American International University – Bangladesh
Department of Computer Science & Engineering

Project Title: K-Nearest Neighbor (KNN) classification algorithm and its relevant tasks to a supervised Dataset.
Course: Introduction to Data Science

Submitted by-	**Submitted to-**
Name: MD. SHOHAIBUR RAHMAN **ID: 20-42424-1** **Section: C** **Summer 2022-2023** **B.Sc. CSE**	**Name: DR. ABDUS SALAM**

Dataset Description:

The Heart Attack Analysis and Prediction Dataset, available at the is a comprehensive collection of data designed for exploring and predicting the occurrence of heart attacks. This dataset contains valuable information about various factors that could contribute to heart attacks.
Source: Kaggle
URL: *https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset*

Attributes:

Attributes in the dataset include:

1. age,
2. sex,
3. chest pain type,
4. resting blood pressure,
5. cholesterol levels,
6. fasting blood sugar,
7. electrocardiographic results,
8. maximum heart rate achieved,
9. exercise-induced angina,
10. ST depression induced by exercise,
11. slope of the peak exercise ST segment,
12. number of major vessels colored by fluoroscopy,
13. thalassemia type and
14. the target variable indicating whether a person had a heart attack or not.

- **Importing the Dataset:** The dataset is located in a file called heart.csv in the current working directory. To begin data pre-processing using R, the first step is to import the dataset. Once imported, the heart.csv file is transformed into an R data frame and stored in a variable named "mydata". After printing the dataset, it looks like this-

R 4.3.1 · E:/DS_final_project/

```
> set.seed(321)
> mydata<-read.csv("E:/DS_final_project/heart.csv",header = TRUE)
>
>
> print(mydata)
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
1	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
2	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
3	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
4	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
5	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
6	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
7	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
8	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
9	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
10	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
11	54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
12	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
13	49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
14	64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
15	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1
16	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
17	58	0	2	120	340	0	1	172	0	0.0	2	0	2	1
18	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1
19	43	1	0	150	247	0	1	171	0	1.5	2	0	2	1
20	69	0	3	140	239	0	1	151	0	1.8	2	2	2	1
21	59	1	0	135	234	0	1	161	0	0.5	1	0	3	1
22	44	1	2	130	233	0	1	179	1	0.4	2	0	2	1
23	42	1	0	140	226	0	1	178	0	0.0	2	0	2	1
24	61	1	2	150	243	1	1	137	1	1.0	1	0	2	1
25	40	1	3	140	199	0	1	178	1	1.4	2	0	3	1
26	71	0	1	160	302	0	1	162	0	0.4	2	2	2	1

- **Finding Missing Values:**

```
> missing_values <- colSums(is.na(mydata))
> print(missing_values)
     age      sex       cp   trtbps     chol      fbs  restecg thalachh     exng  oldpeak      slp      caa    thall   output
       0        0        0        0        0        0        0        0        0        0        0        0        0        0
>
```

No missing values were found across the dataset.

- **Data structure:**

```
> str(mydata)
'data.frame':   303 obs. of  14 variables:
 $ age     : int  63 37 41 56 57 57 56 44 52 57 ...
 $ sex     : int  1 1 0 1 0 1 0 1 1 1 ...
 $ cp      : int  3 2 1 1 0 0 1 1 2 2 ...
 $ trtbps  : int  145 130 130 120 120 140 140 120 172 150 ...
 $ chol    : int  233 250 204 236 354 192 294 263 199 168 ...
 $ fbs     : int  1 0 0 0 0 0 0 0 1 0 ...
 $ restecg : int  0 1 0 1 1 1 0 1 1 1 ...
 $ thalachh: int  150 187 172 178 163 148 153 173 162 174 ...
 $ exng    : int  0 0 0 0 1 0 0 0 0 0 ...
 $ oldpeak : num  2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
 $ slp     : int  0 0 2 2 2 1 1 2 2 2 ...
 $ caa     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ thall   : int  1 2 2 2 2 1 2 3 3 2 ...
 $ output  : int  1 1 1 1 1 1 1 1 1 1 ...
>
```

The dataset consists of all integer / numeric values. So, no conversion was needed since KNN uses only numeric value.

- **Data Summary:**

```
> summary(mydata)
      age              sex               cp             trtbps           chol      
 Min.   :29.00   Min.   :0.0000   Min.   :0.000   Min.   : 94.0   Min.   :126.0  
 1st Qu.:47.50   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:120.0   1st Qu.:211.0  
 Median :55.00   Median :1.0000   Median :1.000   Median :130.0   Median :240.0  
 Mean   :54.37   Mean   :0.6832   Mean   :0.967   Mean   :131.6   Mean   :246.3  
 3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:140.0   3rd Qu.:274.5  
 Max.   :77.00   Max.   :1.0000   Max.   :3.000   Max.   :200.0   Max.   :564.0  
      fbs             restecg          thalachh         exng           oldpeak      
 Min.   :0.0000   Min.   :0.0000   Min.   : 71.0   Min.   :0.0000   Min.   :0.00  
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:133.5   1st Qu.:0.0000   1st Qu.:0.00  
 Median :0.0000   Median :1.0000   Median :153.0   Median :0.0000   Median :0.80  
 Mean   :0.1485   Mean   :0.5281   Mean   :149.6   Mean   :0.3267   Mean   :1.04  
 3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:166.0   3rd Qu.:1.0000   3rd Qu.:1.60  
 Max.   :1.0000   Max.   :2.0000   Max.   :202.0   Max.   :1.0000   Max.   :6.20  
      slp              caa             thall           output      
 Min.   :0.000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000  
 1st Qu.:1.000   1st Qu.:0.0000   1st Qu.:2.000   1st Qu.:0.0000  
 Median :1.000   Median :0.0000   Median :2.000   Median :1.0000  
 Mean   :1.399   Mean   :0.7294   Mean   :2.314   Mean   :0.5446  
 3rd Qu.:2.000   3rd Qu.:1.0000   3rd Qu.:3.000   3rd Qu.:1.0000  
 Max.   :2.000   Max.   :4.0000   Max.   :3.000   Max.   :1.0000  
> 
```

The summary() function was used to get an overview of key statistics and characteristics of the dataset, such as mean, median, minimum, maximum, and quartiles, enabling a quick understanding of the central tendencies and distribution of the data.

- **Scalling:** without target data

```
> scale(mydata[,1:13])
             age        sex         cp      trtbps        chol        fbs     restecg    thalachh       exng     oldpeak        slp        caa      thall
 [1,]  0.95062402  0.6798805  1.96986425  0.76269408 -0.255910365  2.3904835 -1.0041707  0.01541728 -0.69548  1.08554229 -2.2708221 -0.7132490 -2.1453238
 [2,] -1.91214969  0.6798805  1.00092128 -0.09258463  0.072080252 -0.4169448  0.8974776  1.63077374 -0.69548  2.11906724 -2.2708221 -0.7132490 -0.5120748
 [3,] -1.47172297 -1.4659924  0.03197832 -0.09258463 -0.815423771 -0.4169448 -1.0041707  0.97589950 -0.69548  0.31039858  0.9747397 -0.7132490 -0.5120748
 [4,]  0.17987725  0.6798805  0.03197832 -0.66277043 -0.198029668 -0.4169448  0.8974776  1.23784920 -0.69548 -0.20636389  0.9747397 -0.7132490 -0.5120748
 [5,]  0.28998393 -1.4659924 -0.93696465 -0.66277043  2.078611086 -0.4169448  0.8974776  0.58297496  1.43311 -0.37861805  0.9747397 -0.7132490 -0.5120748
 [6,]  0.28998393  0.6798805 -0.93696465  0.47760118 -1.046946559 -0.4169448  0.8974776 -0.07189928 -0.69548 -0.55087221 -0.6480412 -0.7132490 -2.1453238
 [7,]  0.17987725 -1.4659924  0.03197832  0.47760118  0.920997143 -0.4169448 -1.0041707  0.14639213 -0.69548  0.22427150 -0.6480412 -0.7132490 -0.5120748
 [8,] -1.14140292  0.6798805  0.03197832 -0.66277043  0.322896606 -0.4169448  0.8974776  1.01955778 -0.69548 -0.89538052  0.9747397 -0.7132490  1.1211742
 [9,] -0.26054947  0.6798805  1.00092128  2.30219574 -0.911891599  2.3904835  0.8974776  0.53931667 -0.69548 -0.46474513  0.9747397 -0.7132490  1.1211742
[10,]  0.28998393  0.6798805  1.00092128  1.04778698 -1.509992136 -0.4169448  0.8974776  1.06321607 -0.69548  0.48265274  0.9747397 -0.7132490 -0.5120748
[11,] -0.04033611  0.6798805 -0.93696465  0.47760118 -0.140148971 -0.4169448  0.8974776  0.45200011 -0.69548  0.13814442  0.9747397 -0.7132490 -0.5120748
[12,] -0.70097620 -1.4659924  1.00092128 -0.09258463  0.554419395 -0.4169448  0.8974776 -0.46482383 -0.69548 -0.72312637  0.9747397 -0.7132490 -0.5120748
[13,] -0.59086952  0.6798805  0.03197832 -0.09258463  0.380777303 -0.4169448  0.8974776  0.93224122 -0.69548 -0.37861805  0.9747397 -0.7132490 -0.5120748
[14,]  1.06073070  0.6798805  1.96986425 -1.23295623 -0.680368811 -0.4169448 -1.0041707 -0.24653242  1.43311  0.65490690 -0.6480412 -0.7132490 -0.5120748
[15,]  0.40009061 -1.4659924  1.96986425  1.04778698  0.708767921  2.3904835 -1.0041707  0.53931667 -0.69548 -0.03410973  0.9747397 -0.7132490 -0.5120748
[16,] -0.48076284 -1.4659924  1.00092128 -0.66277043 -0.526020285 -0.4169448  0.8974776  0.36468354 -0.69548  0.48265274 -0.6480412 -0.7132490 -0.5120748
[17,]  0.40009061 -1.4659924  1.00092128 -0.66277043  1.808501166 -0.4169448  0.8974776  0.97589950 -0.69548 -0.89538052  0.9747397 -0.7132490 -0.5120748
[18,]  1.28094407 -1.4659924  1.96986425  1.04778698 -0.390965325 -0.4169448  0.8974776 -1.55628090 -0.69548  1.34392353 -2.2708221 -0.7132490 -0.5120748
[19,] -1.25150961  0.6798805 -0.93696465  1.04778698  0.014199555 -0.4169448  0.8974776  0.93224122 -0.69548  0.39652566  0.9747397 -0.7132490 -0.5120748
[20,]  1.61126411 -1.4659924  1.96986425  0.47760118 -0.140148971 -0.4169448  0.8974776  0.05907556 -0.69548  0.65490690  0.9747397  1.2425378 -0.5120748
[21,]  0.51019730  0.6798805 -0.93696465  0.19250828 -0.236616799 -0.4169448  0.8974776  0.49565839 -0.69548 -0.46474513 -0.6480412 -0.7132490  1.1211742
[22,] -1.14140292  0.6798805  1.00092128 -0.09258463 -0.255910365 -0.4169448  0.8974776  1.28150748  1.43311 -0.55087221  0.9747397 -0.7132490 -0.5120748
[23,] -1.36161629  0.6798805 -0.93696465  0.47760118 -0.390965325 -0.4169448  0.8974776  1.23784920 -0.69548 -0.89538052  0.9747397 -0.7132490 -0.5120748
[24,]  0.73041066  0.6798805  1.00092128  1.04778698 -0.062974708  2.3904835  0.8974776 -0.55214039  1.43311 -0.03410973 -0.6480412 -0.7132490 -0.5120748
[25,] -1.58182965  0.6798805  1.96986425  0.47760118 -0.911891599 -0.4169448  0.8974776  1.23784920  1.43311  0.31039858  0.9747397 -0.7132490  1.1211742
```

The scale() function was applied to standardize the numerical features of the dataset without including the target column (column 14 “Output”), ensuring that all feature values have a mean of zero and a standard deviation of one.

- **Round and Correlation Matrix:**

```
> cor_matrix <- cor(mydata[,1:13])
> rounded_cor_matrix <- round(cor_matrix, digits = 3)
> print(rounded_cor_matrix)
            age    sex     cp trtbps   chol    fbs restecg thalachh   exng oldpeak    slp    caa  thall
age       1.000 -0.098 -0.069  0.279  0.214  0.121  -0.116   -0.399  0.097   0.210 -0.169  0.276  0.068
sex      -0.098  1.000 -0.049 -0.057 -0.198  0.045  -0.058   -0.044  0.142   0.096 -0.031  0.118  0.210
cp       -0.069 -0.049  1.000  0.048 -0.077  0.094   0.044    0.296 -0.394  -0.149  0.120 -0.181 -0.162
trtbps    0.279 -0.057  0.048  1.000  0.123  0.178  -0.114   -0.047  0.068   0.193 -0.121  0.101  0.062
chol      0.214 -0.198 -0.077  0.123  1.000  0.013  -0.151   -0.010  0.067   0.054 -0.004  0.071  0.099
fbs       0.121  0.045  0.094  0.178  0.013  1.000  -0.084   -0.009  0.026   0.006 -0.060  0.138 -0.032
restecg  -0.116 -0.058  0.044 -0.114 -0.151 -0.084   1.000    0.044 -0.071  -0.059  0.093 -0.072 -0.012
thalachh -0.399 -0.044  0.296 -0.047 -0.010 -0.009   0.044    1.000 -0.379  -0.344  0.387 -0.213 -0.096
exng      0.097  0.142 -0.394  0.068  0.067  0.026  -0.071   -0.379  1.000   0.288 -0.258  0.116  0.207
oldpeak   0.210  0.096 -0.149  0.193  0.054  0.006  -0.059   -0.344  0.288   1.000 -0.578  0.223  0.210
slp      -0.169 -0.031  0.120 -0.121 -0.004 -0.060   0.093    0.387 -0.258  -0.578  1.000 -0.080 -0.105
caa       0.276  0.118 -0.181  0.101  0.071  0.138  -0.072   -0.213  0.116   0.223 -0.080  1.000  0.152
thall     0.068  0.210 -0.162  0.062  0.099 -0.032  -0.012   -0.096  0.207   0.210 -0.105  0.152  1.000
>
```

Pearson correlation matrix was used to quantify the linear relationship between pairs of variables in a dataset. It helps to identify how strongly and in what direction variables are related, which is valuable for understanding potential patterns and dependencies in the data.

- **Install and load corrplot:**

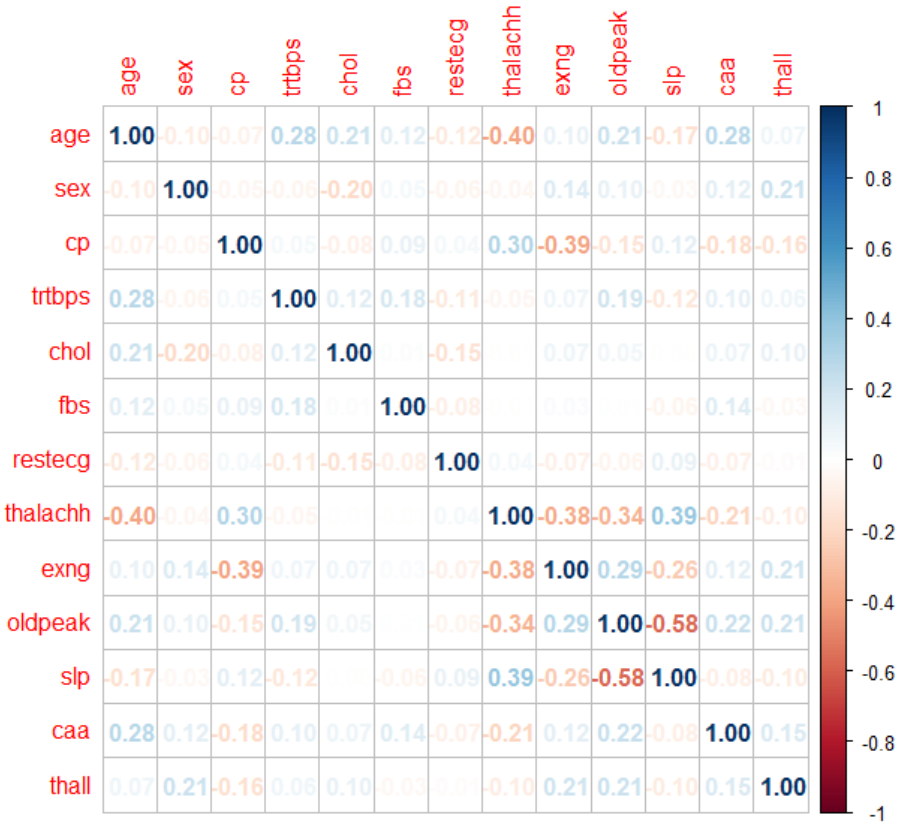
```
> install.packages("corrplot")
WARNING: Rtools is required to build R packages but is not currently installed.

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘C:/Users/DELL/AppData/Local/R/win-library/4.3’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.3/corrplot_0.92.zip'
Content type 'application/zip' length 3844862 bytes (3.7 MB)
downloaded 3.7 MB

package ‘corrplot’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\DELL\AppData\Local\Temp\RtmpOgpa3C\downloaded_packages
> library(corrplot)
corrplot 0.92 loaded
>
```

corrplot library is used to create visual representations of correlation matrices, such as heatmaps or clustered correlation plots.

- **Code:**

```
library(corrplot)
cor_matrix <- cor(mydata[,1:13], method = "pearson")
print(cor_matrix)
corrplot(cor_matrix, method = "number")
```

- **Visualize correlation:**



Pearson correlation method was used in matrix between the first 13 features in the dataset. It displays the correlation values, and creates a visualization using corrplot() with the correlation coefficients shown as numbers in a grid.

- **Define feature and target variable:**

```
> features <- mydata[, 1:13]
> target <- mydata[, 14]
>
```

Dataset was separated into features (independent variables) and the target variable (dependent variable).

- **Training and testing**

```
> set.seed(321)
> train_indices <- sample(seq_len(nrow(features)), size = 0.7 * nrow(features))
> train_data <- features[train_indices, ]
> train_target <- target[train_indices]
> test_data <- features[-train_indices, ]
> test_target <- target[-train_indices]
>
```

Random seed set.seed(321) was used to randomly split data and set 70% of data to train and 30% of data set to test.

Created the train_data (features for training) and train_target (target values for training) datasets.

- **Install and load Caret library**

```
#install.packages("caret")
library(caret)
```

library(caret) and library(class) is required to accomplish KNN algorithm.

- **KNN model matrix:**

```
k <- 10  # Number of neighbors
knn_model <- knn(train_data, test_data, train_target, k)
knn_model
```

'K' is set to 10 which indicates nearest neighbour. knn_model is created by applying the kNN algorithm to the 'train_data' and 'test_data' using the 'train_target' for training and then predicting the classes for the 'test_data'.

knn_model holds the predicted classes (groups) for the test data based on the given value of k.

- **KNN model output:**

```
> knn_model
 [1] 0 0 0 1 1 0 1 0 0 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1
Levels: 0 1
>
```

The levels 0 and 1 indicate the possible classes or outcomes that the algorithm can predict. In this case, 0 and 1 are the class labels. The Levels line specifies the possible class labels in dataset 0 (no heart disease) and 1 (heart disease).

- **KNN accuracy code:**

```
sum<- 0
accuracy <- sum(knn_model == test_target) / length(test_target)
print(paste("Accuracy:", accuracy))
```

Summed up the number of correct predictions (where knn_model matches test_target) and dividing it by the total number of test instances which represents the ratio of correct predictions. It also indicates how well model performs on test data.

- **KNN accuracy determine:**

```
> print(paste("Accuracy:", accuracy))
[1] "Accuracy: 0.67032967032967"
>
```

Accuracy was found “Accuracy: 0.67032967032967”. So about 67% of the predictions made by the model match the actual outcomes of dataset.

- **Confusion matrix code:**

```
confusion_matrix <- confusion_matrix / rowSums(confusion_matrix)
#install.packages("gplots")
library(gplots)

# Create a heatmap with normalized values
heatmap.2(confusion_matrix,
          trace = "none", col = colorRampPalette(c("white", "red"))(100),
          main = "Confusion Matrix", xlab = "Predicted", ylab = "Actual")
```

Confusion matrix values are divided by the sum of each corresponding row's values. The confusion_matrix summarizes the comparison between the

actual test_target values and the predicted knn_model values. It shows how many instances were correctly predicted and how many were predicted inaccurately.

- **Visualize confusion matrix:**

```
> print(confusion_matrix)
      Predicted
Actual         0         1
     0 0.5882353 0.4117647
     1 0.2807018 0.7192982
>
```

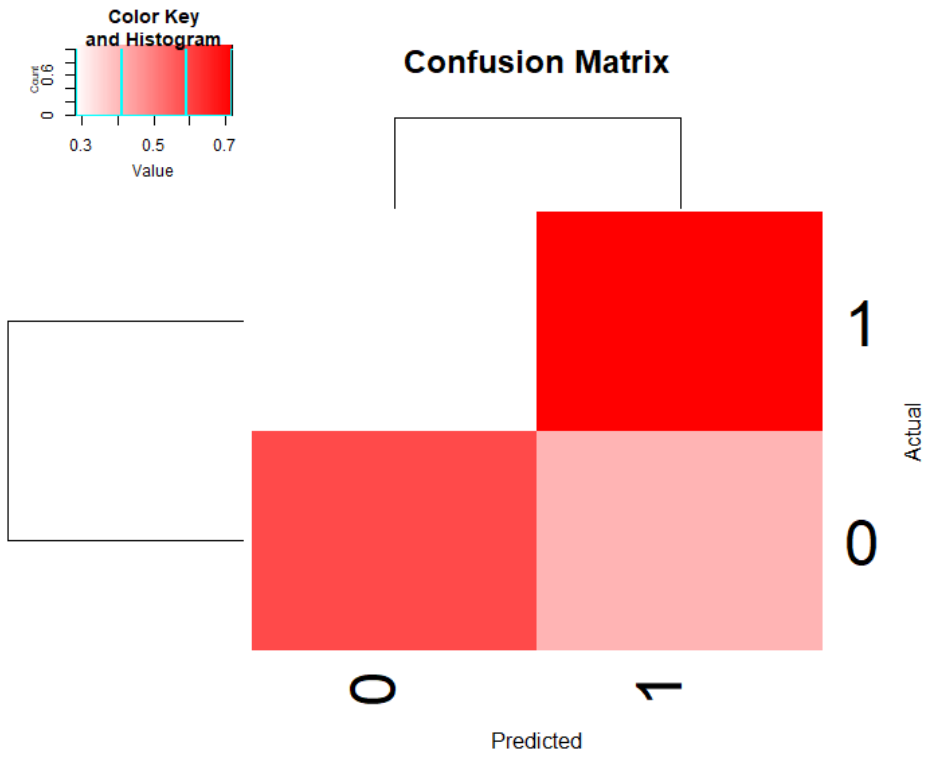
ft_project.R | confusion_matrix | .Rhistory

Filter

	Actual	Predicted	Freq
1	0	0	0.5882353
2	1	0	0.2807018
3	0	1	0.4117647
4	1	1	0.7192982

Here, frequency of 0.5882353 in the confusion matrix represents that 58% of the instances from a particular class were correctly classified by the model and so on for others as well. In a confusion matrix, each row represents the actual or true class labels of the instances, while each column represents the predicted class labels by the model.

- **Heatmap:**

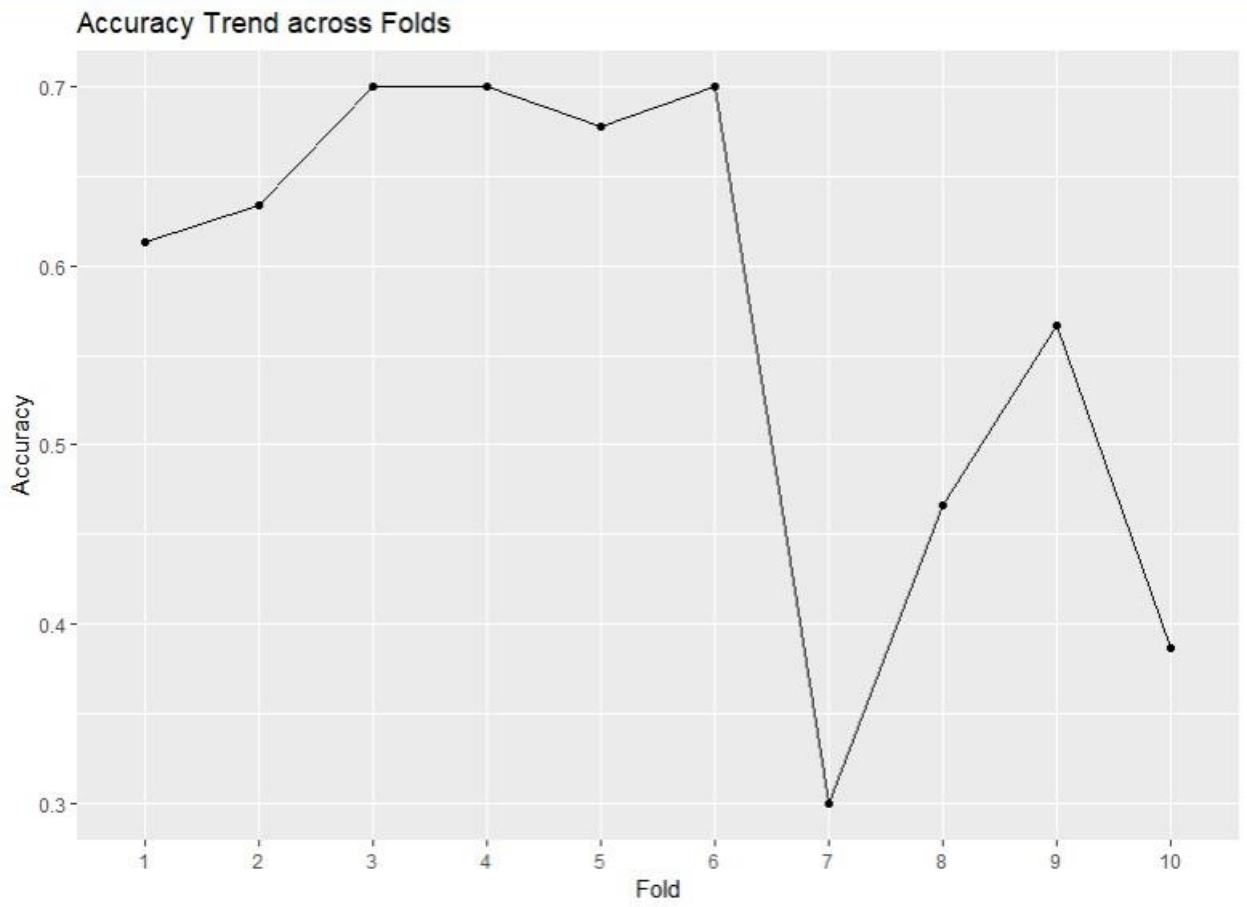


- **10-fold cross validation code :**

```
# Perform 10-fold cross-validation
num_folds <- 10
fold_indices <- cut(seq_along(target), breaks = num_folds, labels = FALSE)
accuracy_scores <- numeric(num_folds)
sum<- 0
for (fold in 1:num_folds) {
       test_indices <- fold_indices == fold
         train_data <- features[!test_indices, ]
         train_target <- target[!test_indices]
         test_data <- features[test_indices, ]
         test_target <- target[test_indices]
        knn_model <- knn(train_data, test_data, train_target, k)
         accuracy_scores[fold] <- sum(knn_model == test_target) / length(test_target)
}
 # Create a line plot with connected points
accuracy_data <- data.frame(Fold = factor(1:num_folds), Accuracy = accuracy_scores)
ggplot(accuracy_data, aes(x = Fold, y = Accuracy, group = 1)) +
       geom_line() +
       geom_point() +
       labs(x = "Fold", y = "Accuracy") +
       ggtitle("Accuracy Trend across Folds")
```

10-fold cross-validation were done by splitting the data into subsets, trains and tests a kNN model for each fold, calculates accuracy scores, and then creates a line plot showing how accuracy changes across the folds.

- **10-fold accuracy plot:**



As 10 folds were selected for cross validation , the plot represents variation in model performance where 1 to 6 model performs correct accuracy with high value (over 60%) and 7^{th} model with lowest (30%).

- **Calculate precision and recall:**
  Code :

```
TP <- confusion_matrix[2, 2]
FP <- confusion_matrix[1, 2]
FN <- confusion_matrix[2, 1]

# Calculate Precision and Recall
precision <- TP / (TP + FP)
recall <- TP / (TP + FN)

# Print Precision and Recall
print(paste("Precision:", precision))
print(paste("Recall:", recall))
```

  Result:

```
> print(paste("Precision:", precision))
[1] "Precision: 0.745454545454545"
> print(paste("Recall:", recall))
[1] "Recall: 0.719298245614035"
```

  Precision and recall serve to assess how well a classification model performs on data. Precision emphasizes accurate positive predictions, vital when mistakes are expensive. Recall prioritizes the model's capability to identify all positives, especially when missing them is undesirable.

- **Conclusion:**
  In conclusion, the analysis of the heart dataset provided valuable insights into predicting heart disease. By utilizing methods like k-nearest neighbors, cross-validation, and performance metrics like accuracy, precision, and recall we gained a deeper understanding of the dataset's patterns and predictive capabilities. These findings can contribute to better decision-making in healthcare and aid in identifying potential risk factors for heart disease.
  Thank you.