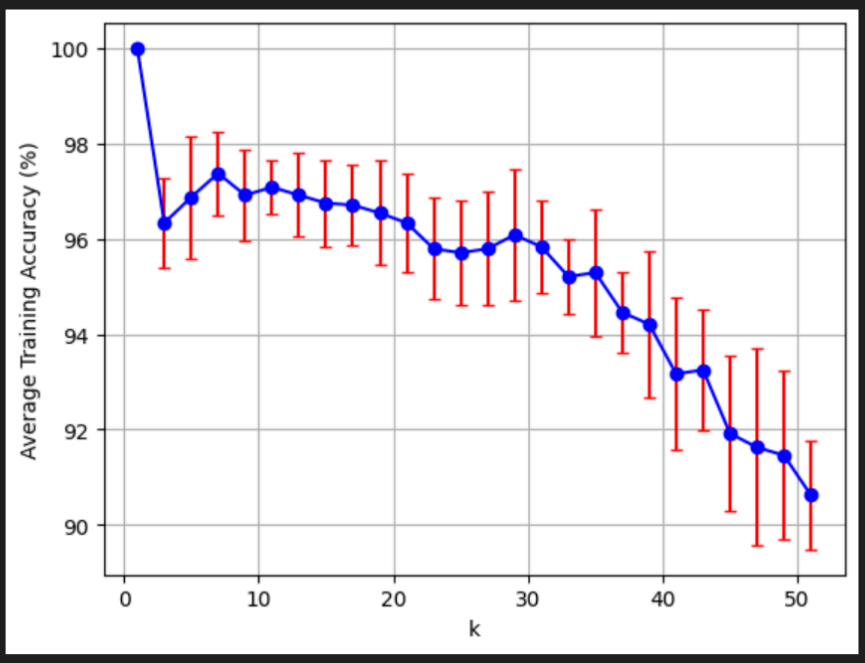
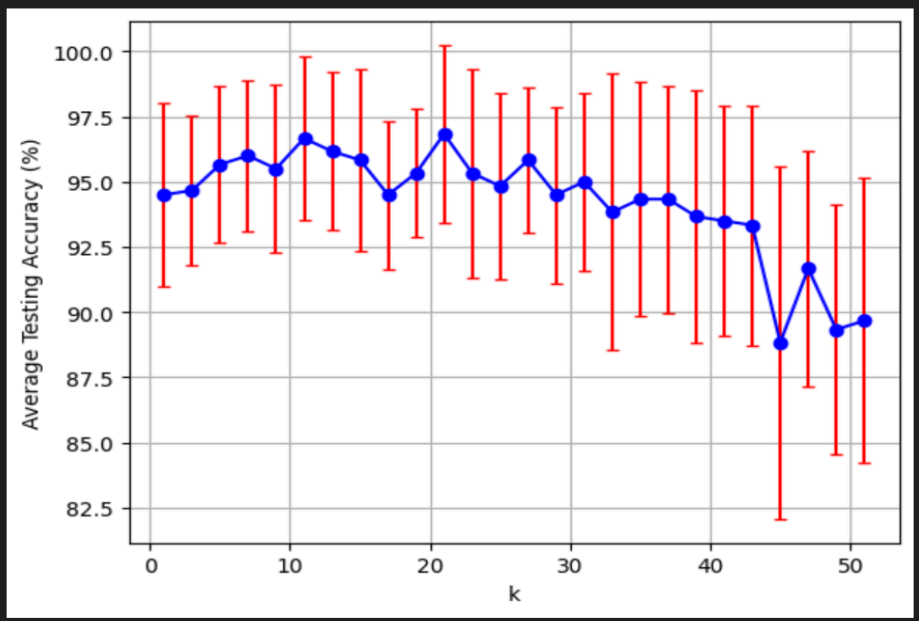
Shreya Birthare (34060222)

KNN

1.
```
Mean for K (1:52:2) (20 runs each) 95.3173076923077
Std Dev for K (1:52:2) (20 runs each) 0.40174380197506765
```


2.
```
Mean for K (1:52:2) (20 runs each) 94.21794871794873
Std Dev for K (1:52:2) (20 runs each) 1.005298677604804
```


3. Training Graph:

   From the graph we can see that OVERALL as K increases the average training decreases. It starts with showing 100% accuracy when k=1 and this makes sense too. If we are calculating the distance of the point in the training data with itself and returning 1-NN and predicting the result based on it, then it is

going to give 100% accuracy. The accuracy then drops quickly when we run the algo with k=2 implying it gives a combination of both correct & incorrect predictions. Max average accuracy is around 97.5% and is observed when k=8 approximately.For k=11, we see that it has the least deviations, suggesting many points are predicted pretty accurately with most points giving more or less the same average accuracy. The outliers & noise can be said to be minimum in this case. From k=11 to 21 we see a good average accuracy of approx 97% to 96.5% and this is very slowly declining. From k=12 to 20 we see good consistency in accuracy with extremely slow decline in the graph and with relatively the same size of the error bars (i.e. relatively same std deviation) indicating a possible plateau in the performance. For k=22 to 29 the average accuracy is almost constant and rises extremely slowly. Starting from k=30, the graph declines rapidly as we increase the value of k. Also, we can see significant standard deviation in the average accuracy. When k is finally 51, we see that the average accuracy drops to almost 91%. As we increased k gradually, it led to smoother decision boundaries; classification became less sensitive to noise & small fluctuations in the data because more neighbors are considered when making predictions. However, as we kept increasing the value of k it led to underfitting, the model became simplistic and couldn't capture the underlying patterns in the data fully.

Generally speaking I can say that the training data went from overfitting to giving good/optimal results to giving underfitted results as we kept increasing the value of k

Testing Graph:
The testing graph starts with k=1 and around 94.5% accuracy. The mean accuracy gradually increases till k=8 then gradually decreases then increases and finally comes to the same average accuracy of 94.5% when k=17 approx. It then rose again and reached its peak with max average accuracy as approx 97%. With gradual increase and decrease in the average accuracy up to k=35, the average accuracy almost becomes constant for 1-2 iterations, slowly then drops and then k=42 approx there is significant drop in the average accuracy and the standard deviation of the average accuracies increase greatly. The high overall average accuracy indicates that the model, on average, performs well on the testing set. Unlike the training accuracy, which showed a general downward trend with increasing k, the testing accuracy does not exhibit a clear and consistent trend. Instead, it fluctuates between k=1 and approximately k=20, the accuracy seems to be relatively stable with small fluctuations. This might suggest that within this range, the model is neither overfitting nor underfitting significantly. After k=20, there is more variability, and the graph shows a slight overall decline in accuracy, which could indicate the onset of underfitting as the model becomes too generalized. Throughout the graph, the error bars are quite large compared to the training graph, suggesting that the testing accuracy is more variable. This is typical since testing sets can differ more from the training data, leading to less consistent performance. The size of the error bars seems to increase slightly as k increases, which might suggest that the model's predictions become less reliable as we consider more neighbors.

4. We can say that for ranges of k from 20 onwards, our model is underfitting. From k=20 to 30, we see slight variations in the average accuracy, with less std dev compared to the testing accuracies for the same range. We start seeing almost equivalent test accuracy % as the training accuracy % with high std devs indicating that the model has started to generalize data points. For k=30 onwards till k=51, we see that the training accuracy starts declining quickly and the size of the error bars also start increasing indicating a higher amount of fluctuation in the average accuracies. Also, we see in our testing graph that for these ranges of k, average accuracy also declines with very high error bars indicating large std deviation in the average accuracies. For these ranges we can therefore say that the model is underfitting. For overfitting, we can see that the model overfits for smaller values of k ranging from k=1 to 7/8 approximately. We see that the model gives very high average accuracies for these values with

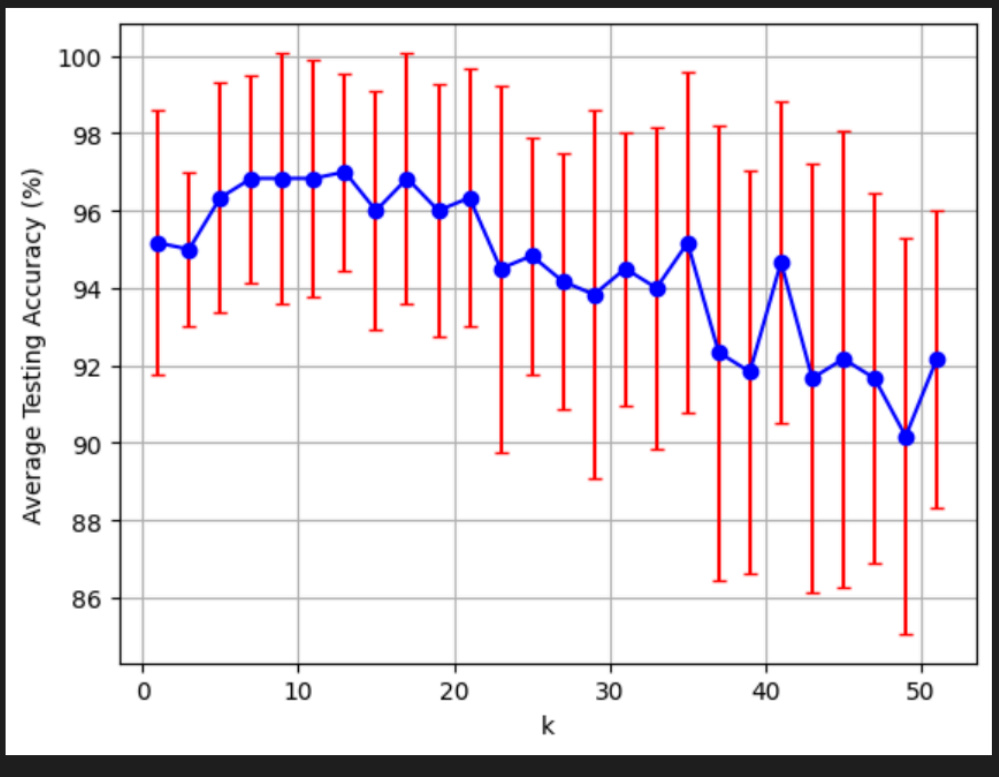
little std deviations in the accuracy when compared to the testing graph for the same values of k where we see less accuracies and high standard deviations indicating that the model is overfitting

5. I would choose k = 11 for fine tuning the algorithm as for it I see high average training accuracy (around 97%) with size of the error bars being small indicating that most of the accuracies revolve around the average accuracy. Also corresponding to that value of k I see that the testing average accuracy is also very high (around 97%). And from the error bar I can see that the accuracies for k=11 on testing ranges from 93% to 100% with most of the accuracies being on the higher end.

6. We train and test the algorithm without normalizing the features this time. We do all the things as usual except for the normalization part and plot the graph for the test

a.

```
Mean for K (1:52:2) (20 runs each) 94.49358974358974
Std Dev for K (1:52:2) (20 runs each) 1.0475458003165061
Without Normalization
```

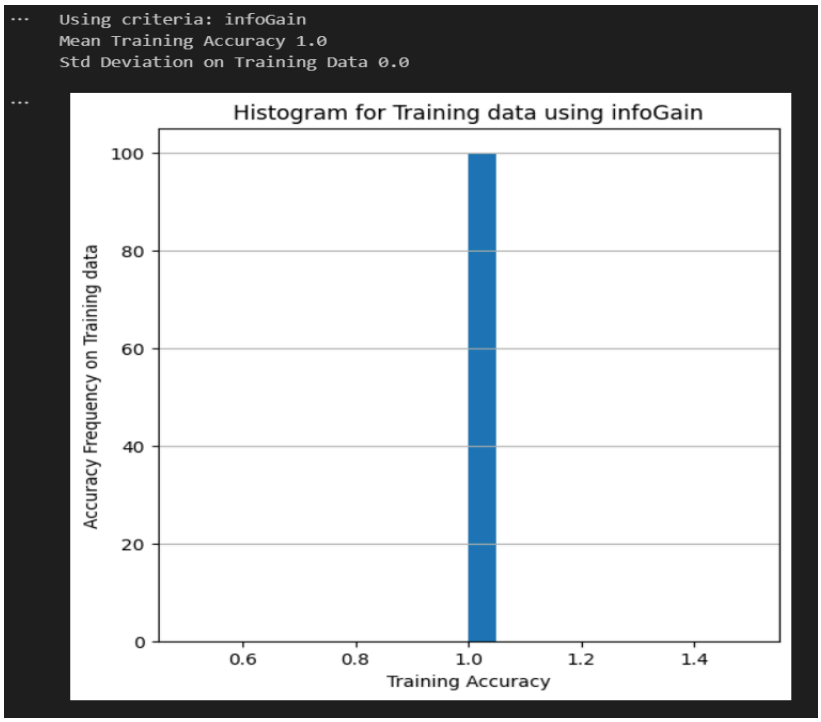


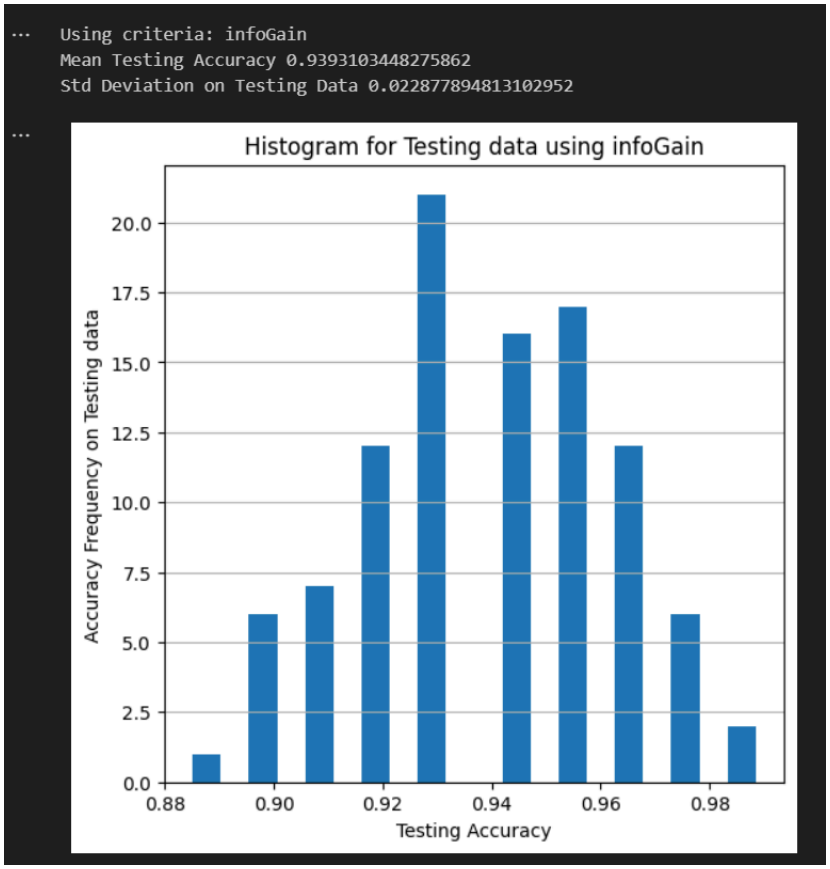
b. I would choose K=12 as the best value for this graph as I observe high test accuracy of almost 97%. Although the error bars for this graph are too high indicating that the std dev is too high, for k=12 we can see that the average accuracy ranges from 94.5% to 99% indicating a good overall average accuracy

c. In terms of average testing accuracies for both with and without normalization, there is not much difference. With normalization, I see an avg test accuracy of 94.21% and without normalization I see an avg test accuracy of 94.49% However, I see bigger error bars without normalization than with normalization. This indicates that there is quite a bit of variance in the accuracies. Some possible reasons for the performance difference could be: k-NN classifiers rely on distance metrics. When features are not normalized, one feature with a larger numerical range could disproportionately influence the distance calculation, causing the classifier to ignore other important features. The variability in performance without normalization suggests that feature scales are impacting the distance calculations, which can make the classifier sensitive to the

particular distribution and scale of the features in the dataset. Another possible reason could be that the dataset itself is simplistic and easily classifiable and has similar magnitude differences between data points for all columns that's why we don't see much significant difference in the average accuracies between the two graphs.

DECISION TREE

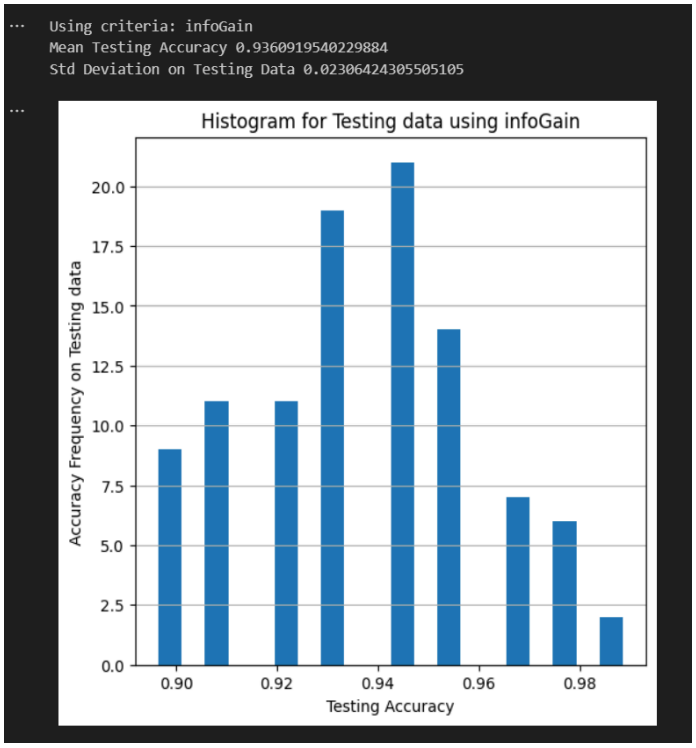
1. 

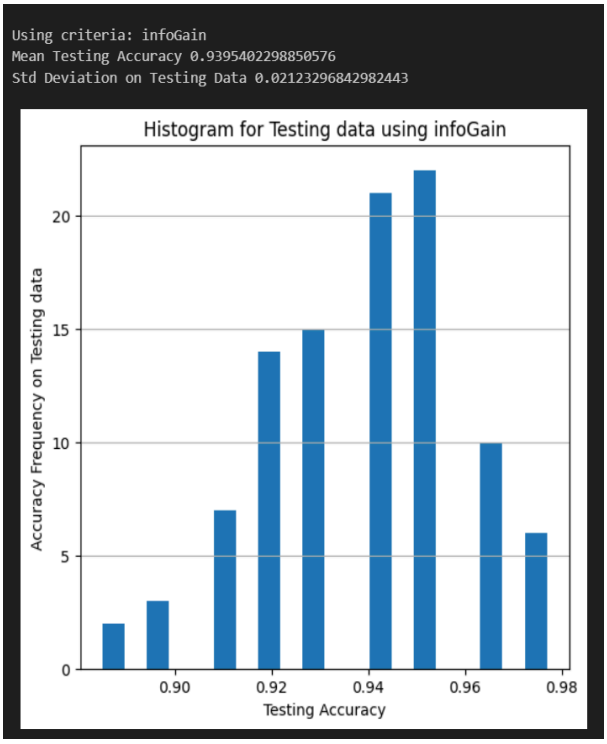
2. 

3. In the first graph we can see that the average training accuracy is 100% with zero standard deviation for all the 100 iterations. We can say that it is overfitting on the training data. This implies that the decision tree is memorizing the training data to give 100% accuracy. In the second graph, we see that the average test accuracy is approx 93.93% with standard deviation as 0.02 This graph obviously has more variance than the training graph and this is obvious as well. As new data comes in for testing. It will look at the trained tree. Depending on how we created the tree, it tries to find the best possible path for determining the result. It may be possible that the training data may not have the same path to give the expected result as the result that you'd observe with the testing data. That is why we see more variance in the testing data than the training data. Here we are seeing a decent training accuracy with reasonable std dev on our testing data. We can see that the testing accuracy ranges from 88%-98% approx and the frequencies of these accuracy are quite widely spread.

4. From the two graphs, I can say that the decision tree is overfitted. Because we observe a 100% testing accuracy indicating that the model is memorizing the training data to build the tree. When we test this tree with the testing data, we see the testing accuracy is quite low compared to the training accuracy and the accuracies are quite distributed, the variance is high. This implies that for a testing instance, the tree does not always have the correct path/branch that it could classify it to.

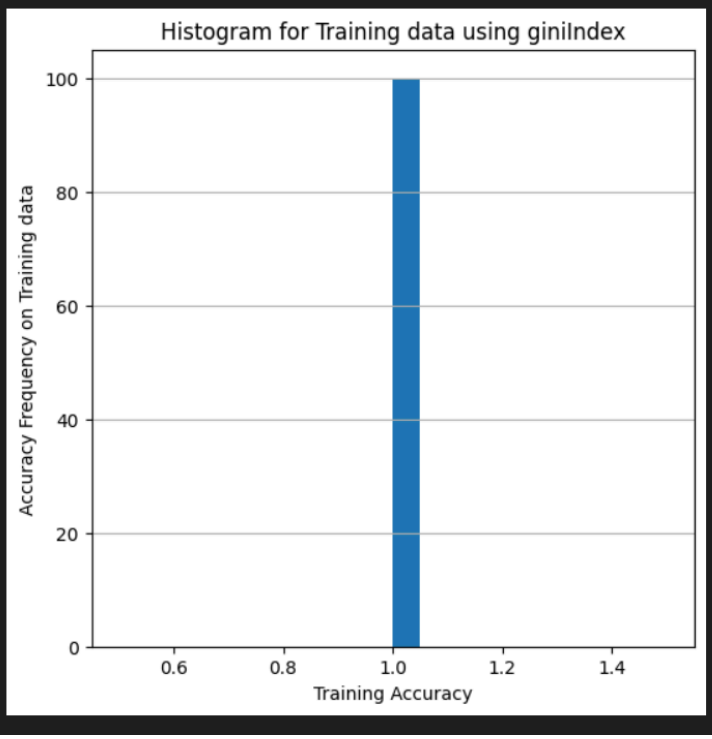
5. Decision trees are known to be non-robust in the sense that small changes in the data can lead to significantly different tree structures. This is because decision trees create hard boundaries based on the feature values, and a slight change in the training set can lead to different decisions about where to place these boundaries. To analyze this, we can print the tree and visually see that every time we run our code, we get a new tree with different attributes as nodes for splitting. This is because on shuffling and splitting the dataset everytime we get a new set of training data that the algo uses for building the tree. Additionally, when I re-run my algo I can see different set of accuracies, frequencies and standard deviations. I can keep re running my code multiple times and see every time i get a different range of accuracies, std devs and frequencies

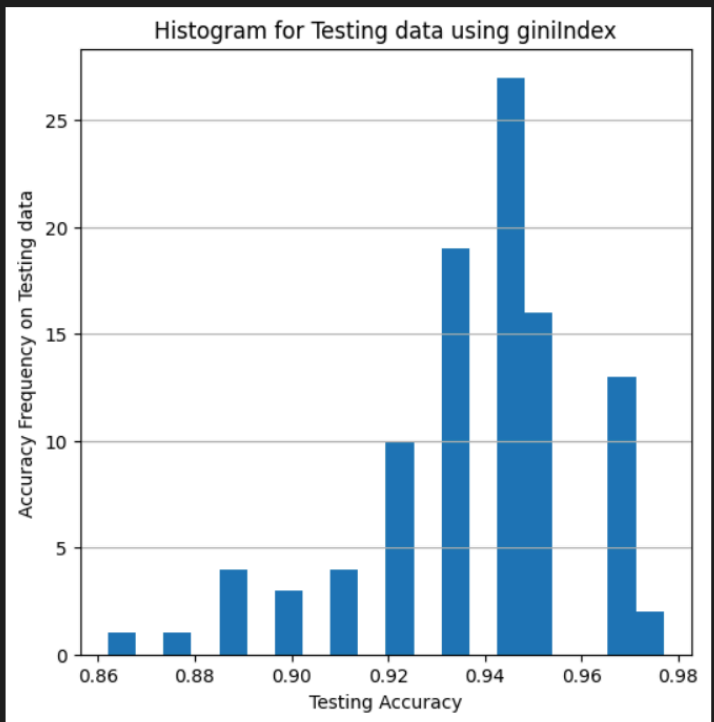




EXTRA POINTS

GINI INDEX: We use gini index now instead of information gain to train our tree.

1.
```
Using criteria: giniIndex
Mean Training Accuracy 1.0
Std Deviation on Training Data 0.0
```


2.
```
Using criteria: giniIndex
Mean Testing Accuracy 0.9370114942528734
Std Deviation on Testing Data 0.02272725110797804
```


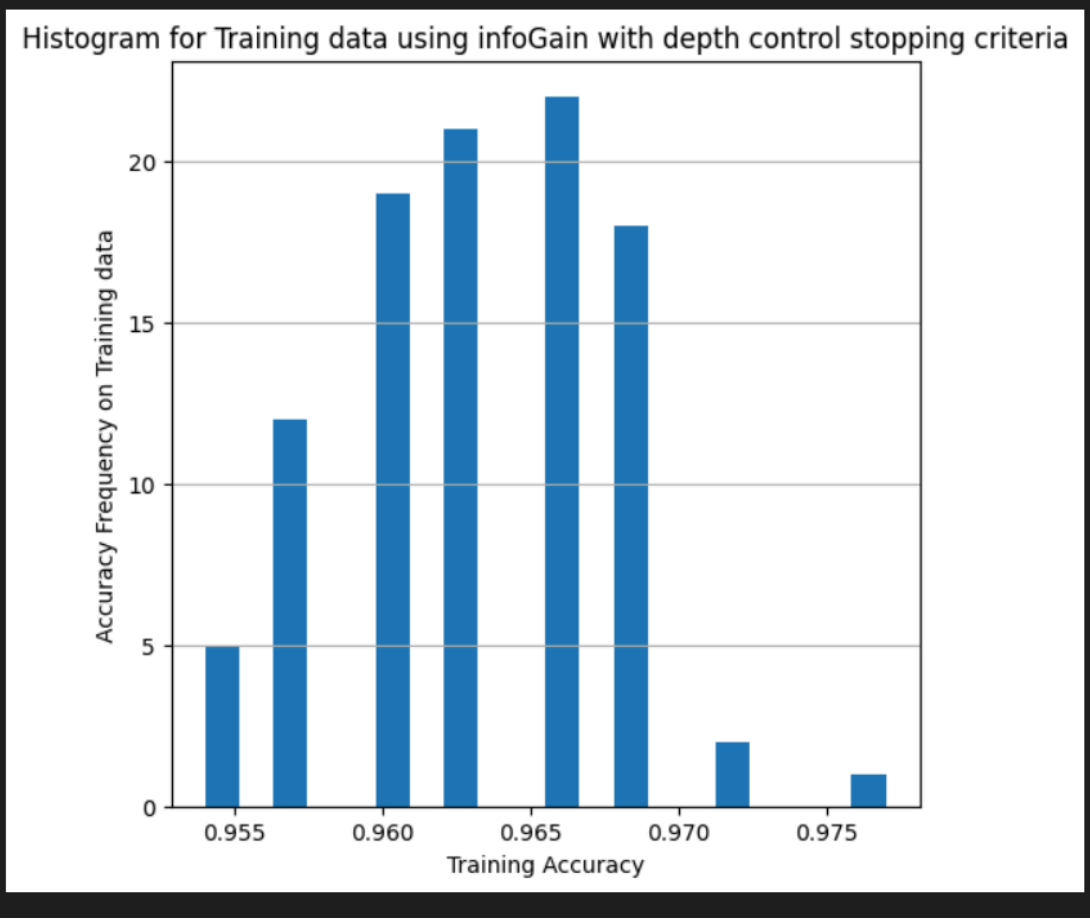
3. In this graph when we use the gini index, we see that the training accuracy is 100% still indicating overfitting. It implies that the model is memorizing the training data instead of generalizing it. For the testing graph we can see the accuracies are more spread out than the testing graph with info gain. Although the average accuracy and standard deviations are more or less the same only, and differ very less, here we can see that the graph has more variance as frequencies of different accuracies fluctuate between 96%-97%.

4. Here also, the model is overfitted because we observe a 100% testing accuracy indicating that the model is memorizing the training data to build the tree. As there is no provision to control the depth of the tree, the model learns the training data. When we test this tree with the testing data, we see the testing accuracy is quite low compared to the training accuracy and the accuracies are quite distributed, the variance is high. This implies that for a testing instance, the tree does not always have the correct path/branch that it could classify it to.

ADDITIONAL STOP CRITERIA

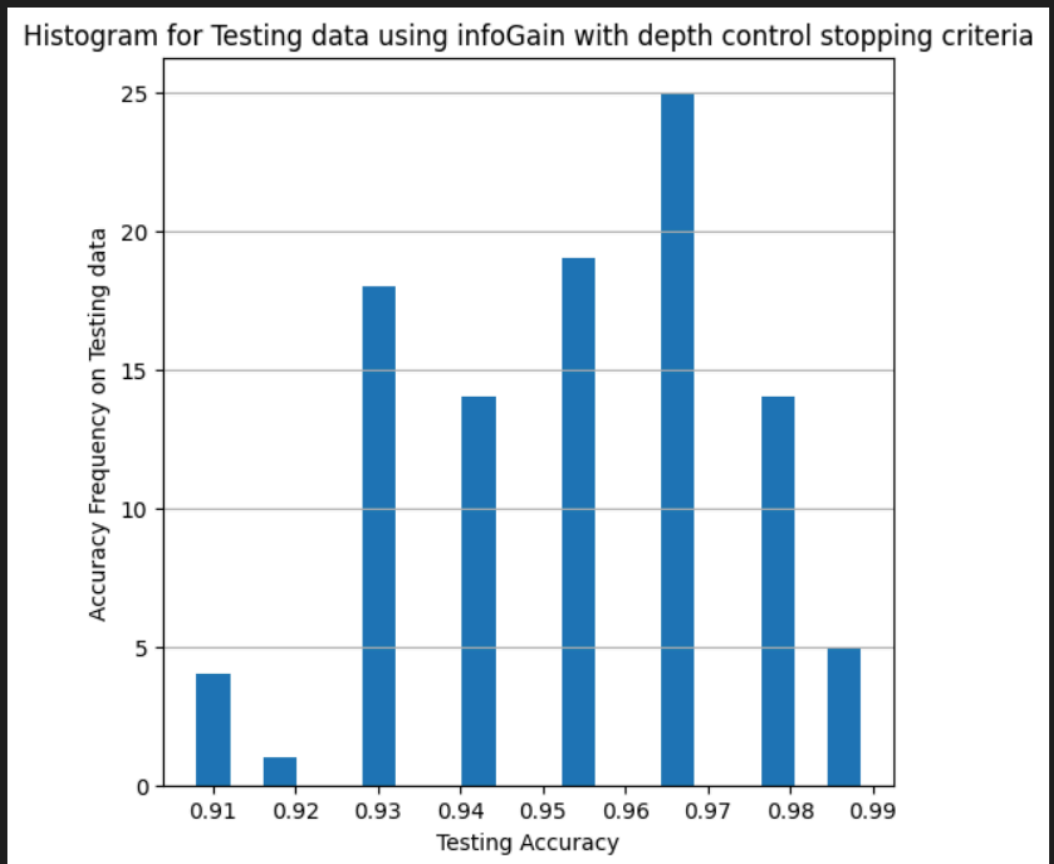
1.

```
Using criteria: infoGain
Mean Training Accuracy 0.9629597701149426
Std Deviation on Training Data 0.004523375525658068
```



2.

```
Using criteria: infoGain
Mean Testing Accuracy 0.9539080459770113
Std Deviation on Testing Data 0.019404191769077248
```



3. On using an additional stopping criteria we see an overall training accuracy of 96.29% with std dev as 0.004. This is significantly better than what we were originally observing without stopping criteria. Here the model does not completely memorize the training data like in the previous case. We see that the training accuracy ranges from 95-97% with most of the frequencies lying in the range of 96%-96.75% For the testing graph also, we can see it achieves a good accuracy of 95.39% with std dev 0.019. The avg accuracies range from 91% to 98.5% indicating that its quite spread out. However we see that a good amount of the test accuracies lie in the range of 95% to 97% These plots make sense as by adding additional stopping criteria we ensure that the tree doesn't grow too deep thereby preventing overfitting of data and generalizing our model.

4. Based on the two graphs, we can say that the model is performing reasonably well. The training and the test accuracies are almost similar and even their max frequencies lie in the same ranges only giving us more confidence in the testing. It means that the model is able to generalize information and predict correct results.