Classification using Machine Learning

1.1 Pre-processing of data

Data pre-processing is a prepare of planning the raw data and making it appropriate for a machine learning model. It is primary and significant step while making a machine learning model. (Java point, n.d.) Therefore, pre-processing of data should be fed to the model before starting it.

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

We have various built-in functions on python that can be used. Such as matplotlib, numpy, pandas, sklearn and so on.

1.1.1 Identifying targets and errors

```
Reading dataset from file and storing it in datatset
```

```
In [2]: dataset=pd.read_csv("C:\\Users/amoks/Desktop/AI year 2/diagnose3.csv")
```

```
Printing first five rows of dataset
```

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

Out[3]:

	ID	Gender	Location	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	0	Female	Dublin	208.558159	5.614409	-958.066369	-928.644832	1765.139560	660.084220	-1013.286854	182.046028	-84.581367	-34.812818
1	1	Male	NaN	241.445491	6.261318	1536.530602	-382.291477	385.692683	NaN	766.383635	85.844136	-84.024830	121.818738
2	2	Male	Belfast	-506.776499	1.383727	1262.104042	738.858091	96.283184	-371.541845	1166.170085	NaN	108.933924	705.209530
3	3	Female	Dublin	-176.557246	-1.240092	-888.517106	59.903715	-1705.316861	1100.794154	-101.064505	125.413522	-187.411209	134.079108
4	4	Female	Dublin	747.951286	2.703544	-1101.426351	-81.990991	1832.101533	19.414153	-555.453669	-0.630859	-324.037781	907.685125

```
Printing last five rows of dataset
```

```
In [4]: dataset.tail()
```

Out[4]:

	ID	Gender	Location	x1	x2	x3	x4	x5	x6	x7	x8	x9	
405	405	Female	London	739.946798	-7.072952	-1788.822609	340.636148	121.817510	860.411866	195.005920	-491.126971	-142.749249	-100.84
406	406	Female	Dublin	147.347048	-12.690624	14.227049	-1142.720102	388.694766	76.144489	-648.145196	-297.827954	-9.001211	542.39
407	407	Female	London	188.954247	-3.371112	-1410.785929	16.496005	241.326946	344.256263	-541.623107	-518.390866	-216.014594	-476.68
408	408	Male	Belfast	398.343793	3.361185	-1205.313593	-471.166844	292.835490	1257.348825	-287.324205	-340.728522	-172.582135	-135.38
409	409	Male	Dublin	-1300.536369	1.226898	3311.444617	201.655657	790.073430	434.378687	-1262.306009	50.823251	496.181174	-652.88

```
In [5]: dataset.shape
Out[5]: (410, 15)
```

```
In [6]: dataset.columns
Out[6]: Index(['ID', 'Gender', 'Location', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7',
               'x8', 'x9', 'x10', 'x11', 'Diagnosis'],
              dtype='object')
```

In [17]:

```python
#Specifying the features X and the target y
X = dataset.iloc[:, 3:16].values
y = dataset.iloc[:, :16].values
d = dataset.iloc[:, 1:3].values ## new variable for the categorical values
```

In [10]:

```python
print(dataset, "\n")
```

```
      ID Gender Location           x1          x2           x3           x4  \
0      0 Female   Dublin   208.558159    5.614409  -958.066369  -928.644832
1      1   Male      NaN   241.445491    6.261318  1536.530602  -382.291477
2      2   Male  Belfast  -506.776499    1.383727  1262.104042   738.858091
3      3 Female   Dublin  -176.557246   -1.240092  -888.517106    59.903715
4      4 Female   Dublin   747.951286    2.703544 -1101.426351   -81.990991
..   ...    ...      ...          ...         ...          ...          ...
405  405 Female   London   739.946798   -7.072952 -1788.822609   340.636148
406  406 Female   Dublin   147.347048  -12.690624    14.227049 -1142.720102
407  407 Female   London   188.954247   -3.371112 -1410.785929    16.496005
408  408   Male  Belfast   398.343793    3.361185 -1205.313593  -471.166844
409  409   Male   Dublin -1300.536369    1.226898  3311.444617   201.655657

              x5           x6           x7          x8          x9  \
0    1765.139560   660.084220 -1013.286854  182.046028  -84.581367
1     385.692683          NaN   766.383635   85.844136  -84.024830
2      96.283184  -371.541845  1166.170085         NaN  108.933924
3   -1705.316861  1100.794154  -101.064505  125.413522 -187.411209
4    1832.101533    19.414153  -555.453669   -0.630859 -324.037781
..           ...          ...          ...         ...         ...
405   121.817510   860.411866   195.005920 -491.126971 -142.749249
406   388.694766    76.144489  -648.145196 -297.827954   -9.001211
407   241.326946   344.256263  -541.623107 -518.390866 -216.014594
408   292.835490  1257.348825  -287.324205 -340.728522 -172.582135
409   790.073430   434.378687 -1262.306009   50.823251  496.181174

            x10          x11  Diagnosis
0    -34.812818  1092.858259          0
1    121.818738 -1274.207207          2
2    705.209530 -1244.345340          1
3    134.079108          NaN          0
4    907.685125  -890.175568          1
..          ...          ...        ...
405 -100.840214  -101.155865          0
406  542.396253  1825.777921          0
407 -476.687321  -914.462913          1
408 -135.382709 -4154.895083          2
409 -652.889320  3534.538388          2

[410 rows x 15 columns]
```

In [41]:

```python
#Printing the features
print(X, "\n")
```

```
[[ 2.08558159e+02  5.61440881e+00 -9.58066369e+02 ... -3.48128181e+01
   1.09285826e+03  0.00000000e+00]
 [ 2.41445491e+02  6.26131795e+00  1.53653060e+03 ...  1.21818738e+02
  -1.27420721e+03  2.00000000e+00]
 [-5.06776499e+02  1.38372706e+00  1.26210404e+03 ...  7.05209530e+02
  -1.24434534e+03  1.00000000e+00]
 ...
 [ 1.88954247e+02 -3.37111224e+00 -1.41078593e+03 ... -4.76687321e+02
  -9.14462913e+02  1.00000000e+00]
 [ 3.98343793e+02  3.36118492e+00 -1.20531359e+03 ... -1.35382709e+02
  -4.15489508e+03  2.00000000e+00]
 [-1.30053637e+03  1.22689796e+00  3.31144462e+03 ... -6.52889320e+02
   3.53453839e+03  2.00000000e+00]]
```

In [42]:

```python
#Printing the target
print(y)
```

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

In [13]:

```python
#Printing categorical Values
print(d)
```

```
[['Female' 'Dublin']
 ['Male' nan]
 ['Male' 'Belfast']
 ['Female' 'Dublin']
 ['Female' 'Dublin']
 ['Male' 'London']
 ['Male' 'Belfast']
 [nan 'Dublin']
 ['Female' 'Belfast']
 ['Male' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Female' 'Dublin']
 ['Female' 'Belfast']
 ['Male' 'Belfast']
 ['Male' 'London']
 ['Female' 'Dublin']
 ['Female' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Female' 'Dublin']
```

```
['Male' 'Dublin']
['Female' 'Belfast']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Male' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Male' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
```

```
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Female' 'Belfast']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'London']
['Female' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'Belfast']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'London']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
```

```
['Female' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'Dublin']
['Female' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Male' 'London']
['Female' 'Belfast']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
```

```
['Female' 'London']
['Female' 'London']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Male' 'London']
['Male' 'London']
['Male' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
```

```
['Male' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Male' 'London']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'Dublin']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'London']
['Female' 'Belfast']
['Female' 'Dublin']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
```

```
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Belfast']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Female' 'Belfast']
```

```
['Male' 'London']
['Male' 'London']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Female' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Dublin']]
```

In [14]:

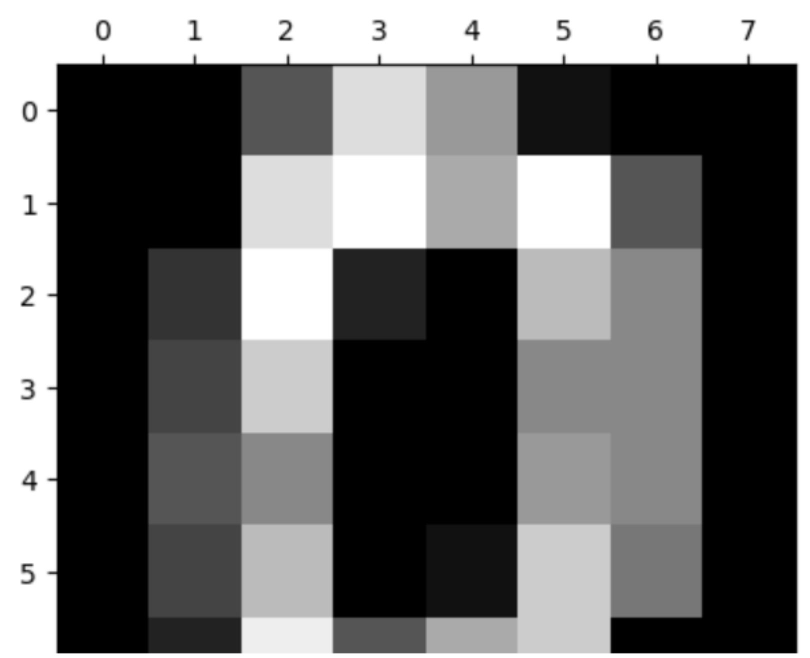
```
#calculates a summary of statistics of the DataFrame columns
display(dataset.describe())
```

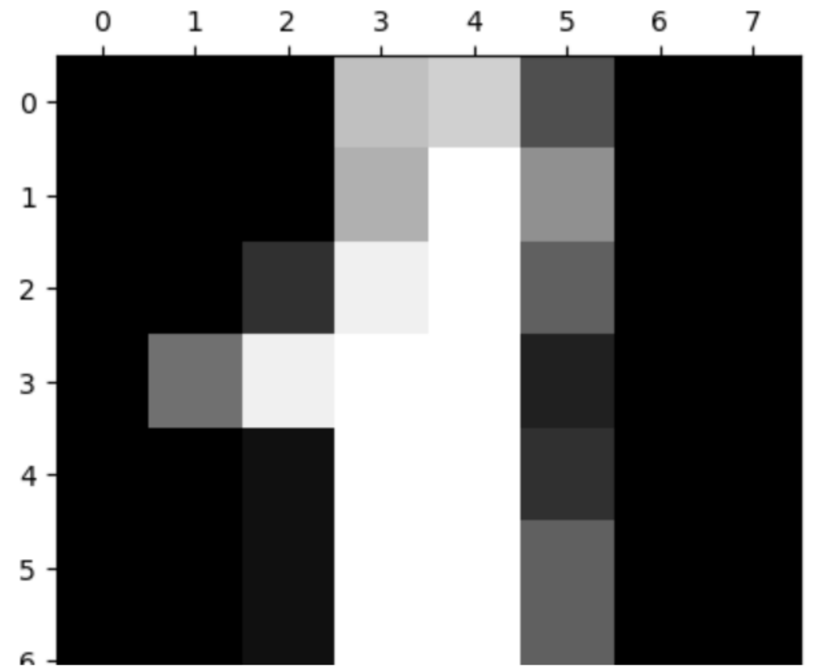
	ID	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
count	410.000000	410.000000	409.000000	410.000000	409.000000	409.000000	409.000000	410.000000	409.000000	409.000000	410.000000
mean	204.500000	88.925990	0.110245	-104.524582	-40.061227	37.869517	288.111241	-122.598297	13.790226	-34.747673	-4.434860
std	118.501055	477.629411	4.518285	1119.567466	600.765486	882.937429	697.456504	836.538704	383.632621	193.889521	618.780405
min	0.000000	-1633.896997	-12.690624	-3941.048835	-1902.722109	-2200.521551	-2187.381028	-2282.120520	-1001.612255	-810.204977	-1762.260397
25%	102.250000	-217.294480	-2.665523	-931.893557	-450.720304	-565.473342	-107.200851	-650.814637	-259.611176	-165.132187	-392.261722
50%	204.500000	103.140263	0.319521	-66.661881	-18.157122	93.393240	292.941433	-191.192316	31.806720	-63.847540	7.571258
75%	306.750000	388.940594	3.296338	702.320924	335.067705	631.662143	741.042405	432.097648	281.096453	103.668373	442.883058
max	409.000000	1662.416728	12.553529	4230.499279	2010.240619	2724.409306	1880.382323	2417.171819	980.078588	498.230841	1535.760498

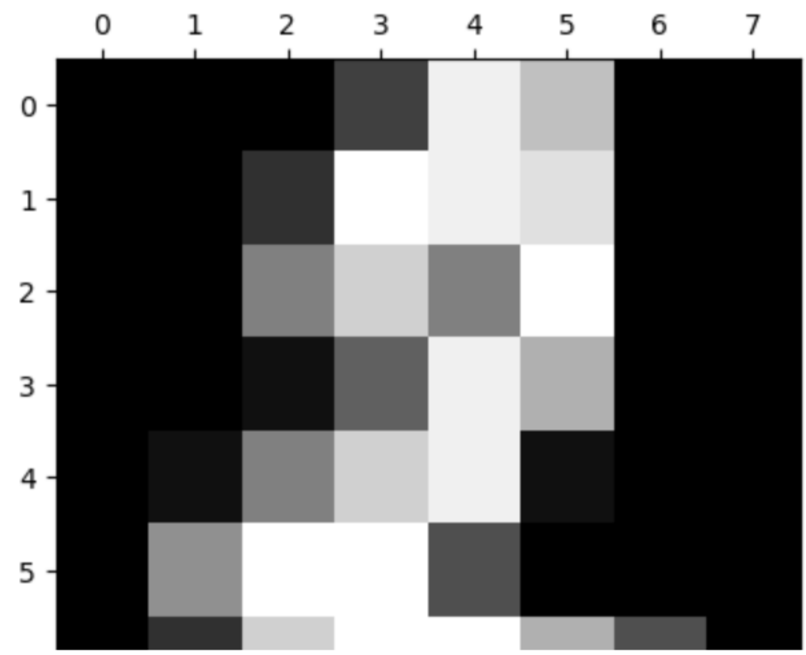
This is the visual representation of dataset in graph.

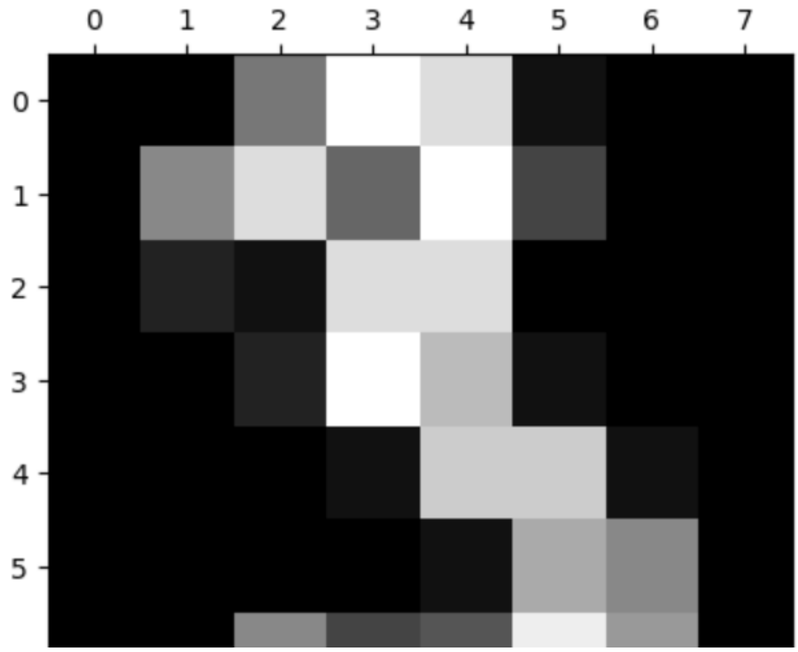
In [60]:

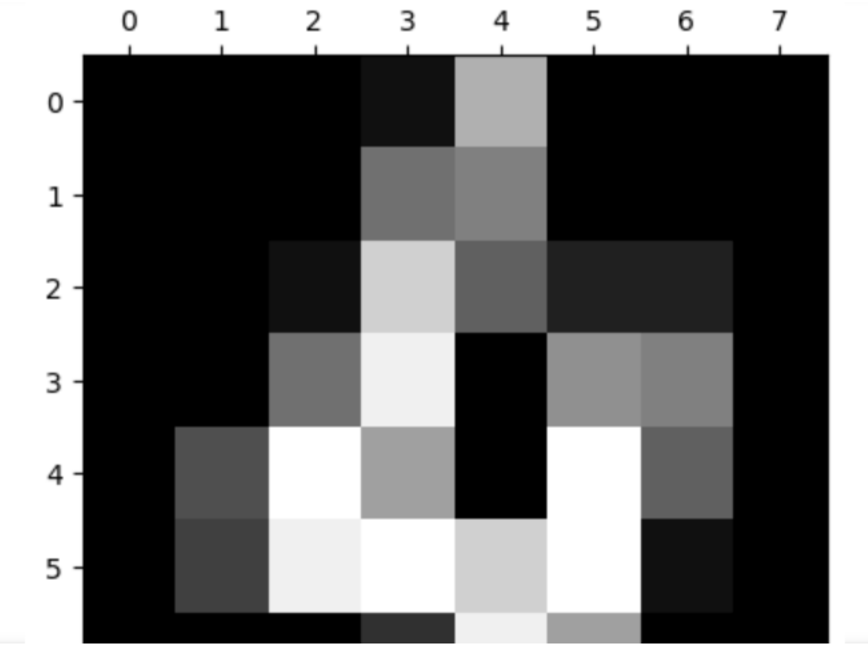
```
plt.gray()

for i in range(5):
    plt.matshow(dataset.images[i])
```






0 1 2 3 4 5 6 7
0 1 2 3 4 5


0 1 2 3 4 5 6 7
0 1 2 3 4 5



1.1.2 Missing values

Missing values should be managed during pre-processing of data as missing values may reduce the accuracy of data and model created at the end can be biased. So, we have used SimpleImputer class from Scikit-learn.

Numerical features

Numerical features are data type expressed in numerical values such as numbers rather than other descriptive language.

In [13]:

```
from sklearn.impute import SimpleImputer
#Importing Simple imputer from Sklearn
# Mean is used along each column to fill in the values for every numeric data feature.
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
#Specifying the columns for imputer
X[:,0:13] = imputer.fit_transform(X[:,0:13])
#Printing X to check if every feature have a value
print(X)
```

```
[[ 2.08558159e+02  5.61440881e+00 -9.58066369e+02 ... -8.45813672e+01
  -3.48128181e+01  1.09285826e+03]
 [ 2.41445491e+02  6.26131795e+00  1.53653060e+03 ... -8.40248299e+01
   1.21818738e+02 -1.27420721e+03]
 [-5.06776499e+02  1.38372706e+00  1.26210404e+03 ...  1.08933924e+02
   7.05209530e+02 -1.24434534e+03]
 ...
 [ 1.88954247e+02 -3.37111224e+00 -1.41078593e+03 ... -2.16014594e+02
  -4.76687321e+02 -9.14462913e+02]
 [ 3.98343793e+02  3.36118492e+00 -1.20531359e+03 ... -1.72582135e+02
  -1.35382709e+02 -4.15489508e+03]
 [-1.30053637e+03  1.22689796e+00  3.31144462e+03 ...  4.96181174e+02
  -6.52889320e+02  3.53453839e+03]]
```

Categorical features

Real world features are more categorical than numerical. It is taken on levels or values. Most of them are non-numerical. Thus, it should be converted to numerical values.

In [15]:
```
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
#Importing Simple imputer from Sklearn
# Mean is used along each column to fill in the values for every textual data feature.
d = imputer.fit_transform(d)# select first and second column of X
# Now, print first and second to column of X to see if succesful.
print(d)
```

```
[['Female' 'Dublin']
 ['Male' 'London']
 ['Male' 'Belfast']
 ['Female' 'Dublin']
 ['Female' 'Dublin']
 ['Male' 'London']
 ['Male' 'Belfast']
 ['Male' 'Dublin']
 ['Female' 'Belfast']
 ['Male' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Female' 'Dublin']
 ['Female' 'Belfast']
 ['Male' 'Belfast']
 ['Male' 'London']
 ['Female' 'Dublin']
 ['Female' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Female' 'Dublin']
 ['Male' 'Dublin']
 ['Female' 'Belfast']
 ['Female' 'London']
 ['Female' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Male' 'London']
 ['Female' 'London']
 ['Male' 'London']
 ['Female' 'London']
 ['Male' 'Belfast']
 ['Male' 'London']
 ['Female' 'Belfast']
 ['Male' 'Belfast']
```

```
['Female' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Male' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Male' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
```

```
['Male' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Female' 'Belfast']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'London']
['Female' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'Belfast']
['Female' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'London']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
```

```
['Male' 'Dublin']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'Dublin']
['Female' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Male' 'London']
['Female' 'Belfast']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
```

```
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'London']
['Female' 'London']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Male' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Belfast']
['Female' 'London']
```

```
['Male' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Male' 'London']
['Male' 'London']
['Male' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Male' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
```

```
['Male' 'London']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'Dublin']
['Female' 'Belfast']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'London']
['Female' 'London']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Female' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Male' 'Dublin']
['Female' 'London']
['Female' 'Belfast']
['Female' 'Dublin']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'Belfast']
['Male' 'Dublin']
['Male' 'London']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
```

```
['Male' 'London']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Belfast']
['Female' 'Belfast']
['Male' 'Dublin']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Male' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Female' 'Dublin']
['Female' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Female' 'London']
['Female' 'London']
['Male' 'London']
['Female' 'Belfast']
['Female' 'London']
['Male' 'London']
['Male' 'Belfast']
```

```
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Female' 'Belfast']
['Male' 'London']
['Male' 'London']
['Male' 'London']
['Male' 'Dublin']
['Male' 'Dublin']
['Male' 'Dublin']
['Female' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Dublin']
['Female' 'Dublin']
['Female' 'Belfast']
['Male' 'London']
['Male' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Male' 'Dublin']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Female' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Female' 'Belfast']
['Male' 'London']
['Female' 'Dublin']
['Male' 'London']
['Male' 'Belfast']
['Female' 'Belfast']
['Female' 'London']
['Female' 'London']
['Male' 'Dublin']
['Male' 'Belfast']
['Female' 'Belfast']
['Male' 'Belfast']
['Female' 'London']
['Female' 'Dublin']
['Female' 'London']
['Male' 'Belfast']
['Male' 'Dublin']]
```

1.1.3 Convert categorical data to numerical data

Most of the machine learning algorithms don’t work directly. They need all input and output variables to be numerical. This makes sense that categorical data should be converted to numerical data.

```
In [16]:
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
# d2 is used to pick up the first column of d
#(the '-1' just means that reshape should use the
# number of elements in the column as appropriate
d2 = d[:,0].reshape(-1,1)
# OneHotEncoder object fit to d2 and transform the data
d3 = encoder.fit_transform(d2).toarray()

# combine the new features with comluns of d
# axis=1 just means to concantenate them by column

d = np.concatenate((d3,d[:,1:2]),axis=1)

# When converting to dummy variables we can remove one of them,
# so remove the first one.

d=d[:, 0:]
print(d)
```

```
[[1.0 0.0 'Dublin']
 [0.0 1.0 'London']
 [0.0 1.0 'Belfast']
 ...
 [1.0 0.0 'London']
 [0.0 1.0 'Belfast']
 [0.0 1.0 'Dublin']]
```

Conversion of categorical data on second column

```
In [17]:
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
# d2 is used to pick up the first column of d
#(the '-1' just means that reshape should use the
# number of elements in the column as appropriate
d2 = d[:,2].reshape(-1,1)
# OneHotEncoder object fit to d2 and transform the data
d3 = encoder.fit_transform(d2).toarray()
# combine the new features with comluns of d
# axis=1 just means to concantenate them by column
d = np.concatenate((d3,d[:,0:2]),axis=1)
d=d[:, 0:]
print(d)
```

```
[[0.0 1.0 0.0 1.0 0.0]
 [0.0 0.0 1.0 0.0 1.0]
 [1.0 0.0 0.0 0.0 1.0]
 ...
 [0.0 0.0 1.0 1.0 0.0]
 [1.0 0.0 0.0 0.0 1.0]
 [0.0 1.0 0.0 0.0 1.0]]
```

1.1.4 Featuring scale

Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data pre-processing step. (atoti, 2022)

```
In [18]:
from sklearn.preprocessing import StandardScaler
#Creating a StandardScaler object
sc = StandardScaler()
#Scaling all of the features of X
X[:,0:13] = sc.fit_transform(X[:,0:13])
#Printing the results after scaling
print(X[1:, :])
```

```
[[ 0.31971616  1.36470605  1.46758475 ... -0.25477285  0.20428547
  -0.59658025]
 [-1.24873023  0.28254072  1.2221669  ...  0.74286288  1.14824482
  -0.57887314]
 [-0.55651428 -0.29959219 -0.70111931 ... -0.78930132  0.22412345
   0.        ]
 ...
 [ 0.20968237 -0.77239035 -1.16818085 ... -0.93718659 -0.76413123
  -0.38326355]
 [ 0.6486113   0.72126885 -0.9844283  ... -0.71263201 -0.21188104
  -2.30473475]
 [-2.91263456  0.24774588  3.05487856 ...  2.74500937 -1.04923605
   2.25484994]]
```

Checking the mean after Feature scaling to make sure is 0

```
In [21]:
print('Mean of x1: {:5.3f}\n'.format(np.mean(X[:,0])))
print('Mean of x2: {:5.3f}\n'.format(np.mean(X[:,1])))
print('Mean of x3: {:5.3f}\n'.format(np.mean(X[:,2])))
print('Mean of x4: {:5.3f}\n'.format(np.mean(X[:,3])))
print('Mean of x5: {:5.3f}\n'.format(np.mean(X[:,4])))
print('Mean of x6: {:5.3f}\n'.format(np.mean(X[:,5])))
print('Mean of x7: {:5.3f}\n'.format(np.mean(X[:,6])))
print('Mean of x8: {:5.3f}\n'.format(np.mean(X[:,7])))
print('Mean of x9: {:5.3f}\n'.format(np.mean(X[:,8])))
print('Mean of x10: {:5.3f}\n'.format(np.mean(X[:,9])))
print('Mean of x11: {:5.3f}\n'.format(np.mean(X[:,10])))
print (X)
```

```
Mean of x1: -0.000

Mean of x2: 0.000

Mean of x3: 0.000

Mean of x4: 0.000

Mean of x5: 0.000

Mean of x6: -0.000

Mean of x7: 0.000

Mean of x8: 0.000
```

```
Mean of x9: -0.000

Mean of x10: -0.000

Mean of x11: 0.000

[[ 0.2507767   1.22117973 -0.76331675 ... -0.25765026 -0.04915326
   0.80701281]
 [ 0.31971616  1.36470605  1.46758475 ... -0.25477285  0.20428547
  -0.59658025]
 [-1.24873023  0.28254072  1.2221669  ...  0.74286288  1.14824482
  -0.57887314]
 ...
 [ 0.20968237 -0.77239035 -1.16818085 ... -0.93718659 -0.76413123
  -0.38326355]
 [ 0.6486113   0.72126885 -0.9844283  ... -0.71263201 -0.21188104
  -2.30473475]
 [-2.91263456  0.24774588  3.05487856 ...  2.74500937 -1.04923605
   2.25484994]]
```

1.1.5 Feature Selection

Feature Selection is the method in which input variables of the model is reduced by using only the reliable data.

In [24]:

```python
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression
#Identifying the relevant features
#Through many trial it is determined that the best K is 10
select = SelectKBest(mutual_info_regression, k=10).fit(X, y)
X = select.transform(X)
print(X)
```


```
[[ 0.2507767   1.22117973 -0.76331675 ... -0.25765026 -0.04915326
   0.80701281]
 [ 0.31971616  1.36470605  1.46758475 ... -0.25477285  0.20428547
  -0.59658025]
 [-1.24873023  0.28254072  1.2221669  ...  0.74286288  1.14824482
  -0.57887314]
 ...
 [ 0.20968237 -0.77239035 -1.16818085 ... -0.93718659 -0.76413123
  -0.38326355]
 [ 0.6486113   0.72126885 -0.9844283  ... -0.71263201 -0.21188104
  -2.30473475]
 [-2.91263456  0.24774588  3.05487856 ...  2.74500937 -1.04923605
   2.25484994]]
```


1.1.6 Splitting the data

Data splitting is done to avoid overfitting. It is done when data is divided into two or more subsets. With two-part split one of them is used to test and evaluate the data and remaining is used to train the data. TRAIN_TEST_SPLIT is a function of Sklearn model selection used to split data into training set and testing set.

The RANDOM_STATE parameter is used to control the randomness involved in the model. Setting random_state a fixed value assure that the same sequence of random numbers is generated each time the code is run.

```
In [29]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1,random_state= 0)
         #printing out the test data
         print(X_test)
         print(y_test)
```

```
[[ 3.26293708e-01 -2.05009395e-02  6.05516745e-01 -1.18438733e+00
  -3.55800286e+00  1.15060619e+00  1.54929202e+00 -2.61348224e-01
   9.92982483e-02  2.27015620e+00]
 [ 3.90960976e-01 -5.09497480e-01 -1.21043514e+00  1.50683715e+00
  -8.21692435e-02 -2.06812699e+00 -1.15398014e+00 -3.53207874e-01
  -8.20523457e-01  1.29827598e+00]
 [ 6.03126013e-01 -4.36698981e-01  1.80479595e-03  1.39556605e+00
  -7.65070679e-01 -5.29095927e-02 -1.11270969e+00 -6.67283313e-01
   3.70072902e-01  4.32131345e-01]
 [-6.80774779e-01  6.10085113e-01  1.31796320e+00 -2.50134517e-02
  -1.05857010e-01 -1.51859336e-01 -1.54721036e+00 -4.44096441e-01
  -3.20010294e-01 -1.35692895e+00]
 [-1.26834807e+00  2.12518984e-01  1.00695826e+00  4.94966742e-01
   1.19865417e-01  6.35715769e-01 -1.36046927e+00  7.94378164e-01
   4.31547641e-01 -6.31743808e-01]
 [ 7.30353529e-01  1.54281773e+00 -7.57427918e-01 -2.63600447e-01
   1.49683647e+00 -7.41173268e-01 -3.99123234e-01 -8.67349623e-01
  -7.88450455e-01 -1.34507514e+00]
 [-1.60947383e+00  6.81641754e-01  2.62495144e+00  1.00767822e+00
  -2.01786320e-01 -3.13895876e-01  5.49100180e-01  2.71427761e+00
   2.05958714e-01  2.69064826e+00]
 [-4.54193611e-01 -3.07899146e-18  5.29193769e-01 -1.91213123e-01
  -3.11055047e-01 -3.52604604e-01  9.31740929e-01  6.86570961e-02
  -1.04455078e+00 -3.65918213e-01]
 [ 1.86564734e-01  5.10666629e-01 -8.39201889e-01 -9.42914758e-01
  -1.04844928e+00 -6.27922631e-01  4.35001514e-01 -9.59296588e-03
  -6.10658653e-01  4.44558909e-01]
 [-2.41567009e-02  2.73404102e-01 -7.12837855e-01  1.13435095e+00
   5.84481267e-01 -8.78096958e-01  1.42214269e+00 -7.15168162e-01
   1.17666232e-01 -7.99179632e-01]
 [ 1.71597860e+00  1.58112242e-01 -1.75940602e+00  9.33816246e-01
  -1.37338507e+00  1.09452760e+00  3.45799450e-01 -1.53480709e+00
  -1.08103358e+00  7.75029029e-01]
 [ 1.52934240e+00 -1.06317611e+00 -7.06557106e-01 -6.67617281e-01
   1.73868975e-01 -1.37732665e+00 -1.41322226e+00 -7.35673915e-01
   2.62198031e-01  2.16832336e-01]
 [ 1.74140453e-01  2.08854349e-01 -6.94668349e-01  2.11313477e+00
   1.14986563e+00  4.28855664e-01  8.74044345e-01 -9.67746356e-01
  -1.61854457e+00 -7.64072195e-01]
 [ 3.11455881e-01 -1.61643068e+00  1.15173576e+00 -1.93070848e+00
  -3.17110464e-01  1.70917684e+00 -6.15496161e-01 -3.67114098e-01
  -3.55113445e-01 -5.89989354e-01]
 [-2.87234932e-01 -5.99873579e-01 -3.63168190e-01  2.36903350e-01
  -7.25100198e-01 -1.16752827e+00 -8.82561124e-01  3.35959011e-01
  -2.73745323e-01  1.44611863e+00]
 [ 4.37421557e-01  8.48630349e-02  1.83322032e+00  6.74165804e-01
  -2.36393413e+00  1.45898942e+00 -1.97694186e+00 -9.39381602e-01
  -1.31243155e+00  4.05098385e-01]
```

```
[ 1.36469023e+00 -1.59369792e+00 -1.50625654e+00  9.53108272e-02
  8.22562560e-01  3.80128555e-01 -1.31936962e+00 -5.58389959e-01
 -1.55989323e-01  9.90011506e-02]
[ 2.35220565e-02 -8.05089938e-01  1.44731141e+00  7.64552481e-01
 -5.05041463e-02  1.75448516e+00 -4.11955028e-01  1.77644924e+00
 -6.21765164e-01  1.35742265e+00]
[ 6.19697279e-02 -1.84658363e+00  5.40006217e-02 -1.61165149e+00
 -1.37770850e+00 -8.87213140e-01 -7.12086879e-01 -2.48574439e-02
 -8.21139884e-01  6.43332550e-01]
[-6.95760047e-01  1.08339226e+00  9.83875130e-01  4.59085994e-01
  1.35709734e+00 -8.81943960e-01  5.57155357e-01  4.64856091e-01
 -1.30475500e-01 -8.97607112e-01]
[-1.39318022e+00 -1.54290288e-01  5.42839488e-01  6.86523384e-01
  2.17171944e-01  1.81944694e+00 -2.00733340e+00  4.42235792e-01
  1.28763454e+00 -3.08300194e-01]
[-1.60313747e+00 -7.70787964e-01  1.59346410e+00 -3.92126387e-01
  9.27938005e-01 -1.31936764e+00  1.41565714e+00  1.01897855e+00
  5.47656939e-01 -1.02502505e-02]
[-1.80495846e+00 -2.32105025e-01  1.47039959e+00  3.70063049e-01
  4.63212089e-01 -9.53583882e-01 -1.53236283e+00  8.08790848e-01
  1.62265162e-01  1.39668655e-01]
[ 1.83260027e-01 -3.53712209e-01  1.15080978e-01  3.21714173e-01
  2.67889094e-01  1.32917858e+00  7.07414070e-01  9.50789482e-01
  7.35102000e-01  2.84858132e-01]
[ 5.74654602e-01  1.52833265e+00 -6.15543160e-01  6.39712242e-01
 -3.67866590e-02 -3.26180716e-01  3.23642854e-01 -2.32906702e-01
 -1.50890787e+00  8.31975784e-01]
[-4.82672822e-01 -3.32944750e-01  7.47895683e-01 -1.84465699e+00
  2.01255431e+00 -3.52692828e-01 -1.08502378e+00  6.76670695e-01
  7.68798132e-03 -1.23254820e+00]
[ 3.86375425e-01 -2.25378490e+00 -7.32789741e-01 -8.67033870e-01
 -2.28442321e+00  1.18562424e+00  1.28689945e+00 -8.63179307e-01
 -1.50344142e+00  6.20093651e-01]
[ 6.27824153e-01 -1.22770478e-01  1.23999610e-01  8.48995130e-01
  6.07363357e-01  8.29413211e-01  1.18853950e+00 -3.71881005e-01
 -1.20594900e+00 -1.42052402e+00]
[-1.23229813e+00  5.33956394e-01  9.89085033e-01 -7.67107267e-01
 -6.00332397e-01  2.36314882e+00 -2.13847470e-01  2.69567828e-01
  2.22163930e+00 -2.74649265e-01]
[-4.41511429e-01 -1.13557242e+00  6.28484274e-01  9.34185258e-01
  9.56106359e-01  1.34713486e+00  6.83393666e-01  9.27133107e-01
 -8.60723699e-01  7.64371905e-01]
[-4.30433912e-01  1.57171712e+00  1.11128090e+00 -9.31526451e-01
 -2.49895275e-01 -7.95120402e-01 -1.33525012e+00  1.42070872e-02
  9.00836962e-01 -4.17567267e-01]
[ 9.42326493e-01  5.02747419e-01 -8.74420141e-01  2.18940565e-02
 -5.58866574e-02 -1.18655681e+00  9.39777039e-01 -4.21203993e-01
  4.13958850e-01  1.54828326e-01]
[-2.97689672e-01  1.71034921e-01  5.24782892e-01  7.50329948e-01
  4.03784362e-01  2.03351711e-01 -3.43726851e-01  1.31631085e+00
 -2.24854086e+00 -1.72066387e-01]
[-1.05897210e-01  1.35376213e+00  4.12586254e-01 -1.82046468e+00
 -3.18404881e-02  1.67017968e+00 -1.68282613e+00  8.30960041e-01
 -5.30951068e-01  4.36063170e-01]
[ 6.29258639e-01  8.08725411e-01  5.44481438e-01 -1.24027660e-01
 -5.27887601e-01  7.08055336e-01  3.09638002e-01 -8.77640661e-01
  4.60289938e-01 -9.25923505e-01]
[-2.86635058e-01  4.69750466e-01  1.12481449e+00 -8.41626808e-01
```

```
   1.35020840e-01  2.95283392e-01  8.88475786e-01  2.38319840e-01
  -1.09966345e+00 -4.26911007e-01]
 [-2.24462867e-01 -2.15390890e+00  6.91250059e-01 -6.01486306e-01
   5.35552160e-01  5.88853590e-01 -1.09778327e+00  8.50089439e-01
  -5.25049317e-01  3.18419773e-01]
 [-1.39191501e+00  1.68288151e-01  1.16156832e+00  5.63108152e-01
  -1.60036160e-01 -3.24952833e-01  2.52495165e+00  9.26525028e-01
   1.30952303e+00  1.22461479e-01]
 [-7.25114754e-01  1.83338447e+00  5.86039768e-01 -5.17629512e-01
   1.04356745e+00  4.64692605e-02 -3.23222412e-01  1.73776507e+00
   4.82718420e-01 -2.37191001e-01]
 [-2.79196892e-01 -1.12285981e+00 -5.50512086e-01 -8.81121845e-02
  -5.20122340e-02 -8.65787237e-01  7.73117970e-01 -4.93142138e-01
  -1.15364790e+00 -5.54113993e-02]
 [ 4.99395769e-01  9.02814146e-01  6.32732411e-01 -1.19881638e+00
   1.15636108e+00  9.22109040e-01 -8.18824848e-01  1.80742706e+00
  -1.02429187e+00  1.15803882e+00]]
[1 0 0 2 1 0 0 2 1 0 1 0 0 2 1 0 0 1 1 2 1 2 2 1 0 2 1 2 1 1 2 0 1 1 2
2 1
 2 1 1 0]
```

10-fold cross validation

The method has one data set which is divided randomly into 10 parts; 9 of those parts are used for training and one tenth is reserved for testing. This procedure is repeated for 10 times each time reserving a different tenth for testing.

Naive Bayes is a simple and powerful algorithm for predictive modelling.

In [34]:
```
# to create a Naive Bayes Classifier
nb = GaussianNB()
# applying k-Fold Cross Validation
cv = KFold(n_splits=10, shuffle=True, random_state = 0)
for kfs in range(1,11):
    # identifing the relevant features based on the training data
    select = SelectKBest(score_func=f_classif, k=kfs).fit(X_train, y_train)
    # transform the training and test inputs
    X_trainFS = select.transform(X_train)
    X_testFS = select.transform(X_test)
    scores = cross_validate(estimator = nb, X = X_trainFS, y = y_train, cv = cv)
    print(scores["test_score"].mean())
```

```
0.45022522522522523
0.4581831831831832
0.5015015015015016
0.5121621621621621
0.49857357357357357
0.4823573573573573
0.4743243243243243
0.4825075075075075
0.4904654654654655
0.48228228228228226
```

1.1.7 Logistic Regression

Logistic Regression is mostly utilised for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. (IBM, n.d.)

```
In [35]:  ##Logistic Regression#
          lr = LogisticRegression()
          lr.fit(X_trainFS, y_train)
          y_pred = lr.predict(X_testFS)
```

Confusion Matrix

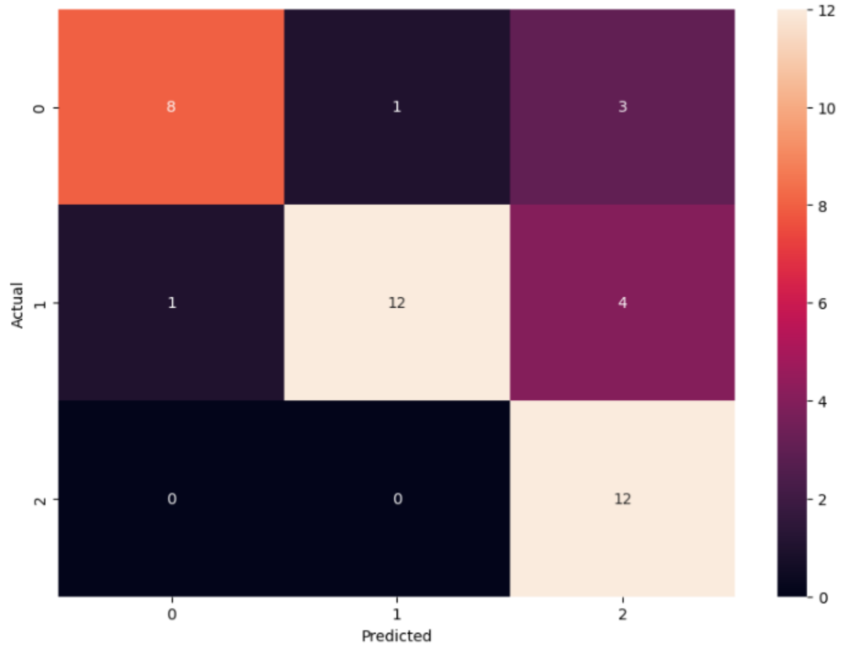
Confusion matrix shows the way in which the model is confused while making predictions. It highlights the errors made by classifier and types of errors being made.

```
In [36]:  ##confusion matrix
          from sklearn.metrics import confusion_matrix
          cm = confusion_matrix(y_test, y_pred)
          print(cm)

          [[ 8  1  3]
           [ 1 12  4]
           [ 0  0 12]]
```

A confusion matrix gives a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes. A table of all the predicted and actual values of a classifier is plotted.

```
In [50]:  import seaborn as sn
```

```
In [51]:  plt.figure(figsize=(10,7))
          sn.heatmap(cm, annot=True)
          plt.xlabel('Predicted')
          plt.ylabel('Actual')

Out[51]:  Text(95.72222222222221, 0.5, 'Actual')
```



Performance Metrics

Performance metrics are inevitable part of machine learning. They show if we are making progress and put a number on it. All machine learning algorithms require a metric to judge performance. (Bajaj, 2023) Performance metrics include: accuracy, precision, recall and F1-score.

Model accuracy is the measurement utilised to ensure which model is best to identify relationships and patterns between variables in a dataset based on the input, training, and data.

Model precision represents the ability of model to predict most positive from all the positive predictions made.

Recall represents the capacity of model to predict positives out of actual positives accurately.

F1 score measures accuracy of the model by describing score as precision and calling score function.

In [37]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,f1_score
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))
print('Micro Precision: {:.2f}'.format(precision_score(y_test, y_pred,average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(y_test, y_pred,average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred,average='micro')))
print('Macro Precision: {:.2f}'.format(precision_score(y_test, y_pred,average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(y_test, y_pred,average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred,average='macro')))
print('Weighted Precision: {:.2f}'.format(precision_score(y_test, y_pred,average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(y_test, y_pred,average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(y_test, y_pred,average='weighted')))
```

```
Accuracy: 0.78

Micro Precision: 0.78
Micro Recall: 0.78
Micro F1-score: 0.78

Macro Precision: 0.81
Macro Recall: 0.79
Macro F1-score: 0.78

Weighted Precision: 0.83
Weighted Recall: 0.78
Weighted F1-score: 0.78
```

1.1.8 Tunning the model LR

Machine Learning have hyperparameters that must be set in to customize the model to the dataset. For this, Grid search is used which is great for spot-checking combinations.

```
In [40]:  from sklearn.datasets import make_blobs
          from sklearn.model_selection import RepeatedStratifiedKFold
          from sklearn.model_selection import GridSearchCV
          from sklearn.linear_model import LogisticRegression
          X, y = make_blobs(n_samples=1000, centers=2, n_features=13, cluster_std=20)
          model = LogisticRegression()
          solvers = ['newton-cg', 'lbfgs', 'liblinear']
          penalty = ['l2']
          c_values = [100, 10, 1.0, 0.1, 0.01]
          grid = dict(solver=solvers,penalty=penalty,C=c_values)
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
          grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,scoring='accuracy',error_score=0)
          grid_result = grid_search.fit(X, y)
          print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
          means = grid_result.cv_results_['mean_test_score']
          stds = grid_result.cv_results_['std_test_score']
          params = grid_result.cv_results_['params']
          for mean, stdev, param in zip(means, stds, params):
              print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.712667 using {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
0.712333 (0.046882) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.712333 (0.046882) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.712333 (0.046882) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.712333 (0.046882) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.712333 (0.046882) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.712333 (0.046882) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.712333 (0.046882) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.712333 (0.046882) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.712000 (0.046361) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.712333 (0.046882) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.712333 (0.046882) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.712000 (0.047286) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.712333 (0.046882) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.712333 (0.046882) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.712667 (0.046614) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

Recursive Feature Elimination

Recursive Feature Elimination is effective for dataset which is more or mostly based on predictable values.

```
In [43]:  from sklearn.linear_model import LogisticRegression
          from numpy import mean
          from numpy import std
          from sklearn.datasets import make_classification
          from sklearn.model_selection import cross_val_score
          from sklearn.model_selection import RepeatedStratifiedKFold
          from sklearn.feature_selection import RFE
          from sklearn.pipeline import Pipeline
          # define dataset
          X, y= make_classification(n_samples=1000, n_features=13, n_informative=5,n_redundant=5, random_state=0)
          rfe = RFE(estimator=LogisticRegression(), n_features_to_select=13)
          model = LogisticRegression(C= 100, penalty= 'l2', solver= 'newton-cg')
          pipeline= Pipeline(steps=[('s',rfe),('m',model)])
          # evaluate model
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
          n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv,n_jobs=-1, error_score='raise')
          # report performance
          print('Accuracy score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

          Accuracy score: 0.913 (0.027)
```

After performing a grid search to determine the tunning parameters for Logistic Regression, a new approach was implemented to simplify the code and allow the program to automatically select the best path for achieving the highest score on the dataset. Despite using the same steps as before, this updated version of the code resulted in better performance metrics for the model during training.

```
In [44]:  X, y= make_classification(n_samples=1000, n_features=13, n_informative=5,n_redundant=5, random_state=0)
          # create pipeline
          rfe = RFE(estimator=LogisticRegression(), n_features_to_select=13)
          model = LogisticRegression(C= 100, penalty= 'l2', solver= 'newton-cg')
          pipeline= Pipeline(steps=[('s',rfe),('m',model)])
          # evaluate model
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
          n_scores = cross_val_score(pipeline, X, y, scoring='precision', cv=cv,n_jobs=-1, error_score='raise')
          # report performance
          print('Precision score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

          Precision score: 0.930 (0.036)
```

```
In [46]:  X, y= make_classification(n_samples=1000, n_features=13, n_informative=5,n_redundant=5, random_state=0)
          # create pipeline
          rfe = RFE(estimator=LogisticRegression(), n_features_to_select=13)
          model = LogisticRegression(C= 100, penalty= 'l2', solver= 'newton-cg')
          pipeline= Pipeline(steps=[('s',rfe),('m',model)])
          # evaluate model
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
          n_scores = cross_val_score(pipeline, X, y, scoring='recall', cv=cv, n_jobs=-1,error_score='raise')
          # report performance
          print('Recall score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

          Recall score: 0.894 (0.038)
```

```
In [47]:  X, y= make_classification(n_samples=1000, n_features=13, n_informative=5,n_redundant=5, random_state=0)
          # create pipeline
          rfe = RFE(estimator=LogisticRegression(), n_features_to_select=13)
          model = LogisticRegression(C= 100, penalty= 'l2', solver= 'newton-cg')
          pipeline= Pipeline(steps=[('s',rfe),('m',model)])
          # evaluate model
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
          n_scores = cross_val_score(pipeline, X, y, scoring='f1', cv=cv, n_jobs=-1,error_score='raise')
          # report performance
          print('F1-score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

          F1-score: 0.911 (0.028)
```

1.1.9 Logistic Regression Scores

Accuracy score: 0.913 (0.027)
Precision score: 0.930 (0.036)
Recall score: 0.894 (0.038)
F1-score: 0.911 (0.028)

According to the scores we got, the performance metrics is more when Recursive Feature Elimination is used which is 20% greater than the previous training set.

1.1.10 Support Vector Machine

Support Vector Machine is a supervised machine learning model which utilise classification algorithm for two classified group problems. (Stecanella, 2017)

In [66]:

```python
from scipy.stats import randint
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV

# Setup the parameters and distributions
param_dist = {
    "C": [0.1, 1, 10, 100],
    "kernel": ["linear", "poly", "rbf", "sigmoid"],
    "degree": [2, 3, 4, 5],
    "gamma": ["scale", "auto"],
    "coef0": [-1, 0, 1],
}

# Instantiate a Support Vector Machine classifier: svm
svm = SVC()

# Instantiate the RandomizedSearchCV object: svm_cv
svm_cv = RandomizedSearchCV(svm, param_dist, cv=5)

# Fit it to the data
svm_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned SVM Parameters: {}".format(svm_cv.best_params_))
print("Best score is {}".format(svm_cv.best_score_))
```

```
Tuned SVM Parameters: {'kernel': 'rbf', 'gamma': 'scale', 'degree': 4, 'coef0': 1, 'C': 1}
Best score is 0.966
```

In [67]:

```python
# evaluate RFE for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline

# define dataset
X, y = make_classification(n_samples=1000, n_features=13, n_informative=5,
                           n_redundant=5, random_state=0)

# create pipeline
rfe = RFE(estimator=SVC(kernel='linear'), n_features_to_select=13)
model = SVC(kernel='linear', C=1.0)
pipeline = Pipeline(steps=[('s',rfe),('m',model)])

# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv,
                           n_jobs=-1, error_score='raise')

# report performance
print('Accuracy score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy score: 0.913 (0.027)
```

In [68]:

```python
# evaluate RFE for classification using SVM
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline

# define dataset
X, y = make_classification(n_samples=1000, n_features=13, n_informative=5,
                           n_redundant=5, random_state=0)

# create pipeline
rfe = RFE(estimator=SVC(kernel='linear'), n_features_to_select=13)
model = SVC(kernel='linear', C=1.0)
pipeline = Pipeline(steps=[('s', rfe), ('m', model)])

# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
n_scores = cross_val_score(pipeline, X, y, scoring='precision', cv=cv,
                           n_jobs=-1, error_score='raise')

# report performance
print('Precision score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Precision score: 0.942 (0.035)
```

In [69]:

```python
# evaluate RFE for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline

# define dataset
X, y = make_classification(n_samples=1000, n_features=13, n_informative=5, n_redundant=5, random_state=0)

# create pipeline
rfe = RFE(estimator=SVC(kernel='linear'), n_features_to_select=13)
model = SVC(kernel='linear')
pipeline = Pipeline(steps=[('s', rfe), ('m', model)])

# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
n_scores = cross_val_score(pipeline, X, y, scoring='recall', cv=cv, n_jobs=-1, error_score='raise')

# report performance
print('Recall score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Recall score: 0.882 (0.038)
```

```
In [71]:  from numpy import mean
          from numpy import std
          from sklearn.datasets import make_classification
          from sklearn.model_selection import cross_val_score
          from sklearn.model_selection import RepeatedStratifiedKFold
          from sklearn.feature_selection import RFE
          from sklearn.svm import SVC
          from sklearn.pipeline import Pipeline

          # define dataset
          X, y = make_classification(n_samples=1000, n_features=13, n_informative=5,
                                     n_redundant=5, random_state=0)

          # create pipeline
          rfe = RFE(estimator=SVC(kernel='linear'), n_features_to_select=5)
          model = SVC(kernel='rbf', C=1, gamma='scale')
          pipeline = Pipeline(steps=[('s',rfe),('m',model)])

          # evaluate model
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=0)
          n_scores = cross_val_score(pipeline, X, y, scoring='f1', cv=cv, n_jobs=-1, error_score='raise')

          # report performance
          print('F1-score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

          F1-score: 0.964 (0.016)
```

1.1.11 Support Vector Machine performance scores

Accuracy score: 0.913 (0.027)
Precision score: 0.942 (0.035)
Recall score: 0.882 (0.038)
F1-score: 0.964 (0.016)

1.2 Conclusion

On looking at the performance metrics of LR and SVM, we can compare which algorithm has more accuracy on terms of given dataset. As mentioned previously, before using the RFE on our data set, the accuracy, precision, recall and F1 score were at the highest were 75% on both algorithms. After finding the best hyperparameters for both training modules, we have left the program to select the best features using the 10 K cross-validation. As a result, we have learned that the highest performance metrics from our data set come after the Support Vector Machine.

Accuracy score of SVM is equal to LR which ensures that both algorithms have similar accuracy.
Precision score SVM (0.942) with over 0.12% higher than LR (0.930).
Recall score SVM [0.882(0.038)] is somehow similar to LR with [0.894(0.038)].
F1-score SVM (0.964) with over 0.53% higher than LR (0.911).

References

atoti, 2022. *When to perform a Feature Scaling?.* [Online]
Available at: https://www.atoti.io/articles/when-to-perform-a-feature-scaling/
[Accessed 10 April 2023].

Bajaj, A., 2023. *Performance Metrics in Machine Learning [Complete Guide].* [Online]
Available at: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide
[Accessed 10 April 2023].

IBM, n.d. *What is logistic regression?.* [Online]
Available at: https://www.ibm.com/topics/logistic-regression
[Accessed 10 April 2023].

Java point, n.d. *Data Preprocessing in Machine learning.* [Online]
Available at: https://www.javatpoint.com/data-preprocessing-machine-learning
[Accessed 10 May 2023].

Stecanella, B., 2017. *Support Vector Machines (SVM) Algorithm Explained.* [Online]
Available at: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/
[Accessed 10 April 2023].