# MACHINE LEARNING AND NEURAL COMPUTING: DATA CLASSIFICATION COURSE-WORK

Brook Raindle
19039840

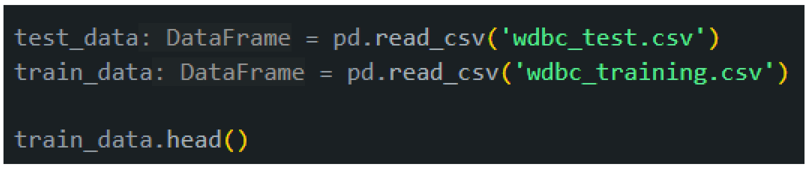
## CONTENTS

## UNDERSTANDING THE DATA

The data is comprised of several columns of information, preceded by a diagnosis classification, and an Identification number. the data is comprised of 10 features, each feature has 3 values related to it, The mean, The standard Error, and the 'worst' or Largest value (Largest = worst, considering we are discussing tumours, higher values and sizes always suggest more serious issues).

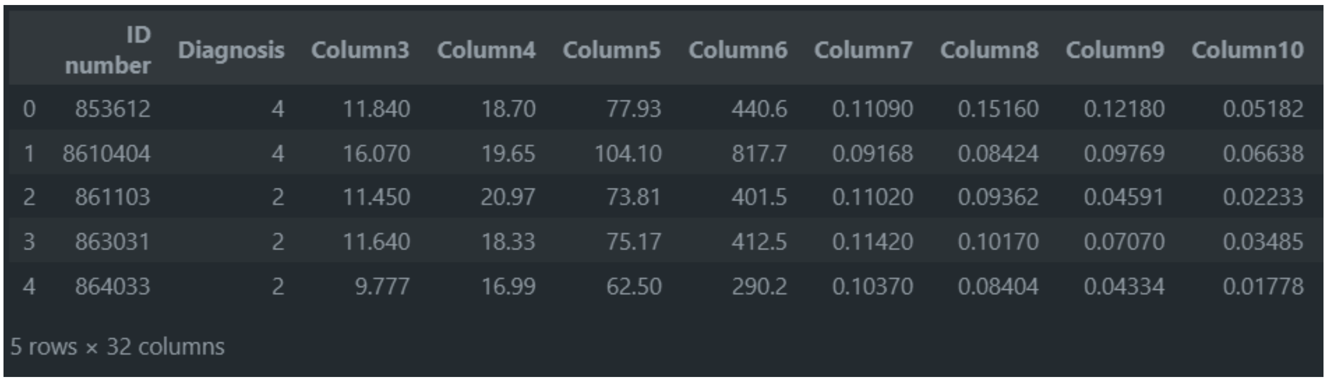
## TASK 1 – DATA EXPLORATION

A) To begin PCA testing, I first loaded both datasets into the Python script using the Pandas library. For each dataset, there are 30 'features', An identification number, and a Diagnosis classification value, in our scenario, 2 represents 'Benign', and 4 represents 'Malignant' they are read using Pandas' 'read_csv()' function as follows.



```
test_data: DataFrame = pd.read_csv('wdbc_test.csv')
train_data: DataFrame = pd.read_csv('wdbc_training.csv')

train_data.head()
```

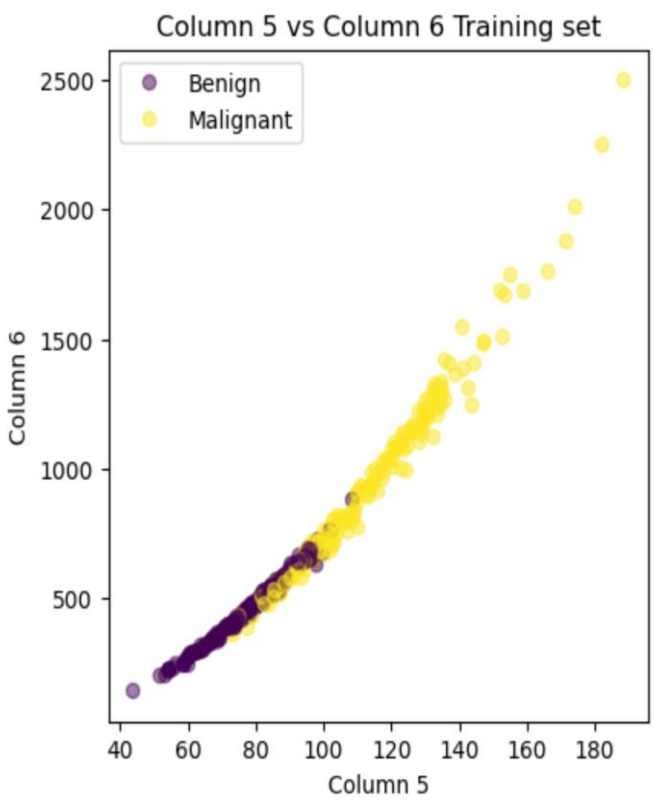
After loading the data and validating the data's purpose using the 'head()' function, where the columns are displayed as seen in the following image. Both ID, and Diagnosis can be seen, And there are example values for a handful of the columns.



| | ID number | Diagnosis | Column3 | Column4 | Column5 | Column6 | Column7 | Column8 | Column9 | Column10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 853612 | 4 | 11.840 | 18.70 | 77.93 | 440.6 | 0.11090 | 0.15160 | 0.12180 | 0.05182 |
| 1 | 8610404 | 4 | 16.070 | 19.65 | 104.10 | 817.7 | 0.09168 | 0.08424 | 0.09769 | 0.06638 |
| 2 | 861103 | 2 | 11.450 | 20.97 | 73.81 | 401.5 | 0.11020 | 0.09362 | 0.04591 | 0.02233 |
| 3 | 863031 | 2 | 11.640 | 18.33 | 75.17 | 412.5 | 0.11420 | 0.10170 | 0.07070 | 0.03485 |
| 4 | 864033 | 2 | 9.777 | 16.99 | 62.50 | 290.2 | 0.10370 | 0.08404 | 0.04334 | 0.01778 |

5 rows × 32 columns

B) Task 1B requires us to show a scatter plot comparing 2 of the features (columns). I chose to compare Column 5 and Column 6. Each element of data is classified and coloured accordingly, as seen in the Legend in the following:

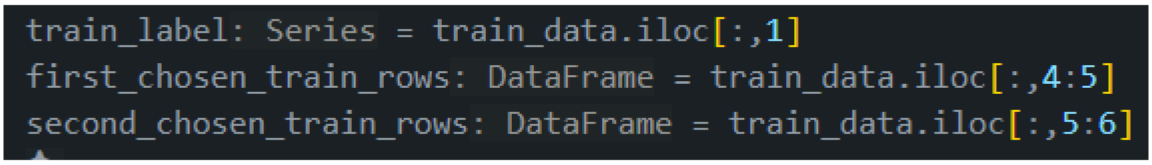


This code will take data from our database, and turn it into a graph. Firstly I assign 2 column data variables named ‘first_chosen_train_rows’ and ‘second_chosen_train_rows’. And I assigned their stored values to their allocated column data, using the ‘train_data.iloc[:,#:#]’ method, as seen in the code snippet below.

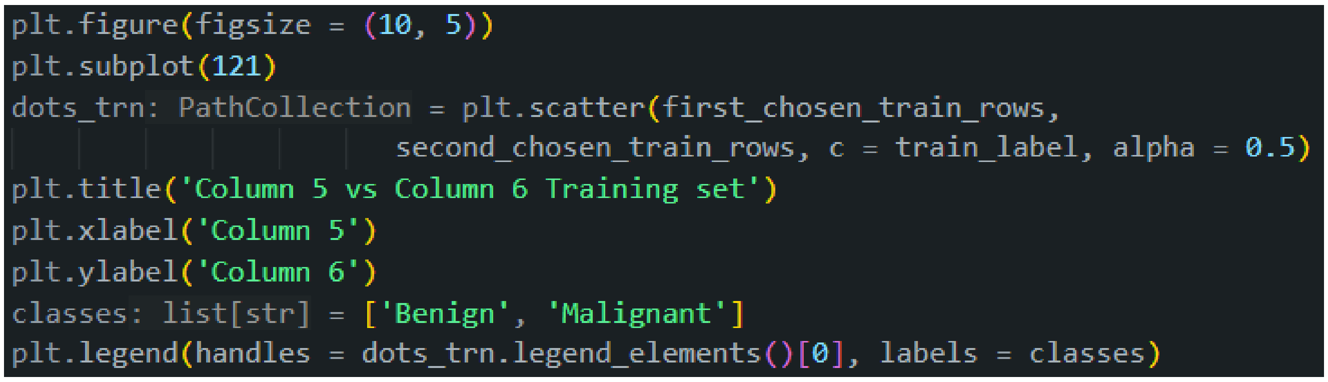
The graph was plotted with matplotlib, using the ‘scatter’ method. I set the x label and y label to ‘column 5’ and ‘column 6’ respectively. Then defining the classes, and creating a legend that correctly reflects the classification of each diagnosis.

A relationship can be seen between column 5, and 6 and their diagnosis. The data shown here represents 2 types of information, along the X-axis, is the mean perimeter, which could also be referred to as a ‘non-linear Circumference’ perhaps. In the Y axis, the Mean area is covered by the cell nucleus.

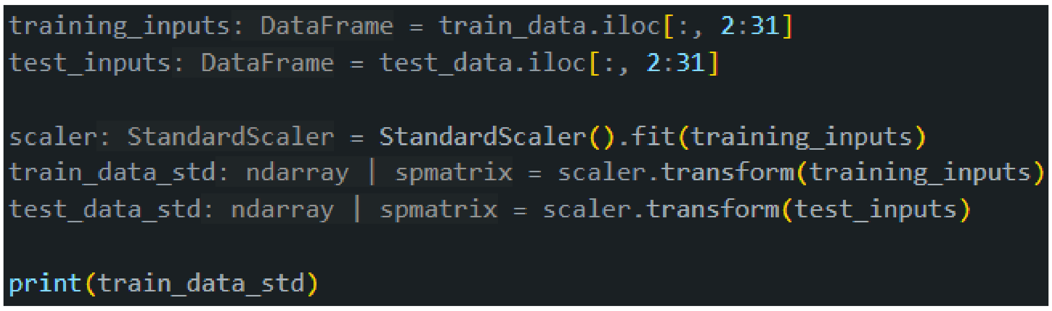
As both values increase, it becomes increasingly likely that the tumour is malignant, as seen in the above image, therefore it can be seen that the two components, Columns 5 and 6, have a significant impact on the classification of a tumour’s nature.



```
train_label: Series = train_data.iloc[:,1]
first_chosen_train_rows: DataFrame = train_data.iloc[:,4:5]
second_chosen_train_rows: DataFrame = train_data.iloc[:,5:6]
```

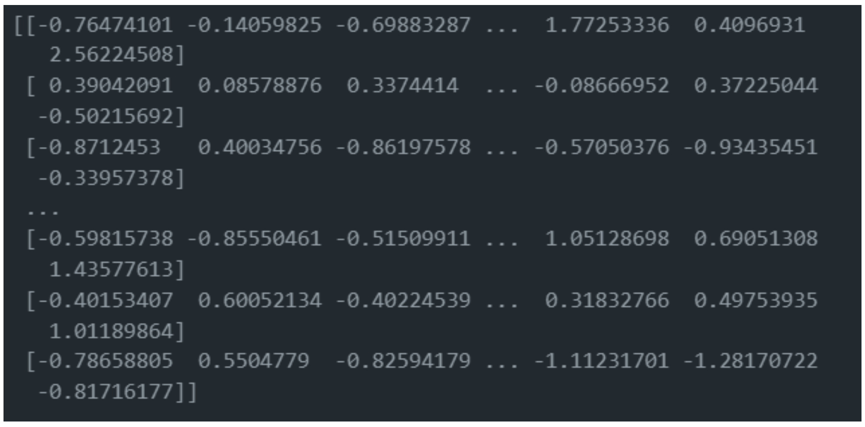


```
plt.figure(figsize = (10, 5))
plt.subplot(121)
dots_trn: PathCollection = plt.scatter(first_chosen_train_rows,
                                       second_chosen_train_rows, c = train_label, alpha = 0.5)
plt.title('Column 5 vs Column 6 Training set')
plt.xlabel('Column 5')
plt.ylabel('Column 6')
classes: list[str] = ['Benign', 'Malignant']
plt.legend(handles = dots_trn.legend_elements()[0], labels = classes)
```



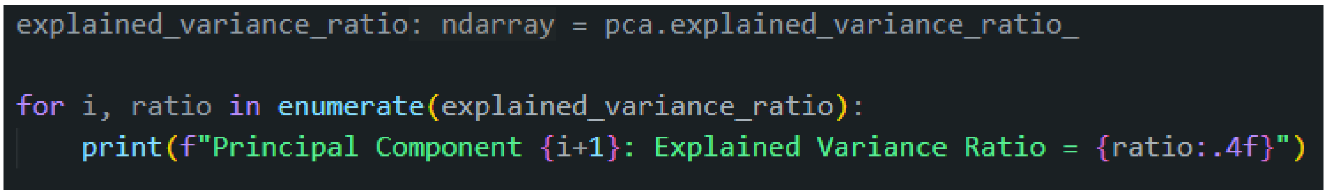
```python
training_inputs: DataFrame = train_data.iloc[:, 2:31]
test_inputs: DataFrame = test_data.iloc[:, 2:31]

scaler: StandardScaler = StandardScaler().fit(training_inputs)
train_data_std: ndarray | spmatrix = scaler.transform(training_inputs)
test_data_std: ndarray | spmatrix = scaler.transform(test_inputs)

print(train_data_std)
```

C) Normalising the dataset requires a function from the sklearn Preprocessing library. Using a scaler, the data can be normalised, however not all elements need to be affected, the feature inputs (columns 3-32) are assigned a variable name to be stored under, 'training_inputs' and then transformed using a StandardScalar(), as seen in the code snippet above. This is then given a new variable name to be used further in the future.
The new normalised data is then printed to the terminal for validation, resulting in the array List seen below

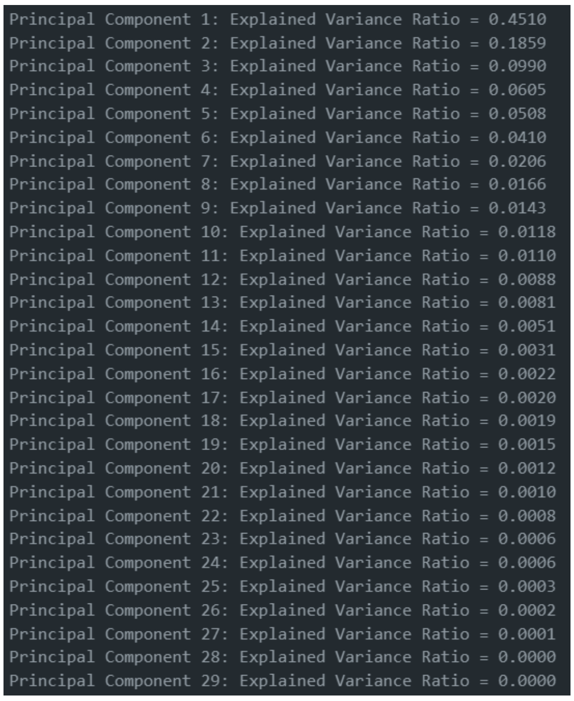


```
[[-0.76474101 -0.14059825 -0.69883287 ...  1.77253336  0.4096931
   2.56224508]
 [ 0.39042091  0.08578876  0.3374414  ... -0.08666952  0.37225044
  -0.50215692]
 [-0.8712453   0.40034756 -0.86197578 ... -0.57050376 -0.93435451
  -0.33957378]
 ...
 [-0.59815738 -0.85550461 -0.51509911 ...  1.05128698  0.69051308
   1.43577613]
 [-0.40153407  0.60052134 -0.40224539 ...  0.31832766  0.49753935
   1.01189864]
 [-0.78658805  0.5504779  -0.82594179 ... -1.11231701 -1.28170722
  -0.81716177]]
```

D) PCA Analysis: a principal component analysis is the process of comparing each component or 'feature' to decipher which of them provides the largest variance, in turn having an impact on the diagnosis classification. We are asked to produce a scree plot, and a comparison between the 2 most impactful components, to discover which of these components are most impactful, the information can be retrieved using the 'PCA.explained_variance_ratio' method as seen below.



```python
explained_variance_ratio: ndarray = pca.explained_variance_ratio_

for i, ratio in enumerate(explained_variance_ratio):
    print(f"Principal Component {i+1}: Explained Variance Ratio = {ratio:.4f}")
```

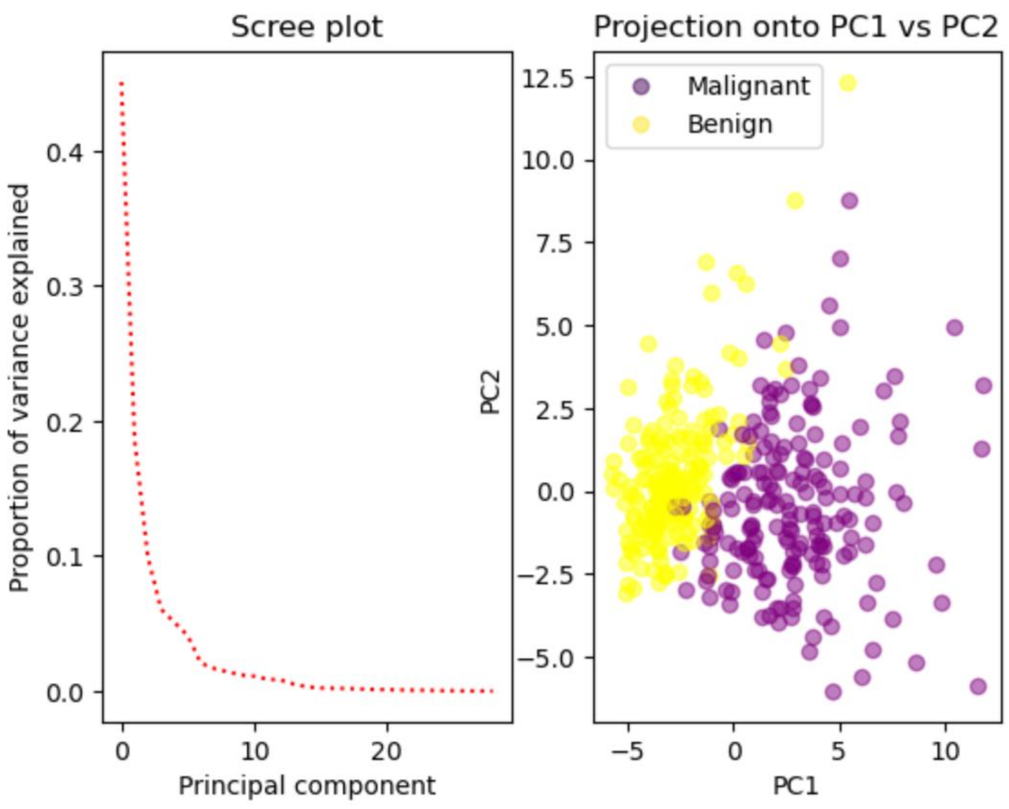


```
Principal Component 1: Explained Variance Ratio = 0.4510
Principal Component 2: Explained Variance Ratio = 0.1859
Principal Component 3: Explained Variance Ratio = 0.0990
Principal Component 4: Explained Variance Ratio = 0.0605
Principal Component 5: Explained Variance Ratio = 0.0508
Principal Component 6: Explained Variance Ratio = 0.0410
Principal Component 7: Explained Variance Ratio = 0.0206
Principal Component 8: Explained Variance Ratio = 0.0166
Principal Component 9: Explained Variance Ratio = 0.0143
Principal Component 10: Explained Variance Ratio = 0.0118
Principal Component 11: Explained Variance Ratio = 0.0110
Principal Component 12: Explained Variance Ratio = 0.0088
Principal Component 13: Explained Variance Ratio = 0.0081
Principal Component 14: Explained Variance Ratio = 0.0051
Principal Component 15: Explained Variance Ratio = 0.0031
Principal Component 16: Explained Variance Ratio = 0.0022
Principal Component 17: Explained Variance Ratio = 0.0020
Principal Component 18: Explained Variance Ratio = 0.0019
Principal Component 19: Explained Variance Ratio = 0.0015
Principal Component 20: Explained Variance Ratio = 0.0012
Principal Component 21: Explained Variance Ratio = 0.0010
Principal Component 22: Explained Variance Ratio = 0.0008
Principal Component 23: Explained Variance Ratio = 0.0006
Principal Component 24: Explained Variance Ratio = 0.0006
Principal Component 25: Explained Variance Ratio = 0.0003
Principal Component 26: Explained Variance Ratio = 0.0002
Principal Component 27: Explained Variance Ratio = 0.0001
Principal Component 28: Explained Variance Ratio = 0.0000
Principal Component 29: Explained Variance Ratio = 0.0000
```

The information shown in this image reflects the list of components, the explained variance ratio will total 1, representing 100% total variance ratio – across all data.

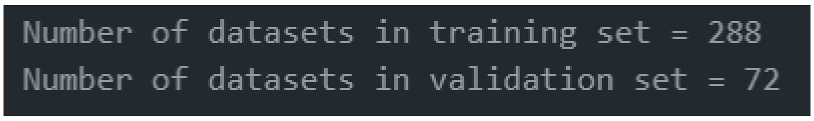
A trend can be seen where as you work your way down the list of components, the variation decreases, this can also be seen in the Scree plot below.
It seems like potentially the first 10 data entries seem to contain the most useful information, which conveniently, is all of the 'mean' values for each feature, as described in the data analysis.

When projecting PC1 and PC2 into a scatter graph, and then displaying them based on their classification, a trend can be extracted. PC1 = radius, PC2 = Texture (standard deviation of Grayscale values). As we can see, as the texture value increases past 0 it becomes extremely improbable that the tumour is benign, and increasingly probable of it being malignant, whereas PC2, shows less of a direct correlation. In combination though, it can be seen that there is a clear line between which malignant and benign can be classified. Moreso vertical suggesting that the trend relates mostly to the X axis (PC1 =radius).



## TASK 2 – DATA PREPARATION

A & B) Task 2 outlines the pre-processing stages of the assignment, where we distribute the data into training and validation sets, which will be used to train the model, we also further transform the data using the SandardScalar() to ensure it is all normalised. Using the 'train_test_split' function from the Sklearn library, we are able to separate the data into 2 parts, the training data, and the validation data, which will be used to compare the model trained using the training data, with the correct labels as seen in the validation data.

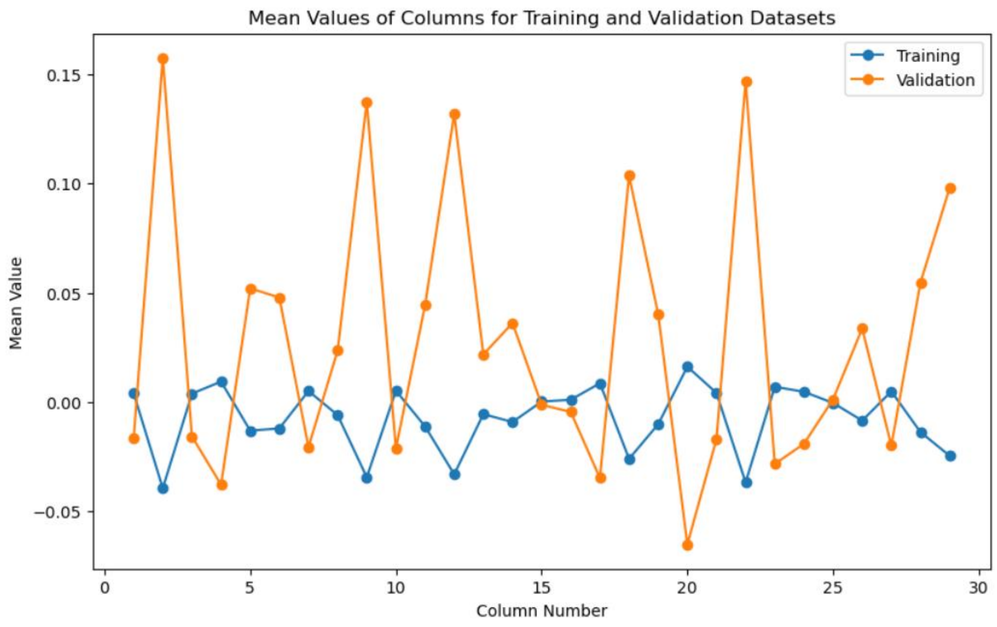


```
Number of datasets in training set = 288
Number of datasets in validation set = 72
```

I chose to separate them into an 8:2 ratio since we are limited to 360 data entries in the training set, and I know more training data = higher accuracy. However, it is still important to give plenty of validation data considering that is the only way to truly know the accuracy of the system In question. The data is split into the following:

| | |
|---|---|
| x_train = Training data set | x_val = Validation data set |
| y_train = Training Label | y_val = Validation Label |

The task also requests that a comparison be made between the validation and training set, and mean values for each column.

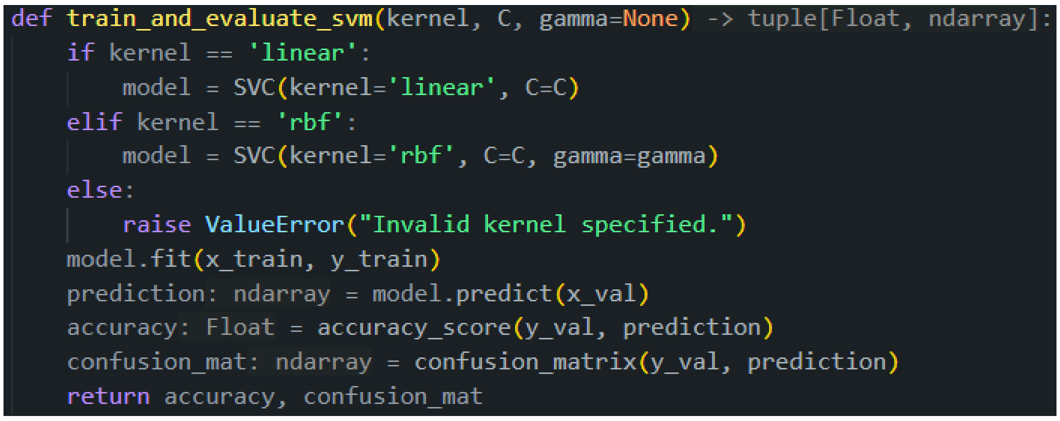


This graph shows he difference in mean value of the training data compared to the validation data after they have been Normalised. There seems to be significantly more variation in the validation dataset compared to the training set, with some values, receiving similar values, but several features with significant spikes in mean value compared to others. this could be because of improper splitting, potentially it could be beneficial to add a random element to the method which splits the data.

## TASK 3 – SVM CLASSIFICATION

**A)** The task is to create a support vector classifier using sklearn and perform several tests to evaluate the best approach.

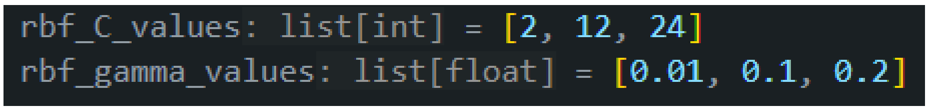
I decided since there would be multiple models created and tested, it would be most efficient to have a callable method, which uses a C value, and Gamma value as parameters, this allows me to only write the model fitting functions once, and call it whenever I want to make a new svc. Firstly, it will check which kernel to use, 'Linear' or 'rbf'. When the system finds out which, it will create a new SVC (I named mine 'model' for simplicity) using the suggested Kernel, and the C value provided as a parameter. If the model uses the 'rbf' kernel, it will also place the gamma parameter as the input.



```python
def train_and_evaluate_svm(kernel, C, gamma=None) -> tuple[Float, ndarray]:
    if kernel == 'linear':
        model = SVC(kernel='linear', C=C)
    elif kernel == 'rbf':
        model = SVC(kernel='rbf', C=C, gamma=gamma)
    else:
        raise ValueError("Invalid kernel specified.")
    model.fit(x_train, y_train)
    prediction: ndarray = model.predict(x_val)
    accuracy: Float = accuracy_score(y_val, prediction)
    confusion_mat: ndarray = confusion_matrix(y_val, prediction)
    return accuracy, confusion_mat
```

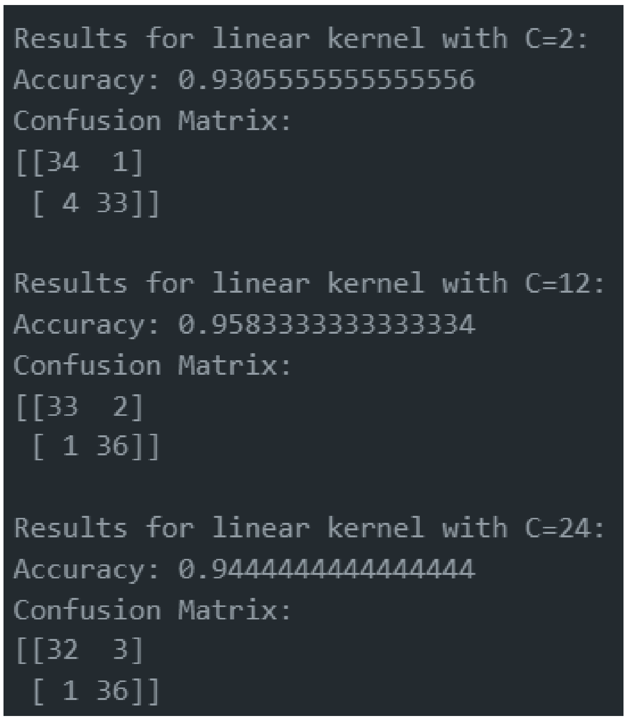
Then, the model is fit, using the 'x_train' data as the training data parameter, and 'y_train' as the training label parameter. These values are put into the model.fit() function provided by Sklearn.
The model is then tested with the predict() function. The x_val is used to test the model immediately and return accuracy values, and a confusion matrix, they are then returned to demonstrate the resulting statistics for analysis.

Now we have a method that creates an SVC using just kernel type, C value and Gamma value, we can test several models and evaluate for the best outcome.



```python
rbf_C_values: list[int] = [2, 12, 24]
rbf_gamma_values: list[float] = [0.01, 0.1, 0.2]
```

I chose to use 2, 12, and 24 for my C values to represent a good range of values, with good variation. And specifically for the rbf kernel, I use 0.01, 0.1, and 0.2 as my tested values.



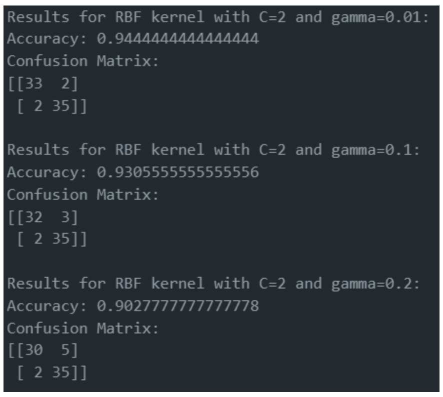
```
Results for linear kernel with C=2:
Accuracy: 0.9305555555555556
Confusion Matrix:
[[34  1]
 [ 4 33]]

Results for linear kernel with C=12:
Accuracy: 0.9583333333333334
Confusion Matrix:
[[33  2]
 [ 1 36]]

Results for linear kernel with C=24:
Accuracy: 0.9444444444444444
Confusion Matrix:
[[32  3]
 [ 1 36]]
```

For linear testing, all that varies is the C value used. Three models are represented in this image, one using C=2, One using C=12, and a final one using C=24.

It can be seen that C=12 provides the best accuracy with 95.8%, compared to the lesser 93.1% from C=2, and 94.4% from C=24. If I decide to go with a Linear kernel model, It would prove best to go with C=12 if I am limited to those 3 options.

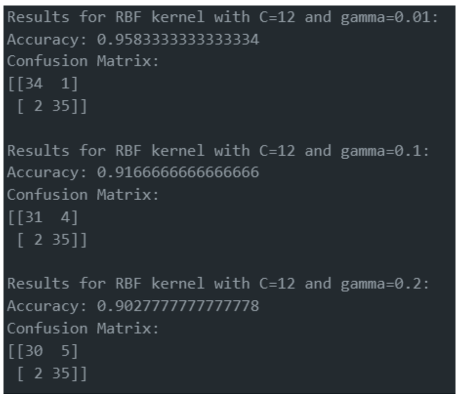
The confusion Matrix' display the predicted results, with the top right, and bottom left values representing the incorrectly classified validation elements. C=2 had a total of 5 incorrect, C=12 had 3, and C=24 had 4.

**<u>C = 2</u>**



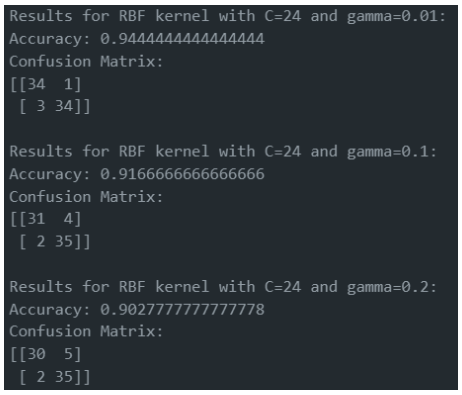
```
Results for RBF kernel with C=2 and gamma=0.01:
Accuracy: 0.9444444444444444
Confusion Matrix:
[[33  2]
 [ 2 35]]

Results for RBF kernel with C=2 and gamma=0.1:
Accuracy: 0.9305555555555556
Confusion Matrix:
[[32  3]
 [ 2 35]]

Results for RBF kernel with C=2 and gamma=0.2:
Accuracy: 0.9027777777777778
Confusion Matrix:
[[30  5]
 [ 2 35]]
```

**<u>C = 12</u>**



```
Results for RBF kernel with C=12 and gamma=0.01:
Accuracy: 0.9583333333333334
Confusion Matrix:
[[34  1]
 [ 2 35]]

Results for RBF kernel with C=12 and gamma=0.1:
Accuracy: 0.9166666666666666
Confusion Matrix:
[[31  4]
 [ 2 35]]

Results for RBF kernel with C=12 and gamma=0.2:
Accuracy: 0.9027777777777778
Confusion Matrix:
[[30  5]
 [ 2 35]]
```

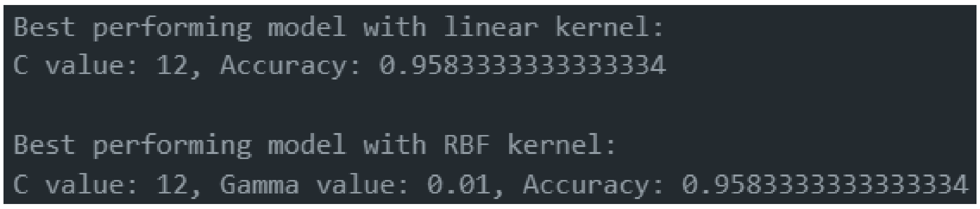
**<u>C = 24</u>**



```
Results for RBF kernel with C=24 and gamma=0.01:
Accuracy: 0.9444444444444444
Confusion Matrix:
[[34  1]
 [ 3 34]]

Results for RBF kernel with C=24 and gamma=0.1:
Accuracy: 0.9166666666666666
Confusion Matrix:
[[31  4]
 [ 2 35]]

Results for RBF kernel with C=24 and gamma=0.2:
Accuracy: 0.9027777777777778
Confusion Matrix:
[[30  5]
 [ 2 35]]
```

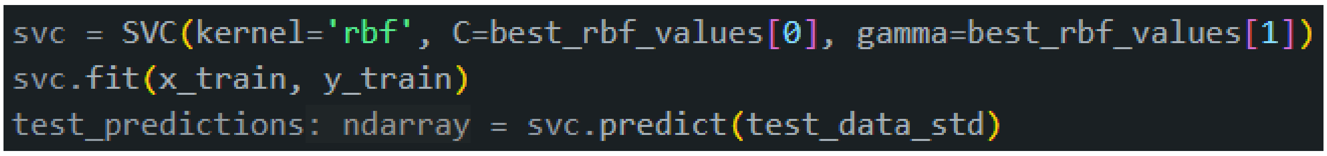
Above, there are 3 images, each containing the 3 separately tested rbf kernals, each with C value, is tested with each gamma value, this ensures a rigorous test which will output the best combination using the rbf kernel.

**B)** As seen, the highest accuracy scoring model uses a C value of 12, and a Gamma value of 0.01. To validate this I use the Max() function to retrieve the highest performing model by referring to the Key=results.



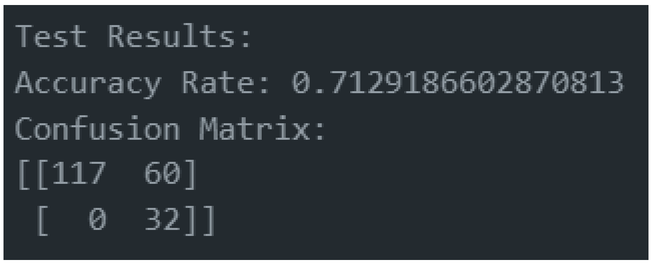
```
Best performing model with linear kernel:
C value: 12, Accuracy: 0.9583333333333334

Best performing model with RBF kernel:
C value: 12, Gamma value: 0.01, Accuracy: 0.9583333333333334
```

**C)** To test the optimal model, I will quickly preprocess the test data, using the same methods as previously used on the training data, to ensure the shape of the data is identical to the data used to train the model. This is important as it means the model produces an accurate representation of the accuracy.



```
svc = SVC(kernel='rbf', C=best_rbf_values[0], gamma=best_rbf_values[1])
svc.fit(x_train, y_train)
test_predictions: ndarray = svc.predict(test_data_std)
```

I use the 2 previously stored "best_rbf_values" Which contain both the C value and the Gamma, used in the most optimal model generated within the evaluation stage performed in 3a. Once again the model is trained using the same training data to ensure consistency.



```
Test Results:
Accuracy Rate: 0.7129186602870813
Confusion Matrix:
[[117  60]
 [  0  32]]
```

The results produced raise several questions. Firstly, let us explain the contents of the confusion matrix. In the top left, the correctly classified benign tumours, in the bottom left, the incorrectly classified Malignant tumours, in the top right, the incorrectly classified benign tumours, and finally, in the bottom right, the correctly classified Malignant tumours.

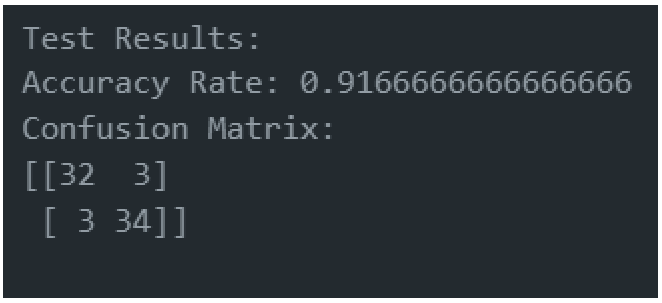
The system can recognise when a malignant tumour is malignant, however, it seems it gets confused roughly 30% of the time when classifying the Benign tumours, this is reflected in the accuracy rate of the model. This could be because there are many more data entries classified as benign compared to malignant. Since it is not a very balanced dataset there tend to be inconsistencies like this.

## TASK 4- SVM WITH REDUCED FEATURES

**A)** Based on the scree plot, seen in Task 1, an as mentioned before, I would reduce the features down to 10, which would include all of the Mean values. I think it is important to still include all of the features, as even a small variation can effect trends, and could hugely benefit the model, I just don't think it is as important to have the 3 variations of each feature, like provided in the full data.

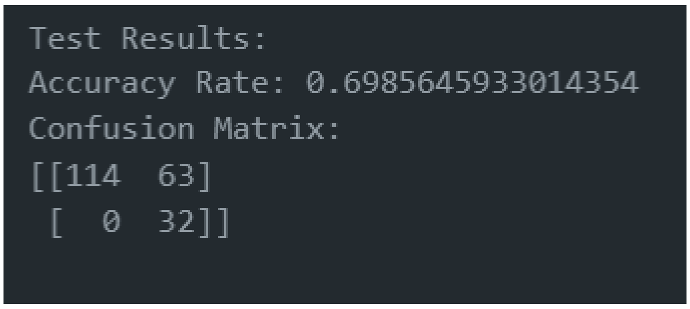
**B&C)** So, continuing with the first 10 features, I selected the first 10 columns of the train_data, and fit_transform() it with StandardScaler() from sklearn. Next, I split the reduced_train_data_scaled into training and validation data, in an 8:2 ratio. I then formed an SVC with the 'rbf' kernel, using our optimised c and gamma values from the past. Fitting it with the train data, and The training label

Its tested with the validation set, which produces the following results:



```
Test Results:
Accuracy Rate: 0.9166666666666666
Confusion Matrix:
[[32  3]
 [ 3 34]]
```

With relatively promising results, we can hope that when testing the model on also reduced test data, there is a good outcome.



```
Test Results:
Accuracy Rate: 0.6985645933014354
Confusion Matrix:
[[114  63]
 [  0  32]]
```

Surprisingly, the model performs less optimally on the reduced data opposed to the model trained with all 30 features. It should always be considered that more data is always beneficial, even if seemingly irrelevant.