Assignment 2 Part A

Name: David Fodor Student code: B00796884 Email: fodor-d@ulster.ac.uk

Preprocessing

Identifying target and features

In [18]:

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

#Importing the dataset
dataset = pd.read_csv('diagnose5.csv')
#Specifying the features X and the target y
X = dataset.iloc[:, 3:16].values
y = dataset.iloc[:, 16].values
#Printing the dataset
print(dataset)
display(dataset.describe())
```

```
      ID  Gender Location           x1           x2           x3          
x4  \
0      0    Male   Dublin  -467.074369  -184.726516  -967.684803   928.017
233   
1      1  Female   Dublin   232.526054  -524.912660   895.377400   335.754
492   
2      2  Female   London   962.877057   -45.760582          NaN   172.804
295   
3      3    Male   London  1274.585571 -1184.654089   170.258993   753.411
953   
4      4  Female  Belfast   875.147413  -169.150411   464.195000 -1381.642
828   
..   ...     ...      ...          ...          ...          ...          
...   
347  347  Female   London   -61.519302  1438.553254  1582.568494  -666.954
544   
348  348  Female   London   139.640327 -1558.417473   -31.728934  1265.082
189   
349  349    Male   Dublin  -510.964488  -980.392637  -384.941829   667.428
958   
350  350  Female  Belfast   -18.396341  1329.757179  1093.447103  -323.641
412   
351  351    Male  Belfast  -970.188522 -1553.758612  -104.805284   196.881
533   

              x5           x6          x7           x8         x9  \
0   -1755.085464  1183.399245  242.841665          NaN  12.970484   
1     -35.968013   477.712918  475.223520  -476.423253  -4.652800   
2    -114.265815   543.326998  486.859442   146.647894  21.044666   
3   -1200.044428  -834.133203  -74.697356  -813.854133  -5.547350   
4    -759.157438   191.039365  300.861131  1306.148683 -15.389974   
..           ...          ...         ...          ...        ...   
347   251.138763   668.830588 -221.592734  1000.853345 -43.069582   
348  1083.070615   365.231993   38.236796  1162.140572  -5.578543   
349   232.130521  -315.556450  408.684047   107.309024  -6.546914   
350  -378.872267  1161.644200   22.105181   206.040858  15.853619   
351  -707.513839 -1745.680850  611.220961   225.467628 -20.885691   

             x10         x11          x12         x13  Diagnosis  
0     183.512809    4.440652 -1062.242048 -234.028050          0  
1     242.128538  228.676505   741.701204  222.117581          0  
2    -780.394848  -22.356532 -1207.530122 -174.247358          1  
3    1003.927950  643.563754  2411.687986 -718.009049          2  
4   -2733.646933 -641.669941  -652.564539 -741.704458          1  
..           ...         ...          ...         ...        ...  
347  1130.265123 -682.039781 -1541.147604  335.762426          0  
348   239.376659  301.830584   534.739056  321.052466          0  
349  -410.464600 -285.847241  1477.733558  438.645820          0  
350 -1133.618964  520.329392 -1272.396845 -280.298474          1  
351 -4910.248566  310.533695   800.678931 -411.411782          2  

[352 rows x 17 columns]
```

	ID	x1	x2	x3	x4	x5	
count	352.000000	351.000000	352.000000	351.000000	352.000000	351.000000	3!
mean	175.500000	-77.311644	310.492082	18.181941	48.596349	-5.167915	-:
std	101.757883	847.965600	1392.940015	905.530758	957.229191	895.246034	9(
min	0.000000	-3050.818857	-4558.753586	-2452.149474	-2372.677643	-2704.774136	-33:
25%	87.750000	-639.727179	-636.880928	-608.147095	-588.971097	-637.326133	-6:
50%	175.500000	-98.234844	348.884873	67.024286	38.191051	-35.053724	-:
75%	263.250000	470.140709	1301.838996	658.112650	698.631330	530.149903	6:
max	351.000000	2365.867028	3939.210461	2453.807595	3286.966138	2576.343059	27!

Missing values

We need to substitute something for the missing values. For this we use the SimpleImputer class from Scikit learn.

In [19]:

```
from sklearn.impute import SimpleImputer
#Creating a SimpleImputer object, specifying how missing values are represented and our chosen strategy for filling them up
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
#Fitting to the data. Here we specify the relevant features from the matrix X.
X[:,0:1] = imputer.fit_transform(X[:,0:1])
X[:,2:3] = imputer.fit_transform(X[:,2:3])
X[:,4:6] = imputer.fit_transform(X[:,4:6])
X[:,7:8] = imputer.fit_transform(X[:,7:8])
X[:,9:10] = imputer.fit_transform(X[:,9:10])
X[:,12:13] = imputer.fit_transform(X[:,12:13])
#Printing X to check if it works as it should
print(X)
```

```
[[ -467.0743686   -184.7265156   -967.6848028  ...     4.44065212
  -1062.242048    -234.0280502 ]
 [  232.5260537   -524.9126604    895.3774003  ...   228.6765046
    741.7012044    222.117581  ]
 [  962.877057     -45.76058249    18.1819414  ...   -22.35653165
  -1207.530122    -174.2473578 ]
 ...
 [ -510.9644882   -980.3926374   -384.941829   ...  -285.8472407
   1477.733558     438.6458203 ]
 [  -18.39634102  1329.757179    1093.447103   ...   520.3293924
  -1272.396845    -280.2984742 ]
 [ -970.1885224  -1553.758612    -104.8052844  ...   310.5336951
    800.6789307   -411.4117824 ]]
```

Feature Scaling

Some features can have very different magnitudes and that can affect the results from machine learning algorithms. The following code scales all of the features.

In [20]:

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

```
[[-0.4609559  -0.35602648 -1.0918232  ...  0.08415121 -0.58975672
  -0.58522175]
 [ 0.36643195 -0.60059581  0.97147242 ...  0.41188176  0.70242476
   0.45364306]
 [ 1.23018721 -0.25611999  0.         ...  0.04498595 -0.69382793
  -0.44907212]
 ...
 [-0.51286289 -0.92805309 -0.44644967 ... -0.34011729  1.22965164
   0.94678287]
 [ 0.06967664  0.73277815  1.19082976 ...  0.83814526 -0.74029257
  -0.69060194]
 [-1.05596774 -1.3402619  -0.13620533 ...  0.53151964  0.74467106
  -0.98921053]]
```

It is a good idea to check the mean of each of the features after the scaling has been done. It should be 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('Mean of x12: {:5.3f}\n'.format(np.mean(X[:,11])))
print('Mean of x13: {:5.3f}\n'.format(np.mean(X[:,12])))
```

```
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

Mean of x12: 0.000

Mean of x13: -0.000
```

Feature Selection

At this point we can select the k best features. After this, we have to modify the features so that they only consist of these selected features.

In [22]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression

#Identifying the relevant features
#Through trial and error it was found that for the best accuracy, the value of k should
either be 9 or 12
#These two produced the exact same results and since selecting 9 features instead of 12
is less demanding, it is the best solution
select = SelectKBest(mutual_info_regression, k=9).fit(X, y)
#Now transforming the features
X = select.transform(X)
#Printing the features
print(X)
```

```
[[-0.4609559  -0.35602648 -1.96025464 ...  0.08415121 -0.58975672
  -0.58522175]
 [ 0.36643195 -0.60059581 -0.03450222 ...  0.41188176  0.70242476
   0.45364306]
 [ 1.23018721 -0.25611999 -0.12221128 ...  0.04498595 -0.69382793
  -0.44907212]
 ...
 [-0.51286289 -0.92805309  0.2658213  ... -0.34011729  1.22965164
   0.94678287]
 [ 0.06967664  0.73277815 -0.41862297 ...  0.83814526 -0.74029257
  -0.69060194]
 [-1.05596774 -1.3402619  -0.7867667  ...  0.53151964  0.74467106
  -0.98921053]]
```

Splitting

We split the data into training and test sets using the function train_test_split from the sklearn.model_selection module in Sci-kit learn (sklearn).

In [23]:

```
from sklearn.model_selection import train_test_split
#The test size is 20% of all the data, and it is selected at random
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
= 0)
#Printing out the test data
print(X_test)
print(y_test)
```

```
[[ 7.12663946e-01 -1.41863267e+00 -9.94935943e-19  0.00000000e+00
   2.70968169e-01  3.11514523e-01 -1.18150611e+00 -2.10415516e-01
  -2.99506035e-01]
 [-1.43380205e+00 -7.10685099e-02  9.00454064e-01  1.54234773e+00
  -2.65467106e+00  1.02702453e+00 -3.52544909e-01 -1.37662333e+00
   2.32570982e+00]
 [ 5.42891844e-01 -6.38827370e-01  1.57394346e+00 -6.50274688e-02
  -1.48503212e+00 -1.68224259e+00  8.90314589e-01 -1.72032712e-01
  -5.01098535e-01]
 [ 2.25252314e-01  3.74045901e-01  7.55758439e-01 -6.24614299e-01
  -9.38220919e-02  9.38729538e-01 -1.20751053e-01  3.16202148e-02
  -7.12272465e-02]
 [-3.57025318e-01  6.42058458e-01  1.20369992e+00 -1.04822714e+00
  -5.32717323e-01  1.69127228e+00  1.21256467e+00 -6.24825224e-01
   1.69218615e+00]
 [ 9.75449240e-01  1.34247533e+00  1.43486558e+00  1.27765086e+00
  -1.55387289e-01  4.08129112e-01 -2.81340986e-01 -1.87747420e+00
   3.49088919e-01]
 [ 1.29241803e+00 -1.18928131e+00 -1.07610012e+00  5.42947342e-02
   1.13048850e+00  3.21102025e-01  3.18121980e-01 -9.81928882e-01
  -4.46593280e-01]
 [-5.28719252e-01 -2.80308274e-01 -6.82628039e-01 -8.06440394e-01
   7.88485048e-02 -5.30170208e-01  1.36360291e+00  4.60488705e-01
  -1.60933784e-01]
 [ 1.84360805e-01  2.73534613e-01 -4.78676797e-01 -5.79062764e-01
   3.17083435e-01 -3.30096735e-01 -3.91189754e-04  1.53439645e-02
  -5.15957947e-01]
 [-2.14836622e-01 -1.54273444e+00  1.62275494e-01 -2.36487558e-01
  -5.70402099e-01  7.34125970e-01  7.00198851e-01  8.40724536e-01
   4.75209296e-01]
 [-8.87913963e-01  8.70896180e-01  2.79430389e-01  3.50305154e-01
  -1.07211935e+00  8.30108008e-01 -1.49162124e+00 -9.55868008e-01
   1.45407094e+00]
 [ 1.81039644e+00  4.84845735e-01  1.86003157e-02  1.08527439e+00
   8.16213161e-01  7.86991271e-01 -7.77607234e-01 -1.21272126e+00
  -6.27709337e-01]
 [-2.81647187e+00 -8.75101690e-01 -4.06663061e-01 -2.17598794e+00
  -2.67834277e-01 -3.52817870e+00 -1.87568787e+00  1.11436497e+00
  -1.44382250e-01]
 [-8.23129256e-01 -7.21448173e-02 -5.85906941e-02  1.18005896e+00
   2.86464525e-01 -4.15178500e-01  6.24964361e-01 -5.74574331e-01
  -4.87395662e-01]
 [-6.67291800e-01  9.06979628e-01 -9.67073149e-01  2.16723032e-01
   3.46957542e-01 -1.37447644e+00 -2.56475583e-01  1.87747927e-01
  -1.14286712e+00]
 [ 1.34090361e-02  9.61025308e-01 -5.95520185e-01 -5.67818223e-01
  -1.38261482e+00  7.60744941e-01  1.60155268e-01  7.43751055e-01
   6.63767905e-01]
 [-5.69470736e-01  5.38390214e-01  3.52241707e-01 -1.82760181e+00
   1.66226403e+00 -2.13012196e-01  3.42063477e-02 -3.23516514e-01
   6.23471426e-01]
 [-3.01827622e-01  7.10299771e-01 -9.20764986e-01 -4.21180669e-01
  -1.44777607e+00  6.67578361e-01 -1.55724463e+00  5.13921082e-01
   4.65721006e-01]
 [-1.05596774e+00 -1.34026190e+00 -7.86766700e-01 -1.79083771e+00
   1.07287776e-01 -2.63613082e+00  5.31519639e-01  7.44671059e-01
  -9.89210528e-01]
 [-2.00499974e-01  4.65045042e-02  1.23191818e+00 -2.60510206e-02
   1.52228172e+00 -2.35307105e-01 -2.02120324e+00 -2.53313865e-01
   1.42469065e+00]
 [-7.86170293e-01 -6.47345829e-01  1.69143213e+00 -6.28826293e-01
```

```
 -7.17137868e-01 -4.40275912e-01 -7.82462458e-02 -9.15380097e-01
  1.92534975e+00]
[-6.12232165e-01  9.08836569e-01  2.36372500e+00 -1.78916962e-01
 -1.36080492e+00 -2.12618376e-01 -6.71420719e-01 -1.26141140e+00
  2.19706089e+00]
[-3.79134638e-01 -2.43945917e-01  8.76472255e-01 -2.74036351e-01
  6.16145890e-01  2.10384779e-01  2.80953872e-01 -3.56072758e-01
  9.51858101e-01]
[ 1.69810256e+00 -1.33447987e+00 -5.40963795e-01 -1.08034839e-01
  3.25766669e-01  3.08711688e-01  1.02709866e+00 -6.28152551e-01
 -1.31652623e+00]
[ 2.13604403e+00 -6.54484549e-01  2.71833183e+00  1.47863636e+00
  1.06191496e+00  2.78615358e+00  6.25744556e-01 -9.19969796e-01
  4.16671846e-01]
[-1.43776020e-01  4.97118122e-01  1.19278568e+00 -7.79609716e-01
 -5.43641310e-01 -1.42168755e-01  7.97128416e-01 -3.46311522e-01
  9.22649302e-01]
[ 9.31283633e-01 -5.83184762e-01  2.32977519e-01  0.00000000e+00
 -1.83802184e-01  4.07120734e-01 -1.02494679e+00  1.68937262e-01
 -5.58945081e-01]
[-1.34125153e+00  2.15384097e-02  1.44107797e+00 -2.59970857e-01
  5.38063830e-01  7.45323277e-01  3.56382433e-01 -6.79322283e-02
  1.28040604e+00]
[-9.38221509e-01  1.41564114e+00  1.24480425e-01  1.80086504e-01
  1.94950669e+00  1.93652932e-01 -2.48919775e-01 -5.87140316e-01
  9.67007574e-01]
[ 1.15941018e-01  4.31914069e-01  2.92655442e-01 -4.56600585e-01
 -3.36283268e-01 -4.17475098e-01 -4.19632308e-01  8.54242110e-01
 -7.06630005e-01]
[ 6.66958718e-01 -4.66512613e-01 -1.40563526e+00 -2.79136414e-01
 -5.55945816e-01 -1.41528026e-01  1.00204681e+00 -6.56209902e-01
 -1.01202768e+00]
[-8.52834760e-01 -7.60993650e-01  1.91172982e+00 -7.10123265e-02
  2.39434813e-01  9.23990528e-01 -6.84927574e-01 -4.05271991e-02
  1.55979785e+00]
[ 1.15580486e-01  9.42940571e-03 -1.07238550e+00 -2.56136642e-01
  5.42982423e-01  2.22808733e-02  1.04946604e+00 -5.14024166e-01
  6.20345604e-02]
[-3.92371939e-01 -1.26403890e+00  2.89180417e+00 -8.76094612e-01
 -4.78157195e-01 -8.77679645e-01 -1.82620725e+00  8.13084247e-01
  4.81380456e-01]
[ 1.67947085e+00 -7.98183701e-01  1.48946490e+00  5.49147485e-01
 -4.78328929e-02  1.40608816e+00 -1.40218448e-01 -5.69402355e-01
  1.34318873e-01]
[ 1.28535550e-01  5.75128663e-01 -7.66189156e-02 -3.91544619e-01
 -4.89332392e-01 -9.85033607e-02 -1.05614951e+00  4.56886088e-02
 -1.58547368e-01]
[-9.25364762e-01 -3.75363286e-01  1.39776634e-01 -5.73480460e-02
 -1.09385851e+00 -7.51281476e-01  1.34509529e+00  5.12765776e-01
  5.03420851e-01]
[-1.24216280e+00 -5.41606475e-01  4.32285491e-01  2.65933387e-01
  1.71960703e-01 -1.27774509e+00  1.50329951e-01  9.49523237e-01
  6.52786585e-01]
[-7.40673072e-01 -1.68375959e+00 -4.91199430e-01 -8.21249335e-01
 -1.70125291e+00 -2.79072286e-02 -9.00745344e-01  1.46723313e+00
 -8.07928724e-01]
[-2.06047864e+00  3.55560163e-01  4.30196063e-01  1.25731223e+00
  1.56417792e-01 -3.75773373e-01 -2.73779563e-01 -2.92754901e-01
  3.92355775e-01]
[ 1.27448709e+00 -1.85407162e-01  4.27196256e-01 -8.55382899e-01
  3.45548922e-01  2.24616840e-01 -7.16297411e-01  2.63248636e+00
```

```
  -1.32147954e+00]
 [ 7.65190674e-01  2.00442437e-01 -7.51048400e-01  1.30783301e+00
  -4.22482641e-01 -1.99988567e-01 -1.13632956e+00 -8.16732909e-01
   2.05152761e-01]
 [ 4.78439539e-01 -4.79425645e-01 -1.15617002e+00  5.79216352e-01
   6.77151174e-01 -2.72669655e-01  1.31152693e+00 -5.35483824e-01
  -3.78651943e-01]
 [-5.37842611e-02  6.88900736e-01  1.44226160e-01  3.62431222e-01
  -1.10123568e+00  6.91276077e-01 -6.81064569e-01  1.82635848e+00
   2.92095530e-01]
 [-1.12382500e+00  1.66036626e+00 -1.65185438e-01  1.63242488e+00
   4.85807690e-01 -1.15758311e+00  2.06060161e+00  1.77059225e-01
  -5.32247905e-01]
 [-1.93973665e-01 -4.74265220e-01  1.08806566e+00 -4.34300197e-01
  -4.31807814e-01  1.12846490e+00  6.28984256e-01 -1.93360113e+00
   2.21751380e+00]
 [-4.85221205e-01  3.06257813e-01  7.19709058e-01 -1.06532314e+00
  -3.60923697e-01 -1.39890530e-01  5.28296929e-01 -8.82607563e-01
   1.02567325e+00]
 [-7.97056991e-01  9.98971083e-01 -1.05124600e+00 -1.39868599e+00
   7.22134379e-01  3.39536550e-01  3.02721138e-01  6.51305485e-02
  -3.79194941e-01]
 [ 1.62154124e+00 -7.28065503e-01  2.91297106e-02  5.17234129e-01
  -1.54368837e+00 -1.26147654e-01  4.45622732e-01 -2.77392741e-01
  -1.46691813e+00]
 [-1.87404694e-01 -2.00526810e-01  1.09357471e+00 -5.36188068e-01
  -7.14435427e-02  9.52750191e-01  3.90041105e-01  2.05709951e+00
   1.13267309e+00]
 [ 1.77243782e-01 -2.27113481e-01 -4.96659864e-01  5.56482487e-01
  -2.55158571e-01  2.33597359e-01 -4.50038511e-01 -3.13495877e-01
   5.98941007e-01]
 [-8.36002466e-01  8.95081920e-02  2.21231084e+00  1.79859575e-01
  -1.98100730e+00 -1.45672331e-01 -9.37700402e-01 -1.14957726e+00
   2.14841854e+00]
 [ 1.28065971e+00  6.03936347e-02 -2.18071494e+00  6.28731978e-01
   1.22777141e+00 -4.33183505e-01 -8.19199792e-02 -1.18477242e+00
  -1.90463907e+00]
 [ 3.45023139e-01  1.64063625e+00 -5.93128434e-02 -1.29123090e+00
  -1.21395801e+00  1.52283753e+00 -1.62298659e+00  7.12284540e-01
   4.57243931e-01]
 [-1.29961009e-01  1.95703522e+00  1.46052690e+00  9.75224687e-01
   2.16151643e+00 -6.66390328e-02 -1.42097324e+00 -1.40308446e+00
   1.09092315e+00]
 [-4.46537343e-01  1.12038221e+00  4.39373237e-01  2.53324636e-02
  -1.06542654e+00  1.03881019e+00 -6.90075214e-01 -1.04863627e+00
   1.41987990e+00]
 [ 1.85260129e-01 -2.50089084e-01  9.99293433e-01  1.05129810e-01
   4.85659430e-01  1.78544590e-01 -5.02122437e-01 -4.72977163e-01
   9.15237698e-02]
 [ 1.57143604e-01 -2.38349613e+00  5.76128791e-01 -1.87113235e+00
  -1.20864342e-01 -1.47540452e+00  1.19872880e+00  7.61193347e-01
  -8.43997944e-01]
 [-1.41064733e+00 -2.20671014e-01  1.97494565e+00  6.89949022e-01
  -2.93006693e+00  1.30518078e-01  4.42305732e-01 -1.43184668e+00
   2.77115099e+00]
 [ 1.14578870e+00 -6.63929006e-01  5.77492509e-01 -1.54693177e+00
  -6.29184874e-01  1.32808435e+00 -1.83235269e-01  1.88625070e+00
   2.84733170e-01]
 [-3.94684779e-01  1.27167598e-01 -1.55803424e+00  3.51755396e-01
  -1.75080511e+00  7.42778863e-02  9.55623517e-02  1.28001780e-01
  -2.21906730e-01]
```

```
 [-2.29736401e-01  2.39789149e+00  1.28940323e+00  8.39930533e-01
   1.56020247e+00  1.36514895e+00 -5.66646674e-01 -1.74093958e+00
   2.24541959e+00]
 [ 6.42147334e-01  4.85447287e-01  1.54158932e+00  5.54163973e-01
   2.77625883e-01  8.64455557e-01 -1.47637710e+00  2.58956259e-01
   2.48823958e-01]
 [-2.39738320e-01  7.51896584e-01  8.13587933e-02 -1.57669274e-01
  -7.29457917e-01  1.00275215e+00  2.35939828e+00 -5.97279939e-01
   1.32393007e+00]
 [ 3.04440078e-01  6.45651367e-01  6.27636650e-01  5.04247393e-01
   1.52120490e+00  2.53076986e-01  7.57275443e-01 -9.80367591e-01
   9.18568654e-02]
 [ 1.06921333e+00 -3.38129621e-01  1.06456157e+00 -9.69103690e-01
   1.59803170e+00 -9.49770568e-02 -2.66384103e+00  1.34614902e-01
   5.24448846e-01]
 [-1.18799718e+00 -8.24446429e-01 -6.48310469e-01  9.19854643e-01
  -6.95018342e-01 -8.54894962e-02 -1.98373340e-01  1.22916114e+00
  -2.42784875e-01]
 [ 1.05647483e+00 -2.67794465e-01 -8.69826270e-01 -1.33798996e-01
  -6.03739881e-02  5.42648532e-01  8.10660601e-01 -7.34210942e-01
  -9.06712845e-01]
 [ 1.63042643e+00 -7.94015133e-01 -4.79198823e-01 -9.82671048e-01
   2.58089232e-01  3.21826506e-01 -6.92687428e-01 -8.73741370e-01
  -1.13263175e+00]
 [ 2.70467944e-01 -4.11455498e-01  1.28023187e+00 -1.71873762e+00
   1.48204867e+00 -1.71942454e+00 -2.94553619e-01  1.36174908e+00
  -9.21628962e-01]
 [-1.70368776e+00 -9.96025673e-01 -2.45771849e+00  8.31590392e-01
  -1.58699049e+00 -2.12268580e+00  1.75072545e+00  2.30262686e-01
  -1.68742325e+00]]
[2 1 0 1 1 0 2 1 2 0 1 0 2 1 1 0 2 0 2 1 1 1 1 2 0 1 2 1 0 2 2 1 2 0 0 1 1
 2 0 1 2 0 2 0 1 1 1 2 0 0 0 1 2 0 0 1 0 2 1 2 0 0 0 1 0 1 0 2 2 2 0]
```

Training

Logistic Regression

In [24]:

```python
#Learning Logistic Regression Classifier
from sklearn.linear_model import LogisticRegression
#Creating a LogisticRegression object
lr = LogisticRegression(random_state = 0)
#Fitting the model to the training data
lr.fit(X_train, y_train)
#Predicting test cases
y_pred = lr.predict(X_test)
```

Testing and evaluation

Confusion Matrix

In [25]:

```
#Constructing the Confusion Matrix
from sklearn.metrics import confusion_matrix
#Creating a confusion_matrix object
cm = confusion_matrix(y_test, y_pred)
#Printing the confusion matrix
print(cm)
#Calculate the accuracy and see if we can get a better result by changing the value of
 k at the 'Feature selection' step
#Since this is not a binary classification, finding the TP, TN, FP and FN values was a
 little more difficult.
#A formula was used from the following site:
#https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-
model-ff9aa3bf7826
#By calculating the accuracy for each of the three cases, and taking the mean of them,
 we can simplify the formula to this:
accuracy = (3 * cm.item(0) + cm.item(1) + cm.item(2) + cm.item(3) + 3 * cm.item(4) + cm
.item(5) + cm.item(6) + cm.item(7) + 3 * cm.item(8) ) / (3 * (cm.item(0) + cm.item(1) +
cm.item(2) + cm.item(3) + cm.item(4) + cm.item(5) + cm.item(6) + cm.item(7) + cm.item(8
) ) )
#Printing the accuracy
print('Accuracy: ')
print(accuracy)
```

```
[[17  8  0]
 [ 0 23  2]
 [ 1  2 18]]
Accuracy: 
0.8779342723004695
```

Training

Decision Tree

In [26]:

```
#Learning Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
#Creating a DecisionTreeClassifier object
dt = DecisionTreeClassifier(criterion = 'entropy', random_state = 0, ccp_alpha = 0.0)
#Fitting the model to the training data
dt.fit(X_train, y_train)
#Predicting test cases
y_pred = dt.predict(X_test)
```

Testing and evaluation

Confusion Matrix

In [27]:

```
# Constructing the Confusion Matrix
from sklearn.metrics import confusion_matrix
#Fitting the model to the training data
cm = confusion_matrix(y_test, y_pred)
#Printing the confusion matrix
print(cm)
accuracy = (cm.item(0) + cm.item(0) + cm.item(0) + cm.item(1) + cm.item(2) + cm.item(3)
+ cm.item(4) + cm.item(4) + cm.item(4) + cm.item(5) + cm.item(6) + cm.item(7) + cm.item
(8) + cm.item(8) + cm.item(8) ) / (3 * (cm.item(0) + cm.item(1) + cm.item(2) + cm.item(
3) + cm.item(4) + cm.item(5) + cm.item(6) + cm.item(7) + cm.item(8) ) )
#Printing the accuracy
print('Accuracy: ')
print(accuracy)
```

```
[[17  5  3]
 [ 3 20  2]
 [ 6  0 15]]
Accuracy: 
0.8215962441314554
```

Conclusion

If we take a look at the accuracy of both the Logistic Regression and the Decision Tree algorithm, we can compare them and conclude which one produces a more accurate result in case of this dataset. This, of course, is not constant, for as we change the value of k, one might be more accurate than the other and vice versa. By trying out every possible value of k, I have concluded that LR produced a higher accuracy for almost any value of k, except for 4 and 5. In case of k = 4, the accuracy of the two classifiers is identical (0.77). In case of k = 5, the accuracy of DT is a bit higher. However, this hardly matters, for I have concluded earlier that the highest possible accuracy that we can achieve in this scenario (for both LR and DT) is if k has the value of 9. In this case, LR has an accuracy 0f 0.88, while DT has only 0.82. The biggest difference in the accuracy of LR and DT is at k = 10. In this case, LR has 0.88, while DT has only 0.79. In conclusion, LR is the better choice for this scenario.