First we load the necessary libraries and data

```r
library(foreign)
library(gam)
```

```r
# Loading required package: splines
# Loading required package: foreach
# Loaded gam 1.14-4
d<-read.dta("https://people.ucsc.edu/~aspearot/Econ_217/org_example.dta")
```

```r
# Warning in read.dta("https://people.ucsc.edu/~aspearot/Econ_217/
# org_example.dta"): value labels ('fipscounty') for 'fipscounty' are missing
# Warning in read.dta("https://people.ucsc.edu/~aspearot/Econ_217/
# org_example.dta"): value labels ('ind12') for 'ind12' are missing
d$gender<-as.factor(ifelse(d$female==1,"Female","Male"))
d<-subset(d,is.na(rw)==FALSE&is.na(educ)==FALSE&is.na(age)==FALSE&is.na(female)==FALSE&is.na(wbho)==FALSE)
```

Restrict to CA in years 2003 and later

```r
sd<-subset(d,year>=2003&state=="CA")
```

Run model (using lo is fine as well since I didn’t specify)

```r
gamfit<-gam(rw~s(age)+educ+wbho+gender,data=sd)
```

Plot results

```r
par(mfrow=c(2,2))
plot(gamfit,terms="s(age)",se=TRUE,rugplot=FALSE,main="Real Wage on Smooth function of Age \n")
abline(h=0)
plot(gamfit,terms="educ",se=TRUE,rugplot=FALSE,main="Real Wage on Education Categories\n")
abline(h=0)
plot(gamfit,terms="wbho",se=TRUE,rugplot=FALSE,main="Real Wage on Race/Ethnicity Categories\n")
abline(h=0)
plot(gamfit,terms="gender",se=TRUE,rugplot=FALSE,main="Real Wage on Gender\n")
abline(h=0)
```
Part B

Load the Caret library to run the KNN model

```r
library(caret)

## Loading required package: lattice
## Loading required package: ggplot2

Run the KNN model

```r
model.knn <- train(rw ~ age+educ+wbho+gender, data = sd, method = "knn")
print(model.knn)
```

```r
## k-Nearest Neighbors
##
## 19622 samples
## 4 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 19622, 19622, 19622, 19622, 19622, 19622, ...
## Resampling results across tuning parameters:
##
## # k   RMSE   Rsquared   MAE
## 5 15.6924 0.2738425 10.24952
## 7 15.5949 0.2788025 10.20420
## 9 15.5344 0.2819668 10.16952
```
RMSE was used to select the optimal model using the smallest value. The final value used for the model was k = 9.

Construct an age vector for generating predictions, and create “maledata” and “femaildata” to generate predictions at each age for a white respondent with a college education.

```r
age<-seq(18,70,by=1)
maledata<-data.frame(age)
maledata$gender<="Male"
maledata$wbho<="White"
maledata$educ<="College"
femaledata<-maledata
femaledata$gender<="Female"
```

Generate male and female predictions by age

```r
maledpredict <- predict(model.knn,maledata)
femalepredict <- predict(model.knn,femaledata)
perglass<-(femalepredict-maledpredict)/maledpredict
glass<-(femalepredict-maledpredict)
```

Plot the actual gap and the percentage gap (either is fine for the answer.)

```r
par(mfrow=c(1,2))
plot(glass-age,type="l")
plot(perglass-age,type="l")
```
Define our sampling function

```r
randomSample = function(df,n) {
  return (df[sample(nrow(df), n, replace=TRUE),])
}
```

Create a dataset with the original percentage predictions and age

```r
res<-data.frame(age,pergap)
```

Run the bootstrap procedure, generating a new estimated curve for each random sample

```r
for(r in 1:20) {
  
  rsd<-randomSample(sd,nrow(sd))
  
  rmodel.knn <- train(rw ~ age+educ+wbho+gender, data = rsd, method = "knn")
  
  r_malepredict <- predict(rmodel.knn,maledata)
  r_femalepredict <- predict(rmodel.knn,femaledata)
  
  res$r_pergap<-((r_femalepredict-r_malepredict)/r_malepredict)
  
  names(res)[ncol(res)]<-paste("pergap",r,sep="_")
  print(r)
}
```

Calculate the 5th and 95th percentile for each age.

```r
res$lower<-NA
res$upper<-NA

for(r in 1:nrow(res)){
  bounds<-quantile(res[r,3:ncol(res)],prob=c(0.05,0.95),na.rm=TRUE)
  res$lower[r]<-as.numeric(bounds[1])
  res$upper[r]<-as.numeric(bounds[2])
}
```

Plot the results, with the bootstrap percentile intervals in red.

```r
plot(pergap~age,res,type="l",lwd=2,col="black",ylim=c(min(res$lower),max(res$upper)))
lines(lower~age,res,lwd=2,col="red",lty="dashed")
lines(upper~age,res,lwd=2,col="red",lty="dashed")
abline(h=0)
```
Problem #2
Generate Test Dataset for the answer

```r
x <- seq(0, 10, length.out = 100)
cutoff1 <- runif(1, 3, 4)
b1 <- runif(1, 3, 4)
cutoff2 <- runif(1, 7, 8)
b2 <- runif(1, -5, -4)
u <- rnorm(length(x), 0, 2)
y <- x + b1 * ifelse(x > cutoff1, x - cutoff1, 0) + b2 * ifelse(x > cutoff2, x - cutoff2, 0) + u
d <- data.frame(x, y)
```

For Part A, we begin by creating a grid over which to search for the two cutoffs:

```r
hs1 <- seq(3, 4, 0.1)
hs2 <- seq(7, 8, 0.1)
```

Search over the grid, running a cross-validation at each step:

```r
for (h1 in 1:length(hs1)) {
  for (h2 in 1:length(hs2)) {
    ### At each pair of potential cutoffs, run a cross-validation procedure
    for (i in 1:nrow(d)) {
      # drop one observation
    }
  }
}
```
d_drop <- d[i,]

# Keep the rest

d_keep <- d[-i,]

# Fit the model

fit <- lm(y ~ x + ifelse(x > hs1[h1], x - hs1[h1], 0) + ifelse(x > hs2[h2], x - hs2[h2], 0), d_keep)

# Predict the dropped observation and calculate the squared error of the prediction

dropfit <- predict(fit, d_drop, se = FALSE)
sqerr <- (d_drop$y - as.numeric(dropfit))^2

# Save Results (this programming is slow in larger applications, but will get the job done)

if (i*h1*h2==1) {results <- data.frame(hs1[h1], hs2[h2], i, sqerr)}
if (i*h1*h2>1) {results <- rbind(results, data.frame(hs1[h1], hs2[h2], i, sqerr))}

}

print(h1)

names(results)[1:2] <- c("h1", "h2")

Next, we find the best pair of cutoffs. In class, we used which.min for a few applications. Here, we’ll simply sum SSR at each pair and assign it to a new variable in the dataset. Then, we will sort to find the minimum, and then select the first row (the lowest SSR)

results$SSRs <- ave(results$sqerr, paste(results$h1, results$h2), FUN = sum, na.rm = TRUE)
results <- results[order(results$SSRs),]

cutoff1_hat <- results$h1[1]
cutoff2_hat <- results$h2[1]

At these cutoffs, we now run the model to generate coefficients, and report the results

fit <- lm(y ~ x + ifelse(x > cutoff1_hat, x - cutoff1_hat, 0) + ifelse(x > cutoff2_hat, x - cutoff2_hat, 0), d)

print(c("Intercept =", round(as.numeric(coef(fit)[1]), digits = 3)))
## [1] "Intercept = -0.534"

print(c("Linear Coefficient =", round(as.numeric(coef(fit)[2]), digits = 3)))
## [1] "Linear Coefficient = 1.081"

print(c("Kink #1 =", cutoff1_hat))
## [1] "Kink #1 = 3.7"

print(c("Coefficient on Kink #1 =", round(as.numeric(coef(fit)[3]), digits = 3)))
## [1] "Coefficient on Kink #1 = 3.29"
Part B

For part B, we define a random sampling function

```r
randomSample = function(df,n) {
  return (df[sample(nrow(df), n, replace=TRUE),])
}
```

Then we iterate using 20 bootstrap samples and then utilize the same procedure within each bootstrap sample.

```r
for(r in 1:20) {
  sd<-randomSample(d,nrow(d))
  hs1<-seq(3,4,0.1)
  hs2<-seq(7,8,0.1)
  for(h1 in 1:length(hs1)){
    for(h2 in 1:length(hs2)){
      for(i in 1:nrow(sd)){
        #drop one observation
        d_drop<-sd[i,]
        #Keep the rest
        d_keep<-sd[-i,]
        #Fit the model
        fit<-lm(y~x+ifelse(x>hs1[h1],x-hs1[h1],0)+ifelse(x>hs2[h2],x-hs2[h2],0),d_keep)
        #Predict the dropped observation and calculate the squared error of the prediction
        dropfit<-predict(fit,d_drop,se=FALSE)
        sqrerr<-(d_drop$y-as.numeric(dropfit))^2
        if(i*h1*h2==1){results<-data.frame(hs1[h1],hs2[h2],i,sqrerr)}
        if(i*h1*h2>1){results<-rbind(results,data.frame(hs1[h1],hs2[h2],i,sqrerr))}
      }
    }
  }
  names(results)[1:2]<-c("h1","h2")
  results$SSRs<-(results$sqrerr,paste(results$h1,results$h2),FUN=sum,na.rm=TRUE)
}
```
results<-results[order(results$SSRs),]

h1_hat<-results$h1[1]
h2_hat<-results$h2[1]

if(r==1){results_h<-data.frame(r,h1_hat,h2_hat)}
if(r>1){results_h<-rbind(results_h,data.frame(r,h1_hat,h2_hat))}

print(r)

generate the confidence intervals for each cutoff, and display the results.
confid_int1<-quantile(results_h$h1_hat,prob=c(0.05,0.95),na.rm=TRUE)
confid_int2<-quantile(results_h$h2_hat,prob=c(0.05,0.95),na.rm=TRUE)

print(c("Lower Bound Kink #1 = ",as.numeric(confid_int1[1])))
## [1] "Lower Bound Kink #1 = " "3.385"
print(c("Kink #1 =",cutoff1_hat))
## [1] "Kink #1 =" "3.7"
print(c("Upper Bound Kink #1 = ",as.numeric(confid_int1[2])))
## [1] "Upper Bound Kink #1 = " "4"
print(c("Lower Bound Kink #2 = ",as.numeric(confid_int2[1])))
## [1] "Lower Bound Kink #2 = " "7.495"
print(c("Kink #2 =",cutoff2_hat))
## [1] "Kink #2 =" "7.9"
print(c("Upper Bound Kink #2 = ",as.numeric(confid_int2[2])))
## [1] "Upper Bound Kink #2 = " "8"