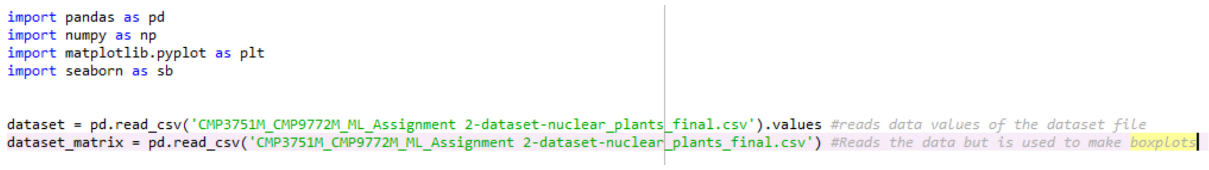
Joseph Doody 16603020@students.lincoln.ac.uk 16603020

Machine Learning CMP3751M Assignment 1

Section 1: Data import, Summary, Pre-processing and Visualisation

I need to first read the dataset .csv file into the Python IDE and I use the pd.read_csv() function which is found in the Pandas library within Python to allow me to read the dataset in. To find out what I am working with in the dataset I have two uses of reading the dataset. One use is so that I can display the titles of the features within the dataset and the other use is to be able to read the data that is held in the features. First, I will output the names of all the column titles so I can then deduct what features are being used in this dataset. Doing this has told me the name of the features within the dataset and how many features there are which is 12. Now I can move onto the next part of finding out summary statistics such as minimum, maximum, mean and variance of each feature and I can match a statistic to its feature.

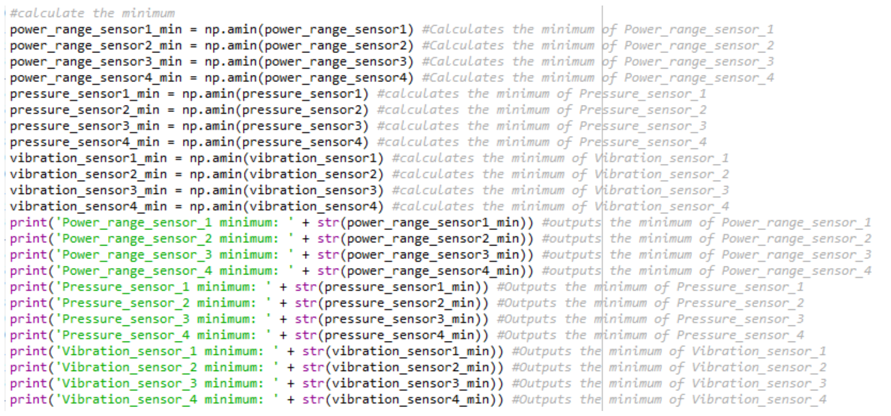


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb


dataset = pd.read_csv('CMP3751M_CMP9772M_ML_Assignment 2-dataset-nuclear_plants_final.csv').values #reads data values of the dataset file
dataset_matrix = pd.read_csv('CMP3751M_CMP9772M_ML_Assignment 2-dataset-nuclear_plants_final.csv') #Reads the data but is used to make boxplots
```

From here I can then make use of the NumPy library to work out the various summary statistics using functions numpy.amin to work out the minimum, numpy.amax to work out the maximum, numpy.mean to work out the mean and numpy.var to work out the variance. I do this for each feature in the dataset and have the results in a table below:

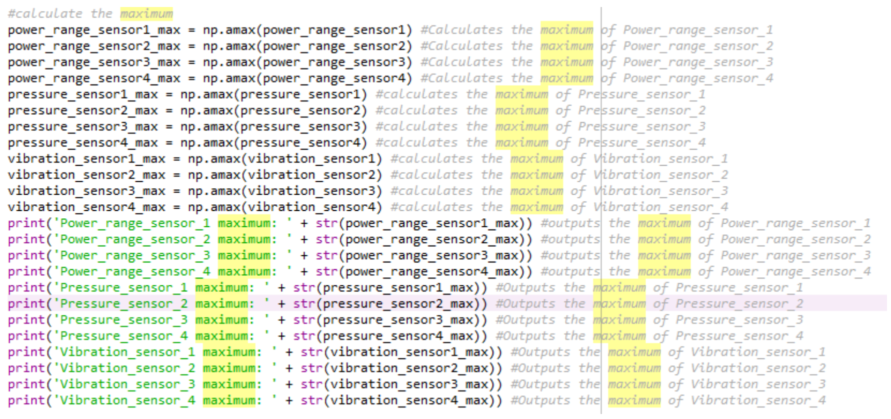
Feature	Minimum	Maximum	Mean	Variance
Power_range_sensor_1	0.0082	12.1	5	7.64
Power_range_sensor_2	0.0403	11.9	6.38	5.34
Power_range_sensor_3	2.58	15.8	9.23	6.41
Power_range_sensor_4	0.0623	17.2	7.36	18.9
Pressure_sensor_1	0.0248	67.98	14.2	136.3
Pressure_sensor_2	0.00826	10.2	3.078	4.52
Pressure_sensor_3	0.00122	12.6	5.75	6.37
Pressure_sensor_4	0.0058	16.6	5	17.3
Vibration_sensor_1	0	36.2	8.16	38.1
Vibration_sensor_2	0.0185	34.9	10	53.8
Vibration_sensor_3	0.0646	53.2	15.2	147.7
Vibration_sensor_4	0.0092	43.2	9.93	52.98

Code snippet to calculate the minimum:



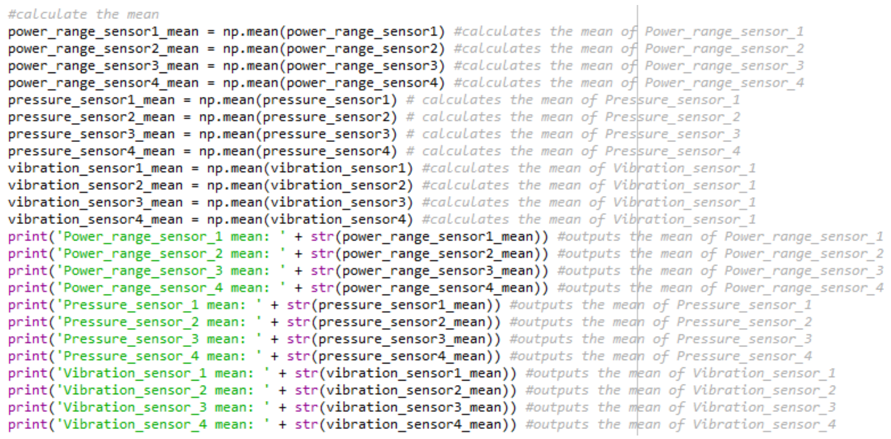
```
#calculate the minimum
power_range_sensor1_min = np.amin(power_range_sensor1) #Calculates the minimum of Power_range_sensor_1
power_range_sensor2_min = np.amin(power_range_sensor2) #Calculates the minimum of Power_range_sensor_2
power_range_sensor3_min = np.amin(power_range_sensor3) #Calculates the minimum of Power_range_sensor_3
power_range_sensor4_min = np.amin(power_range_sensor4) #Calculates the minimum of Power_range_sensor_4
pressure_sensor1_min = np.amin(pressure_sensor1) #calculates the minimum of Pressure_sensor_1
pressure_sensor2_min = np.amin(pressure_sensor2) #calculates the minimum of Pressure_sensor_2
pressure_sensor3_min = np.amin(pressure_sensor3) #calculates the minimum of Pressure_sensor_3
pressure_sensor4_min = np.amin(pressure_sensor4) #calculates the minimum of Pressure_sensor_4
vibration_sensor1_min = np.amin(vibration_sensor1) #calculates the minimum of Vibration_sensor_1
vibration_sensor2_min = np.amin(vibration_sensor2) #calculates the minimum of Vibration_sensor_2
vibration_sensor3_min = np.amin(vibration_sensor3) #calculates the minimum of Vibration_sensor_3
vibration_sensor4_min = np.amin(vibration_sensor4) #calculates the minimum of Vibration_sensor_4
print('Power_range_sensor_1 minimum: ' + str(power_range_sensor1_min)) #outputs the minimum of Power_range_sensor_1
print('Power_range_sensor_2 minimum: ' + str(power_range_sensor2_min)) #outputs the minimum of Power_range_sensor_2
print('Power_range_sensor_3 minimum: ' + str(power_range_sensor3_min)) #outputs the minimum of Power_range_sensor_3
print('Power_range_sensor_4 minimum: ' + str(power_range_sensor4_min)) #outputs the minimum of Power_range_sensor_4
print('Pressure_sensor_1 minimum: ' + str(pressure_sensor1_min)) #Outputs the minimum of Pressure_sensor_1
print('Pressure_sensor_2 minimum: ' + str(pressure_sensor2_min)) #Outputs the minimum of Pressure_sensor_2
print('Pressure_sensor_3 minimum: ' + str(pressure_sensor3_min)) #Outputs the minimum of Pressure_sensor_3
print('Pressure_sensor_4 minimum: ' + str(pressure_sensor4_min)) #Outputs the minimum of Pressure_sensor_4
print('Vibration_sensor_1 minimum: ' + str(vibration_sensor1_min)) #Outputs the minimum of Vibration_sensor_1
print('Vibration_sensor_2 minimum: ' + str(vibration_sensor2_min)) #Outputs the minimum of Vibration_sensor_2
print('Vibration_sensor_3 minimum: ' + str(vibration_sensor3_min)) #Outputs the minimum of Vibration_sensor_3
print('Vibration_sensor_4 minimum: ' + str(vibration_sensor4_min)) #Outputs the minimum of Vibration_sensor_4
```

Code snippet to calculate the maximum:



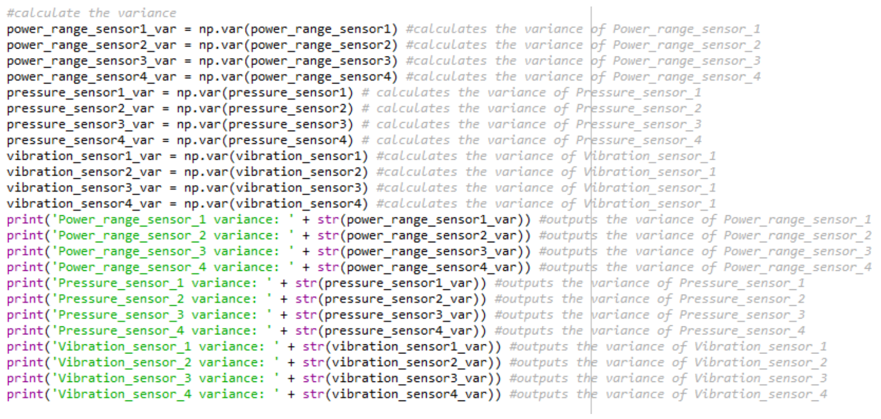
```
#calculate the maximum
power_range_sensor1_max = np.amax(power_range_sensor1) #Calculates the maximum of Power_range_sensor_1
power_range_sensor2_max = np.amax(power_range_sensor2) #Calculates the maximum of Power_range_sensor_2
power_range_sensor3_max = np.amax(power_range_sensor3) #Calculates the maximum of Power_range_sensor_3
power_range_sensor4_max = np.amax(power_range_sensor4) #Calculates the maximum of Power_range_sensor_4
pressure_sensor1_max = np.amax(pressure_sensor1) #calculates the maximum of Pressure_sensor_1
pressure_sensor2_max = np.amax(pressure_sensor2) #calculates the maximum of Pressure_sensor_2
pressure_sensor3_max = np.amax(pressure_sensor3) #calculates the maximum of Pressure_sensor_3
pressure_sensor4_max = np.amax(pressure_sensor4) #calculates the maximum of Pressure_sensor_4
vibration_sensor1_max = np.amax(vibration_sensor1) #calculates the maximum of Vibration_sensor_1
vibration_sensor2_max = np.amax(vibration_sensor2) #calculates the maximum of Vibration_sensor_2
vibration_sensor3_max = np.amax(vibration_sensor3) #calculates the maximum of Vibration_sensor_3
vibration_sensor4_max = np.amax(vibration_sensor4) #calculates the maximum of Vibration_sensor_4
print('Power_range_sensor_1 maximum: ' + str(power_range_sensor1_max)) #outputs the maximum of Power_range_sensor_1
print('Power_range_sensor_2 maximum: ' + str(power_range_sensor2_max)) #outputs the maximum of Power_range_sensor_2
print('Power_range_sensor_3 maximum: ' + str(power_range_sensor3_max)) #outputs the maximum of Power_range_sensor_3
print('Power_range_sensor_4 maximum: ' + str(power_range_sensor4_max)) #outputs the maximum of Power_range_sensor_4
print('Pressure_sensor_1 maximum: ' + str(pressure_sensor1_max)) #Outputs the maximum of Pressure_sensor_1
print('Pressure_sensor_2 maximum: ' + str(pressure_sensor2_max)) #Outputs the maximum of Pressure_sensor_2
print('Pressure_sensor_3 maximum: ' + str(pressure_sensor3_max)) #Outputs the maximum of Pressure_sensor_3
print('Pressure_sensor_4 maximum: ' + str(pressure_sensor4_max)) #Outputs the maximum of Pressure_sensor_4
print('Vibration_sensor_1 maximum: ' + str(vibration_sensor1_max)) #Outputs the maximum of Vibration_sensor_1
print('Vibration_sensor_2 maximum: ' + str(vibration_sensor2_max)) #Outputs the maximum of Vibration_sensor_2
print('Vibration_sensor_3 maximum: ' + str(vibration_sensor3_max)) #Outputs the maximum of Vibration_sensor_3
print('Vibration_sensor_4 maximum: ' + str(vibration_sensor4_max)) #Outputs the maximum of Vibration_sensor_4
```

Code snippet to calculate the mean:



```
#calculate the mean
power_range_sensor1_mean = np.mean(power_range_sensor1) #calculates the mean of Power_range_sensor_1
power_range_sensor2_mean = np.mean(power_range_sensor2) #calculates the mean of Power_range_sensor_2
power_range_sensor3_mean = np.mean(power_range_sensor3) #calculates the mean of Power_range_sensor_3
power_range_sensor4_mean = np.mean(power_range_sensor4) #calculates the mean of Power_range_sensor_4
pressure_sensor1_mean = np.mean(pressure_sensor1) # calculates the mean of Pressure_sensor_1
pressure_sensor2_mean = np.mean(pressure_sensor2) # calculates the mean of Pressure_sensor_2
pressure_sensor3_mean = np.mean(pressure_sensor3) # calculates the mean of Pressure_sensor_3
pressure_sensor4_mean = np.mean(pressure_sensor4) # calculates the mean of Pressure_sensor_4
vibration_sensor1_mean = np.mean(vibration_sensor1) #calculates the mean of Vibration_sensor_1
vibration_sensor2_mean = np.mean(vibration_sensor2) #calculates the mean of Vibration_sensor_1
vibration_sensor3_mean = np.mean(vibration_sensor3) #calculates the mean of Vibration_sensor_1
vibration_sensor4_mean = np.mean(vibration_sensor4) #calculates the mean of Vibration_sensor_1
print('Power_range_sensor_1 mean: ' + str(power_range_sensor1_mean)) #outputs the mean of Power_range_sensor_1
print('Power_range_sensor_2 mean: ' + str(power_range_sensor2_mean)) #outputs the mean of Power_range_sensor_2
print('Power_range_sensor_3 mean: ' + str(power_range_sensor3_mean)) #outputs the mean of Power_range_sensor_3
print('Power_range_sensor_4 mean: ' + str(power_range_sensor4_mean)) #outputs the mean of Power_range_sensor_4
print('Pressure_sensor_1 mean: ' + str(pressure_sensor1_mean)) #outputs the mean of Pressure_sensor_1
print('Pressure_sensor_2 mean: ' + str(pressure_sensor2_mean)) #outputs the mean of Pressure_sensor_2
print('Pressure_sensor_3 mean: ' + str(pressure_sensor3_mean)) #outputs the mean of Pressure_sensor_3
print('Pressure_sensor_4 mean: ' + str(pressure_sensor4_mean)) #outputs the mean of Pressure_sensor_4
print('Vibration_sensor_1 mean: ' + str(vibration_sensor1_mean)) #outputs the mean of Vibration_sensor_1
print('Vibration_sensor_2 mean: ' + str(vibration_sensor2_mean)) #outputs the mean of Vibration_sensor_2
print('Vibration_sensor_3 mean: ' + str(vibration_sensor3_mean)) #outputs the mean of Vibration_sensor_3
print('Vibration_sensor_4 mean: ' + str(vibration_sensor4_mean)) #outputs the mean of Vibration_sensor_4
```

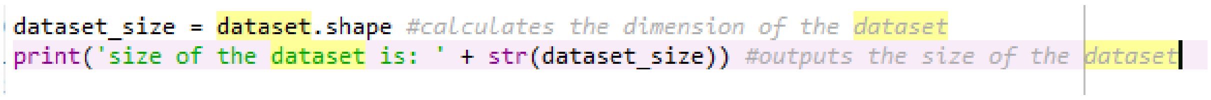
Code snippet to calculate the variance:



```
#calculate the variance
power_range_sensor1_var = np.var(power_range_sensor1) #calculates the variance of Power_range_sensor_1
power_range_sensor2_var = np.var(power_range_sensor2) #calculates the variance of Power_range_sensor_2
power_range_sensor3_var = np.var(power_range_sensor3) #calculates the variance of Power_range_sensor_3
power_range_sensor4_var = np.var(power_range_sensor4) #calculates the variance of Power_range_sensor_4
pressure_sensor1_var = np.var(pressure_sensor1) # calculates the variance of Pressure_sensor_1
pressure_sensor2_var = np.var(pressure_sensor2) # calculates the variance of Pressure_sensor_2
pressure_sensor3_var = np.var(pressure_sensor3) # calculates the variance of Pressure_sensor_3
pressure_sensor4_var = np.var(pressure_sensor4) # calculates the variance of Pressure_sensor_4
vibration_sensor1_var = np.var(vibration_sensor1) #calculates the variance of Vibration_sensor_1
vibration_sensor2_var = np.var(vibration_sensor2) #calculates the variance of Vibration_sensor_1
vibration_sensor3_var = np.var(vibration_sensor3) #calculates the variance of Vibration_sensor_1
vibration_sensor4_var = np.var(vibration_sensor4) #calculates the variance of Vibration_sensor_1
print('Power_range_sensor_1 variance: ' + str(power_range_sensor1_var)) #outputs the variance of Power_range_sensor_1
print('Power_range_sensor_2 variance: ' + str(power_range_sensor2_var)) #outputs the variance of Power_range_sensor_2
print('Power_range_sensor_3 variance: ' + str(power_range_sensor3_var)) #outputs the variance of Power_range_sensor_3
print('Power_range_sensor_4 variance: ' + str(power_range_sensor4_var)) #outputs the variance of Power_range_sensor_4
print('Pressure_sensor_1 variance: ' + str(pressure_sensor1_var)) #outputs the variance of Pressure_sensor_1
print('Pressure_sensor_2 variance: ' + str(pressure_sensor2_var)) #outputs the variance of Pressure_sensor_2
print('Pressure_sensor_3 variance: ' + str(pressure_sensor3_var)) #outputs the variance of Pressure_sensor_3
print('Pressure_sensor_4 variance: ' + str(pressure_sensor4_var)) #outputs the variance of Pressure_sensor_4
print('Vibration_sensor_1 variance: ' + str(vibration_sensor1_var)) #outputs the variance of Vibration_sensor_1
print('Vibration_sensor_2 variance: ' + str(vibration_sensor2_var)) #outputs the variance of Vibration_sensor_2
print('Vibration_sensor_3 variance: ' + str(vibration_sensor3_var)) #outputs the variance of Vibration_sensor_3
print('Vibration_sensor_4 variance: ' + str(vibration_sensor4_var)) #outputs the variance of Vibration_sensor_4
```

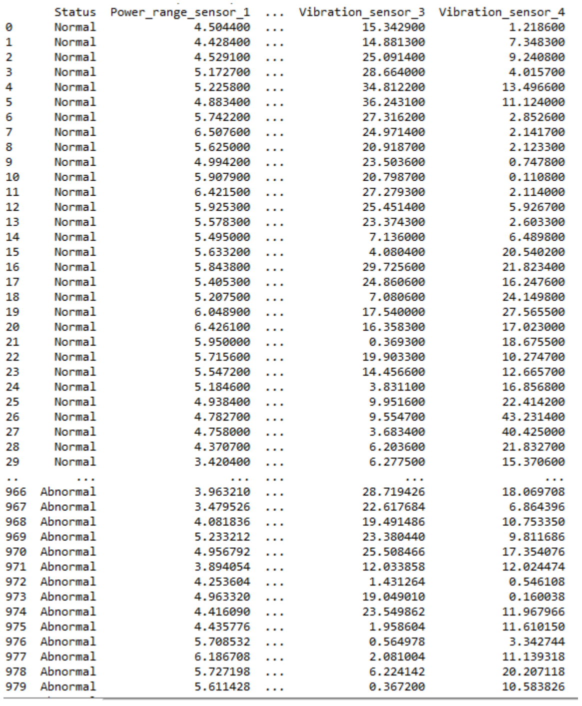
There are other steps of data pre-processing I can find out in this dataset such as determining the size of the dataset and if there are any categorical variables which are variables that can take on one of a limited value.

To work out the size of the dataset I make use of the np.shape() function which will output the rows by columns size of the dataset. From running this, I can say that the size of the dataset is 996x13 (rows to columns)



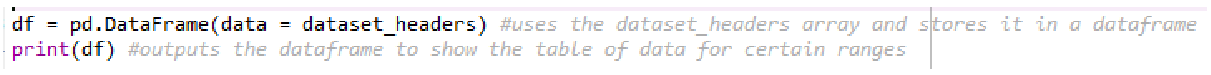
```
dataset_size = dataset.shape #calculates the dimension of the dataset
print('size of the dataset is: ' + str(dataset_size)) #outputs the size of the dataset
```

I know in the dataset that there is a number of features that hold a lot of data but I want to find out if there are any categorical variables within the dataset and to find this out I will use the dataset and use the DataFrame() function found in the Pandas library so that I can output the dataset it self in a table format.



```
       Status  Power_range_sensor_1  ...  Vibration_sensor_3  Vibration_sensor_4
0      Normal              4.504400  ...           15.342900            1.218600
1      Normal              4.428400  ...           14.881300            7.348300
2      Normal              4.529100  ...           25.091400            9.240800
3      Normal              5.172700  ...           28.664000            4.015700
4      Normal              5.225800  ...           34.812200           13.496600
5      Normal              4.883400  ...           36.243100           11.124000
6      Normal              5.742200  ...           27.316200            2.852600
7      Normal              6.507600  ...           24.971400            2.141700
8      Normal              5.625000  ...           20.918700            2.123300
9      Normal              4.994200  ...           23.503600            0.747800
10     Normal              5.907900  ...           20.798700            0.110800
11     Normal              6.421500  ...           27.279300            2.114000
12     Normal              5.925300  ...           25.451400            5.926700
13     Normal              5.578300  ...           23.374300            2.603300
14     Normal              5.495000  ...            7.136000            6.489800
15     Normal              5.633200  ...            4.080400           20.540200
16     Normal              5.843800  ...           29.725600           21.823400
17     Normal              5.405300  ...           24.860600           16.247600
18     Normal              5.207500  ...            7.080600           24.149800
19     Normal              6.048900  ...           17.540000           27.565500
20     Normal              6.426100  ...           16.358300           17.023000
21     Normal              5.950000  ...            0.369300           18.675500
22     Normal              5.715600  ...           19.903300           10.274700
23     Normal              5.547200  ...           14.456600           12.665700
24     Normal              5.184600  ...            3.831100           16.856800
25     Normal              4.938400  ...            9.951600           22.414200
26     Normal              4.782700  ...            9.554700           43.231400
27     Normal              4.758000  ...            3.683400           40.425000
28     Normal              4.370700  ...            6.203600           21.832700
29     Normal              3.420400  ...            6.277500           15.370600
..        ...                   ...  ...                 ...                 ...
966  Abnormal              3.963210  ...           28.719426           18.069708
967  Abnormal              3.479526  ...           22.617684            6.864396
968  Abnormal              4.081836  ...           19.491486           10.753350
969  Abnormal              5.233212  ...           23.380440            9.811686
970  Abnormal              4.956792  ...           25.508466           17.354076
971  Abnormal              3.894054  ...           12.033858           12.024474
972  Abnormal              4.253604  ...            1.431264            0.546108
973  Abnormal              4.963320  ...           19.049010            0.160038
974  Abnormal              4.416090  ...           23.549862           11.967966
975  Abnormal              4.435776  ...            1.958604           11.610150
976  Abnormal              5.708532  ...            0.564978            3.342744
977  Abnormal              6.186708  ...            2.081004           11.139318
978  Abnormal              5.727198  ...            6.224142           20.207118
979  Abnormal              5.611428  ...            0.367200           10.583826
```

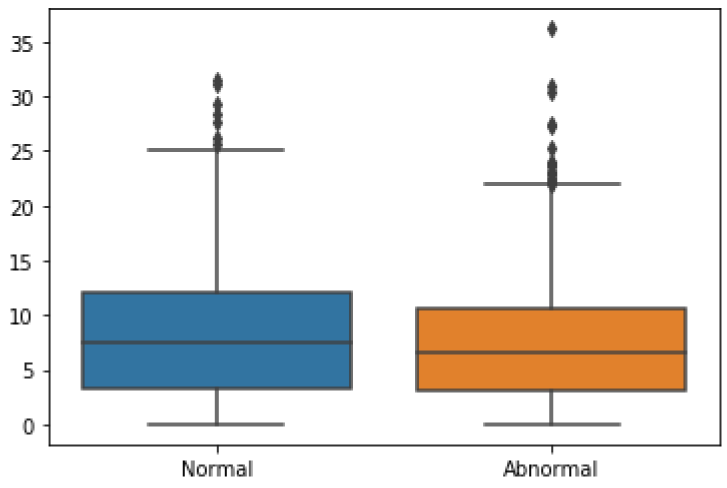
This is the output that is given for when the code is ran. As we know from before there are twelve different features within the dataset and in this example, we cannot see all twelve features since python isn't able to show that much data due to the amount of data there is. However, if we look at the first column 'Status' we have two different variables of data being 'Normal' and 'Abnormal', these would be considered the categorical variables found in this dataset.

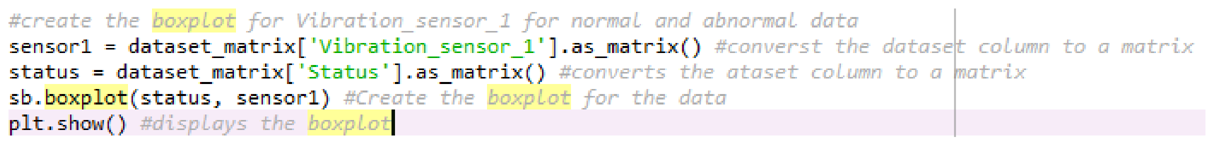


```
df = pd.DataFrame(data = dataset_headers) #uses the dataset_headers array and stores it in a dataframe
print(df) #outputs the dataframe to show the table of data for certain ranges
```

Now I’m going to give some visualisation to the dataset by creating a boxplot and a density plot.

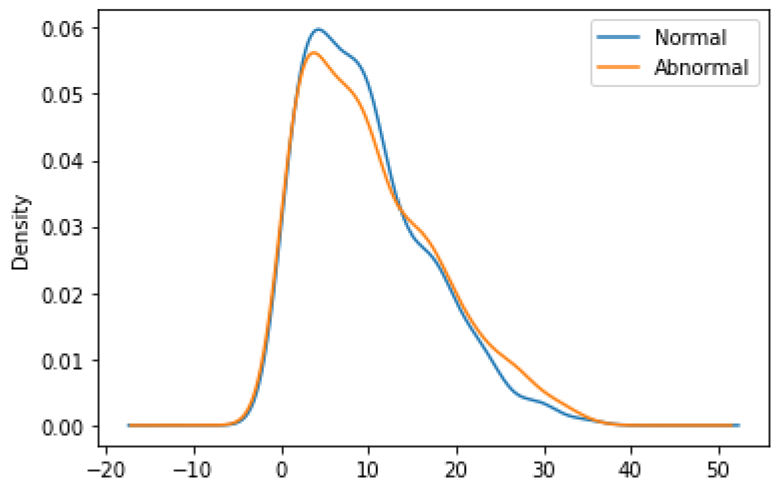
For the boxplot I will show the Vibration_sensor_1 data and show the data based on its status (‘Normal’, ‘Abnormal’). Earlier on when I read the data into python I read it in twice one which would be used to work out the summary statistics which would just get the values from the dataset and one that will just read the data as it is. What I can do with this is then read a specific column from the dataset and convert it into a matrix using .as_matrix(). Because I’m doing this with the Vibration_sensors_1 data and the Status data I will be reading both of the data from the dataset and then outputting it into a boxplot using function seaborn.boxplot(x, y) where the two columns of data will be the x and y within the function.

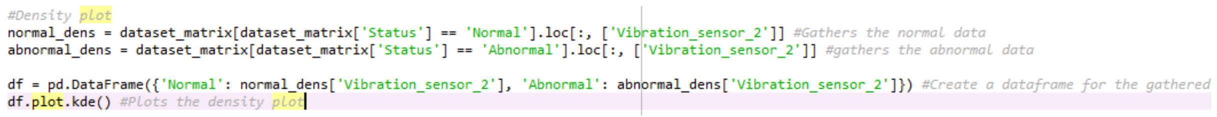




```
#create the boxplot for Vibration_sensor_1 for normal and abnormal data
sensor1 = dataset_matrix['Vibration_sensor_1'].as_matrix() #converst the dataset column to a matrix
status = dataset_matrix['Status'].as_matrix() #converts the ataset column to a matrix
sb.boxplot(status, sensor1) #Create the boxplot for the data
plt.show() #displays the boxplot
```

For the Density Plot I will be looking at the data in the Vibration_sensor_2 based on the status again (‘Normal’, ‘Abnormal’). This time I have split the data up into a Boolean such as normal is one group and abnormal is another group. I then put the split data into a data frame using the function pd.DataFrame. I assign this to a variable named df which I can then use to create the density plot using the function .plot.dke().





```
#Density plot
normal_dens = dataset_matrix[dataset_matrix['Status'] == 'Normal'].loc[:, ['Vibration_sensor_2']] #Gathers the normal data
abnormal_dens = dataset_matrix[dataset_matrix['Status'] == 'Abnormal'].loc[:, ['Vibration_sensor_2']] #gathers the abnormal data

df = pd.DataFrame({'Normal': normal_dens['Vibration_sensor_2'], 'Abnormal': abnormal_dens['Vibration_sensor_2']}) #Create a dataframe for the gathered
df.plot.kde() #Plots the density plot
```

The boxplot shows me the distribution of the data for Vibration_sensor_1. What I can tell from this plot are the minimum and maximum values, the quartile ranges (based on the box), and the median of the data and if there are any outliers laying within the data. I can see that when comparing both Normal and Abnormal data there are a few outliers based around the higher end of the data. Usually when dealing with outliers you wouldn't consider them in the dataset because they won't fit to the regression of the data so if we can see where the outliers are easier we can see where we need to look into to consider data being good. Also by analysing the size of the box we can see how spread the data is, if the box is small the data is close together and if the box is big then the spread of data is bigger.

A density plot will show me the distribution of the data over the interval of the data. It is similar to a histogram plot however it better shows the distribution of the data because it has a better determination of the distribution shape and the visualisation is a curve making it easier to see to the eye compared to bars on a graph.

Personally I would use a box plot because it better shows the outliers in the dataset which we wouldn't use to better analyse the data being visualised. What I can also determine is the spread of data by comparing the lengths of quartiles ranges to their closest minimum or maximum and the median to both points. If there is a big range between a quartile range and its minimum/maximum we can see that there is a lot of spread for the data points and vice versa if the range is small.

Section 2: Discussion of selecting an Algorithm

I will now make a discussion deciding on what method of classification I will use to train the dataset with. I will go through different types of classifiers which will involve:

- Logistic Regression
- K-Nearest-Neighbour (KNN)
- Support Vector Machines (SVM)
- Decision Trees
- Boosted Trees
- Random Forest
- Artificial Neural Networks (ANN)

From here I will go in depth on what the different classification methods are and decide if they would make a good match to the dataset that I am going to train.

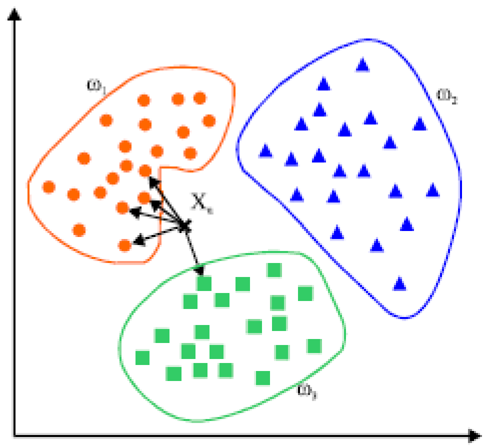
Logistic Regression

Is a type of Linear Classifier which is used to identify binary classification problems which is basically classification model that only uses two class values. It makes use of the Logistic Function which gives an S-shaped curve that can take any real-values number and map it into a value between 0 and 1. Much like linear regression, Logistic regression uses an equation as the representation where it will take an input value (of x) combined linearly using weights or coefficient values to predict an output value (of y). The only difference between them is that the value from a linear regression has an output value modelled as a binary value rather than a numeric value from the Logistic Regression. An advantage to using this type of classification is that it is very efficient and does not require too many computational resources. A disadvantage to this classifier is that like Linear Regression, it does not work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar to each other.

With regards to the dataset being analysed this type of classifier wouldn't work out since we are working with more than two features so I won't be choosing this type of classifier for the dataset

KNN

The idea of the K-Nearest-Neighbours classifier is to create clusters in the data by assigning different data points to a centroid within the plot to create K amount of clusters.



Here we have an example of data plotted on a scatter plot. The data is split into three different groups or clusters. This is the work of the K means algorithm where it has gotten a centroid point in the graph and assigned each data point to its closes centroid.

The Algorithm for KNN classifier goes like this:

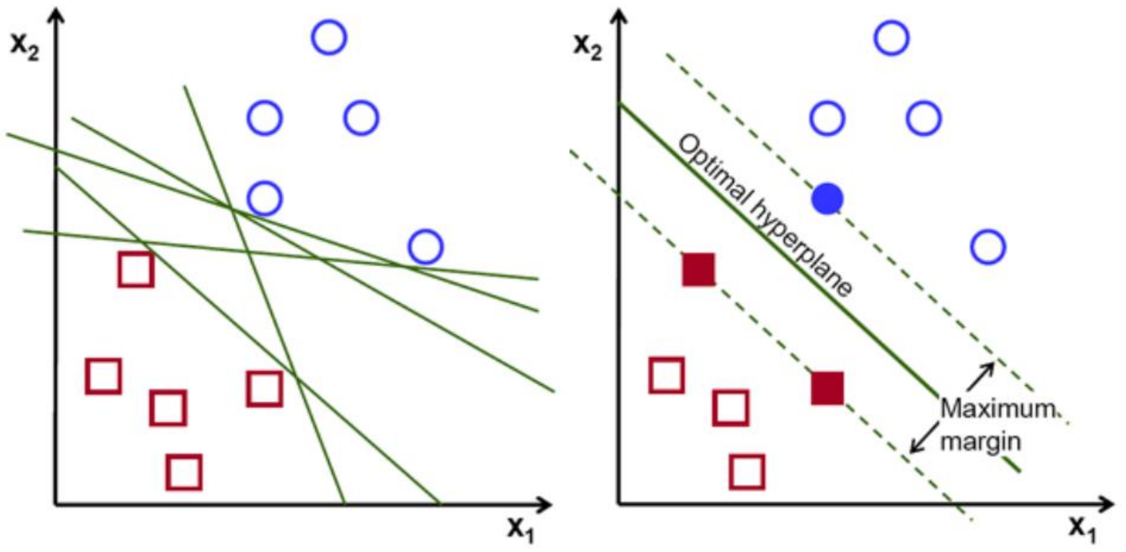
1. Initialise K to your chosen number of neighbours
2. For each example in the data
    a. Calculate the distance between the query example and the current example of the data
    b. Add the distance and the index of the example to an ordered collection
3. Sort the ordered collection of distances and indices from smallest to largest by the distances
4. Pick the first K entries from the sorted collection
5. Get the labels of the selected K entries
6. Return the mode of the K labels

An advantage to this type of classification is that there's no need to build a model, tune several parameters, or make additional assumptions. As for a disadvantage to this type of classifier is that it gets significantly slower as the number of examples or predictors/independent variables increase.

With regards to training the dataset I will not use this mean of classification due to the fact that it is used for finding clusters in data and the idea of this dataset is not to find clusters in the data.

SVM

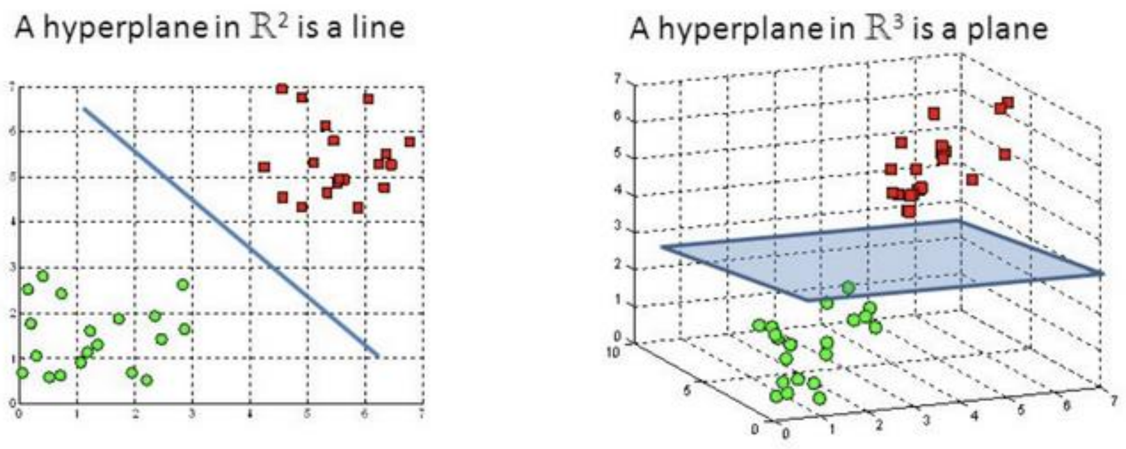
SVM or Support Vector Machine is another classifier that's objective is to find a hyperplane in an N-dimensional space where N is the number of features in the dataset that distinctly classifies the data points.



Possible hyperplanes

In the image example above the green line represents possible hyperplanes in a dataset (left plot) so we want to find the plane that has the maximum margin which provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes are decision boundaries that help classify the data points. Where ever future data appears in the given boundaries you can then predict the different classes for the data. Hyperplanes can be visualised as a dimension which depends on the number of features. So if a there are two features the hyperplane is a line and if there are three features the hyperplane becomes a two-dimensional plane. Once the feature increases past three it gets difficult to visualise the hyperplane.



Hyperplanes in 2D and 3D feature space

An advantage to SVM is that it works relatively well when there is a clear margin separation between classes and a disadvantage to SVM is that once the data set becomes really large, the classifier is not suitable.

With regards to the dataset I won't want to use SVM because the dataset is rather large and uses more than three features so it would be hard to visualise the hyperplane for the dataset.

Decision Tree
The idea of a decision tree is to continuously split the data according to a certain parameter. A decision tree will consist of:

- **Nodes**: test for the value of a certain attribute
- **Branch:** correspond to the outcome of a test and connect to the next node or leaf
- **Leaf Node:** Terminal nodes that predict the outcome

The Decision Tree predicts data based on a yes/no answer to match it to a categorical answer. It makes use of the Divide-and-Conquer Algorithm where it will Select a test for root node and then split the instances of that root node into subsets or in other words based on the decision tree, each branch extending from the node. Repeat it recursively for each branch using only instances that reach the branch and then stop recursion for a branch if all its instances have the same class.

Once a decision tree has been trained and fitted with nodes to match the features in the dataset we can then test some of the data in the decision tree by taking the inputs from the features to give us a prediction based on the categorical features in the dataset. An advantage to this type of classifier is that the accuracy of this classifier is high if he data set is rather simple however once the dataset becomes more complex with more features and the data set being bigger the accuracy then becomes lower and classifier could come to a point of overfitting.

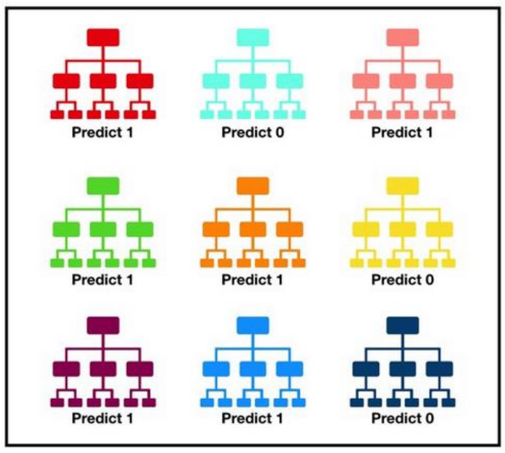
For the dataset in my case I won't want to use this type of classifier because there are a lot of features in the dataset which may cause a high error rate making the predicted data inaccurate.

Boosted Tree
The boosted tree classifier is an extension to the decision tree where it builds each regression tree in a step-wise fashion using a predefined loss function to measure the error in each step and correct it in the next step. This is selecting the optimal tree within these number of decision trees. What this is doing here is trying to correct match the test data in the classifier to create a more accurate prediction.

Just like before with the decision tree it's basically a more accurate classifier which is an advantage if the dataset is simple however vice versa the dataset being big can cause a situation of over-fitting which will give poor predictions to any test data.

In this dataset we have a complex dataset with many features and if it was to overfit based on the training data, our test data could come out with a high error rate. Since we are handling with a lot of data we can only assume there will be outliers to the dataset which would make the classifier no make a good prediction because of it.

Random Forest
For the random forest it again makes use of the decision trees however this time it consists of a large number of individual decision trees that operate as an ensemble. The way this classifier is tested is that it will take on the test data and run it through the different decision trees and whatever class comes out most is the prediction the forest has given.



Tally: Six 1s and Three 0s
Prediction: 1

This is an example of a random forest where it has taken a test data into the number of trees and given an output it will then pull out the result of the most probable prediction as its class.

An advantage to this type of classifier is that if the dataset is small you can get a very strong prediction on the data because it takes some error into account of the data.

The disadvantage to this is when the dataset is big with a lot of features and even if there are random trees there can still be a relatively high error rate, not as a high as a decision tree but still an error rate which my throw off a prediction for test data based on how many outliers that may be in the dataset.

This could be a good classifier to use because its an improvement to decision trees because it uses multiple decision trees and gets the class that has the most votes. It's a good measure to predict the data since the more trees you train the better predictions you can make. The dataset we are using does make use of a lot of features but if we train enough trees we can have different trees have different root nodes so that when we test data a different result may come out and once we have enough of a majority of a vote then the prediction rate for accuracy should be relatively high

ANN

ANN or Artificial Neural Network is a mathematical model inspired by biological neural networks, for classification. It resembles the human brain in the following two ways:

- A neural network acquires knowledge through learning
- A neural network's knowledge is stored within inter-neuron connection strengths known as synaptic weights.

In a typical ANN, it comprises in different layers:

- Input layer – Receives input from the outside world on which network will learn, recognise about or otherwise process
- Output layer – Containing units that respond to the information about how it's learned any task
- Hidden layer – The units are in between input and output layers and its job is to transform the input into something that output unit can use in some way.

To train an ANN it needs a function $\hat{y} = f_\delta(x)$, where δ stands for all the weights and biases present in the network and then all we need is:

- A set of training samples,
- A loss function L(y, $\hat{y}$), and
- An iterative scheme, for example gradient descent, to perform the optimisation process.

Since ANN work like the human brain you can say that it learns from its data so if it does look at the dataset and sees outliers it can learn from the data and identify it as an outlier and produce an accurate as possible outcome, this would be an advantage to this classifier.

However, since the great power this classifier does hold it is hardware dependent so it will need a lot of power for it to run which could make it time consuming depending on the size of the dataset.

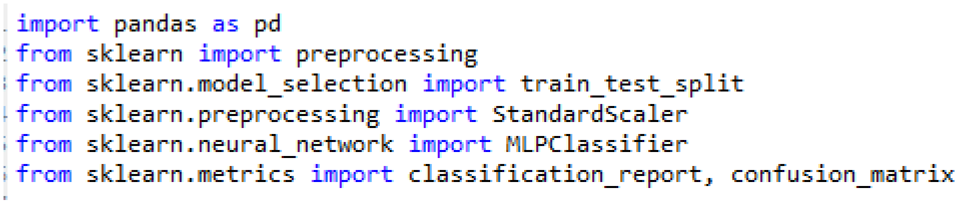
This would be a decent method of classification for the dataset since it would work well with the big dataset and outline any outliers that may be in the dataset.

Overall, we want to find the classifier that is best suited to the dataset. The machine learning intern worked out there was a 90% accuracy rate with using a classifier on the dataset. I believe that the ANN classifier would give a high level of accuracy compared to the other classifiers since it is good to deal with a big set of data like the dataset we are handling. Another would go to the random forest since we can train a lot of trees to match all the features that are within the dataset.

Section 3: Designing Algorithms

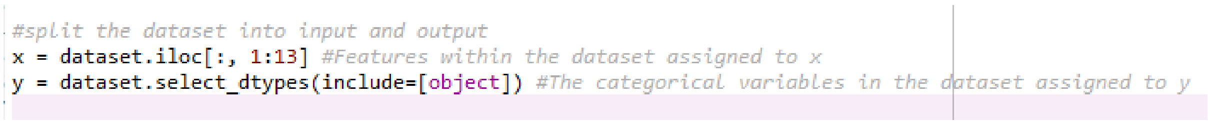
I will now be putting the dataset to the test by randomly choosing 90% of the data as a training set and the rest of the 10% of the data as a test set. The classifiers I will be using will be the Artificial Neural Network classifier and the Random Forests Classifier.

The libraries I have used to perform the ANN classifier:



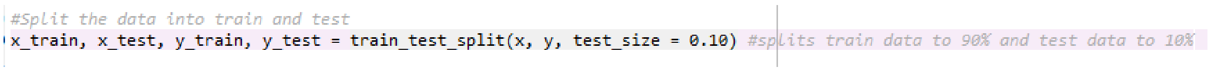
```
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

To split my data into input and output (x and y respectively) I assign the features to a variable x and the categorical variables to a variable y. This will allow me to do the necessary classification on both classifiers to be able to output results based on training data.



```
#split the dataset into input and output
x = dataset.iloc[:, 1:13] #Features within the dataset assigned to x
y = dataset.select_dtypes(include=[object]) #The categorical variables in the dataset assigned to y
```

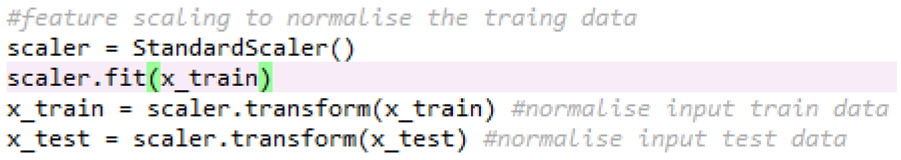
The first thing I need to do is split my data into training set and test set randomly. I make use of the sklearn.model_selection library to import train_test_split function into my program. The function to shuffle my data and split it into the training set and test set is 'train_test_split(x, y, test_size = 0.10)' where the x is the features from the dataset and the y is the categorical variable 'status' in the dataset. Since I'm wanting to work with 90% training set and 10% test set and set the test_size to 0.10 which sets the test set to 10% and automatically sets the training set to 90%. The variables I assign this to are x_train, x_test, y_train and y_test.



```
#Split the data into train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.10) #splits train data to 90% and test data to 10%
```

To ensure that I gather actual predictions it's a good practice to normalise the input data so that all of them can be uniformly evaluated. I will only do this on the input data because the idea is to keep the output data as real as possible so that the neural network can make predictions on real world data there making the test data as real as possible.

I first standardise the features by removing the mean and scaling to the unit variance. I can do this by using the function sklearn.preprocessing.**StandardScaler()** where I will assign it to a variable named scaler once I have a variable to normalise I can then use .fit() alongside variable scaler to normalise training set and the train it.

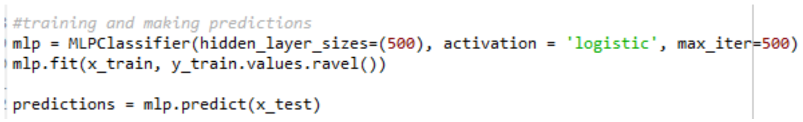


```
#feature scaling to normalise the traing data
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train) #normalise input train data
x_test = scaler.transform(x_test) #normalise input test data
```

Once the training data has been normalised I can then move onto training it while also making predictions based on the test set. The idea of my test is to have a neural network with 500 hidden layers using the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$, and then I want to specify how many iterations I want to take (epochs) which will be 1000.

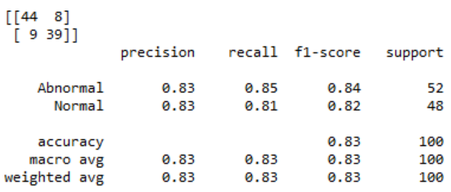
I will use the function MLPClassifier() which is a class from the sklearn.neural_network library and it will take on three parameters. The first parameter will be hidden_layer_sizes which is used to set the size of the hidden layers in the neural network. This parameter will take on the value of 500 because that's how many hidden layers I want. The second parameter will be activation which will determine what function is to be used in the neural network. The default for this parameter is the relu function but I want to use the sigmoid function so this parameter will take on the value 'logistic' .The last parameter will be max_iter which will be used to specify the number of iterations (epochs) that I want the neural network to execute. The value I will give it is 1000 because that's how many times I want the classifier to iterate.

Finally I then want to train the function using the function .fit(). Again I want to use the input values to train the dataset so the data I will be using are the x_train and the y_train. Once all this is done I can then make a prediction by testing the test set to the classification found from the training data. The function I will use is mlp.predict(x_test) where mlp is the ANN training model and the function .predict() predicts the data based on parameter x_test data.

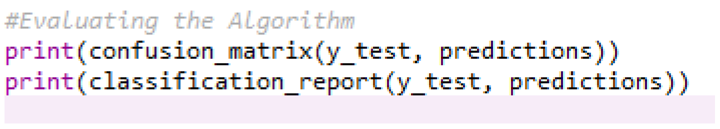


```
#training and making predictions
mlp = MLPClassifier(hidden_layer_sizes=(500), activation = 'logistic', max_iter=500)
mlp.fit(x_train, y_train.values.ravel())

predictions = mlp.predict(x_test)
```

To evaluate the classification I will output the test results in a confusion matrix which is used to evaluate the accuracy of a classification. The function .confusion_matrix() is from the sklearn.metrics library which is used to compare the ground truth data (y_test) to the predicted values from the classifier (prediction). To then show the accuracy of the data I use the function .classification_report() where it will build a text report showing the main classification metrics. This will also take the ground truth data (y_test) and the predicted data from the classifier (prediction) to give me the results.



```
[[44  8]
 [ 9 39]]
              precision    recall  f1-score   support

    Abnormal       0.83      0.85      0.84        52
      Normal       0.83      0.81      0.82        48

    accuracy                           0.83       100
   macro avg       0.83      0.83      0.83       100
weighted avg       0.83      0.83      0.83       100
```

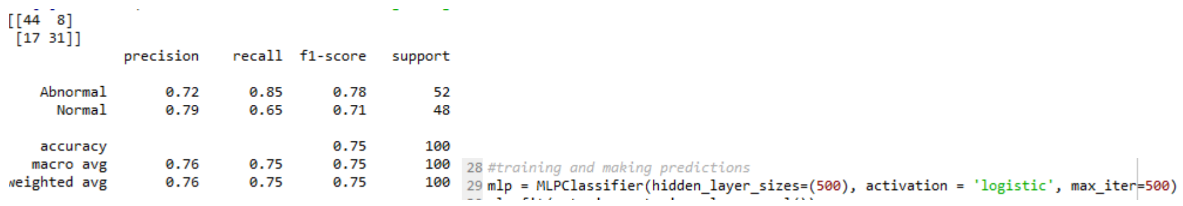


```
#Evaluating the Algorithm
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

So what I know from running the classifier with 1000 epochs and 500 hidden layers the accuracy of the classifier to this dataset is 83%.

As a bonus to this I also wanted to find out what the results would give if the dataset had a different size of epoch. I wanted to see what the accuracy would involve if I set the epoch value to 500 and 2000. To do this I would have to change the max_iter value in the function MLPClassifier().

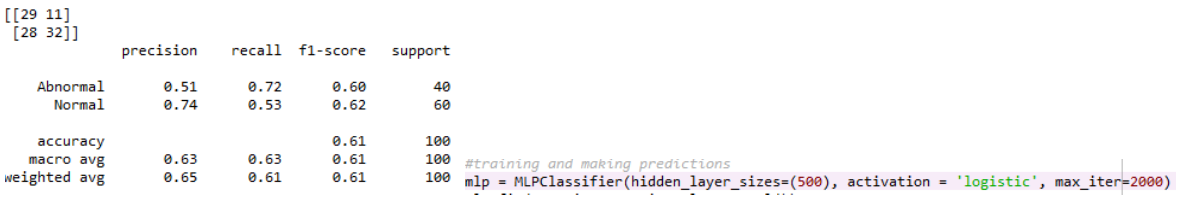
<u>Epoch 500:</u>



```
[[44  8]
 [17 31]]
              precision    recall  f1-score   support

    Abnormal       0.72      0.85      0.78        52
      Normal       0.79      0.65      0.71        48

    accuracy                           0.75       100
   macro avg       0.76      0.75      0.75       100
weighted avg       0.76      0.75      0.75       100
```

```
#training and making predictions
mlp = MLPClassifier(hidden_layer_sizes=(500), activation = 'logistic', max_iter=500)
```

<u>Epoch 2000:</u>



```
[[29 11]
 [28 32]]
              precision    recall  f1-score   support

    Abnormal       0.51      0.72      0.60        40
      Normal       0.74      0.53      0.62        60

    accuracy                           0.61       100
   macro avg       0.63      0.63      0.61       100
weighted avg       0.65      0.61      0.61       100
```

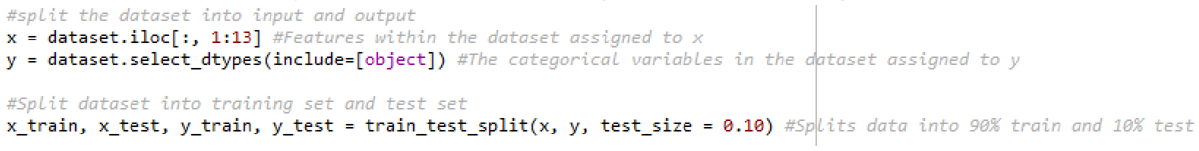
```
#training and making predictions
mlp = MLPClassifier(hidden_layer_sizes=(500), activation = 'logistic', max_iter=2000)
```

I've noticed that the higher the Epoch is the lower the accuracy of the ANN becomes. I think this happens because you are running the classifier more so its testing the training data more which could result into finding more outliers in the dataset to that when we then test the classifier it is resulting in a test giving an error when you compare the predicted value to the ground truth.

For the second classifier I will be creating a Random forests classifier where again just like the ANN classifier I will be using 90% training data and 10% test data to see the accuracy. These are the following libraries I will be using to perform this classifier.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

Just like the ANN classifier I first read in the dataset and split the data into input data and output data assigning them to variables x and y respectively. From here I use the train_test_split() function to shuffle the data and split them into 90% training set and 10% testing set.



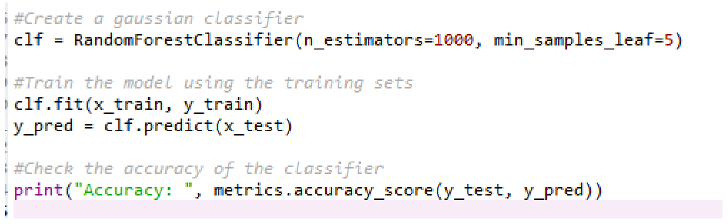
```
#split the dataset into input and output
x = dataset.iloc[:, 1:13] #Features within the dataset assigned to x
y = dataset.select_dtypes(include=[object]) #The categorical variables in the dataset assigned to y

#Split dataset into training set and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.10) #Splits data into 90% train and 10% test
```

This time to create the random forests classifier I use the function RandomForestClassifier() which will take on two parameters. The first parameter is called n_estimators which is used to define how many trees you want to use in the classifier. I want to use 1000 trees so I will set this parameter to 1000. The last parameter is called min_samples_leaf() which is used to determine the minimum number of samples required to be at a leaf node. I want to test two different inputs being 5 and 50 so I will make sure to change them accordingly and see how the accuracy changes.

Once the classifier has been trained I then want to test the classifier by making a predictions based on the input test results. So I first use the function .fit() for both the training data to train the classifier and then use the function .predict to test the input test data to give me a prediction based on the classifier.

Min_leaf_samples 5:

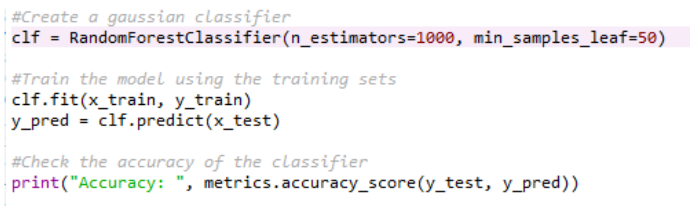
The accuracy gave: 88%



```
#Create a gaussian classifier
clf = RandomForestClassifier(n_estimators=1000, min_samples_leaf=5)

#Train the model using the training sets
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

#Check the accuracy of the classifier
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
```

Min_leaf_samples 50:

The accuracy gave: 78%

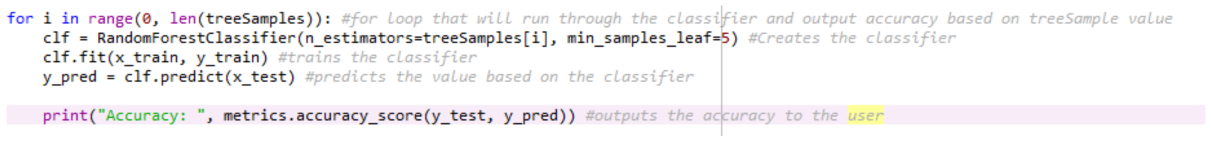


```
#Create a gaussian classifier
clf = RandomForestClassifier(n_estimators=1000, min_samples_leaf=50)

#Train the model using the training sets
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

#Check the accuracy of the classifier
print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))
```

As a bonus I wanted to monitor the accuracy of the classifier depending on how many number of trees I gave to the forest. I wanted to analyse {10, 50, 100, 1000, 5000} trees and see what the accuracy would output. Also in this example I will keep the min_samples_leaf value to 5. To do this I have created a for loop that will read the how many trees I want to use from a list and then create the classifier based on that many trees to then output the accuracy of the classifier based on how many trees have been used

Amount of Trees	Accuracy
10	0.89
50	0.9
100	0.91
1000	0.9
5000	0.9

What I have found out by running this test is that when you increase the number of trees it tries to find the optimal accuracy for the amount of minimum sample leafs you give to the tree. It shows that Once we increase the trees the accuracy may change to when we would have less trees but once we get to a point where the amount of trees is rather great the accuracy stops changing because I assume that it has found the accuracy the classifier can give to the dataset based on the amount of minimum sample leafs you give the to the tree.



```
for i in range(0, len(treeSamples)): #for loop that will run through the classifier and output accuracy based on treeSample value
    clf = RandomForestClassifier(n_estimators=treeSamples[i], min_samples_leaf=5) #Creates the classifier
    clf.fit(x_train, y_train) #trains the classifier
    y_pred = clf.predict(x_test) #predicts the value based on the classifier

    print("Accuracy: ", metrics.accuracy_score(y_test, y_pred)) #outputs the accuracy to the user
```

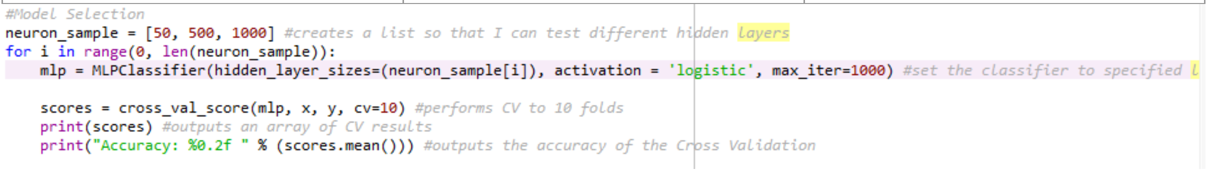
Section 4: Model selection

I've created my ANN and Random forests classifiers and now I want to see what parameters such as how many neurons for ANN and how many trees for Random Forests and see what the most optimal parameter for each classifier is. To do this I will be using the Cross Validation method with 10 folds. I can do this by using the function cross_val_score() which can be found in the sk.learn.model_selection library.

For ANN I want to test three different hidden layers being 50, 500 and 1000. So I first create a list that will store values based on these hidden layers. To then test the different neurons I will make a for loop that will loop the size of the list amount of times and create a classifier and use the amount of neurons set on that iteration of the loop. The epoch I will keep the same at 1000 for each different neuron. From here I will then use the cross_val_score() function to perform the cross validation for the three different hidden layer neurons based on the iteration of the loop it's on. This function will take on four different parameters, the first parameter will be the classifier, the second parameter will be the features, the third parameter will be the 'Status' and the last parameter will be the amount of folds I want to perform in the cross validation. The value I want to give this will be 10 since that is how many folds I will be performing.

After all of this I can output all the results the Cross Validation process has given me by outputting an array that will tell me the accuracy of each given fold and then the overall accuracy of the validation which will be the mean of the 10 folds.

Amount of Neurons	Array	Accuracy
50	0.31, 0.54, 0.62, 0.61, 0.75, 0.58, 0.56, 0.5, 0.5, 0.45	0.54
500	0.25, 0.52, 0.68, 0.61, 0.74, 0.6, 0.55, 0.49, 0.5, 0.53	0.55
1000	0.28, 0.52, 0.67, 0.58, 0.73, 0.64, 0.56, 0.47, 0.52, 0.49	0.55

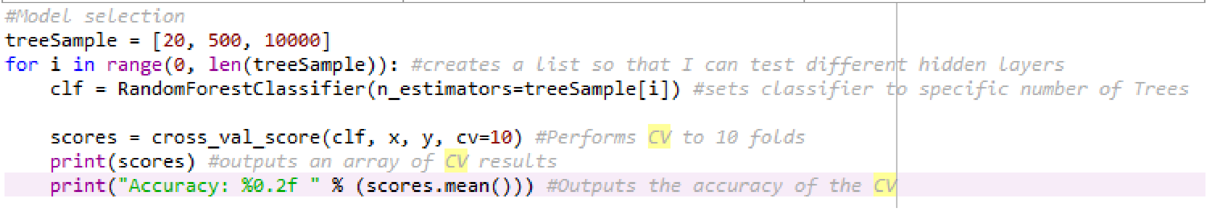


```
#Model Selection
neuron_sample = [50, 500, 1000] #creates a list so that I can test different hidden layers
for i in range(0, len(neuron_sample)):
    mlp = MLPClassifier(hidden_layer_sizes=(neuron_sample[i]), activation = 'logistic', max_iter=1000) #set the classifier to specified l

    scores = cross_val_score(mlp, x, y, cv=10) #performs CV to 10 folds
    print(scores) #outputs an array of CV results
    print("Accuracy: %0.2f " % (scores.mean())) #outputs the accuracy of the Cross Validation
```

For the Random Forests I want to test three different amount of trees in the forest being 20, 500 and 10000 trees. Just like before with ANN I create a list that will hold all the values of the amount of trees I want to use. I then create a for loop that will loop the size of the list amount of times. The for loop will consist of setting the Random Forests classifier by setting the amount of trees being the certain value from the tree list on that iteration.

Just like before we can then use the cross_val_score() function to pass parameters through of the classifier, features, 'Status' values and the amount of folds (still being 10).

Amount of Trees	Array	Accuracy
20	0.36, 0.56, 0.71, 0.63, 0.8, 0.62, 0.64, 0.83, 0.49	0.62
500	0.41, 0.49, 0.73, 0.6, 0.73, 0.52, 0.6, 0.8, 0.49	0.6
10000	0.37, 0.5, 0.72, 0.62, 0.73, 0.52, 0.59, 0.78, 0.49	0.6



```
#Model selection
treeSample = [20, 500, 10000]
for i in range(0, len(treeSample)): #creates a list so that I can test different hidden layers
    clf = RandomForestClassifier(n_estimators=treeSample[i]) #sets classifier to specific number of Trees

    scores = cross_val_score(clf, x, y, cv=10) #Performs CV to 10 folds
    print(scores) #outputs an array of CV results
    print("Accuracy: %0.2f " % (scores.mean())) #Outputs the accuracy of the CV
```

Now that I have performed Cross Validation on both classifiers I can see how each fold (split) compares to the test set data. If I compare both results between the Random Forests and the ANN I can see that the accuracy is better for the Random Forests solely based on the accuracy values both classifiers give when I perform Cross Validation.

I also wanted to see what accuracy would look like within each fold so I would output those values. Every time I would output each folds accuracy for all amount of neuron/amount of trees. The accuracy would appear the same. For the amount of neuron/amount of trees I would stick 500 neurons for the ANN and 500 trees for the Random Forests. I didn't want to go for the highest accuracy of both classifiers because the amount of neurons/trees were not high enough to me to get a reliable source of data. If I was to run it with a different split of data the lower parameters could give complete different accuracy's to the last time it was ran.

I also want to point out that when I ran both classifiers in section three I would get a better accuracy using the Random Forests classifier on the dataset. When running it with 1000 trees I would get around 90% accuracy compared to 70% of the ANN. I think this because the random forests makes use of many decision trees to give me a verdict of the class to the test feature data. I know that a decision tree isn't a good way to analyse data with low amount of features and when the data set is rather big but when we try to test multiple decision trees each with varying nodes and different outputs to counter any error due to cause of outliers to give me a more solid prediction of the classifier.

References

Brownlee, J., 2016. *Logistic Regression for Machine Learning.* [Online]
Available at: https://machinelearningmastery.com/logistic-regression-for-machine-learning/
[Accessed 16 Decemeber 2019].

Chakure, A., 2019. *Decision Tree Classification.* [Online]
Available at: https://towardsdatascience.com/decision-tree-classification-de64fc4d5aac
[Accessed 16 December 2019].

Donges, N., 2018. *The Logistic Regression Algorithm.* [Online]
Available at: https://machinelearningmastery.com/logistic-regression-for-machine-learning/
[Accessed 16 December 2019].

Gandhi, R., 2018. *Support Vector Machine - Introduction to Machine Learning Algorithms.* [Online]
Available at: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
[Accessed 16 December 2019].

Gilley, S., 2019. *Boosted Decision Tree Regression.* [Online]
Available at: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/boosted-decision-tree-regression
[Accessed 16 December 2019].

Harrison, O., 2018. *Machine Learning Basics with the K-Nearest Neighbors Algorithm.* [Online]
Available at: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
[Accessed 16 December 2019].

K, D., 2019. *Top 4 advantages and disadvantages of Support Vector Machine or SVM.* [Online]
Available at: https://medium.com/@dhiraj8899/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107
[Accessed 16 December 2019].

Dr Leontidis, G, 2019. *Introduction to Artificial Neural Networks Principles*[PowerPoint presentaion]. CMP3751: *Machine Learning*. Available at: http://blackboard.lincoln.ac.uk/ [Accessed: 16 December 2019].

Navlani, A., 2018. *Understanding Random Forests Classifiers in Python.* [Online]
Available at: https://www.datacamp.com/community/tutorials/random-forests-classifier-python
[Accessed 16 December 2019].

Robinson, S., 2018. *Introduction to Neural Netowrks with Scikit-Learn.* [Online]
Available at: https://stackabuse.com/introduction-to-neural-networks-with-scikit-learn/
[Accessed 16 December 2019].

Sidana, M., 2017. *Types of Classification Algorithms in Machine Learning.* [Online]
Available at: https://medium.com/@Mandysidana/machine-learning-types-of-classification-9497bd4f2e14
[Accessed 16 Decemeber 2019].

Yiu, T., 2019. *Understanding Random Forest.* [Online]
Available at: https://towardsdatascience.com/understanding-random-forest-58381e0602d2
[Accessed 16 December 2019].