DA5030.P1.Luo.Sui.Zhao

Xinxin Luo, Xin (Sue) Sui, Xiwen Zhao

02/03/2020

Problem 1.1 (0 pts)

Download the data set Glass Identification Database along with its explanation. Note that the data file does not contain header names; you may wish to add those. The description of each column can be found in the data set explanation. This assignment must be completed within an R Markdown Notebook.

```
# Downloaded the data and load into df:
df <- read.csv("glass.data", header = F, stringsAsFactors = F)
glass_names <- read.csv("glass.names", header = F, stringsAsFactors = F)

# Rename the column names:
colnames(df) <- c("ID", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "GlassType")
```

Problem 1.2 (0 pts)

Explore the data set as you see fit and that allows you to get a sense of the data and get comfortable with it.

```
# Explore the data set
head(df)
```

```
##   ID      RI    Na   Mg   Al    Si    K   Ca Ba   Fe GlassType
## 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
```

```
colnames(df)
```

```
##  [1] "ID"        "RI"        "Na"        "Mg"        "Al"
##  [6] "Si"        "K"         "Ca"        "Ba"        "Fe"
## [11] "GlassType"
```

```
str(df)
```

```
## 'data.frame':    214 obs. of  11 variables:
##  $ ID       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ RI       : num  1.52 1.52 1.52 1.52 1.52 ...
##  $ Na       : num  13.6 13.9 13.5 13.2 13.3 ...
##  $ Mg       : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
##  $ Al       : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
##  $ Si       : num  71.8 72.7 73 72.6 73.1 ...
##  $ K        : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
##  $ Ca       : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
##  $ Ba       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Fe       : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
##  $ GlassType: int  1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(df)
```

```
##        ID               RI              Na              Mg
##  Min.   :  1.00   Min.   :1.511   Min.   :10.73   Min.   :0.000
##  1st Qu.: 54.25   1st Qu.:1.517   1st Qu.:12.91   1st Qu.:2.115
##  Median :107.50   Median :1.518   Median :13.30   Median :3.480
##  Mean   :107.50   Mean   :1.518   Mean   :13.41   Mean   :2.685
##  3rd Qu.:160.75   3rd Qu.:1.519   3rd Qu.:13.82   3rd Qu.:3.600
##  Max.   :214.00   Max.   :1.534   Max.   :17.38   Max.   :4.490
##        Al              Si              K                Ca
##  Min.   :0.290   Min.   :69.81   Min.   :0.0000   Min.   : 5.430
##  1st Qu.:1.190   1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240
##  Median :1.360   Median :72.79   Median :0.5550   Median : 8.600
##  Mean   :1.445   Mean   :72.65   Mean   :0.4971   Mean   : 8.957
##  3rd Qu.:1.630   3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172
##  Max.   :3.500   Max.   :75.41   Max.   :6.2100   Max.   :16.190
##        Ba              Fe              GlassType
##  Min.   :0.000   Min.   :0.00000   Min.   :1.00
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:1.00
##  Median :0.000   Median :0.00000   Median :2.00
##  Mean   :0.175   Mean   :0.05701   Mean   :2.78
##  3rd Qu.:0.000   3rd Qu.:0.10000   3rd Qu.:3.00
##  Max.   :3.150   Max.   :0.51000   Max.   :7.00
```

Problem 1.3 (5 pts)

Create a histogram of column 2 (refractive index) and overlay a normal curve; visually determine whether the data is normally distributed. You may use the code from this tutorial.

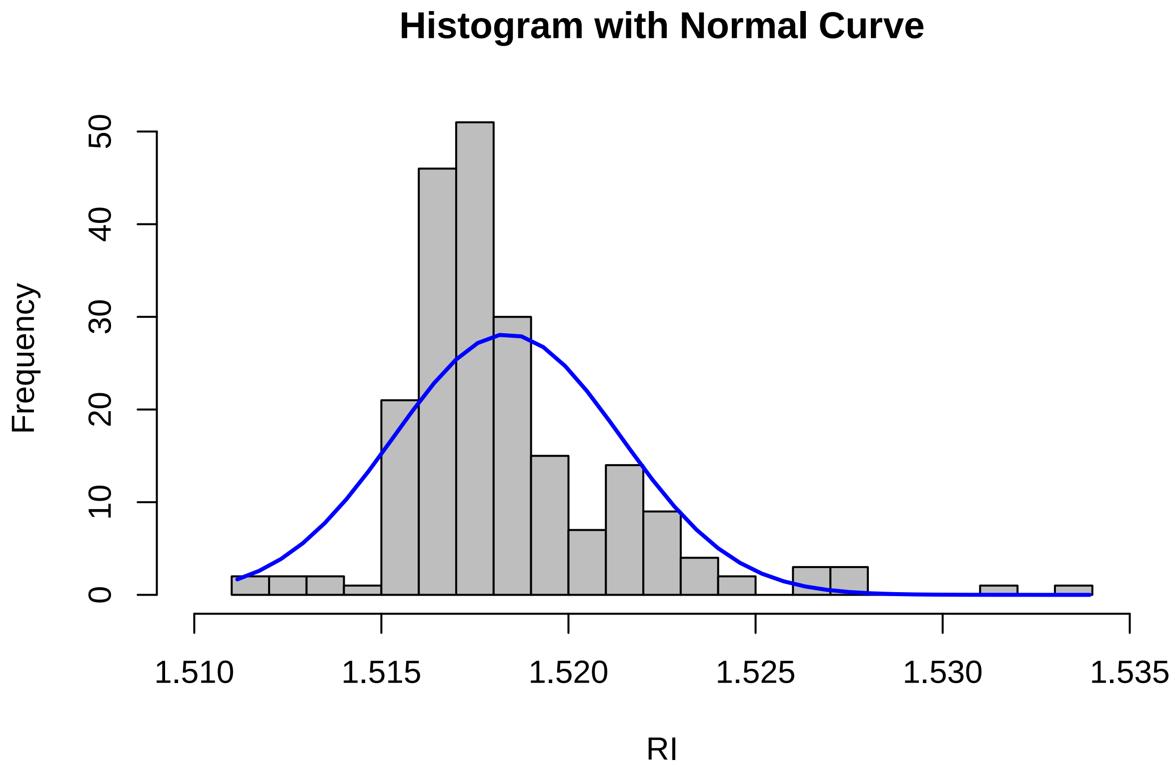
```r
x <- df$RI
#determin the min and max for RI
min(df$RI)
```

```
## [1] 1.51115
```

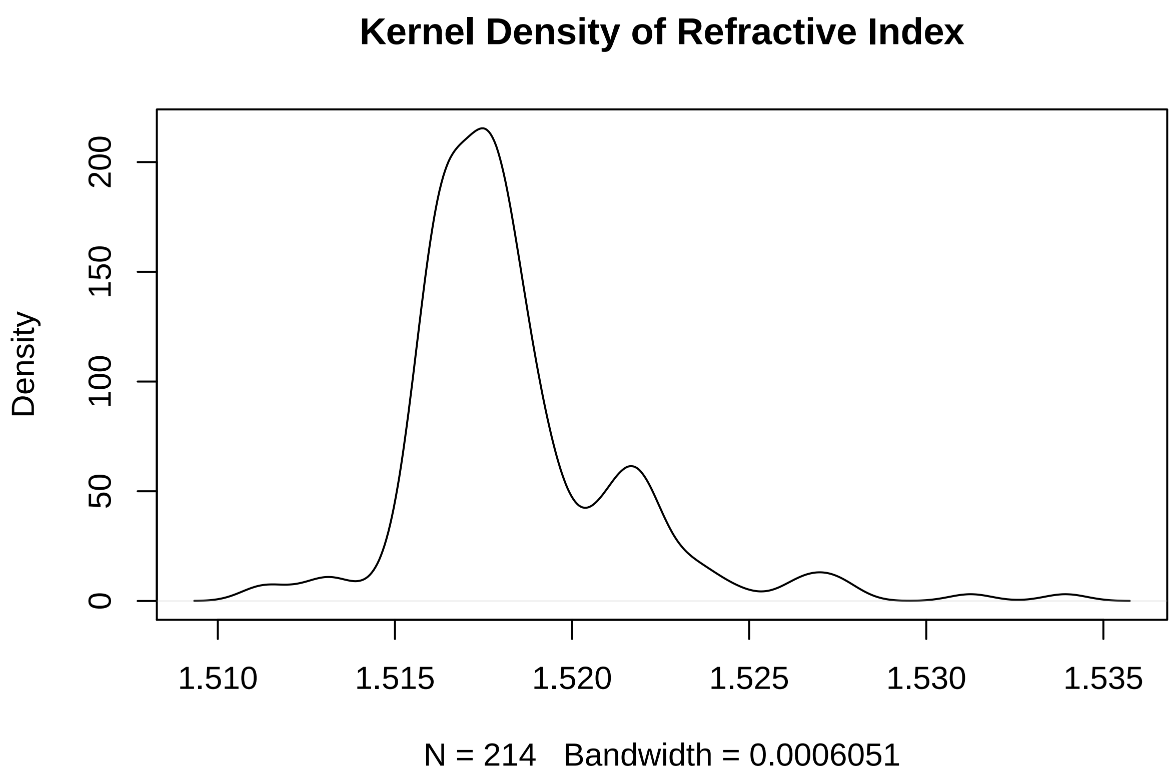
```r
max(df$RI)
```

```
## [1] 1.53393
```

```r
#include all min and max for RI.
h <- hist(df$RI, breaks = 20,xlim = c(1.51,1.535),col = "grey", xlab = "RI",
     main = "Histogram with Normal Curve")
xfit <- seq(min(x), max(x), length = 40)
yfit <- dnorm(xfit, mean = mean(x), sd = sd(x))
yfit <- yfit * diff(h$mids[1:2]) * length(x)
lines(xfit, yfit, col = "blue", lwd = 2)
```




```r
# Kernel Density Plot to double check since it has a better visualization
d <- density(df$RI) # returns the density data
plot(d, main="Kernel Density of Refractive Index") # plots the results
```



```r
summary(df$RI)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.511   1.517   1.518   1.518   1.519   1.534
```

```r
# According to the histogram, Refractive Index from glass data is NOT normally distributed.
# we can observe based on the normal curve that the data has positive skewness.
```

Problem 1.4 (5 pts)

Does the k-NN algorithm require normally distributed data or is it a non-parametric method? Comment on your findings. Answer this in a code block as a comment only.

```r
# k-NN algorithm is a non-parametric method. That is no parameters are learned about the data.
# k-NN classifiers may be considered lazy, but it allows the learner to find natural patterns.
# Therefore the data do not have to be normally distributed.

# Resources used: Machine Learning with R textbook, chapter3 by Brett Lantz.
```

Problem 1.5 (5 pts)

Identify any outliers for the columns using a z-score deviation approach, i.e., consider any values that are more than 2 standard deviations from the mean as outliers. Which are your outliers for each column? What would you do? Do not remove them the outliers.

```r
# Step 1. z-score normalization function
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------ tidy
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
z_normalize <- function(x) {
  return ((x - mean(x)) / sd(x)) }

# Step 2. Normalize the data
df_n <- data.frame("ID"=df$ID) %>% cbind(lapply(df[,2:(ncol(df)-1)], z_normalize)) #loop over the colum

# Step 3. Use a for loop to find out outliers for each column (2 SD from the mean)
for (i in 2:(ncol(df)-1)){
  outliers <- which(abs(df_n[,i]) > 2)
  if (length(outliers) == 0) next
  cat(colnames(df)[i],"\thas outliers", df[outliers,i],"\n")
  }
```

```
## RI   has outliers 1.52667 1.51215 1.52725 1.52475 1.53125 1.53393 1.52664 1.52739 1.52777 1.52614 1.5
## Na   has outliers 11.45 10.73 11.23 11.02 11.56 11.03 17.38 15.79 15.15
## Al   has outliers 0.29 3.5 3.04 3.02 0.34 2.79 2.68 2.54 2.66 2.51 2.74 2.88
## Si   has outliers 70.57 69.81 70.16 74.45 69.89 70.48 70.7 74.55 75.41 70.26 70.43 75.18
## K    has outliers 6.21 6.21 2.7
```

```
## Ca   has outliers 13.24 13.3 16.19 14.68 14.96 14.4 13.44 5.87 12.24 12.5 5.43 5.79
## Ba   has outliers 3.15 2.2 1.19 1.63 1.68 1.59 1.57 1.71 1.55 1.38 2.88 1.59 1.64 1.57 1.67
## Fe   has outliers 0.26 0.3 0.31 0.32 0.34 0.28 0.29 0.28 0.35 0.37 0.51 0.28
```

```
# Step 4. I would use the mean to replace the outliers in each column
```

Problem 1.6 (10 pts)

After removing the ID column (column 1), normalize the numeric columns, except the last one, using z-score standardization. The last column is the glass type and so it is excluded.

```
# Remove column 1 (ID)
df <- df[,-1]

# Z-score standardization function to normalize whole data set
normalize <- function(x) {
  return ((x - mean(x)) / sd(x)) }

# Normalize df without GlassType
df_n <- as.data.frame(lapply(df[,1:(ncol(df)-1)], normalize)) %>%
  cbind("GlassType" = df$GlassType)
```

Problem 1.7 (10 pts)

The data set is sorted, so creating a validation data set requires random selection of elements. Create a stratified sample where you randomly select 20% of each of the cases for each glass type to be part of the validation data set. The remaining cases will form the training data set.

```
# Set the seed, so results are reproducible
set.seed(123)

# Check how many glass types are there:
unique(df_n$GlassType)
```

```
## [1] 1 2 3 5 6 7
```

```
# Using normalized df split each GlassType into 80% to training data set
# and 20% to validation data set
type1 <- filter(df_n, GlassType==1)
split1 <- sample.int(n = nrow(type1), size = floor(.8 * nrow(type1)), replace = F)
t1 <- type1[split1,]
v1 <- type1[-split1,]

type2 <- filter(df_n, GlassType==2)
split2 <- sample.int(n = nrow(type2), size = floor(.8 * nrow(type2)), replace = F)
t2 <- type2[split2,]
v2 <- type2[-split2,]

type3 <- filter(df_n, GlassType==3)
split3 <- sample.int(n = nrow(type3), size = floor(.8 * nrow(type3)), replace = F)
t3 <- type3[split3,]
v3 <- type3[-split3,]

type5 <- filter(df_n, GlassType==5)
split5 <- sample.int(n = nrow(type5), size = floor(.8 * nrow(type5)), replace = F)
t5 <- type5[split5,]
```

```
v5 <- type5[-split5,]

type6 <- filter(df_n, GlassType==6)
split6 <- sample.int(n = nrow(type6), size = floor(.8 * nrow(type6)), replace = F)
t6 <- type6[split6,]
v6 <- type6[-split6,]

type7 <- filter(df_n, GlassType==7)
split7 <- sample.int(n = nrow(type7), size = floor(.8 * nrow(type7)), replace = F)
t7 <- type7[split7,]
v7 <- type7[-split7,]

training <- rbind(t1, t2, t3, t5, t6, t7)
validation <- rbind(v1, v2, v3, v5, v6, v7)
```

Problem 1.8 (20 pts)

Implement the k-NN algorithm in R (do not use an implementation of k-NN from a package) and use your algorithm with a k=5 to predict the glass type for the following two cases: Use the whole normalized data set for this; not just the training data set. Note that you need to normalize the values of the new cases the same way as you normalized the original data. RI = 1.51621 | 12.53 | 3.48 | 1.39 | 73.39 | 0.60 | 8.55 | 0.00 | Fe = 0.08 RI = 1.5893 | 12.71 | 1.85 | 1.82 | 72.62 | 0.52 | 10.51 | 0.00 | Fe = 0.05

```
# Create a new data frame for the two new cases:
case_1 <- c(1.51621, 12.53, 3.48, 1.39, 73.39, 0.60, 8.55, 0.00, 0.08)
case_2 <- c(1.5893, 12.71, 1.85, 1.82, 72.62, 0.52, 10.51, 0.00, 0.05)
df_new <- matrix(c(case_1, case_2), 2, 9, byrow = T)

# Rename the column names
colnames(df_new) <- c("RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe")

df_new <- as_tibble(df_new)

# mean of each column(except ID and GlassType) of whole dataset
mean <- as_tibble(lapply(df[,-10], mean))
# sd of each column(except ID and GlassType) of whole dataset
sd <- as_tibble(lapply(df[,-10], sd))

# Z-score standardization function
normalize <- function(x, mean, sd) {
  return ((x - mean) / sd) }

# Normalize each new case using z-score and whole data's mean and sd, as new cases contains 0 values.
case_1 <- mapply(normalize, df_new[1,], mean, sd)
case_2 <- mapply(normalize, df_new[2,], mean, sd)

# Knn algorithm function
KNN_predict <- function (train, u, k) {

  # 1. Find neighbors
  m <- nrow(train)
  ds <- numeric(m)
  for (i in 1:m) {
    ds[i] <- sqrt(sum((train[i,] - u)^2))
```

```
  }

    # 2. Order the k neighbors
  order <- order(ds)
  k.closest <- order[1:k]

  # Find the mode
  ux <- unique(df_n$GlassType)
  return(ux[which.max(tabulate(match(df_n$GlassType[k.closest], ux)))])
}

# Find glass type for each case with k=5:
case_1_glass_type <- KNN_predict(df_n[,-10], case_1, 5)
cat("The glass type for case 1 is", case_1_glass_type, "\n")
```

```
## The glass type for case 1 is 1
```

```
case_2_glass_type <- KNN_predict(df_n[,-10], case_2, 5)
cat("The glass type for case 2 is", case_2_glass_type)
```

```
## The glass type for case 2 is 2
```

Problem 1.9 (5 pts)

Apply the knn function from the class package with k=5 and redo the cases from Question (8). Compare your answers.

```
library(class)

# case 1
knn(train = df_n[,-10], test = case_1, cl = df_n$GlassType, k = 5)
```

```
## [1] 1
## Levels: 1 2 3 5 6 7
```

```
# case 2
knn(train = df_n[,-10], test = case_2, cl = df_n$GlassType, k = 5)
```

```
## [1] 2
## Levels: 1 2 3 5 6 7
```

```
# The k value used for our own implemented is 5 and for knn() is 5.
# knn() from class package and our implemented knn algorithm generated the same glass type for each cas
```

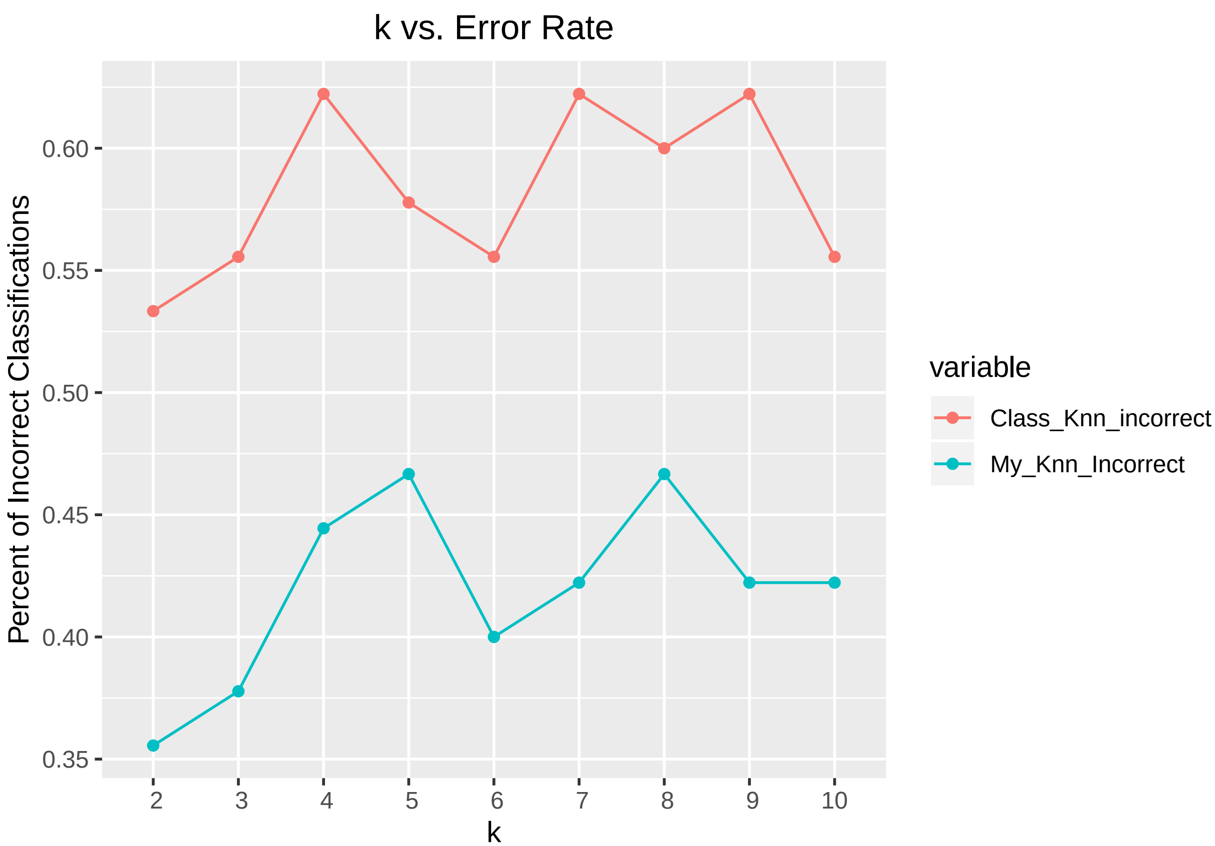
Problem 1.10 (10 pts)

Using your own implementation as well as the class package implementation of kNN, create a plot of k (x-axis) from 2 to 10 versus error rate (percentage of incorrect classifications) for both algorithms using ggplot.

```
# Part I: Use knn() from Class packege to get percentage of incorrect classifications------------

# Create a result matrix to hold the k values
predictions <- matrix(NA, nrow = nrow(validation), ncol = 9)

# k from 2 to 10
k <- c(2:10)
```

```r
# Using for loop for k from 2 to 10
for (i in k){
  # Store the predicted GlassType for each case (column-wise) in predictions matrix
  # for each k (2 to 10)
  predictions[, i-1] <- knn(train = training[,-10], test = validation[,-10],
                           cl = training$GlassType, k = i)
  }


# Function to find total of incorrect classification for each glass type
find_incorrect <- function(x){
  # Compare each column to GlassType, find the incorrects
  incorrects <- nrow(predictions) - sum(x==validation$GlassType)
  # Count the incorrect classifications
  return(sum(incorrects))
}

# Get the total incorrect classifications
total_incorrect <- apply(as.data.frame(predictions), 2, find_incorrect)



# Part II: Use our Knn algorithm function to get percentage of incorrect classifications
KNN_predict_1 <- function (train, u, k) {
  # 1. Find neighbors
  m <- nrow(train)
  ds <- numeric(m)
  for (i in 1:m) {
    ds[i] <- sqrt(sum((train[i,] - u)^2))
  }

  # 2. Order the k neighbors
  order <- order(ds)
  k.closest <- order[1:k]

  # Find the mode
  ux <- unique(training$GlassType)
  return(ux[which.max(tabulate(match(training$GlassType[k.closest], ux)))])
}

# Use a for loop for k from 2 to 10 and store result in data frame called my_knn
my_knn <- as.data.frame(matrix(NA, nrow = nrow(validation), ncol = 9))
for (j in 1:nrow(validation)){
  for (i in k){
    my_knn[j,i-1] <- KNN_predict_1(training[,-10], validation[j,-10], i)
  }
}

# Get the total incorrect classifications
my_incorrect <- apply(my_knn, 2, find_incorrect)

# Get a percentage for both versions of knn algorithm
my_percent <- as.data.frame(cbind(k, My_Knn_Incorrect = (my_incorrect/nrow(my_knn))))
```

```
percent <- as.data.frame(cbind(k, Class_Knn_incorrect = (total_incorrect/nrow(predictions))))


# Part III: Graph both algorithms on one ggplot--------------------------------------------------
# Merge the two data for ggplot
merged <- merge(percent, my_percent, by="k")
merged_1 <- reshape2::melt(merged, id.var="k")

# Creat a ggplot
library(ggplot2)
ggplot(data = merged_1, mapping = aes(x=k, y=value, col=variable)) +
  geom_line() +
  geom_point() +
  scale_x_discrete(limits=seq(k[1], k[length(k)], 1)) +
  xlab("k") +
  ylab("Percent of Incorrect Classifications") +
  ggtitle("k vs. Error Rate") +
  theme(plot.title = element_text(hjust = 0.5))
```



Problem 1.11 (5 pts)

Produce a cross-table confusion matrix showing the accuracy of the classification using knn from the class package with k = 5.

```
library(gmodels)
set.seed(123)
# k = 5
```

```r
knn_5 <- knn(train = training[,-10], test = validation[,-10],
             cl = training$GlassType, k = 5)

# Crosstable
CrossTable(x = validation$GlassType, y = knn_5)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  45
##
##
##                      | knn_5
## validation$GlassType |         1 |         2 |         5 |         6 |         7 | Row Total |
## ---------------------|-----------|-----------|-----------|-----------|-----------|-----------|
##                    1 |         9 |         5 |         0 |         0 |         0 |        14 |
##                      |     2.064 |     0.000 |     0.933 |     0.622 |     1.867 |           |
##                      |     0.643 |     0.357 |     0.000 |     0.000 |     0.000 |     0.311 |
##                      |     0.500 |     0.312 |     0.000 |     0.000 |     0.000 |           |
##                      |     0.200 |     0.111 |     0.000 |     0.000 |     0.000 |           |
## ---------------------|-----------|-----------|-----------|-----------|-----------|-----------|
##                    2 |         7 |         7 |         2 |         0 |         0 |        16 |
##                      |     0.056 |     0.302 |     0.817 |     0.711 |     2.133 |           |
##                      |     0.438 |     0.438 |     0.125 |     0.000 |     0.000 |     0.356 |
##                      |     0.389 |     0.438 |     0.667 |     0.000 |     0.000 |           |
##                      |     0.156 |     0.156 |     0.044 |     0.000 |     0.000 |           |
## ---------------------|-----------|-----------|-----------|-----------|-----------|-----------|
##                    3 |         1 |         3 |         0 |         0 |         0 |         4 |
##                      |     0.225 |     1.750 |     0.267 |     0.178 |     0.533 |           |
##                      |     0.250 |     0.750 |     0.000 |     0.000 |     0.000 |     0.089 |
##                      |     0.056 |     0.188 |     0.000 |     0.000 |     0.000 |           |
##                      |     0.022 |     0.067 |     0.000 |     0.000 |     0.000 |           |
## ---------------------|-----------|-----------|-----------|-----------|-----------|-----------|
##                    5 |         1 |         0 |         1 |         0 |         1 |         3 |
##                      |     0.033 |     1.067 |     3.200 |     0.133 |     0.900 |           |
##                      |     0.333 |     0.000 |     0.333 |     0.000 |     0.333 |     0.067 |
##                      |     0.056 |     0.000 |     0.333 |     0.000 |     0.167 |           |
##                      |     0.022 |     0.000 |     0.022 |     0.000 |     0.022 |           |
## ---------------------|-----------|-----------|-----------|-----------|-----------|-----------|
##                    6 |         0 |         0 |         0 |         2 |         0 |         2 |
##                      |     0.800 |     0.711 |     0.133 |    41.089 |     0.267 |           |
##                      |     0.000 |     0.000 |     0.000 |     1.000 |     0.000 |     0.044 |
##                      |     0.000 |     0.000 |     0.000 |     1.000 |     0.000 |           |
##                      |     0.000 |     0.000 |     0.000 |     0.044 |     0.000 |           |
## ---------------------|-----------|-----------|-----------|-----------|-----------|-----------|
```

```
##                    7 |          0 |          1 |          0 |          0 |          5 |          6 |
##                      |      2.400 |      0.602 |      0.400 |      0.267 |     22.050 |            |
##                      |      0.000 |      0.167 |      0.000 |      0.000 |      0.833 |      0.133 |
##                      |      0.000 |      0.062 |      0.000 |      0.000 |      0.833 |            |
##                      |      0.000 |      0.022 |      0.000 |      0.000 |      0.111 |            |
## ---------------------|------------|------------|------------|------------|------------|------------|
##         Column Total |         18 |         16 |          3 |          2 |          6 |         45 |
##                      |      0.400 |      0.356 |      0.067 |      0.044 |      0.133 |            |
## ---------------------|------------|------------|------------|------------|------------|------------|
##
##
```

Problem 1.12 (10 pts)

Download this (modified) version of the Glass data set containing missing values in column 4. Identify the missing values. Impute the missing values using your version of kNN using the other columns as predictor features.

```
# 1. Read in the data
missing <- read.csv("da5030.glass.data_with_missing_values.csv",
                    header = FALSE, stringsAsFactors = FALSE)
colnames(missing) <- c("ID", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe",
                       "GlassType")

# 2. We can use summary to check, Mg has 9 NA values
summary(missing)
```

```
##        ID               RI              Na              Mg
##  Min.   :  1.00   Min.   :1.511   Min.   :10.73   Min.   :0.000
##  1st Qu.: 54.25   1st Qu.:1.517   1st Qu.:12.91   1st Qu.:2.240
##  Median :107.50   Median :1.518   Median :13.30   Median :3.480
##  Mean   :107.50   Mean   :1.518   Mean   :13.41   Mean   :2.733
##  3rd Qu.:160.75   3rd Qu.:1.519   3rd Qu.:13.82   3rd Qu.:3.610
##  Max.   :214.00   Max.   :1.534   Max.   :17.38   Max.   :4.490
##                                                   NA's   :9
##        Al              Si              K                Ca
##  Min.   :0.290   Min.   :69.81   Min.   :0.0000   Min.   : 5.430
##  1st Qu.:1.190   1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240
##  Median :1.360   Median :72.79   Median :0.5550   Median : 8.600
##  Mean   :1.445   Mean   :72.65   Mean   :0.4971   Mean   : 8.957
##  3rd Qu.:1.630   3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172
##  Max.   :3.500   Max.   :75.41   Max.   :6.2100   Max.   :16.190
##
##        Ba              Fe             GlassType
##  Min.   :0.000   Min.   :0.00000   Min.   :1.00
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:1.00
##  Median :0.000   Median :0.00000   Median :2.00
##  Mean   :0.175   Mean   :0.05701   Mean   :2.78
##  3rd Qu.:0.000   3rd Qu.:0.10000   3rd Qu.:3.00
##  Max.   :3.150   Max.   :0.51000   Max.   :7.00
##
```

```
unknown <- missing[!complete.cases(missing),] %>% select(-ID)
known <- missing[complete.cases(missing),] %>% select(-ID)
```

```r
# 3. z-score normalization without ID, GlassType and NA rows
z_normalize <- function(x){
  return ((x - mean(x)) / sd(x))}

missing <- missing %>% select(c(-ID, -GlassType))
missing <- missing[complete.cases(missing),]
missing_n <- as_tibble(lapply(missing, z_normalize))

# 4 Normalize unknown
mean <- as_tibble(lapply(known[1:9], mean))
sd <- as_tibble(lapply(known[1:9], sd))
# This normalization is special for unknown, mean and sd from known
normalize <- function(x, mean, sd) {
  return ((x - mean) / sd) }

unknown_n <- as_tibble(mapply(normalize, unknown[,-10], mean, sd))

# 5 Store the known Mg in a data frame to be passed into the kNN function,
# for prediction of unknown Mg
target_mg <- as.data.frame(missing$Mg)

# 6. Remove Mg from both training and unknown data set
missing_n <- missing_n%>%select(c(-Mg))
unknown_n <- unknown_n%>%select(c(-Mg))

# 7 Using knn.reg function created from Question 2, (except here use unweighted version)
knn.reg <- function(new_data, target_data, train_data, k){

  # 1. Find neighbors
  m <- nrow(train_data)
  ds <- numeric(m)
  for (i in 1:m){
    ds[i] <- sqrt(sum((train_data[i,] - new_data)^2))
  }

  # 2. Order the k neighbors
  order <- order(ds)
  k.closest <- order[1:k]

  # 3. Take the mean
  for (i in 1:length(k.closest))
    mean <- mean(target_mg$`missing$Mg`[k.closest])
    return(mean)
}

# 8 Use a for loop to predict the 9 missing values in column Mg
Mg <- matrix(NA, 9, 1, byrow = F)
for (i in 1:nrow(unknown_n)){
  Mg[i] <- knn.reg(unknown_n[i,], target_mg, missing_n, 5)
}

# 9 Impute the missing values with the predicted
unknown$Mg <- Mg
```

```
unknown
```

```
##          RI    Na    Mg   Al    Si    K    Ca   Ba   Fe GlassType
## 20  1.51735 13.02 3.542 1.69 72.73 0.54  8.44 0.00 0.07         1
## 30  1.51784 13.08 3.552 1.28 72.86 0.60  8.49 0.00 0.00         1
## 95  1.51629 12.71 3.532 1.49 73.28 0.67  8.24 0.00 0.00         2
## 163 1.52211 14.19 3.462 0.91 71.36 0.23  9.14 0.00 0.37         3
## 169 1.51666 12.86 2.184 1.83 73.88 0.97 10.17 0.00 0.00         5
## 184 1.51969 14.56 1.962 0.56 73.48 0.00 11.22 0.00 0.00         6
## 194 1.51719 14.75 0.000 2.00 73.02 0.00  8.53 1.59 0.08         7
## 200 1.51609 15.01 0.000 2.51 73.05 0.05  8.83 0.53 0.00         7
## 208 1.51831 14.39 0.652 1.82 72.86 1.41  6.47 2.88 0.00         7
```

Problem 2.1 (0 pts)

Investigate this data set of home prices in King County (USA).

```
home <- read.csv("kc_house_data.csv", header = TRUE, stringsAsFactors = FALSE)
head(home)
```

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

```
str(home)
```

```
## 'data.frame':    21613 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  3 3 3 5 3 3 3 3 3 3 ...
```

```
##  $ grade        : int  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: int  1340 1690 2720 1360 1800 4760 2238 1650 1780 2390 ...
##  $ sqft_lot15   : int  5650 7639 8062 5000 7503 101930 6819 9711 8113 7570 ...
```

```r
colnames(home)
```

```
##  [1] "id"            "date"          "price"         "bedrooms"
##  [5] "bathrooms"     "sqft_living"   "sqft_lot"      "floors"
##  [9] "waterfront"    "view"          "condition"     "grade"
## [13] "sqft_above"    "sqft_basement" "yr_built"      "yr_renovated"
## [17] "zipcode"       "lat"           "long"          "sqft_living15"
## [21] "sqft_lot15"
```

```r
summary(home)
```

```
##        id                date               price            bedrooms
##  Min.   :1.000e+06   Length:21613       Min.   :  75000   Min.   : 0.000
##  1st Qu.:2.123e+09   Class :character   1st Qu.: 321950   1st Qu.: 3.000
##  Median :3.905e+09   Mode  :character   Median : 450000   Median : 3.000
##  Mean   :4.580e+09                      Mean   : 540088   Mean   : 3.371
##  3rd Qu.:7.309e+09                      3rd Qu.: 645000   3rd Qu.: 4.000
##  Max.   :9.900e+09                      Max.   :7700000   Max.   :33.000
##    bathrooms      sqft_living       sqft_lot           floors
##  Min.   :0.000   Min.   :  290   Min.   :    520   Min.   :1.000
##  1st Qu.:1.750   1st Qu.: 1427   1st Qu.:   5040   1st Qu.:1.000
##  Median :2.250   Median : 1910   Median :   7618   Median :1.500
##  Mean   :2.115   Mean   : 2080   Mean   :  15107   Mean   :1.494
##  3rd Qu.:2.500   3rd Qu.: 2550   3rd Qu.:  10688   3rd Qu.:2.000
##  Max.   :8.000   Max.   :13540   Max.   :1651359   Max.   :3.500
##    waterfront            view          condition         grade
##  Min.   :0.000000   Min.   :0.0000   Min.   :1.000   Min.   : 1.000
##  1st Qu.:0.000000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.: 7.000
##  Median :0.000000   Median :0.0000   Median :3.000   Median : 7.000
##  Mean   :0.007542   Mean   :0.2343   Mean   :3.409   Mean   : 7.657
##  3rd Qu.:0.000000   3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.: 8.000
##  Max.   :1.000000   Max.   :4.0000   Max.   :5.000   Max.   :13.000
##    sqft_above   sqft_basement       yr_built     yr_renovated
##  Min.   : 290   Min.   :   0.0   Min.   :1900   Min.   :   0.0
##  1st Qu.:1190   1st Qu.:   0.0   1st Qu.:1951   1st Qu.:   0.0
##  Median :1560   Median :   0.0   Median :1975   Median :   0.0
##  Mean   :1788   Mean   : 291.5   Mean   :1971   Mean   :  84.4
##  3rd Qu.:2210   3rd Qu.: 560.0   3rd Qu.:1997   3rd Qu.:   0.0
##  Max.   :9410   Max.   :4820.0   Max.   :2015   Max.   :2015.0
##     zipcode           lat             long        sqft_living15
##  Min.   :98001   Min.   :47.16   Min.   :-122.5   Min.   : 399
##  1st Qu.:98033   1st Qu.:47.47   1st Qu.:-122.3   1st Qu.:1490
##  Median :98065   Median :47.57   Median :-122.2   Median :1840
##  Mean   :98078   Mean   :47.56   Mean   :-122.2   Mean   :1987
```

```
##  3rd Qu.:98118   3rd Qu.:47.68   3rd Qu.:-122.1   3rd Qu.:2360
##  Max.   :98199   Max.   :47.78   Max.   :-121.3   Max.   :6210
##    sqft_lot15
##  Min.   :   651
##  1st Qu.:  5100
##  Median :  7620
##  Mean   : 12768
##  3rd Qu.: 10083
##  Max.   :871200
```

Problem 2.2 (5 pts)

Save the price column in a separate vector/dataframe called target_data. Move all of the columns except the ID, date, price, yr_renovated, zipcode, lat, long, sqft_living15, and sqft_lot15 columns into a new data frame called train_data.

```r
library(tidyverse)

# Save the price column to target_data
target_data <- as.data.frame(home$price)

# train_data with specified columns in stated in the question:
train_data <- home %>% select(c(-id, -date, -price, -yr_renovated, -zipcode,
                                -lat, -long, -sqft_living15, -sqft_lot15))

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
```

Problem 2.3 (5 pts)

Normalize all of the columns (except the boolean columns waterfront and view) using min-max normalization.

```r
# Min-max normalization
min_max_normalize <- function(x){
  return((x - min(x)) / (max(x) - min(x)))
}

# Unselect boolean columns: waterfront and view and normalize
train_data_1 <- train_data %>% select(c(-waterfront, -view))
train_data_n <- as_tibble(lapply(train_data_1, min_max_normalize))

head(train_data_n)
```

```
## # A tibble: 6 x 10
##   bedrooms bathrooms sqft_living sqft_lot floors condition grade sqft_above
##      <dbl>     <dbl>       <dbl>    <dbl>  <dbl>     <dbl> <dbl>      <dbl>
## 1   0.0909     0.125      0.0672  0.00311    0         0.5 0.5       0.0976
## 2   0.0909     0.281      0.172   0.00407    0.4       0.5 0.5       0.206
## 3   0.0606     0.125      0.0362  0.00574    0         0.5 0.417     0.0526
## 4   0.121      0.375      0.126   0.00271    0         1   0.5       0.0833
## 5   0.0909     0.25       0.105   0.00458    0         0.5 0.583     0.152
## 6   0.121      0.562      0.387   0.0614     0         0.5 0.833     0.395
## # ... with 2 more variables: sqft_basement <dbl>, yr_built <dbl>
```

Problem 2.4 (15 pts)

Build a function called knn.reg that implements a regression version of kNN that averages the prices of the k nearest neighbors using a weighted average where the weight is 3 for the closest neighbor, 2 for the second closest and 1 for the remaining neighbors (recall that a weighted average requires that you divide the sum product of the weight and values by the sum of the weights). It must use the following signature: knn.reg (new_data, target_data, train_data, k) where new_data is a data frame with new cases, target_data is a data frame with a single column of prices from (2), train_data is a data frame with the features from (2) that correspond to a price in target_data, and k is the number of nearest neighbors to consider. It must return the predicted price.

```r
# knn.reg function
knn.reg <- function(new_data, target_data, train_data, k){

  # 1. Find neighbors
  m <- nrow(train_data) # # of rows of train data 21613
  ds <- numeric(m) # create a vector
  for (i in 1:m){
    ds[i] <- sqrt(sum((train_data[i,] - new_data)^2))
  }

  # 2. Order the k neighbors
  order <- order(ds)
  k.closest <- order[1:k]

  # 3. Weight
  w <- c(3,2,1,1)
  sum <- 0
  for (i in 1:length(w))
    sum <- sum + target_data$`home$price`[k.closest][i] * w[i]
    return(sum/sum(w))
}
```

Problem 2.5 (5 pts)

Forecast the price of this new home using your regression kNN using k = 4: bedrooms = 4 | bathrooms = 3 | sqft_living = 4852 | sqft_lot = 10244 | floors = 3 | waterfront = 0 | view = 1 | condition = 3 | grade = 11 sqft_above = 1960 | sqft_basement = 820 | yr_built = 1978

```r
# New home without boolean columns: waterfront and view
new_data <- (data.frame(bedrooms = 4, bathrooms = 3, sqft_living = 4852, sqft_lot = 10244,
                        floors = 3, waterfront = 0, view = 1, condition = 3, grade = 11,
                       sqft_above = 1960, sqft_basement = 820, yr_built = 1978)) %>%
```

```
  select(c(-waterfront, -view))

# min and max of train_data:
# (train_data_1 here doesn't have boolean columns: waterfront and view)
min <- as_tibble(lapply(train_data_1, min))
max <- as_tibble(lapply(train_data_1, max))

# min and max function:
min_max_func <- function(x, min, max){
  return((x - min) / (max - min))
}

# Normalize new data with min_max_func and train data's min and max
new_data_n <- mapply(min_max_func, new_data, min, max)

# Forecast new home price using knn.reg() and k=4
new_home_forecasted_price <- knn.reg(new_data_n, target_data, train_data_n, 4)
new_home_forecasted_price
```

```
## [1] 1240001
```

Helpful Link(s): https://rpubs.com/euclid/343644, https://www.datacamp.com/community/tutorials/r-tutorial-apply-family