**Final-Term Project**

**Introduction to Data Science**

**Submitted by: Shahriar Soudeep**

**ID: 20-43823-2**

**Section: C**

**Submitted to: DR. ABDUS SALAM**

* **Dataset Description:**

This “wine-quality red.csv” dataset provides information about various attributes of red wines and their corresponding quality ratings. It is designed to analysis and model the factors that influence the quality of red wines.

The dataset includes the following column:

1. **Fixed Acidity:** This is a measure of the non-volatile acids present in the wine.
2. **Volatile Acidity:** This refers to the number of volatile acids in the wine, which can contribute to an unpleasant vinegar-like taste.
3. **Citric Acid:** This is the concentration of citric acid in the wine, which can influence the wine's freshness and flavor.
4. **Residual Sugar:** The amount of sugar remaining in the wine after fermentation. It can affect the wine's sweetness.
5. **Chlorides:** The concentration of salts in the wine, which can impact taste and mouthfeel.
6. **Free Sulfur Dioxide:** The presence of sulfur dioxide, which is used as a preservative and can affect the wine's stability.
7. **Total Sulfur Dioxide:** The sum of both free and bound sulfur dioxide.
8. **Density:** The density of the wine, which is related to its alcohol content.
9. **pH:** The pH level of the wine, which can affect its acidity and overall taste.
10. **Sulphates:** The concentration of sulfates, which can contribute to wine stability and flavor.
11. **Alcohol:** The alcohol content of the wine.
12. **Quality:** This is the target variable and represents the overall quality rating of the wine. It's usually a numeric values range from 1 to 10.

* **Importing Dataset:**

**Code:**

* library(readr)

data <- read\_csv("F:/Data-Science/Project/Final-term/winequality-red.csv")

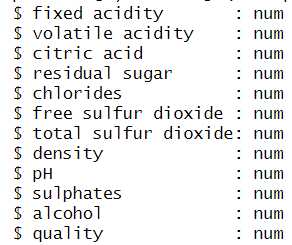
**Description:**

* At first loaded the **‘readr’** library. This library is commonly used for reading and handling structured data, especially CSV files. The **‘read\_csv’** function reads the CSV file and converts it into a structured data frame. The data frame is stored in the variable named data for further use.
* **Check Data Types:**

**Code:**

* str(data)

**Output:**

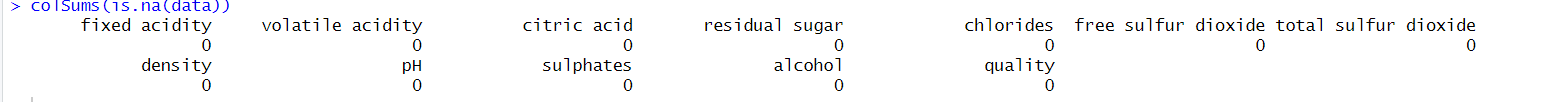
****

**Description:**

* Using the **‘str(data)’** function to find the data types and structure of the imported dataset. This function helps to identify whether the data types are appropriate for each column or should I do any further data type conversions. In my dataset all attributes contain the numerical values.
* **Check Null Values**

**Code:**

* colSums(is.na(data))

**Output:**

**Description:**

* By using the **‘colSums(is.na(data))’** function, I am able to identify the missing(Null) values within each column of the dataset. The **is.na(data)** expression creates a Boolean matrix where TRUE represents a missing value, and FALSE represents a non-missing value. The **‘colSums’** function then calculates the sum of TRUE values in each column and counts the number of missing values in each column. By performing this check, I am able to identify the columns with missing values and can easily understand is there any technique can apply to remove this null value or not. In my selecting dataset there is no missing (Null) values.
* **Apply Normalization :**

**Code:**

* quality <- data$quality

data <- data[, -which(names(data) == "quality")]

min\_max\_normalize <- function(x) {

(x - min(x)) / (max(x) - min(x))

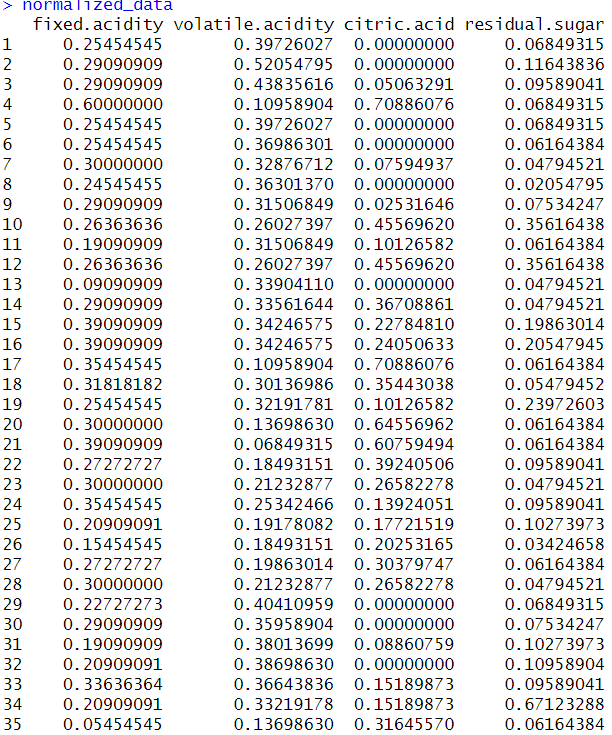
}

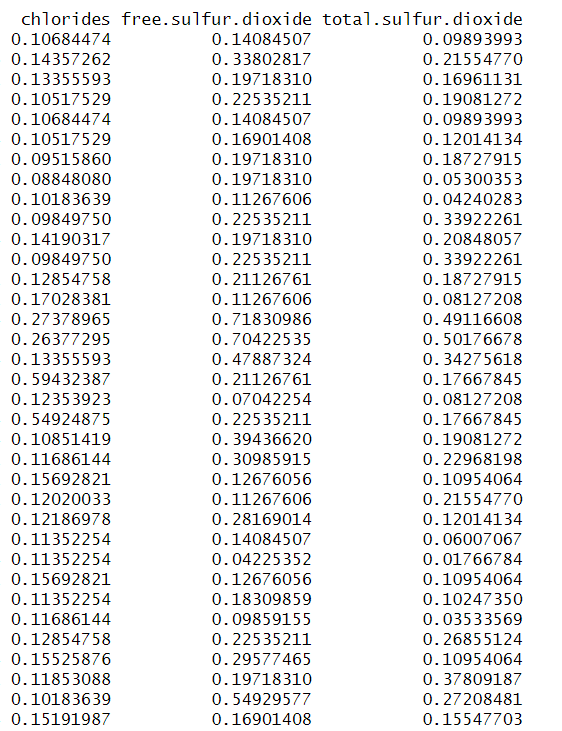
normalized\_data <- as.data.frame(lapply(data, min\_max\_normalize))

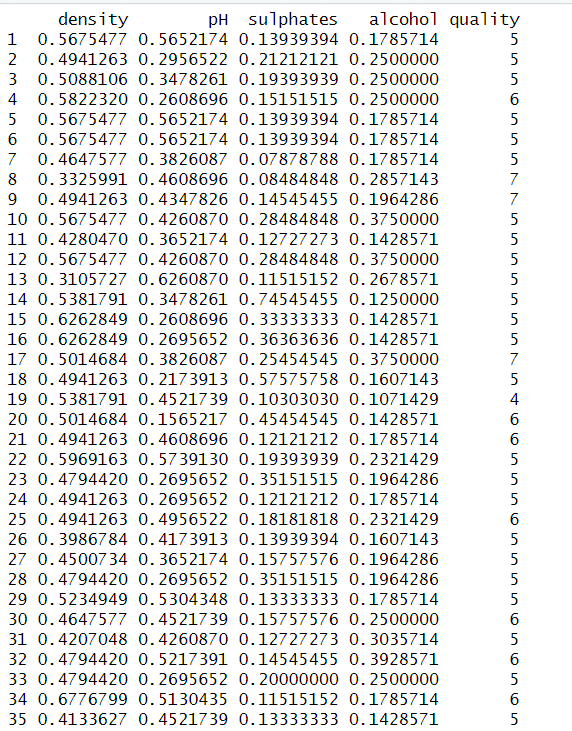
normalized\_data <- cbind(normalized\_data, quality)

normalized\_data

write\_csv(normalized\_data,"F:/Data-Science/Project/Final-term/normalized\_red-wine.csv")

**Output: **

****

****

**Description:**

* Previously we saw, in our dataset there is no categorical value and no Null values. So, we can now apply normalization. This normalization process can helps that all attributes are on a similar range. Range of normalization is **0 to 1**.

At first we extract our class attribute which is ‘quality’ by using ‘**data$quality’** code. Than, I remove the quality column from the dataset because we do the normalization except this quality column. I defined a function named **min\_max\_normalize** that takes a vector as input and applies min-max normalization to it. Min-max normalization scales each attribute to a range of **(0, 1)** by subtracting the minimum value and dividing by the range. I have applied the **min\_max\_normalize** function to each column of the dataset using the **lapply** function. This generates a normalized version of my selecting dataset. After this process, I have added the **quality** attribute to the normalized dataset by using the **cbind** function. After adding the class attributes I have to save the normalized dataset by using **write\_csv** function so that, original dataset remains unchanged.

* **Find The Correlation:**

**Code:**

* dataset<- read.csv("F:/Data-Science/Project/Final-term/normalized\_red-wine.csv")

correlation\_matrix<- cor(dataset[c("fixed.acidity","volatile.acidity","citric.acid","residual.sugar","chlorides ","free.sulfur.dioxide","total.sulfur.dioxide","density","pH","sulphates","alcohol","qua lity")], dataset$quality)

View(correlation\_matrix)

* correlation\_with\_quality <- cor(dataset)

View(correlation\_with\_quality)

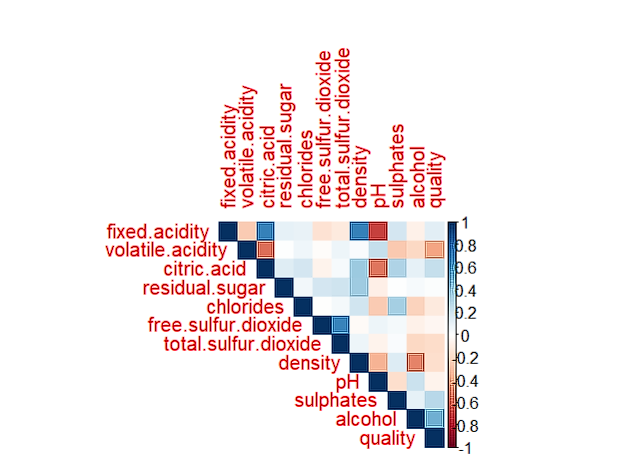
* library(corrplot)

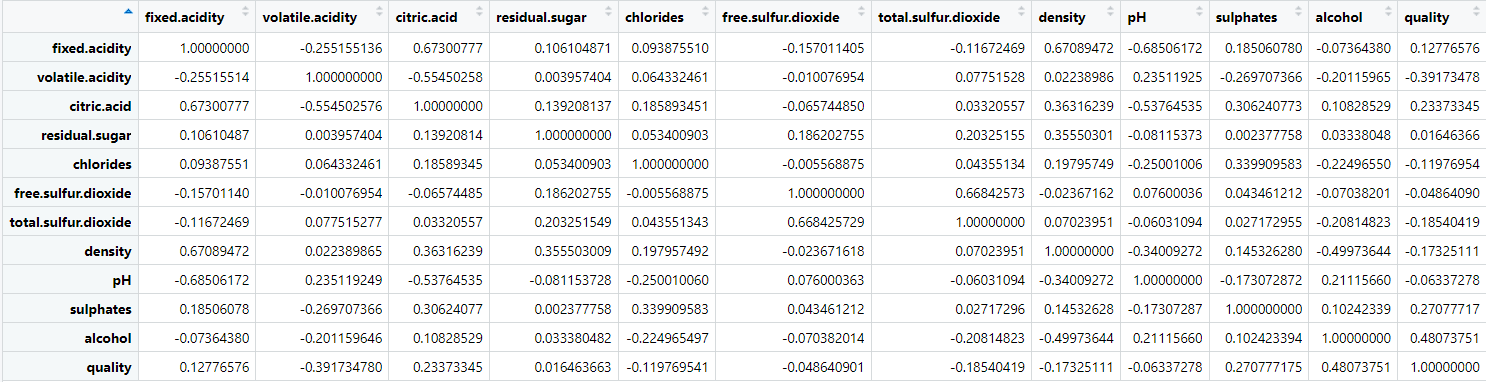
correlation\_matrix <- cor(dataset)

corrplot(correlation\_matrix, method = "color", type = "upper")

**A screenshot of a computer

Description automatically generatedOutput:**

****

****

**Description:**

* At first, we import the normalized dataset by using the **read.csv** function and store it into **dataset** variable. Here, is the process is to find the correlation against our targeted attribute. Using the **cor** function I am going to calculate the correlation value of every column against my class attribute(quality). **View** function isused to show the calculation. Also, here can use **cor(dataset)** to show the correlation matrix of every attribute. Also, we can plot or visually represet a plot to show the correlation matrix by using **corrplot** library.
* **Check for 0 correlation:**

**Code:**

* attributes\_with\_zero\_correlation<- names(correlation\_with\_quality[correlation\_with\_quality == 0])

if (length(attributes\_with\_zero\_correlation) > 0) {

print("Attributes with zero correlation to quality:", "\n")

print(attributes\_with\_zero\_correlation)

} else {

print("No attributes with zero correlation to quality.")

}

**Output:**

****

**Description:**

* The code first calculate the correlation against quality attributes with every attribute. **Names** function used to identify that is there any 0 correlation or not. Then, apply the conditional statement is grater than 0 than show 0 correlation column found otherwise there is no attribute which has 0 correlation with our class attribute. In this “wine-quality red.csv” dataset there is no attributes which has 0 correlation with class attributes.
* **Apply KNN:**

1. **Training and Testing(Build KNN Model):**

**Code:**

* library(class)

library(caret)

features <- dataset[, !colnames(dataset) %in% c("quality")]

target <- dataset$quality

set.seed(123)

train\_indices <- sample(seq\_len(nrow(features)), size = 0.70 \* nrow(features))

train\_data <- features[train\_indices, ]

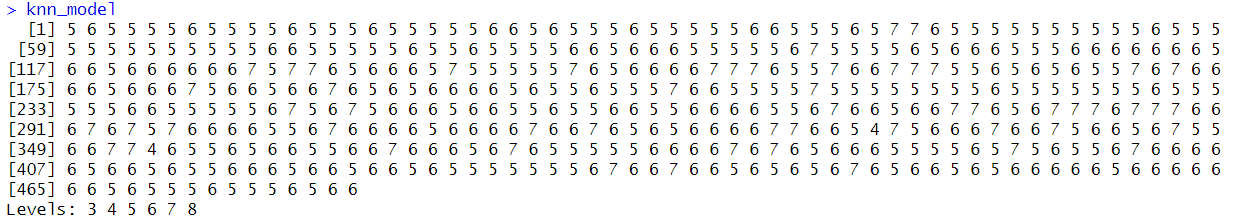
train\_target <- target[train\_indices]

test\_data <- features[-train\_indices, ]

test\_target <- target[-train\_indices]

knn\_model <- knn(train\_data, test\_data, train\_target, k = 10)

knn\_model

**Output:**

**Description:**

* Here, I am building a KNN classification model to predict the quality ratings of the red wine dataset based on the attributes. So, at first import **class** for the implementation of KNN and **caret** library for the partition of the dataset. Here, I am used **set.seed(123)** to get the random result first time and then in every time it gives the same results. **Seq\_len** is used to determine the length and **sample** is used to take the sample size of the training and testing data. Here, I am choosing **70%** of training and **30%** testing data. **knn** function is used to build the KNN model. It uses the training and testing data.

**Accuracy of(Training and Testing):**

**Code:**

* accuracy <- sum(knn\_model == test\_target) / length(test\_target)

cat("The accuracy of trainig and testing is: ", accuracy)

**Output:**

****

**Description:**

* Here, **knn\_model** is the predicted quality ratings and **test\_target** is the actual quality ratings. To finding the accuracy here, the sum of the correct prediction is **divided** by the total number of test data. **cat** is used to show the calculated accuracy value.

**Confusion matrix, Recall and Precision:**

**Code:**

* confusion\_matrix <- table(Actual = test\_target, Predicted = knn\_model)

confusion\_matrix

conf\_matrix<- confusion\_matrix

recall <- numeric(nrow(conf\_matrix))

precision <- numeric(nrow(conf\_matrix))

for (i in 1:nrow(conf\_matrix)) {

true\_positives <- conf\_matrix[i, i]

false\_negatives <- sum(conf\_matrix[i, ]) - true\_positives

false\_positives <- sum(conf\_matrix[, i]) - true\_positives

precision[i] <- true\_positives / (true\_positives + false\_positives)

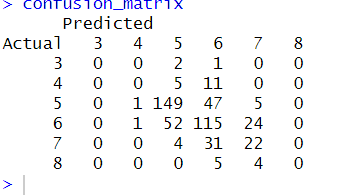
recall[i] <- true\_positives / (true\_positives + false\_negatives)

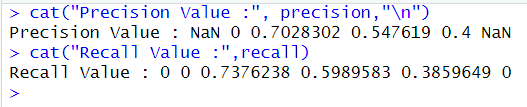
}

cat("Precision value:", precision)

cat("Recall value:",recall,"\n")

**Output:**

****

****

**Description:**

* **Table** function is used to generate the confusion matrix. In confusion matrix here, stays the number of true positive, true negative, false positive, and false negative predictions. View function is used to display the confusion matrix.
* For recall and precision, at first initialize same number of rows in the confusion matrix. Then, iterate a loop for each row of the confusion matrix and calculates true positives, false positives. Then, calculate the recall by using **true\_positives / (true\_positives + false\_negatives)** formula and for precission use **true\_positives / (true\_positives + false\_positives)** formula. **Cat** function is used for display the values.

1. **10-Fold Cross Validation (Build KNN Model)**

**Code:**

* features <- dataset[, !colnames(dataset) %in% c("quality")]

target <- dataset$quality

k <- 10

set.seed(123)

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, ]

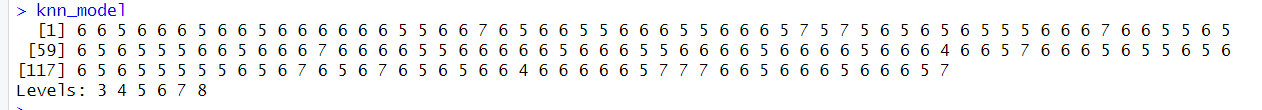
test\_target <- target[test\_indices]

knn\_model <- knn(train\_data, test\_data, train\_target, k)

accuracy\_scores[fold] <- sum(knn\_model == test\_target) / length(test\_target)

}

knn\_model

**Output:**

**Description:**

* At first separate the target attribute quality. Here, k=10(nearest neighbour) and set.seed is used to get the random result and every time it gives the same result. Here, folding number is 10. Cut function is used to generate the fold indices. Fold is used for testing and remaining data is used for training. After that, I am using a loop for iteration in every fold. Within each iteration the training and testing dataset is separated based on current fold. Than, display the KNN model.

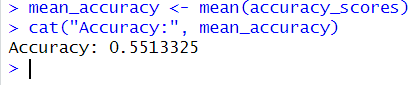
**Accuracy of(10-fold cross validation):**

**Code:**

* mean\_accuracy <- mean(accuracy\_scores)

cat("Accuracy:", mean\_accuracy)

**Output:**

****

**Description:**

* Here, I am going to calculate the mean accuracy of KNN model for all folds. Computing the average accuracy scores from each fold. Than, display the accuracy value.