DALHOUSIE UNIVERSITY

Faculty of Computer Science

CSCI 6515 – Machine Learning for Big Data

Name: Arka Ghosh
Banner ID: B00911033
Assignment: 02

Task 1

In this assignment, I have worked with a dataset [1] that have 14 features among the last one is the "target" with integer values of 0 and 1 that indicates whether a person has heart disease or not. The explanation and the data types of dataset's features has been described in the following screenshots:

Features	Data Type	Explanation
age	Integer	Age of the person
sex	Integer	Gender of the person
cp	Integer	Chest pain types with 4 values (0-3) where each integer refers to different chest pain type
trestbps	Integer	Resting blood pressure
chol	Integer	Serum cholesterol in mg/dl
fbs	Integer	Fasting blood sugar in mg/dl
restecg	Integer	Three different values (0-2) indicating resting electro-cardio graphic results
thalach	Integer	Maximum heart rate
exang	Integer	Exercise induced angina
oldpeak	Float	Exercise-induced ST depression compared to rest
slope	Integer	Slope of the peak exercise
ca	Integer	Number of major vessels ranging from 0-3
thal	Integer	Three different values where 0 = normal, 1 = fixed defect, 2 = reversible defect
target	Integer	The integer value 1 and 0 in the "target" feature indicates presence of heart disease and no presence of heart disease respectively.

```
In [20]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

Figure 1: Information and Data types of the Dataset

The dataset contains 1025 instances in integer and floating values only. This means that there's no categorical data in this dataset. Additionally, the dataset is comparatively clean as there is no null values in the dataset. As the dataset is multi-dimensional, I have used Pearson Correlation technique to check if there are any features highly correlated which are linearly dependent on other features. While increasing the computational effort, these features provide a relatively small contribution to the prediction. Pearson correlation technique assigns a value between -1 to 1 [2]. A function named correlation_features() has been written within the code to check the correlated features. *Abs* been used in this function to take the absolute values as there will be negative values which will be converted into a positive one. The threshold has been set to 0.8 which means the features 80% or more correlated will be return to the function as output. A heatmap is also generated to show correlation of the features but none of the features are highly correlated which means none of the feature in this dataset is linearly highly dependent on other one which makes all of the features more or less equally important in predicting the output. This is why I haven't extracted any feature from the dataset. The heatmap to demonstrate the correlation between the features has been provided below:

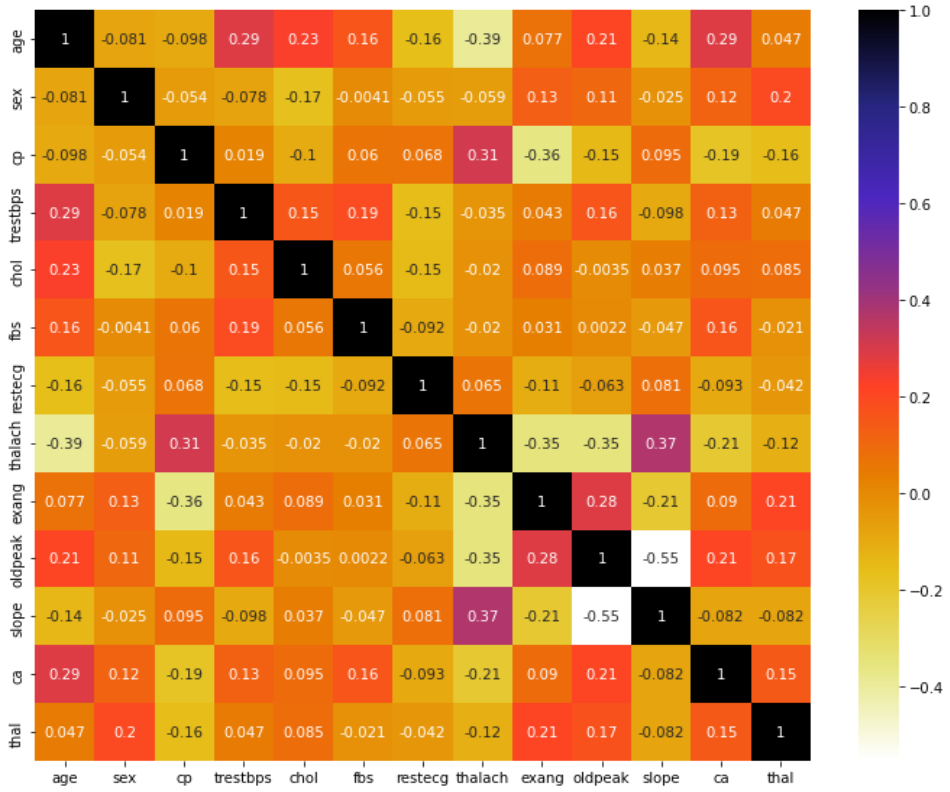


Figure 2: Heatmap to demonstrate the Correlation of the Features

Task 2

(i) K-Means algorithm is a centroid-based unsupervised learning algorithm that divides the unlabeled dataset into various cluster [3]. Here, K indicates the number of the clusters which can be pre-defined or determined by using techniques such as Elbow Method, Silhouette Score etc. For instance, if the value of k is 2, that means there will be two clusters. The steps [4] in the algorithm is explained below:

1. Choosing the number of Clusters (K): The first step is to choose the number of the clusters in which the data will be grouped. It can be chosen randomly or using Elbow Method or Silhouette score.
2. Initializing Centroids: Since the precise center of the data points is originally unknown, random data points are chosen and designated as the centroids for each cluster.
3. Assign data points to the nearest cluster: Suppose, there are X data points. In this step, each data points X_n is assigned to their closet Centroid C_k. In this step, the distance between the Data Point and Centroid is calculated using Euclidean Distance Metric.

$$d(x, y) = \sqrt{\sum_{n=1}^{n} (x_i - y_i)^2}$$

4. Re-initialize Centroids: Calculate the average of all the data points in that cluster, then re-initialize the centroids.
5. Up until the best centroids are chosen and the assignment of data points to the right clusters is stable, step 3 and 4 are repeated.

(ii) Using silhouette score, I have got the optimal number of cluster as 2. The plot of the silhouette score has been provided below:

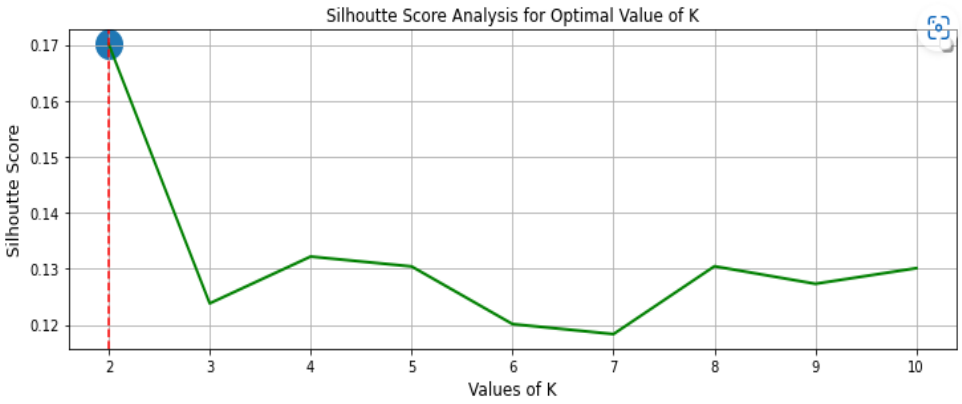


Figure: Plotted Silhouette Score Graph optimal Number of K

From the above screenshot, it can be seen that for k=2, silhouette score is the highest which is around 0.17. So, I have used two clusters to divide the data points. PCA has been used to visualize the clusters data points in 2D dimension and the screenshot has been provided below:

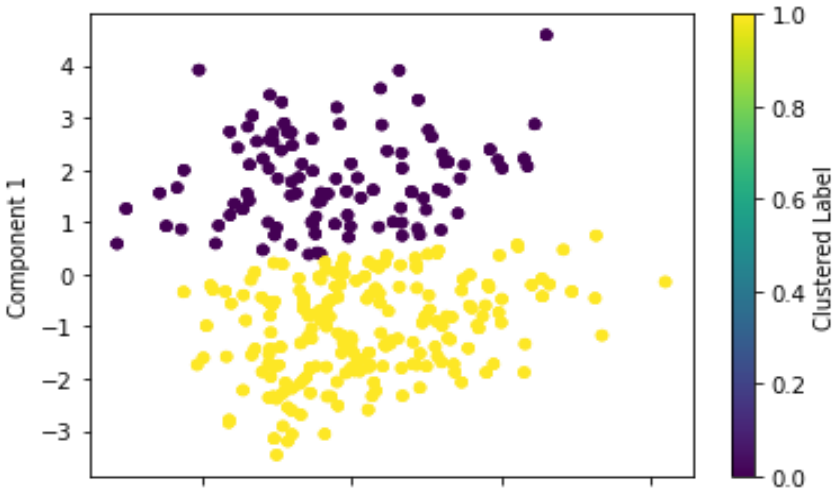


Figure: PCA Visualization of the Clusters

I have used the mean() function to see the average values of the features within each clusters which has been provided below:

Out[244]:

Cluster	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	0.430297	0.147411	0.691118	0.218553	0.129281	0.071454	0.117536	0.845277	0.800921	0.730127	0.663304	0.551720	0.400863
1	0.245131	0.083977	0.393715	0.124505	0.073648	0.040706	0.066958	0.481536	0.456267	0.415938	0.377870	0.314303	0.228363

From the above screenshot, it is noticeable that people with higher mean values of the features belong to the 1st cluster whereas people with the lower mean values of the feature data points belong to the 2nd cluster. It can be said from the above observation that people with heart disease are grouped in the first cluster.

(iii) The silhouette Measure is a technique for interpreting and validating consistency among data clusters and determining the ideal number of clusters [5]. Each point's silhouette coefficient, which quantifies how much a point resembles its own cluster in relation to other clusters, is computed using the silhouette measure. The silhouette's value ranges between [1, -1], with a high value indicating that the data point is well matched to its own cluster and is poorly matched to nearby clusters [6].

The silhouette score that I have got is shown below:

```
In [707]: silhouette_score(X, kmeans.labels_)

Out[707]: 0.1300985532372137
```

As the silhouette measure score is close to 0, it denotes overlapping clusters [5]. So, the data points that been used to create the clusters make it indifferent. In other words, we can say that that there is no significance distance between clusters created from the data points.

(iv) The clusters are not completely relatable to the labels of the dataset. The first 10 instances of the clusters and the labels of the dataset has been provided in the following screenshot:

```
In [724]: dataset_label = pd.DataFrame(final_dataset["target"])
          dataset_label.head(10)
```

Out[724]:

	target
0	0
1	0
2	0
3	0
4	0
5	1
6	0
7	0
8	0
9	0

```
In [639]: #Generated clustered Labels
          clus_label =pd.DataFrame(clustering_dataset["Cluster"])
          clus_label.head(10)
```

Out[639]:

	Cluster
0	0
1	1
2	1
3	0
4	1
5	0
6	1
7	1
8	0
9	1

Figure: First 10 rows of the Dataset Label and Clustered Label

From the above dataset it can be noticed that only 1 instance out of 10 in the original dataset is labelled as 1 and the rest are labelled as 0. On the other hand, 4 instances belong to first cluster and 6 instances belong to cluster second cluster. This clearly indicates that the clusters I have got are not related with the labels of the dataset.

I have used Rand Index to find the similarity between these two which is a measure to find the similarity between two clusters. There are two parameters [7] that are considered while calculating the Adjusted Rand Index. One is the labels_true that indicates ground truth labels to be used as a reference which is the label of the dataset in this case. Another one is the labels_pred that indicates the clusters labels to evaluate which is the clustered labels I have got in this case. Using the Rand index I have gotten a value of 0.364 which has been showed in the following screenshot:

```
In [599]: #Calculating the Rand Index between Clustered Labels and the Initial Labels of the dataset
          from sklearn.metrics.cluster import adjusted_rand_score
          print("The Adjusted Rand Index is:", adjusted_rand_score(labels_true, labels_pred))

          The Adjusted Rand Index is: 0.36409106000433716
```

Figure: Value of Rand Index

As we know Rand Index has a value between 0 and 1 where 0 means that there is no identical match between the two data clusters on any pair of points, and 1 means that they are precisely the same [8]. I have got a very poor Rand Index score of 0.364 which identifies that there are not enough significant match between the clusters and the labels of the dataset.

Task 3

(i) The dataset has been apportioned into train and test sets with 80-20 split for both Decision Tree and Naïve Bayes classifier where 80% of data has been used for training and 20% has been used for testing. I have kept the training data ratio higher as the classifiers can learn the traits and relationships between the features to predict the output more effectively when it is given with good number of data. The confusing matrix for Training and Testing sets of both the models has been provided below:

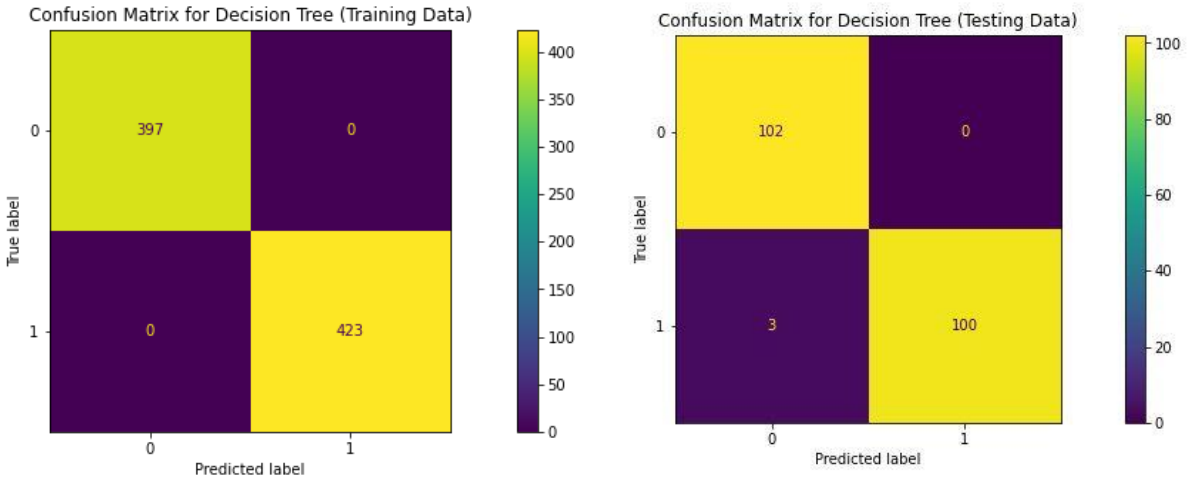


Figure: Confusion Matrix of DT for Training & Testing Data Data

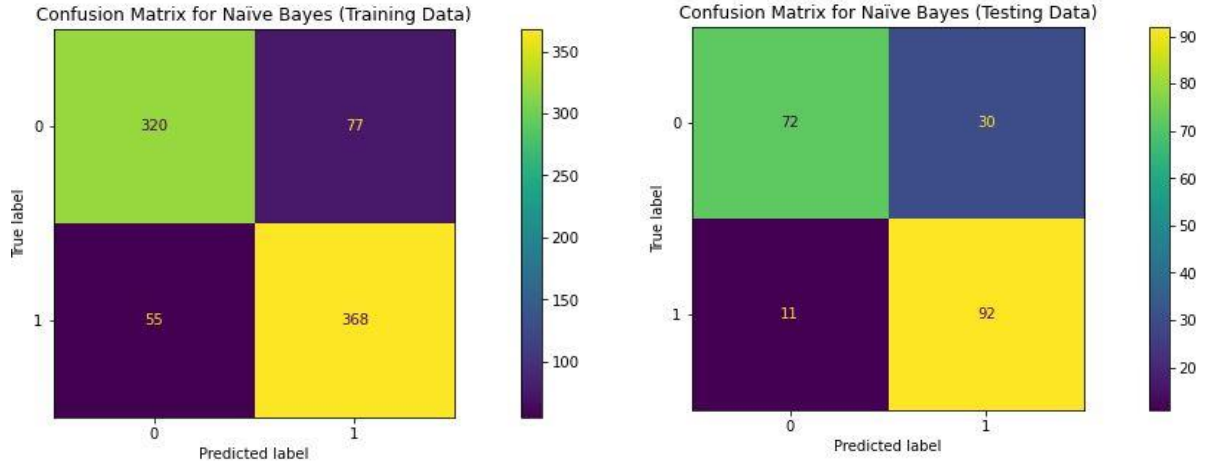


Figure: Confusion Matrix of NB for Training & Testing Data

From the above confusion matrix of Decision tree, it can be seen that the model classifies the data correctly included in training set. On the other hand, only 3 instances out of 205 data has been misclassified while testing. This is an indication that the dataset is well-balanced and as a result the model has learnt the features and relationship between the features well which has benefitted in predicting unforeseen data too effectively. However, Naïve Bayes model didn't work well on both the training and testing dataset which means that the dataset is not well suited for the model as it cannot discriminate the features and the relationship of the feature on the target variable very well.

(ii) I have used accuracy metric to evaluate both the models as it is classification task and the target variable of the dataset that has been used is quite balanced which can be seen from the following screenshot:

```
In [371]: final_dataset['target'].value_counts()

Out[371]: 1    526
          0    499
          Name: target, dtype: int64
```

Figure: Instances in the Target Class

So, out of 1025 instances, target variable has 526 instances which is labelled as "1" and other 499 instances are labelled as "0". This is an indication that the target class is quite well balanced and accuracy is well suited [9] in such cases which give the fraction of the predictions that the model has got right. Using this evaluation metric, the training and testing accuracy I have got for Decision Tree Classifier is 100% and 98.536% respectively. On the other hand, I have got 83.90% training accuracy and 80.0% testing accuracy for Naïve Bayes classifier.

```
In [452]: from sklearn import metrics

          #Acuracy of Training
          print("Training Accuracy of Decision Tree: ",metrics.accuracy_score(y_train, y_trainpred)*100)

          Training Accuracy of Decision Tree:  100.0

In [453]: #Acuracy of Testing
          print("Testing Accuracy of Decision Tree: ",metrics.accuracy_score(y_test, y_testpred)*100)

          Testing Accuracy of Decision Tree:  98.53658536585365

In [454]: print("Training Accuracy of Naive Bayes: ",metrics.accuracy_score(y_train, y_trainpred_NB)*100)

          Training Accuracy of Naive Bayes:  83.90243902439025

In [455]: #Acuracy of Testing
          print("Testing Accuracy of Naive Bayes:: ",metrics.accuracy_score(y_test, y_testpred_NB)*100)

          Testing Accuracy of Naive Bayes::  80.0
```

Figure: Training and Testing accuracy of DT and NB

(iii) Decision tree works the best between the two models by yielding to 98.536% testing accuracy. I have generated few scatter plots for some of the features from the dataset which has been provided below:

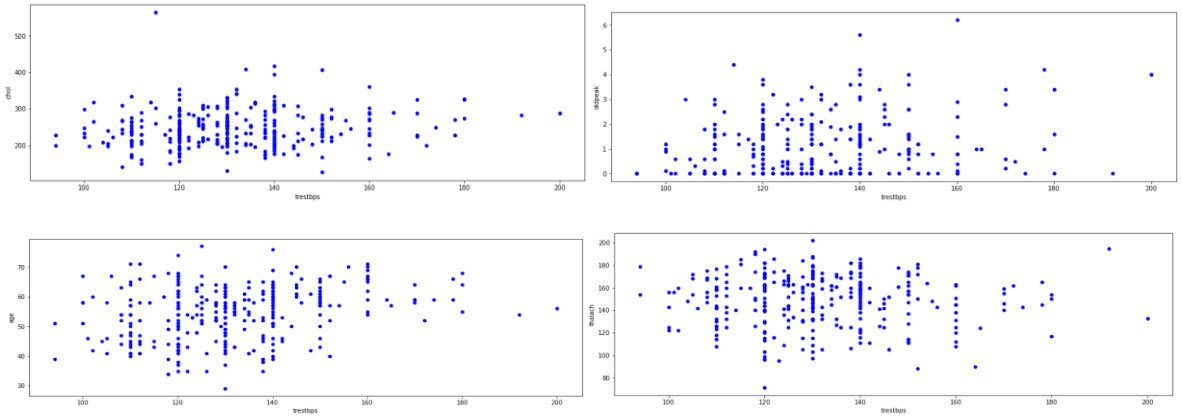


Figure: Scatter Plot of the features from the Dataset

From the above screenshots, it can be noticed that there's no direct line or relationship between the features variables that we have selected for the scatter plot. This indicates that the dataset is a non-linear dataset and Decision Tree Classifier is best suited for handling non-linear dataset [10]. This is why Decision Tree yielded better accuracy than Naïve Bayes. Additionally, decision tree is tree based algorithm that

improves the accuracy, stability, and simplicity of understanding of predictive modelling [11]. They map non-linear interactions pretty effectively in compared compared to linear modelling approaches such as Naïve Bayes, Linear Regression etc.

Statistical Significance Testing

To compare NB and DT, 10 cross validation has been applied to both the models. Then, using the difference received from both the models, Student's t test is done which is combined by random subsamples of the training dataset to compare the performance of different Machine Learning Models. The test's null hypothesis is that the performance of the two applied ML models is identical [12]. In other words, the null hypothesis supposes that the performance of both ML models is equal. The alternative hypothesis, on the other hand, contends that the performance of two applied ML models varies. In order to determine whether to reject the null hypothesis, P-values are used in hypothesis testing. The alternative hypothesis, which claims that the ML models perform differently, is accepted when the P-value is lower than the significance threshold which is 0.05 [13]. Furthermore, a P-value that exceeds the significance limits demonstrates that we are unable to reject the null hypothesis. The following screenshot provides the P value that I have got using Student's t test for both the model:

```
In [629]: #different between the results of 10-cross validation for DT and NB
          difference = [i-j for i,j in zip(DT_accuracy, NB_Accuracy)]

In [630]: #mean of the difference
          diff_mean = np.mean(difference)
          diff_mean

Out[630]: 0.17088330477822197

In [631]: #n_train = datapoints used for training
          #n_test_ = datapoints used for testing
          #n_final = total number of datapoints
          n_train_ = np.median(n_train)
          n_test_ = np.median(n_test)
          n_final = len(n)

In [632]: sigma2 = np.var(difference)
          #modified variance
          sigma2_mod = sigma2 * (1/n_final + n_test_/n_train_)
          sigma2_mod

Out[632]: 0.0004149819450945956

In [633]:
          t_static =  diff_mean / np.sqrt(sigma2_mod)
          t_static

Out[633]: 8.388513995155447

In [634]: Pvalue = ((1 - t.cdf(t_static,n_final-1))*200)
          Pvalue

Out[634]: 0.001512473162601502
```

Figure: Calculated Pvalue for Statistical Significance Testing

The P-value in this case is 0.0015, which is lower than the significance threshold 0.05, indicating that we can reject the null hypothesis. Therefore, the outcomes statistically provide compelling evidence that Decision Tree and Naïve Bayes perform differently.

(iv) All of the features in the dataset that has been used for the assignment are independent not strongly correlated with the other features. Also, it can be said from the Rand Index and Statistical Significance testing that the patterns found in the supervised models is not as same as the one presented in the clustering. Also, our Pvalue couldn't reject the null hypothesis which means that even both the supervised models will work differently for the dataset that's been used.

Summary of Results and Conclusion

Task 1

- The dataset that has been used in this assignment has both integer and floating data types. The dataset has 13 feature variables and 1 target variable
- A Heatmap is generated to see the strongly correlated features so that those features can be excluded. But using the threshold value of 0.8 for correlation, it is seen that none of the features are strongly correlated. So, I have worked with all the features in the later part of the assignment.

Task 2

- Using the Silhouette Score, the optimal number of clusters has been found to be 2.
- From the observation, it can be said that the people with heart disease belong to the first cluster and the second clusters refers to the people with no heart disease.
- The clusters are not completely relatable to the labels of the dataset.
- Also, the calculated Rand Index is comparatively lower which identifies that there are not enough significant match between the clusters and the labels of the dataset.

Task 3

- The dataset has been apportioned into train and test sets with 80-20 split for both Decision Tree and Naïve Bayes classifier.
- Accuracy has been used to evaluate the both the classifiers' performance as it is a classification task and the labels of target variable in the dataset is quite well-balanced which is well-suited for accuracy metric.
- Decision tree outperforms the Naïve Bayes in both the training and testing accuracy. The reason behind this is the dataset which has been used for this assignment has non-linear relationship between the features. Decision Tree handles non-linear dataset effectively.
- The Pvalue of the Statistical Significance Test is 0.001 which is lower than the threshold 0.05. This indicates that the null hypothesis can be rejected and both the model will perform differently.
- Also, the patterns found by both the supervised models are not significantly similar to the ones presented in the clustering.

Reference

[1] D. Lapp, "Heart disease dataset," *Kaggle*, 06-Jun-2019. [Online]. Available: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset. [Accessed: 21-Oct-2022].

[2] M. Madaan, "Feature Selection Techniques," *Naukri.com*. [Online]. Available: https://www.naukri.com/learning/articles/feature-selection-techniques-python-code/. [Accessed: 22-Oct-2022].

[3] "K-means clustering algorithm - javatpoint," *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning. [Accessed: 22-Oct-2022].

[4] N. Sharma, "K-means clustering explained," *neptune.ai*, 22-Jul-2022. [Online]. Available: https://neptune.ai/blog/k-means-clustering. [Accessed: 23-Oct-2022].

[5] A. Bhardwaj, "Silhouette coefficient: Validating clustering techniques," *Medium*, 27-May-2020. [Online]. Available: https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c. [Accessed: 24-Oct-2022].

[6] "Selecting the number of clusters with silhouette analysis on kmeans clustering," *scikit*. [Online]. Available: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html. [Accessed: 24-Oct-2022].

[7] "Sklearn.metrics.rand_score," *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.rand_score.html#:~:text=The%20Rand%20Index%20computes%20a,the%20predicted%20and%20true%20clusterings.&text=Read%20more%20in%20the%20User,n_samples%2C)%2C%20dtype%3Dintegral. [Accessed: 24-Oct-2022].

[8] "Rand index," *Wikipedia*, 17-Sep-2022. [Online]. Available: https://en.wikipedia.org/wiki/Rand_index#:~:text=The%20Rand%20index%20has%20a,clusterings%20are%20exactly%20the%20same. [Accessed: 24-Oct-2022].

[9] M. Olugbenga, "Balanced accuracy: When should you use it?," *neptune.ai*, 22-Jul-2022. [Online]. Available: https://neptune.ai/blog/balanced-accuracy#:~:text=Accuracy%20can%20be%20a%20useful,related%20to%20the%20accuracy%20paradox. [Accessed: 25-Oct-2022].

[10] "Decision tree," *Corporate Finance Institute*, 05-May-2022. [Online]. Available: https://corporatefinanceinstitute.com/resources/knowledge/other/decision-tree/. [Accessed: 25-Oct-2022].

[11] A. Yadav, “Decision trees,” *Medium*, 11-Jan-2019. [Online]. Available: https://towardsdatascience.com/decision-trees-d07e0f420175. [Accessed: 25-Oct-2022].

[12] J. Kiani, “Using the corrected paired student's t-test for comparing machine learning models,” *Medium*, 23-Oct-2019. [Online]. Available: https://medium.com/analytics-vidhya/using-the-corrected-paired-students-t-test-for-comparing-the-performance-of-machine-learning-dc6529eaa97f. [Accessed: 26-Oct-2022].

[13] S. A. [McLeod, “[what a P-value tells you about statistical significance],” *P*, 01-Jan-1970. [Online]. Available: https://www.simplypsychology.org/p-value.html. [Accessed: 26-Oct-2022].

[14] “How to combine PCA and K-means clustering in python?,” *365 Data Science*, 29-Jul-2021. [Online]. Available: https://365datascience.com/tutorials/python-tutorials/pca-k-means/. [Accessed: 26-Oct-2022].

CSCI 6515 - Assignment 2

In [644…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [645…

```
dataset = pd.read_csv("heart.csv", sep =",")
dataset
```

Out[645…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

In [646…

```
dataset.dtypes
```

Out[646…

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

In [647…

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

In [648…

```
dataset.describe()
```

Out[648…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.(
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.

In [649…

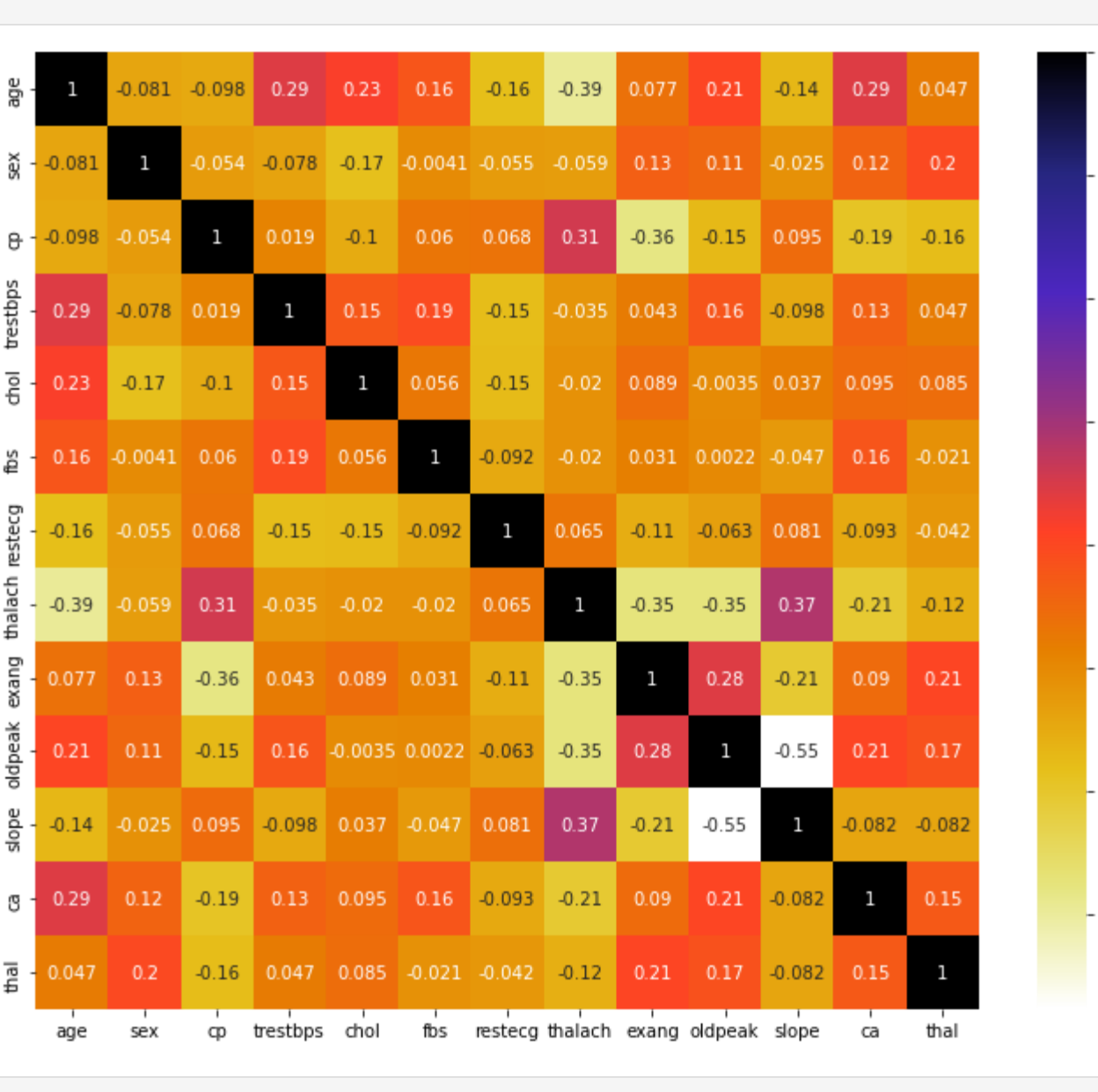
```
X = dataset.drop(["target"], axis = 1)
Y = dataset["target"]
```

In [650…

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
```

In [651…

```
import seaborn as sns
#Using Pearson Correlation
plt.figure(figsize=(12,10))
cor = X_train.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
plt.show()
```



In [652…

```
def correlation_features(dataset, threshold):
    columns_corr = set()
    cor_mat = dataset.corr()
    for i in range(len(cor_mat.columns)):
        for j in range(i):
            if abs(cor_mat.iloc[i, j]) > threshold:
                colname = cor_mat.columns[i]
                columns_corr.add(colname)
    return columns_corr
```

In [653…

```
corr_features = correlation_features(X_train, 0.8)
len(corr_features)
```

Out[653…

```
0
```

In [654…

```
final_dataset = dataset
final_dataset
```

Out[654…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

In [655…

```
final_dataset['target'].value_counts()
```

Out[655…

```
1    526
0    499
Name: target, dtype: int64
```

In [800…

```
final_dataset.describe()
```

Out[800…

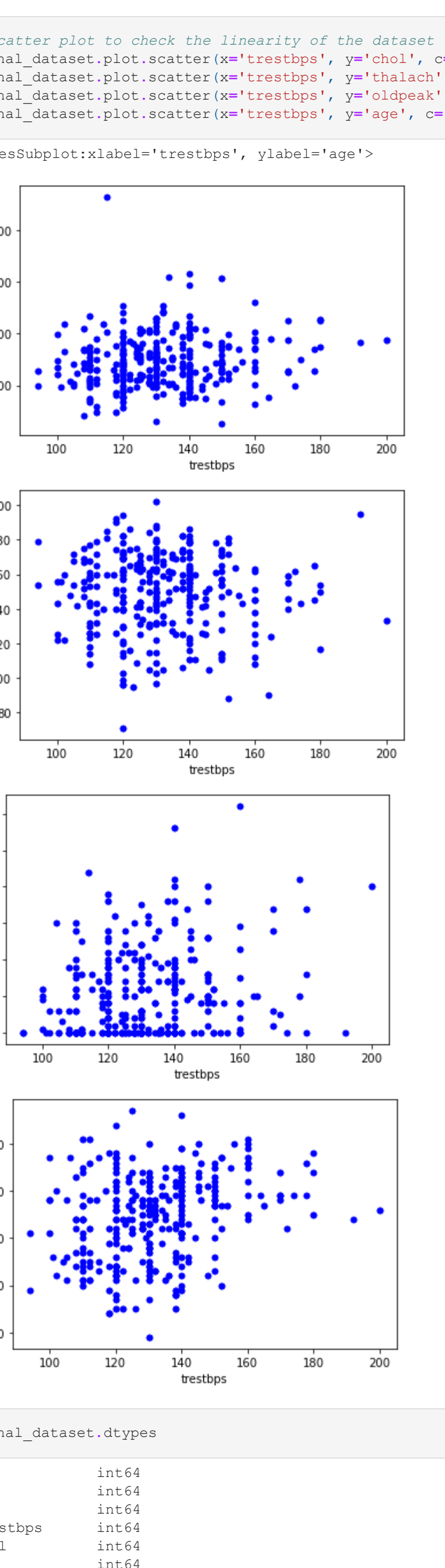
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.(
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.

In [656…

```
#scatter plot to check the linearity of the dataset
final_dataset.plot.scatter(x='trestbps', y='chol', c='b')
final_dataset.plot.scatter(x='trestbps', y='thalach', c='b')
final_dataset.plot.scatter(x='trestbps', y='oldpeak', c='b')
final_dataset.plot.scatter(x='trestbps', y='age', c='b')
```

Out[656…

```
<AxesSubplot:xlabel='trestbps', ylabel='age'>
```



In [657…

```
final_dataset.dtypes
```

Out[657…

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

In [658…

```
final_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

In [659…

```
clustering_dataset = final_dataset.drop(["target"], axis =1)
clustering_dataset
```

Out[659…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3

1025 rows × 13 columns

In [660…

```
cols = clustering_dataset.columns
#cols
```

K Means Clustering

In [661…

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
clustering_dataset = scaler.fit_transform(clustering_dataset)
```

In [662…

```
clustering_dataset = pd.DataFrame(clustering_dataset, columns = [cols])
clustering_dataset
```

Out[662…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	tha
0	-0.268437	0.661504	-0.915755	-0.377636	-0.659332	-0.418878	0.891255	0.821321	-0.712287	-0.060888	0.995433	1.209221	1.08985
1	-0.158157	0.661504	-0.915755	0.479107	-0.833861	2.387330	-1.004049	0.255968	1.403928	1.727137	-2.243675	-0.731971	1.08985
2	1.716595	0.661504	-0.915755	0.764688	-1.396233	-0.418878	0.891255	-1.048692	1.403928	1.301417	-2.243675	-0.731971	1.08985
3	0.724079	0.661504	-0.915755	0.936037	-0.833861	-0.418878	0.891255	0.516900	-0.712287	-0.912329	0.995433	0.238625	1.08985
4	0.834359	-1.511706	-0.915755	0.364875	0.930822	2.387330	0.891255	-1.874977	-0.712287	0.705408	-0.624121	2.179817	-0.52212
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	0.503520	0.661504	0.055931	0.479107	-0.484803	-0.418878	0.891255	0.647366	1.403928	-0.912329	0.995433	-0.731971	-0.52212
1021	0.613800	0.661504	-0.915755	-0.377636	0.232705	-0.418878	-1.004049	-0.352873	1.403928	1.471705	-0.624121	0.238625	1.08985
1022	-0.819834	0.661504	-0.915755	-1.234378	0.562371	-0.418878	-1.004049	-1.353113	1.403928	-0.060888	-0.624121	0.238625	-0.52212
1023	-0.488996	-1.511706	-0.915755	-1.234378	0.155137	-0.418878	-1.004049	0.429923	-0.712287	-0.912329	0.995433	-0.731971	-0.52212
1024	-0.047877	0.661504	-0.915755	-0.663216	-1.124743	-0.418878	0.891255	-1.570556	-0.712287	0.279688	-0.624121	0.238625	1.08985

1025 rows × 13 columns

In [663…

```
clustering_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   (age,)       1025 non-null   float64
 1   (sex,)       1025 non-null   float64
 2   (cp,)        1025 non-null   float64
 3   (trestbps,)  1025 non-null   float64
 4   (chol,)      1025 non-null   float64
 5   (fbs,)       1025 non-null   float64
 6   (restecg,)   1025 non-null   float64
 7   (thalach,)   1025 non-null   float64
 8   (exang,)     1025 non-null   float64
 9   (oldpeak,)   1025 non-null   float64
 10  (slope,)     1025 non-null   float64
 11  (ca,)        1025 non-null   float64
 12  (thal,)      1025 non-null   float64
dtypes: float64(13)
memory usage: 104.2 KB
```

In [664…

```
X = clustering_dataset.iloc[:, :].values
X
```

Out[664…

```
array([[-0.26843658,  0.66150409, -0.91575542, ...,  0.99543334,
         1.20922066,  1.08985168],
       [-0.15815703,  0.66150409, -0.91575542, ..., -2.24367514,
        -0.73197147,  1.08985168],
       [ 1.71659547,  0.66150409, -0.91575542, ..., -2.24367514,
        -0.73197147,  1.08985168],
       ...,
       [-0.81983438,  0.66150409, -0.91575542, ..., -0.6241209 ,
         0.23862459, -0.52212231],
       [-0.4889957 , -1.51170646, -0.91575542, ...,  0.99543334,
        -0.73197147, -0.52212231],
       [-0.04787747,  0.66150409, -0.91575542, ..., -0.6241209 ,
         0.23862459,  1.08985168]])
```

In [665…

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples
```

In [666…

```
for i in range (1, 11):
    kmeans = KMeans(n_clusters = i, random_state = 42)
    kmeans.fit(X)
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=5.
  warnings.warn(
```

In [667…

```
silhouette_score(X, kmeans.labels_)
```

Out[667…

```
0.1300985532372137
```

In [668…

```
kmeans_per_k = [KMeans(init = "k-means++", n_clusters = k, random_state=42).fit(X) for k in range(1,11)]

silhouette_scores = [silhouette_score(X,model.labels_) for model in kmeans_per_k[1:]]

silhouette_scores
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=5.
  warnings.warn(
```

Out[668…

```
[0.1700813691485787,
 0.12381450469263082,
 0.1321750438629156,
 0.13041369796583743,
 0.12012371629555761,
 0.1183363929017515,
 0.1304428912155277,
 0.12731265527562563,
 0.1300985532372137]
```

In [669…

```
highest_sil = max(silhouette_scores)
highest_sil
```

Out[669…

```
0.1700813691485787
```

In [670…

```
clustered_df = pd.DataFrame({'Silhouette Scores':silhouette_scores})
```

In [671…

```
clustered_df["Cluster"] = range(2,11)
clustered_df.set_index("Cluster")
```

Out[671…

	Silhouette Scores
Cluster	
2	0.170081
3	0.123815
4	0.132175
5	0.130414
6	0.120124
7	0.118336
8	0.130443
9	0.127313
10	0.130099

In [672…

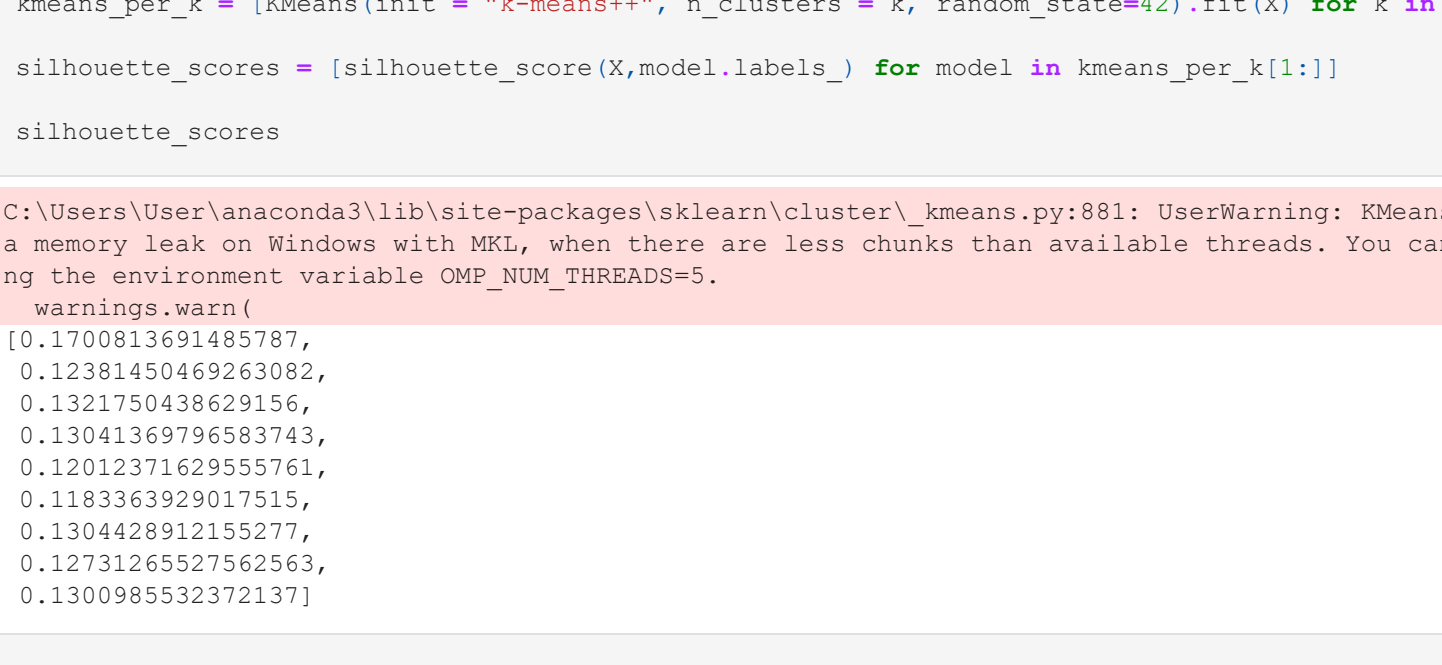
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.plot(range(2,11), silhouette_scores, color = "green", linewidth = 2)
plt.xlabel("Values of K", fontsize = 12)
plt.ylabel("Silhoutte Score", fontsize = 12)
plt.grid()
plt.title("Silhoutte Score Analysis for Optimal Value of K")

k = np.argmax(silhouette_scores) + 2

plt.axvline(x=k, linestyle ="--",c="red")

plt.scatter(k, silhouette_scores[k-2], s=400)
plt.legend(shadow=True)
plt.show()
```

```
No handles with labels found to put in legend.
```



In [673…

```
print("The optimal Value of K is {}".format(k))
```

```
The optimal Value of K is 2
```

In [674…

```
k_means = KMeans (n_clusters = 2)
```

In [675…

```
label = k_means.fit_predict(X)
```

In [676…

```
clustering_dataset["Cluster"] = label
clustering_dataset
```

Out[676…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	tha
0	-0.268437	0.661504	-0.915755	-0.377636	-0.659332	-0.418878	0.891255	0.821321	-0.712287	-0.060888	0.995433	1.209221	1.08985
1	-0.158157	0.661504	-0.915755	0.479107	-0.833861	2.387330	-1.004049	0.255968	1.403928	1.727137	-2.243675	-0.731971	1.08985
2	1.716595	0.661504	-0.915755	0.764688	-1.396233	-0.418878	0.891255	-1.048692	1.403928	1.301417	-2.243675	-0.731971	1.08985
3	0.724079	0.661504	-0.915755	0.936037	-0.833861	-0.418878	0.891255	0.516900	-0.712287	-0.912329	0.995433	0.238625	1.08985
4	0.834359	-1.511706	-0.915755	0.364875	0.930822	2.387330	0.891255	-1.874977	-0.712287	0.705408	-0.624121	2.179817	-0.52212
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	0.503520	0.661504	0.055931	0.479107	-0.484803	-0.418878	0.891255	0.647366	1.403928	-0.912329	0.995433	-0.731971	-0.52212
1021	0.613800	0.661504	-0.915755	-0.377636	0.232705	-0.418878	-1.004049	-0.352873	1.403928	1.471705	-0.624121	0.238625	1.08985
1022	-0.819834	0.661504	-0.915755	-1.234378	0.562371	-0.418878	-1.004049	-1.353113	1.403928	-0.060888	-0.624121	0.238625	-0.52212
1023	-0.488996	-1.511706	-0.915755	-1.234378	0.155137	-0.418878	-1.004049	0.429923	-0.712287	-0.912329	0.995433	-0.731971	-0.52212
1024	-0.047877	0.661504	-0.915755	-0.663216	-1.124743	-0.418878	0.891255	-1.570556	-0.712287	0.279688	-0.624121	0.238625	1.08985

1025 rows × 14 columns

In [681…

```
clustering_dataset1 = clustering_dataset
clustering_dataset1.columns = clustering_dataset1.columns.get_level_values(0)
```

In [682…

```
clus_results = clustering_dataset1.groupby(["Cluster"]).mean()
clus_results.abs()
```

Out[682…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
Cluster													
0	0.245131	0.083977	0.393715	0.124505	0.073648	0.040706	0.066958	0.481536	0.456267	0.415938	0.377870	0.314303	0.228363
1	0.430297	0.147411	0.691118	0.218553	0.129281	0.071454	0.117536	0.845277	0.800921	0.730127	0.663304	0.551720	0.400863

In [683…

```
clustering_dataset['Cluster'].value_counts()
```

Out[683…

```
0    653
1    372
Name: Cluster, dtype: int64
```

In [684…

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(X)
```

Out[684…

```
PCA(n_components=2)
```

In [685…

```
pca.explained_variance_ratio_
```

Out[685…

```
array([0.21369911, 0.11971956])
```

In [686…

```
pca.transform(X)
```

Out[686…

```
array([[-0.52253834, -1.11273256],
       [ 2.5902992 , -0.53346281],
       [ 3.04225226, -1.32783423],
       ...,
       [ 1.2450846 , -1.45731794],
       [-1.62003215,  0.12450307],
       [ 0.93423115, -1.77834406]])
```

In [687…

```
scores_pca = pca.transform(X)
scores_pca
```

Out[687…

```
array([[-0.52253834, -1.11273256],
       [ 2.5902992 , -0.53346281],
       [ 3.04225226, -1.32783423],
       ...,
       [ 1.2450846 , -1.45731794],
       [-1.62003215,  0.12450307],
       [ 0.93423115, -1.77834406]])
```

In [688…

```
k_means = KMeans(init = "k-means++", n_clusters = 2, random_state = 42)
k_means.fit(scores_pca)
```

Out[688…

```
KMeans(n_clusters=2, random_state=42)
```

In [689…

```
clustered_dataset_ = dataset.drop(["target"], axis =1)
clustered_dataset_ = pd.concat([dataset.reset_index(drop=True),pd.DataFrame(scores_pca)], axis = 1)
clustered_dataset_.columns.values[-2: ] = ["Component 1", "Component 2"]
clustered_dataset_["Clustered Label"] = k_means.labels_
```

In [690…

```
clustered_dataset_
```

Out[690…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	Component 1	Component 2	Clustered Label
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0	-0.522538	-1.112733	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0	2.590299	-0.533463	1
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0	3.042252	-1.327834	1
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0	-0.492532	-0.276730	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0	2.187537	1.951687	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1	-0.762392	-0.512495	0
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0	2.374266	-0.940875	1
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0	1.245085	-1.457318	1
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1	-1.620032	0.124503	0
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0	0.934231	-1.778344	1

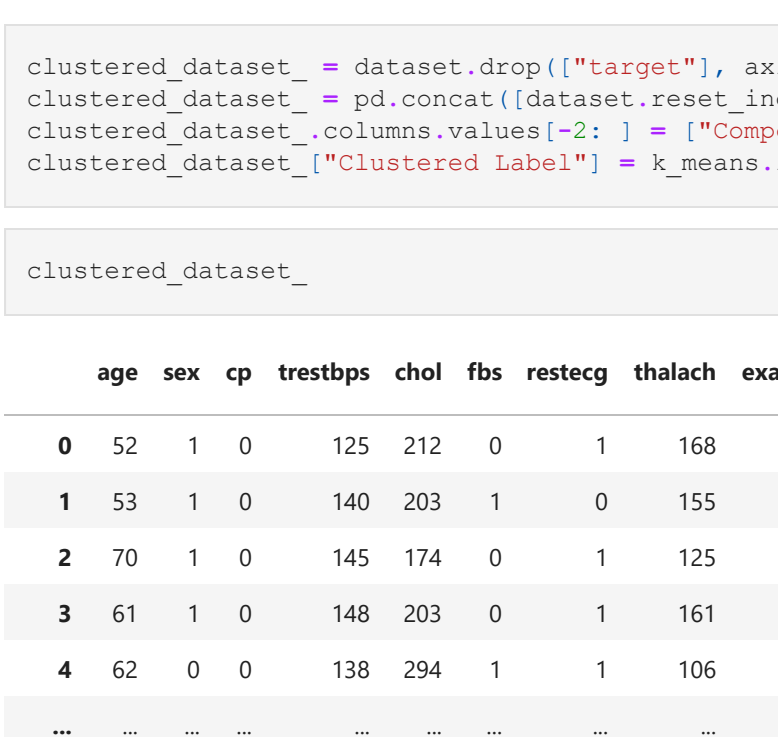
1025 rows × 17 columns

In [691…

```
clustered_dataset_.plot.scatter(x="Component 2", y="Component 1", c = 'Clustered Label', colormap='viridis')
```

Out[691…

```
<AxesSubplot:xlabel='Component 2', ylabel='Component 1'>
```



In [692…

```
final_dataset
```

Out[692…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

In [693…

```
dataset_label = pd.DataFrame(final_dataset["target"])
dataset_label.head(10)
```

Out[693…

	target
0	0
1	0
2	0
3	0
4	0
5	1
6	0
7	0
8	0
9	0

In [694…

```
#Generated clustered labels
clus_label =pd.DataFrame(clustering_dataset["Cluster"])
clus_label.head(10)
```

Out[694…

	Cluster
0	0
1	1
2	1
3	0
4	1
5	0
6	1
7	1
8	0
9	1

In [746…

```
labels_true = final_dataset["target"].values.tolist()
#labels_true
```

In [747…

```
labels_pred = clustering_dataset["Cluster"].values.tolist()
#labels_pred
```

In [748…

```
#Converting the 2D list to 1D shape
#labels_pred_ = clustering_dataset["Cluster"].values.tolist()
#labels_pred = sum(labels_pred_, [])
#labels_pred
```

In [749…

```
#Calculating the Rand Index between Clustered Labels and the Initial Labels of the dataset
from sklearn.metrics.cluster import adjusted_rand_score
print("The Adjusted Rand Index is:", adjusted_rand_score(labels_true, labels_pred))
```

```
The Adjusted Rand Index is: 0.36409106000433716
```

Decision Tree Classifier

In [750…

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
```

In [751…

```
dataset_cols=final_dataset.columns
dataset_cols
```

Out[751…

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

In [752…

```
scaler = StandardScaler()
#final_dataset = scaler.fit_transform(final_dataset)
#final_dataset
```

In [753…

```
final_dataset = pd.DataFrame(final_dataset, columns = dataset_cols)
final_dataset
```

Out[753…

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

In [754…

```
#feature_cols = ["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach", "exang",
               #"oldpeak", "slope", "ca", "thal"]
```

In [755…
```
# Decision Tree Classifier
```

In [756…
```
X = final_dataset.iloc[:,0:13]
Y = final_dataset["target"]
```

In [757…
```
DTscaler = StandardScaler()
X = DTscaler.fit_transform(X)
```

In [758…
```
#X = scaler.fit_transform(final_dataset.values[:,0:13])
#Y = final_dataset.values[:,13]
```

In [759…
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

In [760…
```
clf_DT = DecisionTreeClassifier()
clf_DT = clf_DT.fit(X_train, y_train)
```

In [761…
```
y_trainpred = clf_DT.predict(X_train)
```

In [762…
```
y_testpred = clf_DT.predict(X_test)
```

In [763…
```
from sklearn import metrics

#Acuracy of Training
print("Training Accuracy of Decision Tree: ",metrics.accuracy_score(y_train, y_trainpred)*100)
```

```
Training Accuracy of Decision Tree:  100.0
```

In [764…
```
#Acuracy of Testing
print("Testing Accuracy of Decision Tree: ",metrics.accuracy_score(y_test, y_testpred)*100)
```

```
Testing Accuracy of Decision Tree:  98.53658536585365
```

In [765…
```
print("Training Confusion Matrix: \n", confusion_matrix(y_train, y_trainpred))
```

```
Training Confusion Matrix:
 [[397   0]
 [  0 423]]
```

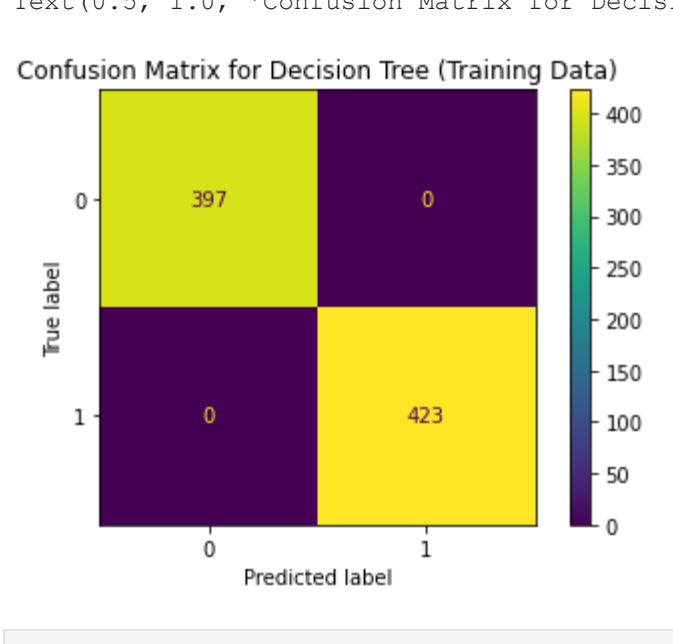
In [766…
```
print("Testing Confusion Matrix: \n", confusion_matrix(y_test, y_testpred))
```

```
Testing Confusion Matrix:
 [[102   0]
 [  3 100]]
```

In [767…
```
disp = ConfusionMatrixDisplay(confusion_matrix(y_train, y_trainpred), clf_DT.classes_).plot()
disp.ax_.set_title("Confusion Matrix for Decision Tree (Training Data)")
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass display_labels=[0
1] as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an
error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

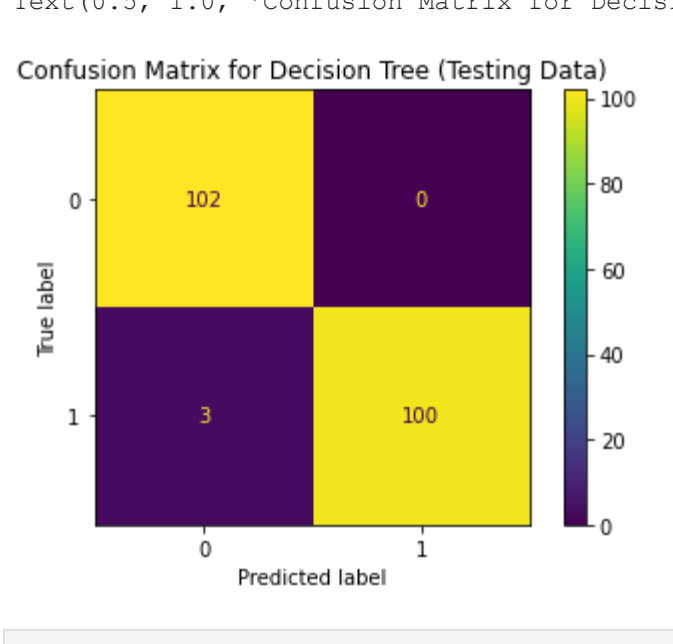
Out[767…
```
Text(0.5, 1.0, 'Confusion Matrix for Decision Tree (Training Data)')
```



In [768…
```
disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_testpred), clf_DT.classes_).plot()
disp.ax_.set_title("Confusion Matrix for Decision Tree (Testing Data)")
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass display_labels=[0
1] as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an
error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

Out[768…
```
Text(0.5, 1.0, 'Confusion Matrix for Decision Tree (Testing Data)')
```



In [769…
```
#Classification Report of Training
print(classification_report(y_train, y_trainpred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       397
           1       1.00      1.00      1.00       423

    accuracy                           1.00       820
   macro avg       1.00      1.00      1.00       820
weighted avg       1.00      1.00      1.00       820
```

In [770…
```
#Classification Report of Testing
print(classification_report(y_test, y_testpred))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103

    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
weighted avg       0.99      0.99      0.99       205
```

Naive Bayes Classifier

In [771…
```
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
```

In [772…
```
X = final_dataset.iloc[:, 0:13]
Y = final_dataset["target"]
```

In [773…
```
Y
```

Out[773…
```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

In [774…
```
NBscaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [775…
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

In [776…
```
clf_NB = GaussianNB()
clf_NB = clf_NB.fit(X_train, y_train)
```

In [777…
```
y_trainpred_NB = clf_NB.predict(X_train)
```

In [778…
```
y_testpred_NB = clf_NB.predict(X_test)
```

In [779…
```
print("Training Accuracy of Naive Bayes: ",metrics.accuracy_score(y_train, y_trainpred_NB)*100)
```

```
Training Accuracy of Naive Bayes:  83.90243902439025
```

In [780…
```
#Acuracy of Testing
print("Testing Accuracy of Naive Bayes:: ",metrics.accuracy_score(y_test, y_testpred_NB)*100)
```

```
Testing Accuracy of Naive Bayes::  80.0
```

In [781…
```
print("Training Confusion Matrix: \n", confusion_matrix(y_train, y_trainpred_NB))
```

```
Training Confusion Matrix:
 [[320  77]
 [ 55 368]]
```

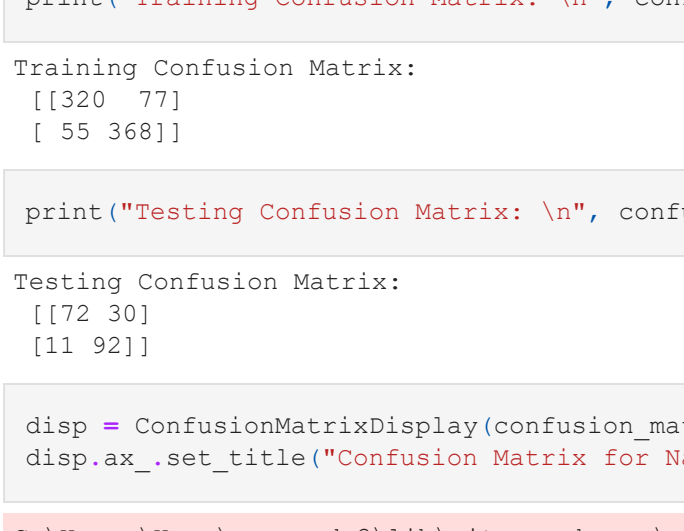
In [782…
```
print("Testing Confusion Matrix: \n", confusion_matrix(y_test, y_testpred_NB))
```

```
Testing Confusion Matrix:
 [[72 30]
 [11 92]]
```

In [783…
```
disp = ConfusionMatrixDisplay(confusion_matrix(y_train, y_trainpred_NB), clf_NB.classes_).plot()
disp.ax_.set_title("Confusion Matrix for Naive Bayes (Training Data)")
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass display_labels=[0
1] as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an
error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

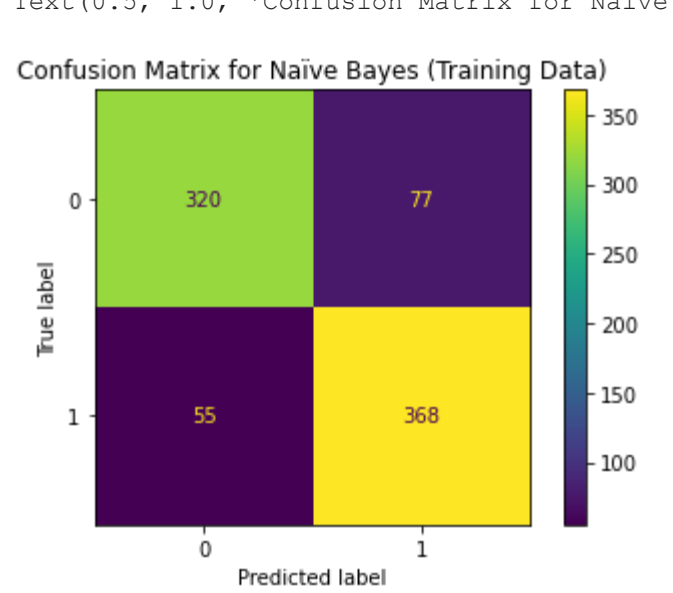
Out[783…
```
Text(0.5, 1.0, 'Confusion Matrix for Naïve Bayes (Training Data)')
```



In [784…
```
disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_testpred_NB), clf_NB.classes_).plot()
disp.ax_.set_title("Confusion Matrix for Naïve Bayes (Testing Data)")
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureWarning: Pass display_labels=[0
1] as keyword args. From version 1.0 (renaming of 0.25) passing these as positional arguments will result in an
error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

Out[784…
```
Text(0.5, 1.0, 'Confusion Matrix for Naïve Bayes (Testing Data)')
```



In [785…
```
#Classification Report of Training
print(classification_report(y_train, y_trainpred_NB))
```

```
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       397
           1       0.83      0.87      0.85       423

    accuracy                           0.84       820
   macro avg       0.84      0.84      0.84       820
weighted avg       0.84      0.84      0.84       820
```

In [786…
```
#Classification Report of Testing
print(classification_report(y_test, y_testpred_NB))
```

```
              precision    recall  f1-score   support

           0       0.87      0.71      0.78       102
           1       0.75      0.89      0.82       103

    accuracy                           0.80       205
   macro avg       0.81      0.80      0.80       205
weighted avg       0.81      0.80      0.80       205
```

Statistical Significance Testing to Compare DT and NB

In [787…
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

In [788…
```
kf = KFold(n_splits=10)
```

In [789…
```
DT_accuracy = []
NB_Accuracy = []

n_train = []
n_test = []
n = []
n = []
i = 1
for train, test in kf.split(X):
    Xtrain, Xtest = X[train], X[test]
    ytrain, ytest = Y[train], Y[test]

    n_train.append(len(ytrain))
    n_test.append(len(ytest))
    n.append(len(Y))

    model_DT = clf_DT.fit(Xtrain, ytrain)
    model_NB = clf_NB.fit(Xtrain, ytrain)

    y_DT = clf_DT.predict(Xtest)
    y_NB = clf_NB.predict(Xtest)

    DT = accuracy_score(ytest, y_DT)
    NB = accuracy_score(ytest, y_NB)

    print("Decision Tree Classifier Fold:", i)
    print(classification_report(ytest, y_DT))
    print("Naive Bayes Classifier Fold:", i)
    print(classification_report(ytest, y_NB))
    print("-------------------------------------------------------")
    DT_accuracy.append(DT)
    NB_Accuracy.append(NB)
    i += 1
```

```
Decision Tree Classifier Fold: 1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       1.00      1.00      1.00        50

    accuracy                           1.00       103
   macro avg       1.00      1.00      1.00       103
weighted avg       1.00      1.00      1.00       103

Naive Bayes Classifier Fold: 1
              precision    recall  f1-score   support

           0       0.93      0.79      0.86        53
           1       0.81      0.94      0.87        50

    accuracy                           0.86       103
   macro avg       0.87      0.87      0.86       103
weighted avg       0.87      0.86      0.86       103

-------------------------------------------------------
Decision Tree Classifier Fold: 2
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        48
           1       1.00      1.00      1.00        55

    accuracy                           1.00       103
   macro avg       1.00      1.00      1.00       103
weighted avg       1.00      1.00      1.00       103

Naive Bayes Classifier Fold: 2
              precision    recall  f1-score   support

           0       0.88      0.88      0.88        48
           1       0.89      0.89      0.89        55

    accuracy                           0.88       103
   macro avg       0.88      0.88      0.88       103
weighted avg       0.88      0.88      0.88       103

-------------------------------------------------------
Decision Tree Classifier Fold: 3
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        43
           1       1.00      1.00      1.00        60

    accuracy                           1.00       103
   macro avg       1.00      1.00      1.00       103
weighted avg       1.00      1.00      1.00       103

Naive Bayes Classifier Fold: 3
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        43
           1       0.90      0.87      0.88        60

    accuracy                           0.86       103
   macro avg       0.86      0.86      0.86       103
weighted avg       0.87      0.86      0.86       103

-------------------------------------------------------
Decision Tree Classifier Fold: 4
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        50
           1       1.00      0.94      0.97        53

    accuracy                           0.97       103
   macro avg       0.97      0.97      0.97       103
weighted avg       0.97      0.97      0.97       103

Naive Bayes Classifier Fold: 4
              precision    recall  f1-score   support

           0       0.76      0.84      0.80        50
           1       0.83      0.75      0.79        53

    accuracy                           0.80       103
   macro avg       0.80      0.80      0.80       103
weighted avg       0.80      0.80      0.80       103

-------------------------------------------------------
Decision Tree Classifier Fold: 5
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        47
           1       1.00      0.95      0.97        56

    accuracy                           0.97       103
   macro avg       0.97      0.97      0.97       103
weighted avg       0.97      0.97      0.97       103

Naive Bayes Classifier Fold: 5
              precision    recall  f1-score   support

           0       0.92      0.70      0.80        47
           1       0.79      0.95      0.86        56

    accuracy                           0.83       103
   macro avg       0.85      0.82      0.83       103
weighted avg       0.85      0.83      0.83       103

-------------------------------------------------------
Decision Tree Classifier Fold: 6
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        52
           1       1.00      1.00      1.00        50

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Naive Bayes Classifier Fold: 6
              precision    recall  f1-score   support

           0       0.95      0.79      0.86        52
           1       0.81      0.96      0.88        50

    accuracy                           0.87       102
   macro avg       0.88      0.87      0.87       102
weighted avg       0.88      0.87      0.87       102

-------------------------------------------------------
Decision Tree Classifier Fold: 7
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       1.00      1.00      1.00        49

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Naive Bayes Classifier Fold: 7
              precision    recall  f1-score   support

           0       0.80      0.75      0.78        53
           1       0.75      0.80      0.77        49

    accuracy                           0.77       102
   macro avg       0.78      0.78      0.77       102
weighted avg       0.78      0.77      0.77       102

-------------------------------------------------------
Decision Tree Classifier Fold: 8
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        48
           1       1.00      1.00      1.00        54

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Naive Bayes Classifier Fold: 8
              precision    recall  f1-score   support

           0       0.75      0.81      0.78        48
           1       0.82      0.76      0.79        54

    accuracy                           0.78       102
   macro avg       0.78      0.79      0.78       102
weighted avg       0.79      0.78      0.78       102

-------------------------------------------------------
Decision Tree Classifier Fold: 9
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        58
           1       1.00      1.00      1.00        44

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Naive Bayes Classifier Fold: 9
              precision    recall  f1-score   support

           0       0.84      0.71      0.77        58
           1       0.68      0.82      0.74        44

    accuracy                           0.75       102
   macro avg       0.76      0.76      0.75       102
weighted avg       0.77      0.75      0.76       102

-------------------------------------------------------
Decision Tree Classifier Fold: 10
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        47
           1       1.00      1.00      1.00        55

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Naive Bayes Classifier Fold: 10
              precision    recall  f1-score   support

           0       0.81      0.74      0.78        47
           1       0.80      0.85      0.82        55

    accuracy                           0.80       102
   macro avg       0.81      0.80      0.80       102
weighted avg       0.80      0.80      0.80       102

-------------------------------------------------------
```

In [790…
```
from scipy.stats import t
```

In [791…
```
#different between the results of 10-cross validation for DT and NB
difference = [i-j for i,j in zip(DT_accuracy, NB_Accuracy)]
```

In [792…
```
#mean of the difference
diff_mean = np.mean(difference)
diff_mean
```

Out[792…
```
0.17088330477822197
```

In [793…
```
#n_train = datapoints used for training
#n_test_ = datapoints used for testing
#n_final = total number of datapoints
n_train_ = np.median(n_train)
n_test_ = np.median(n_test)
n_final = len(n)
```

In [794…
```
sigma2 = np.var(difference)
#modified variance
sigma2_mod = sigma2 * (1/n_final + n_test_/n_train_)
sigma2_mod
```

Out[794…
```
0.0004149819450945956
```

In [795…
```
t_static =  diff_mean / np.sqrt(sigma2_mod)
t_static
```

Out[795…
```
8.388513995155447
```

In [796…
```
Pvalue = ((1 - t.cdf(t_static,n_final-1))*200)
Pvalue
```

Out[796…
```
0.001512473162601502
```