Q1: The k-nearest neighbours (KNN) algorithm is a supervised machine learning algorithm. It relies on labelled input data to learn a function that produces an appropriate output when given new unlabelled data. It's considered a non-parametric method because it doesn't make any assumptions about the underlying data distribution. Simply put, KNN tries to determine what group a data point belongs to by looking at the data points around it.

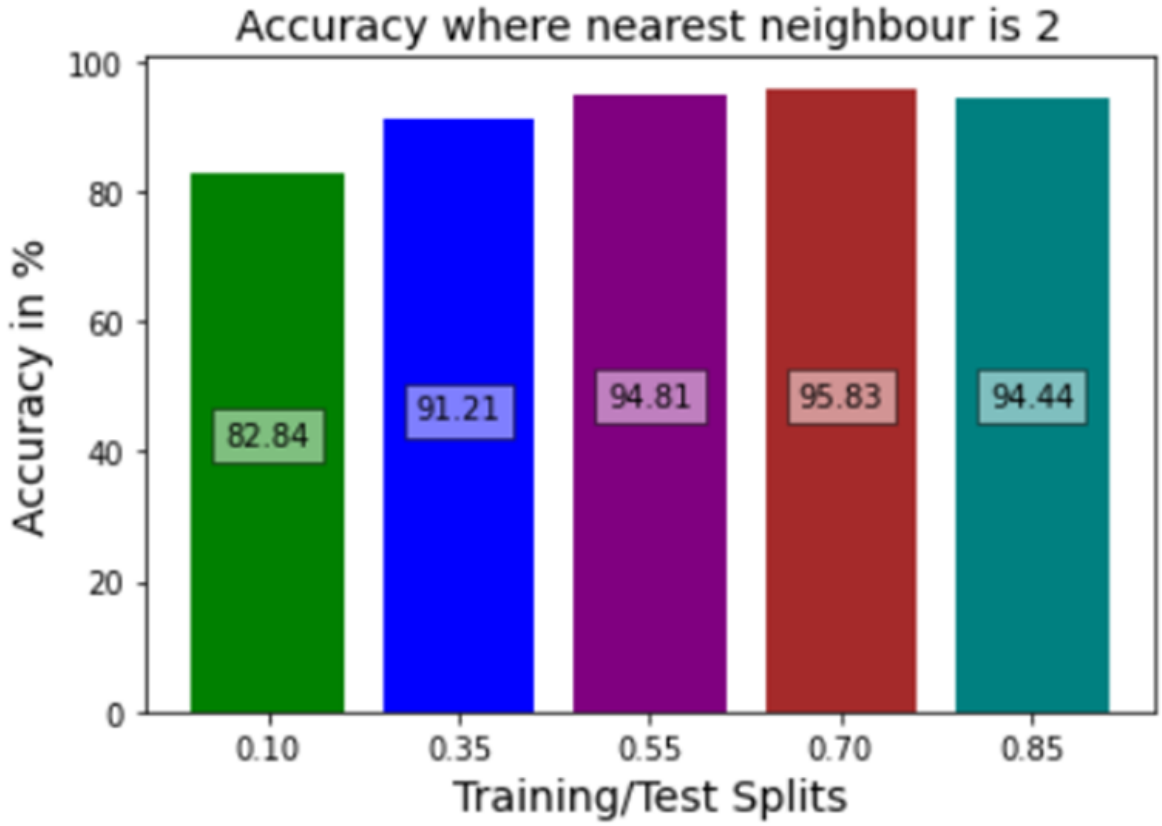
For this task, we will be adjusting the split which is the size of the training and test subset and varying the value of K. We will then plot the accuracy scores after introducing these parameter changes and discuss the results. To further explain the K value in this scenario, K is the size of the collection of the nearest neighbour used for the prediction. If the value of K is equal to one, then we'll use only the nearest neighbour to determine the class of a data point. If the value of K is equal to ten, then we'll use the ten nearest neighbours, and so on.

(i) We have systematically varied the size of the splits for the Knn. Throughout this portion of the testing, the nearest neighbour, (value of k) was kept at a constant of 2. These are the 5 split values that were used:
0.10, 0.35, 0.55, 0.70, 0.85

The accuracy scores that were observed from using these splits are seen in the dataframe below:

Split	Accuracy in %
0.10	82.84
0.35	91.21
0.55	94.81
0.70	95.83
0.85	94.44

The accuracy scores for each of the splits have been plotted in the bar chart below.



From the above graph, we can see that a split of 70% training data performs the best, giving us an accuracy score of **95.83%**. The split of 10% training data gives us the worst accuracy score of **82.84%**. Based on these accuracy scores, I have come to the following conclusions as to why the graph has changed in this way when the parameter is varied.

It appears that the less training data that is used in the training of the model, the less accurate the model is to classify data. This is evident from the graph where we can see the two lowest accuracy scores come from the splits where only 10% and 35% of the data is used to train the model. These splits achieved a score of 82.84% and 91.21% accuracy respectively.
Similarly, the accuracy of the model increases when we provide more training data for the model, this is seen as our three top performing splits come from our highest splits; 0.70, 0.55 and 0.85.

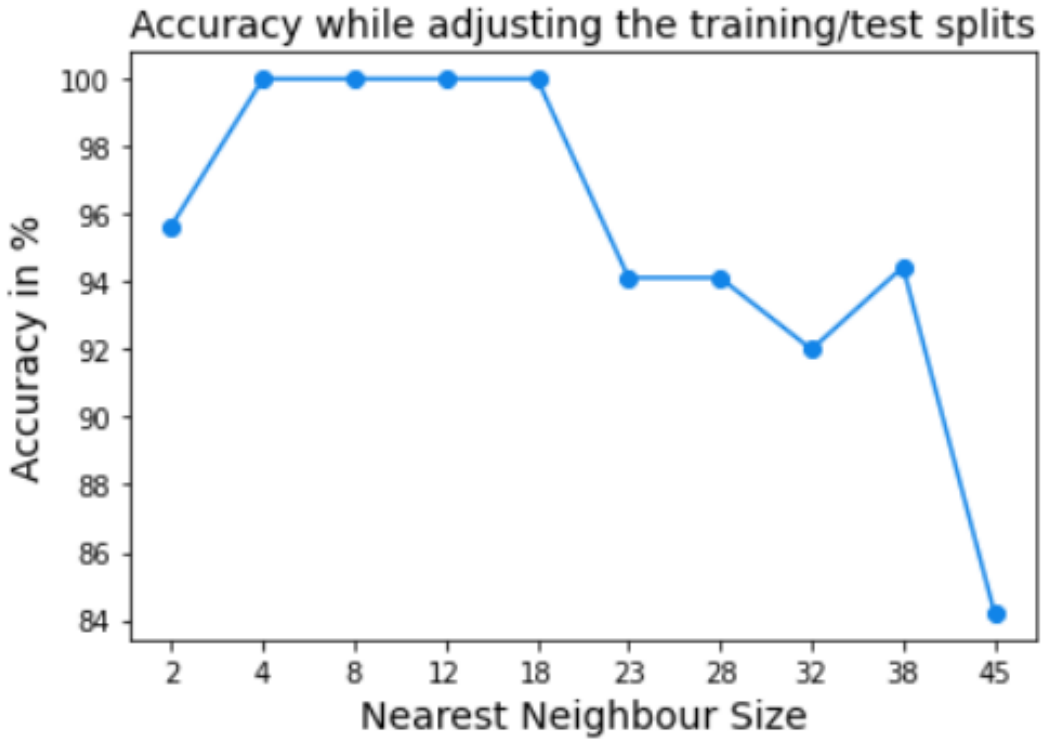
There are dangers involved with train/test split, where it might happen that the split we make isn't random. To mitigate this, we can perform **Cross Validation**, there are various cross validation methods, however we will briefly discuss **K-Folds Cross Validation**. In K-Folds Cross Validation we split our data into a number of different subsets (we will call it k). We use k-1 subsets to train our data and leave the last subset (or the last fold) as test data. We then average the model against each of the folds and then finalize our model. Once that is done, we test it against the test set.

(ii) We have systematically varied the size of "**k**" (the nearest neighbour). Throughout this section of the testing, the split was kept at a default of 0.9. These are the 10 "**k**" values that were used:
2, 4, 8, 12, 18, 23, 28, 32, 38, 45

The accuracy scores that were observed from using these "**k**" values are seen in the dataframe below:

K Scores	Accuracy in %
2	95.65
4	100.00
8	100.00
12	100.00
18	100.00
23	94.12
28	94.12
32	92.00
38	94.44
45	84.21

The accuracy scores for each of the "**k**" values have been plotted in the scatter plot below.



The graph above indicates that best performing "**k**" values appear when k = 4, 8, 12 & 18, each giving an accuracy percentage of **100%**. The worst performing "**k**" value is 45 where the accuracy percentage is **84.21%**. Based on these accuracy scores, I have come to the following conclusions as to why the disparity in scores exist when this parameter is varied.

As the "**k**" value in k-nearest neighbour increases, the accuracy decreases after a point. This is due to many more neighbours being tagged that are next to each other in order to satisfy the "**k**" value. As a result, this has caused underfitting of the data. The large value of "**k**" means that we have created a simple model with low variance and high bias. As an example, from the above graph, the "**k**" value of 45 returns the lowest accuracy score of 84.21%.

Similarly, if the value of "**k**" is too small, we get a complex model that can overfit the data and has high variance with low bias. This can be seen when the "**k**" value is at 2 and receives an accuracy score of **95.65%**. Based on the graph above, it appears a "**k**" value of between 4 and 18 is the most optimal value of "**k**".

This dataset most likely has high variance, given the large jump in accuracy based on different values. This means that the data is spread out and can result in overfitting the model. **Cross validation** is used to help negate this overfitting, by rotating the training and validation sets and training more.

Q2: Naïve Bayes Classifier is a probabilistic machine learning model that's used for classification tasks. The crux of the classifier is based upon the Bayes theorem which can be seen below.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using the above Bayes theorem, we can find the probability of A happening, given that B has occurred. They are fast and easy to implement but their biggest disadvantage is that the requirement of predictors to be independent. In most of the real-life cases, the predictors are dependent, this hinders the performance of the classifier.

We have implemented Naïve Bayes Classifier that does predictions on male/female names. The results below are obtained from using the existing feature based off of the last letter in the name.

```
------ Last letter ------
Anthony is: female
Garret is: male
Cassey is: female
Susanne is: female
Most Informative Features
             last_letter = 'a'            female : male   =     36.9 : 1.0
             last_letter = 'k'              male : female =     32.3 : 1.0
             last_letter = 'f'              male : female =     16.0 : 1.0
             last_letter = 'p'              male : female =     11.9 : 1.0
             last_letter = 'd'              male : female =     10.0 : 1.0
The accuracy is: 0.75
```

We can see from the above that using this feature, the algorithm correctly identifies **“Garret” as male, “Cassey” as female and “Susanne” as female**. Although it fails to classify “**Anthony**” correctly and recognises them as a female.

From the **most informative features**, we can see that the last letter, ‘a’ is female 36.9 times more often than they are male. The last letter ‘k’ is male 32.3 times more often than it is female. These are known as **likelihood ratios** and are very useful for comparing feature outcomes relationships such as above. The results also show us that the last letter feature above, produces an accuracy of 75%.

The new feature I have implemented involves extracting the middle character(s) of a given name. If the length of the name is odd, it will return the middle character and if the length of the name is even, it will return the middle two characters. A snippet of the feature can be seen below:

```
def middle_character(word):
    """If the length of the string is odd return the middle character
    and return the middle two characters if the string length is even"""
    return {'Middle Character': word[(len(word)-1)//2:(len(word)+2)//2]}
```

When the NLTK Naïve Bayes Classifier is ran with the above feature, the following results are observed:

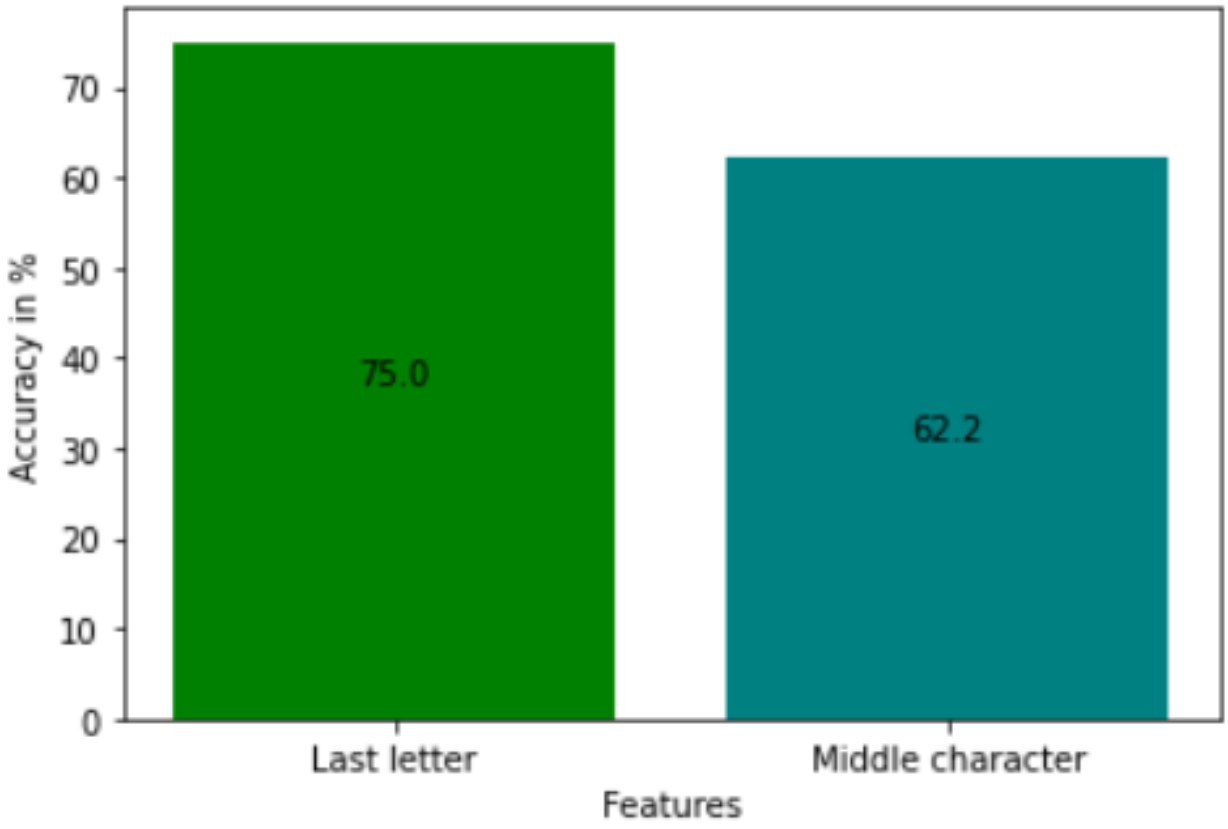
```
------ Middle character ------
Anthony is: male
Garret is: female
Cassey is: female
Susanne is: female
Most Informative Features
        Middle Character = 'lm'             male : female =     11.7 : 1.0
        Middle Character = 'lt'             male : female =      8.3 : 1.0
        Middle Character = 'nr'             male : female =      8.3 : 1.0
        Middle Character = 'sc'             male : female =      7.2 : 1.0
        Middle Character = 'va'             male : female =      7.2 : 1.0
The accuracy is: 0.622
```

We can see from the above that the algorithm correctly identifies **“Anthony” as male, “Cassey” as female and “Susanne” as female**. Although it fails to classify **“Garret”** correctly and recognises them as a **female**.

If we look at the **likelihood ratios** from the **most informative features**, we can see that the middle characters ‘lm’ are male 11.7 times more often than they are female. The middle characters ‘lt’ and ‘nt’ are male 8.3 times more often than they are female. Lastly, the results show us that the feature above, produces an accuracy of **62.2%.**

In terms of the changes/differences between the existing last letter feature and the newly implemented middle character feature using the in-built “show_most_informative_features”, we can see that both features correctly classified 3 out of 4 genders correctly. The “Last letter” feature failed to identify “**Anthony**” as male, and the “Middle character” feature failed to identify “**Garret”** as male. The **likelihood ratios** found much more effective features for distinguishing the names’ genders for the last letter feature (‘a’ is female 36.9 times more vs ‘lm’ is male 11.7 times more). Overall, the last letter feature performed better than the middle character feature with 75% accuracy vs 62.2% accuracy.

Finally, please see the bar chart of accuracy below on the existing feature vs the newly implemented feature.



From the above graph, we can see that the “Last letter” feature provides better accuracy than our “Middle character” feature for classifying data (75% vs 62.2%). I believe the “Last letter” feature was able to achieve a better accuracy score due to its ability to better determine features it found most effective for identifying the names’ genders. As previously mentioned, the “Last letter” “show_most_informative_features” scores are much higher than that of the “Middle character” feature. This highlights that more meaningful features can produce better accuracy and in this scenario that predicting gender works better from using the last letter rather than the middle character(s).