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

January 6, 2025

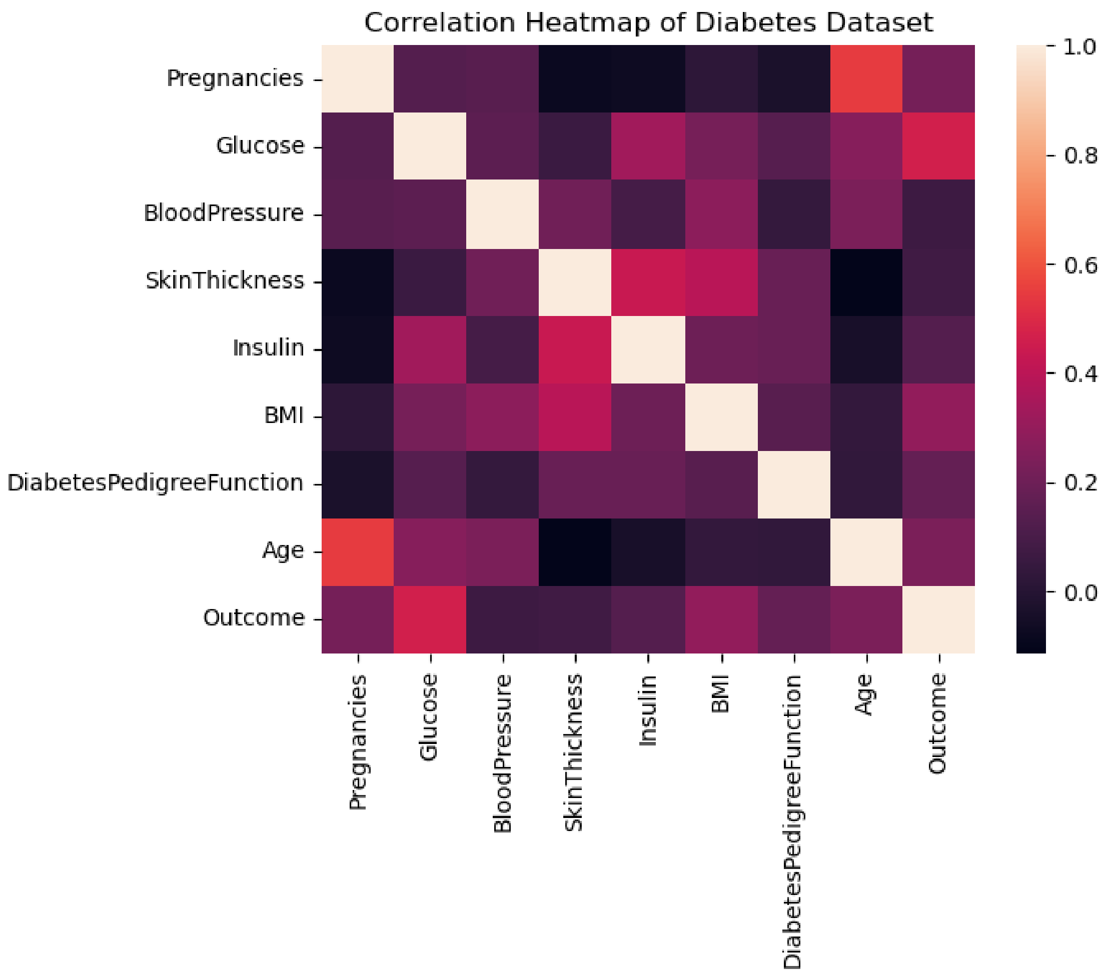
[1]:
```
#Importing Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[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
```

[23]:
```
#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
```

[24]:
```
#DECISION TREE (DT)
from sklearn.tree import DecisionTreeClassifier, plot_tree
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}')

#Plotting the Decision Tree
```

```python
plt.figure(figsize=(15,10))
plot_tree(dt_classifier,feature_names = x.columns,class_names=['No Diabetes',
    'Diabetes'],filled=True,fontsize=6)
plt.title('Diabetes Prediction Decision Tree')
plt.show()

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()
```

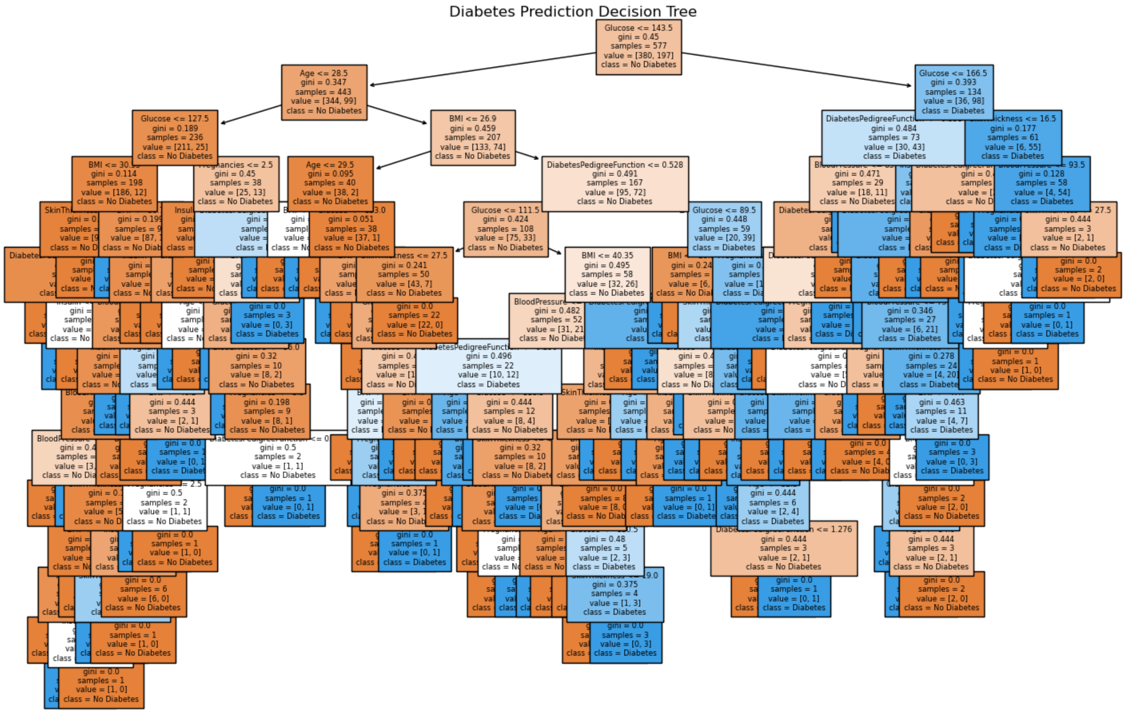
```
Node 0 Gini index: 0.926172154055073
Node 0 Entropy: -5292.490672488443
Node 1 Gini index: 0.7664645115676525
Node 1 Entropy: -3894.485159629873
Node 2 Gini index: 0.4875229918975016
Node 2 Entropy: -1860.3037596493946
Node 3 Gini index: 0.32984607020714635
Node 3 Entropy: -1510.6126107757627
Node 4 Gini index: 0.08079313589591118
Node 4 Entropy: -664.3856189774724
Node 5 Gini index: 0.4394969869215134
Node 5 Entropy: -38.05374780501027
Node 7 Gini index: 1.0
Node 7 Entropy: -2.0
Node 11 Gini index: 0.5066503344840895
```

```
Node 11 Entropy: -648.2415647232904
Node 12 Gini index: 0.6442142137378307
Node 12 Entropy: -406.42797576067073
Node 13 Gini index: 0.5547781633412736
Node 13 Entropy: -369.16017124398627
Node 15 Gini index: 0.5140912790181235
Node 15 Entropy: -361.7749775913361
Node 16 Gini index: 0.4689955935892812
Node 16 Entropy: -354.41343573651113
Node 17 Gini index: 0.9709505944546686
Node 17 Entropy: -11.60964047443681
Node 20 Gini index: 0.3760198509692728
Node 20 Entropy: -317.9747842438563
Node 21 Gini index: 0.49418293484978865
Node 21 Entropy: -192.74977452827116
Node 22 Gini index: 0.3227569588973983
Node 22 Entropy: -172.97373660251154
Node 24 Gini index: 0.8112781244591328
Node 24 Entropy: -24.0
Node 26 Gini index: 1.0
Node 26 Entropy: -8.0
Node 28 Gini index: 0.9182958340544896
Node 28 Entropy: -4.754887502163468
Node 31 Gini index: 0.9182958340544896
Node 31 Entropy: -4.754887502163468
Node 36 Gini index: 0.9709505944546686
Node 36 Entropy: -11.60964047443681
Node 38 Gini index: 0.9182958340544896
Node 38 Entropy: -4.754887502163468
Node 42 Gini index: 0.9268190639645772
Node 42 Entropy: -199.42124551085624
Node 43 Gini index: 0.3095434291503252
Node 43 Entropy: -75.05865002596161
Node 44 Gini index: 0.7219280948873623
Node 44 Entropy: -11.60964047443681
Node 46 Gini index: 1.0
Node 46 Entropy: -2.0
Node 50 Gini index: 0.9709505944546686
Node 50 Entropy: -86.43856189774725
Node 51 Gini index: 0.9852281360342516
Node 51 Entropy: -53.302968908806456
Node 52 Gini index: 0.8453509366224365
Node 52 Entropy: -38.05374780501027
Node 53 Gini index: 0.9852281360342516
Node 53 Entropy: -19.651484454403228
Node 54 Gini index: 0.7219280948873623
Node 54 Entropy: -11.60964047443681
Node 61 Gini index: 0.9405781991505306
```

```
Node 61 Entropy: -1592.5518002023603
Node 62 Gini index: 0.18717625687320816
Node 62 Entropy: -179.5249055930738
Node 64 Gini index: 0.7219280948873623
Node 64 Entropy: -11.60964047443681
Node 67 Gini index: 0.9834537187362689
Node 67 Entropy: -1277.3175378087608
Node 68 Gini index: 0.885612871398971
Node 68 Entropy: -762.4237512704516
Node 70 Gini index: 0.9440870182837795
Node 70 Entropy: -616.1313520576979
Node 71 Gini index: 0.9736680645496201
Node 71 Entropy: -536.9546635134159
Node 73 Gini index: 0.9837082626231857
Node 73 Entropy: -505.754247590989
Node 75 Gini index: 0.9919924034538556
Node 75 Entropy: -474.8424910217125
Node 76 Gini index: 0.8112781244591328
Node 76 Entropy: -43.01955000865387
Node 77 Gini index: 0.4689955935892812
Node 77 Entropy: -33.219280948873624
Node 81 Gini index: 0.9652016987500656
Node 81 Entropy: -384.0
Node 82 Gini index: 0.905200296956048
Node 82 Entropy: -303.57978409184955
Node 83 Gini index: 0.9649567669505688
Node 83 Entropy: -219.65963218934144
Node 85 Gini index: 0.9867867202680318
Node 85 Entropy: -192.74977452827116
Node 86 Gini index: 0.9991421039919088
Node 86 Entropy: -140.88144885869957
Node 87 Gini index: 0.9828586897127056
Node 87 Entropy: -122.21143267166839
Node 88 Gini index: 0.6840384356390417
Node 88 Entropy: -38.05374780501027
Node 90 Gini index: 1.0
Node 90 Entropy: -8.0
Node 93 Gini index: 0.9709505944546686
Node 93 Entropy: -58.60335893412778
Node 95 Gini index: 1.0
Node 95 Entropy: -43.01955000865387
Node 96 Gini index: 0.9709505944546686
Node 96 Entropy: -33.219280948873624
Node 98 Gini index: 0.9182958340544896
Node 98 Entropy: -15.509775004326936
Node 100 Gini index: 0.9182958340544896
Node 100 Entropy: -4.754887502163468
Node 105 Gini index: 0.5435644431995964
```

```
Node 105 Entropy: -24.0
Node 108 Gini index: 0.41381685030363374
Node 108 Entropy: -43.01955000865387
Node 111 Gini index: 0.8453509366224365
Node 111 Entropy: -38.05374780501027
Node 112 Gini index: 0.8112781244591328
Node 112 Entropy: -8.0
Node 117 Gini index: 0.934068055375491
Node 117 Entropy: -354.41343573651113
Node 119 Gini index: 0.885612871398971
Node 119 Entropy: -325.2118756352258
Node 120 Gini index: 0.97663491144401
Node 120 Entropy: -206.1306865356277
Node 121 Gini index: 0.9975025463691153
Node 121 Entropy: -172.97373660251154
Node 123 Gini index: 0.9709505944546686
Node 123 Entropy: -147.20671786825557
Node 124 Gini index: 0.961236604722876
Node 124 Entropy: -48.105716335834195
Node 126 Gini index: 0.8453509366224365
Node 126 Entropy: -38.05374780501027
Node 128 Gini index: 0.8112781244591328
Node 128 Entropy: -8.0
Node 131 Gini index: 0.7871265862012691
Node 131 Entropy: -69.48686830125577
Node 133 Gini index: 0.9709505944546686
Node 133 Entropy: -33.219280948873624
Node 135 Gini index: 0.8112781244591328
Node 135 Entropy: -24.0
Node 137 Gini index: 0.9182958340544896
Node 137 Entropy: -4.754887502163468
Node 141 Gini index: 0.3227569588973983
Node 141 Entropy: -69.48686830125577
Node 144 Gini index: 0.8395304981054318
Node 144 Entropy: -946.8559515213415
Node 145 Gini index: 0.9770012394218561
Node 145 Entropy: -451.8571927982413
Node 146 Gini index: 0.9575534837147482
Node 146 Entropy: -140.88144885869957
Node 147 Gini index: 0.8554508105601307
Node 147 Entropy: -116.09640474436812
Node 148 Gini index: 0.9886994082884974
Node 148 Entropy: -64.0
Node 150 Gini index: 0.9957274520849256
Node 150 Entropy: -48.105716335834195
Node 151 Gini index: 0.9709505944546686
Node 151 Entropy: -33.219280948873624
Node 152 Gini index: 0.9182958340544896
```

```
Node 152 Entropy: -15.509775004326936
Node 159 Gini index: 0.8453509366224365
Node 159 Entropy: -240.21499122004107
Node 160 Gini index: 0.6892019851173655
Node 160 Entropy: -199.42124551085624
Node 162 Gini index: 0.8112781244591328
Node 162 Entropy: -134.6059378176129
Node 163 Gini index: 0.9023932827949789
Node 163 Entropy: -98.10749561002054
Node 164 Gini index: 0.9640787648082292
Node 164 Entropy: -75.05865002596161
Node 165 Gini index: 0.9709505944546686
Node 165 Entropy: -33.219280948873624
Node 166 Gini index: 0.8112781244591328
Node 166 Entropy: -24.0
Node 167 Gini index: 1.0
Node 167 Entropy: -8.0
Node 169 Gini index: 0.9182958340544896
Node 169 Entropy: -4.754887502163468
Node 174 Gini index: 0.5435644431995964
Node 174 Entropy: -24.0
Node 176 Gini index: 1.0
Node 176 Entropy: -2.0
Node 181 Gini index: 0.6500224216483541
Node 181 Entropy: -15.509775004326936
Node 184 Gini index: 0.46377734988775166
Node 184 Entropy: -361.7749775913361
Node 185 Gini index: 0.9182958340544896
Node 185 Entropy: -4.754887502163468
Node 188 Gini index: 0.362051251733998
Node 188 Entropy: -339.76289771739914
Node 189 Gini index: 0.225363639127395
Node 189 Entropy: -317.9747842438563
Node 190 Gini index: 0.13503620280212764
Node 190 Entropy: -303.57978409184955
Node 192 Gini index: 1.0
Node 192 Entropy: -2.0
Node 195 Gini index: 1.0
Node 195 Entropy: -2.0
Node 198 Gini index: 0.9182958340544896
Node 198 Entropy: -4.754887502163468
```

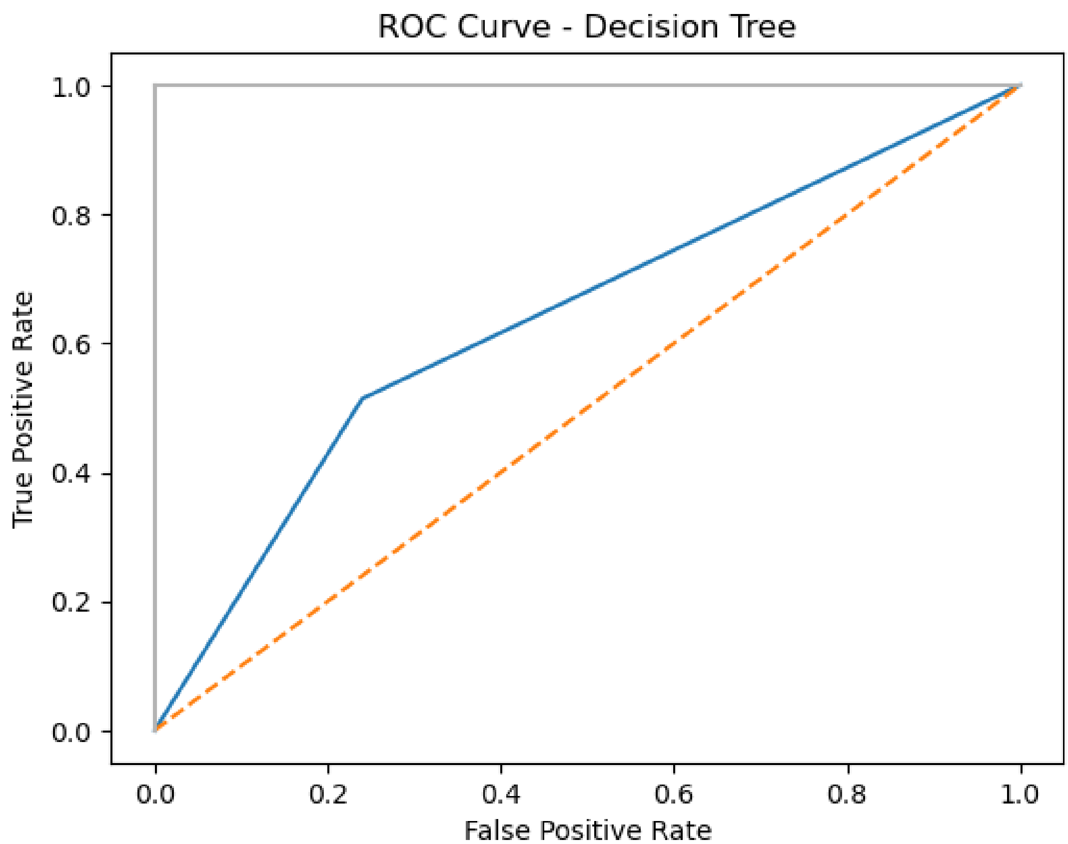



```
              precision    recall  f1-score   support

           0       0.72      0.76      0.74       121
           1       0.56      0.51      0.54        72

    accuracy                           0.67       193
   macro avg       0.64      0.64      0.64       193
weighted avg       0.66      0.67      0.67       193

[[92 29]
 [35 37]]
Decision Tree Accuracy =  0.6683937823834197
Decision Tree Sensitivity =  0.7603305785123967
Decision Tree Specificity =  0.5138888888888888
Decision Tree Precision =  0.7244094488188977
```



[18]:
```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()
```

[18]:

	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
```

[19]:
```
#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
```

[20]:
```
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       0.76      0.81      0.78       126
           1       0.59      0.52      0.56        67

    accuracy                           0.71       193
   macro avg       0.68      0.67      0.67       193
weighted avg       0.70      0.71      0.71       193

[[102  24]
 [ 32  35]]
K-Nearest Neighbor Accuracy =  0.7098445595854922
K-Nearest Neighbor Sensitivity =  0.8095238095238095
K-Nearest Neighbor Specificity =  0.5223880597014925
K-Nearest Neighbor Precision =  0.7611940298507462
```