Diabetes Detection using DT and KNN - 21F21484 VIVEIK CATARAM SAICHANDRA

January 4, 2025

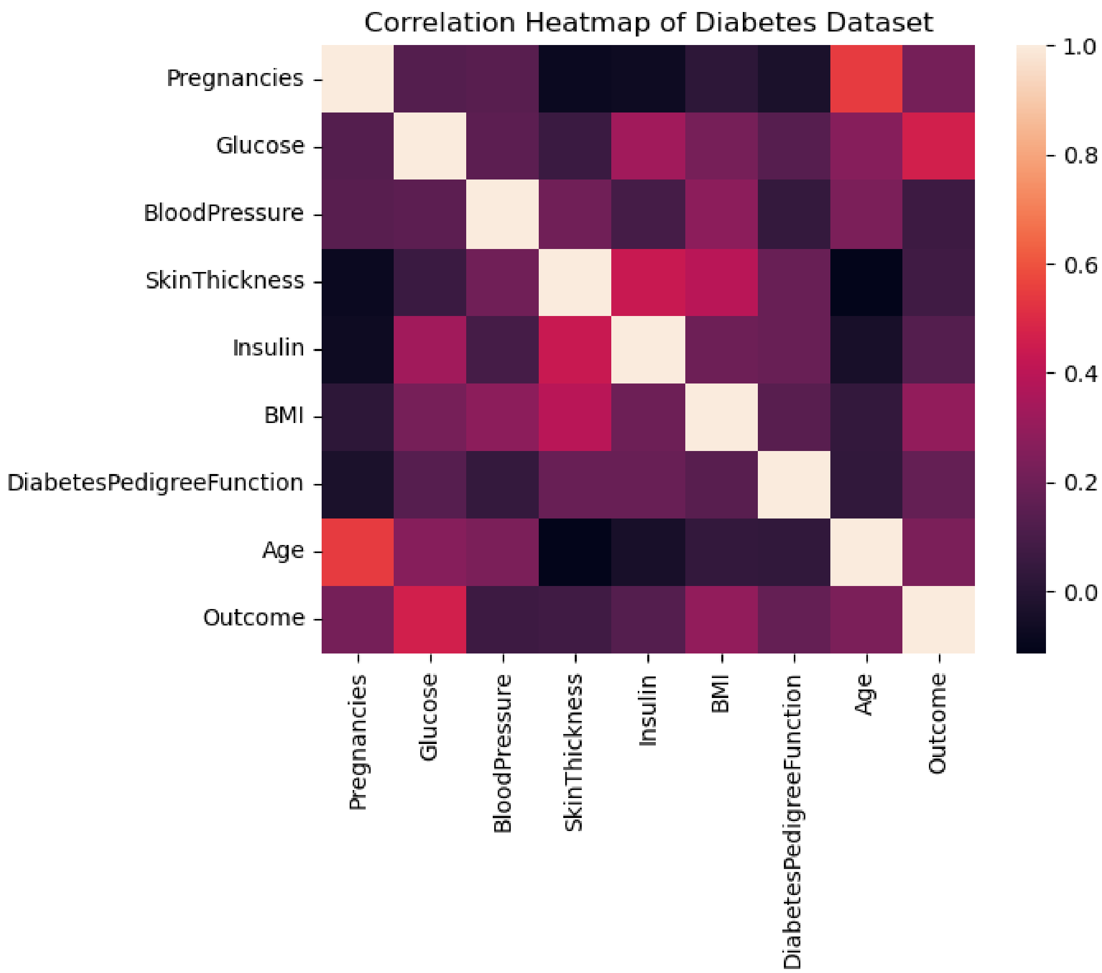
[1]:
```
#Importing Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
#Importing and Reading the Dataset
dataset = pd.read_csv('diabetes.csv')
dataset.head()
```

[2]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[3]:
```
#Plotting Heatmap of the Dataset
plt.figure(1)
sns.heatmap(dataset.corr())
plt.title('Correlation Heatmap of Diabetes Dataset')
plt.show()
```



[4]:
```
# Replace 0 values with the median of the feature's distribution to handle Zero␣
↪(0) values
columns_with_zeros = ['Pregnancies', 'Glucose', 'BloodPressure',␣
↪'SkinThickness', 'Insulin', 'BMI'] #List contatining parameters with 0 values
for column in columns_with_zeros:
    median = dataset[column].median()
    dataset[column] = dataset[column].replace(0, median)

dataset.head()
```

[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	27	33.6	
1	1	85	66	29	27	26.6	
2	8	183	64	23	27	23.3	
3	1	89	66	23	94	28.1	
4	3	137	40	35	168	43.1	

```
   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[5]:
```
#Splitting dataset into train-test data
from sklearn.model_selection import train_test_split

x = dataset.drop(columns = ['Outcome']) #Features
y = dataset['Outcome'] #Target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)␣
 ↪#Train-Test Split
```

[6]:
```
#DECISION TREE (DT)
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, roc_curve

dt_classifier = DecisionTreeClassifier() #Decision Tree Classifier model
dt_classifier.fit(x_train, y_train) #Training the Classifier with the training␣
 ↪data
dt_pred = dt_classifier.predict(x_test) #Using the trained Classifier model to␣
 ↪predict the testing data

dt_model = DecisionTreeClassifier(criterion='entropy') #Decision Tree␣
 ↪Classifier model for calculating Information Gain
dt_model.fit(x_train, y_train) #Training the Classifier with the training data
importances = dt_model.feature_importances_ #Retrieving the Feature importances
tree_structure = dt_model.tree_ #Accessing the tree structure for calculating␣
 ↪the Gini index and Entropy

# Displaying the Gini index and Entropy for each node (split) in the tree
for i in range(tree_structure.node_count):
    if tree_structure.children_left[i] != tree_structure.children_right[i]:  #␣
 ↪If condition for the case where it is a split node
        gini_index = tree_structure.impurity[i]  # Gini index impurity at the␣
 ↪split node
        print(f'Node {i} Gini index: {gini_index}')
        entropy_value = -np.sum(tree_structure.weighted_n_node_samples[i]) * np.
 ↪log2(tree_structure.weighted_n_node_samples[i])  # Calculating the Entropy
        print(f'Node {i} Entropy: {entropy_value}')

dt_report = classification_report(y_test, dt_pred) #Classification Report
```

```
dt_matrix = confusion_matrix(y_test, dt_pred) #Confusion Matrix
dt_accuracy = accuracy_score(y_test, dt_pred) #Accuracy Score
dt_fp_rate, dt_tp_rate, dt_threshold = roc_curve(y_test, dt_pred) #ROC Curve
dt_sensitivity = dt_matrix[0,0] / (dt_matrix[0,0] + dt_matrix[0,1])␣
 ↪#Sensitivity = TP / (TP + FN)
dt_specificity = dt_matrix[1,1] / (dt_matrix[1,1] + dt_matrix[1,0])␣
 ↪#Specificity = TN / (TN + FP)
dt_precision = dt_matrix[0,0] / (dt_matrix[0,0] + dt_matrix[1,0]) #Precision =␣
 ↪TP / (TP + FP)


print(dt_report)
print(dt_matrix)
print('Decision Tree Accuracy = ',dt_accuracy)
print('Decision Tree Sensitivity = ',dt_sensitivity)
print('Decision Tree Specificity = ',dt_specificity)
print('Decision Tree Precision = ',dt_precision)
plt.title('ROC Curve - Decision Tree')
plt.plot(dt_fp_rate, dt_tp_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
Node 0 Gini index: 0.929418954323556
Node 0 Entropy: -5292.490672488443
Node 1 Gini index: 0.705098104499076
Node 1 Entropy: -2849.3429560032205
Node 3 Gini index: 0.8112781244591328
Node 3 Entropy: -2085.8156313873983
Node 4 Gini index: 0.5200579828858849
Node 4 Entropy: -972.4303953655921
Node 5 Gini index: 0.4618043316752015
Node 5 Entropy: -938.3525639216582
Node 6 Gini index: 0.26014536394771426
Node 6 Entropy: -592.2093122580814
Node 8 Gini index: 0.4689955935892812
Node 8 Entropy: -212.8771237954945
Node 9 Gini index: 0.7424875695421236
Node 9 Entropy: -80.71062275542812
Node 11 Gini index: 0.9709505944546686
Node 11 Entropy: -33.219280948873624
Node 12 Gini index: 0.5916727785823275
Node 12 Entropy: -19.651484454403228
Node 17 Gini index: 0.74959525725948
Node 17 Entropy: -226.47733175670794
```

```
Node 18 Gini index: 0.37123232664087563
Node 18 Entropy: -134.6059378176129
Node 20 Gini index: 0.9182958340544896
Node 20 Entropy: -15.509775004326936
Node 22 Gini index: 0.9182958340544896
Node 22 Entropy: -4.754887502163468
Node 25 Gini index: 1.0
Node 25 Entropy: -53.302968908806456
Node 26 Gini index: 0.7642045065086203
Node 26 Entropy: -28.52932501298081
Node 27 Gini index: 1.0
Node 27 Entropy: -8.0
Node 32 Gini index: 0.8112781244591328
Node 32 Entropy: -8.0
Node 35 Gini index: 0.9699914856791789
Node 35 Entropy: -853.9292841567265
Node 36 Gini index: 0.8609652558547649
Node 36 Entropy: -568.4299824400822
Node 37 Gini index: 0.2580186686648155
Node 37 Entropy: -104.0419249893113
Node 39 Gini index: 0.9182958340544896
Node 39 Entropy: -4.754887502163468
Node 42 Gini index: 0.9500796252338518
Node 42 Entropy: -391.4539078468495
Node 44 Gini index: 0.9747785474909672
Node 44 Entropy: -347.07593991234864
Node 46 Gini index: 0.9935704757706079
Node 46 Entropy: -303.57978409184955
Node 47 Gini index: 1.0
Node 47 Entropy: -268.0782000346155
Node 48 Gini index: 0.9819407868640977
Node 48 Entropy: -199.42124551085624
Node 50 Gini index: 0.9640787648082292
Node 50 Entropy: -186.11730005192322
Node 51 Gini index: 0.907165767573082
Node 51 Entropy: -153.58008562199313
Node 53 Gini index: 0.9402859586706309
Node 53 Entropy: -134.6059378176129
Node 55 Gini index: 0.8904916402194913
Node 55 Entropy: -122.21143267166839
Node 56 Gini index: 0.4689955935892812
Node 56 Entropy: -33.219280948873624
Node 57 Gini index: 1.0
Node 57 Entropy: -2.0
Node 61 Gini index: 0.9886994082884974
Node 61 Entropy: -64.0
Node 62 Gini index: 0.8112781244591328
Node 62 Entropy: -43.01955000865387
```

```
Node 63 Gini index: 1.0
Node 63 Entropy: -15.509775004326936
Node 65 Gini index: 0.8112781244591328
Node 65 Entropy: -8.0
Node 70 Gini index: 0.7219280948873623
Node 70 Entropy: -11.60964047443681
Node 73 Gini index: 0.7219280948873623
Node 73 Entropy: -33.219280948873624
Node 77 Gini index: 0.898058793450166
Node 77 Entropy: -179.5249055930738
Node 78 Gini index: 0.975119064940866
Node 78 Entropy: -128.38196255841365
Node 79 Gini index: 0.6500224216483541
Node 79 Entropy: -15.509775004326936
Node 82 Gini index: 0.863120568566631
Node 82 Entropy: -92.23866587835397
Node 84 Gini index: 0.9544340029249649
Node 84 Entropy: -64.0
Node 85 Gini index: 1.0
Node 85 Entropy: -43.01955000865387
Node 86 Gini index: 0.9709505944546686
Node 86 Entropy: -33.219280948873624
Node 88 Gini index: 0.9852281360342516
Node 88 Entropy: -19.651484454403228
Node 89 Gini index: 0.7219280948873623
Node 89 Entropy: -11.60964047443681
Node 96 Gini index: 0.9885081741986363
Node 96 Entropy: -1878.9666276672906
Node 97 Gini index: 0.828055725379504
Node 97 Entropy: -421.4881875176937
Node 98 Gini index: 0.2580186686648155
Node 98 Entropy: -104.0419249893113
Node 99 Gini index: 0.9182958340544896
Node 99 Entropy: -4.754887502163468
Node 103 Gini index: 0.9503376699710269
Node 103 Entropy: -254.0838499786226
Node 104 Gini index: 0.9952525494396791
Node 104 Entropy: -192.74977452827116
Node 105 Gini index: 0.8708644692353646
Node 105 Entropy: -110.03910001730775
Node 107 Gini index: 0.9886994082884974
Node 107 Entropy: -64.0
Node 108 Gini index: 0.9456603046006402
Node 108 Entropy: -38.05374780501027
Node 110 Gini index: 0.7642045065086203
Node 110 Entropy: -28.52932501298081
Node 112 Gini index: 0.5435644431995964
Node 112 Entropy: -24.0
```

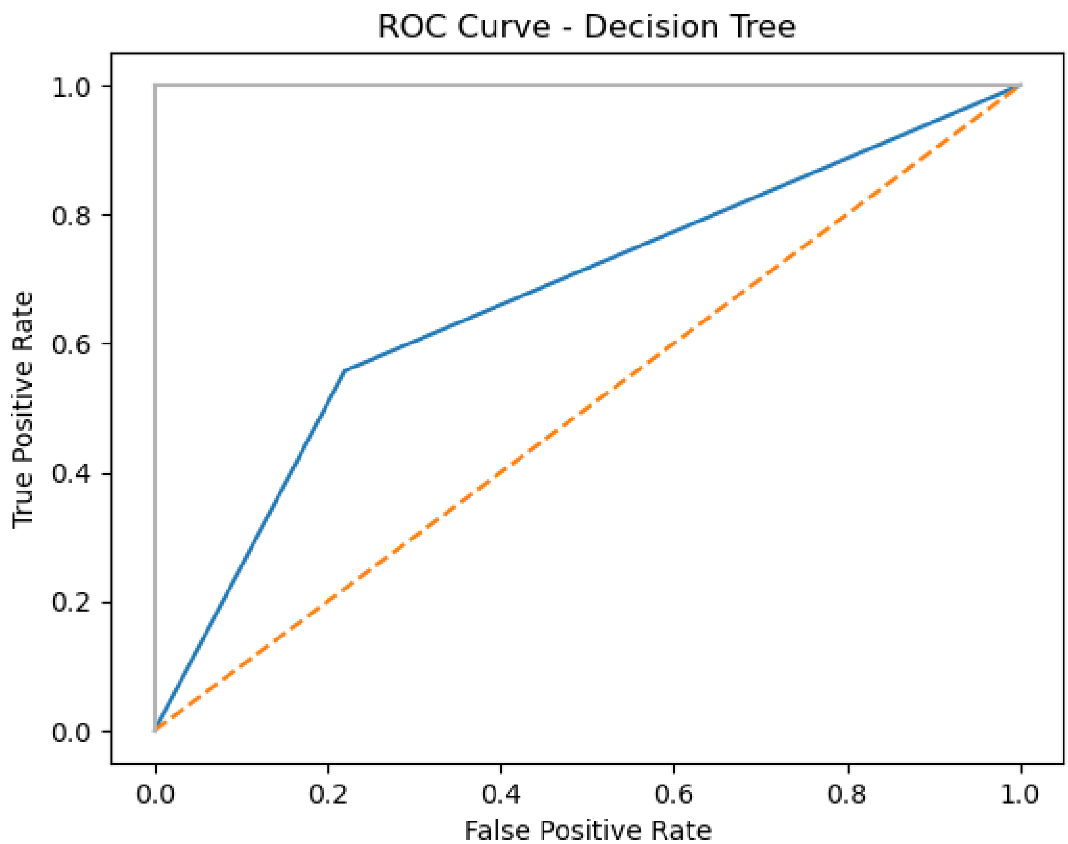
```
Node 113 Gini index: 1.0
Node 113 Entropy: -2.0
Node 118 Gini index: 0.7793498372920852
Node 118 Entropy: -48.105716335834195
Node 120 Gini index: 0.9709505944546686
Node 120 Entropy: -11.60964047443681
Node 124 Gini index: 0.8972961254736168
Node 124 Entropy: -1250.7486247316892
Node 125 Gini index: 0.9649567669505688
Node 125 Entropy: -853.9292841567265
Node 126 Gini index: 0.9971803988942642
Node 126 Entropy: -632.156400069231
Node 127 Gini index: 0.9940302114769565
Node 127 Entropy: -482.54256363350737
Node 129 Gini index: 0.9798687566511527
Node 129 Entropy: -444.23460010384645
Node 131 Gini index: 0.9940302114769565
Node 131 Entropy: -398.93001187765793
Node 132 Gini index: 0.9998061328047111
Node 132 Entropy: -361.7749775913361
Node 133 Gini index: 0.9980008838722996
Node 133 Entropy: -332.47473080739024
Node 134 Gini index: 0.9023932827949789
Node 134 Entropy: -98.10749561002054
Node 135 Gini index: 0.9957274520849256
Node 135 Entropy: -48.105716335834195
Node 136 Gini index: 0.5916727785823275
Node 136 Entropy: -19.651484454403228
Node 139 Gini index: 0.6500224216483541
Node 139 Entropy: -15.509775004326936
Node 143 Gini index: 0.9275265884316759
Node 143 Entropy: -179.5249055930738
Node 144 Gini index: 0.4394969869215134
Node 144 Entropy: -38.05374780501027
Node 147 Gini index: 0.9949848281859701
Node 147 Entropy: -110.03910001730775
Node 148 Gini index: 0.9774178175281716
Node 148 Entropy: -69.48686830125577
Node 150 Gini index: 0.9957274520849256
Node 150 Entropy: -48.105716335834195
Node 152 Gini index: 0.9940302114769565
Node 152 Entropy: -38.05374780501027
Node 154 Gini index: 0.9182958340544896
Node 154 Entropy: -28.52932501298081
Node 155 Gini index: 0.8112781244591328
Node 155 Entropy: -24.0
Node 156 Gini index: 1.0
Node 156 Entropy: -8.0
```

```
Node 161 Gini index: 0.5916727785823275
Node 161 Entropy: -19.651484454403228
Node 166 Gini index: 0.6292492238560345
Node 166 Entropy: -80.71062275542812
Node 168 Gini index: 0.9544340029249649
Node 168 Entropy: -24.0
Node 169 Gini index: 0.8112781244591328
Node 169 Entropy: -8.0
Node 173 Gini index: 0.5032583347756457
Node 173 Entropy: -128.38196255841365
Node 174 Gini index: 0.9182958340544896
Node 174 Entropy: -4.754887502163468
Node 177 Gini index: 0.24988229283318547
Node 177 Entropy: -110.03910001730775
Node 179 Gini index: 1.0
Node 179 Entropy: -2.0
Node 182 Gini index: 0.4959690720618337
Node 182 Entropy: -254.0838499786226
Node 184 Gini index: 0.676941869780886
Node 184 Entropy: -134.6059378176129
Node 185 Gini index: 0.5293608652873644
Node 185 Entropy: -116.09640474436812
Node 186 Gini index: 0.8453509366224365
Node 186 Entropy: -38.05374780501027
Node 187 Gini index: 0.9709505944546686
Node 187 Entropy: -11.60964047443681
Node 189 Gini index: 0.9182958340544896
Node 189 Entropy: -4.754887502163468
Node 194 Gini index: 0.9182958340544896
Node 194 Entropy: -4.754887502163468
              precision    recall  f1-score   support

           0       0.76      0.78      0.77       123
           1       0.59      0.56      0.57        70

    accuracy                           0.70       193
   macro avg       0.67      0.67      0.67       193
weighted avg       0.70      0.70      0.70       193

[[96 27]
 [31 39]]
Decision Tree Accuracy =  0.6994818652849741
Decision Tree Sensitivity =  0.7804878048780488
Decision Tree Specificity =  0.5571428571428572
Decision Tree Precision =  0.7559055118110236
```



[7]:
```python
#K-NEAREST NEIGHBOUR (KNN)
from sklearn.preprocessing import MinMaxScaler

#Normalization Preproccesing Technique
scaler = MinMaxScaler()

#Columns that require normalization
scale_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
normalized_data = scaler.fit_transform(dataset[scale_columns])
normalized_dataset = pd.DataFrame(normalized_data, columns = scale_columns)␣
↪#Converting normalized data into dataset
normalized_dataset['Outcome'] = dataset['Outcome'] #Adding the unaltered Target␣
↪variable to the Normalized Dataset
normalized_dataset.head()
```

[7]:
```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0       0.3125  0.670968       0.489796       0.304348  0.015625  0.314928
1       0.0000  0.264516       0.428571       0.239130  0.015625  0.171779
```

```
2       0.4375  0.896774       0.408163       0.173913  0.015625  0.104294
3       0.0000  0.290323       0.428571       0.173913  0.096154  0.202454
4       0.1250  0.600000       0.163265       0.304348  0.185096  0.509202

   DiabetesPedigreeFunction       Age  Outcome
0                  0.234415  0.483333        1
1                  0.116567  0.166667        0
2                  0.253629  0.183333        1
3                  0.038002  0.000000        0
4                  0.943638  0.200000        1
```

[8]:
```
#Splitting dataset into train-test data
x_normal = normalized_dataset.drop(columns = 'Outcome') #Features
y_normal = normalized_dataset['Outcome'] #Target
x_train_normal, x_test_normal, y_train_normal, y_test_normal =␣
↪train_test_split(x_normal, y_normal, test_size = 0.25) #Train-Test Split
```

[9]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 5) #K-Nearest Neighbor␣
↪Classifier model
knn_classifier.fit(x_train_normal, y_train_normal) #Training the Classifier␣
↪with the training data
knn_pred = knn_classifier.predict(x_test_normal) #Using the trained Classifier␣
↪model to predict the testing data

knn_report = classification_report(y_test_normal, knn_pred) #Classification␣
↪Report
knn_matrix = confusion_matrix(y_test_normal, knn_pred) #Confusion Matrix
knn_accuracy = accuracy_score(y_test_normal, knn_pred) #Accuracy Score
knn_fp_rate, knn_tp_rate, knn_threshold = roc_curve(y_test_normal, knn_pred)␣
↪#ROC Curve
knn_sensitivity = knn_matrix[0,0] / (knn_matrix[0,0] + knn_matrix[0,1])␣
↪#Sensitivity = TP / (TP + FN)
knn_specificity = knn_matrix[1,1] / (knn_matrix[1,1] + knn_matrix[1,0])␣
↪#Specificity = TN / (TN + FP)
knn_precision = knn_matrix[0,0] / (knn_matrix[0,0] + knn_matrix[1,0])␣
↪#Precision = TP / (TP + FP)


print(knn_report)
print(knn_matrix)
print('K-Nearest Neighbor Accuracy = ',knn_accuracy)
print('K-Nearest Neighbor Sensitivity = ',knn_sensitivity)
print('K-Nearest Neighbor Specificity = ',knn_specificity)
print('K-Nearest Neighbor Precision = ',knn_precision)
plt.title('ROC Curve - K-Nearest Neighbor')
```

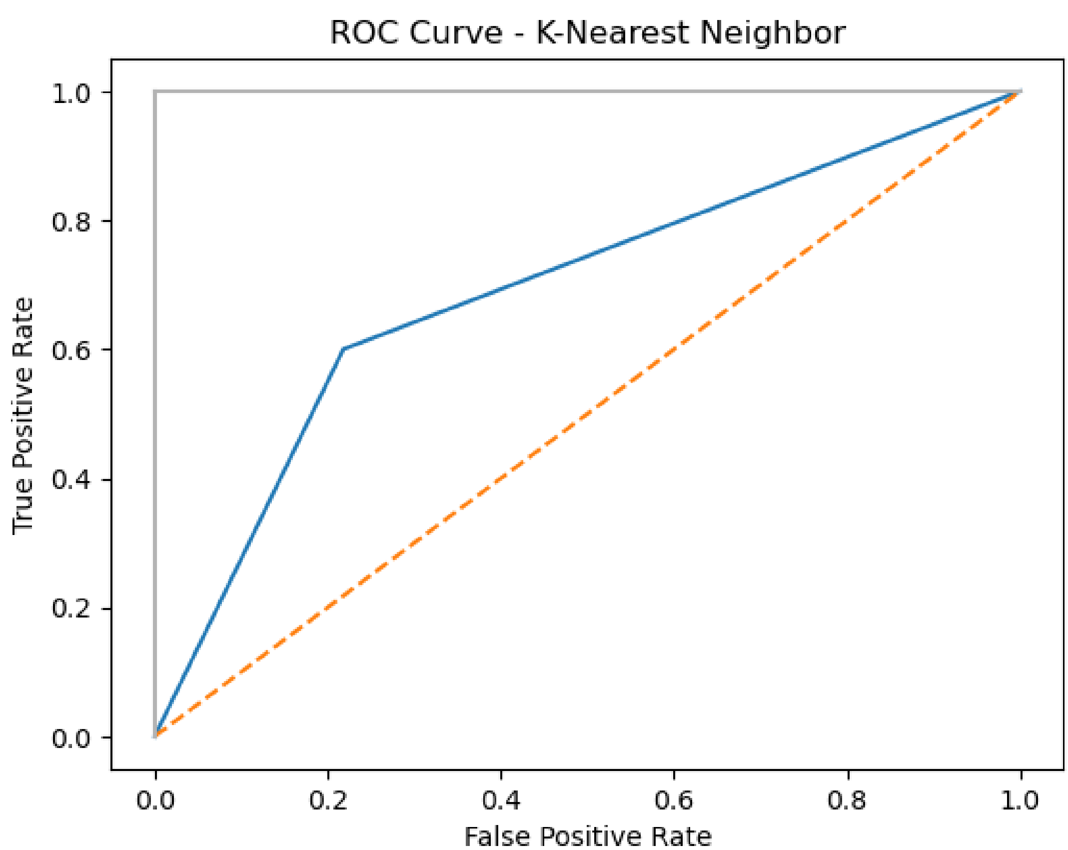
```
plt.plot(knn_fp_rate, knn_tp_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.80       133
           1       0.55      0.60      0.58        60

    accuracy                           0.73       193
   macro avg       0.68      0.69      0.69       193
weighted avg       0.73      0.73      0.73       193

[[104  29]
 [ 24  36]]
K-Nearest Neighbor Accuracy =  0.7253886010362695
K-Nearest Neighbor Sensitivity =  0.7819548872180451
K-Nearest Neighbor Specificity =  0.6
K-Nearest Neighbor Precision =  0.8125
```



```
[ ]:
```