DA5030.P1.Verma

Shivam Verma

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(class)
```

```
## Warning: package 'class' was built under R version 3.6.2
```

```
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(gmodels)
```

##Problem 1

Question 1. Read and load the csv file.

```
g <- read.csv("glass.csv", header = FALSE)
# Renaming the headers
colnames(g) <- c("ID","RI","Na","Mg", "Al","Si", "K","Ca", "Ba", "Fe", "Type")
```

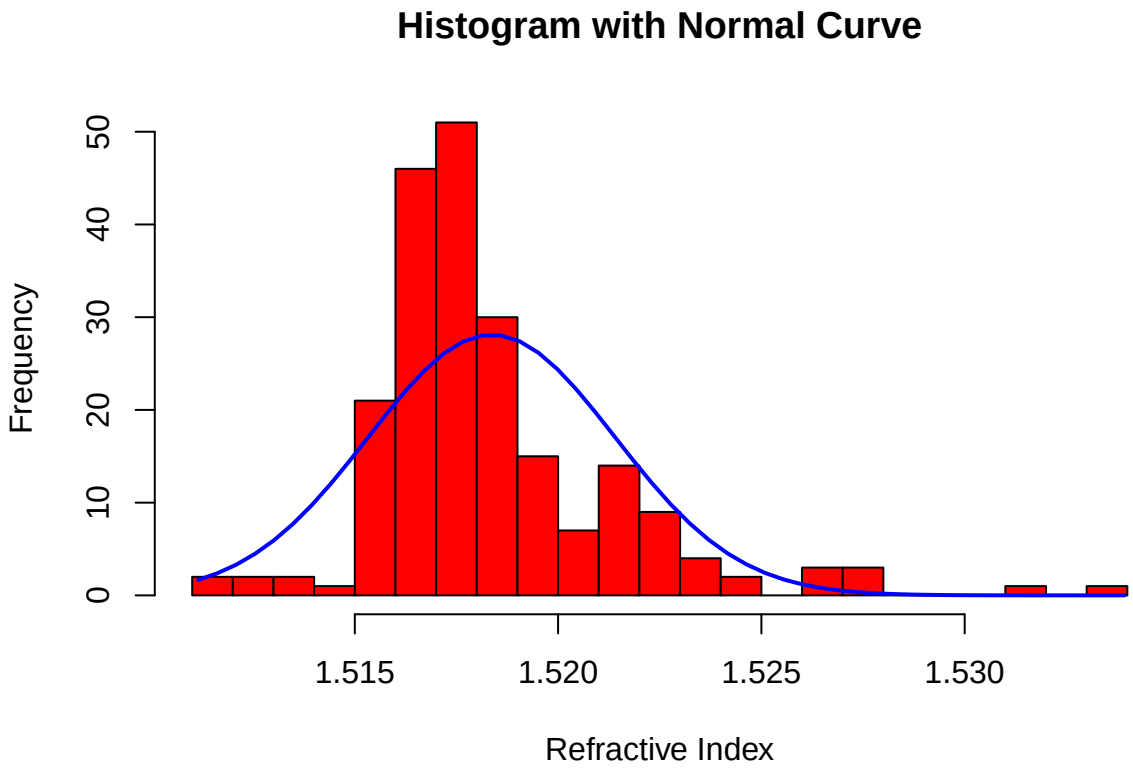
Question 2.

```
#Exploring the dataset
head(g)
```

```
##   ID      RI    Na   Mg   Al    Si    K   Ca Ba   Fe Type
## 1  1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00    1
## 2  2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00    1
## 3  3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00    1
## 4  4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00    1
## 5  5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00    1
## 6  6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26    1
```

Question 3. After Visually examining the histogram we can say that the data is not perfectly normally distributed.

```
# Histogram of refractive index with overlayed normal curve
x<- g$RI
h<-hist(x, breaks=20, col="red", xlab="Refractive Index", main="Histogram with Normal Curve")
xfit<-seq(min(x),max(x),length=50)
yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))
# Visualization of data if it fits normally
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
```



Question 4. KNN is a non-parametric algorithm as this algorithm does not make any assumptions of the distribution of the data set. Thus, there's no necessity that dataset have to be normally distributed to apply KNN.

```
# KNN is a non-parametric algorithm as this algorithm does not make any assumptions of the distribution
```

Question 5. Identified outliers for each columns using a z-score deviation approach, i.e., considered any values that are more than 2 standard deviations from the mean as outliers.

We can drop outliers or replace with the nearest "good" data or replace it with the mean or the median.

```r
# Created a dataset containing only numeric values to find outliers.
df <- g[,c(2:11)]
# Identifying the outliers.
for (i in 1:9) {
zscore <- abs(((df[,i] - mean(df[,i]))) / (sd(df[,i])))
outliers <- which(zscore > 2)
# Printing row numbers which include outliers in each column.
print(colnames(df[i]))
print(outliers)
}
```

```
## [1] "RI"
##  [1]  48  57 104 106 107 108 111 112 113 132 185 186
## [1] "Na"
## [1] 106 107 111 112 166 167 185 190 201
## [1] "Mg"
## integer(0)
## [1] "Al"
##  [1]  22 164 172 173 185 193 196 197 199 200 209 210
## [1] "Si"
##  [1] 104 107 108 110 164 172 173 181 185 189 190 202
## [1] "K"
## [1] 172 173 202
## [1] "Ca"
##  [1] 106 107 108 111 112 113 132 164 171 174 186 187
## [1] "Ba"
##  [1] 107 164 186 187 190 194 195 204 206 207 208 211 212 213 214
## [1] "Fe"
##  [1]   6  45  57  72 106 107 119 136 146 163 175 176
```

Question 6. Normalizing the numeric columns, except the last one (the glass type), using z-score standardization.

```r
# Normalizing df as it excludes ID.
for (i in 1:9) {

  df[,i] <- (((df[,i] - mean(df[,i]))) / (sd(df[,i])))

}
head(df)
```

```
##           RI         Na        Mg         Al          Si          K         Ca
## 1  0.8708258  0.2842867 1.2517037 -0.6908222 -1.12444556 -0.67013422 -0.1454254
## 2 -0.2487502  0.5904328 0.6346799 -0.1700615  0.10207972 -0.02615193 -0.7918771
## 3 -0.7196308  0.1495824 0.6000157  0.1904651  0.43776033 -0.16414813 -0.8270103
## 4 -0.2322859 -0.2422846 0.6970756 -0.3102663 -0.05284979  0.11184428 -0.5178378
## 5 -0.3113148 -0.1688095 0.6485456 -0.4104126  0.55395746  0.08117845 -0.6232375
## 6 -0.7920739 -0.7566101 0.6416128  0.3506992  0.41193874  0.21917466 -0.6232375
##           Ba         Fe Type
## 1 -0.3520514 -0.5850791    1
## 2 -0.3520514 -0.5850791    1
## 3 -0.3520514 -0.5850791    1
## 4 -0.3520514 -0.5850791    1
```

```
## 5 -0.3520514 -0.5850791    1
## 6 -0.3520514  2.0832652    1
```

Question 7. Created a partition of data into valid and train, including 20% of each of the glass type in both data using CreateDataPartition function from caret package on the bases of glass type.

```
index <- createDataPartition(y = df$Type, p = 0.2, list = FALSE)
# creating valid and train sets
knn_train <- df[-index,]
knn_valid <- df[index,]
head(knn_train)
```

```
##           RI         Na        Mg         Al           Si           K         Ca
## 1  0.8708258  0.2842867 1.2517037 -0.6908222 -1.12444556 -0.67013422 -0.1454254
## 2 -0.2487502  0.5904328 0.6346799 -0.1700615  0.10207972 -0.02615193 -0.7918771
## 3 -0.7196308  0.1495824 0.6000157  0.1904651  0.43776033 -0.16414813 -0.8270103
## 4 -0.2322859 -0.2422846 0.6970756 -0.3102663 -0.05284979  0.11184428 -0.5178378
## 7 -0.3080219 -0.1320720 0.6346799 -0.6107051  0.56686825  0.12717719 -0.5529710
## 8 -0.2652146 -0.3157597 0.6416128 -0.7909685  0.76053014  0.11184428 -0.5037845
##           Ba         Fe Type
## 1 -0.3520514 -0.5850791    1
## 2 -0.3520514 -0.5850791    1
## 3 -0.3520514 -0.5850791    1
## 4 -0.3520514 -0.5850791    1
## 7 -0.3520514 -0.5850791    1
## 8 -0.3520514 -0.5850791    1
```

```
head(knn_train )
```

```
##           RI         Na        Mg         Al           Si           K         Ca
## 1  0.8708258  0.2842867 1.2517037 -0.6908222 -1.12444556 -0.67013422 -0.1454254
## 2 -0.2487502  0.5904328 0.6346799 -0.1700615  0.10207972 -0.02615193 -0.7918771
## 3 -0.7196308  0.1495824 0.6000157  0.1904651  0.43776033 -0.16414813 -0.8270103
## 4 -0.2322859 -0.2422846 0.6970756 -0.3102663 -0.05284979  0.11184428 -0.5178378
## 7 -0.3080219 -0.1320720 0.6346799 -0.6107051  0.56686825  0.12717719 -0.5529710
## 8 -0.2652146 -0.3157597 0.6416128 -0.7909685  0.76053014  0.11184428 -0.5037845
##           Ba         Fe Type
## 1 -0.3520514 -0.5850791    1
## 2 -0.3520514 -0.5850791    1
## 3 -0.3520514 -0.5850791    1
## 4 -0.3520514 -0.5850791    1
## 7 -0.3520514 -0.5850791    1
## 8 -0.3520514 -0.5850791    1
```

Question 8. Created a KNN prediction model to predict unkown glass type of 2 new samples, it will take normalized data of both train and new data and predict the type of glass on based on nearest eucledian distance calculated using sqrt of sum of each element in row and the ordering it to ascending order and extracting nearest neighbours.

```
# creating new dataframe to normalize new variables for prediction.
k <- g[-1]
# adding the new variables to the dataset so that everything gets normalized at one go.
```

```
k[215,] <- c(1.51621, 12.53, 3.48, 1.39, 73.39, 0.60, 8.55, 0.00, 0.08, 0)
k[216,] <- c(1.5893, 12.71, 1.85, 1.82, 72.62, 0.52, 10.51 ,0.00, 0.05, 0)
# Normalizing using z-scores.
for (i in 1:9) {

  k[,i] <- (((k[,i] - mean(k[,i]))) / (sd(k[,i])))

}
# arranging normalized data to apply knn.
# b,c are unknown cases
a <- k[1:214, c(1:10)]
b <- k[215, c(1:9)]
c <- k[216, c(1:9)]

# Created mode function
mode <- function(x)
{
  uniqx <- unique(x)
  uniqx[which.max(tabulate(match(x,uniqx)))]
}

# Manually created KNN function.
get_type <- function(x){
  # empty array to store distance
  d <- numeric(214)
  for(i in 1:214)
  {
    # calculating euclidean distance
    d[i] <- sqrt(sum((a[i,(1:9)]- x[1, (1:9)])^2))
  }
# getting top 5 nearest neighbours as k = 5
neighbor <- k$Type[order(d)] [1:5]
(mode(neighbor))

}

# passing arguments in fucntion
unknown1_type <- get_type(b)
unknown2_type <- get_type(c)
# printing result
print(paste0("Type of first Unkown Element : " , unknown1_type ))
```

```
## [1] "Type of first Unkown Element : 1"
```

```
print(paste0("Type of second Unkown Element : " , unknown2_type))
```

```
## [1] "Type of second Unkown Element : 2"
```

Question 9. Applied the knn function from the class package

```
predUnk1 <- knn(train = a[,1:9], test = b[1,1:9], cl = a$Type, k=10)
predUnk2 <- knn(train = a[,1:9], test = c[1,1:9], cl = a$Type, k=10)

# printing result
print(paste0("Type of first Unkown Element : " , predUnk1))
```

```
## [1] "Type of first Unkown Element : 1"
```

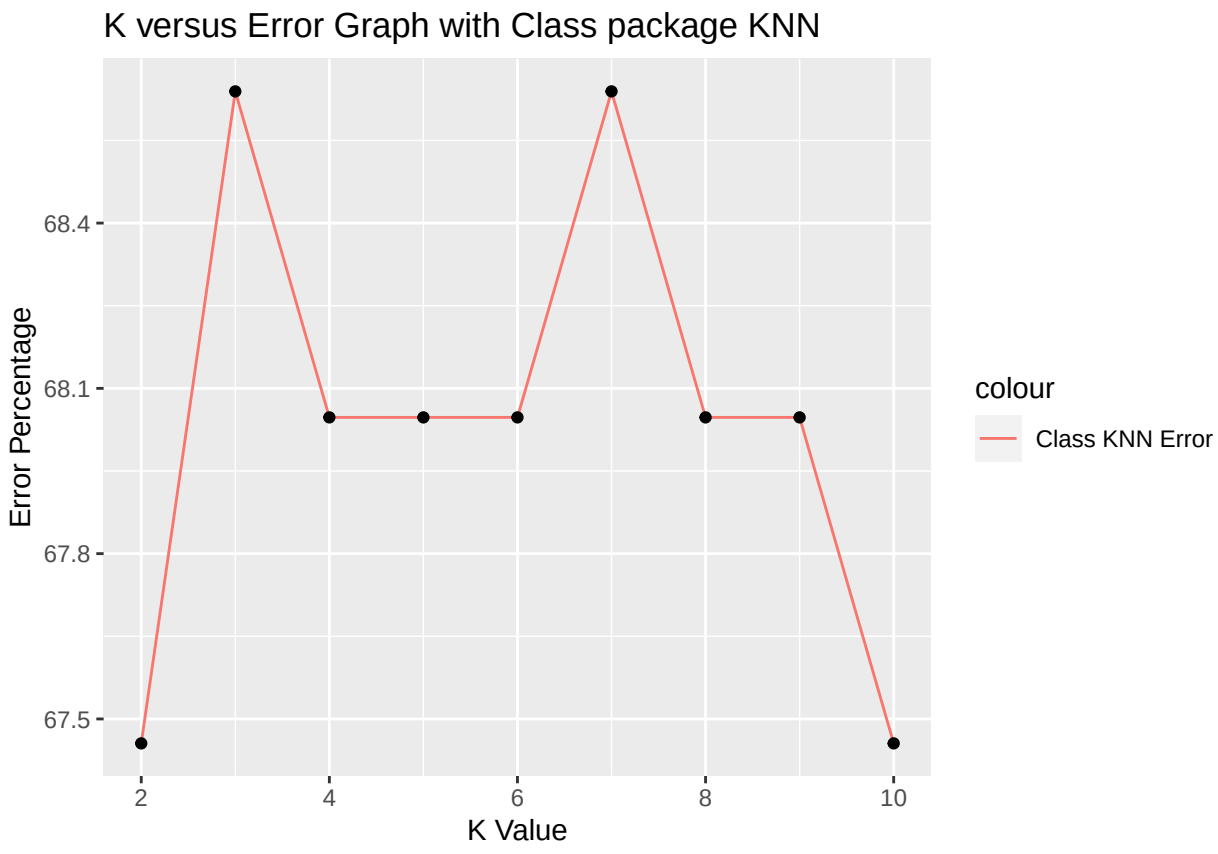
```
print(paste0("Type of Second Unkown Element : " , predUnk2))
```

```
## [1] "Type of Second Unkown Element : 2"
```

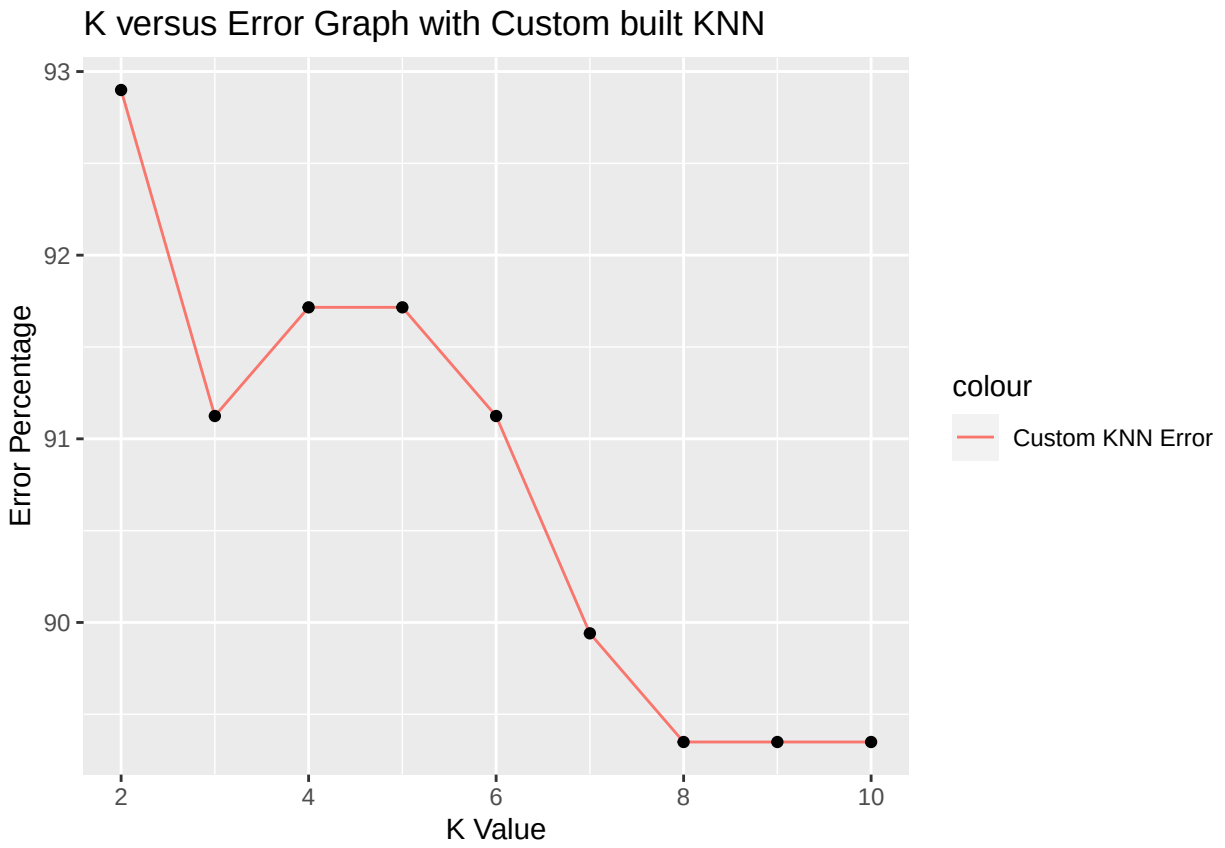
Question 10. Create a plot of k (x-axis) from 2 to 10 versus error rate (percentage of incorrect classifications) for both algorithms using ggplot.

```
set.seed(123)
# creating test and valid data
train <- knn_train[,1:9]
test <- knn_valid[,1:9]
# setting type of train data
train_labels <- knn_train$Type
# empty array to store values
trainPred <- rep(0, nrow(train))
# empty data frame to store errors
newCl <- data.frame(k= c(2:10) , perc_error = rep(NA, 9))

for(i in 2:10)
{
  # class package knn to predict type
  trainPred <- knn(train = train, test = test, train_labels, i)
  # calculating error percentage on base of results
  newCl[i-1,2] <- sum(train_labels != trainPred) / nrow(train) *100
}

# Mode function
mode <- function(x)
{
  uniqx <- unique(x)
  uniqx[which.max(tabulate(match(x,uniqx)))]
}

# Manually created KNN function.
get_type <- function(x,k){
  # empty array to store distance
  d <- numeric(214)
  for(i in 1:214)
  {
    # calculating euclidean distance
    d[i] <- sqrt(sum((df[i,(1:9)]- df[x, (1:9)])^2))
  }
# getting top 5 nearest neighbours as k = 5
neighbor <- df$Type[order(d)] [1:k]
```

```
(mode(neighbor))

}
# creating test and valid data
train <- knn_train[,1:9]
test <- knn_valid[,1:9]
train_labels1 <- knn_train$Type
# empty array to store values
trainPred1 <- rep(0, nrow(train))
# empty data frame
newCl1 <- data.frame(k= c(2:10) , perc_error = rep(NA, 9))
for (i in 2:10){
# class package knn to predict type
# getting prediction for each row in test dataset and storing value in new array
  for (j in 1:nrow(test)){
trainPred1[j] <- get_type(rownames(test)[j],i) }
  # calculating error percentage on base of results
newCl1[i-1,2] <- sum(train_labels1 != trainPred1) / nrow(train) * 100 }

# Plotting k vs error graph for Class package KNN
ggplot(newCl, aes(x=newCl[,1], y = newCl[,2])) + geom_line(aes(color="Class KNN Error")) + geom_point()
```




```
# Plotting k vs error graph for Custom built KNN
ggplot(newCl1, aes(x=newCl1[,1], y = newCl1[,2])) + geom_line(aes(color="Custom KNN Error")) + geom_poi
```



Question 11. Cross-table confusion matrix showing accuracy for class package knn with k = 5

```
# passing knn prediction with k = 5
Pred <- knn(train = knn_train[,1:9], test = knn_valid[,1:9], cl = knn_train$Type, k=5)
# printing confusion matrix
CrossTable(x= knn_valid$Type, y=Pred, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  45
##
##
##                | Pred
## knn_valid$Type |         1 |         2 |         3 |         5 |         6 |         7 | Row Total |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##              1 |        11 |         1 |         0 |         0 |         0 |         0 |        12 |
##                |     0.917 |     0.083 |     0.000 |     0.000 |     0.000 |     0.000 |     0.267 |
##                |     0.579 |     0.071 |     0.000 |     0.000 |     0.000 |     0.000 |           |
```

```
##                |     0.244 |     0.022 |     0.000 |     0.000 |     0.000 |     0.000 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##              2 |         6 |        11 |         0 |         1 |         0 |         0 |        18 |
##                |     0.333 |     0.611 |     0.000 |     0.056 |     0.000 |     0.000 |     0.400 |
##                |     0.316 |     0.786 |     0.000 |     0.333 |     0.000 |     0.000 |           |
##                |     0.133 |     0.244 |     0.000 |     0.022 |     0.000 |     0.000 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##              3 |         2 |         1 |         1 |         0 |         0 |         0 |         4 |
##                |     0.500 |     0.250 |     0.250 |     0.000 |     0.000 |     0.000 |     0.089 |
##                |     0.105 |     0.071 |     1.000 |     0.000 |     0.000 |     0.000 |           |
##                |     0.044 |     0.022 |     0.022 |     0.000 |     0.000 |     0.000 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##              5 |         0 |         1 |         0 |         2 |         0 |         0 |         3 |
##                |     0.000 |     0.333 |     0.000 |     0.667 |     0.000 |     0.000 |     0.067 |
##                |     0.000 |     0.071 |     0.000 |     0.667 |     0.000 |     0.000 |           |
##                |     0.000 |     0.022 |     0.000 |     0.044 |     0.000 |     0.000 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##              6 |         0 |         0 |         0 |         0 |         3 |         0 |         3 |
##                |     0.000 |     0.000 |     0.000 |     0.000 |     1.000 |     0.000 |     0.067 |
##                |     0.000 |     0.000 |     0.000 |     0.000 |     1.000 |     0.000 |           |
##                |     0.000 |     0.000 |     0.000 |     0.000 |     0.067 |     0.000 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##              7 |         0 |         0 |         0 |         0 |         0 |         5 |         5 |
##                |     0.000 |     0.000 |     0.000 |     0.000 |     0.000 |     1.000 |     0.111 |
##                |     0.000 |     0.000 |     0.000 |     0.000 |     0.000 |     1.000 |           |
##                |     0.000 |     0.000 |     0.000 |     0.000 |     0.000 |     0.111 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##   Column Total |        19 |        14 |         1 |         3 |         3 |         5 |        45 |
##                |     0.422 |     0.311 |     0.022 |     0.067 |     0.067 |     0.111 |           |
## ---------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
##
##
```

```
# printing confusion matrix from caret for accuracy
caret::confusionMatrix(factor(knn_valid$Type), factor(Pred))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  5  6  7
##          1 11  1  0  0  0  0
##          2  6 11  0  1  0  0
##          3  2  1  1  0  0  0
##          5  0  1  0  2  0  0
##          6  0  0  0  0  3  0
##          7  0  0  0  0  0  5
##
## Overall Statistics
##
##                Accuracy : 0.7333
##                  95% CI : (0.5806, 0.854)
##     No Information Rate : 0.4222
##     P-Value [Acc > NIR] : 2.329e-05
##
```

```
##                   Kappa : 0.6395
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
## Sensitivity            0.5789   0.7857  1.00000  0.66667  1.00000   1.0000
## Specificity            0.9615   0.7742  0.93182  0.97619  1.00000   1.0000
## Pos Pred Value         0.9167   0.6111  0.25000  0.66667  1.00000   1.0000
## Neg Pred Value         0.7576   0.8889  1.00000  0.97619  1.00000   1.0000
## Prevalence             0.4222   0.3111  0.02222  0.06667  0.06667   0.1111
## Detection Rate         0.2444   0.2444  0.02222  0.04444  0.06667   0.1111
## Detection Prevalence   0.2667   0.4000  0.08889  0.06667  0.06667   0.1111
## Balanced Accuracy      0.7702   0.7800  0.96591  0.82143  1.00000   1.0000
```

Question 12. Imputed missing values in 4th column in modified glass data using prediction with manually builded knn.reg with weighted average as linear regression. For using K = 4

```
# Reading the data set
g_missing <- read.csv("glass.data_missing_values.csv", header = FALSE)
# renaming the columns
colnames(g_missing) <- c("ID","RI","Na","Mg", "Al","Si", "K","Ca", "Ba", "Fe", "Type")
# locating the NA values
which(is.na(g_missing), arr.ind=TRUE)
```

```
##       row col
##  [1,]  20   4
##  [2,]  30   4
##  [3,]  95   4
##  [4,] 163   4
##  [5,] 169   4
##  [6,] 184   4
##  [7,] 194   4
##  [8,] 200   4
##  [9,] 208   4
```

```
target_data <- data.frame (Mg=g_missing[,4])
# removing ID and Mg for training data
train_data <- (g_missing[, c(-1,-4)])
train_data_N <- train_data
for (i in 1:ncol(train_data_N)){
  # skipping type to normalize
  if (i == 10){}
  else{ train_data_N[i] <- (train_data_N[i] - min(train_data_N[i])) / (max(train_data_N[i])- min(train_
}
# getting normalized rows to impute their values
new_data <- train_data_N[c(20,30,95,163,169,184,194,200,208),]
train_data_N <- train_data_N[-c(20,30,95,163,169,184,194,200,208),]
sumW <- 0
sumK <- 0
# custom weighted avg knn function to predict
knn.reg <- function(new_data, target_data, train_data, k)
```

```
{
  # empty array for eucledian distance
  euc_dist <- numeric(nrow(train_data))
  # calculating euclidean distance
  for(i in 1:nrow(train_data))
  {
    euc_dist[i] <- sqrt(sum((train_data[i,(1:9)]- new_data[1, (1:9)])^2))
  }
  # getting K nearest neighbours
  neighbor_knnreg <- target_data[order(euc_dist),] [1:k]

  # calculating weighted average
  for (j in 1:k)
  {
    if(j==1){
      sumW <- sumW + (neighbor_knnreg[j]*3)
      sumK <- 3 + sumK
    }
    else if (j==2){
      sumW <- sumW + (neighbor_knnreg[j]*2)
      sumK <- 2 + sumK
    }
    else{
      sumW <- sumW + (neighbor_knnreg[j]*1)
      sumK <- 1 + sumK
    }

  }
  Avg <- sumW / sumK
  Avg
}
# passing specific row as new_data, target_data, whole normalized data, with k = 4
for (i in 1: nrow(new_data)){
  Mg <- knn.reg(new_data[i,], target_data, train_data_N, 4)
  # printing results
  print(paste0("Value imputes for row : " , i, " is " , Mg))

}
```

```
## [1] "Value imputes for row : 1 is 3.36857142857143"
## [1] "Value imputes for row : 2 is 3.41"
## [1] "Value imputes for row : 3 is 3.71857142857143"
## [1] "Value imputes for row : 4 is 3.57857142857143"
## [1] "Value imputes for row : 5 is 2.87285714285714"
## [1] "Value imputes for row : 6 is 0.23"
## [1] "Value imputes for row : 7 is 1.43142857142857"
## [1] "Value imputes for row : 8 is 0.508571428571429"
## [1] "Value imputes for row : 9 is 0.248571428571429"
```

##Problem 2

Question 1.

```r
# reading dataset
h <- read.csv("kc_house_data.csv", stringsAsFactors = FALSE)
head(h)
```

```
##           id            date   price bedrooms bathrooms sqft_living sqft_lot
## 1 7129300520 20141013T000000  221900        3      1.00        1180     5650
## 2 6414100192 20141209T000000  538000        3      2.25        2570     7242
## 3 5631500400 20150225T000000  180000        2      1.00         770    10000
## 4 2487200875 20141209T000000  604000        4      3.00        1960     5000
## 5 1954400510 20150218T000000  510000        3      2.00        1680     8080
## 6 7237550310 20140512T000000 1225000        4      4.50        5420   101930
##   floors waterfront view condition grade sqft_above sqft_basement yr_built
## 1      1          0    0         3     7       1180             0     1955
## 2      2          0    0         3     7       2170           400     1951
## 3      1          0    0         3     6        770             0     1933
## 4      1          0    0         5     7       1050           910     1965
## 5      1          0    0         3     8       1680             0     1987
## 6      1          0    0         3    11       3890          1530     2001
##   yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
## 1            0   98178 47.5112 -122.257          1340       5650
## 2         1991   98125 47.7210 -122.319          1690       7639
## 3            0   98028 47.7379 -122.233          2720       8062
## 4            0   98136 47.5208 -122.393          1360       5000
## 5            0   98074 47.6168 -122.045          1800       7503
## 6            0   98053 47.6561 -122.005          4760     101930
```

Question 2. Creating train and target data

```r
# target_data extracting house price column
target_data <- data.frame(price = h[,3])
# train dataset with specific column only
train_data <- h[,4:15]
head(target_data)
```

```
##     price
## 1  221900
## 2  538000
## 3  180000
## 4  604000
## 5  510000
## 6 1225000
```

```r
head(train_data)
```

```
##   bedrooms bathrooms sqft_living sqft_lot floors waterfront view condition
## 1        3      1.00        1180     5650      1          0    0         3
## 2        3      2.25        2570     7242      2          0    0         3
## 3        2      1.00         770    10000      1          0    0         3
## 4        4      3.00        1960     5000      1          0    0         5
## 5        3      2.00        1680     8080      1          0    0         3
## 6        4      4.50        5420   101930      1          0    0         3
##   grade sqft_above sqft_basement yr_built
```

```
## 1     7       1180             0     1955
## 2     7       2170           400     1951
## 3     6        770             0     1933
## 4     7       1050           910     1965
## 5     8       1680             0     1987
## 6    11       3890          1530     2001
```

Question 3.

```
# Normalizing train data set
train_data_N <- train_data
for (i in 1:ncol(train_data_N)){
  # skipping over waterfront and view
  if (i == 6 | i == 7){}
  else{ train_data_N[i] <- (train_data_N[i] - min(train_data_N[i])) / (max(train_data_N[i])- min(train_
}
head(train_data_N)
```

```
##     bedrooms bathrooms sqft_living    sqft_lot floors waterfront view condition
## 1 0.09090909   0.12500  0.06716981 0.003107511    0.0          0    0       0.5
## 2 0.09090909   0.28125  0.17207547 0.004071869    0.4          0    0       0.5
## 3 0.06060606   0.12500  0.03622642 0.005742535    0.0          0    0       0.5
## 4 0.12121212   0.37500  0.12603774 0.002713772    0.0          0    0       1.0
## 5 0.09090909   0.25000  0.10490566 0.004579490    0.0          0    0       0.5
## 6 0.12121212   0.56250  0.38716981 0.061429370    0.0          0    0       0.5
##       grade sqft_above sqft_basement  yr_built
## 1 0.5000000 0.09758772    0.00000000 0.4782609
## 2 0.5000000 0.20614035    0.08298755 0.4434783
## 3 0.4166667 0.05263158    0.00000000 0.2869565
## 4 0.5000000 0.08333333    0.18879668 0.5652174
## 5 0.5833333 0.15241228    0.00000000 0.7565217
## 6 0.8333333 0.39473684    0.31742739 0.8782609
```

Question 4. Built own knn.reg function with nearest neighbour weighted average function to predict the price of new house, along with weighted average of 3 for nearest neighbour, 2 for 2nd nearest neighbour and 1 for all others. This function take 4 arguments new_data, target_data, train_data and K all of the data set passed is normalized data set apart from target_data as we have to predict original value of that.

```
sumW <- 0
sumK <- 0
# Custom weighted avg knn function
knn.reg <- function(new_data, target_data, train_data, k)
{
  # empty array for eucledian distance
  euc_dist <- numeric(nrow(train_data))
  # calculating eucledian distance
  for(i in 1:nrow(train_data))
  {
    euc_dist[i] <- sqrt(sum((train_data[i,(1:12)]- new_data[1, (1:12)])^2))
  }
  # getting k nearest neighbors
  neighbor_knnreg <- target_data[order(euc_dist),] [1:k]
```

```
  # calculating weighted avg
  for (j in 1:k)
  {
    if(j==1){
      sumW <- sumW + (neighbor_knnreg[j]*3)
      sumK <- 3 + sumK
    }
    else if (j==2){
      sumW <- sumW + (neighbor_knnreg[j]*2)
      sumK <- 2 + sumK
    }
    else{
      sumW <- sumW + (neighbor_knnreg[j]*1)
      sumK <- 1 + sumK
    }

  }
  priceAvg <- sumW / sumK
  priceAvg
}
```

Question 5. Passed new case to predict the estimate price of new house using our own knn.reg function using k = 4

```
# data to predict
new_data <- data.frame(bedrooms = 4,  bathrooms = 3,  sqft_living = 4852, sqft_lot = 10244, floors = 3,

new_data_N <- new_data
# normalizing the new data
for(i in 1:ncol(new_data_N))
{
  # skipping waterfront and view
  if (i == 6 | i == 7){}
  else{new_data_N[i] <- (new_data_N[i] - min(train_data[i])) / (max(train_data[i])- min(train_data[i]))
}
# passing specific row as new_data, target_data, whole normalized data, with k = 4
getPrice <- knn.reg(new_data_N, target_data, train_data_N, 4)
# Printing results
paste0("Price of new house as predicted by KNN.reg is : " ,round(getPrice))
```

```
## [1] "Price of new house as predicted by KNN.reg is : 676007"
```