Practical Assessment (Assignment) Even Semester
Course: B.Sc. (H) Computer Science Semester VI (Section B)

Paper Name: Data Mining

Date: 29th April 2023

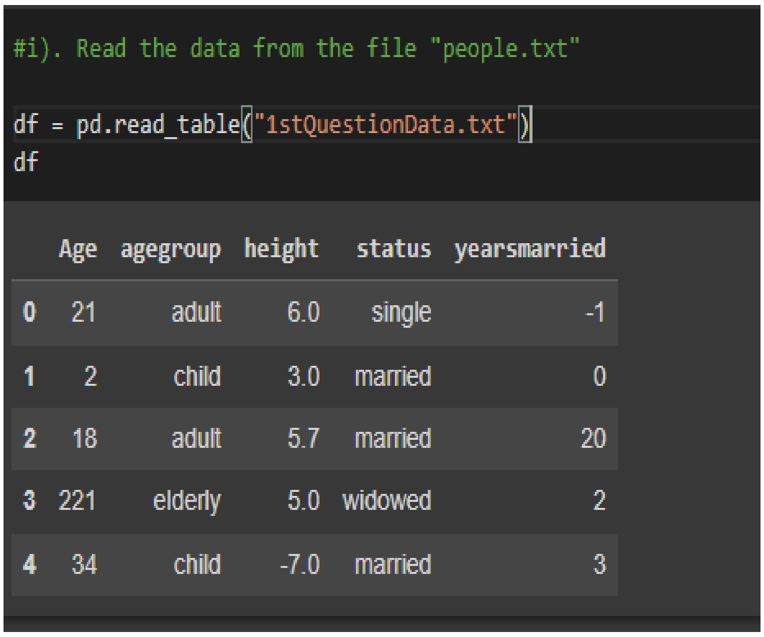
Name of the Candidate: Vishal Maurya
Roll No.: 5750
Examination Roll No.: 20035570117
Mobile Number: 9354643132
e-Mail Id: vishal205750@keshav.du.ac.in

Submitted To: Nidhi Ma'am

Q1. Create a file “people.txt” with the following data:

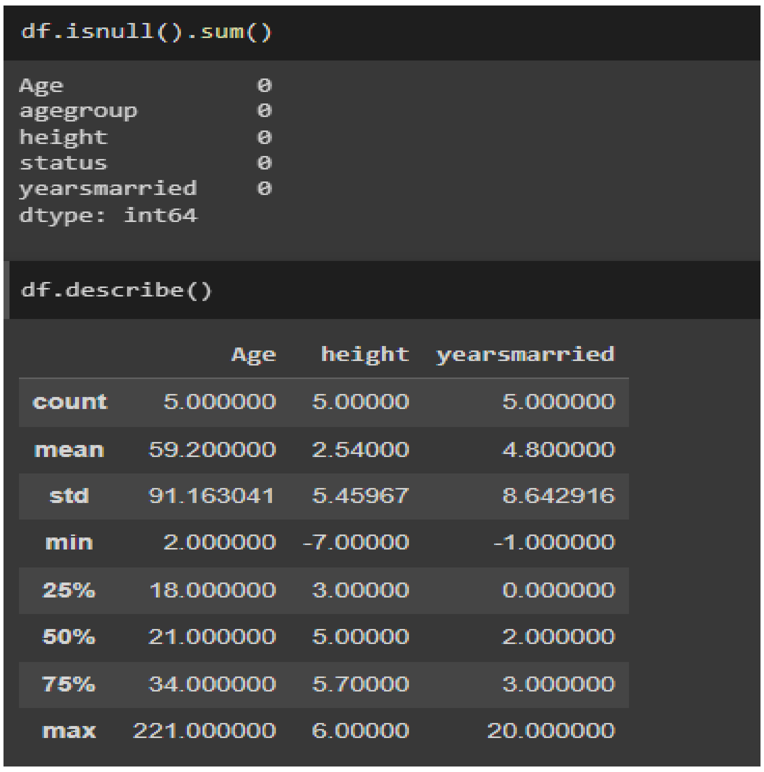
Age	agegroup	height	status	yearsmarried
21	adult	6.0	single	-1
2	child	3	married	0
18	adult	5.7	married	20
221	elderly	5	widowed	2
34	child	-7	married	3

i) Read the data from the file “people.txt”.



```
#i). Read the data from the file "people.txt"

df = pd.read_table("1stQuestionData.txt")
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
```



```
df.isnull().sum()
```

```
Age             0
agegroup        0
height          0
status          0
yearsmarried    0
dtype: int64
```

```
df.describe()
```

```
              Age    height  yearsmarried
count    5.000000   5.00000      5.000000
mean    59.200000   2.54000      4.800000
std     91.163041   5.45967      8.642916
min      2.000000  -7.00000     -1.000000
25%     18.000000   3.00000      0.000000
50%     21.000000   5.00000      2.000000
75%     34.000000   5.70000      3.000000
max    221.000000   6.00000     20.000000
```

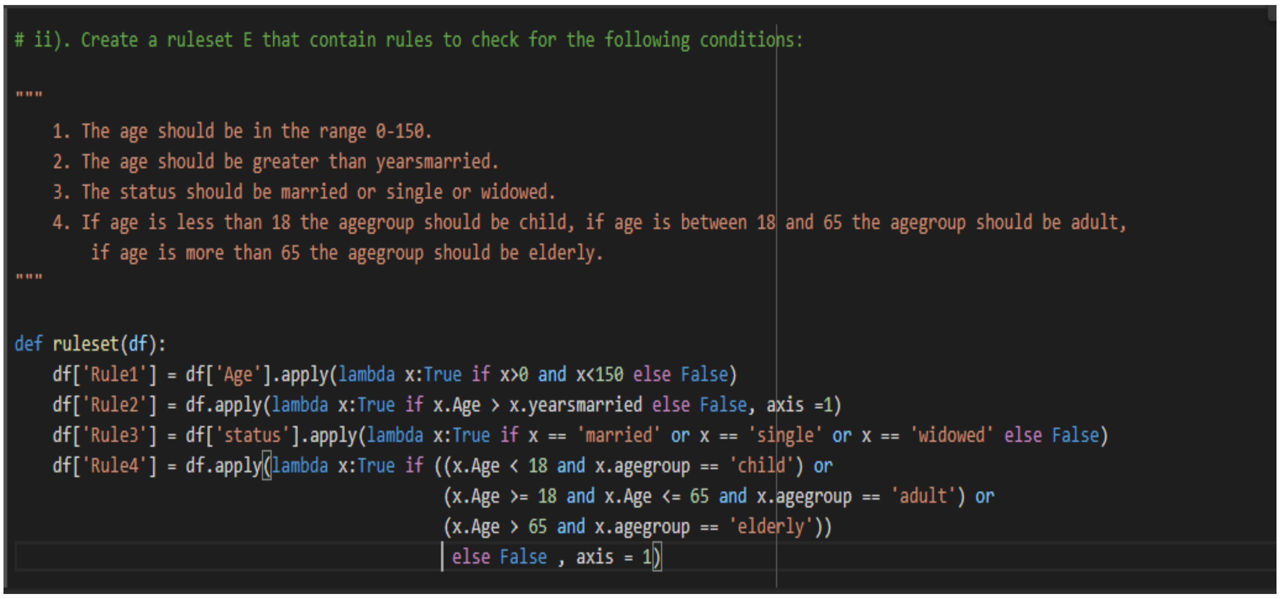
ii) Create a ruleset E that contain rules to check for the following conditions:

1. The age should be in the range 0-150.

2. The age should be greater than years married.

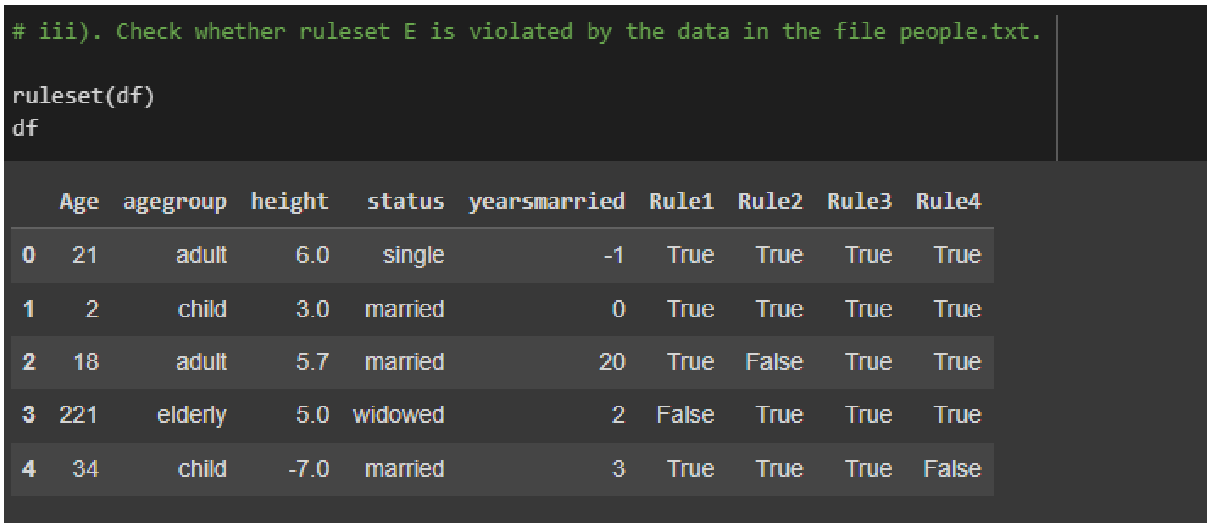
3. The status should be married or single or widowed.

4. If age is less than 18 the age group should be child, if age is between 18 and 65 the age group should be adult, if age is more than 65 the age group should be elderly.



```
# 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.
"""

def ruleset(df):
    df['Rule1'] = df['Age'].apply(lambda x:True if x>0 and x<150 else False)
    df['Rule2'] = df.apply(lambda x:True if x.Age > x.yearsmarried else False, axis =1)
    df['Rule3'] = df['status'].apply(lambda x:True if x == 'married' or x == 'single' or x == 'widowed' else False)
    df['Rule4'] = df.apply(lambda x:True if ((x.Age < 18 and x.agegroup == 'child') or
                                              (x.Age >= 18 and x.Age <= 65 and x.agegroup == 'adult') or
                                              (x.Age > 65 and x.agegroup == 'elderly'))
                                              else False , axis = 1)
```

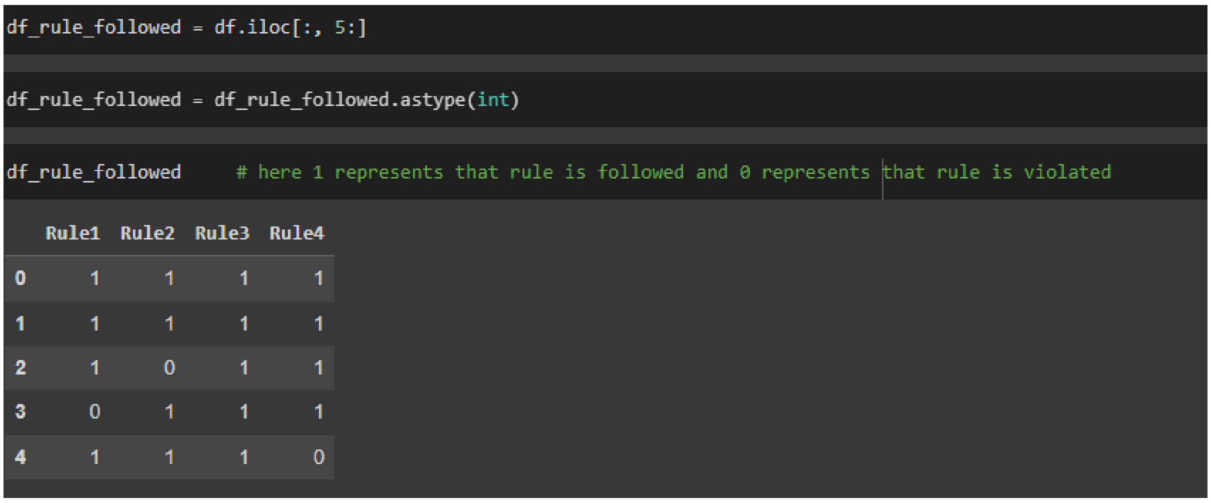
iii) Check whether ruleset E is violated by the data in the file people.txt.



```
# iii). Check whether ruleset E is violated by the data in the file people.txt.

ruleset(df)
df
```

	Age	agegroup	height	status	yearsmarried	Rule1	Rule2	Rule3	Rule4
0	21	adult	6.0	single	-1	True	True	True	True
1	2	child	3.0	married	0	True	True	True	True
2	18	adult	5.7	married	20	True	False	True	True
3	221	elderly	5.0	widowed	2	False	True	True	True
4	34	child	-7.0	married	3	True	True	True	False

iv) Summarize the results obtained in part (iii).

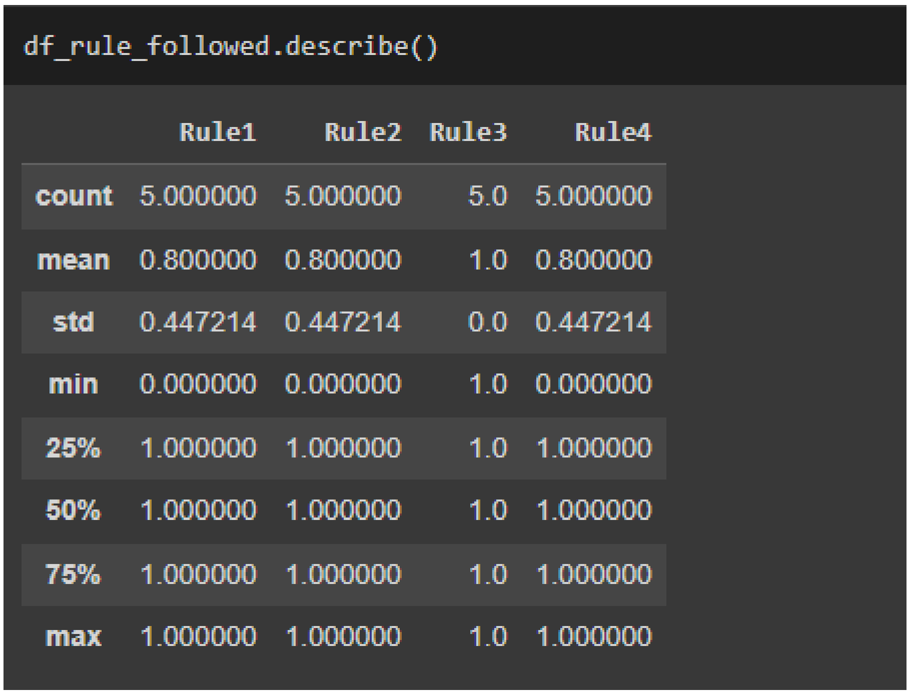


```python
df_rule_followed = df.iloc[:, 5:]
```

```python
df_rule_followed = df_rule_followed.astype(int)
```

```python
df_rule_followed     # here 1 represents that rule is followed and 0 represents that rule is violated
```

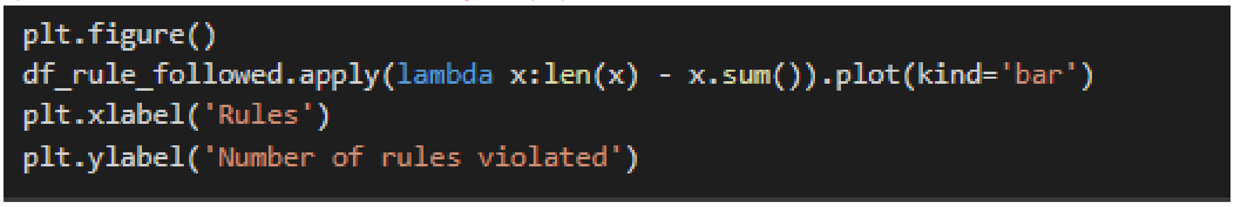
```
   Rule1  Rule2  Rule3  Rule4
0      1      1      1      1
1      1      1      1      1
2      1      0      1      1
3      0      1      1      1
4      1      1      1      0
```



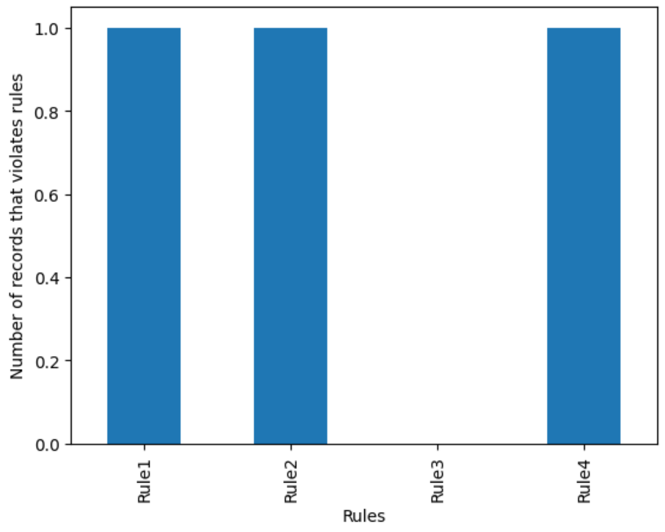
```python
df_rule_followed.describe()
```

```
          Rule1     Rule2  Rule3     Rule4
count  5.000000  5.000000    5.0  5.000000
mean   0.800000  0.800000    1.0  0.800000
std    0.447214  0.447214    0.0  0.447214
min    0.000000  0.000000    1.0  0.000000
25%    1.000000  1.000000    1.0  1.000000
50%    1.000000  1.000000    1.0  1.000000
75%    1.000000  1.000000    1.0  1.000000
max    1.000000  1.000000    1.0  1.000000
```

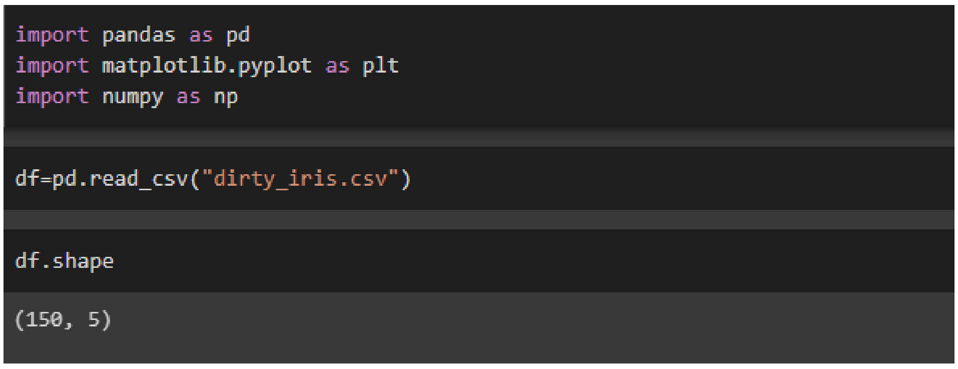
v) Visualize the results obtained in part (iii)



```python
plt.figure()
df_rule_followed.apply(lambda x:len(x) - x.sum()).plot(kind='bar')
plt.xlabel('Rules')
plt.ylabel('Number of rules violated')
```



Q2. Perform the following preprocessing tasks on the dirty_iris dataset.

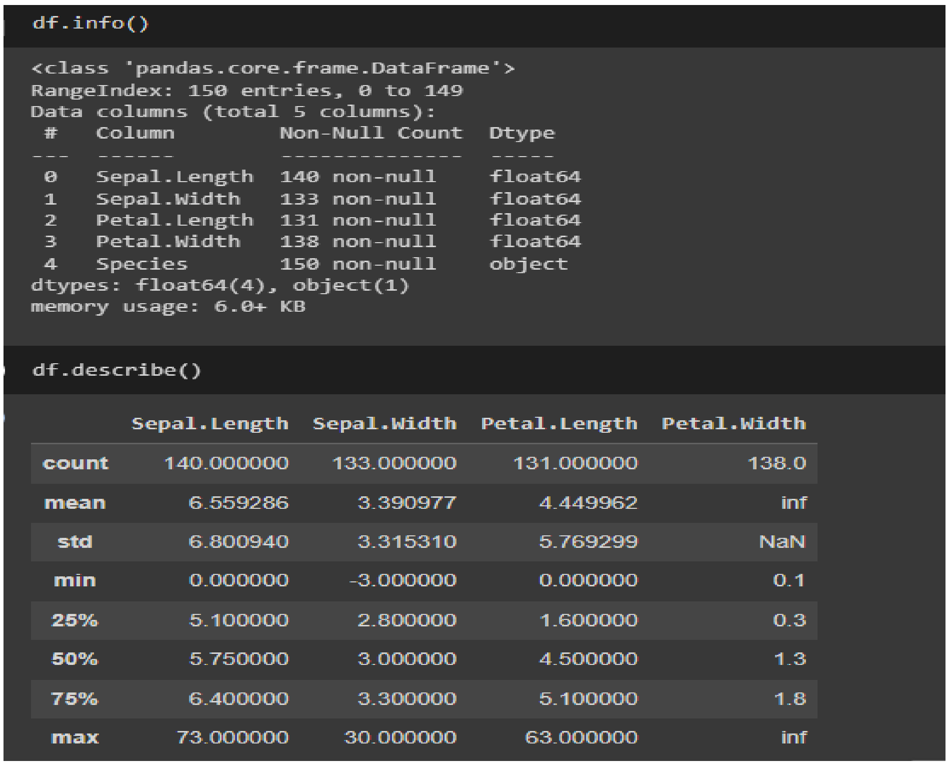


```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
df=pd.read_csv("dirty_iris.csv")
```

```
df.shape
```

```
(150, 5)
```

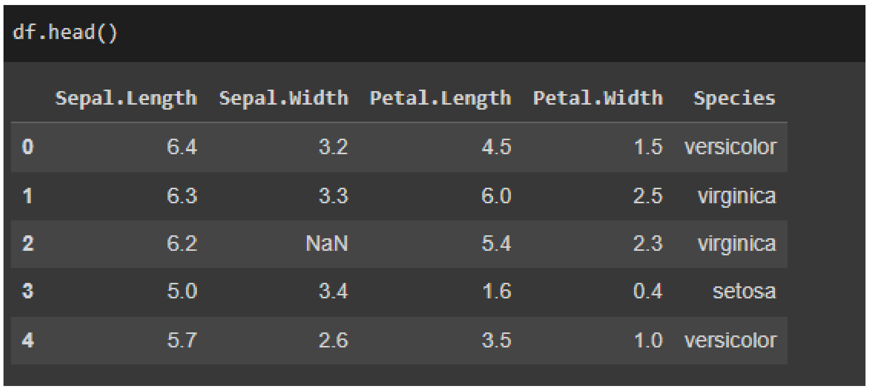


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Sepal.Length  140 non-null    float64
 1   Sepal.Width   133 non-null    float64
 2   Petal.Length  131 non-null    float64
 3   Petal.Width   138 non-null    float64
 4   Species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
df.describe()
```

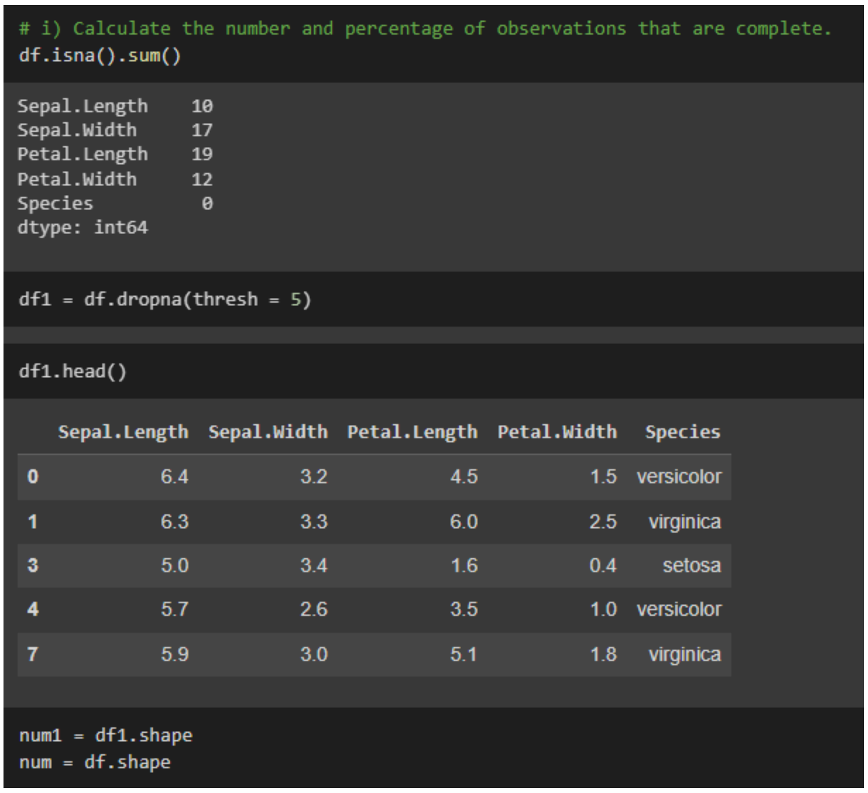
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	140.000000	133.000000	131.000000	138.0
mean	6.559286	3.390977	4.449962	inf
std	6.800940	3.315310	5.769299	NaN
min	0.000000	-3.000000	0.000000	0.1
25%	5.100000	2.800000	1.600000	0.3
50%	5.750000	3.000000	4.500000	1.3
75%	6.400000	3.300000	5.100000	1.8
max	73.000000	30.000000	63.000000	inf



```
df.head()
```

	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.



```
# i) Calculate the number and percentage of observations that are complete.
df.isna().sum()
```

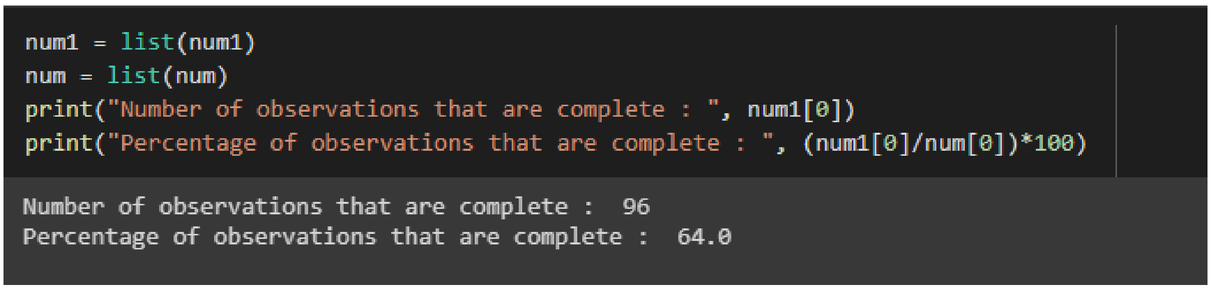
```
Sepal.Length    10
Sepal.Width     17
Petal.Length    19
Petal.Width     12
Species          0
dtype: int64
```

```
df1 = df.dropna(thresh = 5)
```

```
df1.head()
```

	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
3	5.0	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1.0	versicolor
7	5.9	3.0	5.1	1.8	virginica

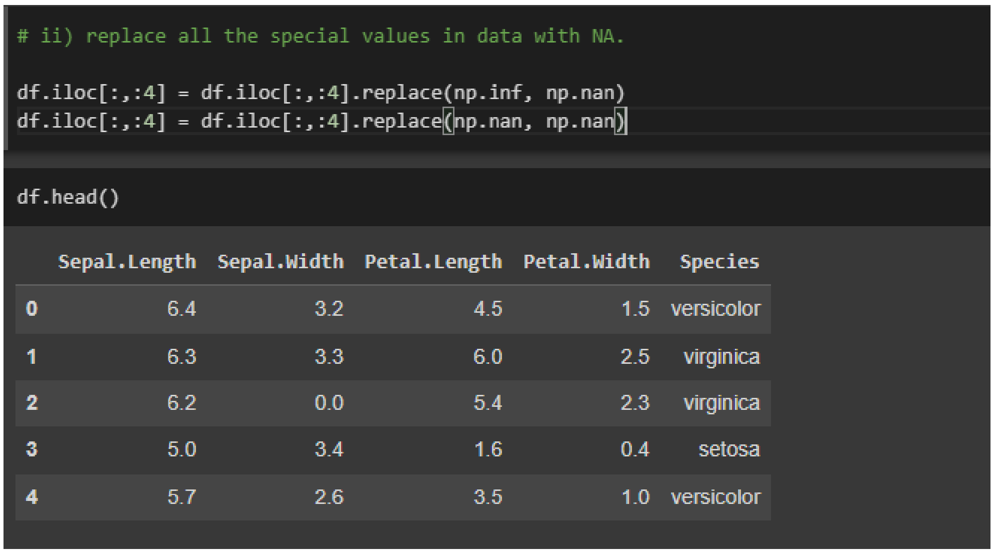
```
num1 = df1.shape
num = df.shape
```



```
num1 = list(num1)
num = list(num)
print("Number of observations that are complete : ", num1[0])
print("Percentage of observations that are complete : ", (num1[0]/num[0])*100)
```

```
Number of observations that are complete :  96
Percentage of observations that are complete :  64.0
```

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



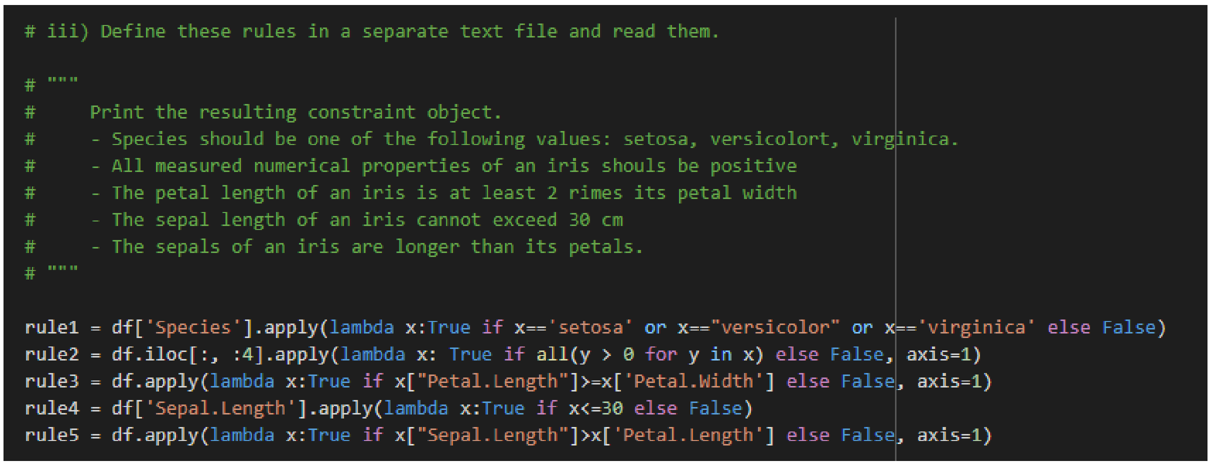
```
# ii) replace all the special values in data with NA.

df.iloc[:,:4] = df.iloc[:,:4].replace(np.inf, np.nan)
df.iloc[:,:4] = df.iloc[:,:4].replace(np.nan, np.nan)
```

```
df.head()
```

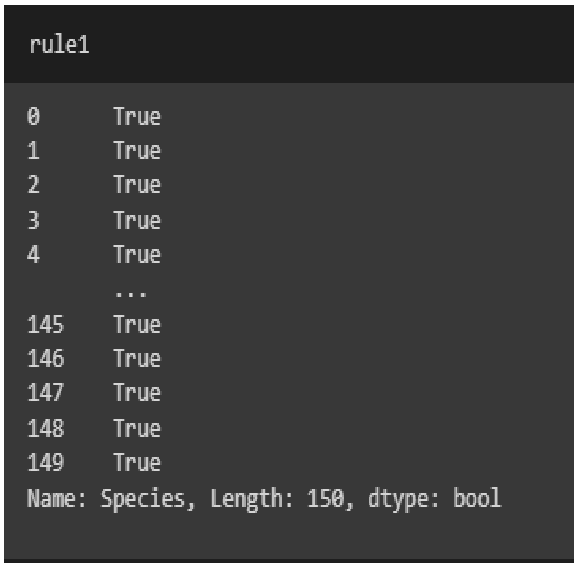
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	6.4	3.2	4.5	1.5	versicolor
1	6.3	3.3	6.0	2.5	virginica
2	6.2	0.0	5.4	2.3	virginica
3	5.0	3.4	1.6	0.4	setosa
4	5.7	2.6	3.5	1.0	versicolor

iii) Define these rules in a separate text file and read them. (Use editfile function in R (package editrules). Use similar function in Python).

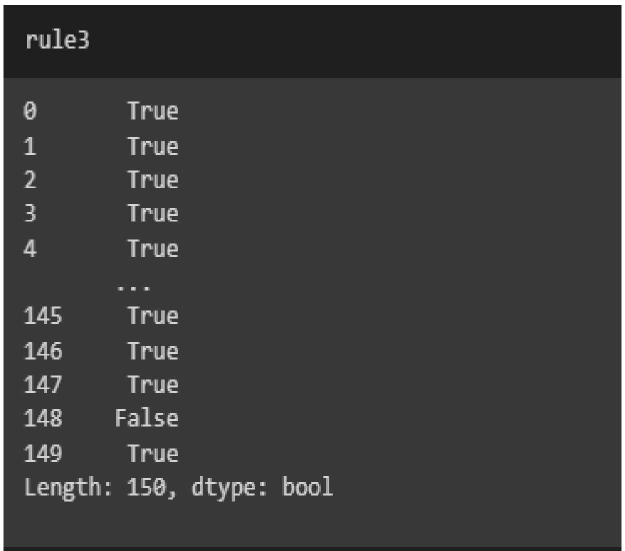
Print the resulting constraint object.
– Species should be one of the following values: setosa, versicolor or virginica.
– All measured numerical properties of an iris should be positive.
– The petal length of an iris is at least 2 times its petal width.
– The sepal length of an iris cannot exceed 30 cm.
– The sepals of an iris are longer than its petals.



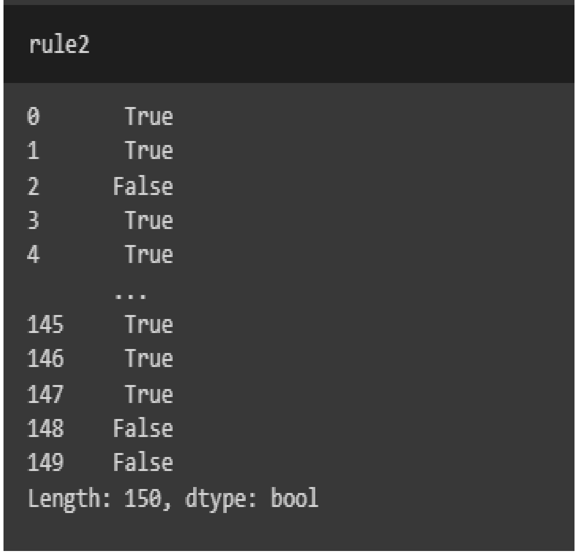
```
# iii) Define these rules in a separate text file and read them.

# """
#     Print the resulting constraint object.
#     - Species should be one of the following values: setosa, versicolort, virginica.
#     - All measured numerical properties of an iris shouls be positive
#     - The petal length of an iris is at least 2 rimes its petal width
#     - The sepal length of an iris cannot exceed 30 cm
#     - The sepals of an iris are longer than its petals.
# """

rule1 = df['Species'].apply(lambda x:True if x=='setosa' or x=="versicolor" or x=='virginica' else False)
rule2 = df.iloc[:, :4].apply(lambda x: True if all(y > 0 for y in x) else False, axis=1)
rule3 = df.apply(lambda x:True if x["Petal.Length"]>=x['Petal.Width'] else False, axis=1)
rule4 = df['Sepal.Length'].apply(lambda x:True if x<=30 else False)
rule5 = df.apply(lambda x:True if x["Sepal.Length"]>x['Petal.Length'] else False, axis=1)
```



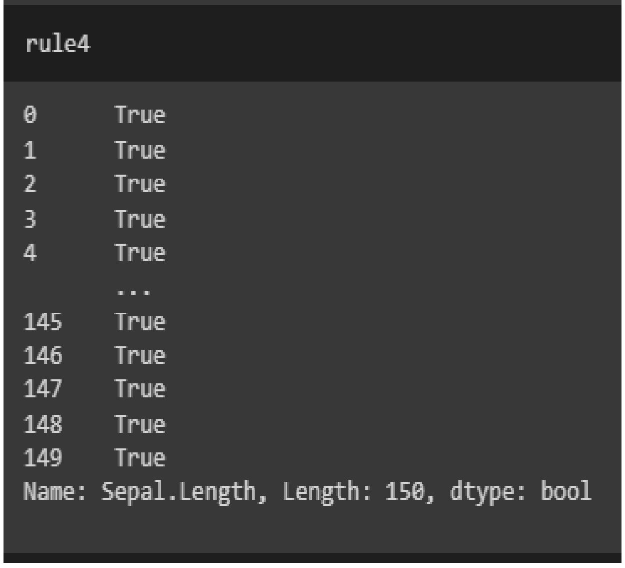
```
rule1

0      True
1      True
2      True
3      True
4      True
       ...
145    True
146    True
147    True
148    True
149    True
Name: Species, Length: 150, dtype: bool
```



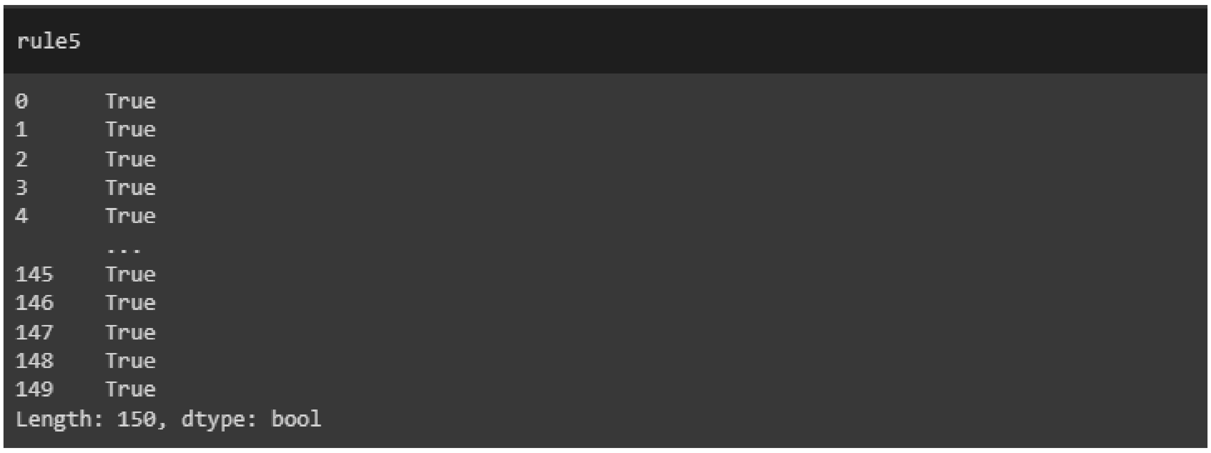
```
rule3

0       True
1       True
2       True
3       True
4       True
       ...
145     True
146     True
147     True
148    False
149     True
Length: 150, dtype: bool
```



```
rule2

0       True
1       True
2      False
3       True
4       True
       ...
145     True
146     True
147     True
148    False
149    False
Length: 150, dtype: bool
```

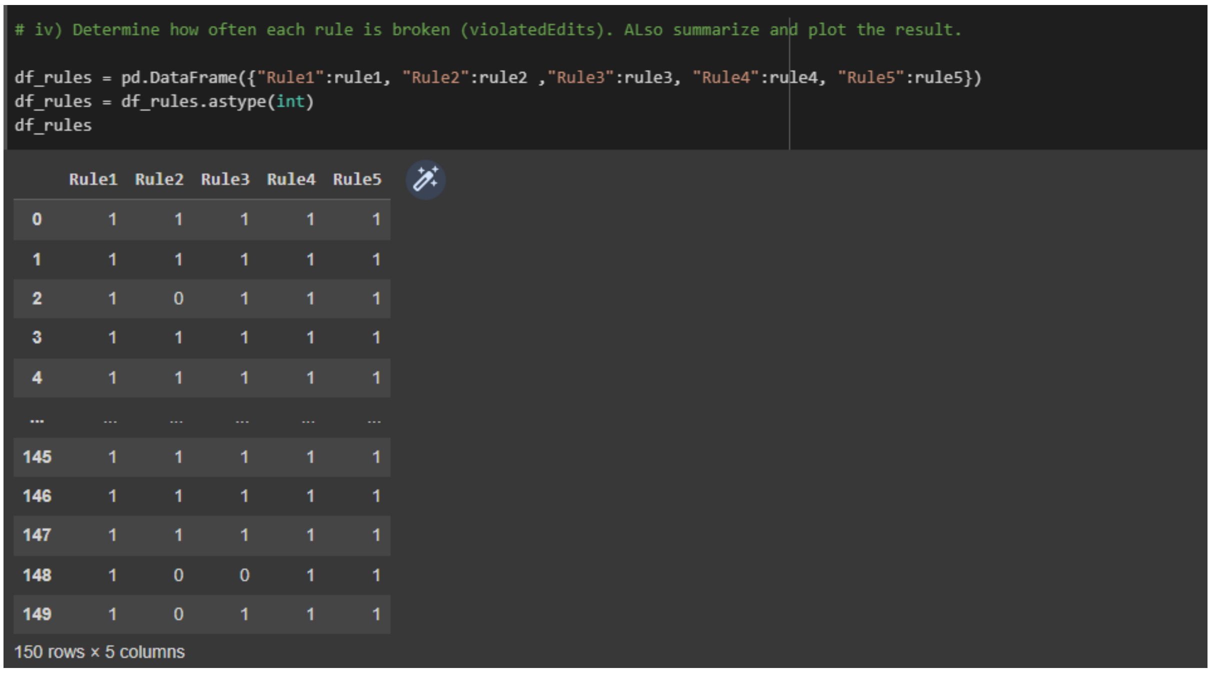


```
rule4

0      True
1      True
2      True
3      True
4      True
       ...
145    True
146    True
147    True
148    True
149    True
Name: Sepal.Length, Length: 150, dtype: bool
```



```
rule5

0      True
1      True
2      True
3      True
4      True
       ...
145    True
146    True
147    True
148    True
149    True
Length: 150, dtype: bool
```

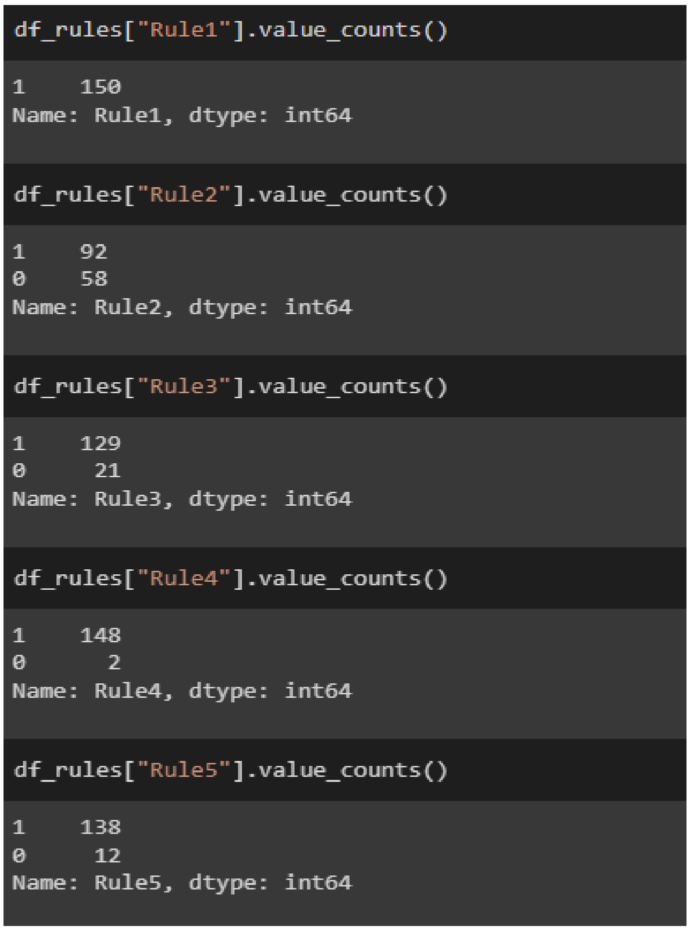
iv) Determine how often each rule is broken (violatedEdits). Also summarize and plot the result.



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

df_rules = pd.DataFrame({"Rule1":rule1, "Rule2":rule2 ,"Rule3":rule3, "Rule4":rule4, "Rule5":rule5})
df_rules = df_rules.astype(int)
df_rules
```

	Rule1	Rule2	Rule3	Rule4	Rule5
0	1	1	1	1	1
1	1	1	1	1	1
2	1	0	1	1	1
3	1	1	1	1	1
4	1	1	1	1	1
...	...	...	...	...	...
145	1	1	1	1	1
146	1	1	1	1	1
147	1	1	1	1	1
148	1	0	0	1	1
149	1	0	1	1	1

150 rows × 5 columns



```
df_rules["Rule1"].value_counts()
```

```
1    150
Name: Rule1, dtype: int64
```

```
df_rules["Rule2"].value_counts()
```

```
1    92
0    58
Name: Rule2, dtype: int64
```

```
df_rules["Rule3"].value_counts()
```

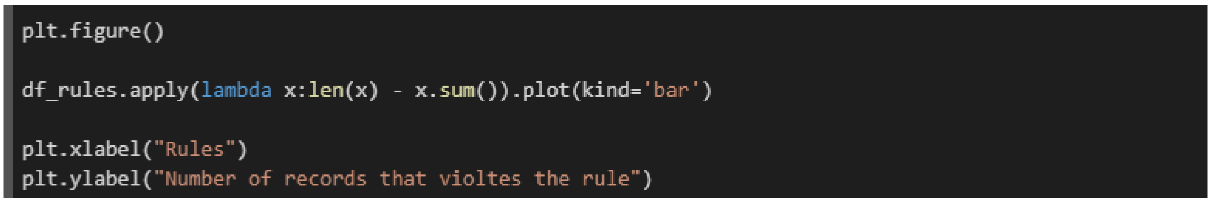
```
1    129
0     21
Name: Rule3, dtype: int64
```

```
df_rules["Rule4"].value_counts()
```

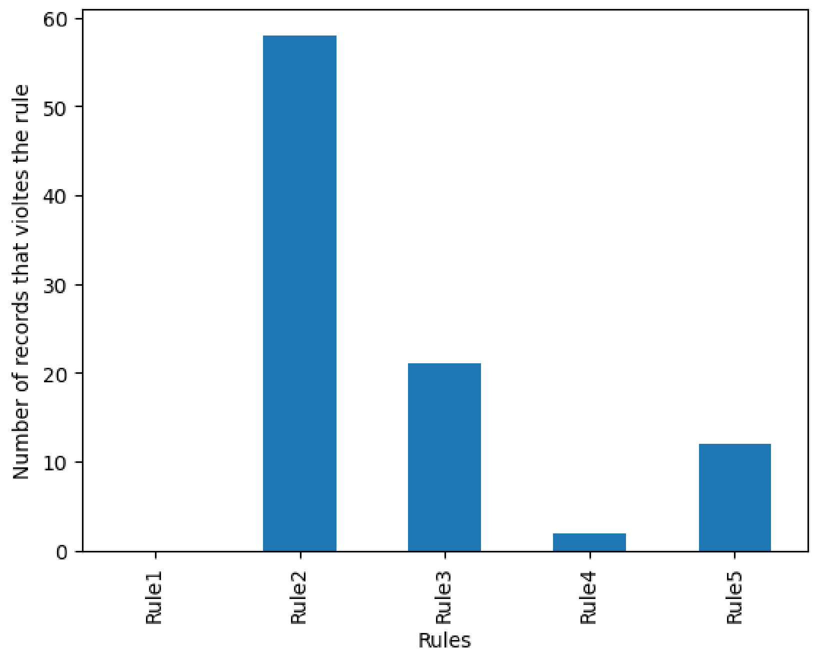
```
1    148
0      2
Name: Rule4, dtype: int64
```

```
df_rules["Rule5"].value_counts()
```

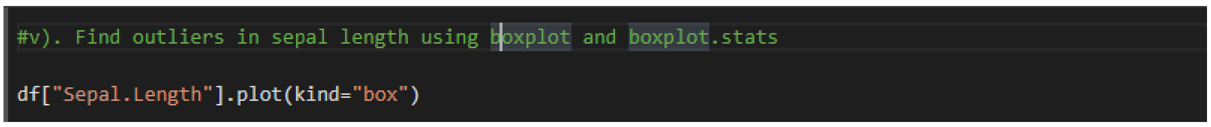
```
1    138
0     12
Name: Rule5, dtype: int64
```



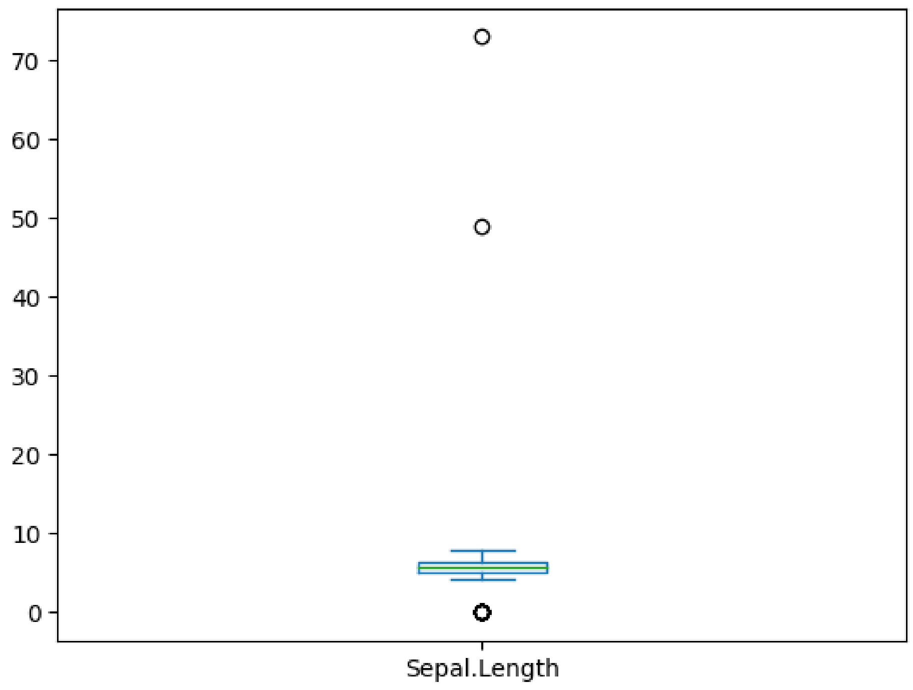
```
plt.figure()

df_rules.apply(lambda x:len(x) - x.sum()).plot(kind='bar')

plt.xlabel("Rules")
plt.ylabel("Number of records that violtes the rule")
```

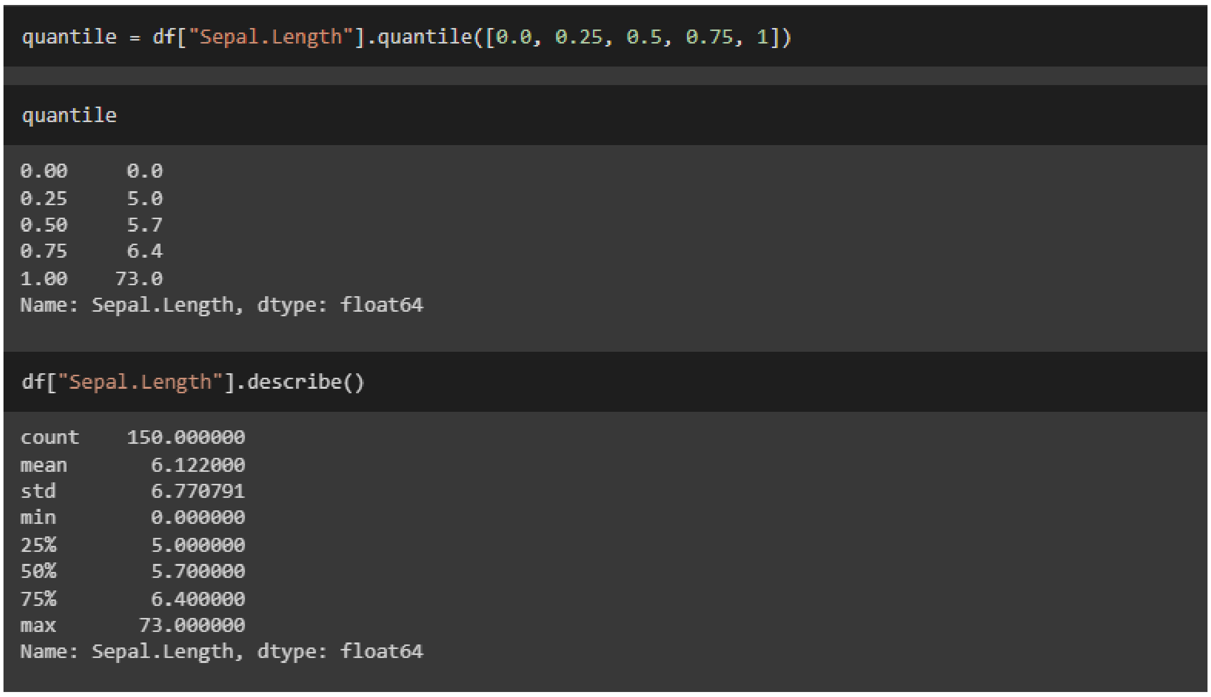


v) Find outliers in sepal length using boxplot and boxplot.stats



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

df["Sepal.Length"].plot(kind="box")
```





```
quantile = df["Sepal.Length"].quantile([0.0, 0.25, 0.5, 0.75, 1])
```

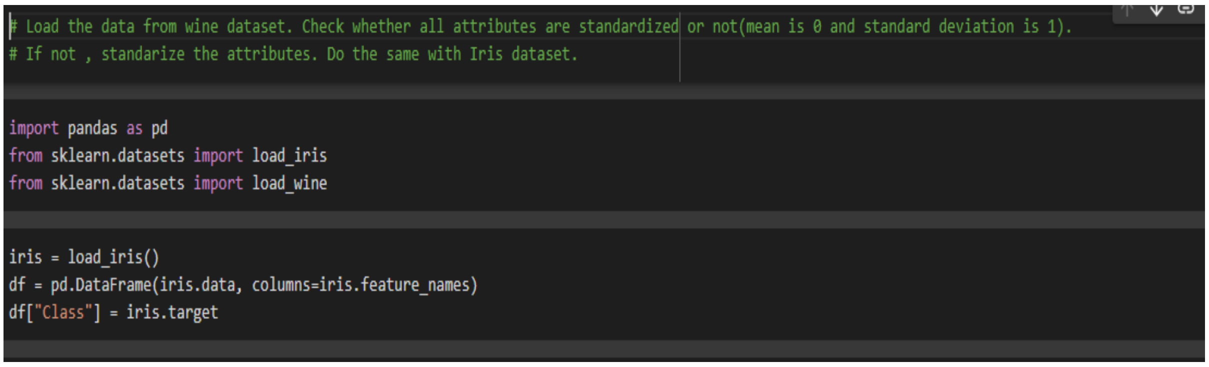
```
quantile
```

```
0.00     0.0
0.25     5.0
0.50     5.7
0.75     6.4
1.00    73.0
Name: Sepal.Length, dtype: float64
```

```
df["Sepal.Length"].describe()
```

```
count    150.000000
mean       6.122000
std        6.770791
min        0.000000
25%        5.000000
50%        5.700000
75%        6.400000
max       73.000000
Name: Sepal.Length, dtype: float64
```

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.

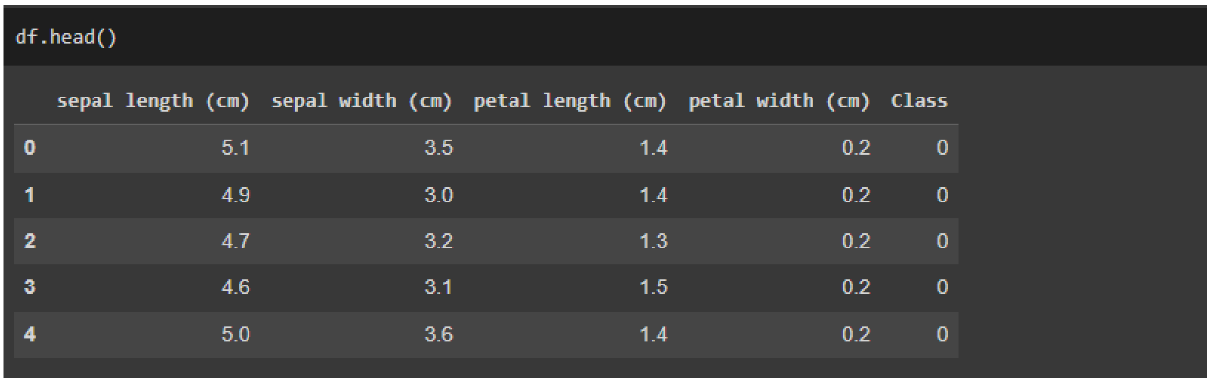


```
# Load the data from wine dataset. Check whether all attributes are standardized or not(mean is 0 and standard deviation is 1).
# If not , standarize the attributes. Do the same with Iris dataset.
```

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.datasets import load_wine
```

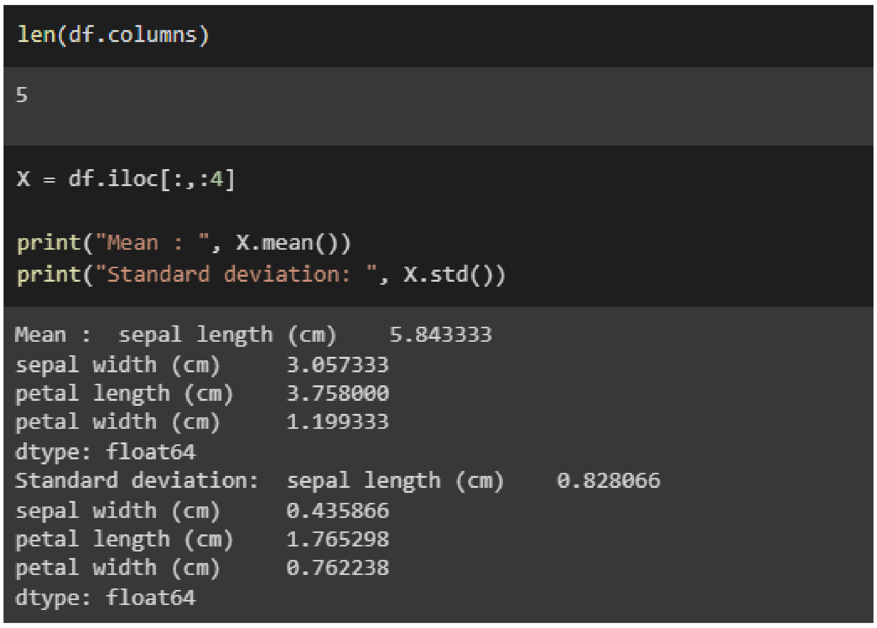
```
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df["Class"] = iris.target
```

For Iris Dataset:



```
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

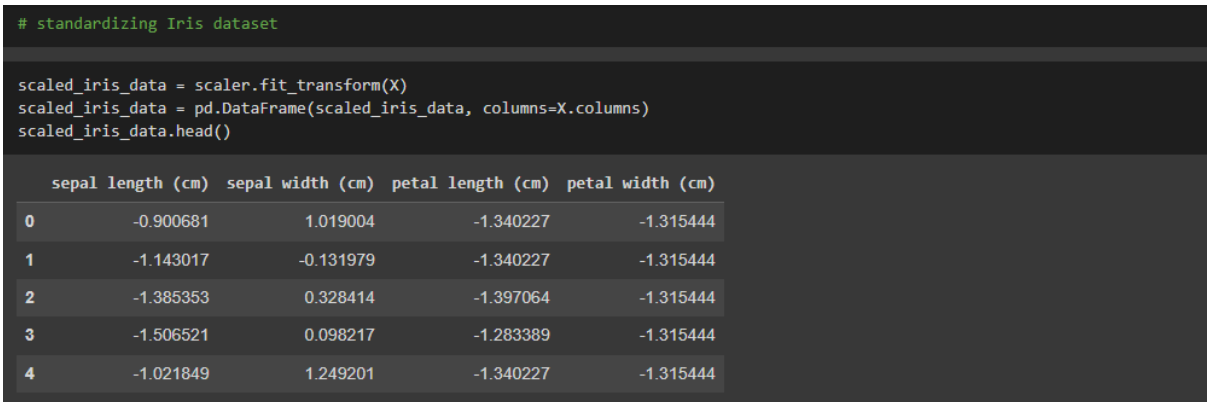


```
len(df.columns)
```

```
5
```

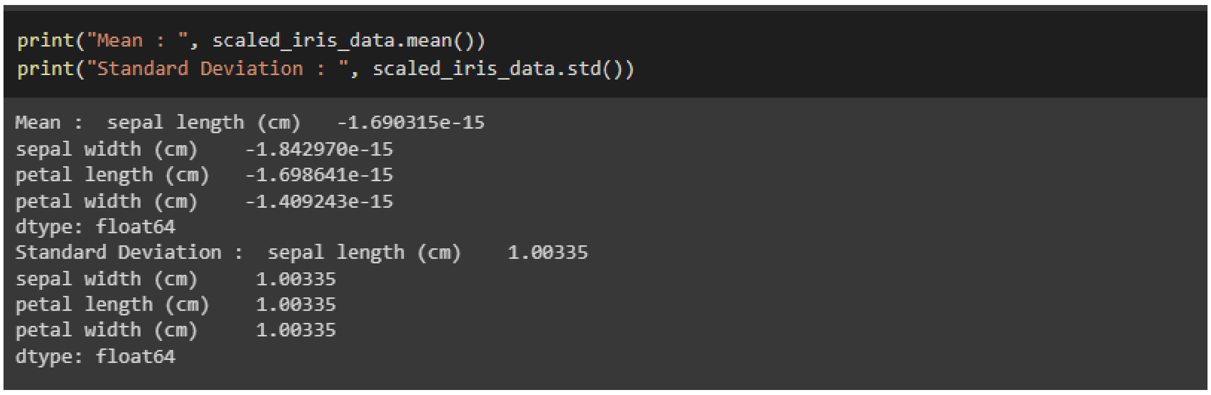
```
X = df.iloc[:,:4]

print("Mean : ", X.mean())
print("Standard deviation: ", X.std())
```

```
Mean :  sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
dtype: float64
Standard deviation:  sepal length (cm)    0.828066
sepal width (cm)     0.435866
petal length (cm)    1.765298
petal width (cm)     0.762238
dtype: float64
```



```
# standardizing Iris dataset

scaled_iris_data = scaler.fit_transform(X)
scaled_iris_data = pd.DataFrame(scaled_iris_data, columns=X.columns)
scaled_iris_data.head()
```

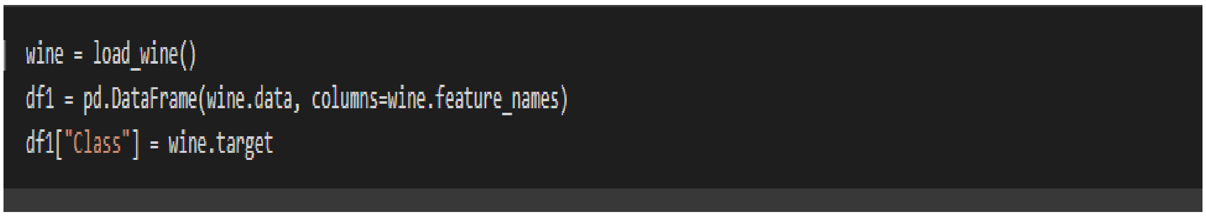
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444



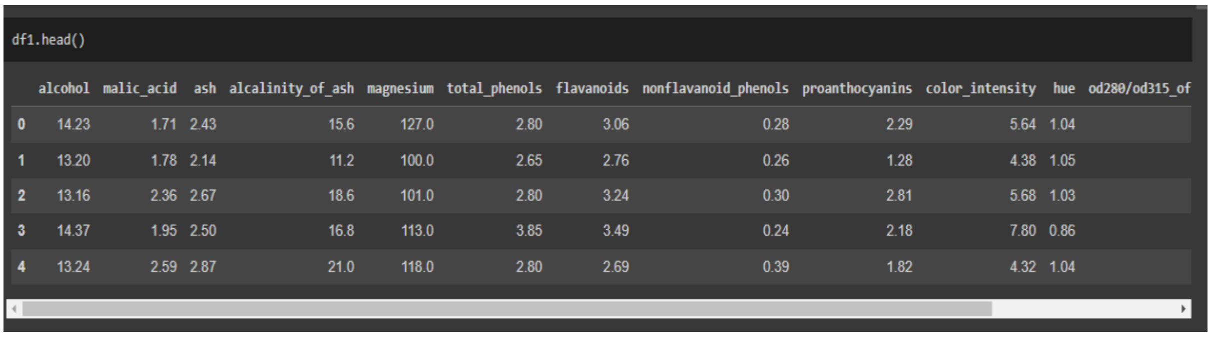
```
print("Mean : ", scaled_iris_data.mean())
print("Standard Deviation : ", scaled_iris_data.std())
```

```
Mean :  sepal length (cm)   -1.690315e-15
sepal width (cm)    -1.842970e-15
petal length (cm)   -1.698641e-15
petal width (cm)    -1.409243e-15
dtype: float64
Standard Deviation :  sepal length (cm)    1.00335
sepal width (cm)     1.00335
petal length (cm)    1.00335
petal width (cm)     1.00335
dtype: float64
```

For Wine Dataset

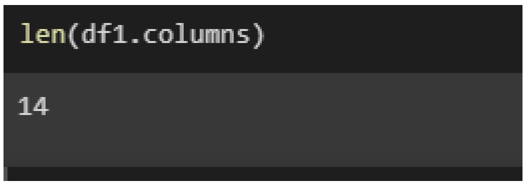


```
wine = load_wine()
df1 = pd.DataFrame(wine.data, columns=wine.feature_names)
df1["Class"] = wine.target
```



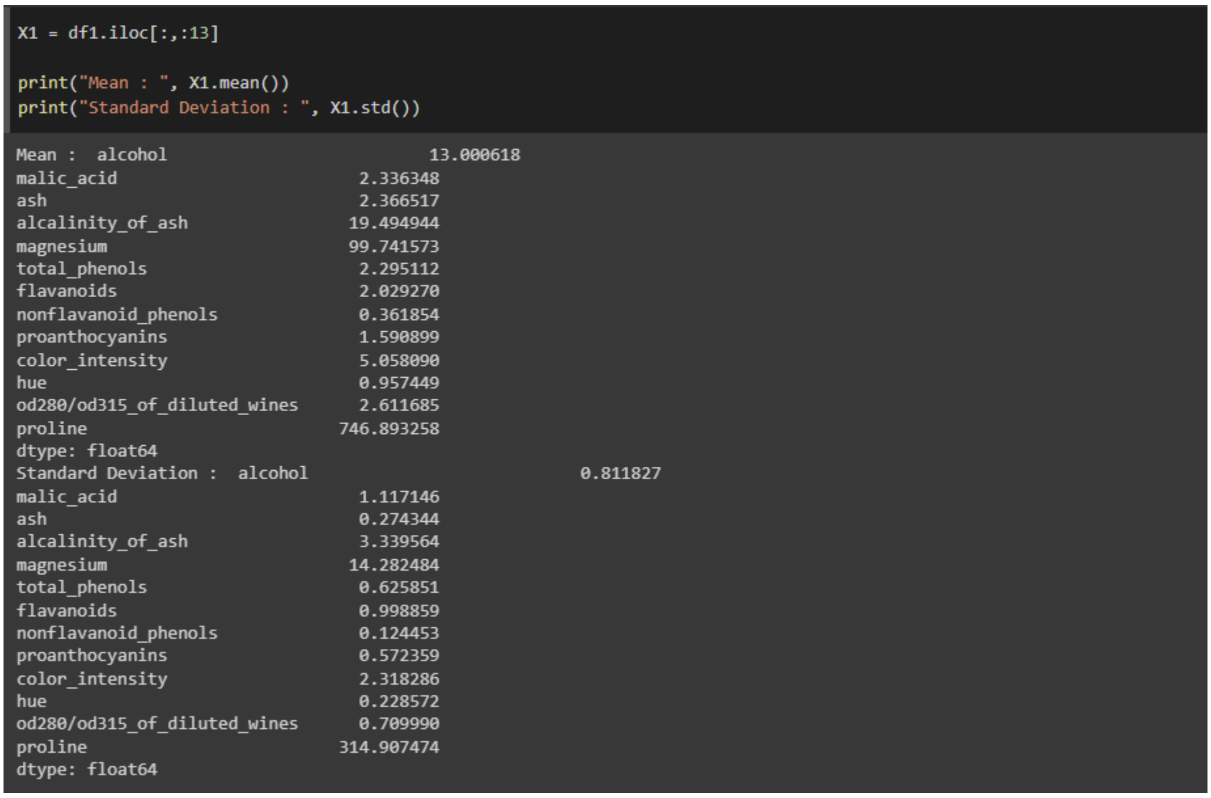
```
df1.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	



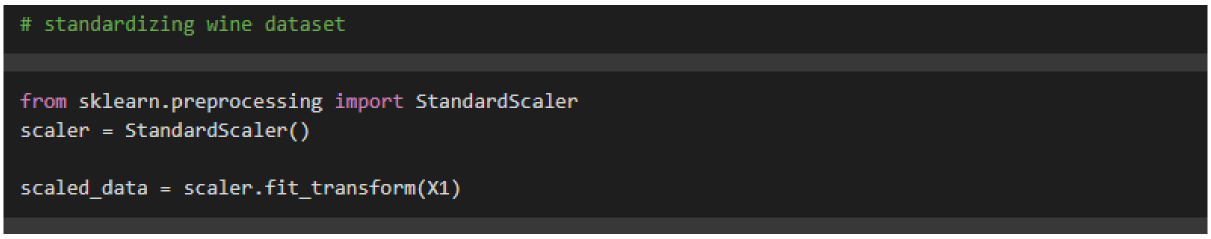
```
len(df1.columns)
```

```
14
```

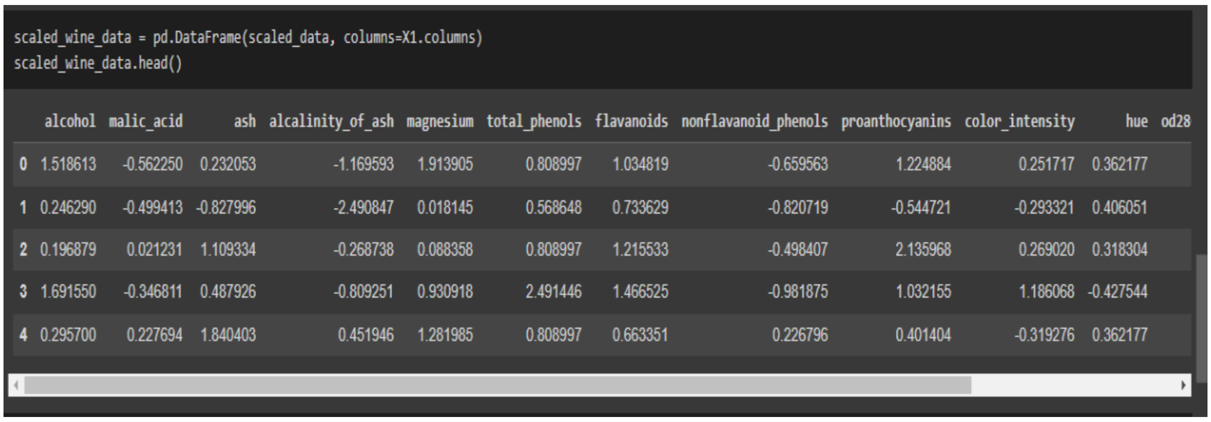


```
X1 = df1.iloc[:,:13]

print("Mean : ", X1.mean())
print("Standard Deviation : ", X1.std())
```

```
Mean :  alcohol                          13.000618
malic_acid                       2.336348
ash                              2.366517
alcalinity_of_ash               19.494944
magnesium                       99.741573
total_phenols                    2.295112
flavanoids                       2.029270
nonflavanoid_phenols             0.361854
proanthocyanins                  1.590899
color_intensity                  5.058090
hue                              0.957449
od280/od315_of_diluted_wines     2.611685
proline                        746.893258
dtype: float64
Standard Deviation :  alcohol                          0.811827
malic_acid                       1.117146
ash                              0.274344
alcalinity_of_ash                3.339564
magnesium                       14.282484
total_phenols                    0.625851
flavanoids                       0.998859
nonflavanoid_phenols             0.124453
proanthocyanins                  0.572359
color_intensity                  2.318286
hue                              0.228572
od280/od315_of_diluted_wines     0.709990
proline                        314.907474
dtype: float64
```

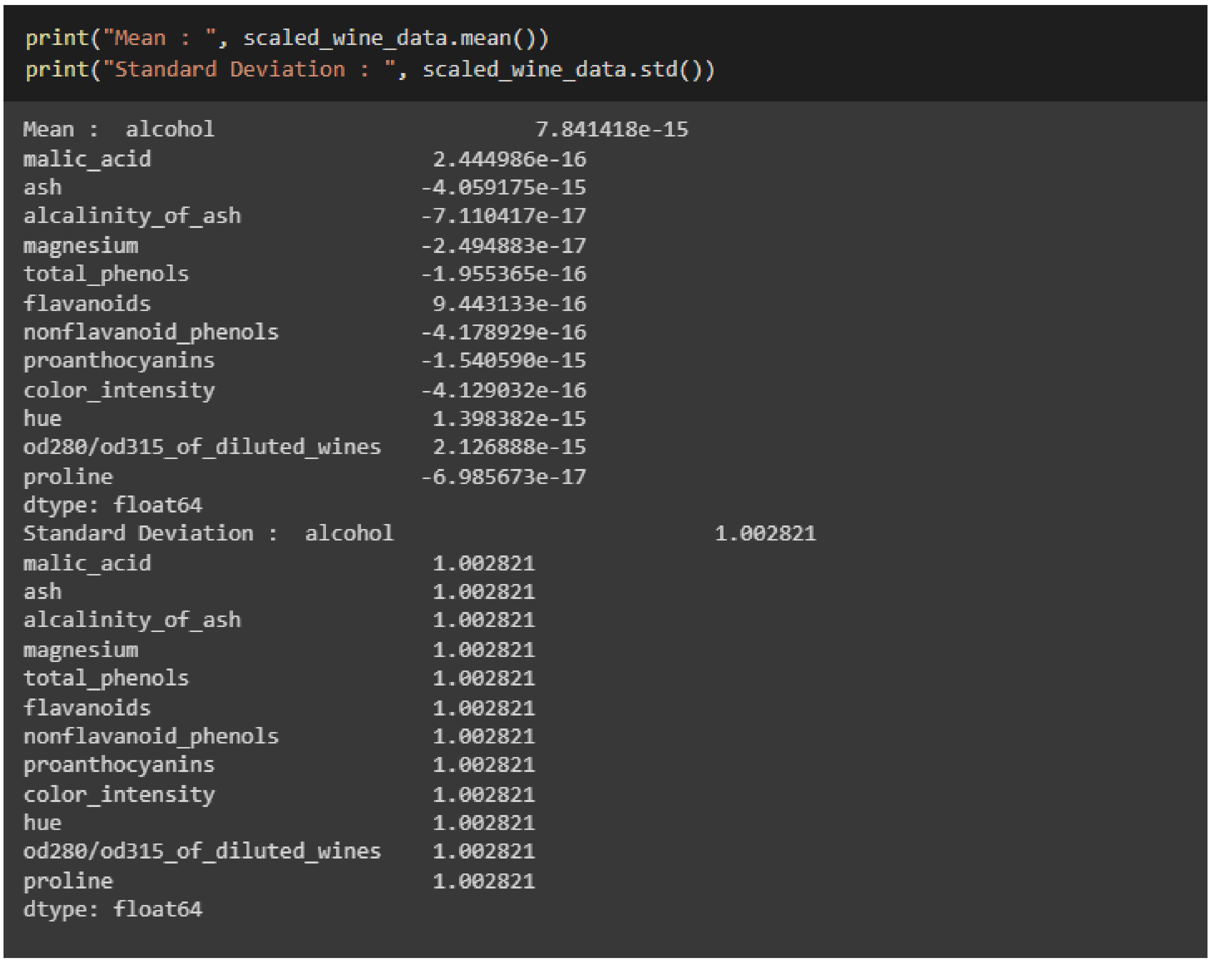


```
# standardizing wine dataset

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaled_data = scaler.fit_transform(X1)
```



```
scaled_wine_data = pd.DataFrame(scaled_data, columns=X1.columns)
scaled_wine_data.head()
```

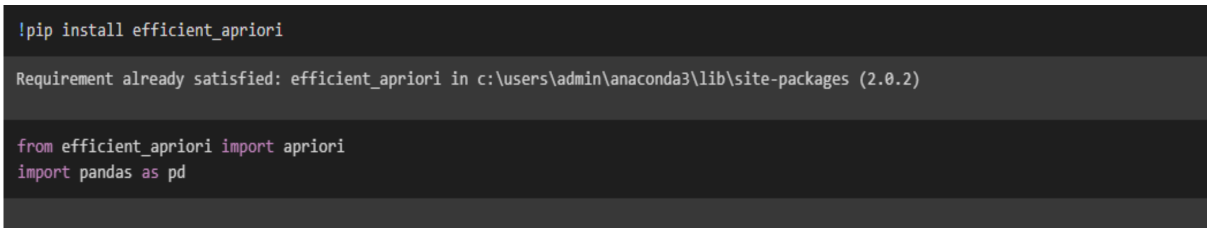
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od28
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819	-0.659563	1.224884	0.251717	0.362177	
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733629	-0.820719	-0.544721	-0.293321	0.406051	
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.215533	-0.498407	2.135968	0.269020	0.318304	
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.466525	-0.981875	1.032155	1.186068	-0.427544	
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351	0.226796	0.401404	-0.319276	0.362177	



```
print("Mean : ", scaled_wine_data.mean())
print("Standard Deviation : ", scaled_wine_data.std())
```

```
Mean :  alcohol                         7.841418e-15
malic_acid                      2.444986e-16
ash                            -4.059175e-15
alcalinity_of_ash              -7.110417e-17
magnesium                      -2.494883e-17
total_phenols                  -1.955365e-16
flavanoids                      9.443133e-16
nonflavanoid_phenols           -4.178929e-16
proanthocyanins                -1.540590e-15
color_intensity                -4.129032e-16
hue                             1.398382e-15
od280/od315_of_diluted_wines    2.126888e-15
proline                        -6.985673e-17
dtype: float64
Standard Deviation :  alcohol                         1.002821
malic_acid                      1.002821
ash                             1.002821
alcalinity_of_ash               1.002821
magnesium                       1.002821
total_phenols                   1.002821
flavanoids                      1.002821
nonflavanoid_phenols            1.002821
proanthocyanins                 1.002821
color_intensity                 1.002821
hue                             1.002821
od280/od315_of_diluted_wines    1.002821
proline                         1.002821
dtype: float64
```

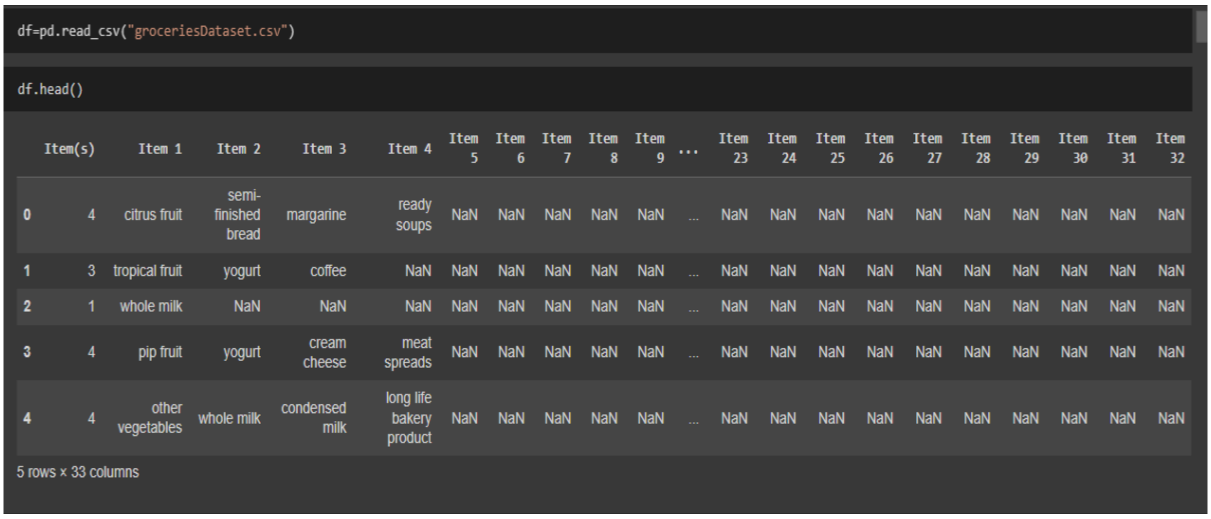
Q4. Run Apriori algorithm to find frequent itemsets and association rules



```
!pip install efficient_apriori
```

```
Requirement already satisfied: efficient_apriori in c:\users\admin\anaconda3\lib\site-packages (2.0.2)
```

```
from efficient_apriori import apriori
import pandas as pd
```

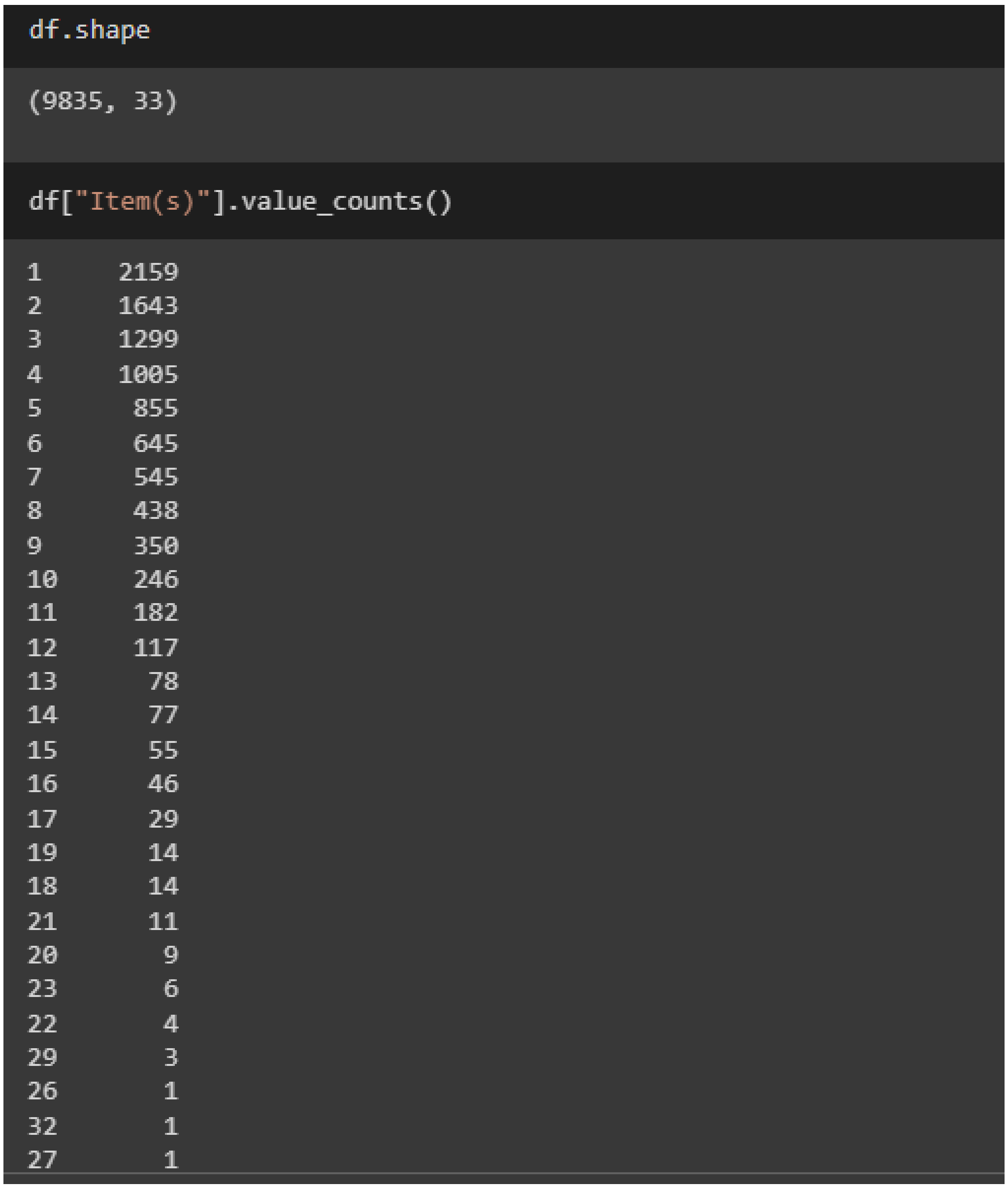


```
df=pd.read_csv("groceriesDataset.csv")
```

```
df.head()
```

	Item(s)	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	...	Item 23	Item 24	Item 25	Item 26	Item 27	Item 28	Item 29	Item 30	Item 31	Item 32
0	4	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	3	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 33 columns

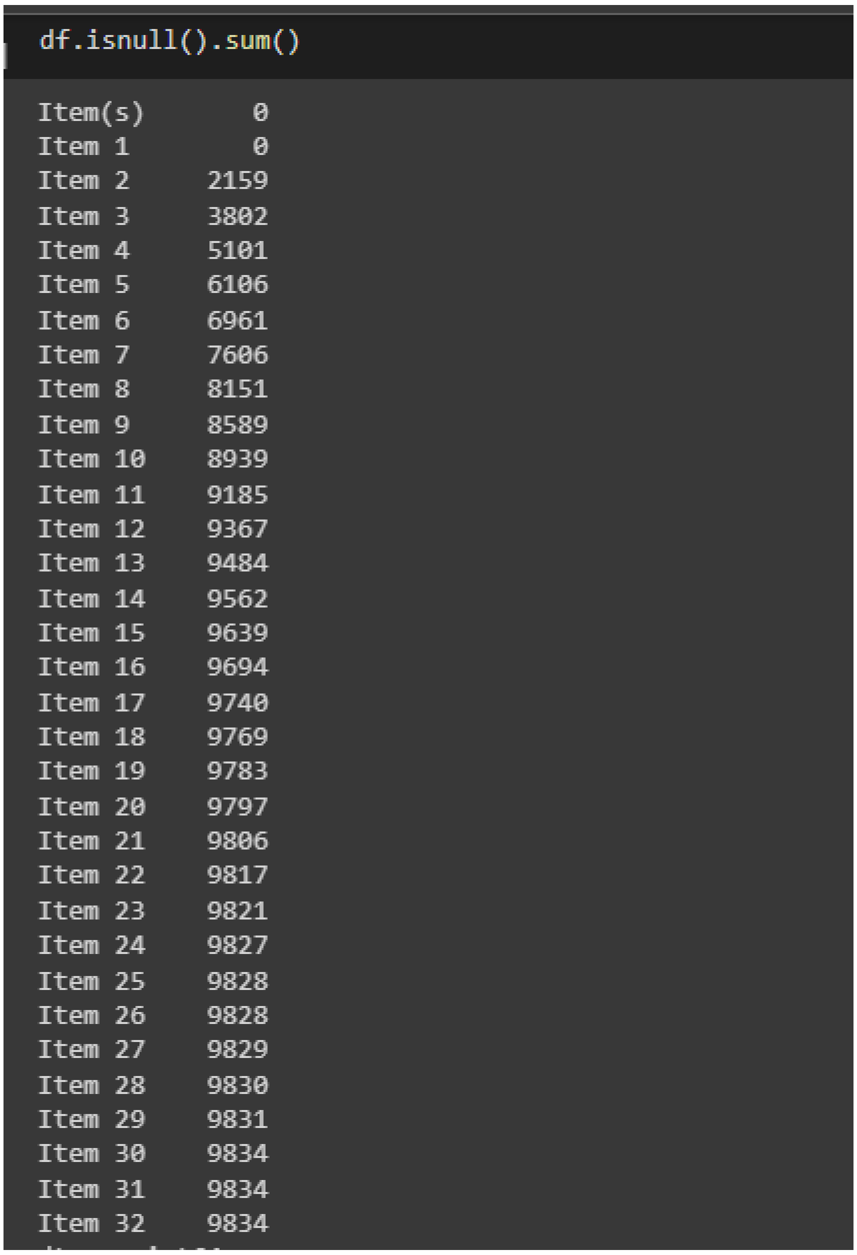


```
df.shape
```

```
(9835, 33)
```

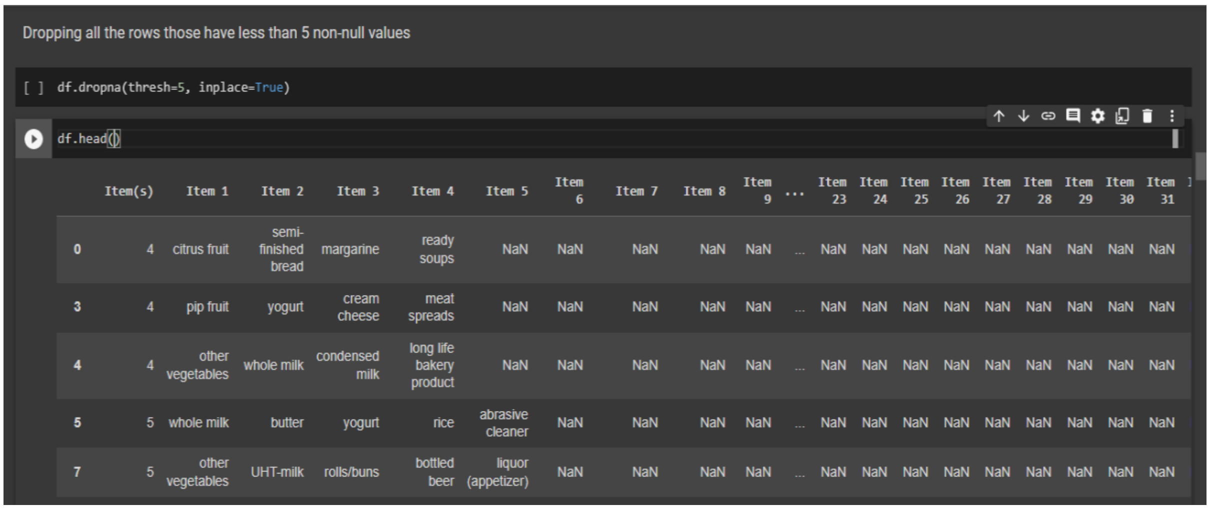
```
df["Item(s)"].value_counts()
```

```
1     2159
2     1643
3     1299
4     1005
5      855
6      645
7      545
8      438
9      350
10     246
11     182
12     117
13      78
14      77
15      55
16      46
17      29
19      14
18      14
21      11
20       9
23       6
22       4
29       3
26       1
32       1
27       1
```



```
df.isnull().sum()
```

```
Item(s)       0
Item 1        0
Item 2     2159
Item 3     3802
Item 4     5101
Item 5     6106
Item 6     6961
Item 7     7606
Item 8     8151
Item 9     8589
Item 10    8939
Item 11    9185
Item 12    9367
Item 13    9484
Item 14    9562
Item 15    9639
Item 16    9694
Item 17    9740
Item 18    9769
Item 19    9783
Item 20    9797
Item 21    9806
Item 22    9817
Item 23    9821
Item 24    9827
Item 25    9828
Item 26    9828
Item 27    9829
Item 28    9830
Item 29    9831
Item 30    9834
Item 31    9834
Item 32    9834
```

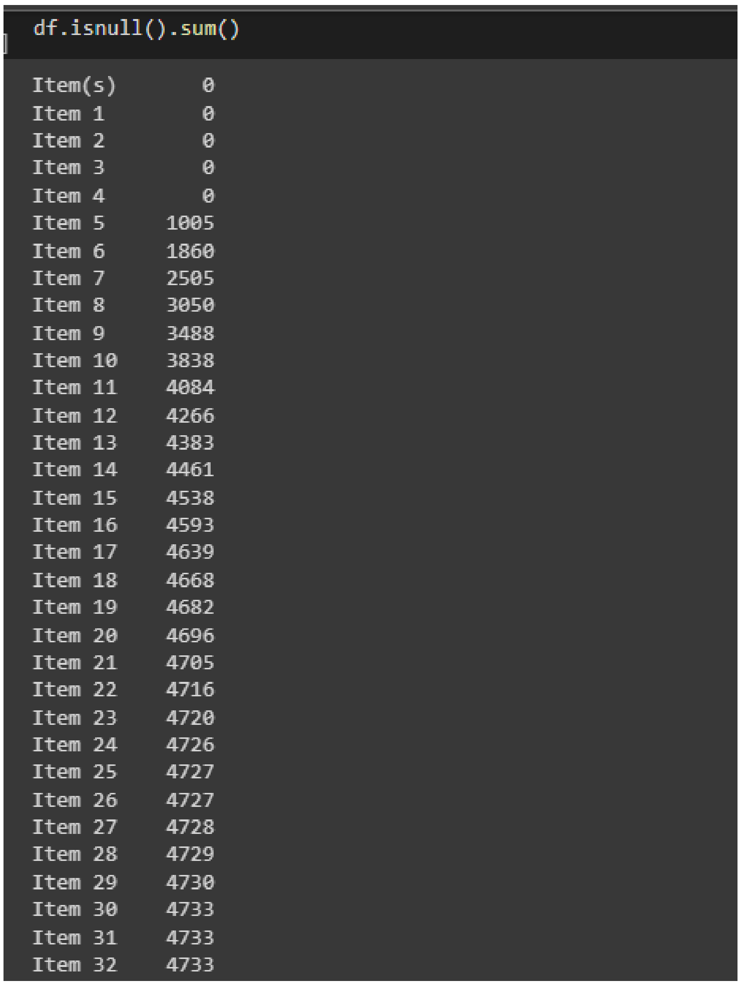


Dropping all the rows those have less than 5 non-null values

```
[ ] df.dropna(thresh=5, inplace=True)
```

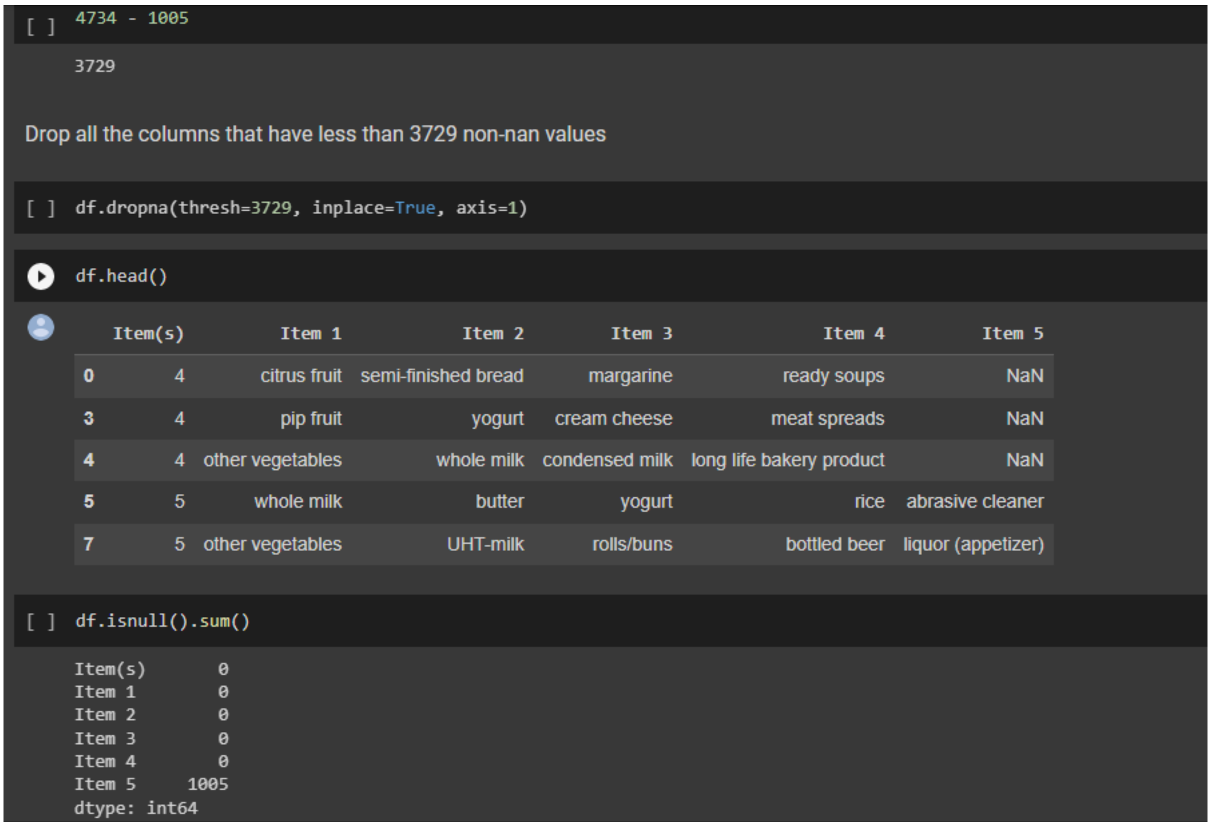
```
df.head()
```

	Item(s)	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	Item 9	...	Item 23	Item 24	Item 25	Item 26	Item 27	Item 28	Item 29	Item 30	Item 31
0	4	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	5	whole milk	butter	yogurt	rice	abrasive cleaner	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	5	other vegetables	UHT-milk	rolls/buns	bottled beer	liquor (appetizer)	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN



```
df.isnull().sum()
```

```
Item(s)       0
Item 1        0
Item 2        0
Item 3        0
Item 4        0
Item 5     1005
Item 6     1860
Item 7     2505
Item 8     3050
Item 9     3488
Item 10    3838
Item 11    4084
Item 12    4266
Item 13    4383
Item 14    4461
Item 15    4538
Item 16    4593
Item 17    4639
Item 18    4668
Item 19    4682
Item 20    4696
Item 21    4705
Item 22    4716
Item 23    4720
Item 24    4726
Item 25    4727
Item 26    4727
Item 27    4728
Item 28    4729
Item 29    4730
Item 30    4733
Item 31    4733
Item 32    4733
```



```
[ ] 4734 - 1005
```

```
3729
```

Drop all the columns that have less than 3729 non-nan values

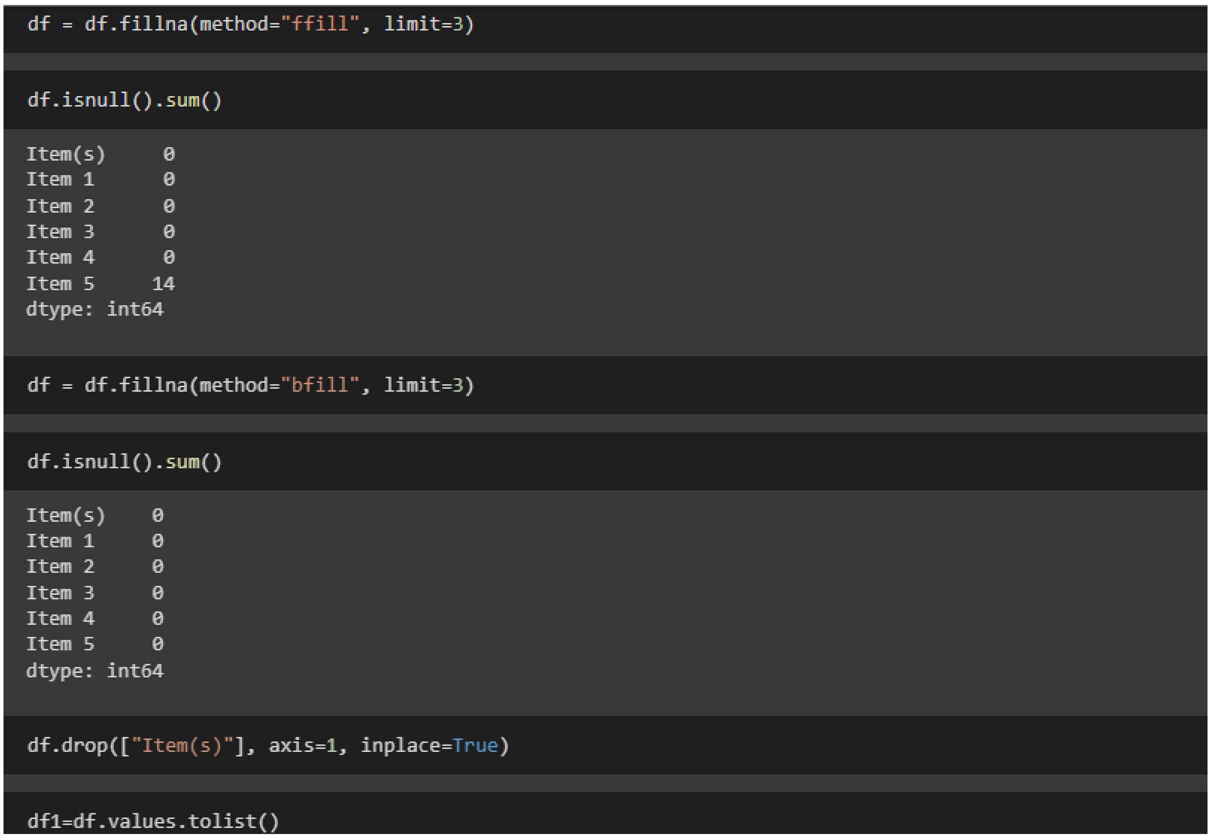
```
[ ] df.dropna(thresh=3729, inplace=True, axis=1)
```

```
df.head()
```

	Item(s)	Item 1	Item 2	Item 3	Item 4	Item 5
0	4	citrus fruit	semi-finished bread	margarine	ready soups	NaN
3	4	pip fruit	yogurt	cream cheese	meat spreads	NaN
4	4	other vegetables	whole milk	condensed milk	long life bakery product	NaN
5	5	whole milk	butter	yogurt	rice	abrasive cleaner
7	5	other vegetables	UHT-milk	rolls/buns	bottled beer	liquor (appetizer)

```
[ ] df.isnull().sum()
```

```
Item(s)       0
Item 1        0
Item 2        0
Item 3        0
Item 4        0
Item 5     1005
dtype: int64
```



```
df = df.fillna(method="ffill", limit=3)
```

```
df.isnull().sum()
```

```
Item(s)     0
Item 1      0
Item 2      0
Item 3      0
Item 4      0
Item 5     14
dtype: int64
```

```
df = df.fillna(method="bfill", limit=3)
```

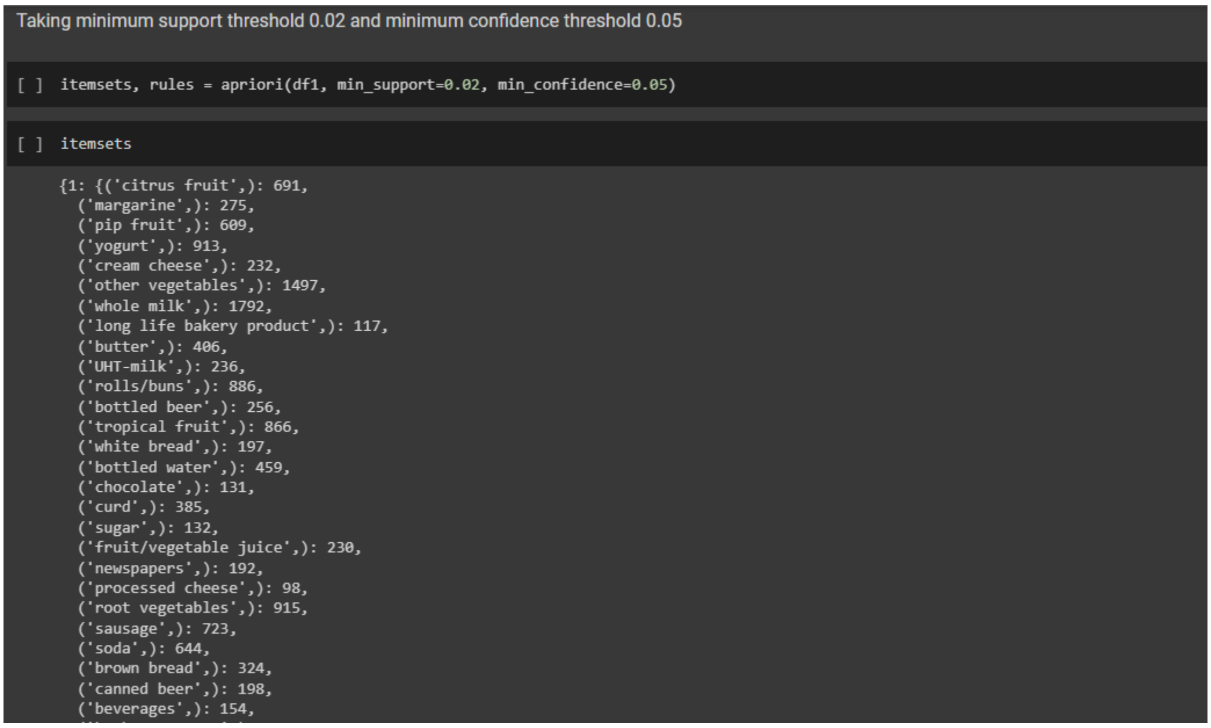
```
df.isnull().sum()
```

```
Item(s)    0
Item 1     0
Item 2     0
Item 3     0
Item 4     0
Item 5     0
dtype: int64
```

```
df.drop(["Item(s)"], axis=1, inplace=True)
```

```
df1=df.values.tolist()
```

4.1 Use minimum support as 2% and minimum confidence as 5%

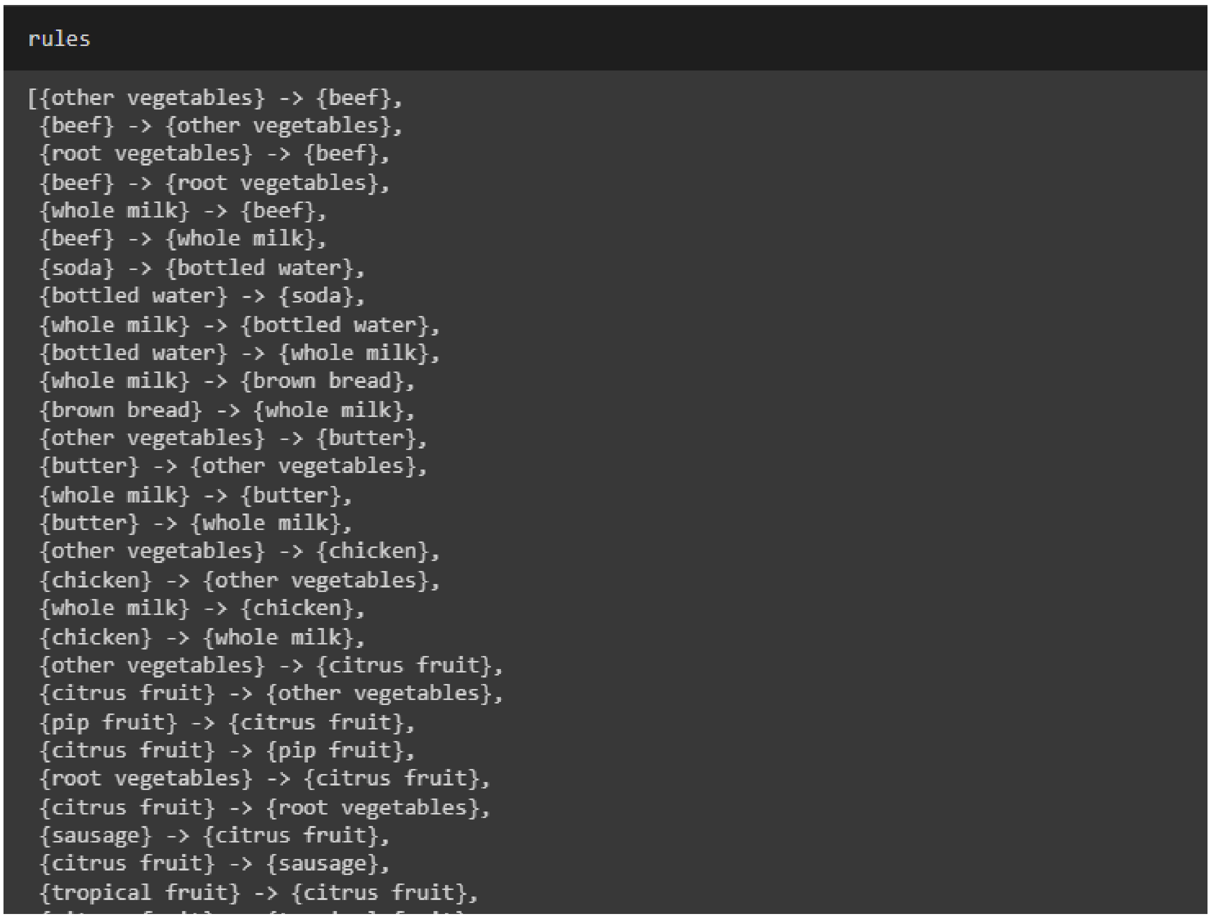


Taking minimum support threshold 0.02 and minimum confidence threshold 0.05

```
[ ]  itemsets, rules = apriori(df1, min_support=0.02, min_confidence=0.05)
```

```
[ ]  itemsets
```

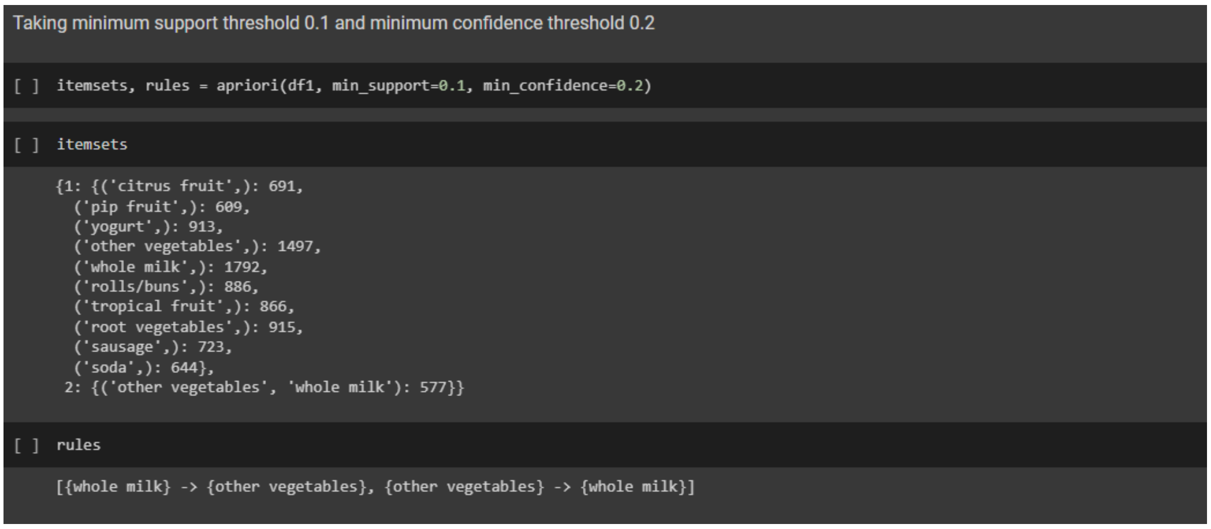
```
{1: {('citrus fruit',): 691,
  ('margarine',): 275,
  ('pip fruit',): 609,
  ('yogurt',): 913,
  ('cream cheese',): 232,
  ('other vegetables',): 1497,
  ('whole milk',): 1792,
  ('long life bakery product',): 117,
  ('butter',): 406,
  ('UHT-milk',): 236,
  ('rolls/buns',): 886,
  ('bottled beer',): 256,
  ('tropical fruit',): 866,
  ('white bread',): 197,
  ('bottled water',): 459,
  ('chocolate',): 131,
  ('curd',): 385,
  ('sugar',): 132,
  ('fruit/vegetable juice',): 230,
  ('newspapers',): 192,
  ('processed cheese',): 98,
  ('root vegetables',): 915,
  ('sausage',): 723,
  ('soda',): 644,
  ('brown bread',): 324,
  ('canned beer',): 198,
  ('beverages',): 154,
```



```
rules
```

```
[{other vegetables} -> {beef},
 {beef} -> {other vegetables},
 {root vegetables} -> {beef},
 {beef} -> {root vegetables},
 {whole milk} -> {beef},
 {beef} -> {whole milk},
 {soda} -> {bottled water},
 {bottled water} -> {soda},
 {whole milk} -> {bottled water},
 {bottled water} -> {whole milk},
 {whole milk} -> {brown bread},
 {brown bread} -> {whole milk},
 {other vegetables} -> {butter},
 {butter} -> {other vegetables},
 {whole milk} -> {butter},
 {butter} -> {whole milk},
 {other vegetables} -> {chicken},
 {chicken} -> {other vegetables},
 {whole milk} -> {chicken},
 {chicken} -> {whole milk},
 {other vegetables} -> {citrus fruit},
 {citrus fruit} -> {other vegetables},
 {pip fruit} -> {citrus fruit},
 {citrus fruit} -> {pip fruit},
 {root vegetables} -> {citrus fruit},
 {citrus fruit} -> {root vegetables},
 {sausage} -> {citrus fruit},
 {citrus fruit} -> {sausage},
 {tropical fruit} -> {citrus fruit},
```

4.2 Use minimum support as 10% and minimum confidence as 20 %



Taking minimum support threshold 0.1 and minimum confidence threshold 0.2

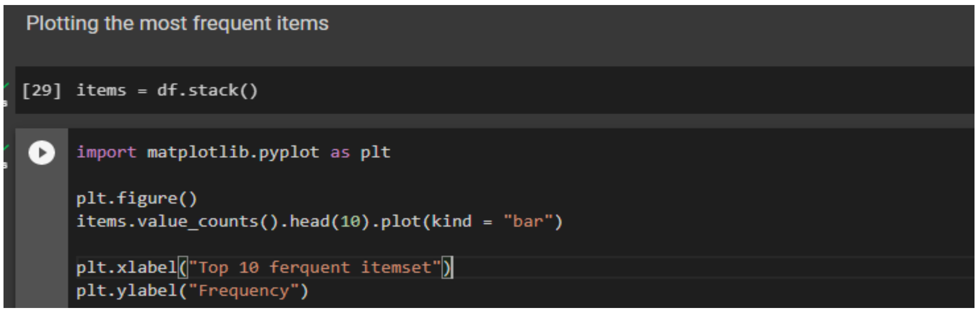
```
[ ] itemsets, rules = apriori(df1, min_support=0.1, min_confidence=0.2)
```

```
[ ] itemsets
```

```
{1: {('citrus fruit',): 691,
  ('pip fruit',): 609,
  ('yogurt',): 913,
  ('other vegetables',): 1497,
  ('whole milk',): 1792,
  ('rolls/buns',): 886,
  ('tropical fruit',): 866,
  ('root vegetables',): 915,
  ('sausage',): 723,
  ('soda',): 644},
 2: {('other vegetables', 'whole milk'): 577}}
```

```
[ ] rules
```

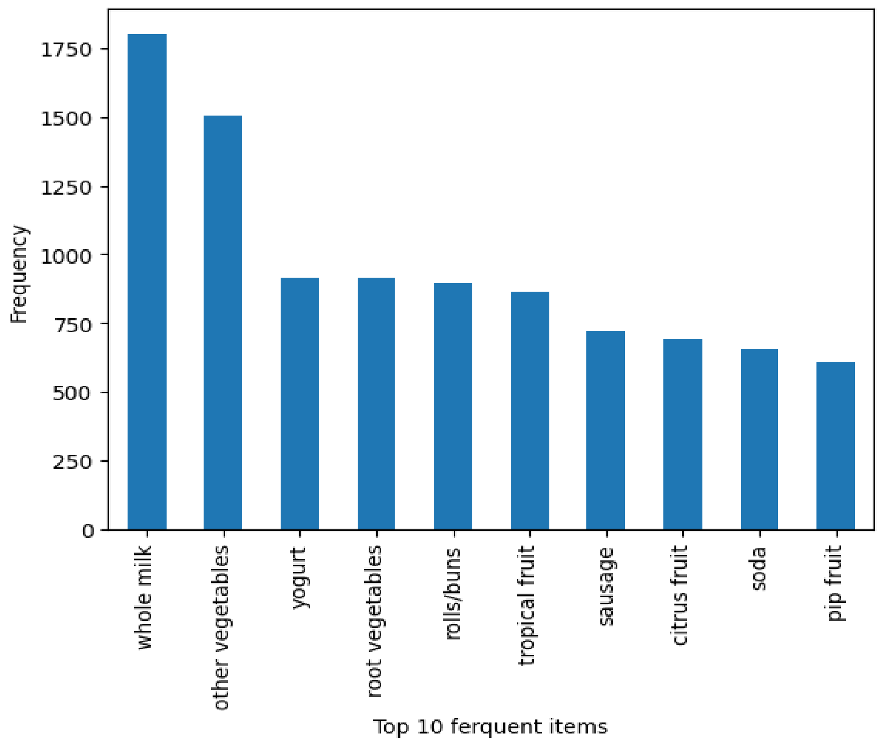
```
[{whole milk} -> {other vegetables}, {other vegetables} -> {whole milk}]
```



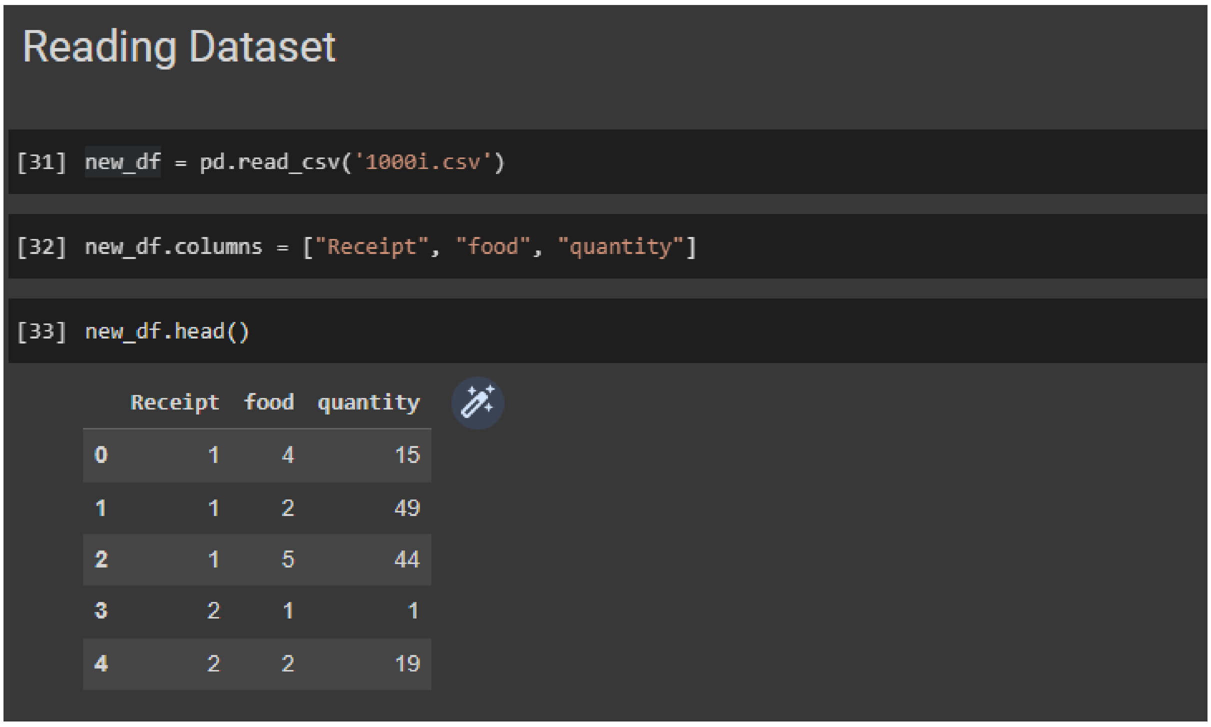
Plotting the most frequent items

```
[29] items = df.stack()
```

```
import matplotlib.pyplot as plt

plt.figure()
items.value_counts().head(10).plot(kind = "bar")

plt.xlabel("Top 10 ferquent itemset")
plt.ylabel("Frequency")
```



Another Dataset



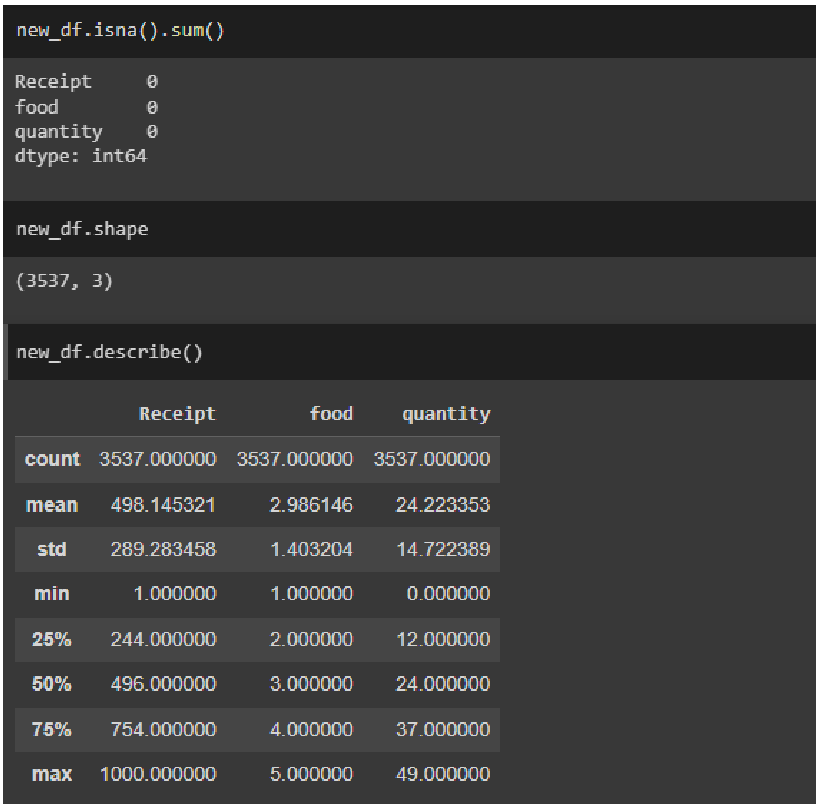
Reading Dataset

```
[31] new_df = pd.read_csv('1000i.csv')
```

```
[32] new_df.columns = ["Receipt", "food", "quantity"]
```

```
[33] new_df.head()
```

	Receipt	food	quantity
0	1	4	15
1	1	2	49
2	1	5	44
3	2	1	1
4	2	2	19



```python
new_df.isna().sum()
```

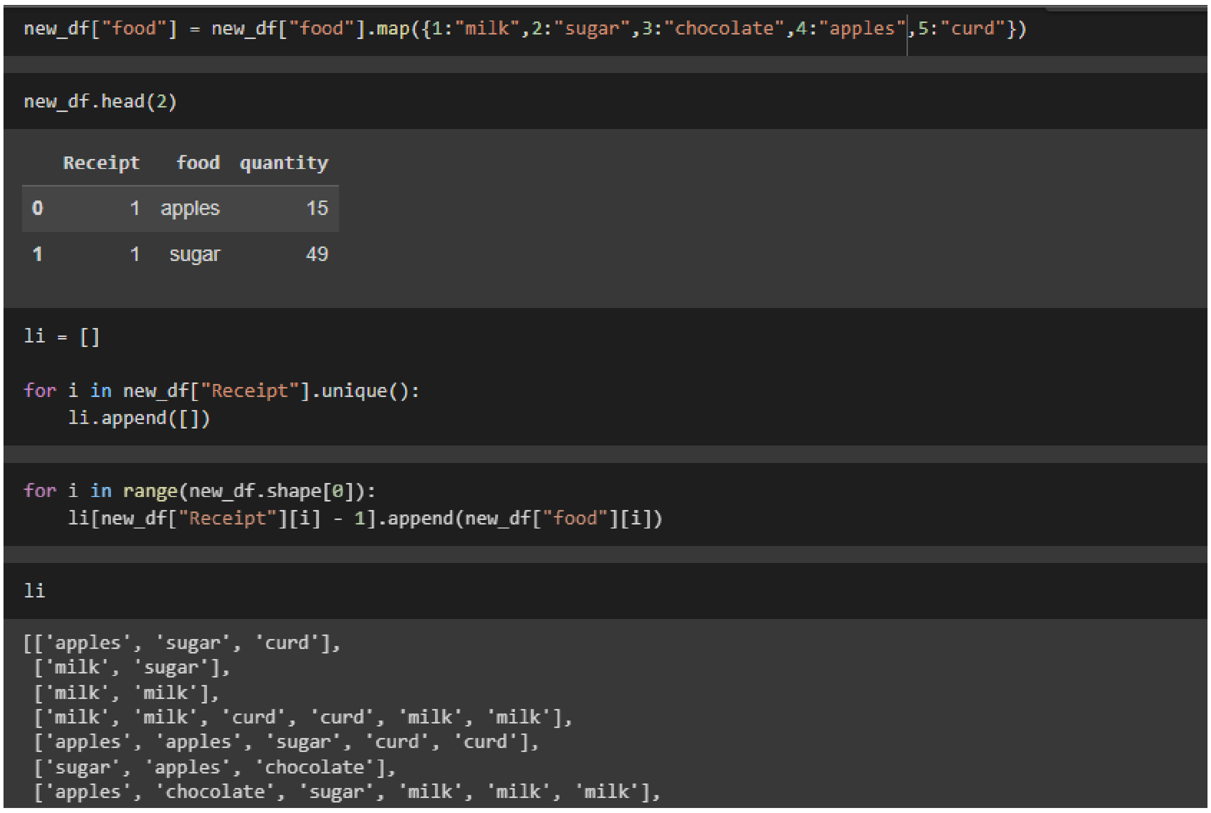
```
Receipt     0
food        0
quantity    0
dtype: int64
```

```python
new_df.shape
```

```
(3537, 3)
```

```python
new_df.describe()
```

	Receipt	food	quantity
count	3537.000000	3537.000000	3537.000000
mean	498.145321	2.986146	24.223353
std	289.283458	1.403204	14.722389
min	1.000000	1.000000	0.000000
25%	244.000000	2.000000	12.000000
50%	496.000000	3.000000	24.000000
75%	754.000000	4.000000	37.000000
max	1000.000000	5.000000	49.000000



```python
new_df["food"] = new_df["food"].map({1:"milk",2:"sugar",3:"chocolate",4:"apples",5:"curd"})
```

```python
new_df.head(2)
```

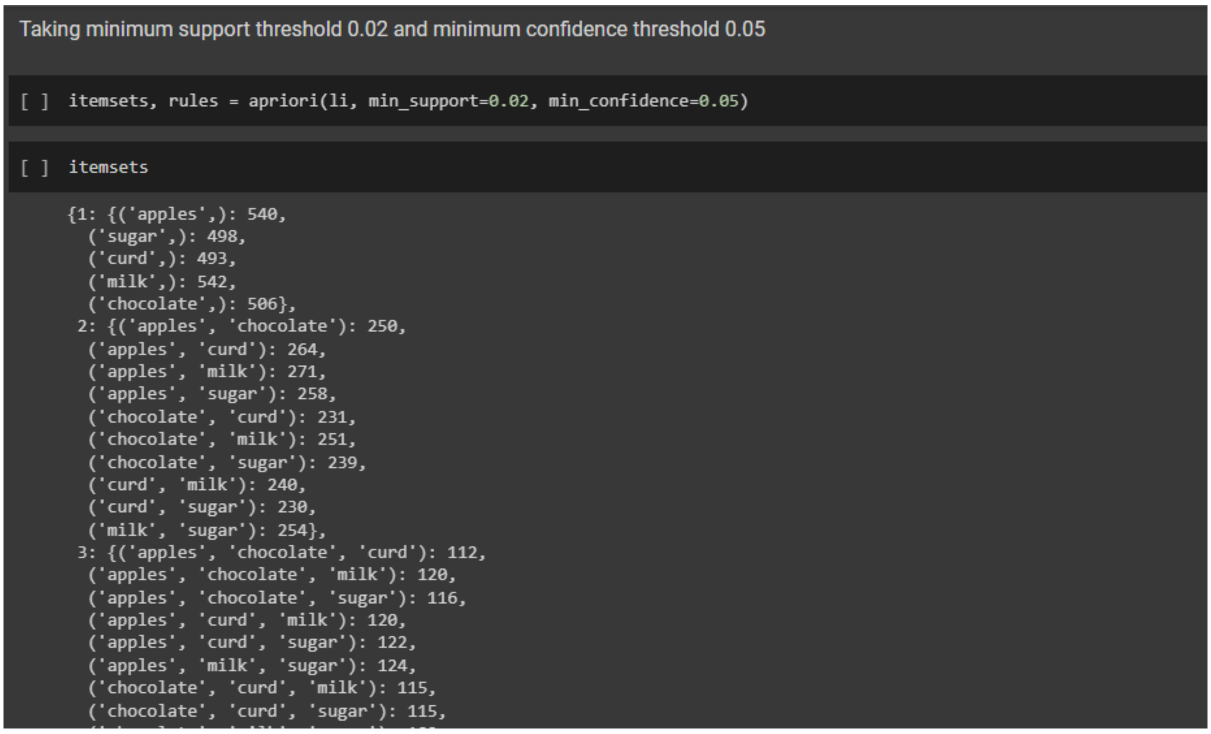
	Receipt	food	quantity
0	1	apples	15
1	1	sugar	49

```python
li = []

for i in new_df["Receipt"].unique():
    li.append([])
```

```python
for i in range(new_df.shape[0]):
    li[new_df["Receipt"][i] - 1].append(new_df["food"][i])
```

```python
li
```

```
[['apples', 'sugar', 'curd'],
 ['milk', 'sugar'],
 ['milk', 'milk'],
 ['milk', 'milk', 'curd', 'curd', 'milk', 'milk'],
 ['apples', 'apples', 'sugar', 'curd', 'curd'],
 ['sugar', 'apples', 'chocolate'],
 ['apples', 'chocolate', 'sugar', 'milk', 'milk', 'milk'],
```

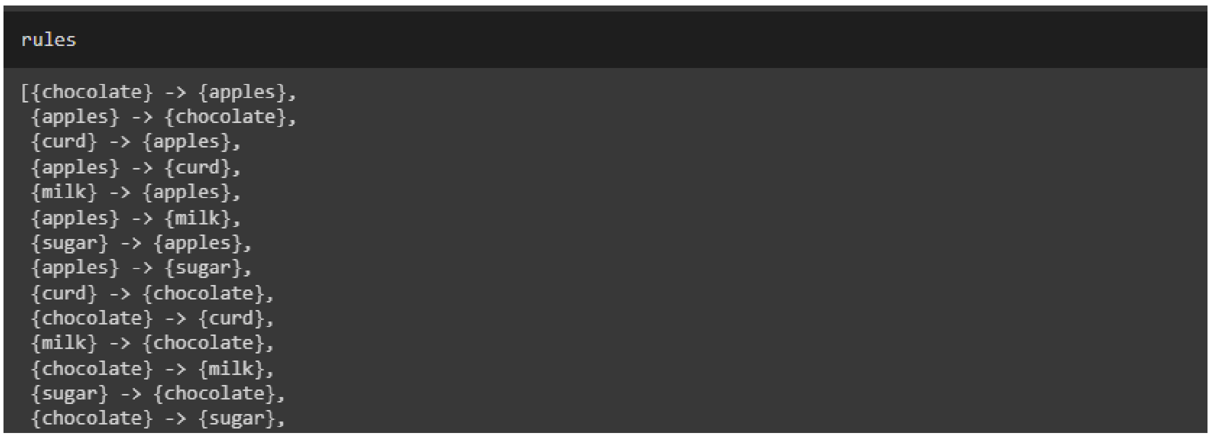


Taking minimum support threshold 0.02 and minimum confidence threshold 0.05

```
[ ]  itemsets, rules = apriori(li, min_support=0.02, min_confidence=0.05)
```

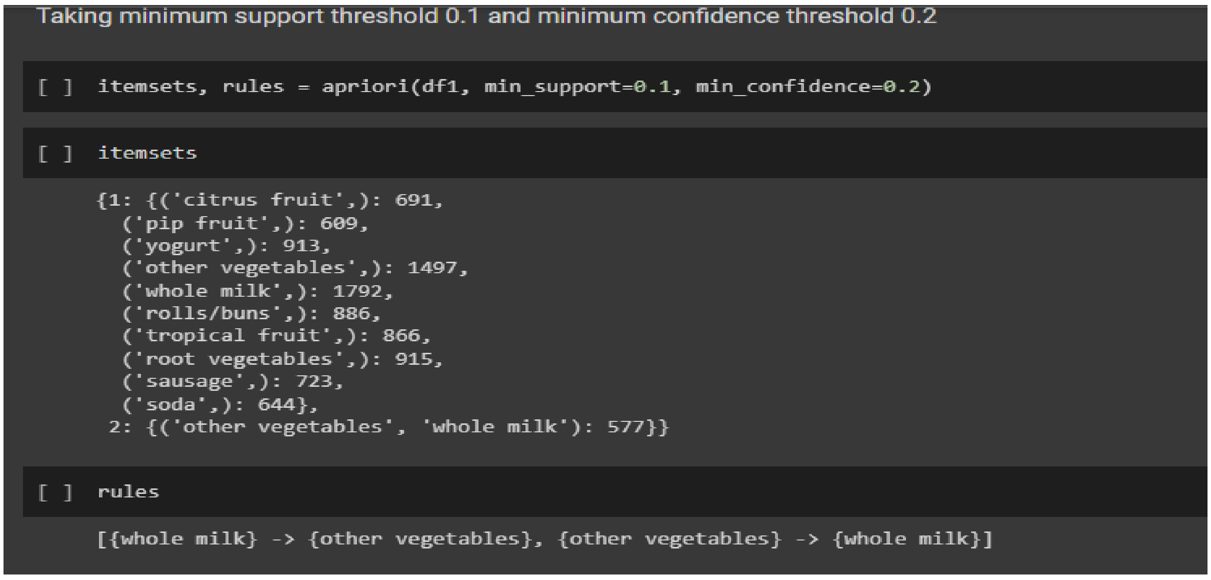
```
[ ]  itemsets
```

```
{1: {('apples',): 540,
  ('sugar',): 498,
  ('curd',): 493,
  ('milk',): 542,
  ('chocolate',): 506},
 2: {('apples', 'chocolate'): 250,
  ('apples', 'curd'): 264,
  ('apples', 'milk'): 271,
  ('apples', 'sugar'): 258,
  ('chocolate', 'curd'): 231,
  ('chocolate', 'milk'): 251,
  ('chocolate', 'sugar'): 239,
  ('curd', 'milk'): 240,
  ('curd', 'sugar'): 230,
  ('milk', 'sugar'): 254},
 3: {('apples', 'chocolate', 'curd'): 112,
  ('apples', 'chocolate', 'milk'): 120,
  ('apples', 'chocolate', 'sugar'): 116,
  ('apples', 'curd', 'milk'): 120,
  ('apples', 'curd', 'sugar'): 122,
  ('apples', 'milk', 'sugar'): 124,
  ('chocolate', 'curd', 'milk'): 115,
  ('chocolate', 'curd', 'sugar'): 115,
```



```
rules
```

```
[{chocolate} -> {apples},
 {apples} -> {chocolate},
 {curd} -> {apples},
 {apples} -> {curd},
 {milk} -> {apples},
 {apples} -> {milk},
 {sugar} -> {apples},
 {apples} -> {sugar},
 {curd} -> {chocolate},
 {chocolate} -> {curd},
 {milk} -> {chocolate},
 {chocolate} -> {milk},
 {sugar} -> {chocolate},
 {chocolate} -> {sugar},
```



Taking minimum support threshold 0.1 and minimum confidence threshold 0.2

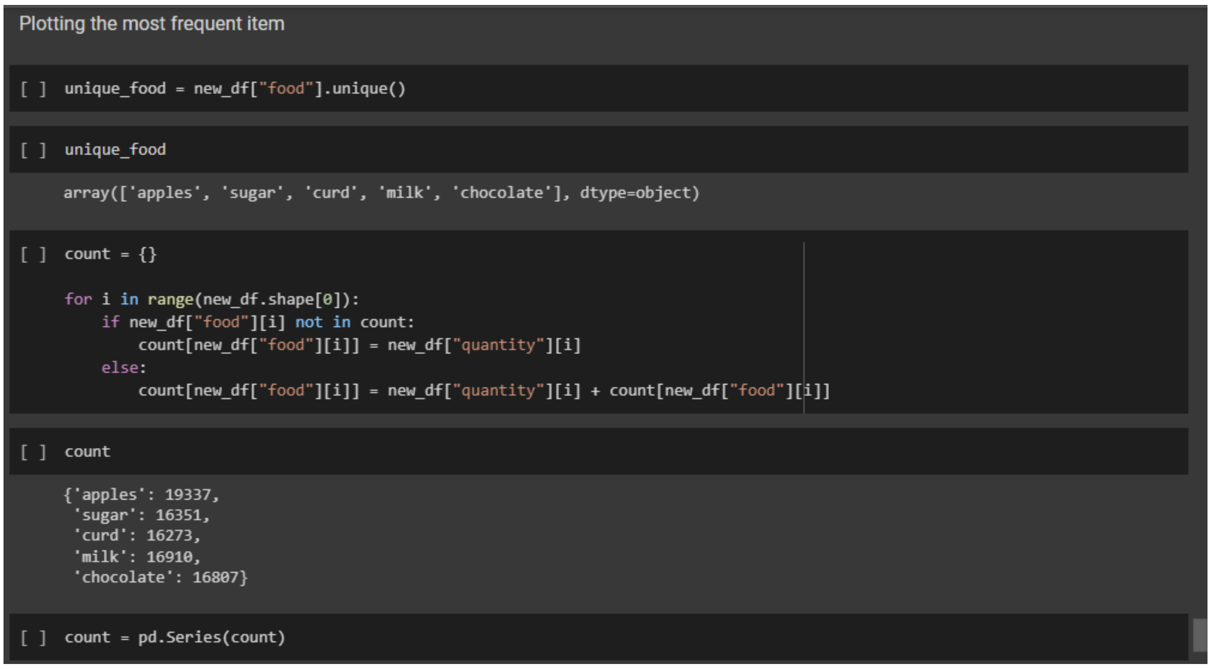
```
[ ]  itemsets, rules = apriori(df1, min_support=0.1, min_confidence=0.2)
```

```
[ ]  itemsets
```

```
{1: {('citrus fruit',): 691,
  ('pip fruit',): 609,
  ('yogurt',): 913,
  ('other vegetables',): 1497,
  ('whole milk',): 1792,
  ('rolls/buns',): 886,
  ('tropical fruit',): 866,
  ('root vegetables',): 915,
  ('sausage',): 723,
  ('soda',): 644},
 2: {('other vegetables', 'whole milk'): 577}}
```

```
[ ]  rules
```

```
[{whole milk} -> {other vegetables}, {other vegetables} -> {whole milk}]
```



Plotting the most frequent item

```
[ ] unique_food = new_df["food"].unique()
```

```
[ ] unique_food
```

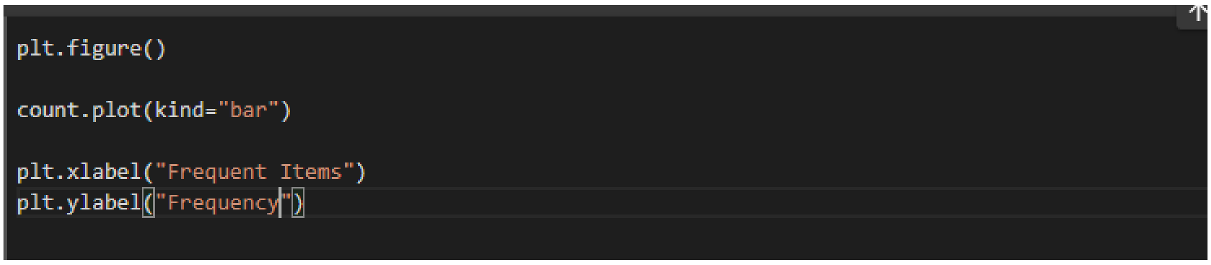
```
array(['apples', 'sugar', 'curd', 'milk', 'chocolate'], dtype=object)
```

```
[ ] count = {}

    for i in range(new_df.shape[0]):
        if new_df["food"][i] not in count:
            count[new_df["food"][i]] = new_df["quantity"][i]
        else:
            count[new_df["food"][i]] = new_df["quantity"][i] + count[new_df["food"][i]]
```

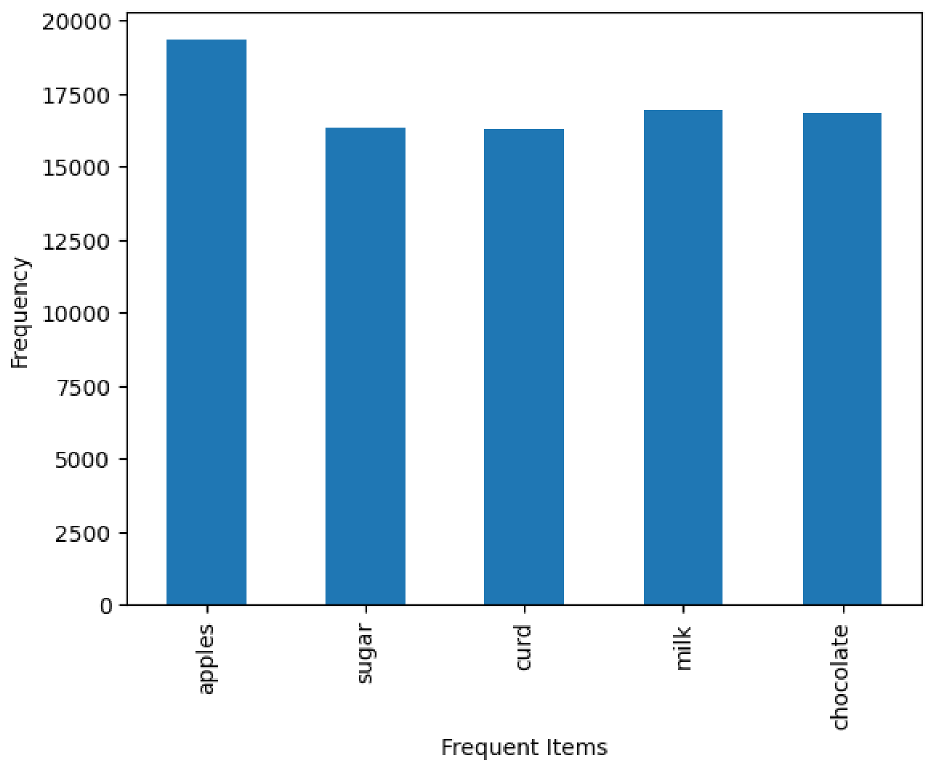
```
[ ] count
```

```
{'apples': 19337,
 'sugar': 16351,
 'curd': 16273,
 'milk': 16910,
 'chocolate': 16807}
```

```
[ ] count = pd.Series(count)
```



```
plt.figure()

count.plot(kind="bar")

plt.xlabel("Frequent Items")
plt.ylabel("Frequency")
```



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

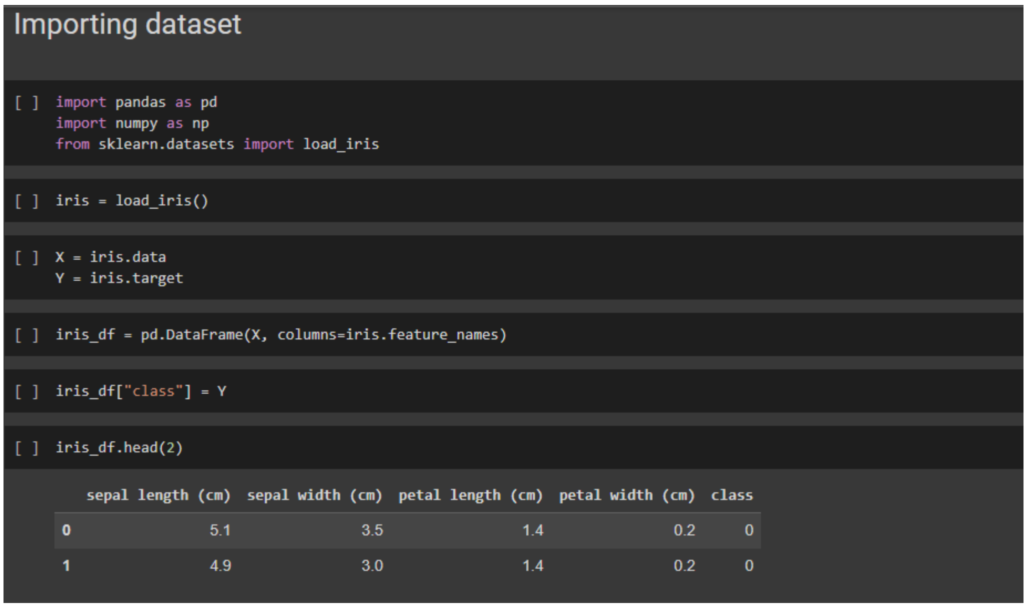
i) hold out method

ii) Random subsampling

iii) Cross-Validation. Compare the accuracy of the classifiers obtained.

5.3 Data is scaled to standard format.

For Iris Dataset



Importing dataset

```
[ ] import pandas as pd
    import numpy as np
    from sklearn.datasets import load_iris
```

```
[ ] iris = load_iris()
```

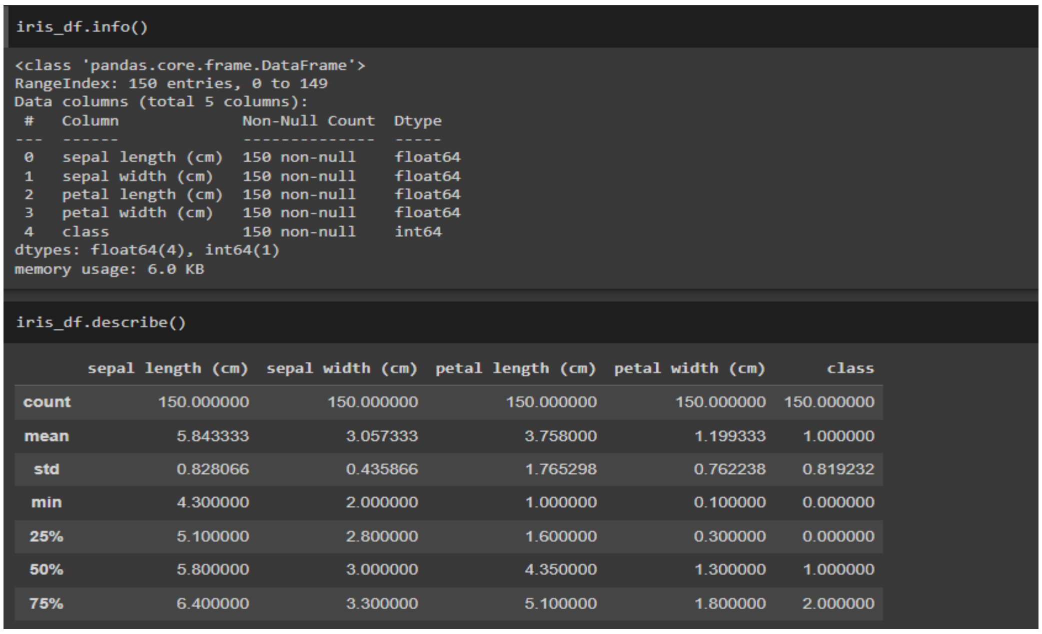
```
[ ] X = iris.data
    Y = iris.target
```

```
[ ] iris_df = pd.DataFrame(X, columns=iris.feature_names)
```

```
[ ] iris_df["class"] = Y
```

```
[ ] iris_df.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0

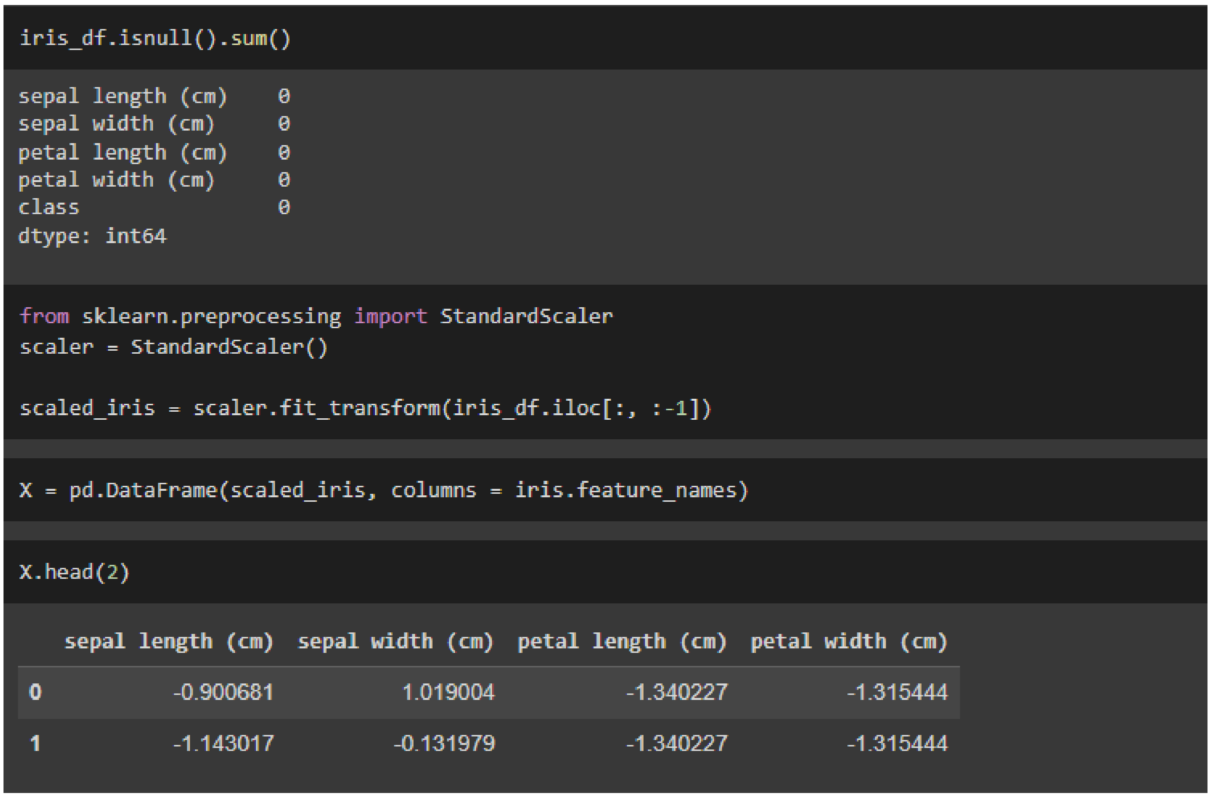


```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   class              150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

```
iris_df.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000



```
iris_df.isnull().sum()
```

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
class                0
dtype: int64
```

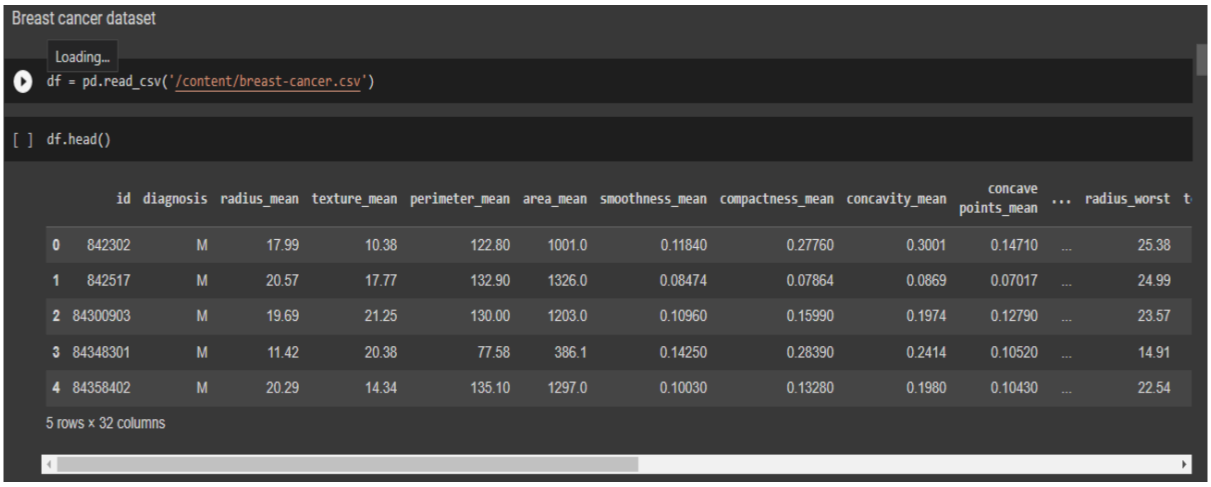
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaled_iris = scaler.fit_transform(iris_df.iloc[:, :-1])
```

```
X = pd.DataFrame(scaled_iris, columns = iris.feature_names)
```

```
X.head(2)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444

For Breast Cancer Dataset:



Breast cancer dataset

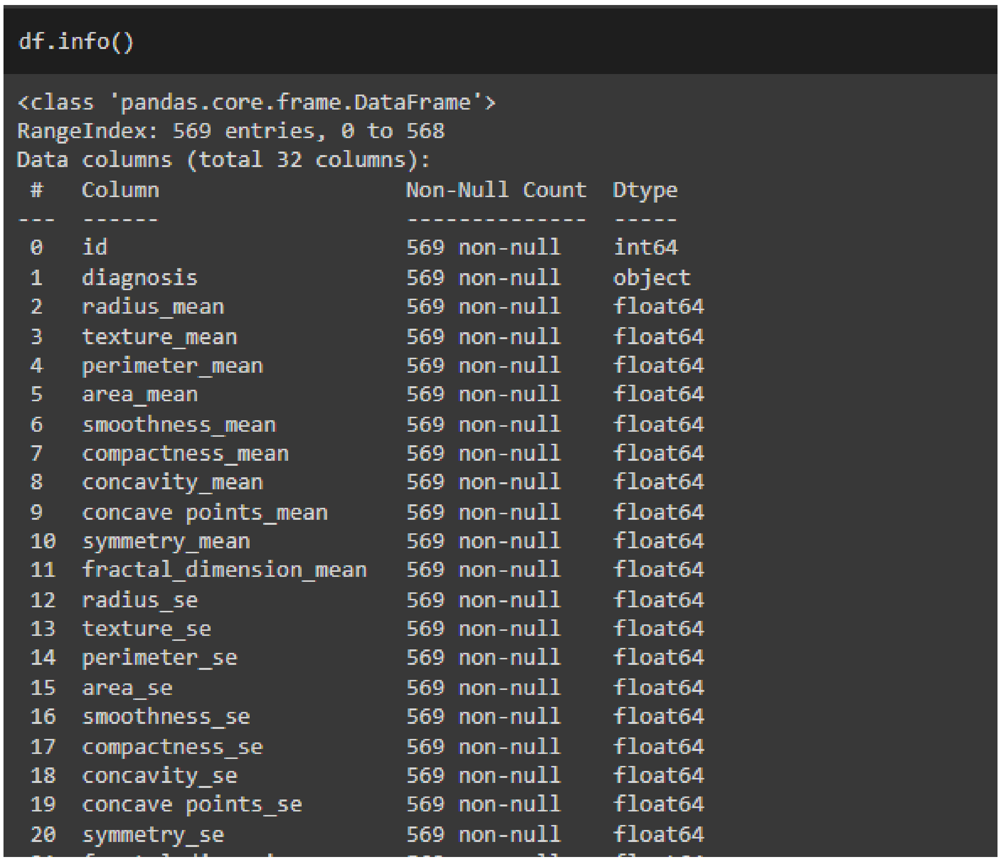
Loading...

```
df = pd.read_csv('/content/breast-cancer.csv')
```

```
[ ] df.head()
```

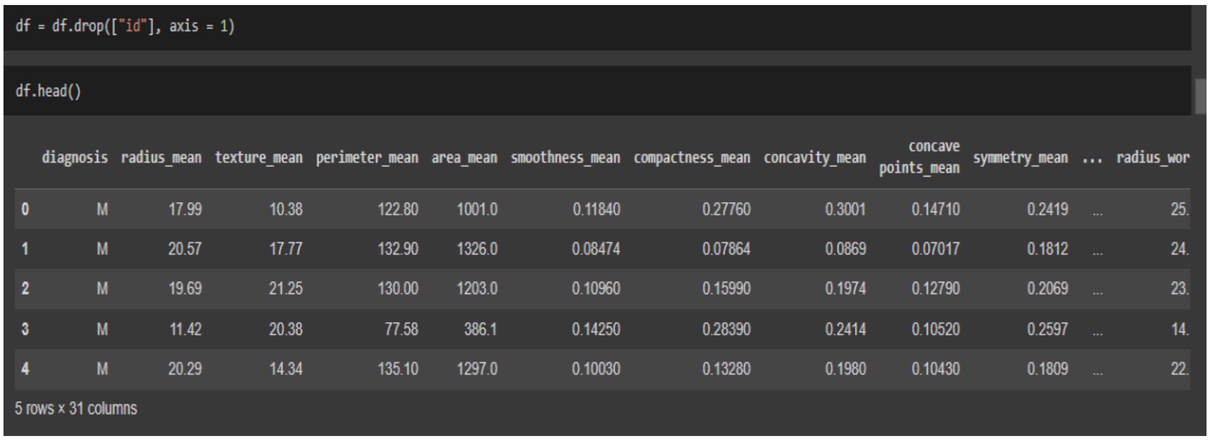
	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	t
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	

5 rows × 32 columns



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
```

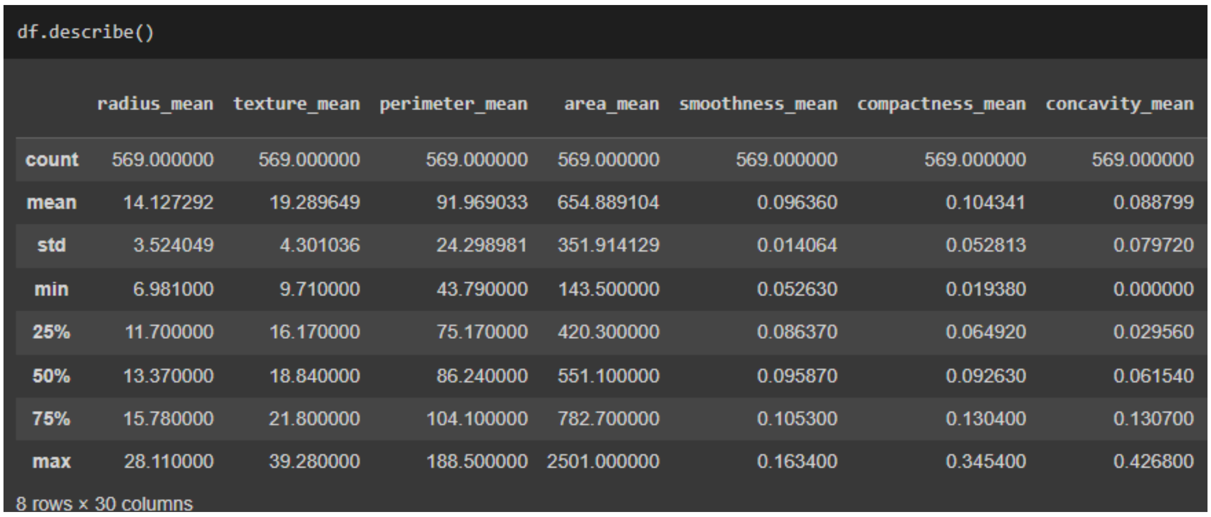


```
df = df.drop(["id"], axis = 1)
```

```
df.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_wor
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.

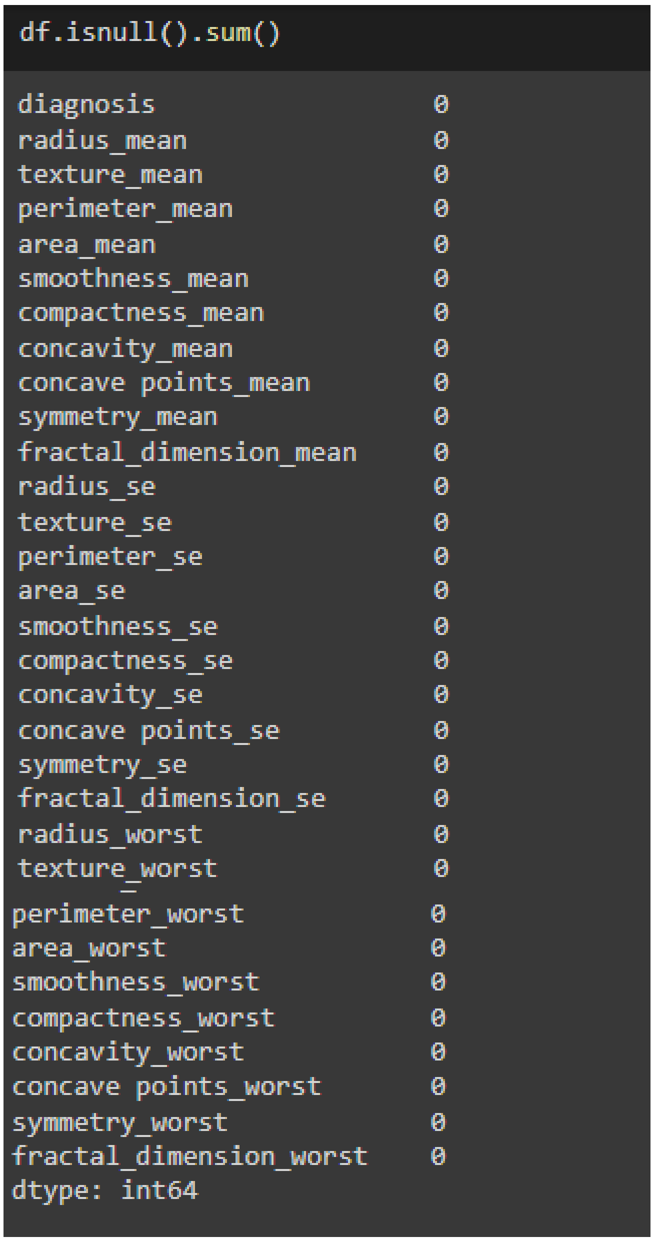
5 rows × 31 columns



```
df.describe()
```

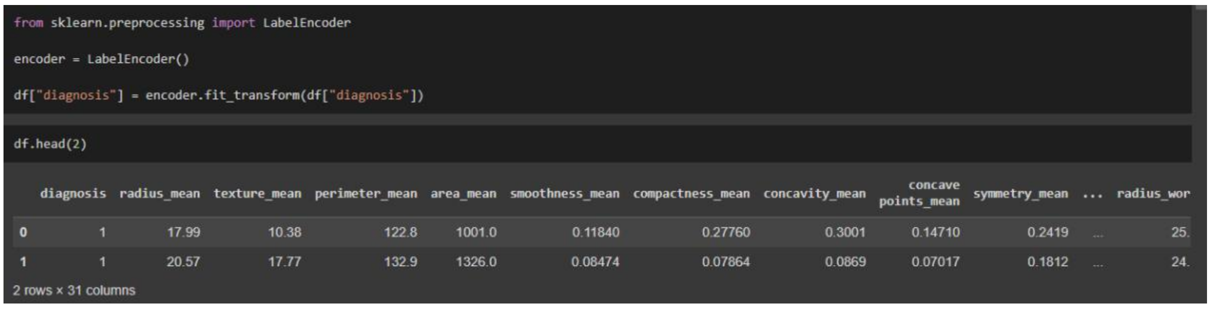
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800

8 rows × 30 columns



```
df.isnull().sum()
```

```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

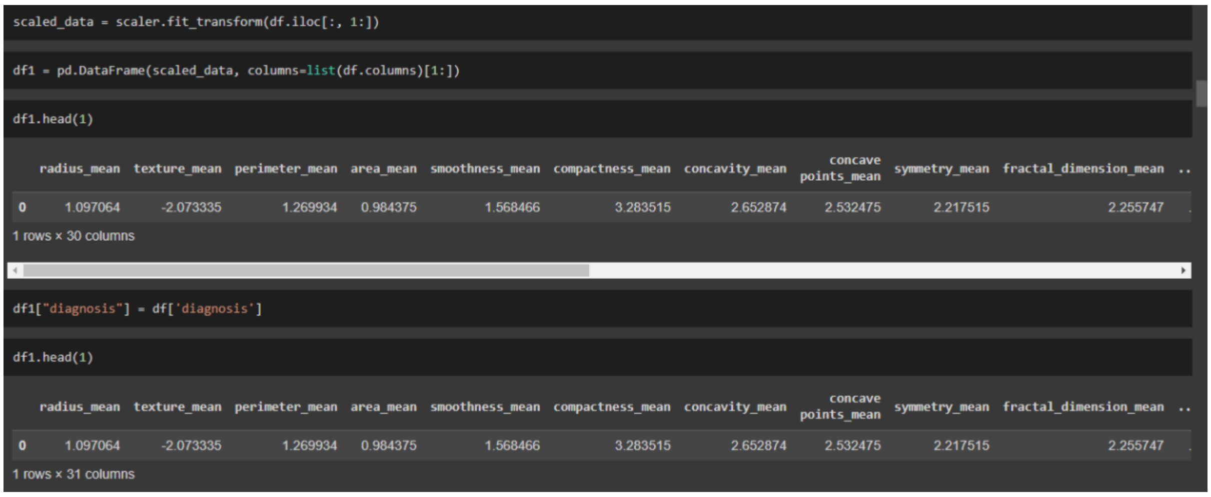


```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df["diagnosis"] = encoder.fit_transform(df["diagnosis"])
```

```
df.head(2)
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_wor
0	1	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.
1	1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.

2 rows × 31 columns



```
scaled_data = scaler.fit_transform(df.iloc[:, 1:])
```

```
df1 = pd.DataFrame(scaled_data, columns=list(df.columns)[1:])
```

```
df1.head(1)
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	..
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	.

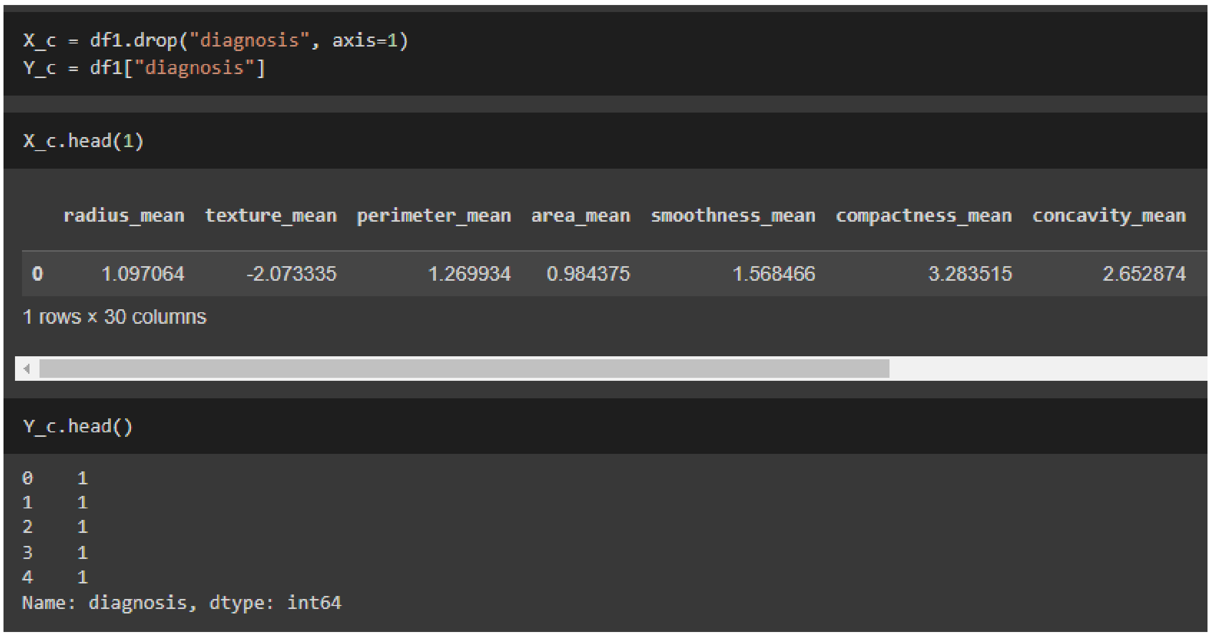
1 rows × 30 columns

```
df1["diagnosis"] = df['diagnosis']
```

```
df1.head(1)
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	..
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	.

1 rows × 31 columns



```
X_c = df1.drop("diagnosis", axis=1)
Y_c = df1["diagnosis"]
```

```
X_c.head(1)
```

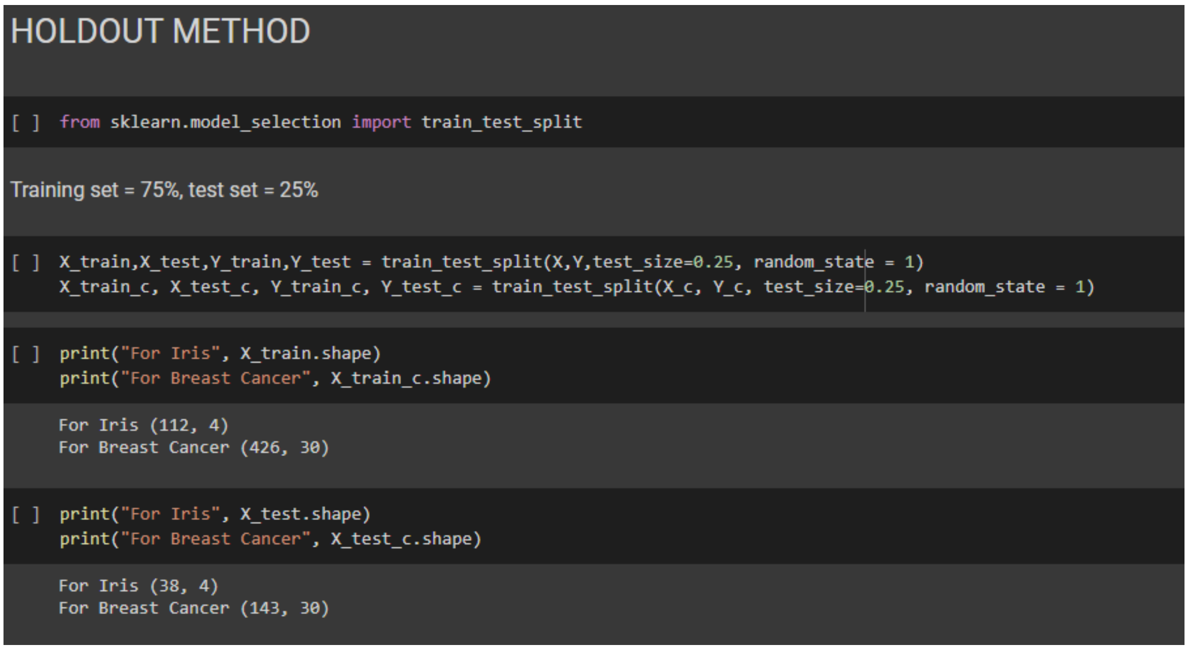
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874

1 rows × 30 columns

```
Y_c.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: diagnosis, dtype: int64
```

Holdout Method



HOLDOUT METHOD

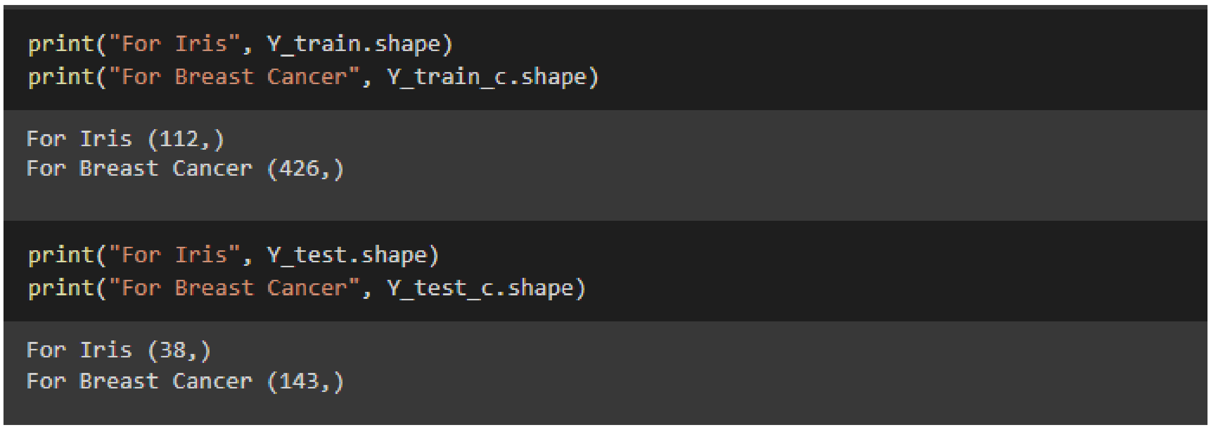
```
[ ]  from sklearn.model_selection import train_test_split
```

Training set = 75%, test set = 25%

```
[ ]  X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25, random_state = 1)
     X_train_c, X_test_c, Y_train_c, Y_test_c = train_test_split(X_c, Y_c, test_size=0.25, random_state = 1)
```

```
[ ]  print("For Iris", X_train.shape)
     print("For Breast Cancer", X_train_c.shape)

     For Iris (112, 4)
     For Breast Cancer (426, 30)
```

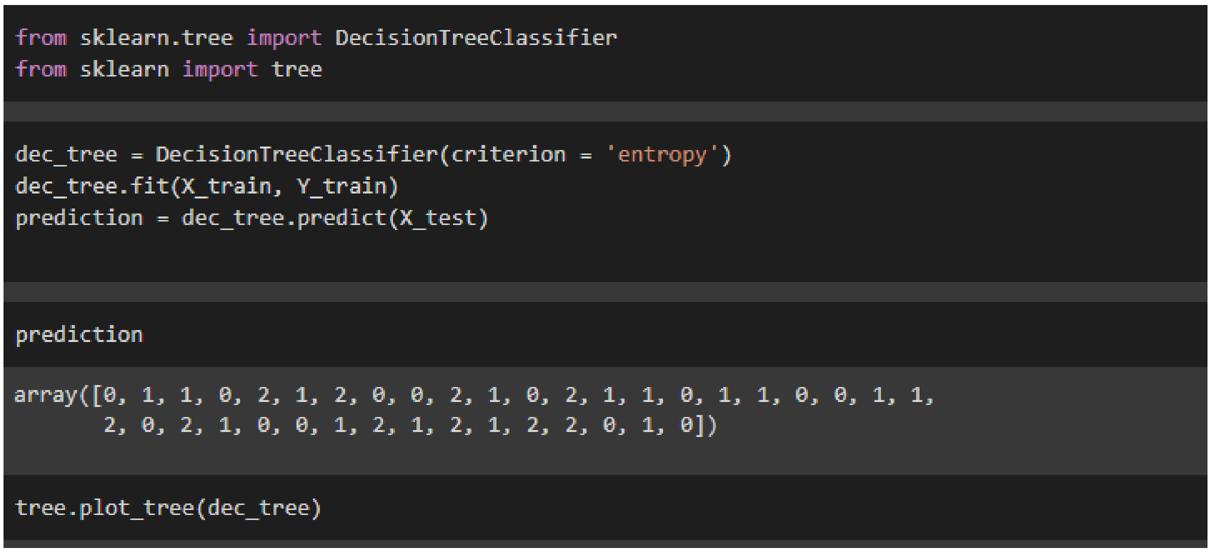
```
[ ]  print("For Iris", X_test.shape)
     print("For Breast Cancer", X_test_c.shape)

     For Iris (38, 4)
     For Breast Cancer (143, 30)
```



```
print("For Iris", Y_train.shape)
print("For Breast Cancer", Y_train_c.shape)

For Iris (112,)
For Breast Cancer (426,)
```

```
print("For Iris", Y_test.shape)
print("For Breast Cancer", Y_test_c.shape)

For Iris (38,)
For Breast Cancer (143,)
```

Decision Tree (On Iris dataset).

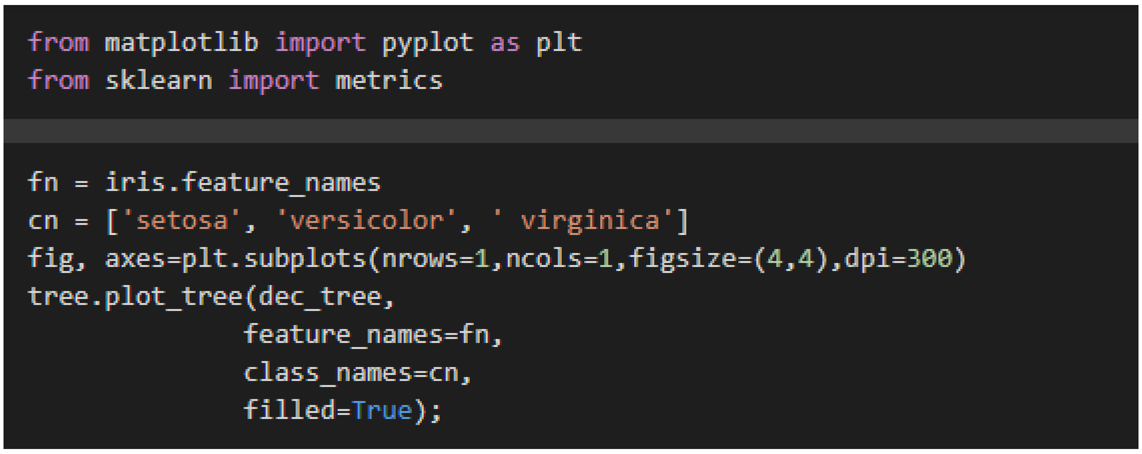


```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
dec_tree = DecisionTreeClassifier(criterion = 'entropy')
dec_tree.fit(X_train, Y_train)
prediction = dec_tree.predict(X_test)
```

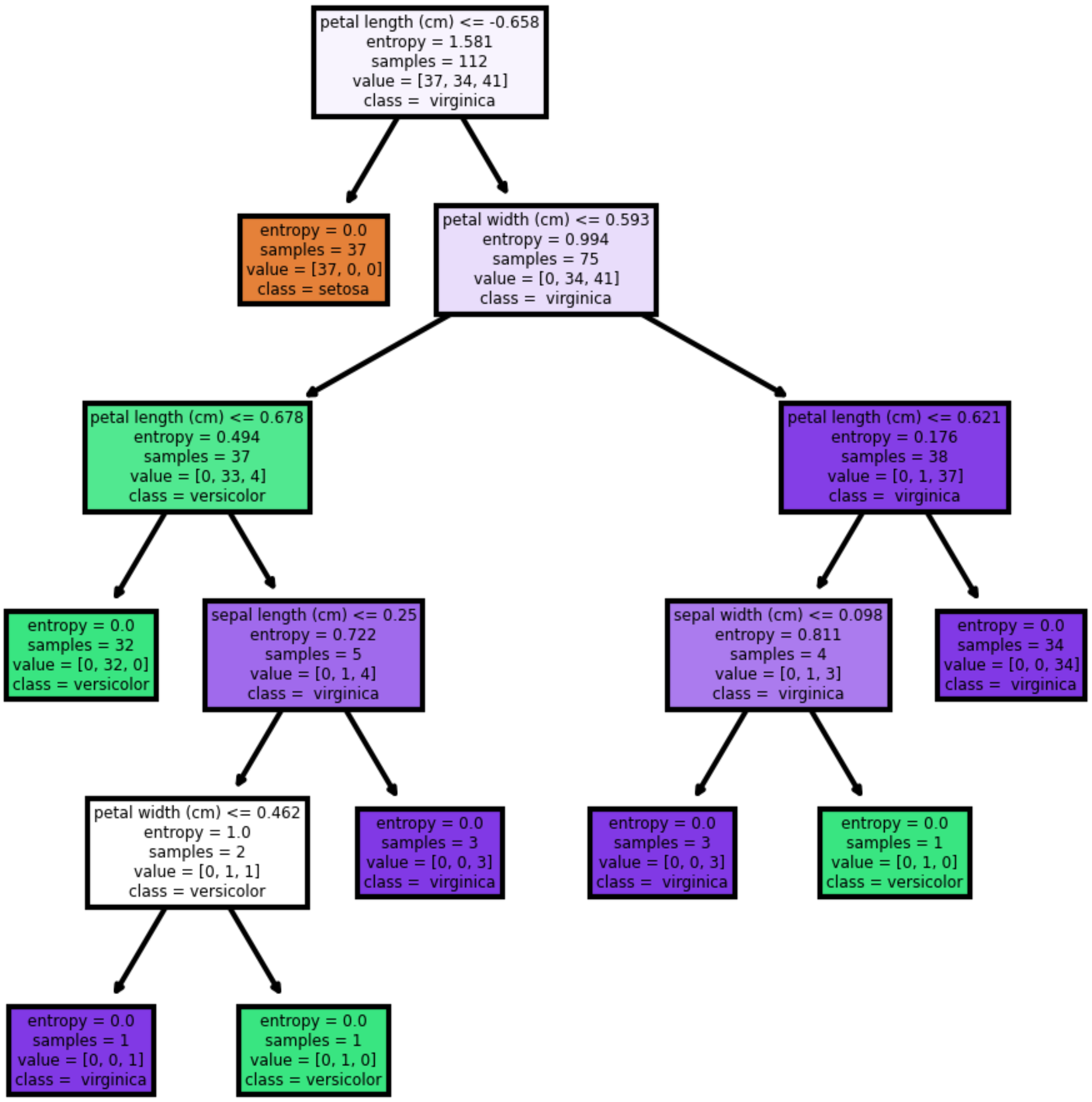
```
prediction

array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0])
```

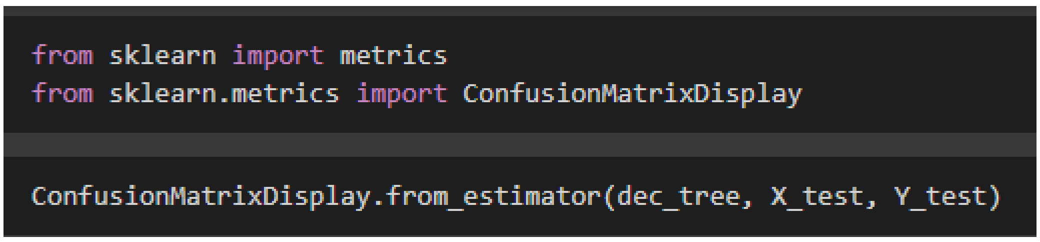
```
tree.plot_tree(dec_tree)
```



```python
from matplotlib import pyplot as plt
from sklearn import metrics
```

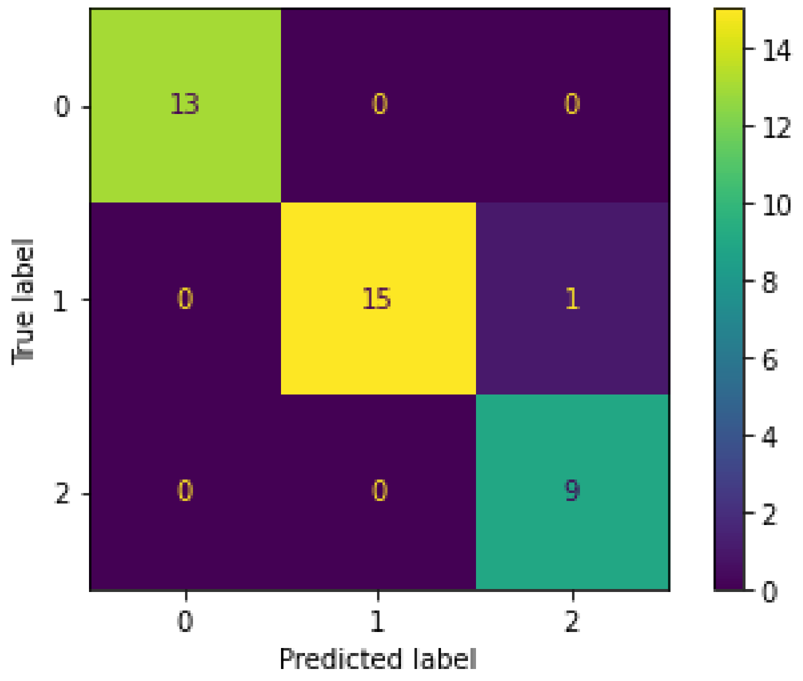
```python
fn = iris.feature_names
cn = ['setosa', 'versicolor', ' virginica']
fig, axes=plt.subplots(nrows=1,ncols=1,figsize=(4,4),dpi=300)
tree.plot_tree(dec_tree,
               feature_names=fn,
               class_names=cn,
               filled=True);
```

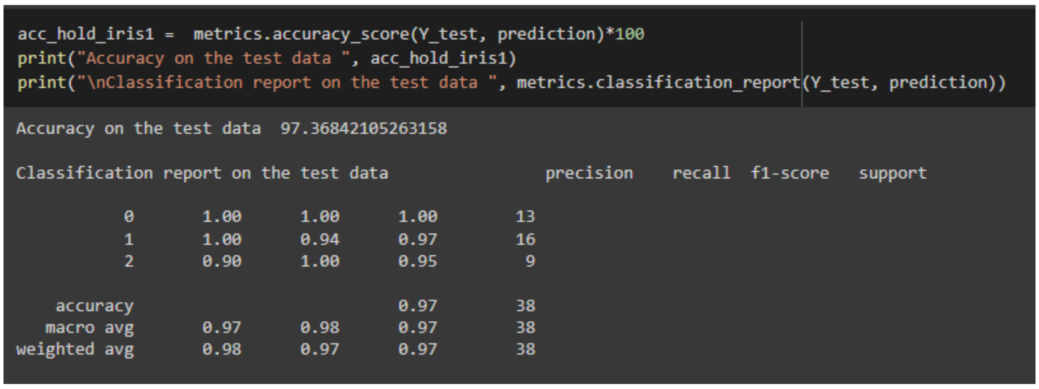




```
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay
```

```
ConfusionMatrixDisplay.from_estimator(dec_tree, X_test, Y_test)
```

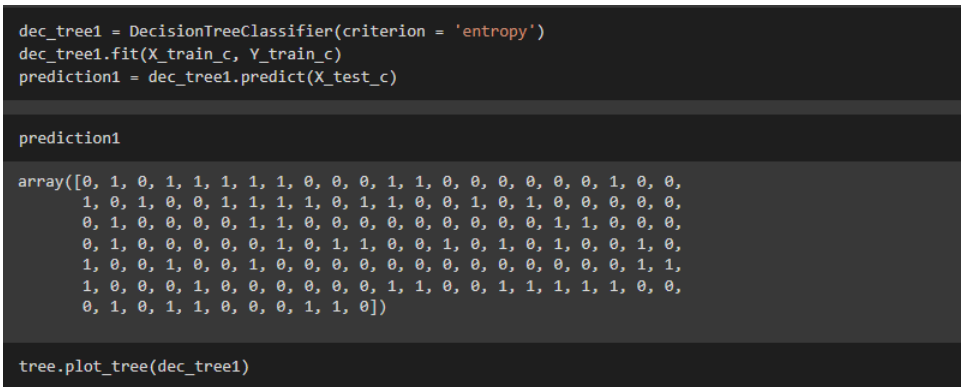




```
acc_hold_iris1 =  metrics.accuracy_score(Y_test, prediction)*100
print("Accuracy on the test data ", acc_hold_iris1)
print("\nClassification report on the test data ", metrics.classification_report(Y_test, prediction))
```

```
Accuracy on the test data  97.36842105263158

Classification report on the test data                precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        16
           2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

Decision Tree (On Breast Cancer Dataset)



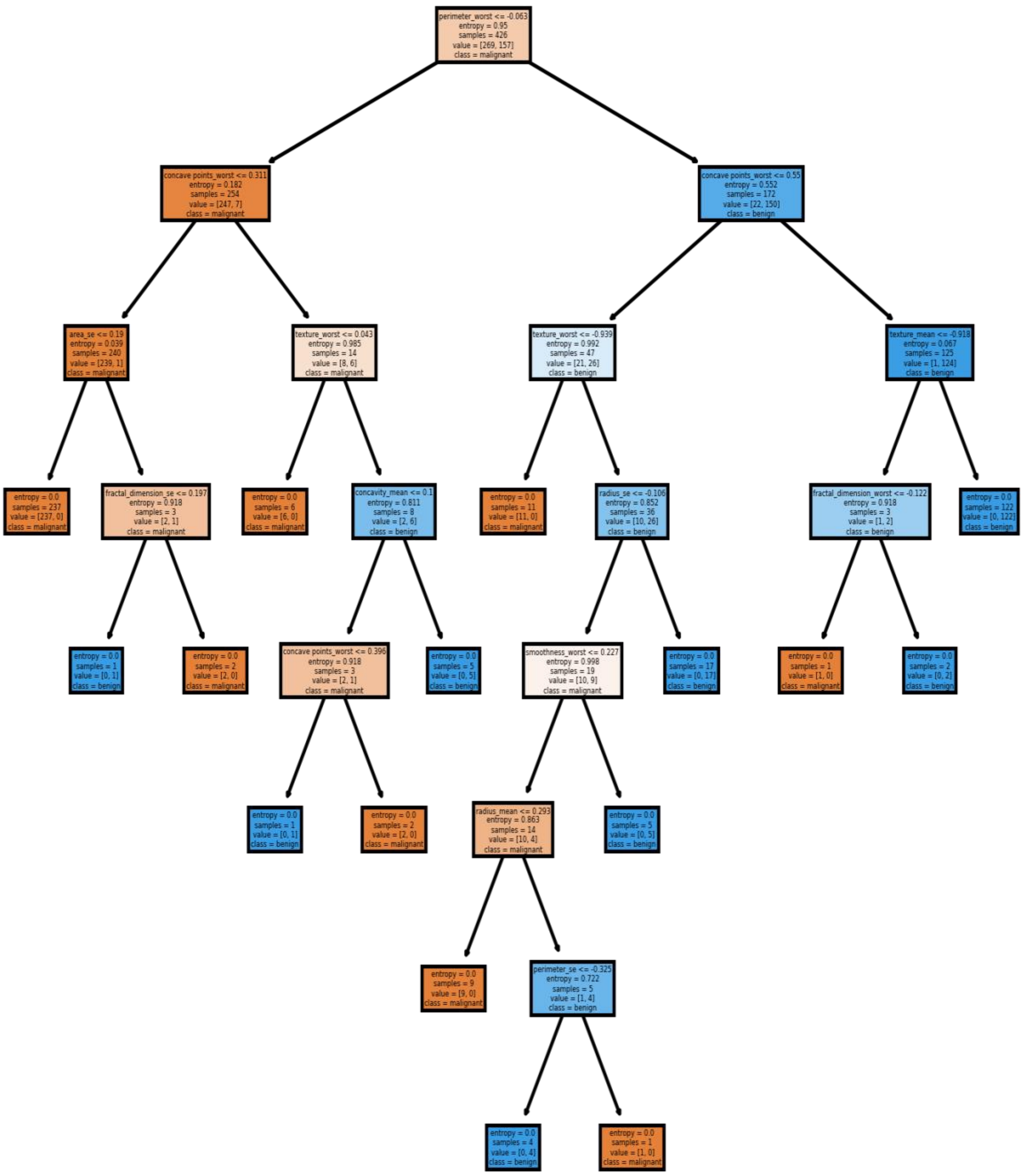
```
dec_tree1 = DecisionTreeClassifier(criterion = 'entropy')
dec_tree1.fit(X_train_c, Y_train_c)
prediction1 = dec_tree1.predict(X_test_c)
```

```
prediction1
```

```
array([0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0])
```

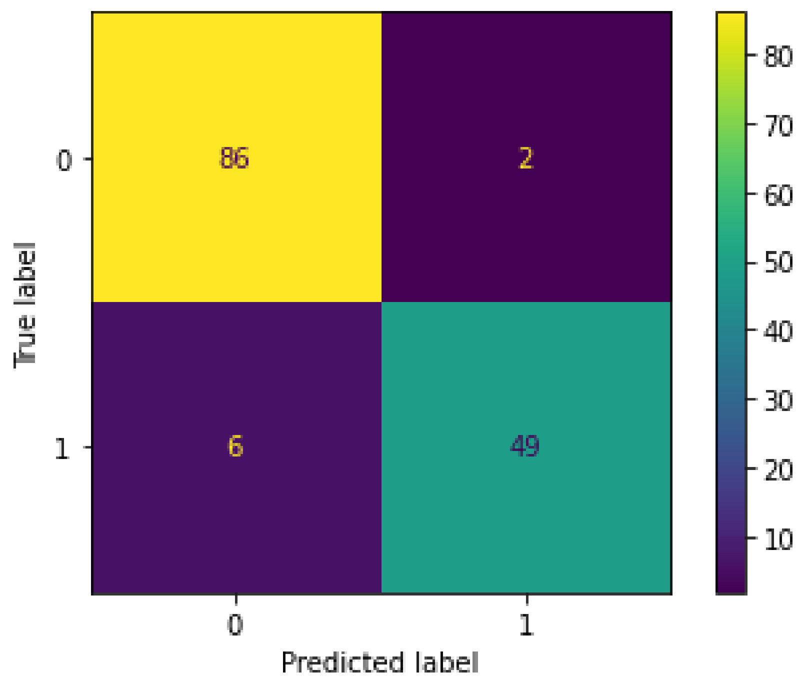
```
tree.plot_tree(dec_tree1)
```

```python
fn = list(df1.columns[:-1])
cn = ['malignant', 'benign']
fig, axes=plt.subplots(nrows=1,ncols=1,figsize=(6,6),dpi=300)
tree.plot_tree(dec_tree1,
               feature_names=fn,
               class_names=cn,
               filled=True);
```



perimeter_worst <= -0.063 entropy = 0.95 samples = 426 value = [269, 157] class = malignant
concave points_worst <= 0.311 entropy = 0.182 samples = 254 value = [247, 7] class = malignant
concave points_worst <= 0.55 entropy = 0.552 samples = 172 value = [22, 150] class = benign
area_se <= 0.19 entropy = 0.039 samples = 240 value = [239, 1] class = malignant
texture_worst <= 0.043 entropy = 0.985 samples = 14 value = [8, 6] class = malignant
texture_worst <= -0.939 entropy = 0.992 samples = 47 value = [21, 26] class = benign
texture_mean <= -0.918 entropy = 0.067 samples = 125 value = [1, 124] class = benign
entropy = 0.0 samples = 237 value = [237, 0] class = malignant
fractal_dimension_se <= 0.197 entropy = 0.918 samples = 3 value = [2, 1] class = malignant
entropy = 0.0 samples = 6 value = [6, 0] class = malignant
concavity_mean <= 0.1 entropy = 0.811 samples = 8 value = [2, 6] class = benign
entropy = 0.0 samples = 11 value = [11, 0] class = malignant
radius_se <= -0.106 entropy = 0.852 samples = 36 value = [10, 26] class = benign
fractal_dimension_worst <= -0.122 entropy = 0.918 samples = 3 value = [1, 2] class = benign
entropy = 0.0 samples = 122 value = [0, 122] class = benign
entropy = 0.0 samples = 1 value = [0, 1] class = benign
entropy = 0.0 samples = 2 value = [2, 0] class = malignant
concave points_worst <= 0.396 entropy = 0.918 samples = 3 value = [2, 1] class = malignant
entropy = 0.0 samples = 5 value = [0, 5] class = benign
smoothness_worst <= 0.227 entropy = 0.998 samples = 19 value = [10, 9] class = malignant
entropy = 0.0 samples = 17 value = [0, 17] class = benign
entropy = 0.0 samples = 1 value = [1, 0] class = malignant
entropy = 0.0 samples = 2 value = [0, 2] class = benign
entropy = 0.0 samples = 1 value = [0, 1] class = benign
entropy = 0.0 samples = 2 value = [2, 0] class = malignant
radius_mean <= 0.293 entropy = 0.863 samples = 14 value = [10, 4] class = malignant
entropy = 0.0 samples = 5 value = [0, 5] class = benign
entropy = 0.0 samples = 9 value = [9, 0] class = malignant
perimeter_se <= -0.325 entropy = 0.722 samples = 5 value = [1, 4] class = benign
entropy = 0.0 samples = 4 value = [0, 4] class = benign
entropy = 0.0 samples = 1 value = [1, 0] class = malignant

```
ConfusionMatrixDisplay.from_estimator(dec_tree1, X_test_c, Y_test_c)
```



```
acc_hold_breast1 = metrics.accuracy_score(Y_test_c, prediction1)*100

print("Accuracy on the test data ", acc_hold_breast1)
print("\nClassification report on the test data ", metrics.classification_report(Y_test_c, prediction1))
```

```
Accuracy on the test data  94.4055944055944

Classification report on the test data                precision    recall  f1-score   support

           0       0.93      0.98      0.96        88
           1       0.96      0.89      0.92        55

    accuracy                           0.94       143
   macro avg       0.95      0.93      0.94       143
weighted avg       0.94      0.94      0.94       143
```

KNN (On Iris Dataset)

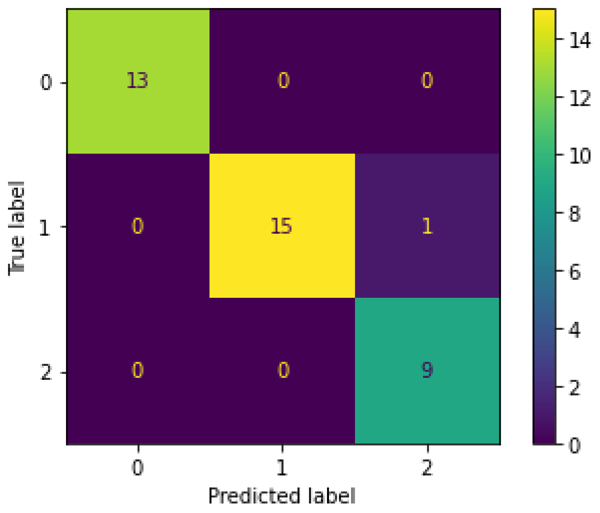
```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
prediction2 = knn.predict(X_test)
```

```
prediction2
```

```
array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0])
```

```
ConfusionMatrixDisplay.from_estimator(knn, X_test, Y_test)
```



```
acc_hold_iris_knn = metrics.accuracy_score(Y_test, prediction2)*100

print("Accuracy on the test data ", acc_hold_iris_knn)
print("\nClassification report on the test data ", metrics.classification_report(Y_test, prediction2))
```

```
Accuracy on the test data  97.36842105263158

Classification report on the test data               precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        16
           2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

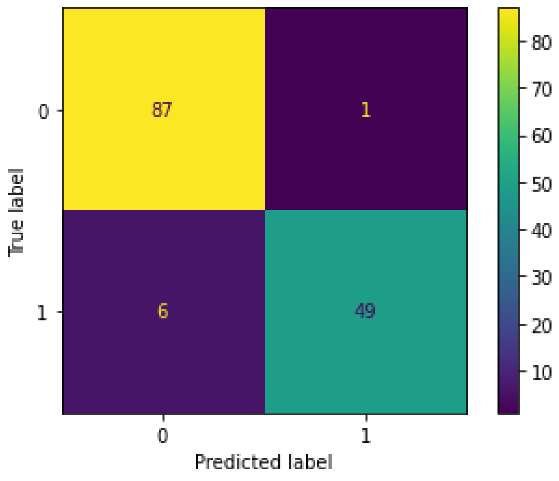
KNN (On Breast Cancer Dataset)

```
knn1 = KNeighborsClassifier()
knn1.fit(X_train_c, Y_train_c)
prediction_c = knn1.predict(X_test_c)
```

```
prediction_c
```

```
array([0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0])
```

```
ConfusionMatrixDisplay.from_estimator(knn1, X_test_c, Y_test_c)
```



```
acc_hold_breast_knn = metrics.accuracy_score(Y_test_c, prediction_c)*100

print("Accuracy on the test data ", acc_hold_breast_knn)
print("\nClassification report on the test data ", metrics.classification_report(Y_test_c, prediction_c))
```

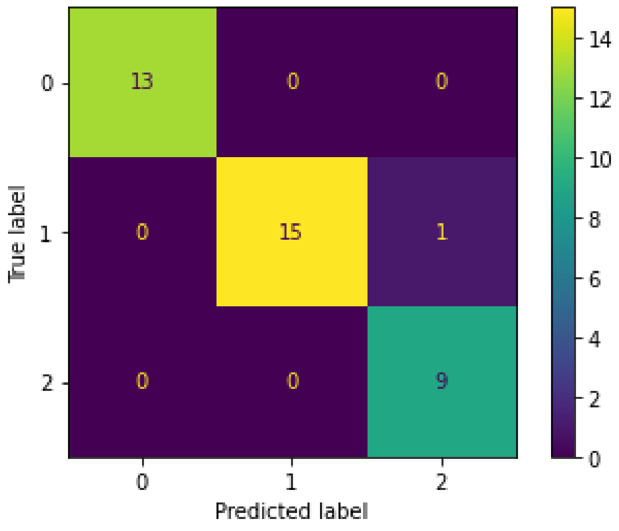
```
Accuracy on the test data  95.1048951048951

Classification report on the test data                precision    recall  f1-score   support

           0       0.94      0.99      0.96        88
           1       0.98      0.89      0.93        55

    accuracy                           0.95       143
   macro avg       0.96      0.94      0.95       143
weighted avg       0.95      0.95      0.95       143
```

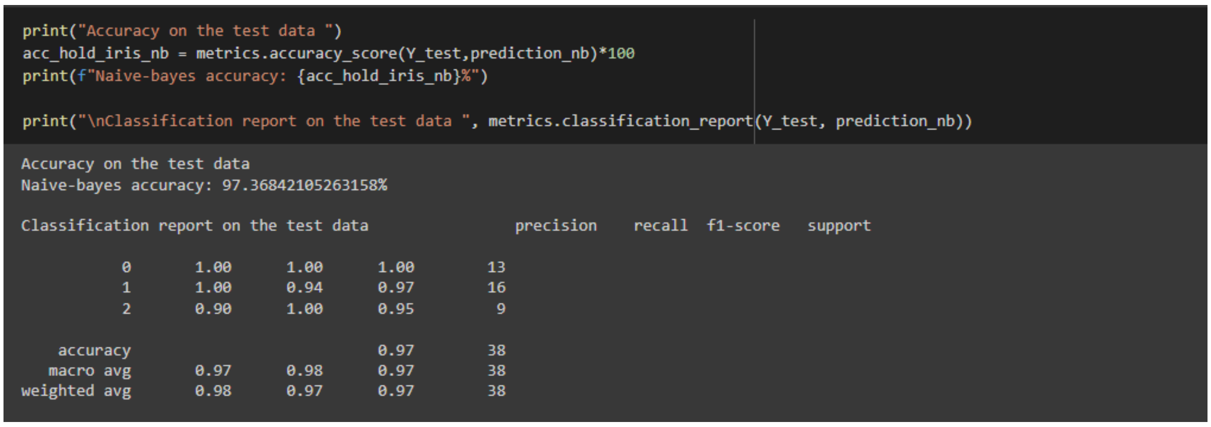
Naïve Bayes (On Iris Dataset)

```
from sklearn.naive_bayes import GaussianNB
```

```
nb = GaussianNB()
nb.fit(X_train,Y_train)
prediction_nb=nb.predict(X_test)
```

```
ConfusionMatrixDisplay.from_estimator(nb, X_test, Y_test)
```

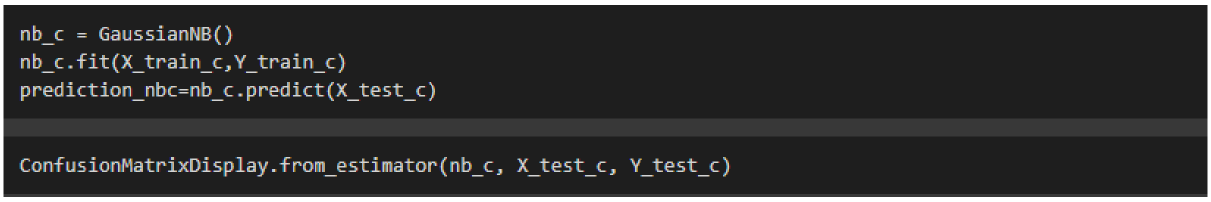




```
print("Accuracy on the test data ")
acc_hold_iris_nb = metrics.accuracy_score(Y_test,prediction_nb)*100
print(f"Naive-bayes accuracy: {acc_hold_iris_nb}%")

print("\nClassification report on the test data ", metrics.classification_report(Y_test, prediction_nb))
```

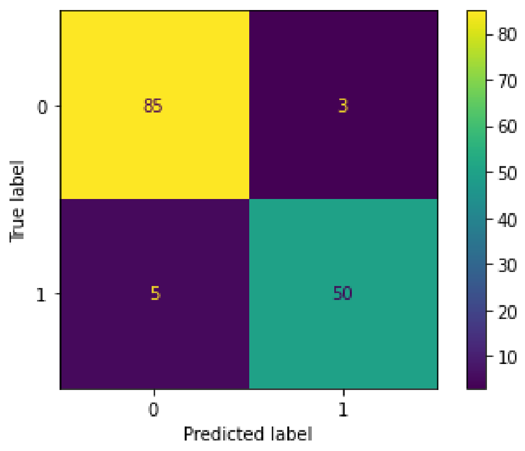
```
Accuracy on the test data
Naive-bayes accuracy: 97.36842105263158%

Classification report on the test data                precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        16
           2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

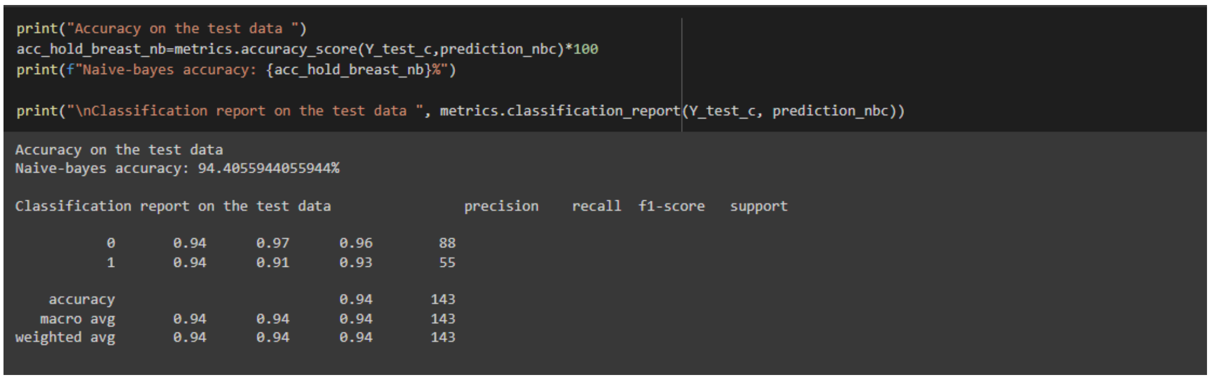
Naïve Bayes (On Breast Cancer Dataset)



```
nb_c = GaussianNB()
nb_c.fit(X_train_c,Y_train_c)
prediction_nbc=nb_c.predict(X_test_c)
```

```
ConfusionMatrixDisplay.from_estimator(nb_c, X_test_c, Y_test_c)
```



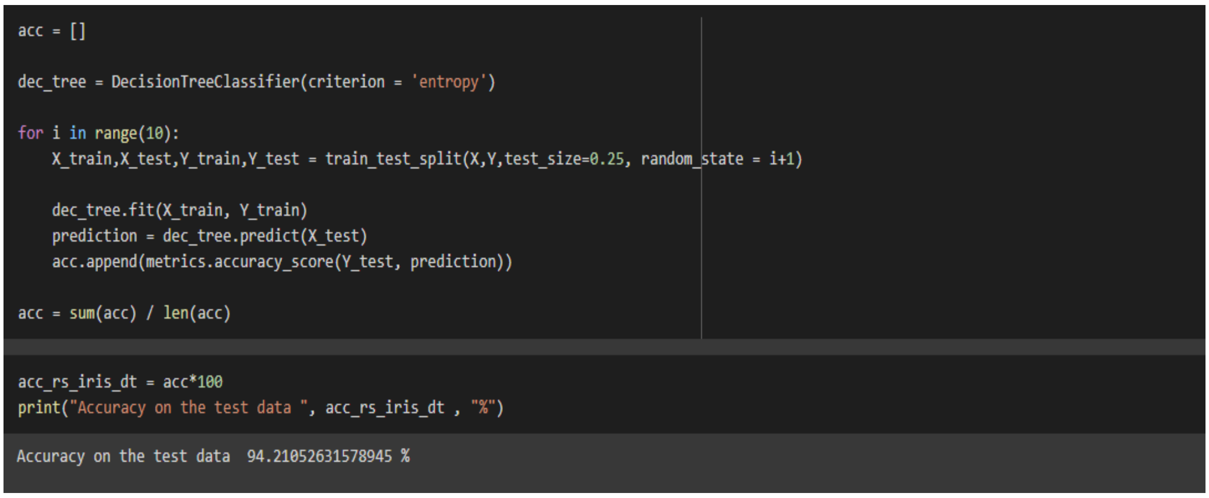


```
print("Accuracy on the test data ")
acc_hold_breast_nb=metrics.accuracy_score(Y_test_c,prediction_nbc)*100
print(f"Naive-bayes accuracy: {acc_hold_breast_nb}%")

print("\nClassification report on the test data ", metrics.classification_report(Y_test_c, prediction_nbc))
```

```
Accuracy on the test data
Naive-bayes accuracy: 94.4055944055944%

Classification report on the test data                precision    recall  f1-score   support

           0       0.94      0.97      0.96        88
           1       0.94      0.91      0.93        55

    accuracy                           0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143
```

Random Subsampling

Decision Tree (On Iris Dataset)

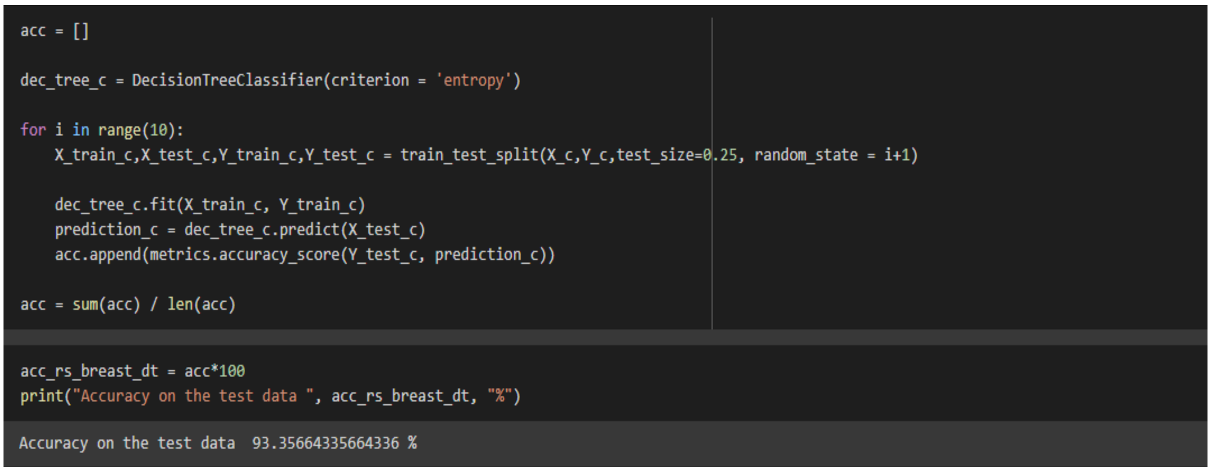


```
acc = []

dec_tree = DecisionTreeClassifier(criterion = 'entropy')

for i in range(10):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25, random_state = i+1)

    dec_tree.fit(X_train, Y_train)
    prediction = dec_tree.predict(X_test)
    acc.append(metrics.accuracy_score(Y_test, prediction))

acc = sum(acc) / len(acc)
```

```
acc_rs_iris_dt = acc*100
print("Accuracy on the test data ", acc_rs_iris_dt , "%")
```

```
Accuracy on the test data  94.21052631578945 %
```

Decision Tree (Breast Cancer Dataset)

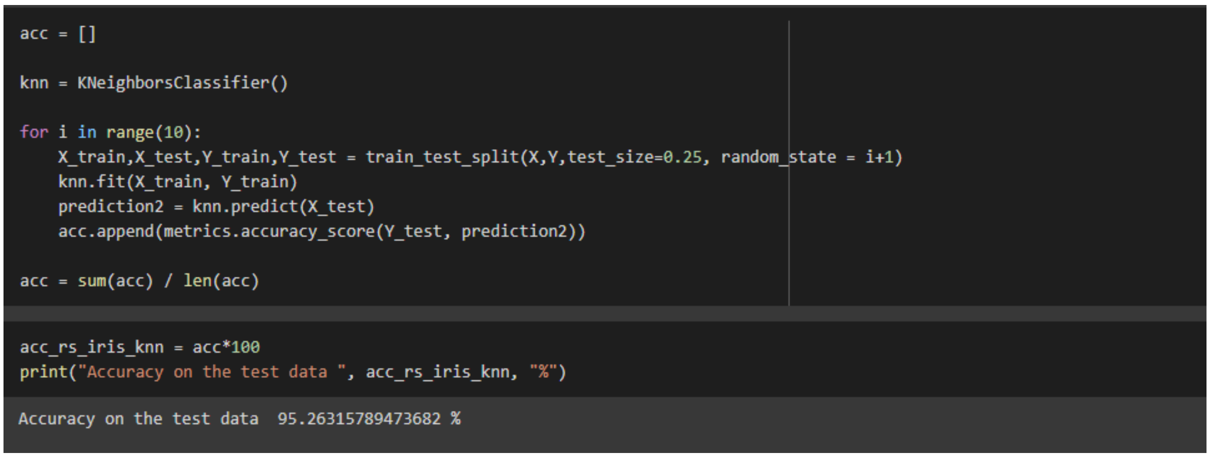


```
acc = []

dec_tree_c = DecisionTreeClassifier(criterion = 'entropy')

for i in range(10):
    X_train_c,X_test_c,Y_train_c,Y_test_c = train_test_split(X_c,Y_c,test_size=0.25, random_state = i+1)

    dec_tree_c.fit(X_train_c, Y_train_c)
    prediction_c = dec_tree_c.predict(X_test_c)
    acc.append(metrics.accuracy_score(Y_test_c, prediction_c))

acc = sum(acc) / len(acc)
```

```
acc_rs_breast_dt = acc*100
print("Accuracy on the test data ", acc_rs_breast_dt, "%")
```

```
Accuracy on the test data  93.35664335664336 %
```

KNN (Iris Dataset)

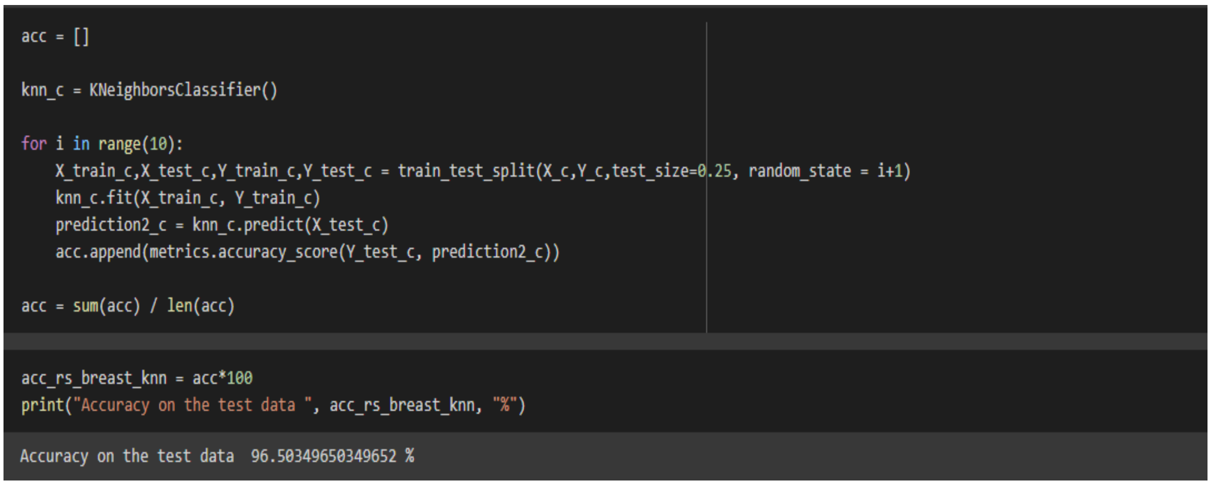


```
acc = []

knn = KNeighborsClassifier()

for i in range(10):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25, random_state = i+1)
    knn.fit(X_train, Y_train)
    prediction2 = knn.predict(X_test)
    acc.append(metrics.accuracy_score(Y_test, prediction2))

acc = sum(acc) / len(acc)
```

```
acc_rs_iris_knn = acc*100
print("Accuracy on the test data ", acc_rs_iris_knn, "%")
```

```
Accuracy on the test data  95.26315789473682 %
```

KNN (Breast Cancer Dataset)

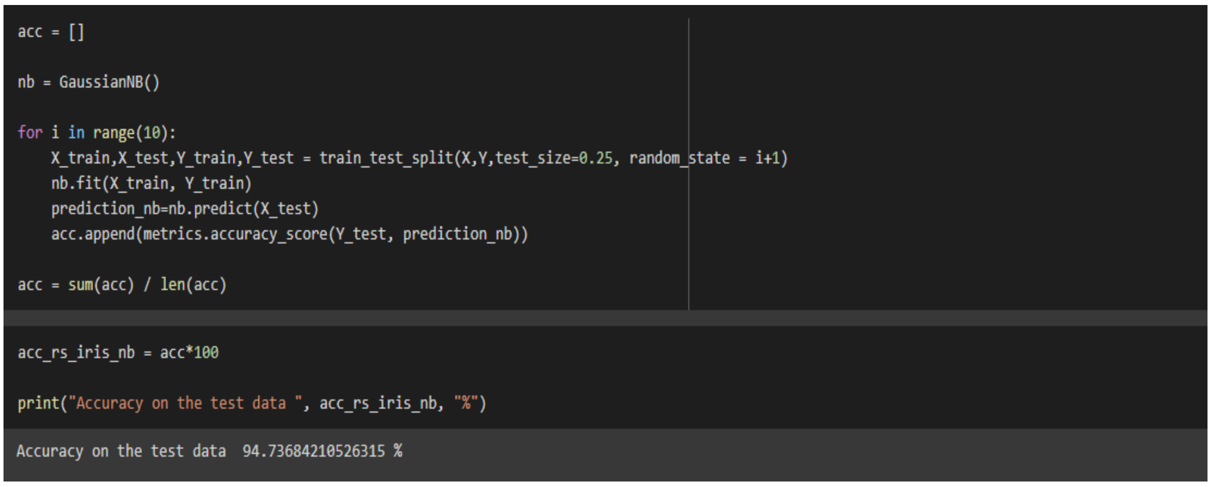


```
acc = []

knn_c = KNeighborsClassifier()

for i in range(10):
    X_train_c,X_test_c,Y_train_c,Y_test_c = train_test_split(X_c,Y_c,test_size=0.25, random_state = i+1)
    knn_c.fit(X_train_c, Y_train_c)
    prediction2_c = knn_c.predict(X_test_c)
    acc.append(metrics.accuracy_score(Y_test_c, prediction2_c))

acc = sum(acc) / len(acc)
```

```
acc_rs_breast_knn = acc*100
print("Accuracy on the test data ", acc_rs_breast_knn, "%")
```

```
Accuracy on the test data  96.50349650349652 %
```

Naïve Bayes (On Iris Dataset)

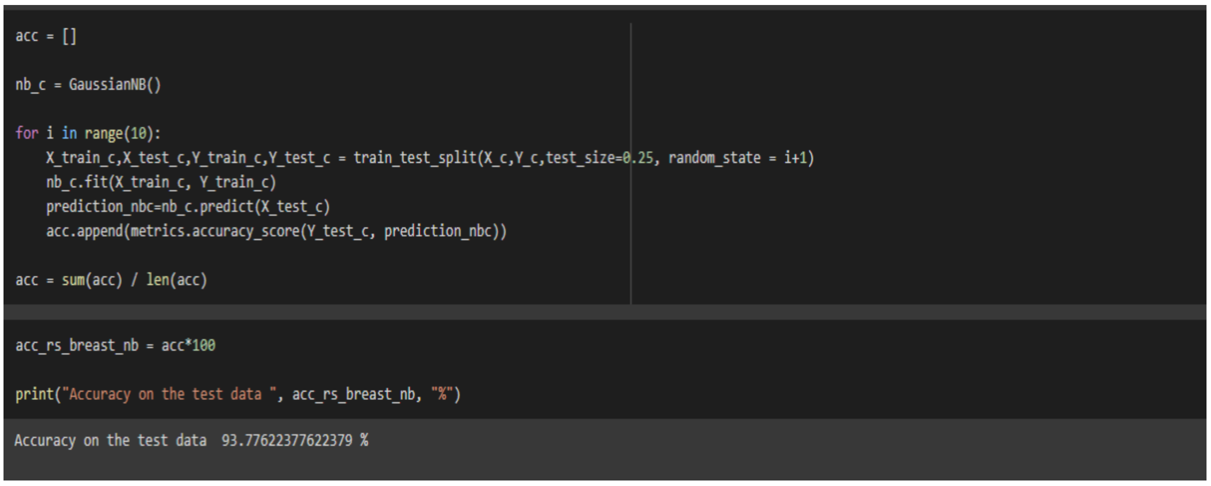


```
acc = []

nb = GaussianNB()

for i in range(10):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25, random_state = i+1)
    nb.fit(X_train, Y_train)
    prediction_nb=nb.predict(X_test)
    acc.append(metrics.accuracy_score(Y_test, prediction_nb))

acc = sum(acc) / len(acc)
```

```
acc_rs_iris_nb = acc*100

print("Accuracy on the test data ", acc_rs_iris_nb, "%")
```

```
Accuracy on the test data  94.73684210526315 %
```

Naïve Bayes (On Breast Cancer Dataset)



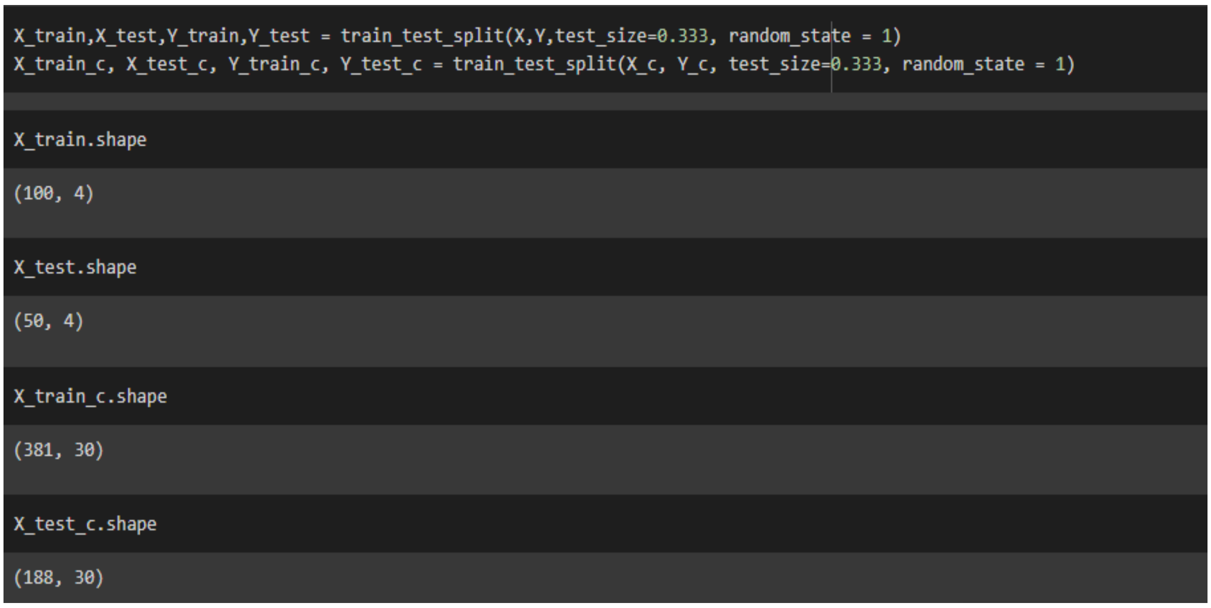
```
acc = []

nb_c = GaussianNB()

for i in range(10):
    X_train_c,X_test_c,Y_train_c,Y_test_c = train_test_split(X_c,Y_c,test_size=0.25, random_state = i+1)
    nb_c.fit(X_train_c, Y_train_c)
    prediction_nbc=nb_c.predict(X_test_c)
    acc.append(metrics.accuracy_score(Y_test_c, prediction_nbc))

acc = sum(acc) / len(acc)
```

```
acc_rs_breast_nb = acc*100

print("Accuracy on the test data ", acc_rs_breast_nb, "%")
```

```
Accuracy on the test data  93.77622377622379 %
```

Training set = 66.6% (2/3rd of total), Test set = 33.3%

Holdout Method



```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.333, random_state = 1)
X_train_c, X_test_c, Y_train_c, Y_test_c = train_test_split(X_c, Y_c, test_size=0.333, random_state = 1)
```

```
X_train.shape
```

```
(100, 4)
```

```
X_test.shape
```

```
(50, 4)
```

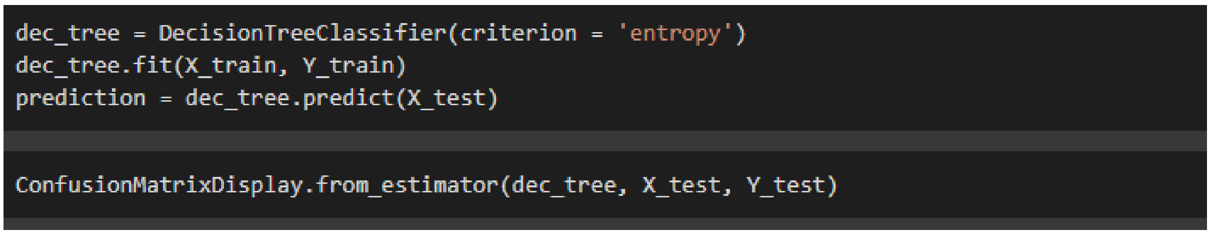
```
X_train_c.shape
```

```
(381, 30)
```

```
X_test_c.shape
```

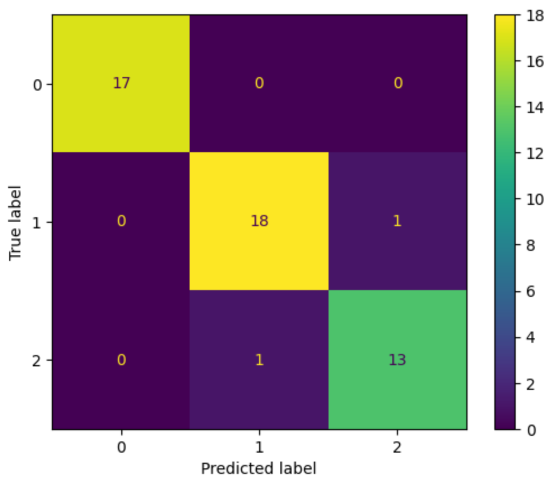
```
(188, 30)
```

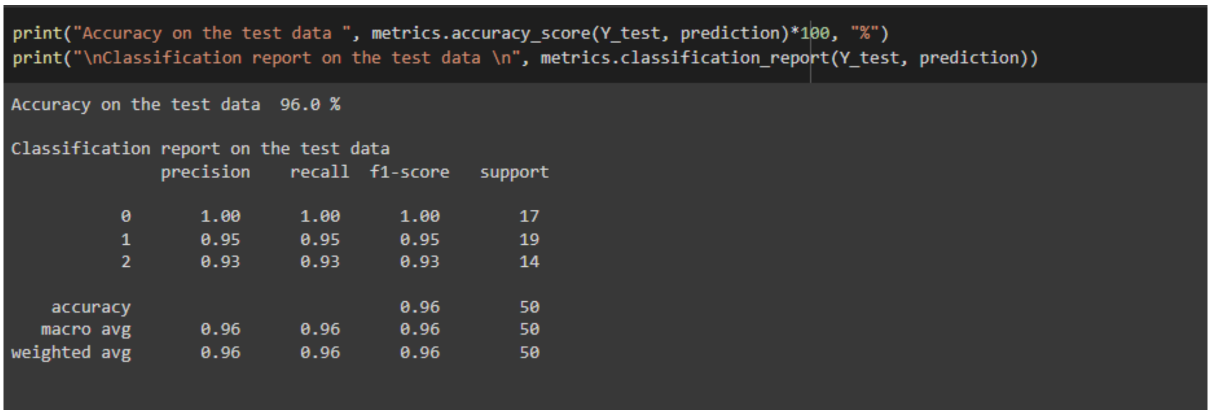
Decision Tree (On Iris Dataset)



```
dec_tree = DecisionTreeClassifier(criterion = 'entropy')
dec_tree.fit(X_train, Y_train)
prediction = dec_tree.predict(X_test)
```

```
ConfusionMatrixDisplay.from_estimator(dec_tree, X_test, Y_test)
```





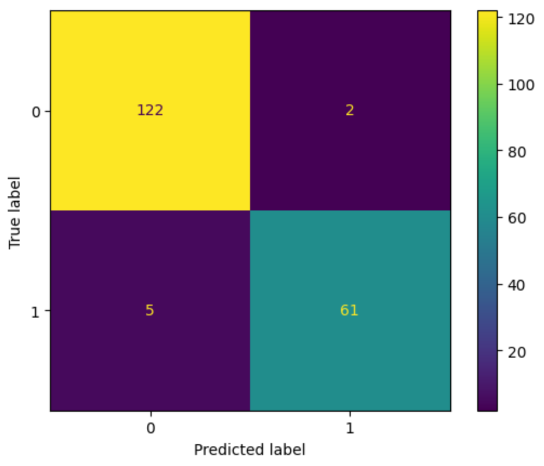
```
print("Accuracy on the test data ", metrics.accuracy_score(Y_test, prediction)*100, "%")
print("\nClassification report on the test data \n", metrics.classification_report(Y_test, prediction))
```

```
Accuracy on the test data  96.0 %

Classification report on the test data 
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.95      0.95      0.95        19
           2       0.93      0.93      0.93        14

    accuracy                           0.96        50
   macro avg       0.96      0.96      0.96        50
weighted avg       0.96      0.96      0.96        50
```

Decision Tree (On Breast Cancer Dataset)

```
dec_tree1_c = DecisionTreeClassifier(criterion = 'entropy')
dec_tree1_c.fit(X_train_c, Y_train_c)
prediction_c = dec_tree1_c.predict(X_test_c)
```

```
ConfusionMatrixDisplay.from_estimator(dec_tree1, X_test_c, Y_test_c)
```



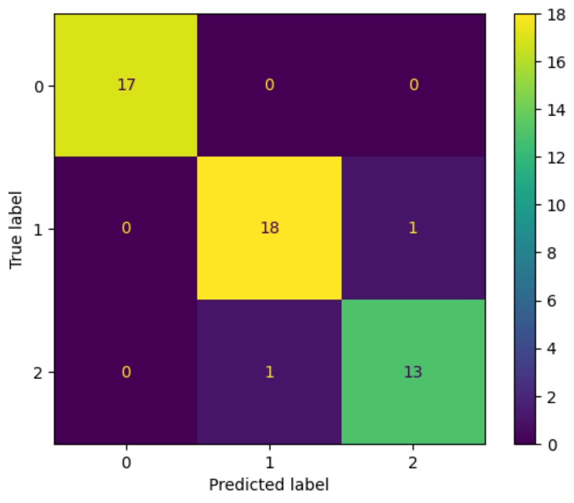
```
print("Accuracy on the test data ", metrics.accuracy_score(Y_test_c, prediction_c)*100, "%")
print("\nClassification report on the test data \n", metrics.classification_report(Y_test_c, prediction_c))
```

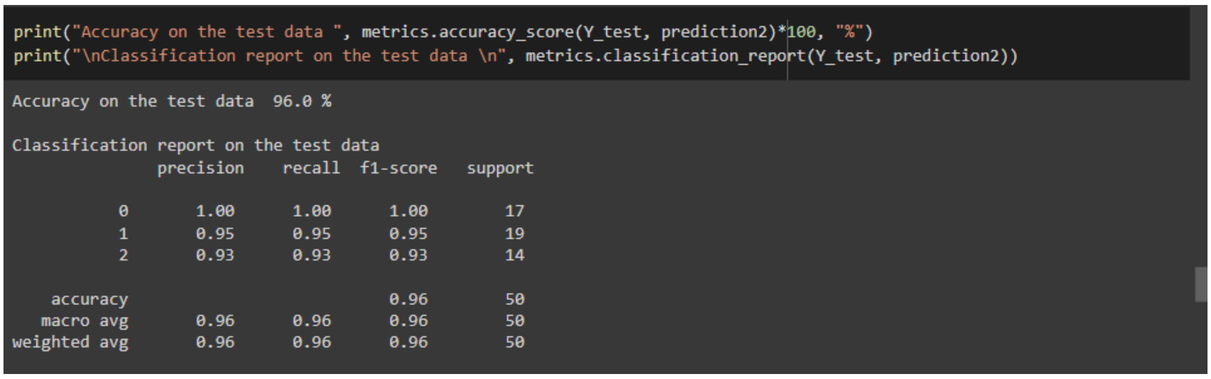
```
Accuracy on the test data  91.05263157894737 %

Classification report on the test data
               precision    recall  f1-score   support

           0       0.95      0.91      0.93       124
           1       0.85      0.91      0.88        66

    accuracy                           0.91       190
   macro avg       0.90      0.91      0.90       190
weighted avg       0.91      0.91      0.91       190
```

KNN (On Iris Dataset)

```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
prediction2 = knn.predict(X_test)
```

```
ConfusionMatrixDisplay.from_estimator(knn, X_test, Y_test)
```

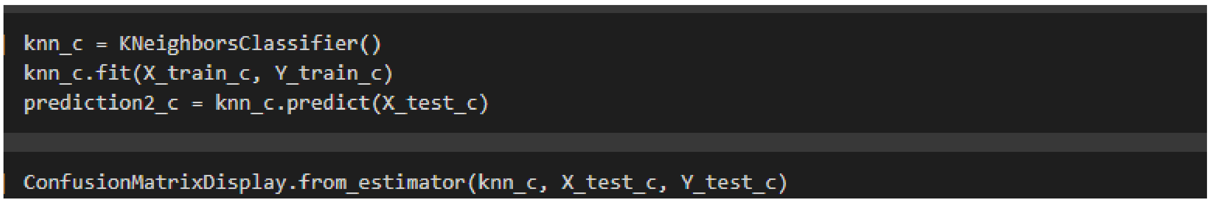




```
print("Accuracy on the test data ", metrics.accuracy_score(Y_test, prediction2)*100, "%")
print("\nClassification report on the test data \n", metrics.classification_report(Y_test, prediction2))
```

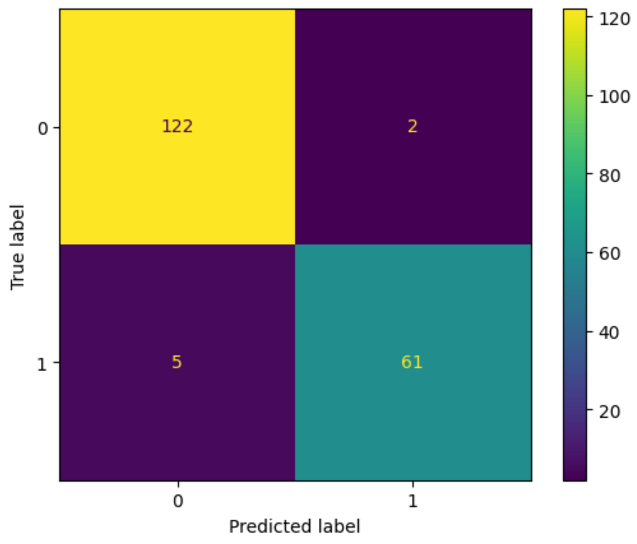
```
Accuracy on the test data  96.0 %

Classification report on the test data
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.95      0.95      0.95        19
           2       0.93      0.93      0.93        14

    accuracy                           0.96        50
   macro avg       0.96      0.96      0.96        50
weighted avg       0.96      0.96      0.96        50
```

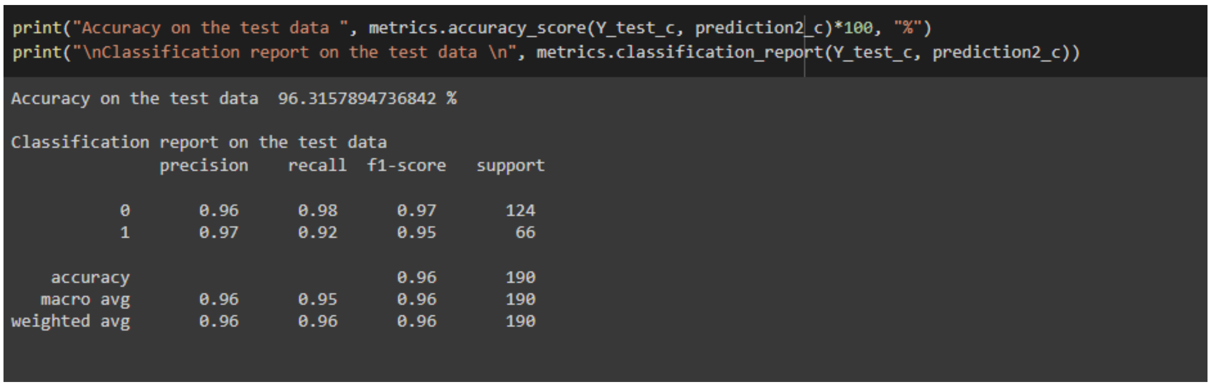
KNN (On Breast Cancer Dataset)



```
knn_c = KNeighborsClassifier()
knn_c.fit(X_train_c, Y_train_c)
prediction2_c = knn_c.predict(X_test_c)
```

```
ConfusionMatrixDisplay.from_estimator(knn_c, X_test_c, Y_test_c)
```

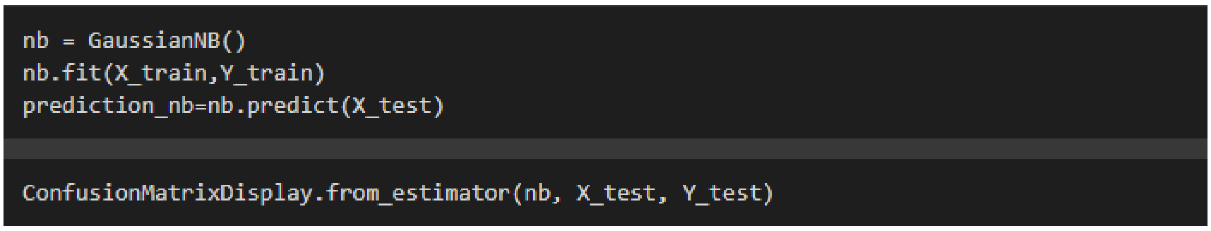




```
print("Accuracy on the test data ", metrics.accuracy_score(Y_test_c, prediction2_c)*100, "%")
print("\nClassification report on the test data \n", metrics.classification_report(Y_test_c, prediction2_c))
```

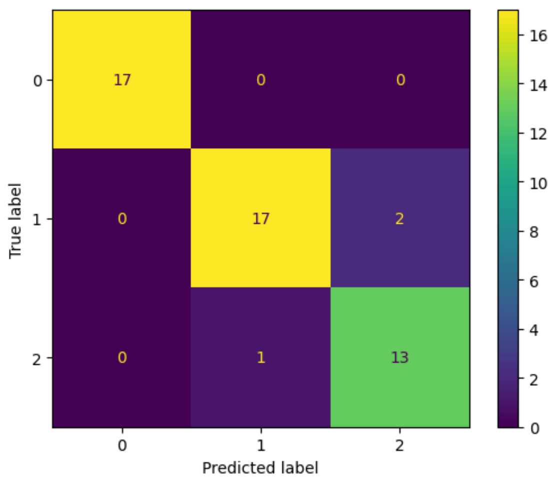
```
Accuracy on the test data  96.3157894736842 %

Classification report on the test data
               precision    recall  f1-score   support

           0       0.96      0.98      0.97       124
           1       0.97      0.92      0.95        66

    accuracy                           0.96       190
   macro avg       0.96      0.95      0.96       190
weighted avg       0.96      0.96      0.96       190
```

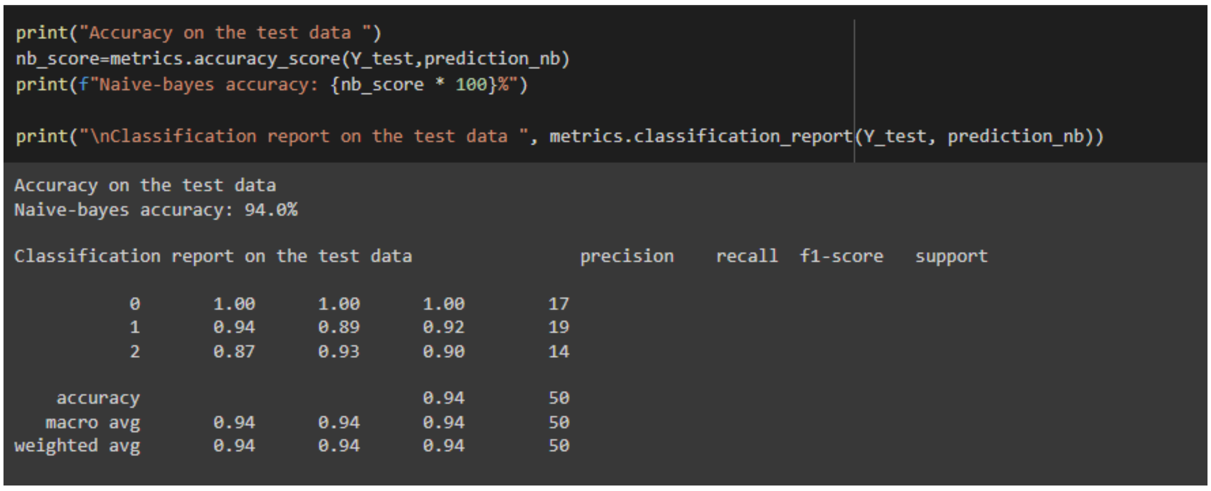
Naïve Bayes (On Iris Dataset)



```
nb = GaussianNB()
nb.fit(X_train,Y_train)
prediction_nb=nb.predict(X_test)
```

```
ConfusionMatrixDisplay.from_estimator(nb, X_test, Y_test)
```





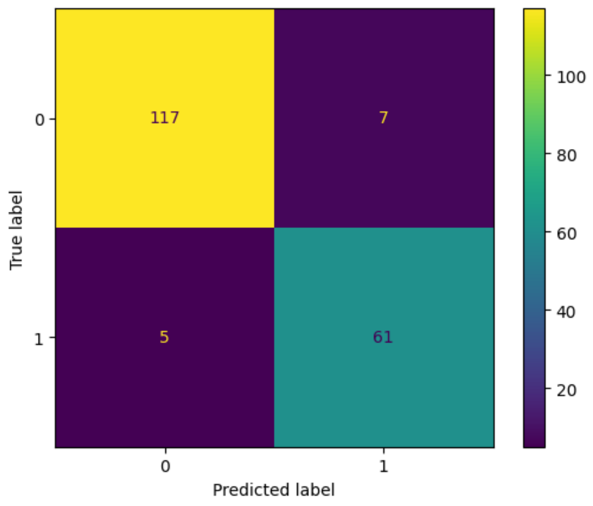
```
print("Accuracy on the test data ")
nb_score=metrics.accuracy_score(Y_test,prediction_nb)
print(f"Naive-bayes accuracy: {nb_score * 100}%")

print("\nClassification report on the test data ", metrics.classification_report(Y_test, prediction_nb))
```

```
Accuracy on the test data
Naive-bayes accuracy: 94.0%

Classification report on the test data                precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.94      0.89      0.92        19
           2       0.87      0.93      0.90        14

    accuracy                           0.94        50
   macro avg       0.94      0.94      0.94        50
weighted avg       0.94      0.94      0.94        50
```

Naïve Bayes (On Breast Cancer Dataset)

```
nb_c = GaussianNB()
nb_c.fit(X_train_c,Y_train_c)
prediction_nbc=nb_c.predict(X_test_c)
```

```
ConfusionMatrixDisplay.from_estimator(nb_c, X_test_c, Y_test_c)
```



```
print("Accuracy on the test data ")
nb_score=metrics.accuracy_score(Y_test_c,prediction_nbc)
print(f"Naive-bayes accuracy: {nb_score * 100}%")

print("\nClassification report on the test data ", metrics.classification_report(Y_test_c, prediction_nbc))
```

```
Accuracy on the test data
Naive-bayes accuracy: 93.6842105263158%

Classification report on the test data                precision    recall  f1-score   support

           0       0.96      0.94      0.95       124
           1       0.90      0.92      0.91        66

    accuracy                           0.94       190
   macro avg       0.93      0.93      0.93       190
weighted avg       0.94      0.94      0.94       190
```

Random Subsampling

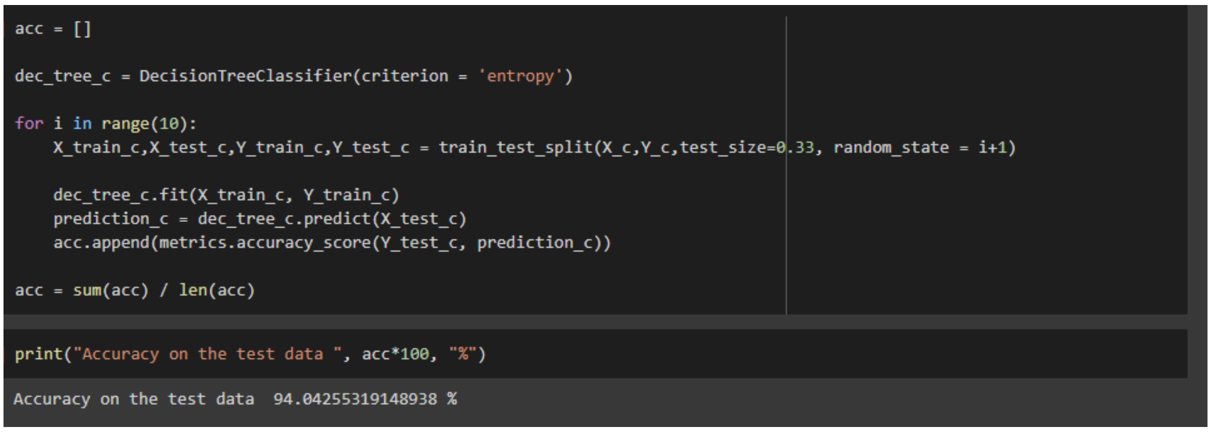
Decision Tree (On Iris dataset)

```
acc = []

dec_tree = DecisionTreeClassifier(criterion = 'entropy')

for i in range(10):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.33, random_state = i+1)

    dec_tree.fit(X_train, Y_train)
    prediction = dec_tree.predict(X_test)
    acc.append(metrics.accuracy_score(Y_test, prediction))

acc = sum(acc) / len(acc)
```

```
print("Accuracy on the test data ", acc*100, "%")
```

```
Accuracy on the test data  94.60000000000001 %
```

Decision Tree (On Breast Cancer dataset)

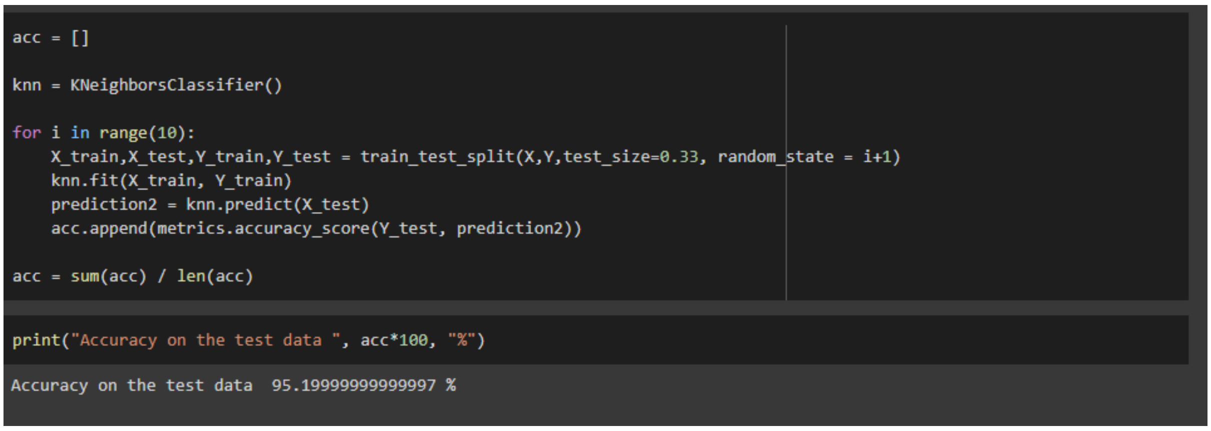


```
acc = []

dec_tree_c = DecisionTreeClassifier(criterion = 'entropy')

for i in range(10):
    X_train_c,X_test_c,Y_train_c,Y_test_c = train_test_split(X_c,Y_c,test_size=0.33, random_state = i+1)

    dec_tree_c.fit(X_train_c, Y_train_c)
    prediction_c = dec_tree_c.predict(X_test_c)
    acc.append(metrics.accuracy_score(Y_test_c, prediction_c))

acc = sum(acc) / len(acc)
```

```
print("Accuracy on the test data ", acc*100, "%")
```

```
Accuracy on the test data  94.04255319148938 %
```

KNN (On Iris Dataset)

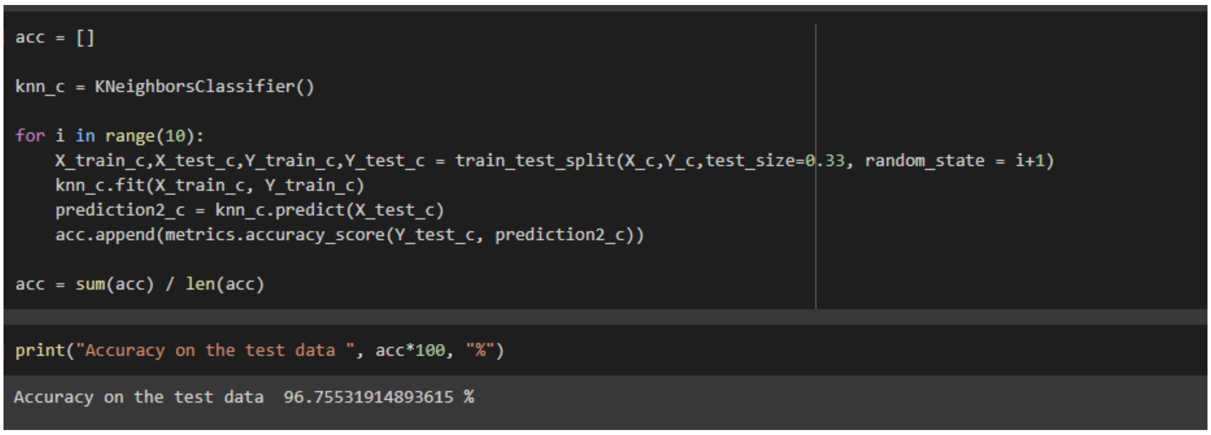


```
acc = []

knn = KNeighborsClassifier()

for i in range(10):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.33, random_state = i+1)
    knn.fit(X_train, Y_train)
    prediction2 = knn.predict(X_test)
    acc.append(metrics.accuracy_score(Y_test, prediction2))

acc = sum(acc) / len(acc)
```

```
print("Accuracy on the test data ", acc*100, "%")
```

```
Accuracy on the test data  95.19999999999997 %
```

KNN (On Breast Cancer Dataset)

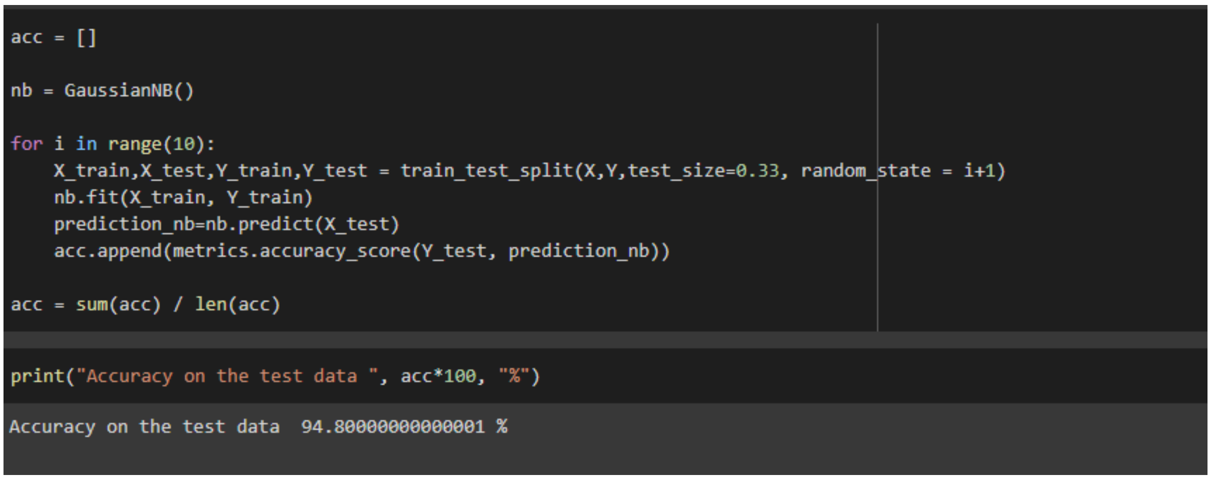


```
acc = []

knn_c = KNeighborsClassifier()

for i in range(10):
    X_train_c,X_test_c,Y_train_c,Y_test_c = train_test_split(X_c,Y_c,test_size=0.33, random_state = i+1)
    knn_c.fit(X_train_c, Y_train_c)
    prediction2_c = knn_c.predict(X_test_c)
    acc.append(metrics.accuracy_score(Y_test_c, prediction2_c))

acc = sum(acc) / len(acc)
```

```
print("Accuracy on the test data ", acc*100, "%")
```

```
Accuracy on the test data  96.75531914893615 %
```

Naïve Bayes (On Iris Dataset)

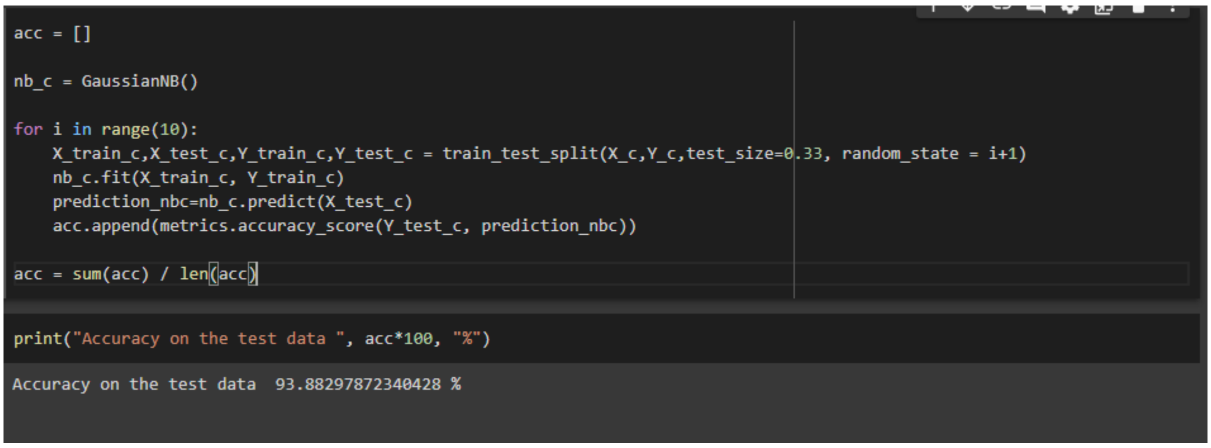


```
acc = []

nb = GaussianNB()

for i in range(10):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.33, random_state = i+1)
    nb.fit(X_train, Y_train)
    prediction_nb=nb.predict(X_test)
    acc.append(metrics.accuracy_score(Y_test, prediction_nb))

acc = sum(acc) / len(acc)
```

```
print("Accuracy on the test data ", acc*100, "%")
```

```
Accuracy on the test data  94.80000000000001 %
```

Naïve Bayes (On Breast Cancer Dataset)



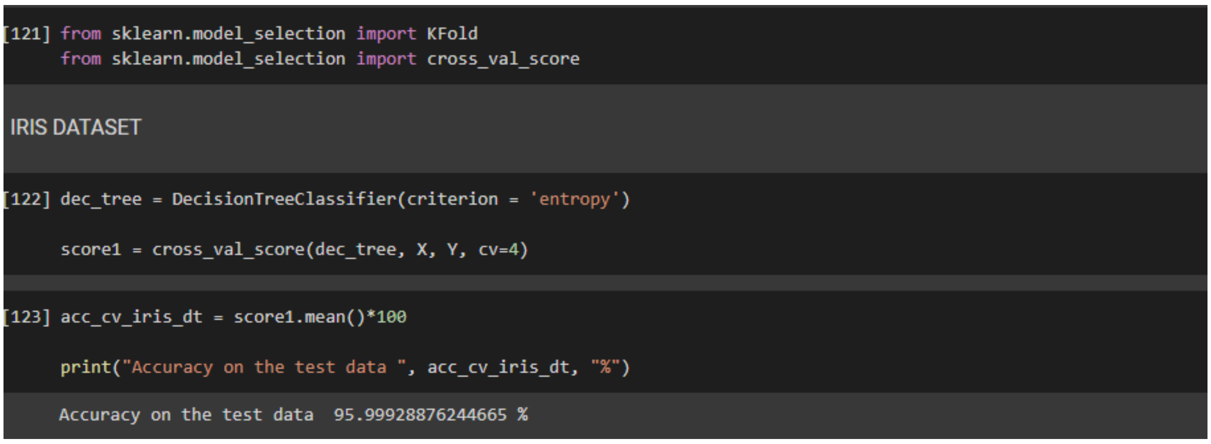
```
acc = []

nb_c = GaussianNB()

for i in range(10):
    X_train_c,X_test_c,Y_train_c,Y_test_c = train_test_split(X_c,Y_c,test_size=0.33, random_state = i+1)
    nb_c.fit(X_train_c, Y_train_c)
    prediction_nbc=nb_c.predict(X_test_c)
    acc.append(metrics.accuracy_score(Y_test_c, prediction_nbc))

acc = sum(acc) / len(acc)
```

```
print("Accuracy on the test data ", acc*100, "%")
```

```
Accuracy on the test data  93.88297872340428 %
```

Cross Validation

Decision Tree (On Iris Dataset)



```
[121] from sklearn.model_selection import KFold
      from sklearn.model_selection import cross_val_score
```

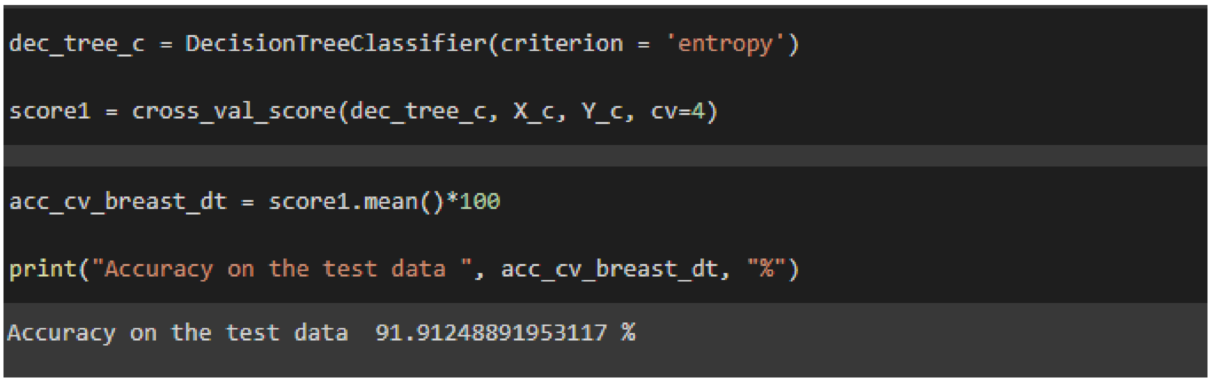
IRIS DATASET

```
[122] dec_tree = DecisionTreeClassifier(criterion = 'entropy')

      score1 = cross_val_score(dec_tree, X, Y, cv=4)
```

```
[123] acc_cv_iris_dt = score1.mean()*100

      print("Accuracy on the test data ", acc_cv_iris_dt, "%")
```

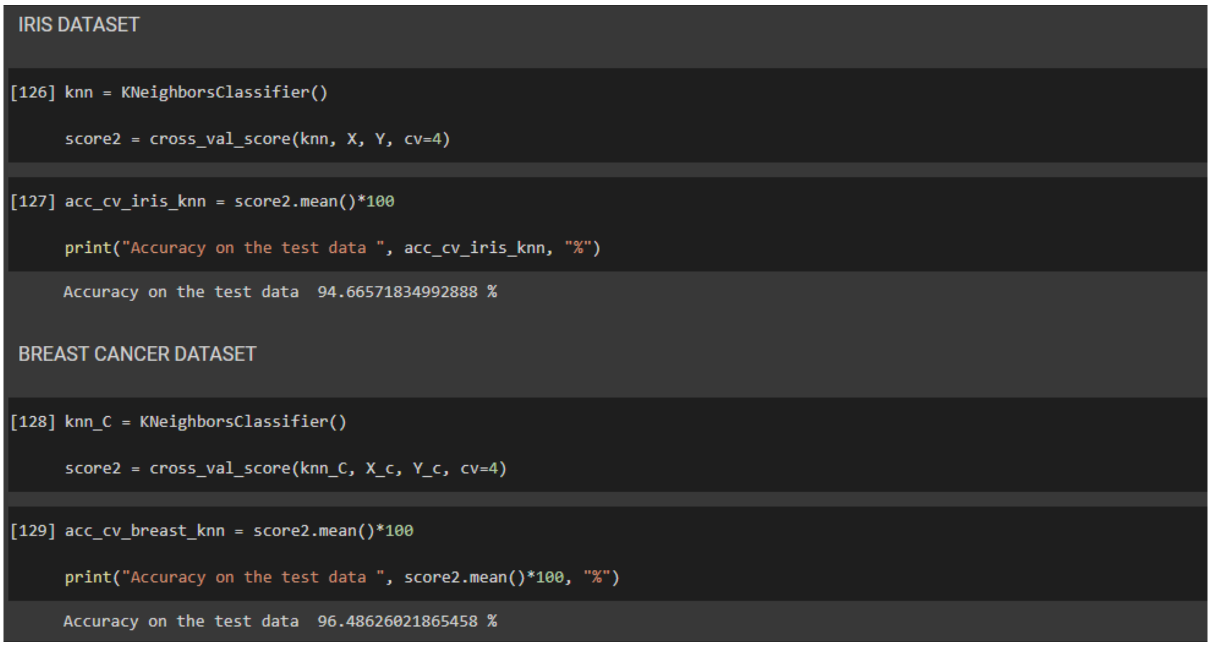
```
      Accuracy on the test data  95.99928876244665 %
```

Decision Tree (On Breast Cancer Dataset)



```
dec_tree_c = DecisionTreeClassifier(criterion = 'entropy')

score1 = cross_val_score(dec_tree_c, X_c, Y_c, cv=4)

acc_cv_breast_dt = score1.mean()*100

print("Accuracy on the test data ", acc_cv_breast_dt, "%")

Accuracy on the test data  91.91248891953117 %
```

KNN (On Iris and Breast Cancer Dataset)



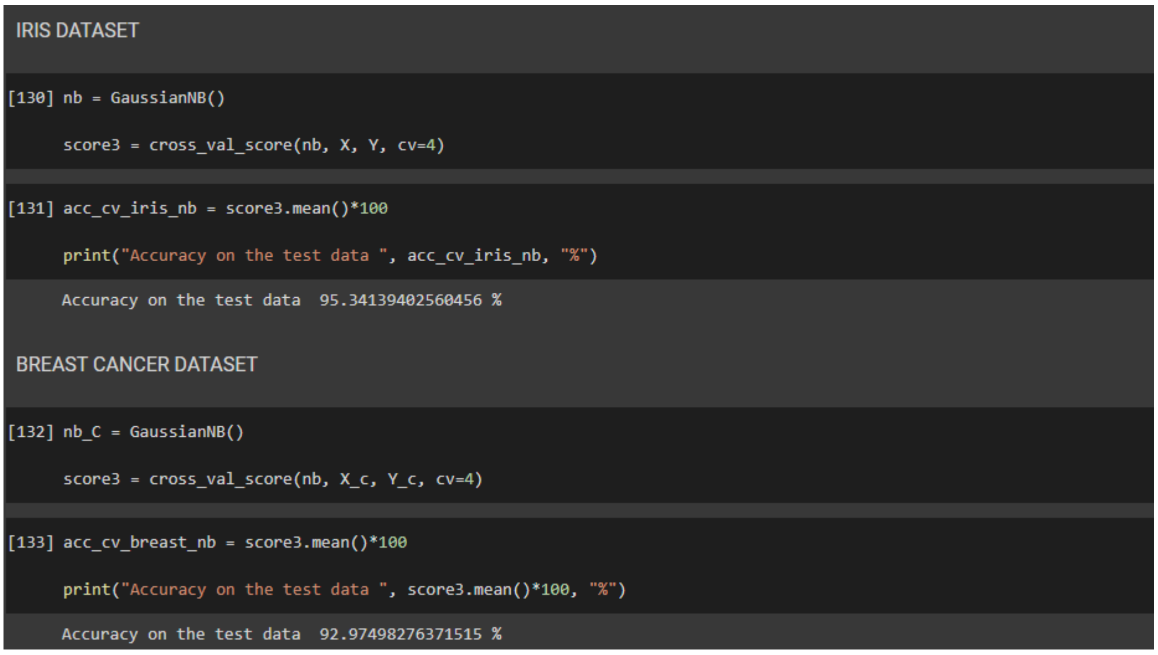
IRIS DATASET

```
[126] knn = KNeighborsClassifier()

      score2 = cross_val_score(knn, X, Y, cv=4)

[127] acc_cv_iris_knn = score2.mean()*100

      print("Accuracy on the test data ", acc_cv_iris_knn, "%")

      Accuracy on the test data  94.66571834992888 %
```

BREAST CANCER DATASET

```
[128] knn_C = KNeighborsClassifier()

      score2 = cross_val_score(knn_C, X_c, Y_c, cv=4)

[129] acc_cv_breast_knn = score2.mean()*100

      print("Accuracy on the test data ", score2.mean()*100, "%")

      Accuracy on the test data  96.48626021865458 %
```

Naïve bayes (On Iris and Breast Cancer Dataset)



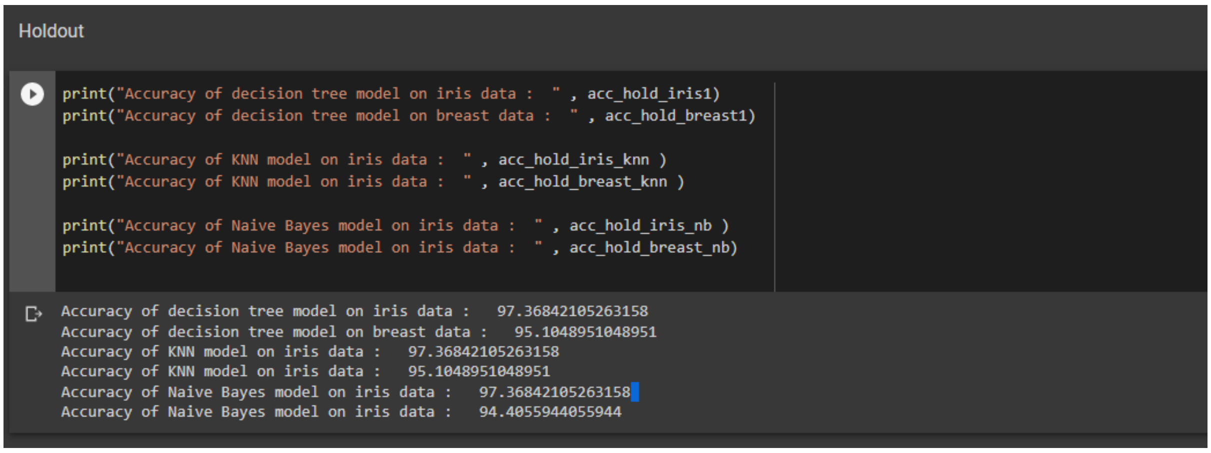
IRIS DATASET

```
[130] nb = GaussianNB()

      score3 = cross_val_score(nb, X, Y, cv=4)

[131] acc_cv_iris_nb = score3.mean()*100

      print("Accuracy on the test data ", acc_cv_iris_nb, "%")

      Accuracy on the test data  95.34139402560456 %
```

BREAST CANCER DATASET

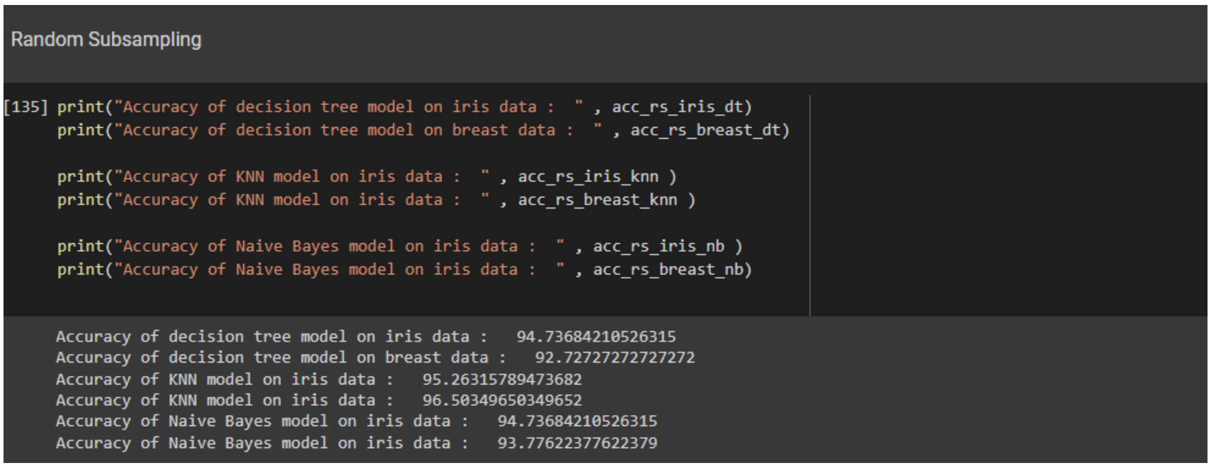
```
[132] nb_C = GaussianNB()

      score3 = cross_val_score(nb, X_c, Y_c, cv=4)

[133] acc_cv_breast_nb = score3.mean()*100

      print("Accuracy on the test data ", score3.mean()*100, "%")

      Accuracy on the test data  92.97498276371515 %
```

Result:


Holdout

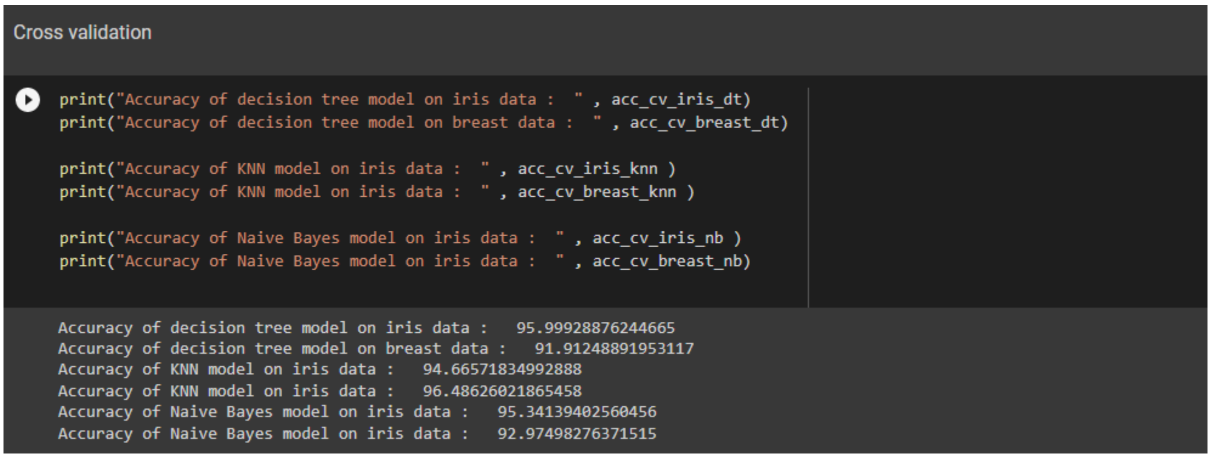
```
print("Accuracy of decision tree model on iris data :  " , acc_hold_iris1)
print("Accuracy of decision tree model on breast data :  " , acc_hold_breast1)

print("Accuracy of KNN model on iris data :  " , acc_hold_iris_knn )
print("Accuracy of KNN model on iris data :  " , acc_hold_breast_knn )

print("Accuracy of Naive Bayes model on iris data :  " , acc_hold_iris_nb )
print("Accuracy of Naive Bayes model on iris data :  " , acc_hold_breast_nb)
```

```
Accuracy of decision tree model on iris data :   97.36842105263158
Accuracy of decision tree model on breast data :   95.1048951048951
Accuracy of KNN model on iris data :   97.36842105263158
Accuracy of KNN model on iris data :   95.1048951048951
Accuracy of Naive Bayes model on iris data :   97.36842105263158
Accuracy of Naive Bayes model on iris data :   94.4055944055944
```


Random Subsampling

```
[135] print("Accuracy of decision tree model on iris data :  " , acc_rs_iris_dt)
      print("Accuracy of decision tree model on breast data :  " , acc_rs_breast_dt)

      print("Accuracy of KNN model on iris data :  " , acc_rs_iris_knn )
      print("Accuracy of KNN model on iris data :  " , acc_rs_breast_knn )

      print("Accuracy of Naive Bayes model on iris data :  " , acc_rs_iris_nb )
      print("Accuracy of Naive Bayes model on iris data :  " , acc_rs_breast_nb)
```

```
Accuracy of decision tree model on iris data :   94.73684210526315
Accuracy of decision tree model on breast data :   92.72727272727272
Accuracy of KNN model on iris data :   95.26315789473682
Accuracy of KNN model on iris data :   96.50349650349652
Accuracy of Naive Bayes model on iris data :   94.73684210526315
Accuracy of Naive Bayes model on iris data :   93.77622377622379
```

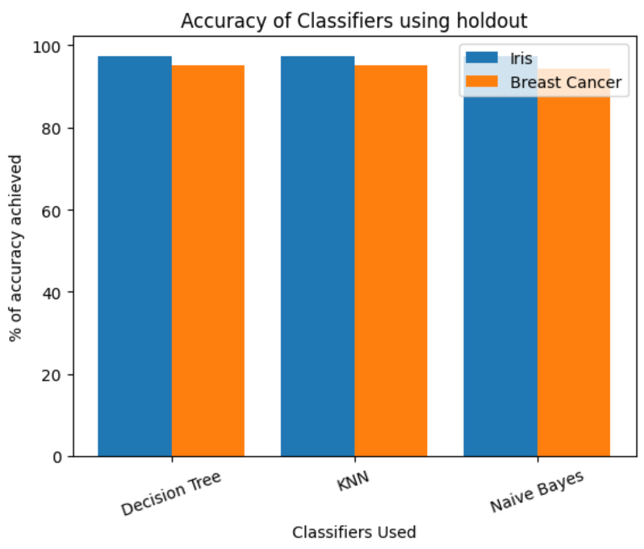

Cross validation

```
print("Accuracy of decision tree model on iris data :  " , acc_cv_iris_dt)
print("Accuracy of decision tree model on breast data :  " , acc_cv_breast_dt)

print("Accuracy of KNN model on iris data :  " , acc_cv_iris_knn )
print("Accuracy of KNN model on iris data :  " , acc_cv_breast_knn )

print("Accuracy of Naive Bayes model on iris data :  " , acc_cv_iris_nb )
print("Accuracy of Naive Bayes model on iris data :  " , acc_cv_breast_nb)
```

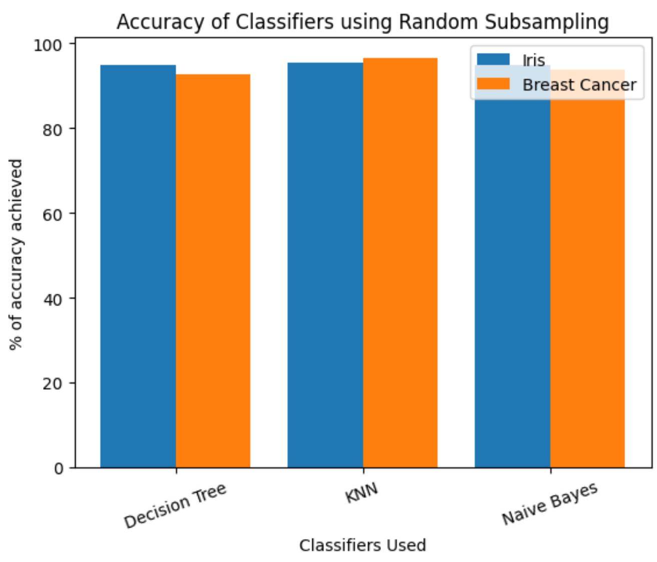
```
Accuracy of decision tree model on iris data :   95.99928876244665
Accuracy of decision tree model on breast data :   91.91248891953117
Accuracy of KNN model on iris data :   94.66571834992888
Accuracy of KNN model on iris data :   96.48626021865458
Accuracy of Naive Bayes model on iris data :   95.34139402560456
Accuracy of Naive Bayes model on iris data :   92.97498276371515
```

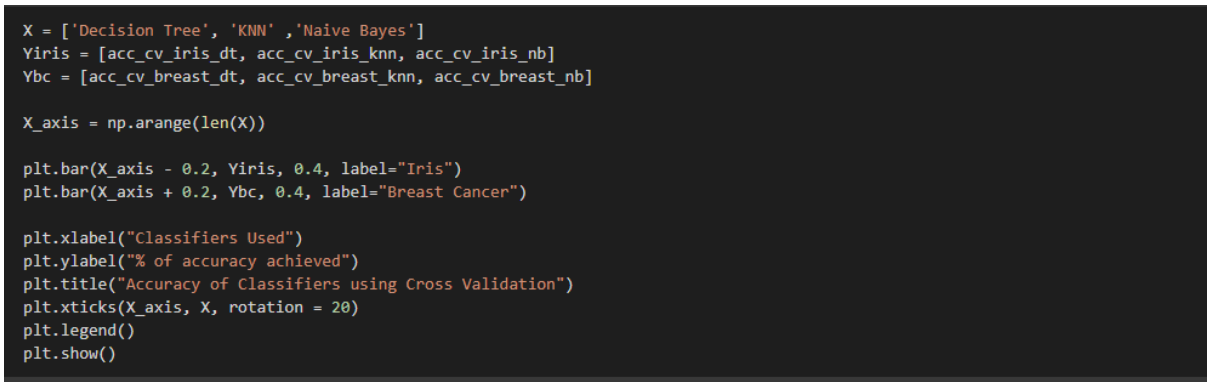
```
import matplotlib.pyplot as plt
```

```
X = ['Decision Tree', 'KNN' ,'Naive Bayes']
Yiris = [acc_hold_iris1, acc_hold_iris_knn, acc_hold_iris_nb]
Ybc = [acc_hold_breast1, acc_hold_breast_knn, acc_hold_breast_nb]

X_axis = np.arange(len(X))

plt.bar(X_axis - 0.2, Yiris, 0.4, label="Iris")
plt.bar(X_axis + 0.2, Ybc, 0.4, label="Breast Cancer")

plt.xlabel("Classifiers Used")
plt.ylabel("% of accuracy achieved")
plt.title("Accuracy of Classifiers using holdout")
plt.xticks(X_axis, X, rotation = 20)
plt.legend()
plt.show()
```

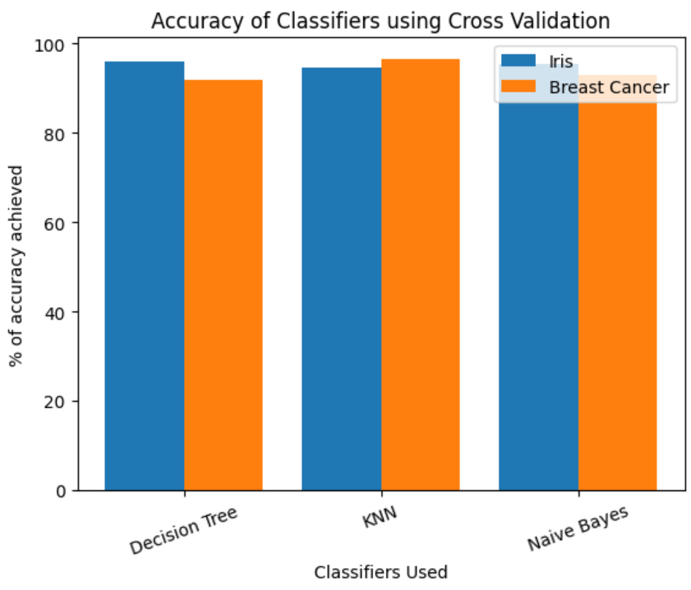


```
X = ['Decision Tree', 'KNN' ,'Naive Bayes']
Yiris = [acc_rs_iris_dt, acc_rs_iris_knn, acc_rs_iris_nb]
Ybc = [acc_rs_breast_dt, acc_rs_breast_knn, acc_rs_breast_nb]

X_axis = np.arange(len(X))

plt.bar(X_axis - 0.2, Yiris, 0.4, label="Iris")
plt.bar(X_axis + 0.2, Ybc, 0.4, label="Breast Cancer")

plt.xlabel("Classifiers Used")
plt.ylabel("% of accuracy achieved")
plt.title("Accuracy of Classifiers using Random Subsampling")
plt.xticks(X_axis, X, rotation = 20)
plt.legend()
plt.show()
```

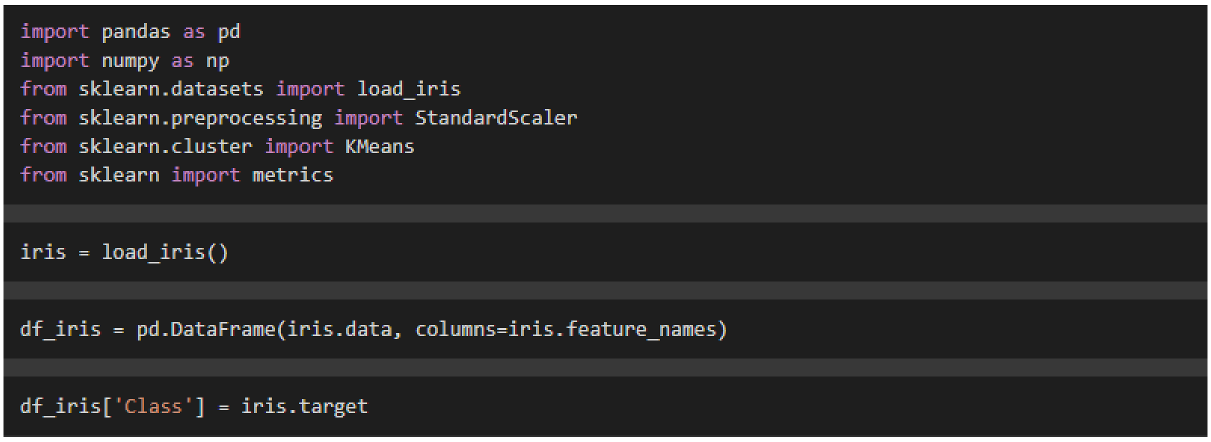




```
X = ['Decision Tree', 'KNN' ,'Naive Bayes']
Yiris = [acc_cv_iris_dt, acc_cv_iris_knn, acc_cv_iris_nb]
Ybc = [acc_cv_breast_dt, acc_cv_breast_knn, acc_cv_breast_nb]

X_axis = np.arange(len(X))

plt.bar(X_axis - 0.2, Yiris, 0.4, label="Iris")
plt.bar(X_axis + 0.2, Ybc, 0.4, label="Breast Cancer")

plt.xlabel("Classifiers Used")
plt.ylabel("% of accuracy achieved")
plt.title("Accuracy of Classifiers using Cross Validation")
plt.xticks(X_axis, X, rotation = 20)
plt.legend()
plt.show()
```



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

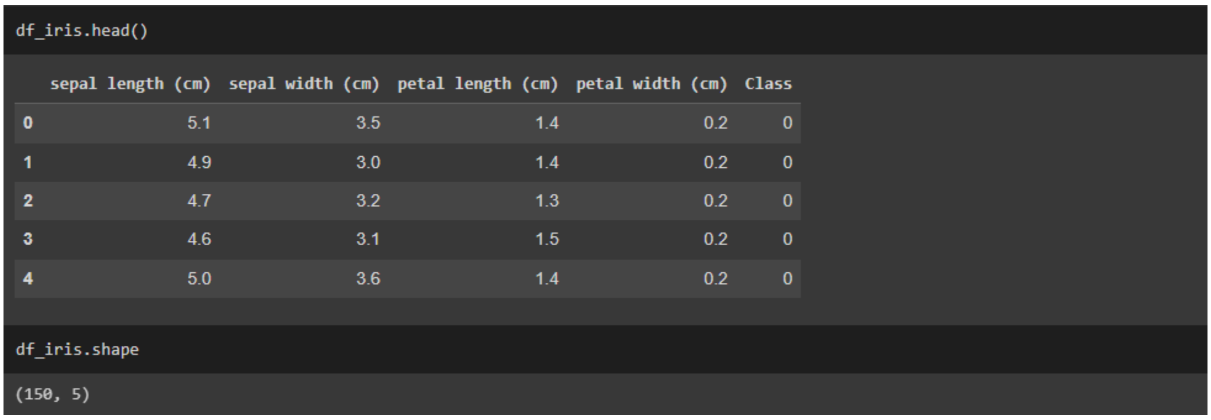


```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn import metrics
```

```python
iris = load_iris()
```

```python
df_iris = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```python
df_iris['Class'] = iris.target
```

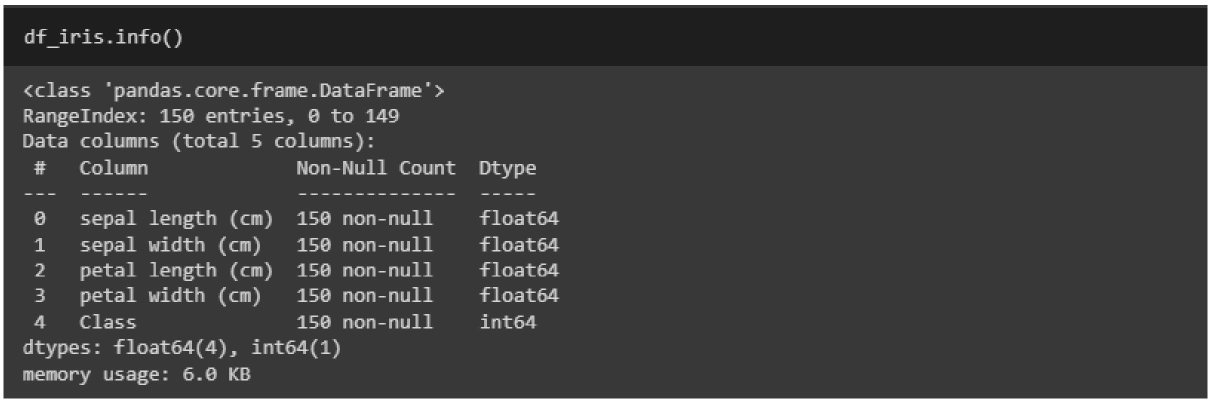


```python
df_iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

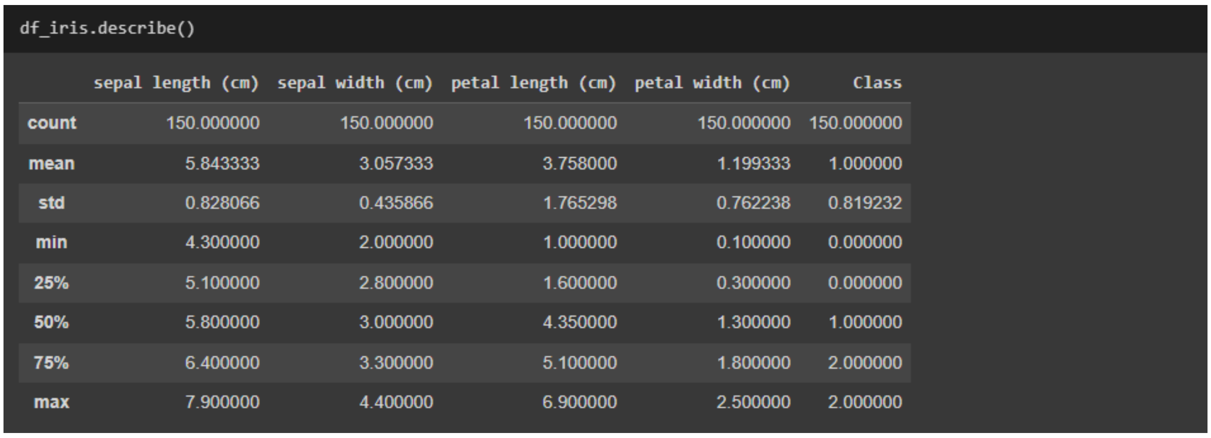
```python
df_iris.shape
```

```
(150, 5)
```



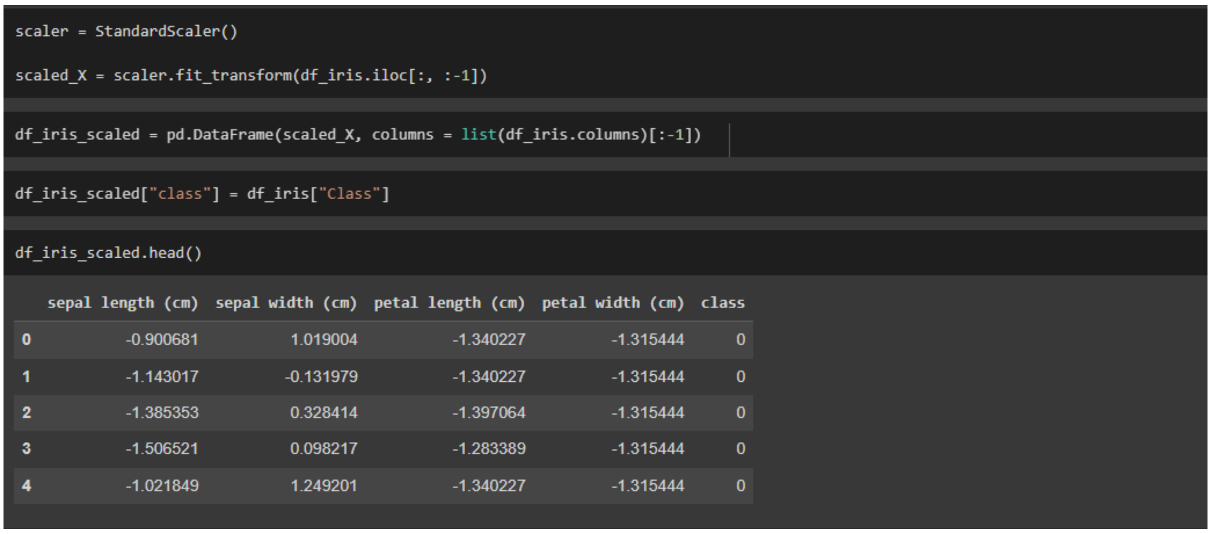
```python
df_iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   Class              150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```



```python
df_iris.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Class
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000



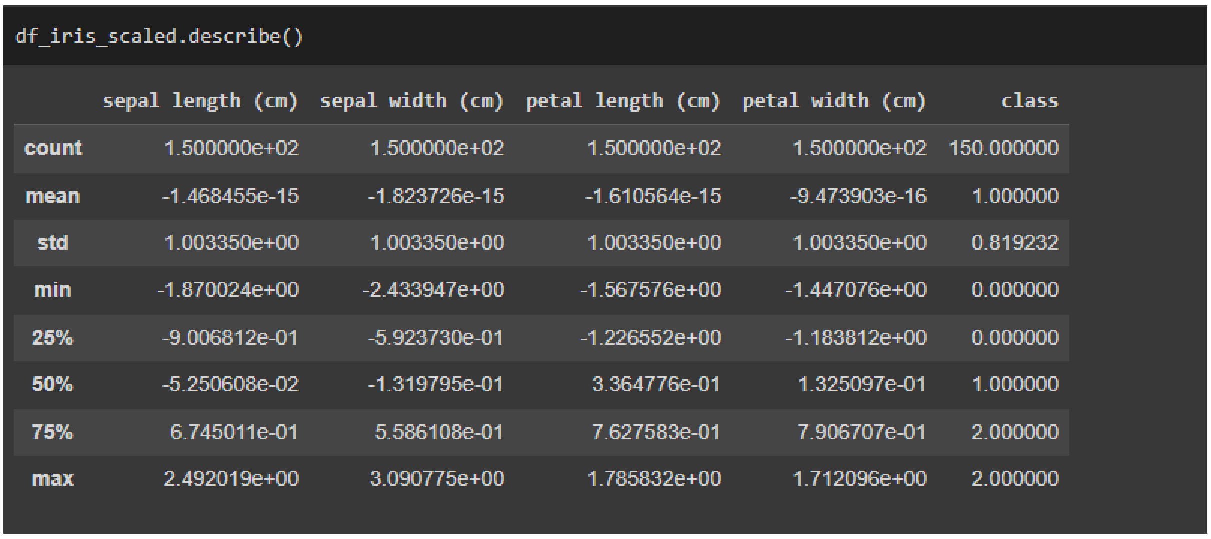
```
scaler = StandardScaler()

scaled_X = scaler.fit_transform(df_iris.iloc[:, :-1])
```

```
df_iris_scaled = pd.DataFrame(scaled_X, columns = list(df_iris.columns)[:-1])
```

```
df_iris_scaled["class"] = df_iris["Class"]
```

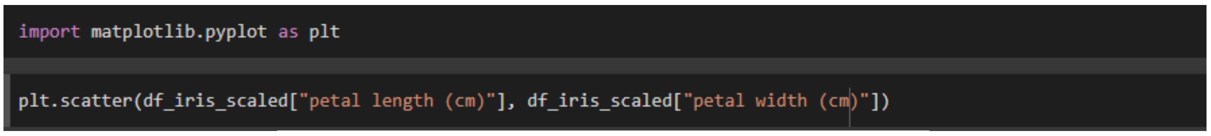
```
df_iris_scaled.head()
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  class
0          -0.900681          1.019004          -1.340227         -1.315444      0
1          -1.143017         -0.131979          -1.340227         -1.315444      0
2          -1.385353          0.328414          -1.397064         -1.315444      0
3          -1.506521          0.098217          -1.283389         -1.315444      0
4          -1.021849          1.249201          -1.340227         -1.315444      0
```



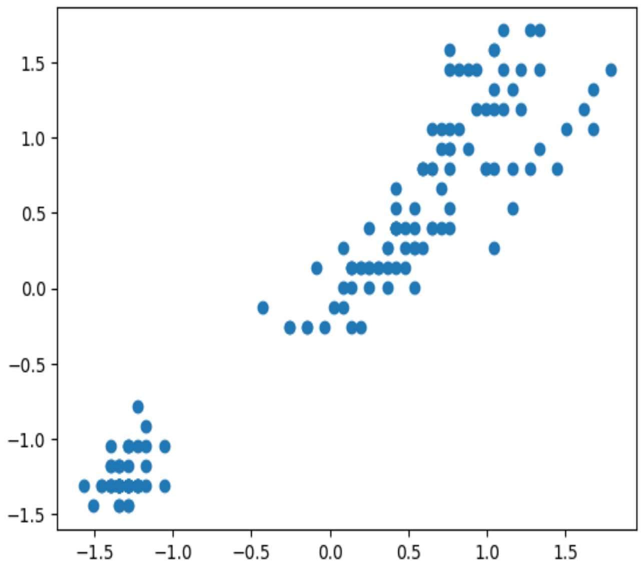
```
df_iris_scaled.describe()
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)       class
count       1.500000e+02      1.500000e+02       1.500000e+02      1.500000e+02  150.000000
mean       -1.468455e-15     -1.823726e-15      -1.610564e-15     -9.473903e-16    1.000000
std         1.003350e+00      1.003350e+00       1.003350e+00      1.003350e+00    0.819232
min        -1.870024e+00     -2.433947e+00      -1.567576e+00     -1.447076e+00    0.000000
25%        -9.006812e-01     -5.923730e-01      -1.226552e+00     -1.183812e+00    0.000000
50%        -5.250608e-02     -1.319795e-01       3.364776e-01      1.325097e-01    1.000000
75%         6.745011e-01      5.586108e-01       7.627583e-01      7.906707e-01    2.000000
max         2.492019e+00      3.090775e+00       1.785832e+00      1.712096e+00    2.000000
```



```
import matplotlib.pyplot as plt
```

```
plt.scatter(df_iris_scaled["petal length (cm)"], df_iris_scaled["petal width (cm)"])
```



```
X = df_iris_scaled.iloc[:, :-1]
Y = df_iris_scaled["class"]
```

Kmeans Clustering

```
from sklearn.cluster import KMeans
```

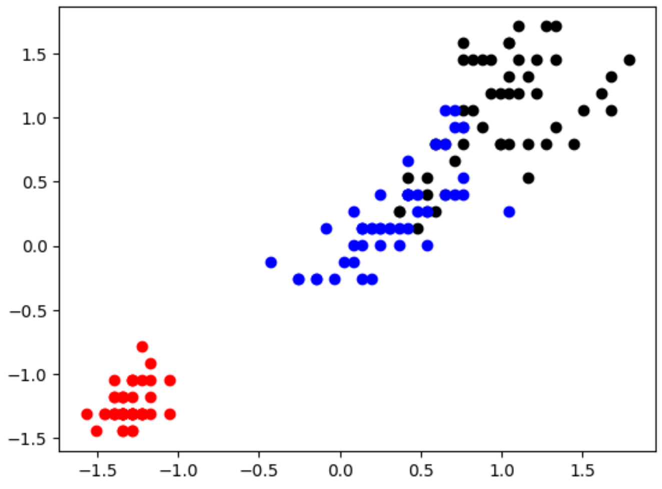
```
kmeans = KMeans(n_clusters = 3, random_state=1, n_init = 'auto')
kmeans.fit(X)
```

```
KMeans
KMeans(n_clusters=3, n_init='auto', random_state=1)
```

```
center = kmeans.cluster_centers_
```

```
identified_clusters = kmeans.fit_predict(X)
identified_clusters
```

```
temp1 = X[identified_clusters == 0]
temp2 = X[identified_clusters == 1]
temp3 = X[identified_clusters == 2]

#Plotting the results
plt.scatter(temp1.iloc[:,2] , temp1.iloc[:,3] , color = 'red')
plt.scatter(temp2.iloc[:,2] , temp2.iloc[:,3] , color = 'black')
plt.scatter(temp3.iloc[:,2], temp3.iloc[:, 3], color='blue')
plt.show()
```

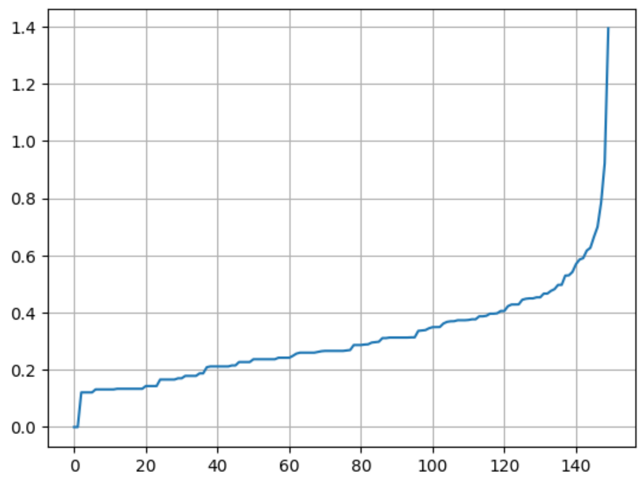


DBScan

```python
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
```

```python
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(X)
distances, indices = nbrs.kneighbors(X)
```

```python
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
plt.grid()
plt.show()
```



```python
dbscan = DBSCAN(eps=0.6, min_samples=3)
dbscan.fit(X)
```

```
DBSCAN
DBSCAN(eps=0.6, min_samples=3)
```

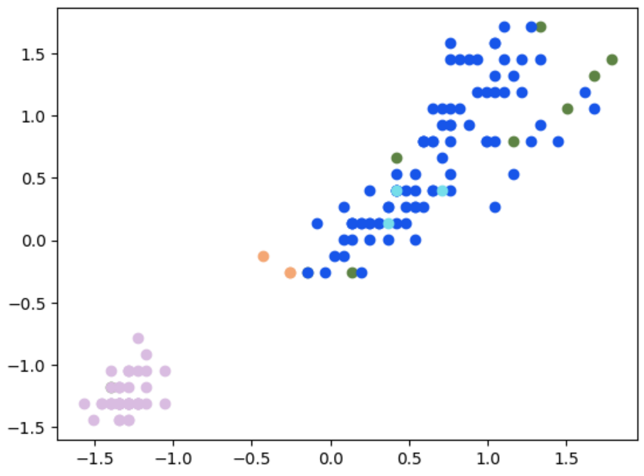
```python
identified_clusters_db = dbscan.fit_predict(X)
identified_clusters_db
```

```python
unique_clusters = np.unique(identified_clusters_db)
```

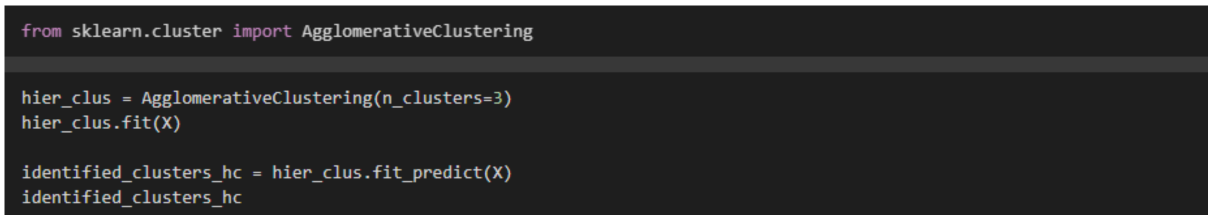
```python
li = []

for i in range(len(unique_clusters)):
  li.append(X[identified_clusters_db == unique_clusters[i]])
```

```python
import random
no_of_colors = len(li)

color=["#"+''.join([random.choice('0123456789ABCDEF') for i in range(6)]) for j in range(no_of_colors)]
```

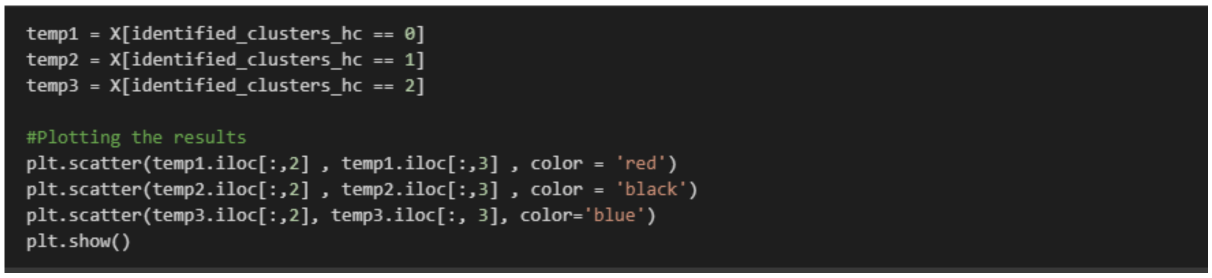
```python
#Plotting the results

for i in range(len(li)):
  plt.scatter(li[i].iloc[:,2] , li[i].iloc[:,3] , color = color[i])
plt.show()
```



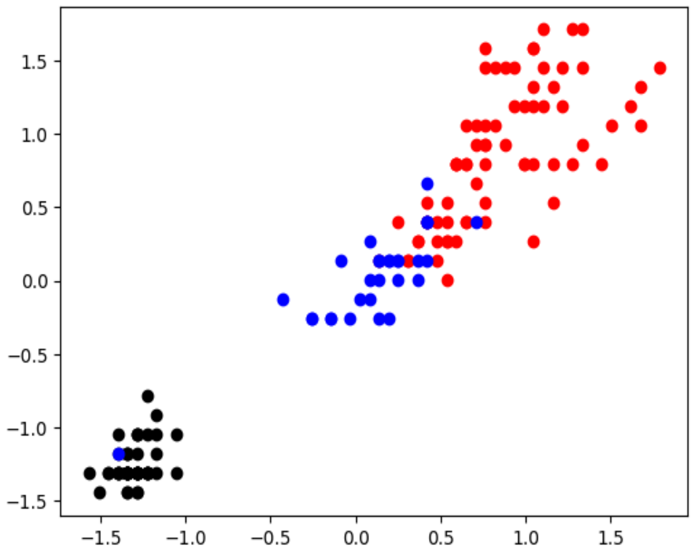
Hierarchical Clustering



```
from sklearn.cluster import AgglomerativeClustering

hier_clus = AgglomerativeClustering(n_clusters=3)
hier_clus.fit(X)

identified_clusters_hc = hier_clus.fit_predict(X)
identified_clusters_hc
```

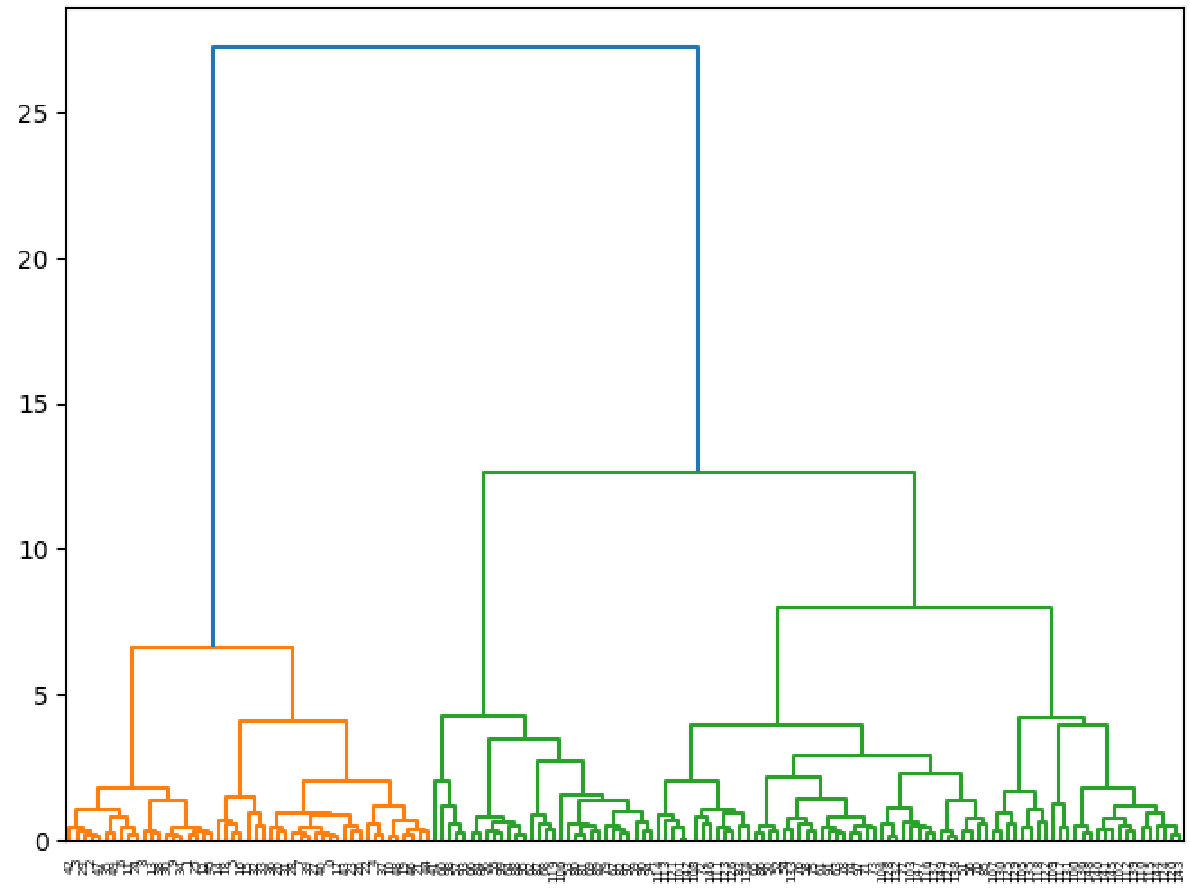


```
temp1 = X[identified_clusters_hc == 0]
temp2 = X[identified_clusters_hc == 1]
temp3 = X[identified_clusters_hc == 2]

#Plotting the results
plt.scatter(temp1.iloc[:,2] , temp1.iloc[:,3] , color = 'red')
plt.scatter(temp2.iloc[:,2] , temp2.iloc[:,3] , color = 'black')
plt.scatter(temp3.iloc[:,2], temp3.iloc[:, 3], color='blue')
plt.show()
```



```python
from scipy.cluster import hierarchy
```

```python
clusters = hierarchy.linkage(X, method="ward")

plt.figure(figsize=(8, 6))
dendrogram = hierarchy.dendrogram(clusters)
```



25
20
15
10
5
0