“Performing K-nearest neighbor (KNN) and Decision Tree for Heart Disease Prediction.”

Presented by: Moaz Tayea

Table Of Contents:

Table Of Figures:

Introduction:

The k-nearest neighbor (KNN) technique is an instance-based learning technique that does not begin with a comprehensive theoretical model. As an alternative, it employs a slightly simpler concept, and the instance nearby in the input space is likely to belong to the same class. A KNN categorizes an example to the class that appears most persistently among its k nearby neighbors. The constraint k is used to fine-tune the classification work.[1]

Decision tree is one of the potent techniques frequently employed in a variety of fields, including machine learning, image processing, and pattern recognition [3]. This method is commonly used in classification problems. It works well with both continuous and categorical attributes. Based on the most significant predictors, this algorithm divides the population into two or more similar sets. The Decision Tree algorithm first computes the entropy of each attribute. The dataset is then divided using variables or predictors with the highest information gain or lowest entropy. These two steps are repeated with the remaining attributes.

Objective:

Heart disease is a major source of public health concern around the world. Due to a lack of health awareness and poor consumption habits, the number of heart patients is rapidly increasing. As a result, it is critical to have a framework that can detect the presence of heart disease in thousands of samples in real-time. The potential of two machine learning techniques for predicting heart disease was assessed (K-nearest neighbor and Decision Tree).

Justification of used algorithms:

In this paper, we classified patient data to predict whether a patient has heart disease. We used the heart disease dataset for this classification. The Heart Disease dataset contains 303 patient records, with 14 attributes for each record. These 14 characteristics are used to assess and predict whether a patient has heart disease or not. If a patient has heart disease, it is treated as 1, otherwise, it is treated as 0. Two classification algorithms were used to classify and predict whether a patient has heart disease or not. K-nearest neighbors and decision trees are two machine learning algorithms.[5]

KNN is a nonparametric supervised learning algorithm that builds model structures from given datasets. The number k in KNN determines how many nearest neighbors are used to provide the best result for the dataset. If 'x' is the point whose label is to be predicted, we must first find the 'k' closest point to 'x'. Then, vote on how many nearest data points represent a specific class. The point 'a' will go to the class that receives the most votes. We can use any of these distance calculation methods to find the closest point to our data points, such as Euclidean distance, Hamming distance, and Manhattan distance. Finding the best k value for a dataset is a difficult task. The k value for each dataset is unique. However, some assumptions state that the best k value is an odd number. We can say that the value of k is the problem's controlling factor. According to research, if we use a smaller k value, we may experience overfitting; that is, noise has a greater impact on our prediction than an actual prediction. If we use a larger value of k, the computation becomes more expensive. As a result, we must determine the optimal value of k. We can find the best k value by running our operation on the dataset with various k values. For that dataset, the k value that produces the best results will be used.[7]. So we choose KNN because:

- Simple.
- Fast training with small datasets.
- Good for numerous multi-class categorical targets [4].
- High prediction accuracy

The decision tree is a structure that resembles a tree. The features are represented by the internal node or non-leaf node, the decision rules are represented by the branch, and the result is represented by each leaf node. The topmost node in the decision tree is the tree's root. Based on the features in the dataset, this root node learns to classify it. It recursively divides the dataset. It provides a flowchart-like representation that is suitable for human comprehension and aids in decision making. A decision tree is a white box machine learning algorithm model that provides detailed information about the process. It is a nonparametric approach. The attribute selection measure process is used to split the dataset using features; it is a heuristic splitting rule. Gain, gain ratio, Gini index, and other popular selection measures. That is why we have chosen Decision tree because it has some advantages such as :

- Prediction Accuracy [6].

- Simple to be understood.
- Relevant to incremental learning [4].
- Efficient memory usage.
- Fast performing ML model.
- Ability to handle noisy and scalable data.
- Having various measures to find the best split attribute,
- like: Entropy, Gini index, Information gain etc.

Steps of building the two models:

1. Explore the data and choose the type of algorithm.

Jupyter notebook, freely available software for performing machine learning operations, was used. For data analysis, we must import the Python NumPy and Pandas modules. To plot various graphs, we must import the Matplotlib module, which contains all graph plotting methods.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
```

Figure 1

Our data is stored in a CSV file, which must be imported into the notebook as a data frame using the Python Pandas module and display the first five records from the dataset using df.head() function

```
df = pd.read_csv('heart_disease.csv')
```

```
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Figure 2

We can use various data analyses and machine learning algorithms for classification and prediction after importing the data [2].

Importing the sklearn module, which contains all the necessary algorithms and functions, is required for machine learning.

Here we will be experimenting with 2 algorithms

1. KNeighborsClassifier
2. DecisionTreeClassifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

Figure 3

To check the info of the data to know its datatype and knowing if it has missing values or not we use df.info().

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Figure 4

Descriptive statistics, which exclude NaN values, summaries the central tendency, dispersion, and shape of a dataset's distribution. We use df.describe().

```
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

Figure 5

2. Feature Selection.

We applied feature selection here basically to find out if all the features are positively correlated or negatively correlated. (Fig 7) (Fig 8)

Feature Selection

In [34]:

```
import seaborn as sns
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Figure 6

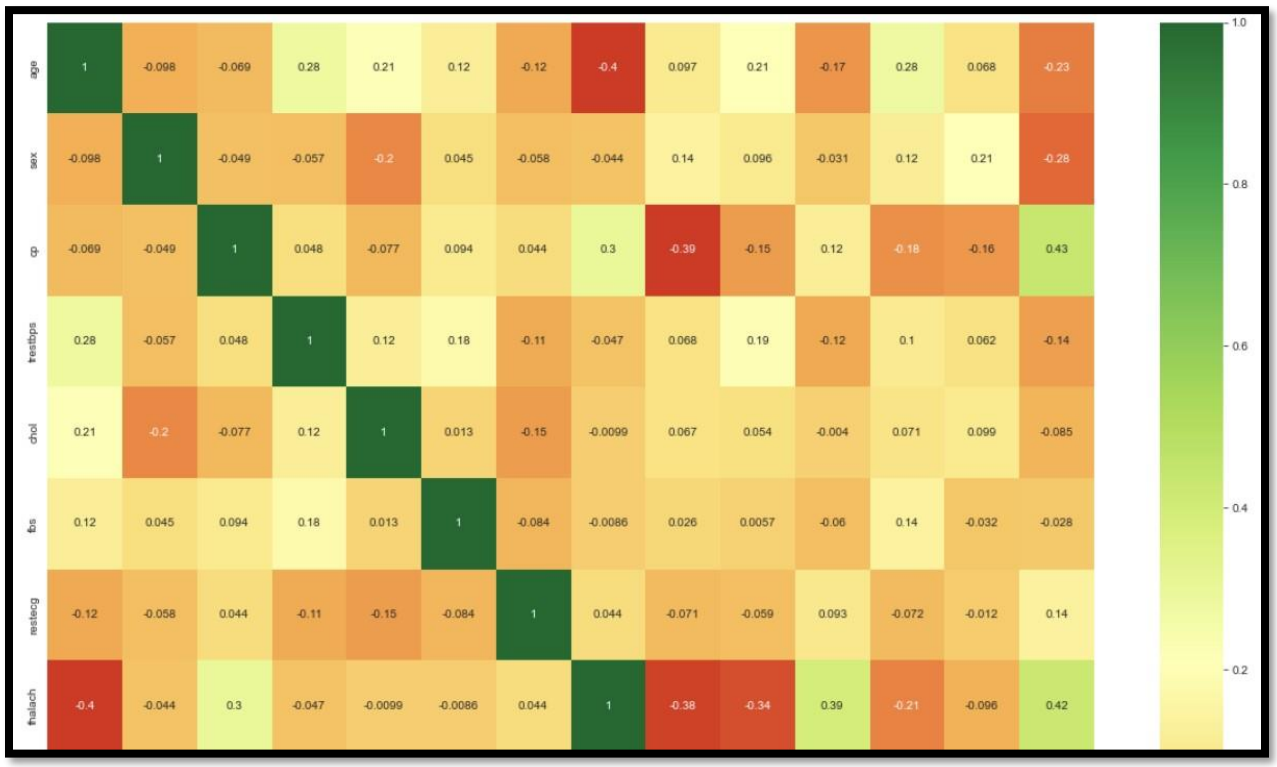

Figure 7

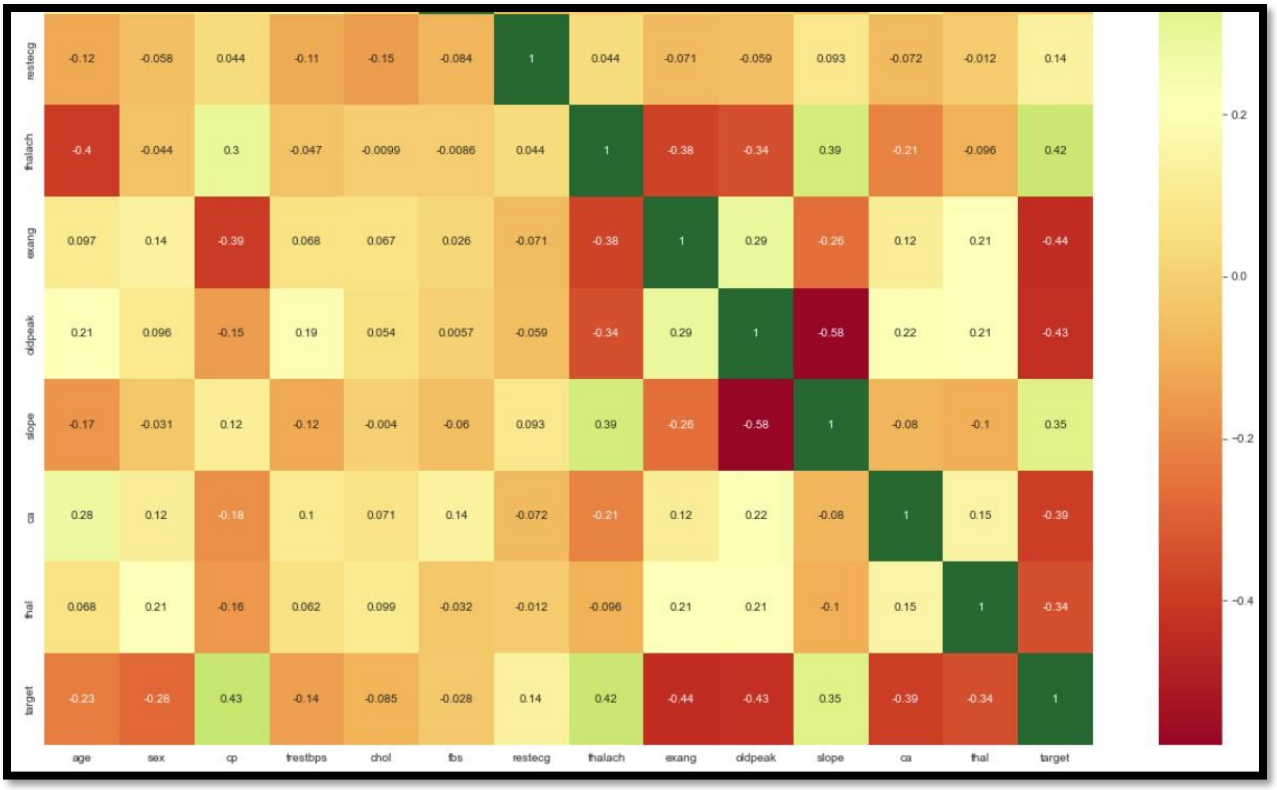

Figure 8

It's always a good practice to work with a dataset where the target classes are of approximately equal size. Thus, let's check for the same. (Fig 9)

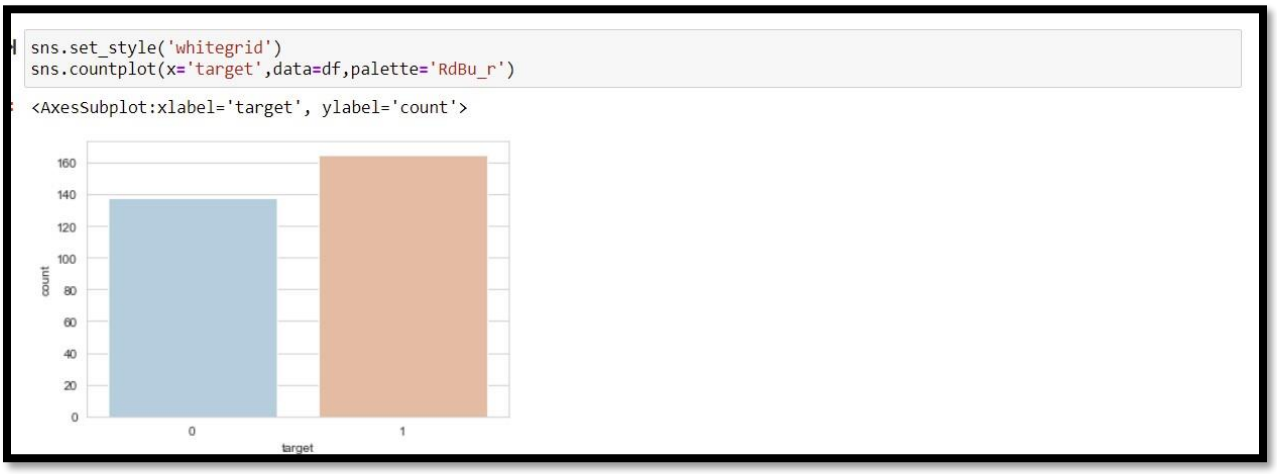

Figure 9

Fig 9. Zero means that the person has no heart disease and 1 means that he has heart disease.

3. Data Preprocessing.

After exploring the dataset, we observed that we need to convert some categorical variables into dummy variables and scale all the values before training the Machine Learning models. First, we will use the get_dummies method to create dummy columns for categorical variables. Then we imported train_test_split function from sklearn.model_selection to split the dataset into a training set and test set, after that we imported the StandardScaler function from sklearn. preprocessing to standardize the data values into a standard format.

Data Processing

```
dataset = pd.get_dummies(df, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[columns_to_scale] = standardScaler.fit_transform(dataset[columns_to_scale])
```

```
dataset.head()
```

9]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	thal_
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	0	1	0	0	...	0	1	0	0	0	0	0	1	
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	0	1	0	0	...	0	1	0	0	0	0	0	0	
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	1	0	0	1	...	1	1	0	0	0	0	0	0	
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	0	1	0	1	...	1	1	0	0	0	0	0	0	
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	1	0	1	0	...	1	1	0	0	0	0	0	0	

5 rows × 31 columns

Figure 10

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.2,random_state=0)
```

```
X_train = standardScaler.fit_transform(X_train)
X_test = standardScaler.transform(X_test)
```

Figure 11

* Standardization is a scaling technique wherein it makes the data scale-free by converting the statistical distribution of the data into the below format:

$$z = \frac{x - \mu}{\sigma}$$

Equation 1

Before we start applying the machine learning algorithms to the dataset, we can see what the features of our dataset look like in plots; for this, we are going to use Matplotlib. (Fig 12)(Fig 13)

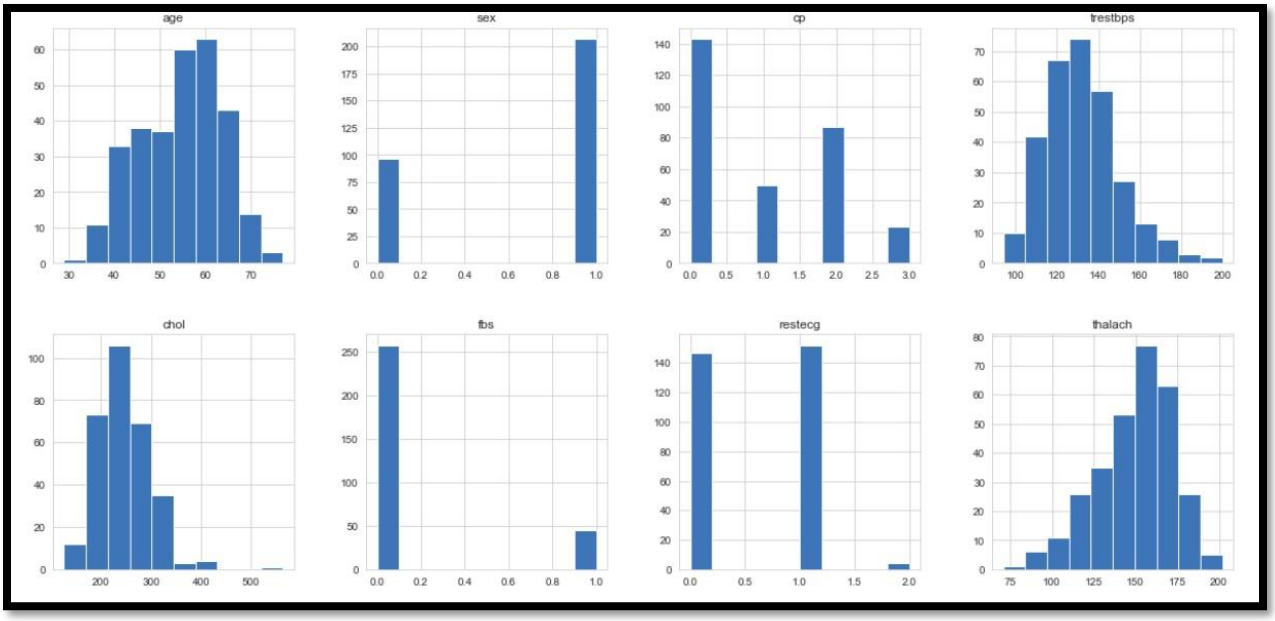

Figure 12

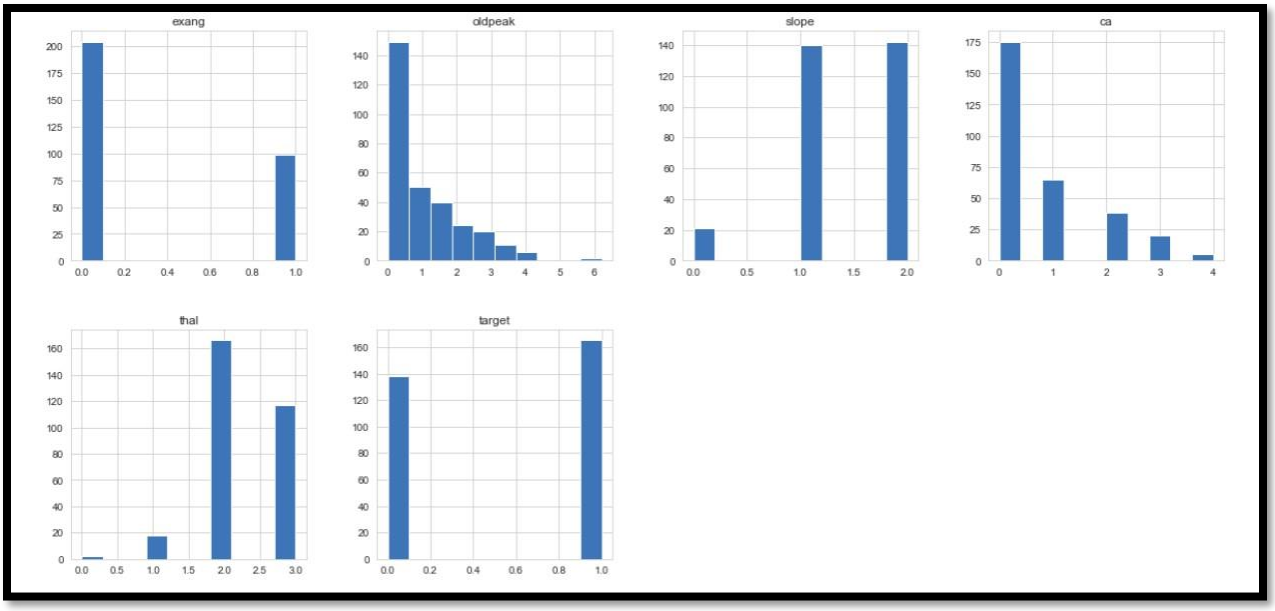

Figure 13

Now we can apply the machine learning algorithms to the dataset.

4. **Split the prepared dataset, apply the algorithms and perform accuracy score.**

K-Nearest Neighbor (KNN) Algorithm:

We must first separate the features and labels from the dataset before applying our KNN algorithm to it. We took all the dataset's features in X variable, and the label, i.e., the target column of our dataset, in Y variable.

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

Figure 14

We have no idea which k value is best suited to our dataset.

As a result, we will examine each k value in our model. In this case, we chose a k value between (1, 21); that is, all values between this range are used as k to produce an accurate score. We used accuracy_score techniques to accomplish this. It provides an accurate measure of the machine learning model's performance, that is, what we expect from our model. For each value of k, we used the accuracy_score. The k value with the highest score is chosen for further processing. To find out all the scores for all the values in the range (1, 21), we created a graph with Matplotlib. (Fig 4).

KNN

```
from sklearn.metrics import classification_report, accuracy_score
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    knn_classifier.fit(X_train,y_train)
    y_pred = knn_classifier.predict(X_test)
    knn_scores.append(accuracy_score(y_test,y_pred))
```

Figure 15

```
plt.plot([k for k in range(1,21)], knn_scores, color = 'red')
for i in range(1,21):
    plt.text(i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks([i for i in range(1,21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('K Neighbors Classifier scores for different K values')
plt.rcParams["figure.figsize"] = (20,1)
```

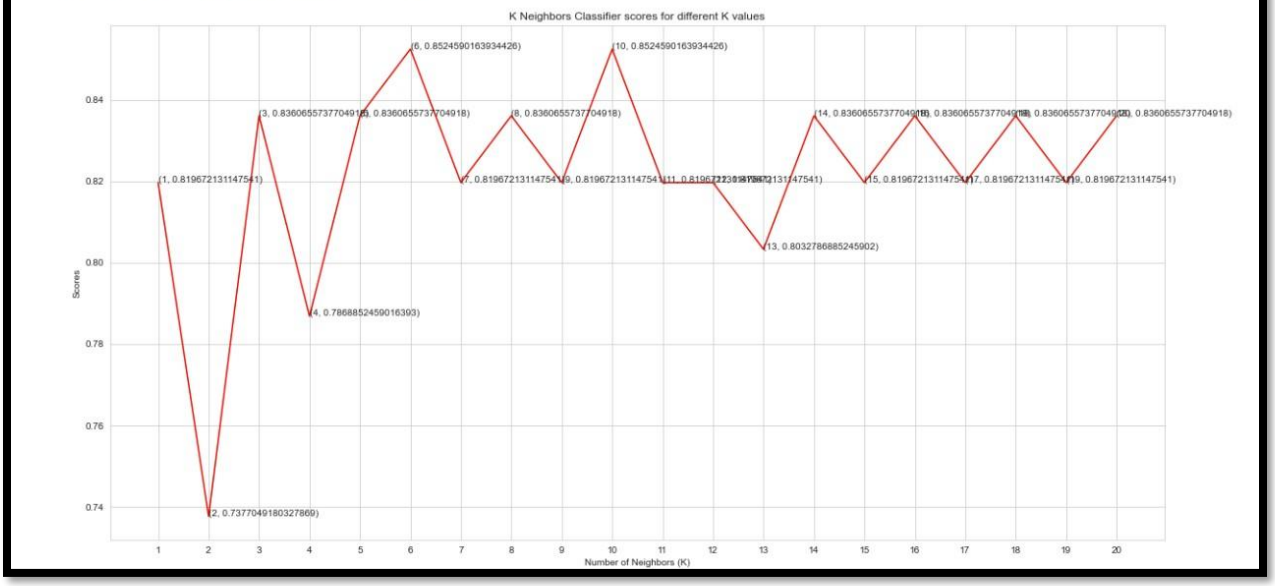

Figure 16

Fig 4. Finding the best value of ‘k’ from a range of values

This graph clearly shows that the score at k = 10 is 0.85, which is the highest of all the k values in the range (1, 21). As a result, we chose k = 10 for subsequent operations.

We used accuracy score techniques to predict the score because it produces the most accurate results from the model. The accuracy value = 0.85 was obtained by applying this k value to the KNN classifier.

```
knn_classifier = KNeighborsClassifier(n_neighbors=10)
knn_classifier.fit(X_train,y_train)
y_pred = knn_classifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.78      0.82        27
           1       0.84      0.91      0.87        34

    accuracy                           0.85        61
   macro avg       0.86      0.84      0.85        61
weighted avg       0.85      0.85      0.85        61
```

Figure 17

Decision Tree Algorithm:

To apply the decision tree algorithm to our dataset, we must first import the decision tree classifier from the sklearn module. A decision tree problem is an example of an NP-complete problem. In practice, this algorithm employs heuristics and is a greedy algorithm. As a result, each node has a locally optimal solution. This means that the algorithm makes no guarantees about returning the global optimal tree. As a result, we must use a random seed to control the random choices; it can be any number. We've chosen random state = 7. The accuracy value in this dataset is 0.79. By adjusting the parameters, these values can be improved even further.

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
decision_tree_classifier = DecisionTreeClassifier(random_state=7)
decision_tree_classifier.fit(X_train,y_train)
y_pred_dt = decision_tree_classifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.75      0.78      0.76        27
           1       0.82      0.79      0.81        34

    accuracy                           0.79        61
   macro avg       0.78      0.79      0.78        61
weighted avg       0.79      0.79      0.79        61
```

Figure 18

Comparison – Results and Discussion:

On our dataset, we used two classification algorithms to predict heart disease and see which model would give us the highest accuracy score. With k = 10, we get an accuracy value of 0.85 for KNN. The accuracy value for decision tree classification is 0.79. We have compared the two classification algorithms in terms of their accuracy score 0.85 for k-nearest neighbors and 0.79 for decision tree. At the end of our performance, we found out that the K-Nearest Neighbors (KNN) classification algorithm has higher accuracy score than the accuracy score of the Decision tree classification algorithm. From these results in our paper, the K-Nearest Neighbors model has the highest accuracy score.

Conclusion:

In our case, we have learned many libraries and functions such as sk-learn for importing and usin g several machine learning techniques also functions to split the dataset into train set and test set also how to apply standardization to make some features on one scale and how to make feature s election to see the positive and negative correlation between all the features in our dataset. In app lying the K-Nearest Neighbors (KNN) classification algorithm we used accuracy score to see wh ich k is the best one and to get the best accuracy this score accuracy used also with the decision tr ee classification algorithm last thing is classification report we learned from it how to compare between precision, recall, f1-score and support and get from it the accuracy score.

In this paper, we used only two machine learning algorithms to get the accuracy value, but other machine learning algorithms can be used on the dataset and may perform better. By adjusting tuning parameters and other adjustment parameters, we can also improve the performance of these algorithms.

References:

1. Aha DW (1997) Lazy learning. Kluwer academic publishers, Berlin
2. Chen, A.H., Huang, S.Y., Hong, P.S., Cheng, C.H., Lin, E.J.: HDPS: Heart disease prediction system. In: 2011 Computing in Cardiology IEEE, pp. 557–560 (2011)
3. G. Stein, B. Chen, A. S. Wu, and K. A. Hua, "Decision tree classifier for network intrusion detection with GA-based feature selection," in Proceedings of the 43rd annual Southeast regional conferenceVolume 2, 2005, pp. 136–141.
4. H. Bhavsar and A. Ganatra, "A Comparative Study of Training Algorithms for Supervised Machine Learning", International Journal of Soft Computing and Engineering (IJSCE), Vol. 2, Issue. 4, September 2012
5. Rairikar, A., Kulkarni, V., Sabale, V., Kale, H., Lamgunde, A.: Heart disease prediction using data mining techniques. In: Intelligent Computing and Control (I2C2), IEEE International Conference on 2017 Jun, pp. 1–8 (2007)
6. S.Archana and Dr. K.Elangovan, "Survey of Classification Techniques in Data Mining", International Journal of Computer Science and Mobile Applications, Vol. 2 Issue. 2, February 2014.
7. Santhana Krishnan, J., Geetha, S.: Prediction of heart disease using machine learning algorithms. In: 1st International Conference on Innovations in Information and Communication Technology (ICIICT), IEEE (2019)