```r
#==================================================================================
#==== CODE FOR STATISTICAL LEARNING II, hw5 PROBLEM 3 =============================
#==================================================================================




# Necessary libraries
library(geometry)
library(tidyr)
library(dplyr)

# Set the working directory
setwd("C:/Users/jrdha/OneDrive/Desktop/USU_Fa2018/Moon__SLDM2/hw5/problem4")

# trainData is in trainData$X
given_data <- R.matlab::readMat("anomaly.mat") %>% lapply(t) %>% lapply(as_tibble)
trainData <- as.data.frame(given_data$X)
names(trainData)[1] <- 'X'
trainData$index <- rownames(trainData)

# Store the given test points 1 and 2
testPt_1 <- given_data$xtest1$V1
testPt_2 <- given_data$xtest2$V1





#==================================================================================
#==== ALL FUNCTIONS ===============================================================
#==================================================================================



# This function returns (calculating first) the Gaussian kernel for the vector
# of values passed in.
# Takes as arguments: vector (self-explanatory), h is the bandwidth parameter.
g_Kernel <- function(vector, h){
  vec_len <- length(vector)
  vector.h <- vector/h
  if(vec_len > 1){
    kernel_val <- ((2*pi)^(-vec_len/2))*exp(-0.5*(dot(vector.h, vector.h)))
  } else {
    kernel_val <- ((2*pi)^(-vec_len/2))*exp(-0.5*(vector.h * vector.h))
  }
  return((h^(-vec_len)) * kernel_val)
}
```

```r
# This function calculates (and returns) the value of the objective function for
# the data that is passed in via LS-LOOCV using a given bandwidth.
# Takes as arguments: given_data (self-explanatory), h is the bandwidth param.
est_BandWidth <- function(given_data, h){
  data_len <- nrow(given_data)
  t1 <- 0
  t2 <- 0
  ind <- given_data$index
  X <- given_data$X

  # Loop through all data
  for(i in 1:data_len){
    for(j in 1:data_len){
      newTerm1 <- g_Kernel(vector = (X[i] - X[j]), h = (sqrt(2)*h))
      t1 <- t1 + newTerm1
      if(j != i){
        newTerm2 <- g_Kernel(vector = (X[i] - X[j]), h = h)
        t2 <- t2 + newTerm2
      }
    }
  }
  t1 <- (1/(data_len^2)) * t1
  t2 <- (2/(data_len*(data_len-1))) * t2
  final_val <- t1 - t2
  return(final_val)
}


# This function returns (after calculating) the KDE for all points in a given
# range. It calls the KDE_one_pt function for each of these points.
# Takes as arguments: range (range of values for which KDE is calculated),
# X (the training dataframe), h (bandwidth parameter).
KDE_all_pts <- function(range, X, h){

  density_estimate <- data.frame(x = range, density = double(length(range)))
  for(i in 1:length(range)){
    density_estimate$density[i] <- KDE_one_pt(point = range[i], X = X, h = h)
  }
  return(density_estimate)
}
```

```r
# This function returns (after calculating) the KDE for a single point.
# Takes as arguments: point (the point for which KDE is being estimated),
# X (training dataframe), h (bandwidth parameter).
KDE_one_pt <- function(point, X, h){
  term <- 0
  len_data <- length(X)
  for(i in 1:len_data){
    val <- g_Kernel(vector = (X[i] - point), h = h)
    term <- term + val
  }
  f_hat <- (1/len_data) * term
  return(f_hat)
}


# This function returns (after calculating) the LOO (leave one out) KDE, and is
# used by the outlierScore_1 function.
# Takes as arguments: point (point on which we're calculating KDE), X (training
# dataframe), exclude.i (index indicating which single point is to be excluded),
# h (bandwidth).
f_hat.i <- function(point, X, exclude.i, h){

  term <- 0
  n <- length(X)
  for(i in 1:n){
    if(i != exclude.i){
      val <- g_Kernel(vector = (point - X[i]), h = h)
      term <- term + val
    }
  }
  f_hat <- (1/(n-1)) * term
  return(f_hat)
}


# This function returns (after calculating) the OutlierScore_1 that is defined
# in part (c) of problem 4.
# Takes as arguments: point (calculating OutlierScore_1 for this point),
# X (training dataframe), h (bandwidth).
outlierScore_1 <- function(point, X, h){

  len_data <- length(X)
  numer_frac <- KDE_one_pt(point = point, X = X, h = h)
  denom_frac <- 0
  for(i in 1:len_data){
    term<- f_hat.i(point = X[i], X = X, exclude.i = i, h = h)
    denom_frac <- term + denom_frac
  }
  denom_frac <- (1/len_data) * denom_frac
  return(numer_frac/denom_frac)
}
```

```r
# This function returns (after calculating) an object, KNN, that contains the K
# nearest neighbors to a point, and the distances of each of these neighbors to
# that point.
# Takes as arguments: point (finding KNN and their distances for this point),
# X (training dataframe), k (the "k" in KNN: number of neighbors).
findKNN_distances <- function(point, X, k){

  pointvector <- rep(point, length(X))
  distance <- abs(X - pointvector)
  given_data <- data.frame(X = X, distance = distance)
  given_data <- arrange(given_data, distance)
  KNN <- slice(given_data, 1:k)
  return(KNN)
}


# This function returns (after calculating) the OutlierScore_2 that is defined
# in part (e) of problem 4.
# Takes as arguments: point (calculating OutlierScore_1 for this point),
# X (training dataframe), k (the "k" in KNN: number of neighbors).
outlierScore_2 <- function(point, X, k){

  KNN <- findKNN_distances(point, X, k)
  numer_frac <- KNN$distance[k]

  denom_frac <- (1/k) * sum(KNN$distance)

  return(numer_frac/denom_frac)
}




#================================================================================
#===== FUNCTION CALLS, CODE THAT GIVES ANSWERS TO QUESTIONS IN PROBLEM 4 =======
#================================================================================

# Define a grid of bandwidths to search through.
grid_list <- seq(.01,.5, by = .01)
len_grid <- length(grid_list)

# Using LS-LOOCV w/Gaussian kerenel calculate objective function for
# each value in grid_list
h_vals <- data.frame(index = 1:len_grid, h = grid_list, objF = rep(0, len_grid))
for(i in 1:nrow(h_vals)){
  objF <- est_BandWidth(given_data = trainData, h = h_vals$h[i])
  h_vals$index[i] <- i
  h_vals$objF[i] <- objF
}
```

```r
# Get the bandwith parameter h for which objective function is minimized (h_hat)
min_h <- h_vals[h_vals$objF == min(h_vals$objF),]
h_hat <- min_h$h
h_hat

# Define a sequence such that the KDE representation looks smooth (granularity).
density_range <- seq(-2, 4, by = 0.01)
# Estimate the density.
density_estimate <- KDE_all_pts(range = density_range,
                                X = trainData$X,
                                h = h_hat)
# Plot the estimated density obtained via KDE
plot(density_estimate$x,
     density_estimate$density,
     type = 'l',
     main = "Kernel Density Estimate Plot, h = 0.08",
     xlab = "X values",
     ylab = "density")

# OutlierScore_1 for xtest1 and xtest2 using optimal bandwith parameter h
os1_testPt_1 <- outlierScore_1(point = testPt_1, X = trainData$X, h = 0.08)
os1_testPt_2 <- outlierScore_1(point = testPt_2, X = trainData$X, h = 0.08)
os1_testPt_1
os1_testPt_2

# OutlierScore_2 for xtest1 with values of k specified in the prompt
os2_testPt_1.100 <- outlierScore_2(point = testPt_1, X = trainData$X, k = 100)
os2_testPt_1.150 <- outlierScore_2(point = testPt_1, X = trainData$X, k = 150)
os2_testPt_1.200 <- outlierScore_2(point = testPt_1, X = trainData$X, k = 200)
os2_testPt_1.100
os2_testPt_1.150
os2_testPt_1.200

# OutlierScore_2 for xtest2, with values of k specified in the prompt
os2_testPt_2.100 <- outlierScore_2(point = testPt_2, X = trainData$X, k = 100)
os2_testPt_2.150 <- outlierScore_2(point = testPt_2, X = trainData$X, k = 150)
os2_testPt_2.200 <- outlierScore_2(point = testPt_2, X = trainData$X, k = 200)
os2_testPt_2.100
os2_testPt_2.150
os2_testPt_2.200
```