#===============================================================================  
#==== 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)  
}

# 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)   
}

# 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)  
}  
  
  
# 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  
}  
  
# 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