```
#### STAT 641 Assignment 3

# 2.)
Assign3_BrainSize <- read.csv("C:/Users/jackr/OneDrive/Desktop/Graduate School Courses/STAT 641 - Methods of STAT I/RawData/Assign3_BrainSize.csv")
Assign3_BrainSize

LLitterSize = Assign3_BrainSize$Large.Litter.Size[1:44]
SLitterSize = Assign3_BrainSize$ï..Small.Litter.Size

quantile(LLitterSize, na.rm = TRUE)

# 3.)
# a)
# probably most appropriate way to calculate the kernel density in R
d = density(LLitterSize, kernel = "g", bw = 3, na.rm = TRUE)
dd = approxfun(d$x, d$y)
dd(3)
dd(16)

# Gaussian kernel density estimate for a hard coded Y value
n = length(LLitterSize)
h = 3
y = 3
kern_Density_estimate = NULL
for(i in seq_along(LLitterSize)) {
  kern_Density_estimate[i] = (1/(n*h))*(1/sqrt(2*pi))*exp(-0.5*((y-LLitterSize[i])/h)^2)
}
sum(kern_Density_estimate)




n = length(LLitterSize)
h = 3
y = 16
kern_Density_estimate = NULL
for(i in seq_along(LLitterSize)) {
  kern_Density_estimate[i] = (1/(n*h))*(1/sqrt(2*pi))*exp(-0.5*((y-LLitterSize[i])/h)^2)
}
sum(kern_Density_estimate)


# kernel density estimate for multiple x values.
# Y-values here are chosen just as a sequence from 0 to the max value of our dataset (rounded up to the nearest integer)
# n = length(LLitterSize)
# h = 3
# Y_vals = seq(from = 0, to = max(ceiling(LLitterSize)), by=1)
# kern_Density_estimate = NULL
# kern_Density_temp = NULL
```

```
# for(i in seq_along(Y_vals)) {
#   for(j in seq_along(LLitterSize)){
#     kern_Density_temp[j] = (1/(n*h))*(1/sqrt(2*pi))*exp(-0.5*((Y_vals[i]-
LLitterSize[j])/h)^2)
#   }
#   kern_Density_estimate[i] = sum(kern_Density_temp)
#   if(i == max(Y_vals)){ # if statement is fucking up this loop.. idk why..
figure out later, not needed for hw
#     dataf = cbind(Y_vals, kern_Density_estimate)
#   }
# }
# kern_Density_estimate


ceiling(length(LLitterSize)/5)

#b)
hist(LLitterSize, freq=FALSE, breaks = ceiling(length(LLitterSize)/5))
lines(d, col="red", lwd=2)
# f(3) approx .08
# f(16) approx .02

#c-d)
LLitterSize[which(LLitterSize == min(abs(16-LLitterSize), na.rm = TRUE) + 16)]
LLitterSize[which(LLitterSize == max(abs(16-LLitterSize), na.rm = TRUE) + 16)]

# or can use our loop above with y = 16

LLitterSize[which(kern_Density_estimate == min(kern_Density_estimate))]
LLitterSize[which(kern_Density_estimate == max(kern_Density_estimate))]


# 4.)
# (a)
# pdf estimate for Large Litter Size
plot(density(LLitterSize, kernel = "g", bw = 3))
# cdf estimate for Large litter size
QLarge = quantile(LLitterSize, probs = seq(0,1,0.01))
plot(QLarge, y = seq(0,1,0.01), xlim = c(0,36),
     type = "l", xlab = "Brain Weight", ylab = "Density",
     main = "Emperical Distribution Function of Large Litter Size Brain
Weights", cex.main = .75)
# quantile estimate for Large litter size
plot(y = QLarge, x = seq(0,1,0.01), ylim = c(0,36),
     type = "l", xlab = "Density", ylab = "Brain Weight",
     main = "Emperical Quantile Function of Large Litter Size Brain Weights",
cex.main = .75)



#pdf estimate for Small Litter Size
plot(density(SLitterSize,  kernel = "g", bw = 3))
# cdf estimate for Small litter size
QSmall = quantile(SLitterSize, probs = seq(0,1,0.01))
```

```
plot(QSmall, y = seq(0,1,0.01), xlim = c(0,36),
     type = "l", xlab = "Brain Weight", ylab = "Density",
     main = "Emperical Distribution Function of Small Litter Size Brain
Weights", cex.main = .75)
# quantile estimate for Large litter size
plot(y = QSmall, x = seq(0,1,0.01), ylim = c(0,36),
     type = "l", xlab = "Density", ylab = "Brain Weight",
     main = "Emperical Quantile Function of Small Litter Size Brain Weights",
cex.main = .75)
```