PRACTICAL 1

[2]:
```
import pandas as pd
import numpy as np
```

[5]:
```
df=pd.read_table('people.txt')
df
```

[5]:
```
Age agegroup height status yearsmarried
```

	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

[6]:
```
rule1=df['Age'].apply(lambda x:True if x>=0 and x<=150 else False)
rule1
```

[6]:
```
0     True
1     True
2     True
3    False
4     True
Name: Age, dtype: bool
```

[7]:
```
rule2=df.apply(lambda x: True if x.Age>x.yearsmarried else False, axis=1)
rule2
```

[7]:
```
0     True
1     True
2    False
3     True
4     True
dtype: bool
```

[8]:
```
rule3=df.apply(lambda x: True if x.status=="married" or x.status=="single" or x.
↪status=="widowed" else False, axis=1)
rule3
```

[8]:
```
0    True
1    True
2    True
3    True
4    True
dtype: bool
```

[9]:
```
rule4=df.apply(lambda x: True if(x.Age<18 and x.agegroup=="child")or
               (x.Age<65 and x.agegroup=="adult")or
               (x.Age>65 and x.agegroup=="elderly") else False, axis=1)
rule4
```

[9]:
```
0     True
1     True
2     True
3     True
4    False
dtype: bool
```

[10]:
```
rules=pd.DataFrame({"Rule 1":rule1,"Rule 2":rule2, "Rule 3":rule3, "Rule 4":
 ↪rule4})
rules
```

[10]:

	Rule 1	Rule 2	Rule 3	Rule 4
0	True	True	True	True
1	True	True	True	True
2	True	False	True	True
3	False	True	True	True
4	True	True	True	False

[11]:
```
rules.astype(int)
```

[11]:

	Rule 1	Rule 2	Rule 3	Rule 4
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

[12]:
```
print("numbers of rule violated : ",len (rules)-rules["Rule 1"].sum())
print("count of both \n", rules["Rule 1"].value_counts())
```

```
numbers of rule violated :  1
count of both
 True     4
False    1
Name: Rule 1, dtype: int64
```

[64]:
```
print("numbers of rule violated : ",len (rules)-rules["Rule 2"].sum())
print("count of both \n", rules["Rule 2"].value_counts())
```

```
numbers of rule violated :  1
count of both
 True     4
False    1
Name: Rule 2, dtype: int64
```

[65]:
```
print("numbers of rule violated : ",len (rules)-rules["Rule 3"].sum())
print("count of both \n", rules["Rule 3"].value_counts())
```

```
numbers of rule violated :  0
count of both
 True    5
Name: Rule 3, dtype: int64
```

[66]:
```
print("numbers of rule violated : ",len (rules)-rules["Rule 4"].sum())
print("count of both \n", rules["Rule 4"].value_counts())
```

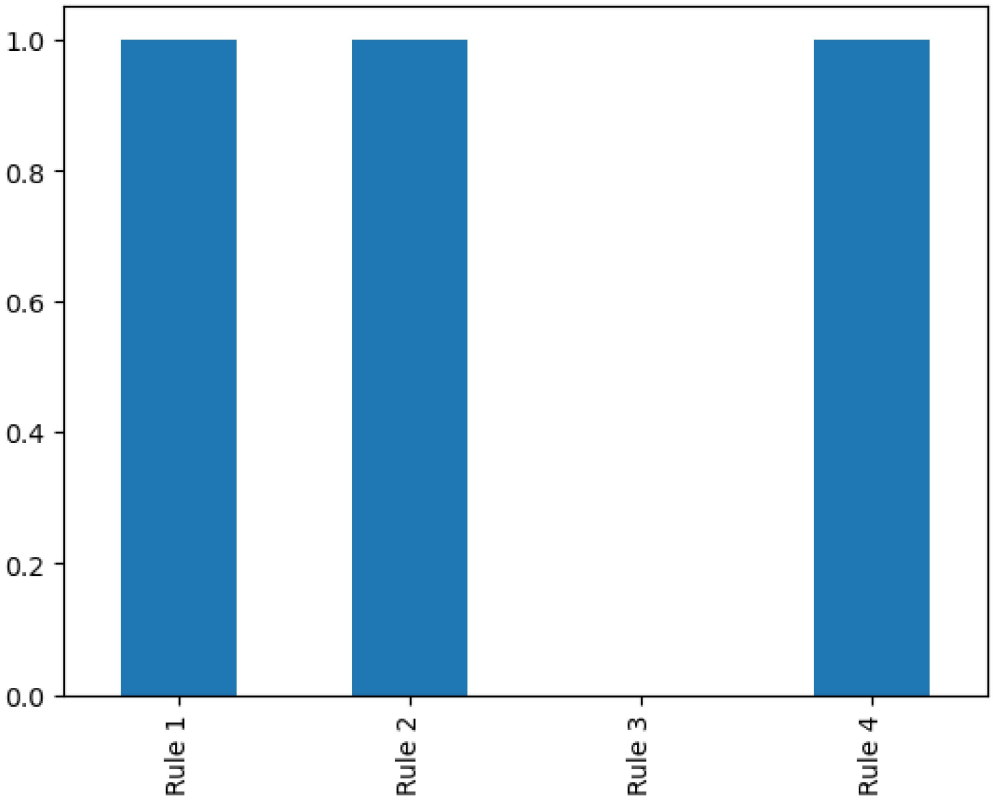
```
numbers of rule violated :  1
count of both
 True     4
False    1
Name: Rule 4, dtype: int64
```

[67]:
```
import matplotlib.pyplot as plt
plt.figure()
```

[67]:
```
<Figure size 640x480 with 0 Axes>
```

```
<Figure size 640x480 with 0 Axes>
```

[68]:
```
rules.apply(lambda x:len(x)-x.sum()).plot(kind='bar')
plt.xlabel="rules"
plt.ylabel="number of records that violates the rule"
```


1.0
0.8
0.6
0.4
0.2
0.0
Rule 1
Rule 2
Rule 3
Rule 4

PRACTICAL 2

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
df=pd.read_csv('dirty_iris.csv')
df
```

[2]:

	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
..	…	…	…	…	…
145	6.7	3.1	5.6	2.4	virginica
146	5.6	3.0	4.5	1.5	versicolor
147	5.2	3.5	1.5	0.2	setosa
148	6.4	3.1	NaN	1.8	virginica
149	5.8	2.6	4.0	NaN	versicolor

[150 rows x 5 columns]

[17]:
```
#i)
df1=df.dropna()
df1
```

[17]:

	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
..	…	…	…	…	…
143	6.3	2.9	5.6	1.8	virginica
144	5.7	2.5	5.0	2.0	virginica
145	6.7	3.1	5.6	2.4	virginica
146	5.6	3.0	4.5	1.5	versicolor
147	5.2	3.5	1.5	0.2	setosa

```
[96 rows x 5 columns]
```

[4]:
```
df2=(df.isna().any(axis=1)==0).sum()
df2
```

[4]:
```
96
```

[5]:
```
df3=(df2/len(df))*100
print("percentage of No. of observation",df3)
```

```
percentage of No. of observation 64.0
```

[29]:
```
#ii)
df['Sepal.Length'].unique()
df['Sepal.Width'].unique()
df['Petal.Length'].unique()
df['Petal.Width'].unique()
df['Petal.Width'].replace(np.inf,np.nan,inplace=True)
df['Petal.Width'].unique()
```

[29]:
```
array([1.5, 2.5, 2.3, 0.4, 1. , 0.2, nan, 1.8, 0.6, 1.6, 1.4, 1.3, 0.1,
       2.1, 2. , 1.2, 1.9, 2.2, 0.3, 1.1, 0. , 1.7, 2.4, 0.5])
```

[13]:
```
rule1=(df["Species"]=="setosa") | (df["Species"]=="virginica") |␣
↪(df["Species"]=="versicolor")
a=rule1.count()-rule1.sum()
a
```

[13]:
```
0
```

[18]:
```
rule2=(df["Sepal.Length"]>0) | (df["Petal.Length"]>0) | (df["Petal.Width"]>0) |␣
↪(df["Sepal.Length"]>0)
b=rule2.count()-rule2.sum()
b
```

```
0
```

[15]:
```
rule3=(df["Petal.Length"]) >(df["Petal.Width"].astype("float64")*2)
c=rule3.count()-rule3.sum()
c
```

[15]:
```
33
```

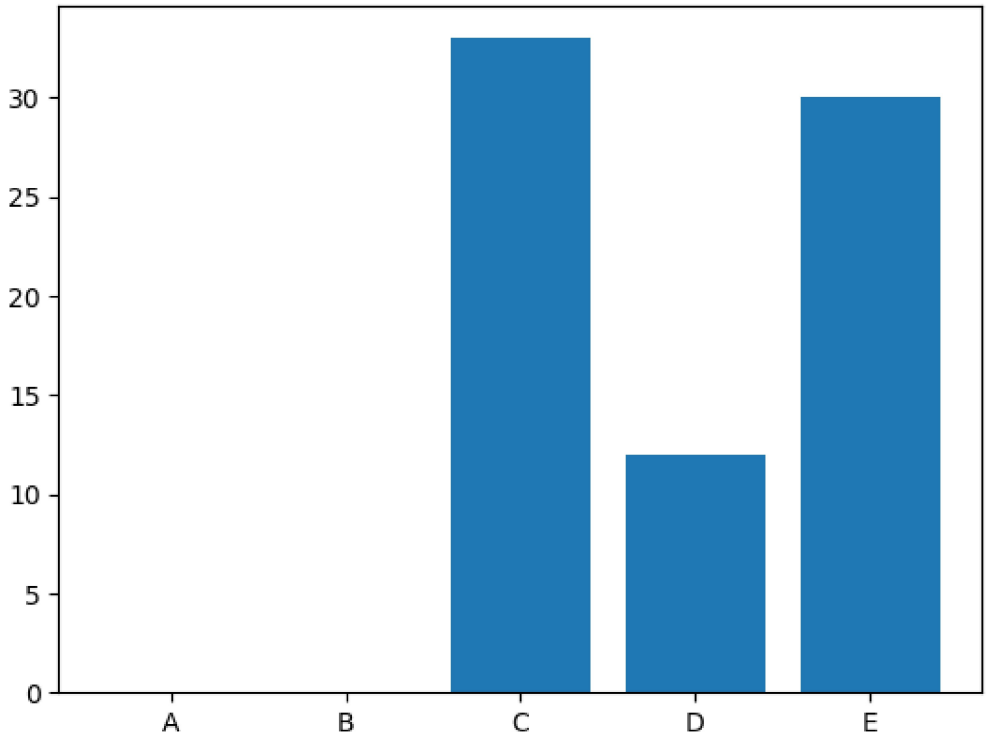
[16]:
```
rule4=df["Sepal.Length"]<30
d=rule4.count()-rule4.sum()
d
```

[16]: 12

[21]:
```
rule5= df["Sepal.Length"]>df["Petal.Length"]
e = rule5.count() - rule5.sum()
e
```

[21]: 30

[25]:
```
import matplotlib.pyplot as plt
x=["A","B","C","D","E"]
y=[a,b,c,d,e]
plt.bar(x,y)
rule5 = df["Sepal.Length"]>df["Petal.Length"]
```



PRACTICAL 3 NOrmalization

[1]:
```
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
```

[2]:
```
from sklearn.datasets import load_wine, load_iris iris=load_iris()
wine=load_wine()
```

[3]:
```
import pandas as pd
```

```
import numpy as np
df = pd.DataFrame(data=wine.data,columns=wine.feature_names)
df
```

[3]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	\
0	14.23	1.71	2.43	15.6	127.0	2.80	
1	13.20	1.78	2.14	11.2	100.0	2.65	
2	13.16	2.36	2.67	18.6	101.0	2.80	
3	14.37	1.95	2.50	16.8	113.0	3.85	
4	13.24	2.59	2.87	21.0	118.0	2.80	
..	…	…	…	…	…	…	
173	13.71	5.65	2.45	20.5	95.0	1.68	
174	13.40	3.91	2.48	23.0	102.0	1.80	
175	13.27	4.28	2.26	20.0	120.0	1.59	
176	13.17	2.59	2.37	20.0	120.0	1.65	
177	14.13	4.10	2.74	24.5	96.0	2.05	

	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	\
0	3.06	0.28	2.29	5.64	1.04	
1	2.76	0.26	1.28	4.38	1.05	
2	3.24	0.30	2.81	5.68	1.03	
3	3.49	0.24	2.18	7.80	0.86	
4	2.69	0.39	1.82	4.32	1.04	
..	…	…	…	…	…	
173	0.61	0.52	1.06	7.70	0.64	
174	0.75	0.43	1.41	7.30	0.70	
175	0.69	0.43	1.35	10.20	0.59	
176	0.68	0.53	1.46	9.30	0.60	
177	0.76	0.56	1.35	9.20	0.61	

```
                     diluted_wines proline
0                             3.92   1065.0
1                             3.40   1050.0
2                             3.17   1185.0
3                             3.45   1480.0
4                             2.93    735.0
..                             ...      ...
173                           1.74    740.0
174                           1.56    750.0
175                           1.56    835.0
176                           1.62    840.0
177                           1.60    560.0

[178 rows x 13 columns]
```

[4]:
```
min_max_scaler=MinMaxScaler()
normalized_data=min_max_scaler.fit_transform(df)
print(normalized_data)
```

```
[[0.84210526 0.1916996  0.57219251 … 0.45528455 0.97069597 0.56134094]
 [0.57105263 0.2055336  0.4171123  … 0.46341463 0.78021978 0.55064194]
 [0.56052632 0.3201581  0.70053476 … 0.44715447 0.6959707  0.64693295]
 …
 [0.58947368 0.69960474 0.48128342 … 0.08943089 0.10622711 0.39728959]
 [0.56315789 0.36561265 0.54010695 … 0.09756098 0.12820513 0.40085592]
 [0.81578947 0.66403162 0.73796791 … 0.10569106 0.12087912 0.20114123]]
```

[5]:
```
print(normalized_data.mean())
print(normalized_data.std())
```

```
0.4084913360554177
0.21688619487069796
```

[6]:
```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df
```

[6]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
```

```
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

[7]:
```
min_max_scaler=MinMaxScaler()
normalized_data=min_max_scaler.fit_transform(df)
print(normalized_data)
```

```
[[0.22222222 0.625      0.06779661 0.04166667]
 [0.16666667 0.41666667 0.06779661 0.04166667]
 [0.11111111 0.5        0.05084746 0.04166667]
 [0.08333333 0.45833333 0.08474576 0.04166667]
 [0.19444444 0.66666667 0.06779661 0.04166667]
 [0.30555556 0.79166667 0.11864407 0.125     ]
 [0.08333333 0.58333333 0.06779661 0.08333333]
 [0.19444444 0.58333333 0.08474576 0.04166667]
 [0.02777778 0.375      0.06779661 0.04166667]
 [0.16666667 0.45833333 0.08474576 0.        ]
 [0.30555556 0.70833333 0.08474576 0.04166667]
 [0.13888889 0.58333333 0.10169492 0.04166667]
 [0.13888889 0.41666667 0.06779661 0.        ]
 [0.         0.41666667 0.01694915 0.        ]
 [0.41666667 0.83333333 0.03389831 0.04166667]
 [0.38888889 1.         0.08474576 0.125     ]
 [0.30555556 0.79166667 0.05084746 0.125     ]
 [0.22222222 0.625      0.06779661 0.08333333]
 [0.38888889 0.75       0.11864407 0.08333333]
 [0.22222222 0.75       0.08474576 0.08333333]
 [0.30555556 0.58333333 0.11864407 0.04166667]
 [0.22222222 0.70833333 0.08474576 0.125     ]
 [0.08333333 0.66666667 0.         0.04166667]
 [0.22222222 0.54166667 0.11864407 0.16666667]
 [0.13888889 0.58333333 0.15254237 0.04166667]
 [0.19444444 0.41666667 0.10169492 0.04166667]
 [0.19444444 0.58333333 0.10169492 0.125     ]
 [0.25       0.625      0.08474576 0.04166667]
 [0.25       0.58333333 0.06779661 0.04166667]
 [0.11111111 0.5        0.10169492 0.04166667]
 [0.13888889 0.45833333 0.10169492 0.04166667]
 [0.30555556 0.58333333 0.08474576 0.125     ]
 [0.25       0.875      0.08474576 0.        ]
 [0.33333333 0.91666667 0.06779661 0.04166667]
 [0.16666667 0.45833333 0.08474576 0.04166667]
 [0.19444444 0.5        0.03389831 0.04166667]
 [0.33333333 0.625      0.05084746 0.04166667]
 [0.16666667 0.66666667 0.06779661 0.        ]
```

```
[0.02777778 0.41666667 0.05084746 0.04166667]
[0.22222222 0.58333333 0.08474576 0.04166667]
[0.19444444 0.625      0.05084746 0.08333333]
[0.05555556 0.125      0.05084746 0.08333333]
[0.02777778 0.5        0.05084746 0.04166667]
[0.19444444 0.625      0.10169492 0.20833333]
[0.22222222 0.75       0.15254237 0.125     ]
[0.13888889 0.41666667 0.06779661 0.08333333]
[0.22222222 0.75       0.10169492 0.04166667]
[0.08333333 0.5        0.06779661 0.04166667]
[0.27777778 0.70833333 0.08474576 0.04166667]
[0.19444444 0.54166667 0.06779661 0.04166667]
[0.75       0.5        0.62711864 0.54166667]
[0.58333333 0.5        0.59322034 0.58333333]
[0.72222222 0.45833333 0.66101695 0.58333333]
[0.33333333 0.125      0.50847458 0.5       ]
[0.61111111 0.33333333 0.61016949 0.58333333]
[0.38888889 0.33333333 0.59322034 0.5       ]
[0.55555556 0.54166667 0.62711864 0.625     ]
[0.16666667 0.16666667 0.38983051 0.375     ]
[0.63888889 0.375      0.61016949 0.5       ]
[0.25       0.29166667 0.49152542 0.54166667]
[0.19444444 0.         0.42372881 0.375     ]
[0.44444444 0.41666667 0.54237288 0.58333333]
[0.47222222 0.08333333 0.50847458 0.375     ]
[0.5        0.375      0.62711864 0.54166667]
[0.36111111 0.375      0.44067797 0.5       ]
[0.66666667 0.45833333 0.57627119 0.54166667]
[0.36111111 0.41666667 0.59322034 0.58333333]
[0.41666667 0.29166667 0.52542373 0.375     ]
[0.52777778 0.08333333 0.59322034 0.58333333]
[0.36111111 0.20833333 0.49152542 0.41666667]
[0.44444444 0.5        0.6440678  0.70833333]
[0.5        0.33333333 0.50847458 0.5       ]
[0.55555556 0.20833333 0.66101695 0.58333333]
[0.5        0.33333333 0.62711864 0.45833333]
[0.58333333 0.375      0.55932203 0.5       ]
[0.63888889 0.41666667 0.57627119 0.54166667]
[0.69444444 0.33333333 0.6440678  0.54166667]
[0.66666667 0.41666667 0.6779661  0.66666667]
[0.47222222 0.375      0.59322034 0.58333333]
[0.38888889 0.25       0.42372881 0.375     ]
[0.33333333 0.16666667 0.47457627 0.41666667]
[0.33333333 0.16666667 0.45762712 0.375     ]
[0.41666667 0.29166667 0.49152542 0.45833333]
[0.47222222 0.29166667 0.69491525 0.625     ]
[0.30555556 0.41666667 0.59322034 0.58333333]
[0.47222222 0.58333333 0.59322034 0.625     ]
```

```
[0.66666667 0.45833333 0.62711864 0.58333333]
[0.55555556 0.125      0.57627119 0.5       ]
[0.36111111 0.41666667 0.52542373 0.5       ]
[0.33333333 0.20833333 0.50847458 0.5       ]
[0.33333333 0.25       0.57627119 0.45833333]
[0.5        0.41666667 0.61016949 0.54166667]
[0.41666667 0.25       0.50847458 0.45833333]
[0.19444444 0.125      0.38983051 0.375     ]
[0.36111111 0.29166667 0.54237288 0.5       ]
[0.38888889 0.41666667 0.54237288 0.45833333]
[0.38888889 0.375      0.54237288 0.5       ]
[0.52777778 0.375      0.55932203 0.5       ]
[0.22222222 0.20833333 0.33898305 0.41666667]
[0.38888889 0.33333333 0.52542373 0.5       ]
[0.55555556 0.54166667 0.84745763 1.        ]
[0.41666667 0.29166667 0.69491525 0.75      ]
[0.77777778 0.41666667 0.83050847 0.83333333]
[0.55555556 0.375      0.77966102 0.70833333]
[0.61111111 0.41666667 0.81355932 0.875     ]
[0.91666667 0.41666667 0.94915254 0.83333333]
[0.16666667 0.20833333 0.59322034 0.66666667]
[0.83333333 0.375      0.89830508 0.70833333]
[0.66666667 0.20833333 0.81355932 0.70833333]
[0.80555556 0.66666667 0.86440678 1.        ]
[0.61111111 0.5        0.69491525 0.79166667]
[0.58333333 0.29166667 0.72881356 0.75      ]
[0.69444444 0.41666667 0.76271186 0.83333333]
[0.38888889 0.20833333 0.6779661  0.79166667]
[0.41666667 0.33333333 0.69491525 0.95833333]
[0.58333333 0.5        0.72881356 0.91666667]
[0.61111111 0.41666667 0.76271186 0.70833333]
[0.94444444 0.75       0.96610169 0.875     ]
[0.94444444 0.25       1.         0.91666667]
[0.47222222 0.08333333 0.6779661  0.58333333]
[0.72222222 0.5        0.79661017 0.91666667]
[0.36111111 0.33333333 0.66101695 0.79166667]
[0.94444444 0.33333333 0.96610169 0.79166667]
[0.55555556 0.29166667 0.66101695 0.70833333]
[0.66666667 0.54166667 0.79661017 0.83333333]
[0.80555556 0.5        0.84745763 0.70833333]
[0.52777778 0.33333333 0.6440678  0.70833333]
[0.5        0.41666667 0.66101695 0.70833333]
[0.58333333 0.33333333 0.77966102 0.83333333]
[0.80555556 0.41666667 0.81355932 0.625     ]
[0.86111111 0.33333333 0.86440678 0.75      ]
[1.         0.75       0.91525424 0.79166667]
[0.58333333 0.33333333 0.77966102 0.875     ]
[0.55555556 0.33333333 0.69491525 0.58333333]
```

```
[0.5        0.25       0.77966102 0.54166667]
[0.94444444 0.41666667 0.86440678 0.91666667]
[0.55555556 0.58333333 0.77966102 0.95833333]
[0.58333333 0.45833333 0.76271186 0.70833333]
[0.47222222 0.41666667 0.6440678  0.70833333]
[0.72222222 0.45833333 0.74576271 0.83333333]
[0.66666667 0.45833333 0.77966102 0.95833333]
[0.72222222 0.45833333 0.69491525 0.91666667]
[0.41666667 0.29166667 0.69491525 0.75      ]
[0.69444444 0.5        0.83050847 0.91666667]
[0.66666667 0.54166667 0.79661017 1.        ]
[0.66666667 0.41666667 0.71186441 0.91666667]
[0.55555556 0.20833333 0.6779661  0.75      ]
[0.61111111 0.41666667 0.71186441 0.79166667]
[0.52777778 0.58333333 0.74576271 0.91666667]
[0.44444444 0.41666667 0.69491525 0.70833333]]
```

[8]:
```
print(normalized_data.mean())
print(normalized_data.std())
```

```
0.4486931104833647
0.26236724966560704
```

PRACTICAL 3 Standardization

[15]:
```
from sklearn.datasets import load_iris import pandas
as pd
import numpy as np
```

[16]:
```
iris =load_iris()
iris
```

[16]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2], [4.9, 3. , 1.4,
        0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
```

```
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
```

```
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
```

```
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants
dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of
Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4
numeric, predictive attributes and the class\n    :Attribute Information:\n
- sepal length in cm\n        - sepal width in cm\n        - petal length in
cm\n        - petal width in cm\n        - class:\n                - Iris-
Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n
\n    :Summary Statistics:\n\n    ============== ==== ==== ======= =====
====================\n                    Min  Max   Mean    SD   Class
Correlation\n    ============== ==== ==== ======= ===== ====================\n
sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4
```

```
3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490
(high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n
============== ==== ==== ======= ===== ====================\n\n    :Missing
Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n
:Creator: R.A. Fisher\n    :Donor: Michael Marshall
(MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris
database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s
paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning
Repository, which has two wrong data points.\n\nThis is perhaps the best known
database to be found in the\npattern recognition literature.  Fisher\'s paper is
a classic in the field and\nis referenced frequently to this day.  (See Duda &
Hart, for example.)  The\ndata set contains 3 classes of 50 instances each,
where each class refers to a\ntype of iris plant.  One class is linearly
separable from the other 2; the\nlatter are NOT linearly separable from each
other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple
measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188
(1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley,
NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and
Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See
page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New
System\n     Structure and Classification Rule for Recognition in Partially
Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and
Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972)
"The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information
Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.
Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3
classes in the data.\n   - Many, many more …',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

[17]:
```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df['Class'] = iris.target
df.mean()
```

[17]:
```
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
Class                1.000000
dtype: float64
```

[18]:
```
df.iloc[:,:4]
df.std()
```

[18]:
```
sepal length (cm)    0.828066
sepal width (cm)     0.435866
petal length (cm)    1.765298
petal width (cm)     0.762238
Class                0.819232
dtype: float64
```

[33]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.iloc[:,:4])
scaled_data.mean()
```

[33]:
```
-1.4684549872375404e-15
```

[25]:
```
scaled_data.std()
```

[25]:
```
1.0
```

[20]:
```
from sklearn.datasets import load_wine
import pandas as pd
```

[31]:
```
wine = load_wine()
wine
```

[31]:
```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, …, 1.040e+00, 3.920e+00,
         1.065e+03],
        [1.320e+01, 1.780e+00, 2.140e+00, …, 1.050e+00, 3.400e+00,
         1.050e+03],
        [1.316e+01, 2.360e+00, 2.670e+00, …, 1.030e+00, 3.170e+00,
         1.185e+03],
        …,
        [1.327e+01, 4.280e+00, 2.260e+00, …, 5.900e-01, 1.560e+00,
         8.350e+02],
        [1.317e+01, 2.590e+00, 2.370e+00, …, 6.000e-01, 1.620e+00,
         8.400e+02],
        [1.413e+01, 4.100e+00, 2.740e+00, …, 6.100e-01, 1.600e+00,
         5.600e+02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2]),
```

```
 'frame': None,
 'target_names': array(['class_0', 'class_1', 'class_2'], dtype='<U7'),
 'DESCR': '.. _wine_dataset:\n\nWine recognition
dataset\n------------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 178 (50 in each of three classes)\n    :Number of
Attributes: 13 numeric, predictive attributes and the class\n    :Attribute
Information:\n \t\t- Alcohol\n \t\t- Malic acid\n \t\t- Ash\n\t\t- Alcalinity of
ash  \n \t\t- Magnesium\n\t\t- Total phenols\n \t\t- Flavanoids\n \t\t-
Nonflavanoid phenols\n \t\t- Proanthocyanins\n\t\t- Color intensity\n \t\t-
Hue\n \t\t- OD280/OD315 of diluted wines\n \t\t- Proline\n\n    - class:\n
- class_0\n            - class_1\n            - class_2\n\t\t\n    :Summary
Statistics:\n    \n    ============================== ==== ===== ======= =====\n
Min   Max   Mean     SD\n    ============================== ==== ===== =======
=====\n    Alcohol:                      11.0  14.8    13.0   0.8\n    Malic
Acid:                   0.74  5.80    2.34  1.12\n    Ash:
1.36  3.23    2.36  0.27\n    Alcalinity of Ash:            10.6  30.0    19.5
3.3\n    Magnesium:                    70.0 162.0    99.7  14.3\n    Total
Phenols:                0.98  3.88    2.29  0.63\n    Flavanoids:
0.34  5.08    2.03  1.00\n    Nonflavanoid Phenols:         0.13  0.66    0.36
0.12\n    Proanthocyanins:              0.41  3.58    1.59  0.57\n    Colour
Intensity:              1.3  13.0     5.1   2.3\n    Hue:
0.48  1.71    0.96  0.23\n    OD280/OD315 of diluted wines: 1.27  4.00    2.61
0.71\n    Proline:                      278  1680     746   315\n
============================== ==== ===== ======= =====\n\n    :Missing Attribute
Values: None\n    :Class Distribution: class_0 (59), class_1 (71), class_2
(48)\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall
(MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThis is a copy of UCI
ML Wine recognition datasets.\nhttps://archive.ics.uci.edu/ml/machine-learning-
databases/wine/wine.data\n\nThe data is the results of a chemical analysis of
wines grown in the same\nregion in Italy by three different cultivators. There
are thirteen different\nmeasurements taken for different constituents found in
the three types of\nwine.\n\nOriginal Owners: \n\nForina, M. et al, PARVUS -
\nAn Extendible Package for Data Exploration, Classification and Correlation.
\nInstitute of Pharmaceutical and Food Analysis and Technologies,\nVia Brigata
Salerno, 16147 Genoa, Italy.\n\nCitation:\n\nLichman, M. (2013). UCI Machine
Learning Repository\n[https://archive.ics.uci.edu/ml]. Irvine, CA: University of
California,\nSchool of Information and Computer Science. \n\n.. topic::
References\n\n  (1) S. Aeberhard, D. Coomans and O. de Vel, \n  Comparison of
Classifiers in High Dimensional Settings, \n  Tech. Rep. no. 92-02, (1992),
Dept. of Computer Science and Dept. of  \n  Mathematics and Statistics, James
Cook University of North Queensland. \n  (Also submitted to Technometrics). \n\n
The data was used with many others for comparing various \n  classifiers. The
classes are separable, though only RDA \n  has achieved 100% correct
classification. \n  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed
data)) \n  (All results using the leave-one-out technique) \n\n  (2) S.
Aeberhard, D. Coomans and O. de Vel, \n  "THE CLASSIFICATION PERFORMANCE OF RDA"
\n  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of \n
```

```
Mathematics and Statistics, James Cook University of North Queensland. \n  (Also
submitted to Journal of Chemometrics).\n',
 'feature_names': ['alcohol',
  'malic_acid',
  'ash',
  'alcalinity_of_ash',
  'magnesium',
  'total_phenols',
  'flavanoids',
  'nonflavanoid_phenols',
  'proanthocyanins',
  'color_intensity',
  'hue',
  'od280/od315_of_diluted_wines',
  'proline']}
```

[32]:
```
df1 = pd.DataFrame(wine.data,columns=wine.feature_names)
df1['Class'] = wine.target
df1
```

[32]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	\
0	14.23	1.71	2.43	15.6	127.0	2.80	
1	13.20	1.78	2.14	11.2	100.0	2.65	
2	13.16	2.36	2.67	18.6	101.0	2.80	
3	14.37	1.95	2.50	16.8	113.0	3.85	
4	13.24	2.59	2.87	21.0	118.0	2.80	
..	…	…	…	…	…	…	
173	13.71	5.65	2.45	20.5	95.0	1.68	
174	13.40	3.91	2.48	23.0	102.0	1.80	
175	13.27	4.28	2.26	20.0	120.0	1.59	
176	13.17	2.59	2.37	20.0	120.0	1.65	
177	14.13	4.10	2.74	24.5	96.0	2.05	

	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	\
0	3.06	0.28	2.29	5.64	1.04	
1	2.76	0.26	1.28	4.38	1.05	
2	3.24	0.30	2.81	5.68	1.03	
3	3.49	0.24	2.18	7.80	0.86	
4	2.69	0.39	1.82	4.32	1.04	
..	…	…	…	…	…	
173	0.61	0.52	1.06	7.70	0.64	
174	0.75	0.43	1.41	7.30	0.70	
175	0.69	0.43	1.35	10.20	0.59	
176	0.68	0.53	1.46	9.30	0.60	
177	0.76	0.56	1.35	9.20	0.61	

	od280/od315_of_diluted_wines	proline	Class

```
0                             3.92   1065.0      0
1                             3.40   1050.0      0
2                             3.17   1185.0      0
3                             3.45   1480.0      0
4                             2.93    735.0      0
..                             …      …    …
173                           1.74    740.0      2
174                           1.56    750.0      2
175                           1.56    835.0      2
176                           1.62    840.0      2
177                           1.60    560.0      2

[178 rows x 14 columns]
```

[26]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df1.iloc[:,:13])
print(scaled_data)
```

```
[[ 1.51861254 -0.5622498   0.23205254 …  0.36217728  1.84791957
   1.01300893]
 [ 0.24628963 -0.49941338 -0.82799632 …  0.40605066  1.1134493
   0.96524152]
 [ 0.19687903  0.02123125  1.10933436 …  0.31830389  0.78858745
   1.39514818]
 …
 [ 0.33275817  1.74474449 -0.38935541 … -1.61212515 -1.48544548
   0.28057537]
 [ 0.20923168  0.22769377  0.01273209 … -1.56825176 -1.40069891
   0.29649784]
 [ 1.39508604  1.58316512  1.36520822 … -1.52437837 -1.42894777
  -0.59516041]]
```

[28]:
```
scaled_data.mean()
```

[28]:
```
4.66735072755122e-16
```

[30]:
```
scaled_data.std()
```

[30]:
```
1.0
```

[34]:

[35]:

[36]:

Practical 4

[1]:
```
#!pip install efficient-apriori
```

[2]:
```
from efficient_apriori import apriori
import pandas as pd
import numpy as np
```

[3]:
```
groceries=pd.read_csv('groceriesDataset4.csv') bakery=pd.read_csv('1000-out
2.csv')
```

[4]:
```
groceries
```

[4]:

	Item(s)	Item 1	Item 2	Item 3	\
0	4	citrus fruit	semi-finished bread	margarine	
1	3	tropical fruit	yogurt	coffee	
2	1	whole milk	NaN	NaN	
3	4	pip fruit	yogurt	cream cheese	
4	4	other vegetables	whole milk	condensed milk	
…	…	…	…	…	
9830	17	sausage	chicken	beef	
9831	1	cooking chocolate	NaN	NaN	
9832	10	chicken	citrus fruit	other vegetables	
9833	4	semi-finished bread	bottled water	soda	
9834	5	chicken	tropical fruit	other vegetables	

	Item 4	Item 5	Item 6	\
0	ready soups	NaN	NaN	
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	meat spreads	NaN	NaN	
4	long life bakery product	NaN	NaN	
…	…	…	…	
9830	hamburger meat	citrus fruit	grapes	
9831	NaN	NaN	NaN	
9832	butter	yogurt	frozen dessert	
9833	bottled beer	NaN	NaN	
9834	vinegar	shopping bags	NaN	

```
                Item 7      Item 8  Item 9  … Item 23 Item 24 Item 25  \
0                  NaN         NaN     NaN  …     NaN      NaN      NaN
1                  NaN         NaN     NaN  …     NaN      NaN      NaN
2                  NaN         NaN     NaN  …     NaN      NaN      NaN
3                  NaN         NaN     NaN  …     NaN      NaN      NaN
4                  NaN         NaN     NaN  …     NaN      NaN      NaN
…                  …           …       …    …     …        …        …
9830  root vegetables  whole milk  butter  …     NaN      NaN      NaN
9831               NaN         NaN     NaN  …     NaN      NaN      NaN
9832    domestic eggs  rolls/buns     rum  …     NaN      NaN      NaN
9833               NaN         NaN     NaN  …     NaN      NaN      NaN
9834               NaN         NaN     NaN  …     NaN      NaN      NaN

     Item 26 Item 27 Item 28 Item 29 Item 30 Item 31 Item 32
0        NaN     NaN     NaN     NaN     NaN     NaN     NaN
1        NaN     NaN     NaN     NaN     NaN     NaN     NaN
2        NaN     NaN     NaN     NaN     NaN     NaN     NaN
3        NaN     NaN     NaN     NaN     NaN     NaN     NaN
4        NaN     NaN     NaN     NaN     NaN     NaN     NaN
…        …       …       …       …       …       …       …
9830     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9831     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9832     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9833     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9834     NaN     NaN     NaN     NaN     NaN     NaN     NaN

[9835 rows x 33 columns]
```

[5]:
```
groceries=groceries.drop(['Item(s)'],axis=1)
groceries
```

[5]:
```
                    Item 1               Item 2           Item 3  \
0            citrus fruit  semi-finished bread         margarine
1         tropical fruit                yogurt           coffee
2              whole milk                   NaN              NaN
3               pip fruit                yogurt      cream cheese
4      other vegetables           whole milk    condensed milk
…                    …                    …                …
9830            sausage              chicken             beef
9831  cooking chocolate                   NaN              NaN
9832            chicken         citrus fruit  other vegetables
9833  semi-finished bread        bottled water             soda
9834            chicken       tropical fruit  other vegetables

                       Item 4         Item 5          Item 6  \
0                 ready soups            NaN             NaN
1                         NaN            NaN             NaN
```

```
2                          NaN            NaN             NaN
3                meat spreads             NaN             NaN
4     long life bakery product            NaN             NaN
…                          …              …               …
9830           hamburger meat   citrus fruit           grapes
9831                      NaN            NaN             NaN
9832                   butter         yogurt  frozen dessert
9833            bottled beer             NaN             NaN
9834                  vinegar  shopping bags             NaN

               Item 7      Item 8  Item 9             Item 10  … Item 23  \
0                 NaN         NaN     NaN                 NaN  …     NaN
1                 NaN         NaN     NaN                 NaN  …     NaN
2                 NaN         NaN     NaN                 NaN  …     NaN
3                 NaN         NaN     NaN                 NaN  …     NaN
4                 NaN         NaN     NaN                 NaN  …     NaN
…                 …           …       …                   …  …     …
9830  root vegetables  whole milk  butter  whipped/sour cream  …     NaN
9831              NaN         NaN     NaN                 NaN  …     NaN
9832    domestic eggs  rolls/buns     rum     cling film/bags  …     NaN
9833              NaN         NaN     NaN                 NaN  …     NaN
9834              NaN         NaN     NaN                 NaN  …     NaN

     Item 24 Item 25 Item 26 Item 27 Item 28 Item 29 Item 30 Item 31 Item 32
0        NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
1        NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
2        NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
3        NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
4        NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
…        …       …       …       …       …       …       …       …
9830     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9831     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9832     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9833     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
9834     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN

[9835 rows x 32 columns]
```

[6]:
```
f=lambda record:[x for x in record if x is not np.nan]
transactions=list(map(f, groceries.values))
transactions[:5]
```

[6]:
```
[['citrus fruit', 'semi-finished bread', 'margarine', 'ready soups'],
 ['tropical fruit', 'yogurt', 'coffee'],
 ['whole milk'],
 ['pip fruit', 'yogurt', 'cream cheese', 'meat spreads'],
 ['other vegetables',
```

```
 'whole milk',
 'condensed milk',
 'long life bakery product']]
```

[7]:
```
itemsets, rules=apriori(transactions, min_support=0.055, min_confidence=0.035)
```

[8]:
```
itemsets
```

[8]:
```
{1: {('citrus fruit',): 814,
  ('margarine',): 576,
  ('tropical fruit',): 1032,
  ('yogurt',): 1372,
  ('coffee',): 571,
  ('whole milk',): 2513,
  ('pip fruit',): 744,
  ('other vegetables',): 1903,
  ('butter',): 545,
  ('rolls/buns',): 1809,
  ('bottled beer',): 792,
  ('bottled water',): 1087,
  ('frankfurter',): 580,
  ('soda',): 1715,
  ('fruit/vegetable juice',): 711,
  ('newspapers',): 785,
  ('pastry',): 875,
  ('root vegetables',): 1072,
  ('canned beer',): 764,
  ('sausage',): 924,
  ('brown bread',): 638,
  ('shopping bags',): 969,
  ('pork',): 567,
  ('whipped/sour cream',): 705,
  ('domestic eggs',): 624},
 2: {('other vegetables', 'whole milk'): 736,
  ('rolls/buns', 'whole milk'): 557,
  ('whole milk', 'yogurt'): 551}}
```

[9]:
```
rules
```

[9]:
```
[{whole milk} -> {other vegetables},
 {other vegetables} -> {whole milk},
 {whole milk} -> {rolls/buns},
 {rolls/buns} -> {whole milk},
 {yogurt} -> {whole milk},
 {whole milk} -> {yogurt}]
```

[10]:
```
frequent_itemsets=[]
for size, itemset in itemsets.items():
    for items, support in itemset.items():
        frequent_itemsets.append((items, support))
```

[11]:
```
frequent_itemsets
```

[11]:
```
[(('citrus fruit',), 814),
 (('margarine',), 576),
 (('tropical fruit',), 1032),
 (('yogurt',), 1372),
 (('coffee',), 571),
 (('whole milk',), 2513),
 (('pip fruit',), 744),
 (('other vegetables',), 1903),
 (('butter',), 545),
 (('rolls/buns',), 1809),
 (('bottled beer',), 792),
 (('bottled water',), 1087),
 (('frankfurter',), 580),
 (('soda',), 1715),
 (('fruit/vegetable juice',), 711),
 (('newspapers',), 785),
 (('pastry',), 875),
 (('root vegetables',), 1072),
 (('canned beer',), 764),
 (('sausage',), 924),
 (('brown bread',), 638),
 (('shopping bags',), 969),
 (('pork',), 567),
 (('whipped/sour cream',), 705),
 (('domestic eggs',), 624),
 (('other vegetables', 'whole milk'), 736),
 (('rolls/buns', 'whole milk'), 557),
 (('whole milk', 'yogurt'), 551)]
```

[12]:
```
sorted_itemsets=sorted(frequent_itemsets, key=lambda x:x[1], reverse=True)
```

[13]:
```
for items, support in sorted_itemsets:
    print(f"Items:{','.join(items)}-Support:{support}")
```

```
Items:whole milk-Support:2513
Items:other vegetables-Support:1903
Items:rolls/buns-Support:1809
Items:soda-Support:1715
Items:yogurt-Support:1372
Items:bottled water-Support:1087
```

```
Items:root vegetables-Support:1072
Items:tropical fruit-Support:1032
Items:shopping bags-Support:969
Items:sausage-Support:924
Items:pastry-Support:875
Items:citrus fruit-Support:814
Items:bottled beer-Support:792
Items:newspapers-Support:785
Items:canned beer-Support:764
Items:pip fruit-Support:744
Items:other vegetables,whole milk-Support:736
Items:fruit/vegetable juice-Support:711
Items:whipped/sour cream-Support:705
Items:brown bread-Support:638
Items:domestic eggs-Support:624
Items:frankfurter-Support:580
Items:margarine-Support:576
Items:coffee-Support:571
Items:pork-Support:567
Items:rolls/buns,whole milk-Support:557
Items:whole milk,yogurt-Support:551
Items:butter-Support:545
```

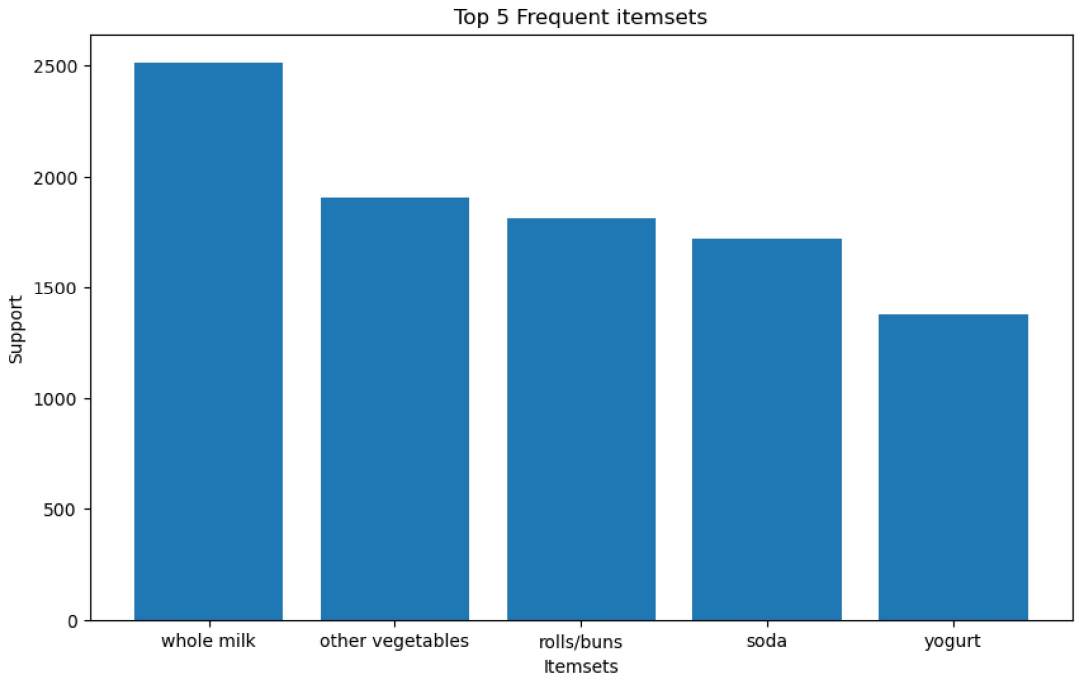
[14]:
```
top5=sorted_itemsets[:5]
top5
```

[14]:
```
[(('whole milk',), 2513),
 (('other vegetables',), 1903),
 (('rolls/buns',), 1809),
 (('soda',), 1715),
 (('yogurt',), 1372)]
```

[15]:
```
import matplotlib.pyplot as plt
```

[16]:
```
item_names=[','.join(items) for items, _ in top5]
support_values=[support for _, support in top5]
```

[17]:
```
plt.figure(figsize=(10,6))
plt.bar(item_names,support_values)
plt.xlabel('Itemsets')
plt.ylabel('Support')
plt.title('Top 5 Frequent itemsets')
plt.show()
```



[18]:
```
bakery=pd.read_csv('1000i.csv')
```

[19]:
```
bakery
```

[19]:

	1	3	7
0	1	4	15
1	1	2	49
2	1	5	44
3	2	1	1
4	2	2	19
…	…	..	..
3532	999	5	35
3533	999	2	3
3534	1000	4	15
3535	1000	3	47
3536	1000	3	34

[3537 rows x 3 columns]

[20]:
```
f=lambda record:[x for x in record if x is not np.nan]
transactions=list(map(f, bakery.values))
transactions[:5]
```

[20]:
```
[[1, 4, 15], [1, 2, 49], [1, 5, 44], [2, 1, 1], [2, 2, 19]]
```

[21]:
```
itemsets, rules=apriori(transactions, min_support=0.011, min_confidence=0.010)
itemsets
```

[21]:
```
{1: {(1,): 787,
  (4,): 851,
  (15,): 76,
  (2,): 748,
  (49,): 61,
  (5,): 745,
  (44,): 80,
  (19,): 80,
  (3,): 766,
  (18,): 87,
  (35,): 79,
  (9,): 92,
  (23,): 84,
  (7,): 98,
  (14,): 98,
  (21,): 48,
  (12,): 82,
  (31,): 92,
  (36,): 86,
  (48,): 80,
  (8,): 40,
  (28,): 104,
  (27,): 95,
  (10,): 44,
  (11,): 73,
  (24,): 69,
  (40,): 71,
  (41,): 75,
  (43,): 65,
  (20,): 44,
  (13,): 57,
  (26,): 50,
  (22,): 111,
  (39,): 62,
  (16,): 83,
  (32,): 80,
  (45,): 98,
  (17,): 57,
  (37,): 69,
  (47,): 75,
  (34,): 47,
  (25,): 41,
  (29,): 64,
  (0,): 84,
```

```
 (42,): 84,
 (30,): 51,
 (33,): 80,
 (46,): 88},
2: {(1, 2): 40, (1, 4): 40, (2, 4): 39, (3, 4): 39, (4, 5): 40}}
```

[22]:
```
rules
```

[22]:
```
[{2} -> {1},
 {1} -> {2},
 {4} -> {1},
 {1} -> {4},
 {4} -> {2},
 {2} -> {4},
 {4} -> {3},
 {3} -> {4},
 {5} -> {4},
 {4} -> {5}]
```

[23]:
```
frequent_itemsets=[]
for size, itemset in itemsets.items():
    for items, support in itemset.items():
        frequent_itemsets.append((items, support))
```

[24]:
```
frequent_itemsets
```

[24]:
```
[((1,), 787),
 ((4,), 851),
 ((15,), 76),
 ((2,), 748),
 ((49,), 61),
 ((5,), 745),
 ((44,), 80),
 ((19,), 80),
 ((3,), 766),
 ((18,), 87),
 ((35,), 79),
 ((9,), 92),
 ((23,), 84),
 ((7,), 98),
 ((14,), 98),
 ((21,), 48),
 ((12,), 82),
 ((31,), 92),
 ((36,), 86),
 ((48,), 80),
 ((8,), 40),
```

```
 ((28,), 104),
 ((27,), 95),
 ((10,), 44),
 ((11,), 73),
 ((24,), 69),
 ((40,), 71),
 ((41,), 75),
 ((43,), 65),
 ((20,), 44),
 ((13,), 57),
 ((26,), 50),
 ((22,), 111),
 ((39,), 62),
 ((16,), 83),
 ((32,), 80),
 ((45,), 98),
 ((17,), 57),
 ((37,), 69),
 ((47,), 75),
 ((34,), 47),
 ((25,), 41),
 ((29,), 64),
 ((0,), 84),
 ((42,), 84),
 ((30,), 51),
 ((33,), 80),
 ((46,), 88),
 ((1, 2), 40),
 ((1, 4), 40),
 ((2, 4), 39),
 ((3, 4), 39),
 ((4, 5), 40)]
```

[25]:
```
sorted_itemsets=sorted(frequent_itemsets, key=lambda x:x[1], reverse=True)
```

[26]:
```
top5=sorted_itemsets[:5]
top5
```

[26]:
```
[((4,), 851), ((1,), 787), ((3,), 766), ((2,), 748), ((5,), 745)]
```

Practical 5 (Iris)

IRIS DATASET

```
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.tree import plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings

iris = load_iris()
x=iris.data
y=iris.target
print(x.shape)
print(y.shape)
```

```
(150, 4)
(150,)
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x)
```

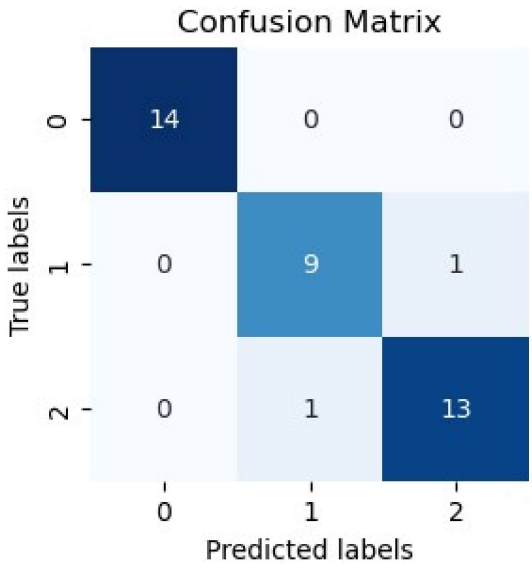
1. Hold Out Method, Test size = 25%

```
k_size = 0.25
r_seeds = 100
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size =
k_size, random_state=r_seeds)
print("Shape of X_train: ", X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of y_test: ", y_test.shape)
```

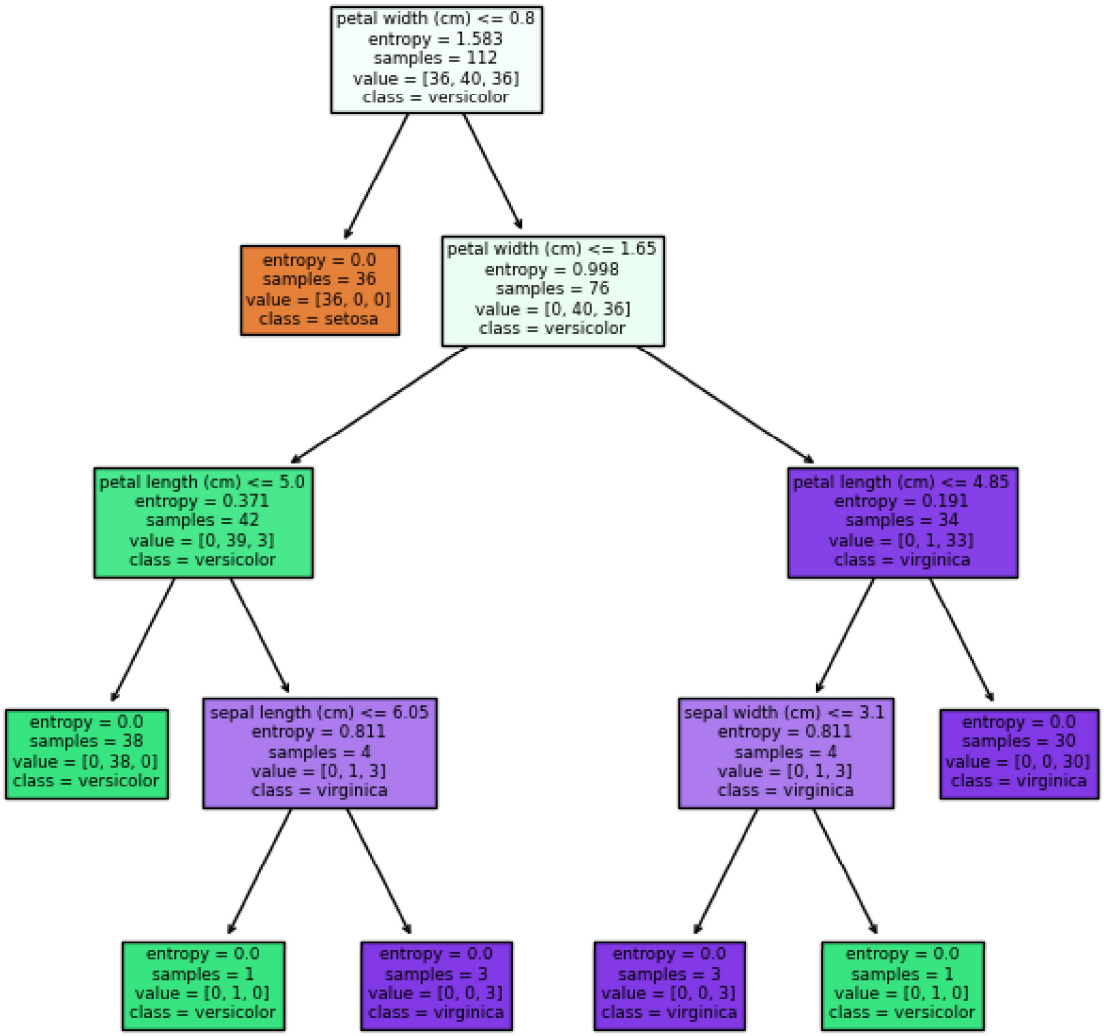
```
Shape of X_train:  (112, 4)
Shape of X_test:  (38, 4)
Shape of y_train:  (112,)
Shape of y_test:  (38,)
```

1. Decision Tree Classification

```
deci_tree = DecisionTreeClassifier(criterion='entropy')
deci_tree.fit(X_train, y_train)
prediction = deci_tree.predict(X_test)

cm = confusion_matrix(y_test, prediction)

plt.figure(figsize=(3, 3))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Plot the decision tree
plt.figure(figsize=(8, 8))
plot_tree(deci_tree, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names)

plt.show()
```




```
accuracy_dt = accuracy_score(y_test, prediction)
print("Accuracy on Hold Out method using classification:",
accuracy_dt*100,"%")

report = classification_report(y_test, prediction)
print("Classification Report:\n", report)
```

```
Accuracy on Hold Out method using classification: 94.73684210526315 %
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.90      0.90      0.90        10
```

```
           2       0.93      0.93      0.93        14

    accuracy                           0.95        38
   macro avg       0.94      0.94      0.94        38
weighted avg       0.95      0.95      0.95        38
```

1. KNearest Neighbour Classification

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


# Ignore FutureWarning from scikit-learn
warnings.filterwarnings("ignore", category=FutureWarning)

accuracy_knn = accuracy_score(y_test, prediction2)
print("Accuracy on Hold Out method using classification:",
accuracy_knn*100,"%")

report = classification_report(y_test, prediction2)
print("Classification Report:\n", report)
```

```
Accuracy on Hold Out method using classification: 97.36842105263158 %
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.91      1.00      0.95        10
           2       1.00      0.93      0.96        14

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

1. Naive Bayes

```
nb = GaussianNB()
nb.fit(X_train, y_train)

GaussianNB()

prediction3 = nb.predict(X_test)

accuracy_nb = accuracy_score(y_test, prediction3)
print("Accuracy on Hold Out method using classification:",
accuracy_nb*100,"%")

report = classification_report(y_test, prediction3)
print("Classification Report:\n", report)
```

```
Accuracy on Hold Out method using classification: 94.73684210526315 %
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.90      0.90      0.90        10
           2       0.93      0.93      0.93        14

    accuracy                           0.95        38
   macro avg       0.94      0.94      0.94        38
weighted avg       0.95      0.95      0.95        38
```

1. Random Subsampling, Test Size = 25%

```
iris = load_iris()
x = iris.data
y = iris.target

subsample_size = 25
test_size = 0.25  # 25% of the data will be used for testing
indices = np.random.choice(len(x), size=subsample_size, replace=True)

x_subsample = x[indices]
y_subsample = y[indices]

# Split the subsampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_subsample,
y_subsample, test_size=test_size, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Decision Tree Classifier on the scaled training data
deci_tree = DecisionTreeClassifier(criterion="entropy")
deci_tree.fit(X_train_scaled, y_train)

# Predict on the scaled test data
y_pred = deci_tree.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```python
# Generate classification report
report = classification_report(y_test, y_pred,
target_names=iris.target_names)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.7142857142857143
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         3
  versicolor       0.00      0.00      0.00         2
   virginica       0.50      1.00      0.67         2

    accuracy                           0.71         7
   macro avg       0.50      0.67      0.56         7
weighted avg       0.57      0.71      0.62         7


C:\Users\CSLab\anaconda3\lib\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\CSLab\anaconda3\lib\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\CSLab\anaconda3\lib\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Practical 5 (Breast Cancer)

[60]:
```
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler from sklearn.model_selection
import StratifiedKFold from sklearn.datasets import load_breast_cancer from
sklearn.neighbors import KNeighborsClassifier from sklearn.tree import
DecisionTreeClassifier from sklearn.metrics import classification_report from
sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split from sklearn.metrics import␣
confusion_matrix,precision_score,recall_score,f1_score,ConfusionMatrixDisplay
```

[61]:
```
svm = load_breast_cancer()
x = svm.data
y = svm.target
```

[62]:
```
print(x.shape)
```

```
(569, 30)
```

[63]:
```
print(y.shape)
```

```
(569,)
```

[64]:
```
k_size = 0.25
r_seed = 100
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=k_size,random_state=r_seed)
```

[65]:
```
print("Shape of X_train" ,X_train.shape)
print("Shape of y_train" ,y_train.shape)
print("Shape of X_train" ,X_test.shape)
print("Shape of y_train" ,y_test.shape)
```

```
Shape of X_train (426, 30)
Shape of y_train (426,)
Shape of X_train (143, 30)
Shape of y_train (143,)
```

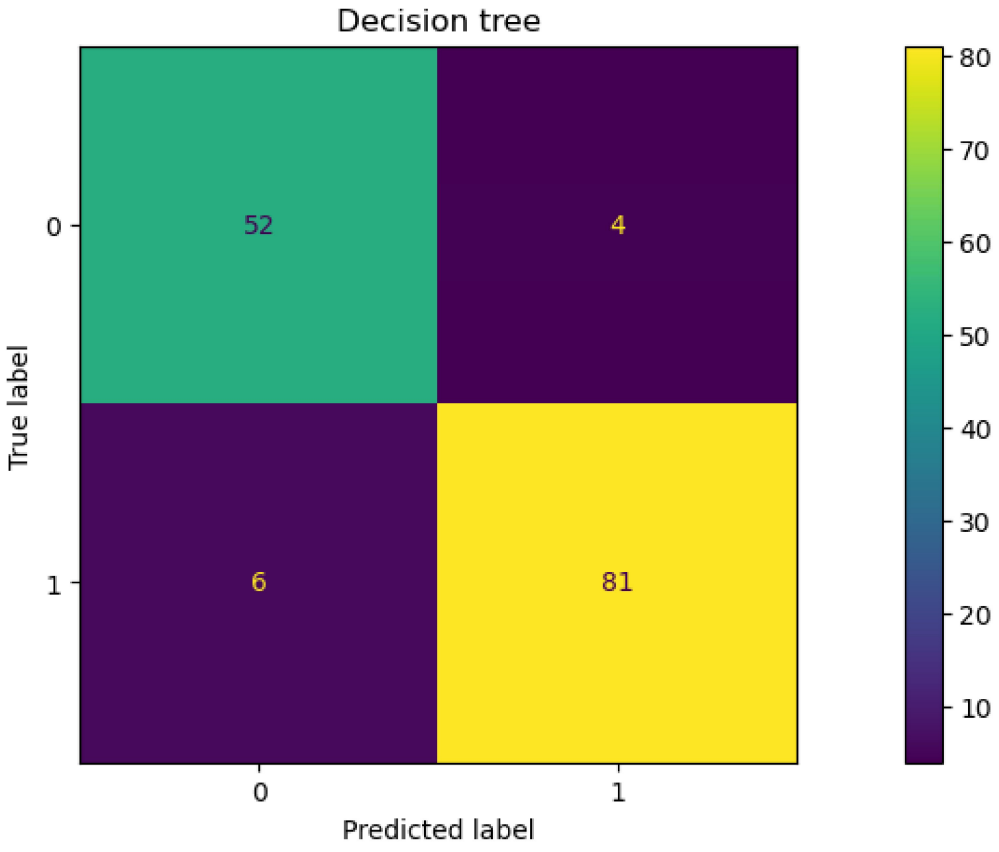
1 Decision Tree of Hold Out

[66]:
```
deci_tree =DecisionTreeClassifier(criterion ="entropy") # By default gini
deci_tree.fit(X_train,y_train)
prediction=deci_tree.predict(X_test)
accuracy_hold=accuracy_score(y_test,prediction)
a1= accuracy_hold*100
print( "Hold out ",a1)
```

```
Hold out  93.00699300699301
```

[67]:
```
fig,(ax1)=plt.subplots()
fig.set_size_inches(15,5)
ax1.set_title("Decision tree")
ConfusionMatrixDisplay.from_estimator(deci_tree,X_test,y_test,ax=ax1)
```

[67]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x269d3d1db80>



2 KNN

[68]:
```
#KNN (Nearst Neighbour)
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
print("Accuracy ")
a2 = knn.score(X_test,y_test)*100
print(a2)
```

```
Accuracy
95.8041958041958
```

```
C:\Users\CSLab\anaconda3\lib\site-
packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will
change: the default value of `keepdims` will become False, the `axis` over which
the statistic is taken will be eliminated, and the value None will no longer be
accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

3 Naive bayes

[69]:
```
from sklearn.naive_bayes import GaussianNB
#fitting the model
nb=GaussianNB()
nb.fit(X_train,y_train)
prediction_nb =nb.predict(X_test)

print("Accuracy of Test Data")
nb_score=accuracy_score(y_test,prediction_nb)

a3 = nb_score*100
print("Naive Bayes Accuracy :",a3,"%")
print("Classification report on test data")
print(classification_report(y_test,prediction_nb))
```

```
Accuracy of Test Data
Naive Bayes Accuracy : 94.4055944055944 %
Classification report on test data
              precision    recall  f1-score   support

           0       0.94      0.91      0.93        56
           1       0.94      0.97      0.95        87

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

```
weighted avg       0.94      0.94      0.94       143
```

[70]:
```
import numpy as np
k=20
dta = []
for i in range(k):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.3)
    deci_tree.fit(x_train, y_train)
    dta.append(deci_tree.score(x_test, y_test))
a4 = np.mean(dta)*100
print("Decision tree accuracy: " ,a4,"%")
```

```
Decision tree accuracy:  92.39766081871343 %
```

4 Cross validation

[71]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(deci_tree,svm.data,svm.target,cv=5)
```

[72]:
```
mean_accuracy =scores.mean()
std_accuracy=scores.std()
```

[73]:
```
a5 = mean_accuracy*100
print(a5)
```

```
93.85033379909953
```

[77]:
```
print("Accurcay of Hold Out ",a1)
print("Accurcay of KNN ",a2)
print("Accurcay of Naive Bayes ",a3)
print("Accurcay of Cross Validation ",a4)
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
Accurcay of Hold Out  93.00699300699301
Accurcay of KNN  95.8041958041958
Accurcay of Naive Bayes  94.4055944055944
Accurcay of Cross Validation  92.39766081871343
X_train shape: (18, 30)
y_train shape: (18,)
```

5 Random Sub-Sampling

[80]:
```
svm = load_breast_cancer()
x = svm.data
y = svm.target

subsample_size = 25
test_size = 0.25  # 25% of the data will be used for testing
indices = np.random.choice(len(x), size=subsample_size, replace=True)

x_subsample = x[indices]
y_subsample = y[indices]

# Split the subsampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_subsample, y_subsample,␣
 ↪test_size=test_size, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Decision Tree Classifier on the scaled training data
deci_tree = DecisionTreeClassifier(criterion="entropy")
deci_tree.fit(X_train_scaled, y_train)

# Predict on the scaled test data
y_pred = deci_tree.predict(X_test_scaled)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate classification report
report = classification_report(y_test, y_pred, target_names=iris.target_names)
print("Classification Report:")
print(report)
```

```
Accuracy: 90.0
```

Practical 6 (Iris)

[4]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from scipy.cluster.hierarchy import dendrogram, linkage
```

[5]:
```
iris = load_iris()
x = iris.data
y = iris.target
```

[6]:
```
# Normalize the data using Standardization
scaler = StandardScaler()
x_normalized = scaler.fit_transform(x)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=300)
y_kmeans = kmeans.fit_predict(x_normalized)

# Evaluate KMeans clustering using accuracy score (Note: KMeans is unsupervised)
kmeans_score = accuracy_score(y, y_kmeans)
print("Accuracy Score - KMeans:", kmeans_score * 100)
```

```
Accuracy Score - KMeans: 24.0
```

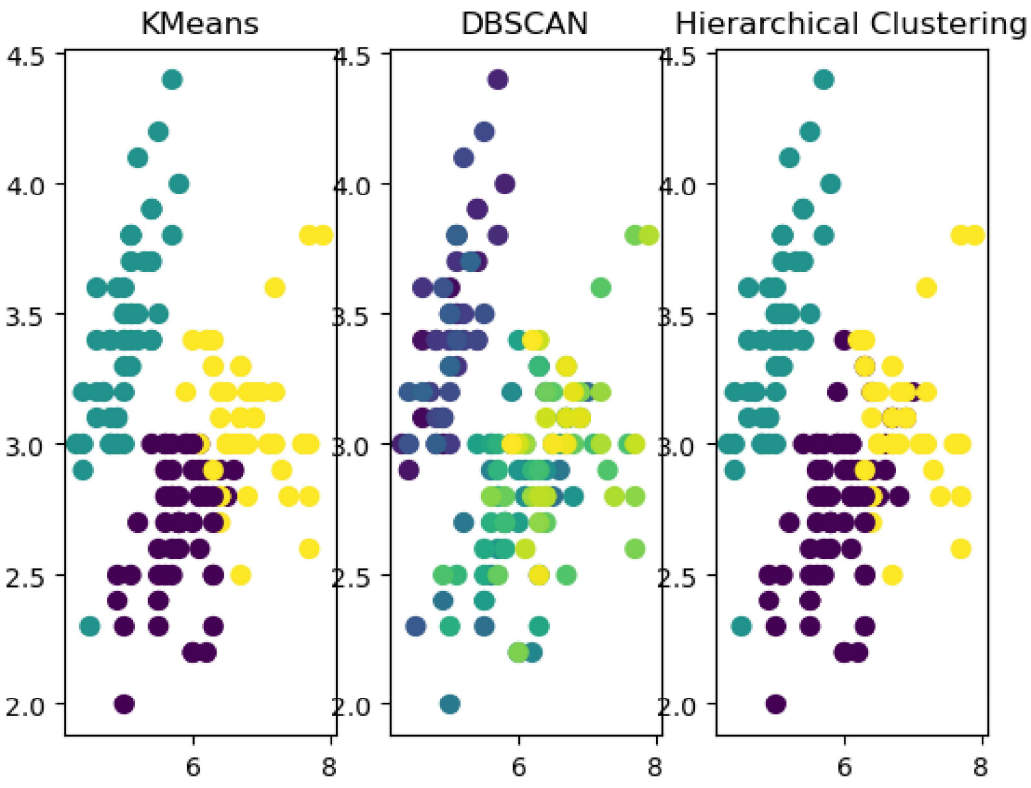
[7]:
```
# Normalize the data using Standardization
scaler = StandardScaler()
x_normalized = scaler.fit_transform(x)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.1, min_samples=1)
y_dbscan = dbscan.fit_predict(x_normalized)

#dbscan_score = accuracy_score(y, y_dbscan)

#print("Accuracy Score - DBSCAN:", dbscan_score*100)
```

[8]:
```
hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
y_hc = hc.fit_predict(x)
hc_score = accuracy_score(y, y_hc)
print("Accuracy Score - Hierarchical Clustering:", hc_score*100)
```

```
Accuracy Score - Hierarchical Clustering: 23.333333333333332
```

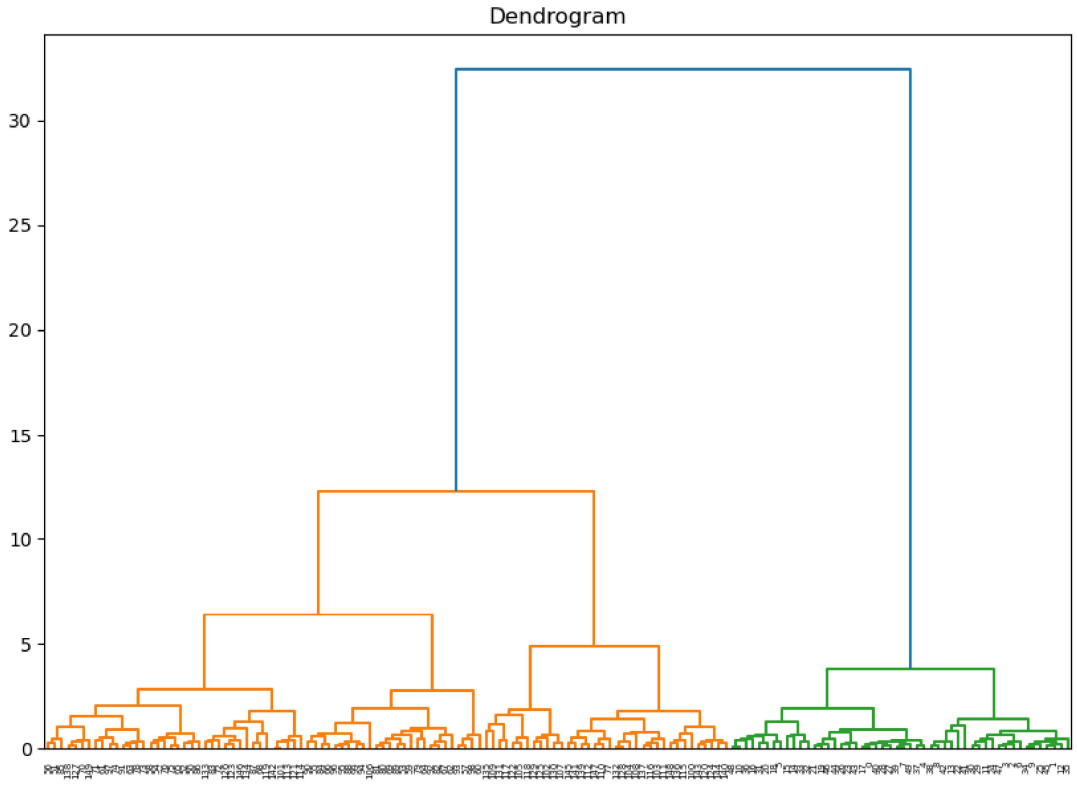
[9]:
```
plt.subplot(131)
plt.scatter(x[:, 0], x[:, 1], c=y_kmeans, s=50, cmap='viridis')
plt.title("KMeans")
plt.subplot(132)
plt.scatter(x[:, 0], x[:, 1], c=y_dbscan, s=50, cmap='viridis')
plt.title("DBSCAN")
plt.subplot(133)
plt.scatter(x[:, 0], x[:, 1], c=y_hc, s=50, cmap='viridis')
plt.title("Hierarchical Clustering")
```

[9]:
```
Text(0.5, 1.0, 'Hierarchical Clustering')
```



[10]:
```
)
```

```
linked = linkage(x, 'ward') plt.figure(figsize=(10, 7)dendrogram(linked,
  orientation='top', distance_sort='descending',␣ ↪show_leaf_counts=True)
plt.title("Dendrogram") plt.show()
```



Practical 6 (Breast Cancer)

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from scipy.cluster.hierarchy import dendrogram, linkage

brct = load_breast_cancer()
x = brct.data
y = brct.target


# Normalize the data using Standardization
scaler = StandardScaler()
x_normalized = scaler.fit_transform(x)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10,
max_iter=300)
y_kmeans = kmeans.fit_predict(x_normalized)

# Evaluate KMeans clustering using accuracy score (Note: KMeans is
unsupervised)
kmeans_score = accuracy_score(y, y_kmeans)
print("Accuracy Score - KMeans:", kmeans_score * 100)
```

```
Accuracy Score - KMeans: 67.66256590509666
```

```
# Normalize the data using Standardization
scaler = StandardScaler()
x_normalized = scaler.fit_transform(x)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.1, min_samples=1)
y_dbscan = dbscan.fit_predict(x_normalized)

#dbscan_score = accuracy_score(y, y_dbscan)

#print("Accuracy Score - DBSCAN:", dbscan_score*100)
```

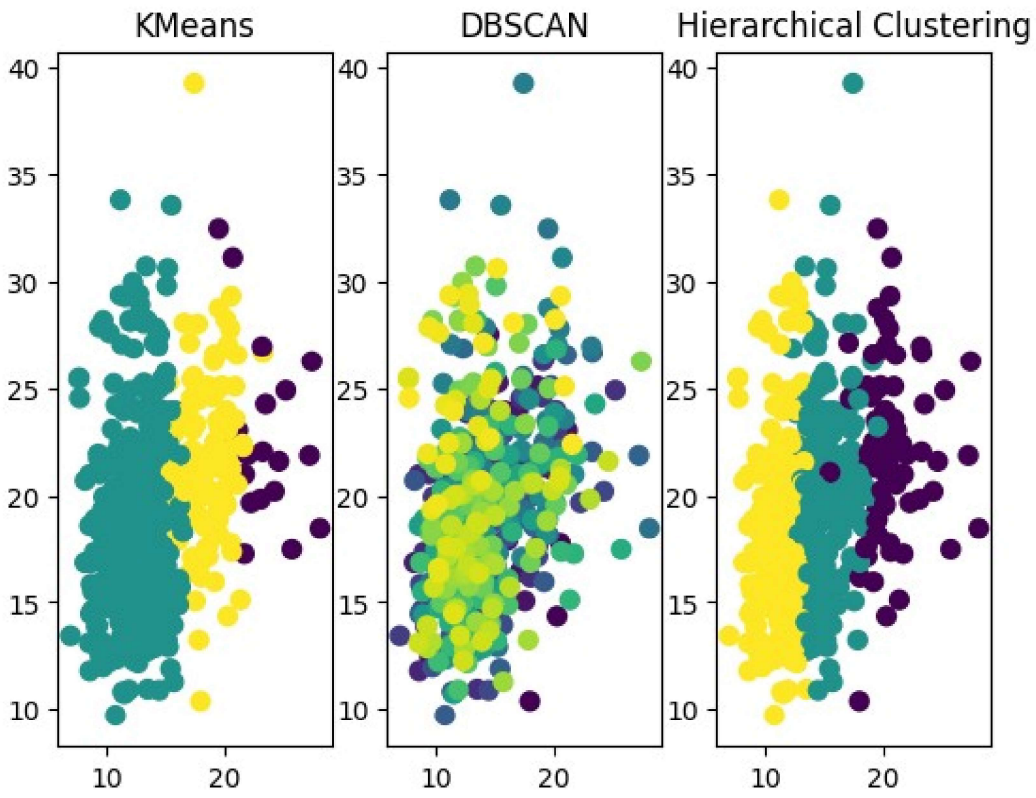
```
Accuracy Score - DBSCAN: 0.17574692442882248
```

```
hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
y_hc = hc.fit_predict(x)
hc_score = accuracy_score(y, y_hc)
print("Accuracy Score - Hierarchical Clustering:", hc_score*100)
```

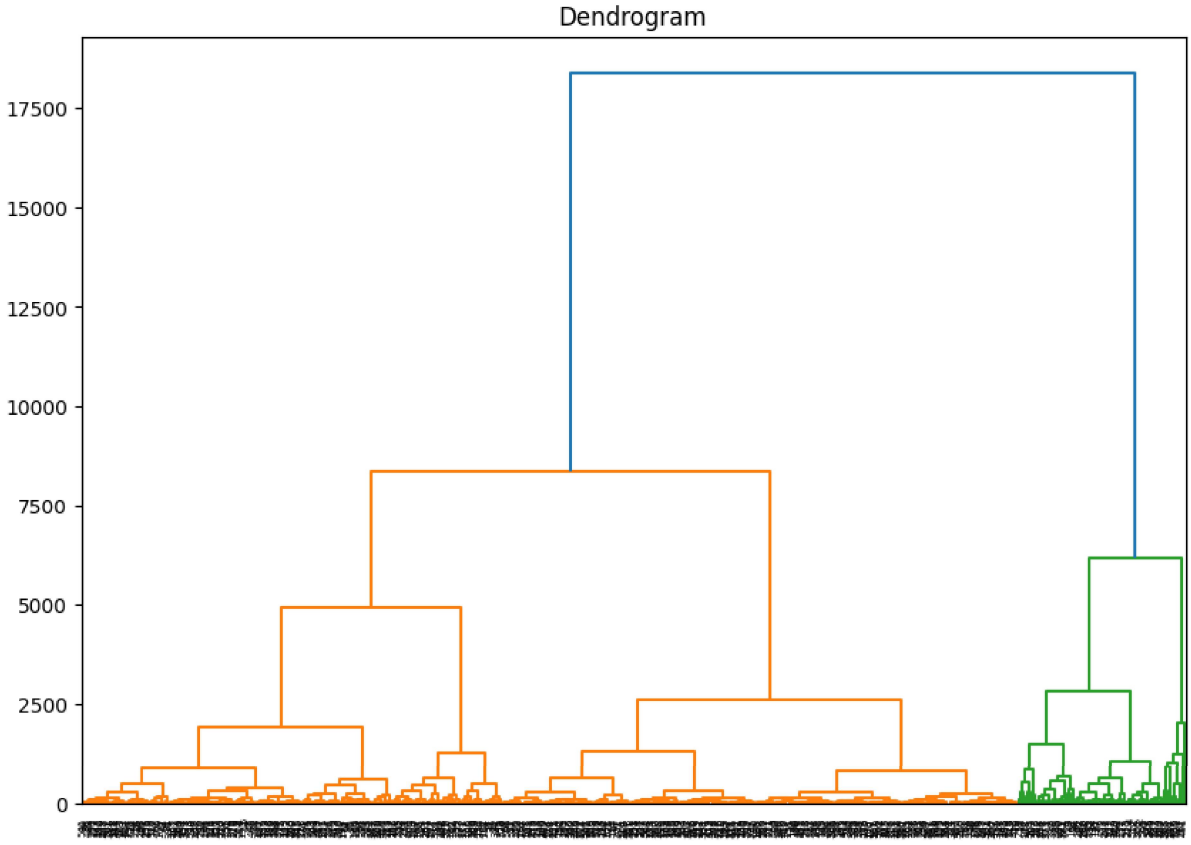
```
Accuracy Score - Hierarchical Clustering: 32.161687170474515
```

```
plt.subplot(131)
plt.scatter(x[:, 0], x[:, 1], c=y_kmeans, s=50, cmap='viridis')
plt.title("KMeans")
plt.subplot(132)
plt.scatter(x[:, 0], x[:, 1], c=y_dbscan, s=50, cmap='viridis')
plt.title("DBSCAN")
plt.subplot(133)
plt.scatter(x[:, 0], x[:, 1], c=y_hc, s=50, cmap='viridis')
plt.title("Hierarchical Clustering")
```

```
Text(0.5, 1.0, 'Hierarchical Clustering')
```




```
linked = linkage(x, 'ward')
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending',
show_leaf_counts=True)
plt.title("Dendrogram")
plt.show()
```


Dendrogram
17500
15000
12500
10000
7500
5000
2500
0