[redacted]

Q1)
From the code, we found that the weighted-averaging method resulted in higher values for all three of precision, recall, and F-score than the macro-averaging method. The macro-averaging method basically takes the precisions and the recalls of all the classes in the dataset, whereas the weighted-averaging method sums up the classes' precision/recall multiplied by the number of instances that are in that class out of the total number of instances. The weighted-averaging method seems to incur a higher precision and recall, and therefore F-score, than the macro-averaging method because rather than simply taking the means it values each class' precision and recall according to its proportion size in the dataset so that there is less generalisability and therefore more accurately portraying the dataset, further proven with the higher F-score value.

Q2)
For this assignment, we were told to assume that "*numeric attributes are Gaussian-distributed*". However, this is not always true as not all data in the world are Gaussian-distributed. The same goes for the data provided in the training dataset.

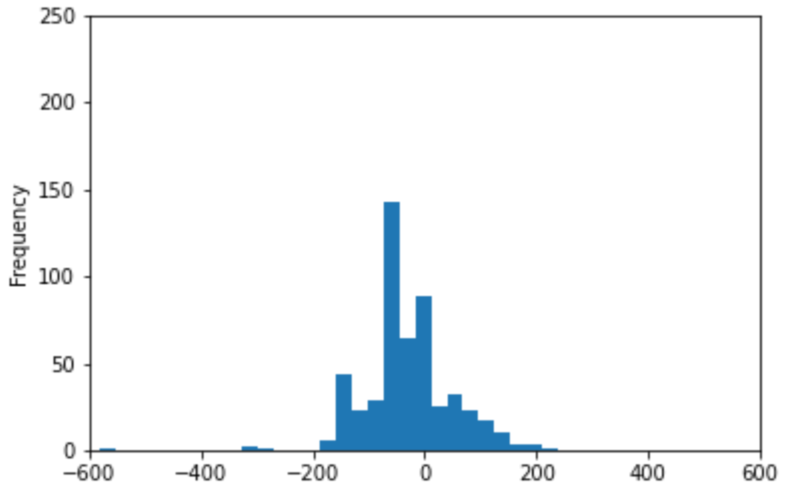


Fig.1 - Graphical Representation of wristR Points (x4)

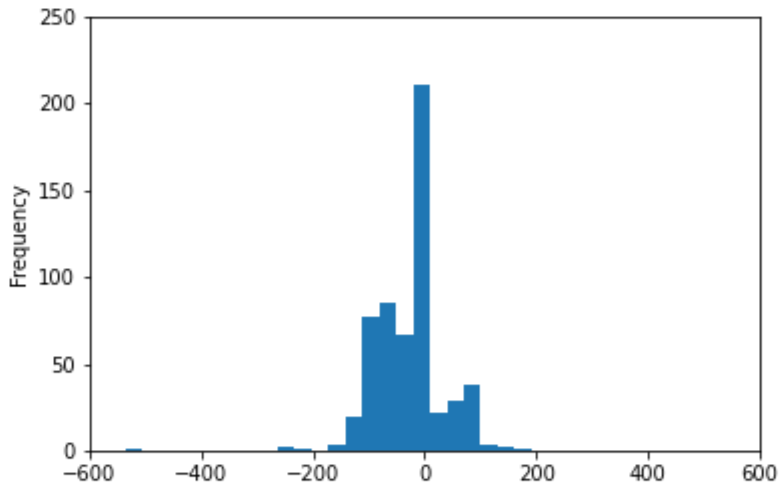


Fig.2 - Graphical Representation of kneeR Points (x8)

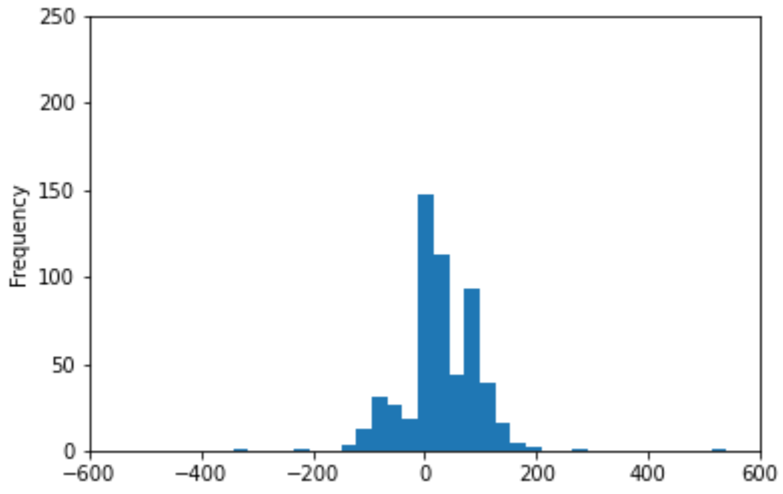


Fig.3 - Graphical Representation of kneeL Points (x10)

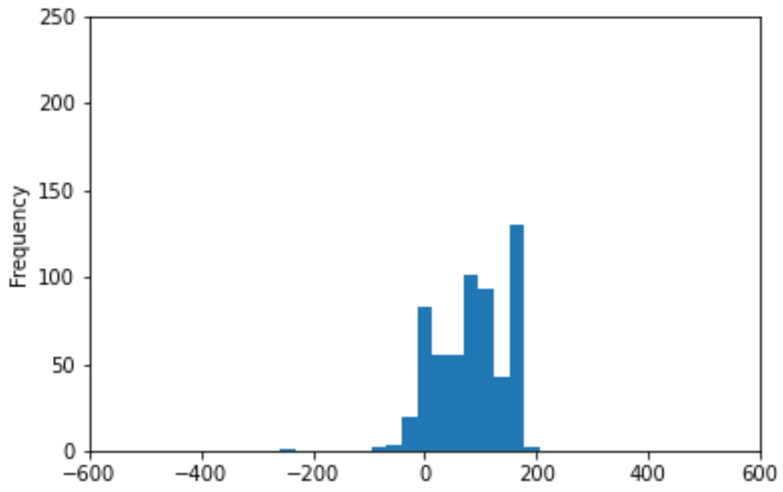


Fig.4 - Graphical Representation of head Points (y1)

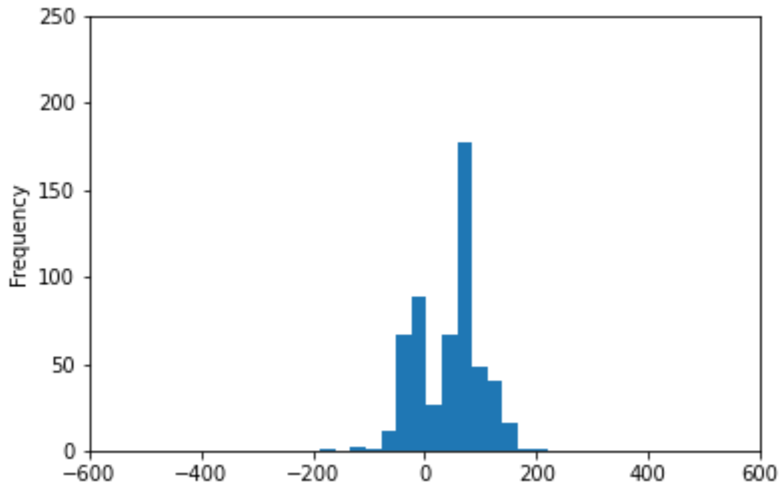


Fig.5 - Graphical Representation of elbowL Points (y5)

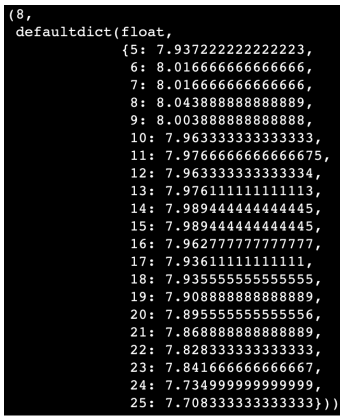
From the 5 figures above (which were the standouts that did not visually represent a Gaussian distribution among the 22 unique attributes from the data set), we can visually observe that the data are not Gaussian-distributed. This would negatively impact the overall precision of the classifier as the training data, which we have assumed to be Gaussian-distributed, would be skewed (positively or negatively depending on the attribute). As a result of that, our classifier would incorrectly predict classes.

Q3)

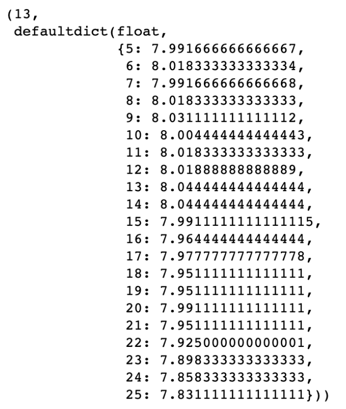
The normal Naïve Bayes model takes the sample mean and sample standard deviation of the training set of each class attribute into the Gaussian distribution likelihood. The KDE Naïve Bayes model uses a different likelihood which given a class consisted of the sum of Gaussian distributions using the residual between the test attribute instance and the value of that attribute from every instance in the training set for each attribute, each with a mean of 0 a standard deviation (kernel) of our choice, divided by the number of valid instances for each class attribute. Our KDE Naïve Bayes model resulted in a higher accuracy than our normal Naïve Bayes model for all kernel values between 5-25, with a highest at 77.59%. This would be because the KDE method does not straight away assume that the attributes values all follow a normal distribution but instead learns and uses the true distribution of the values by going through every single one of them to obtain a more reliable result.

Q4)

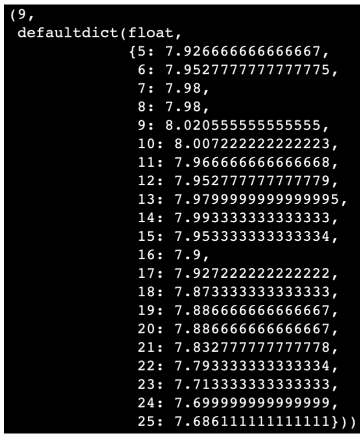
Cross validation was used to find the optimal kernel bandwidth. The dataset is partitioned into 10 folds, where each fold will act as a test set in their respective iterations. The data is trained on the 9 partitions and tested on the 1 test partition. The test partition is run through a KDE Naïve Bayes model multiple times, each with a kernel bandwidth of 5 increasing to 25. Since the code randomly partitions the dataset each time it is called, the optimal kernel bandwidth is subject to change. As a result, the code was run 10 times and the data is shown below:



```
(8,
 defaultdict(float,
             {5: 7.937222222222223,
              6: 8.016666666666666,
              7: 8.016666666666666,
              8: 8.043888888888889,
              9: 8.003888888888888,
              10: 7.963333333333333,
              11: 7.9766666666666675,
              12: 7.963333333333334,
              13: 7.976111111111113,
              14: 7.989444444444445,
              15: 7.989444444444445,
              16: 7.962777777777777,
              17: 7.93611111111111,
              18: 7.935555555555555,
              19: 7.908888888888889,
              20: 7.895555555555556,
              21: 7.868888888888889,
              22: 7.828333333333333,
              23: 7.841666666666667,
              24: 7.734999999999999,
              25: 7.708333333333333}))
```



```
(13,
 defaultdict(float,
             {5: 7.991666666666667,
              6: 8.018333333333334,
              7: 7.991666666666668,
              8: 8.018333333333333,
              9: 8.031111111111112,
              10: 8.004444444444443,
              11: 8.018333333333333,
              12: 8.01888888888889,
              13: 8.044444444444444,
              14: 8.044444444444444,
              15: 7.9911111111111115,
              16: 7.964444444444444,
              17: 7.977777777777778,
              18: 7.951111111111111,
              19: 7.951111111111111,
              20: 7.991111111111111,
              21: 7.951111111111111,
              22: 7.925000000000001,
              23: 7.898333333333333,
              24: 7.858333333333333,
              25: 7.831111111111111}))
```



```
(9,
 defaultdict(float,
             {5: 7.926666666666667,
              6: 7.9527777777777775,
              7: 7.98,
              8: 7.98,
              9: 8.020555555555555,
              10: 8.007222222222223,
              11: 7.966666666666668,
              12: 7.952777777777779,
              13: 7.9799999999999995,
              14: 7.993333333333333,
              15: 7.953333333333334,
              16: 7.9,
              17: 7.927222222222222,
              18: 7.873333333333333,
              19: 7.886666666666667,
              20: 7.886666666666667,
              21: 7.832777777777778,
              22: 7.793333333333334,
              23: 7.713333333333333,
              24: 7.699999999999999,
              25: 7.686111111111111}))
```

Fig.6 - First 3 Outputs of Q4

Kernel Bandwidth (σ)	Frequency	Highest Accuracy (%)
6	1	80.0572
8	4	81.3833
9	4	81.1278
13	1	80.44444

Fig.7 - Summary of Outputs

Since both 8 and 9 have the same frequency, we conclude that 8 is the optimal kernel bandwidth as it obtained the highest frequency. Compared to an arbitrary kernel bandwidth, doing cross validation ensures that the bandwidth chosen is optimal. An arbitrary kernel is merely a random guess which might not yield the best result for our algorithm.