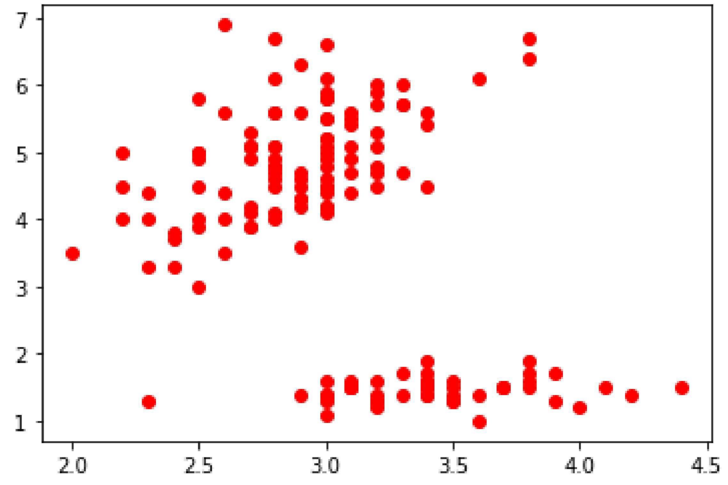
	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa
144	6.7	3.0	5.2	2.3	Iris-virginica
145	6.3	2.5	5.0	1.9	Iris-virginica
146	6.5	3.0	5.2	2.0	Iris-virginica
147	6.2	3.4	5.4	2.3	Iris-virginica
148	5.9	3.0	5.1	1.8	Iris-virginica

149 rows x 5 columns

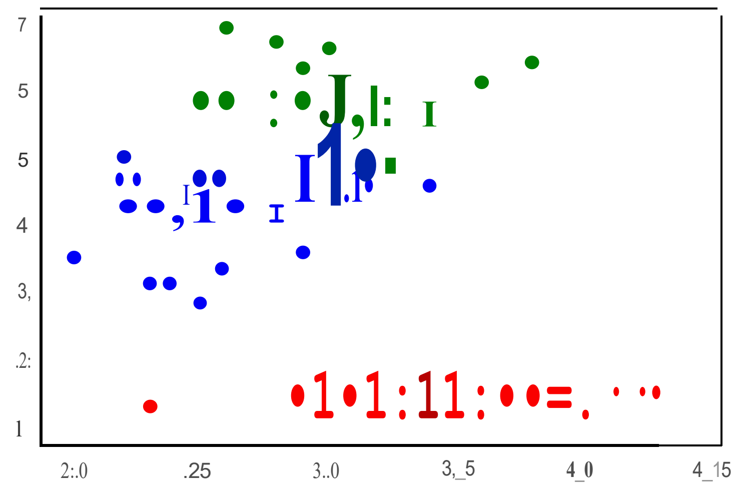
In [3]:
```
sep_length =iris.values[:,0]
pet_length =iris.values[:,1]
# Plotting LabeLized data set

# Taking Sepal Width and Petal Length as our two features for clustering


for i in range(150):
    if i<=49:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'ro')
    if i>49 and i<=99:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'bo')
    if i>99:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'go')
plt.show()
```



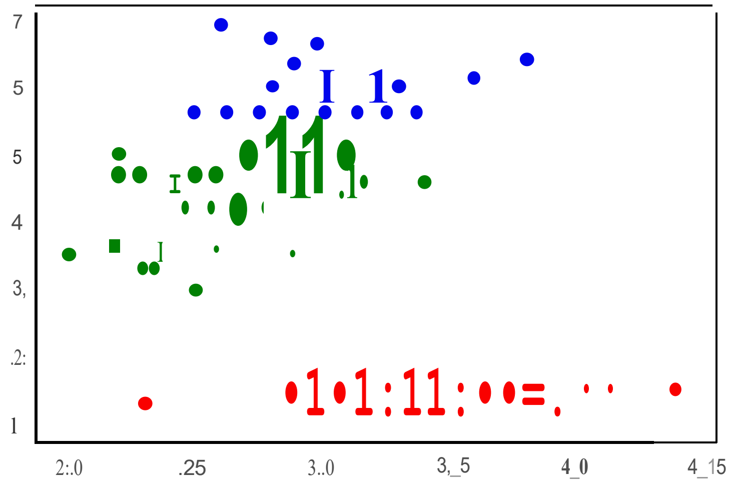
In [5]:
```
# Plotting unLabeLized iris data set

plt.plot(iris.values[:,1],iris.values[:,2],'ro')
plt.show()
```



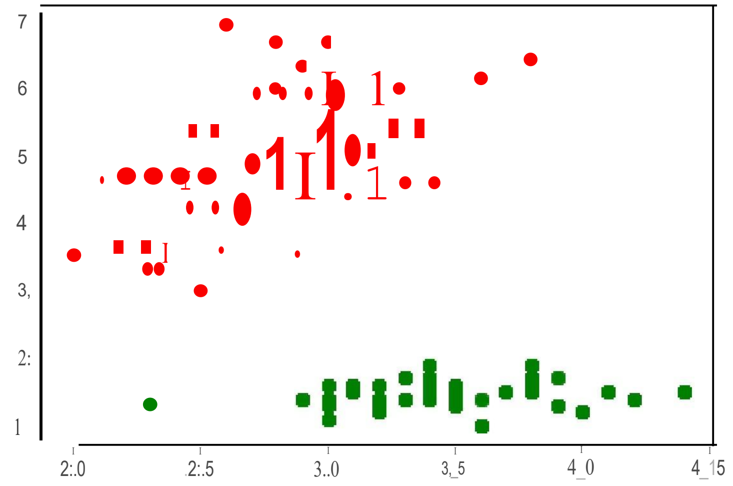
In [6]:
```
##Clustering using KMeans Clustering Algorithm

estimatorl = KMeans(n_clusters=3)

estimatorl.fit(iris.values[:,1:3])

###Plotting Clustered data points using K Means with 3 clusters

for i in range(149):
    if estimatorl.labels_[i]==0:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'go')
            plt.plot(estimatorl.cluster_centers_[:,0],estimatorl.cluster_centers_[:,1],'o',c='black')
    elif estimatorl.labels_[i]==1:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'ro')
            plt.plot(estimatorl.cluster_centers_[:,0],estimatorl.cluster_centers_[:,1],'o',c='black')
    elif estimatorl.labels_[i]==2:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'bo')
            plt.plot(estimatorl.cluster_centers_[:,0],estimatorl.cluster_centers_[:,1],'o',c='black')
plt.show()
```



In [7]:
```
#Black points are centroids

##Clustering using Hierarchical Clustering Algorithm

estimator2 = hierarchical.AgglomerativeClustering(n_clusters=3)

estimator2.fit(iris.values[:,1:3])

for i in range(149):
    if estimator2.labels_[i]==0:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'go')
    elif estimator2.labels_[i]==1:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'ro')
    elif estimator2.labels_[i]==2:
            plt.plot(iris.values[i:,1],iris.values[i:,2],'bo')
pH.show()
```



```
In [14]:  ##Clustering using DBSCAN Clustering Algorithm

          estimator3 = DBSCAN()

          estimator3.fit(iris.values[:,1:3])

          for i in range(149):
              if        estimator3.labels_[i]==0:
                      plt.plot(iris.values[i:,1],iris.values[i:,2],'go')
              elif estimator3.labels_[i]==1:
                      plt.plot(iris.values[i:,1],iris.values[i:,2],'ro')
              elif estimator3.labels_[i]==2:
                      plt.plot(iris.values[i:,1],iris.values[i:,2],'bo')
          plt.show()
```



```
In [ ]:
```