Advanced data concepts in R

In this handout, we explore some more advanced data manipulation and coding techniques. Some concepts covered in this handout include:

- `for()` loops
- Loop wrapping: `apply()`; `colMeans()`; `colSums()`
- Summarizing data subsets with `by()`
- String searching with `grep()`; `Date` classes; and more.

`for()` loops

Looping is a very important concept, not only in R but in any advanced computer language. Understanding the fundamentals of looping will help understand many of the more advanced R functions.

A "loop" generally defined is "something done repeatedly." In R, these are often calls `for()` loops, because writing them requires use of the `for()` function. The basic form of a loop is as follows:

```
for(counter) {
  Statements
}
```

For example, suppose we have a sequence of numbers `1, 3, 5, ..., 99`. I want a new vector where each element is the square of the original numbers. The following loop is a way to do this:

```
myseq <- seq(1,99,by=2)
length(myseq)
myseq.squared <- c()
for (i in 1:50){
  myseq.squared[i] <- myseq[i]^2
}
myseq.squared
```

Note the steps we took in this `for()` loop:

1. We first created an empty vector, `myseq.squared`, to store the squared elements of `myseq`.

2. We made the counter element of the `for()` loop go from 1 to 50, because we wanted to square the i^{th} element of `myseq` for $i = 1, \ldots, 50$.
3. As i went from 1 to 50, the i^{th} element of `myseq` was squared, and became the i^{th} element of `myseq.squared`.

Important note: `for()` loops are notoriously slow in R. Although the above code appeared to run instantaeously, this is because we were performing a very simple operation

(squaring), and doing it only 50 times. R is much faster at carrying out operations on entire vectors, thus when possible it is best to *vectorize*. For example, we could have created the `myseq.squared` object simply by running the command `myseq.squared <- myseq^2`. This would have been a much more efficient way of doing the same task. The following example more clearly illustrates this.

```
X <- rnorm(100000) #Generates 100,000 Normal(0,1) random variables
Y <- rnorm(100000)
#Goal: want to add X and Y.


#Using a for() loop (takes about 18 seconds)
XplusY <- c()
for(i in 1:length(X)){
  XplusY[i] <- X[i] + Y[i]
}


#Vectorizing (takes < 1 second)
XplusY <- X + Y
```

Of course, sometimes vectorizing is impossible and `for()` loops become necessary. Consider the `iris` data we looked at in the last handout. We can use `summary(iris)` to get the mean and quantiles of the four numeric variables, but the standard deviations are not given. We can write a loop that will calculate the standard deviations of each of these variables, and return them in a vector:

```
data(iris)
sd.vec <- c()
for(i in 1:4){
  sd.vec[i] <- sd(iris[,i])
}
names(sd.vec) <- names(iris)[1:4]
sd.vec
```

Note that we can make this loop more flexible: rather than having to manually enter in the indices of the numeric columns, we can add an `if()` statement so that only the standard deviations of the numeric columns will be calculated:

```
sd.vec <- c()
for(i in 1:ncol(iris)){
  if(class(iris[,i])=='numeric') {
    sd.vec[i] <- sd(iris[,i])
    }
  if(class(iris[,i])!='numeric') {
    sd.vec[i] <- NA
    }
}
names(sd.vec) <- names(iris)
sd.vec
```

Exercise 1

Re-consider the birth weight data from the previous handout. Create a vector called `nlevels`, where the i^{th} element of `nlevels` is the number of categories of the i^{th} variable if that variable is categorical, or is "NA" if the i^{th} variable is numeric.

Key to Exercise 1

```
#Note more efficiently written code: only one if() statement required.
nlevels <- rep(NA,ncol(births))
for(i in 1:ncol(births)){
  if(class(births[,i]) == 'factor') {
    nlevels[i] <- length(levels(births[,i]))
  }
names(nlevels) <- names(births)
}
nlevels
```

Loop wrapping: `apply()`

Now that you understand the fundamentals of the `for()` loop, we will move on to discuss some loop wrappers. Loosely defined, a "wrapper" is a function that "wraps" up other tasks (e.g. loops) into a nicely-packaged function. One of these wrappers is the `apply()` function. As its names suggests, `apply()` will *apply* a function over rows or columns of a data set.

Recall the example from above, where we calculated standard deviations for each numeric column of the `iris` data. We did so using a `for()` loop, but we can do this using fewer commands by use of `apply()`. Look at the `?apply` help file. The primary arguments are `X`, a matrix (this can include data frames); `margin`, a vector that gives the dimensions to loop over; and `FUN`, the name of the function you want to *apply* to each row or column. To calculate the standard deviation of the numeric `iris` variables, run the following command:

```
apply(iris[,1:4],2,sd) #Question: why is the second argument "2"?
```

We can also calculate the column means and sums using these commands:

```
apply(iris[,1:4],2,mean)
apply(iris[,1:4],2,sum)
#Note equivalent wrappers for apply(); see ?colMeans, ?colSums.
colMeans(iris[,1:4])
colSums(iris[,1:4])
```

Exercise 2

Using a `for()` loop and either `apply()` or `colMeans()`, write code to calculate the column means of petal and sepal width and length *for each species*. The output of your function should be a 3×4 matrix called `MeanMat` which should look like this:

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
```

Hint: you may find the following functions helpful: `matrix()`, `unique()`, `rownames()`, and `colnames()`. Try to see the relevant help files to understand a function, including the examples at the bottom of the help page.

Key to Exercise 2

```
MeanMat <- matrix(NA,3,4)
species <- unique(iris$Species)
```

```
for(i in 1:3){
  data.subset <- iris[iris$Species==species[i],]
  MeanMat[i,] <- apply(data.subset[,1:4],2,mean)
  #Could also use MeanMat[i,] <- colMeans(data.subset[,1:4])
}
rownames(MeanMat) <- species
colnames(MeanMat) <- names(iris)[1:4]
MeanMat
```

Subsetting data: the by() command

Very often we want to create data summaries by levels of a categorical variable. We can do this by brute force using square brackets, but this gets cumbersome quickly. For example, what if we want to summarize infant birth weights by race, using the birth weight data set? We can do this "the long way" using the following commands:

```
summary(births$bwt[births$race=='asian'])
summary(births$bwt[births$race=='white'])
summary(births$bwt[births$race=='black'])
summary(births$bwt[births$race=='hispanic'])
```

This approach has several drawbacks. First, it requires the programmer to know what all the levels of "race" are, and specify them explicitly. Second, it requires one line of code per unique race. Often there will be many levels, which makes this approach infeasible. We can do this much more efficiently using `by()`:

```
by(births$bwt,births$race,summary)
#Make the output prettier:
simplify2array(by(births$bwt,births$race,summary))
```

Exercise 3

A. Use `by()` to determine how many infants there are of each race.
B. Consider the 3 × 4 matrix `MeanMat` you created in Exercise 2:

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
```

Write code to create this matrix that does *not* use a `for()` loop.

Key to Exercise 3

```
#3a:
by(births,births$race,nrow)
#or:
by(births$age,births$race,length)


#3b:
means <- by(iris[,1:4],iris$Species,colMeans)
meanArray <- simplify2array(means)
MeanMat <- t(meanArray)
MeanMat
```

Audio data: a data analysis

Background

The data `audio.csv` contains data on 117 subjects with mild cognitive impairment (MCI). The subjects were volunteers in a longitudinal clinical trial of a new drug (LG-03812) that was investigated foa ny ability to improve or preserve attention in these patients. The focus of this trial was not the effectiveness of the drug as it pertains to MCI, but but on potential toxicity of the drug on the inner ear (ototoxicity). Subjects were randomized to one of three dosage groups: 0.50 mg/day, 0.25 mg/day, or a matching placebo. The planned duration of treatment was 12 months, after which time the primary effect of treatment on patients' attention would be assessed using the digit symbol substitution test (DSST). Secondary outcomes included other measures of cognitive function, as well as measures of treatment safety including hearing thresholds.

Randomization was stratified by baseline measures of attention (DSST < 35 vs DSST $\geq$ 35). At baseline and each follow-up visit, measures were made to assess the drug's ototoxicity. A tone was sounded near the patient's right and left ear, at frequencies of 250 Hz, 500 Hz, 1000 Hz, 2000 Hz, 3000 Hz, and 4000 Hz. For each frequency, the administrators measured the decibel threshold required for the patient to discern the tone, and recorded the difference from baseline in the decibel threshold.

Research questions

Many of the questions relevant to this data set require more advanced methods for longitudinal data. However, we can simplify this data to answer questions at a level of a first or second course in statistics. Because of the longitudinal nature of these data, nontrivial data cleaning and manipulation needs to take place before the data are even in a form suitable for simple statistical tests. Some research questions are as follows:

1. Was the randomization successful? If not, we would detect an association between DSST threshold (<35 or $\geq$ 35) and dose group. Is there evidence of such an association? (Type of test: χ^2 test of association)
2. Is there a difference in the average number of visits in the study across gender? (Type of test: Independent samples t-test)
3. Is there a difference in the average length of time in the study across gender? (Type of test: Independent samples t-test)
4. Is there a difference across dosage group in the average decibel threshold, averaged across frequency and time? (Type of test: one-way ANOVA)

DSCI: Research Question #1

Clearly, we cannot answer any of these questions with the data as-is. We need to simplify it down to one row per patient.

Start with the original data by using the `read.csv()` command. We then want to reduce it, by getting rid of duplicated patients; create a new variable called DSSTThresh, and carry out the chi-squared test:

```
long <- read.csv('audio.csv')
short <- long[!duplicated(long$Subject),] #Investigate this command
head(short)
short$DSSTThresh <- ifelse(short$DSST < 35, 1, 0)
```

The data are now good to go! Create a stacked bar graph of DSST threshold by dosage group, and carry out the chi-squared test.

DSCI: Research Question #2

For Question 2 we want to add a new variable called `nvisits` that equals the number of visits for each patient. We can use the `by()` command for this:

```
nvisits <- by(long,long$Subject,nrow)
short$nvisits <- nvisits
head(long[,c('Subject','Sex','Race')])
head(short[,c('Subject','Sex','Race','nvisits')])
```

There's a problem with the data now! Can you identify it? To fix it, we need to back up a bit. Investigate the following:

```
short <- long[!duplicated(long$Subject),]
order(short$Subject) #Why is the first element "2"?  Why does "1" not show up until the 48th elem
ent?
short <- short[order(short$Subject),]
head(short) #That's better!
nvisits <- by(long,long$Subject,nrow)
short$nvisits <- nvisits
head(short[,c('Subject','Sex','Race','nvisits')])
short[short$Subject==2001,] #Verify that subject "2001" has 2 visits: yep!
```

We're good! Create a boxplot of number of visits by sex, and carry out the t-test.

DSCI: Research Question #3

Now we need to compuate the length of time on treatment for each individual. Intuitively, we want to compute `short$StopDate-short$StartDate`. But try to run this command: R doesn't like it! Why not?

To compute "time on treatment" we need to change the class of both `StopDate` and `StartDate` to class `Date`. What class are they currently?

To change the class, investigate the following code:

```
#Need to transform to "date" class:
as.Date(short$StopDate) #Why doesn't it work? See ?as.Date
as.Date(short$StartDate,format="%m/%d/%y") #What's wrong with this?
as.Date(short$StartDate,format="%m/%d/%Y") #Good to go!
short$TimeInTrial <- as.Date(short$StopDate,format="%m/%d/%Y") -
                     as.Date(short$StartDate,format="%m/%d/%Y")
head(short$TimeInTrial)
summary(short$TimeInTrial)  #Is of class "difftime"; should probably change to "numeric"
short$TimeInTrial <- as.numeric(short$TimeInTrial)
```

```
summary(short$TimeInTrial)
class(short$TimeInTrial)
```

We're good! Create a boxplot of time in trial by sex, and carry out the t-test.

DSCI: Research Question #4

Answering #4 is slightly more involved than the previous three, and will require use of `apply()`. We want to average across the number of visits, across frequency, for each patient. To do this we need to start from the original "long-form" data set. But there is a hiccup: some patients are missing measurements on some frequencies and visits. We will need to account for this when we calculate means.

```
apply(long[,12:ncol(long)],2,mean)
colMeans(long[,12:ncol(long)]) #Equivalent to above
by(long[,12:ncol(long)],long$Subject,colMeans,na.rm=TRUE) #Note use of na.rm=TRUE.  Need to simpl
ify this:
simplify2array(by(long[,12:ncol(long)],long$Subject,colMeans)) #Need to transpose:
head(t(simplify2array(by(long[,12:ncol(long)],long$Subject,colMeans)))) #Looks good!  Add it to "
short"
short[,12:27] <- t(simplify2array(by(long[,12:ncol(long)],long$Subject,colMeans,na.rm=T)))


#Now want to average across right and left ear
#One way which requires counting, which can be very error prone:
short$RightAvg <- apply(short[,12:19],1,mean)
#Alternative that searches for what we want in the variable names:
grep('R[0-9]',names(short))
short$RightAvg <- apply(short[,grep('R[0-9]',names(short))],1,mean)
short$LeftAvg <- apply(short[,grep('L[0-9]',names(short))],1,mean)
```

A lot went on in the above code! Make sure you understand what each line accomplishes.

Now that the data cleaning is done, create boxplots of average threshold on the right ear by dosage, and do the same for the left ear. We can also carry out the one-way ANOVAs as follows:

```
#Left ear:
left.anova <- aov(LeftAvg~factor(Dose),data=short)
summary(left.anova)
#No significant difference; but if there were we could carry out Tukey's HSD to correct the famil
ywise error rate:
TukeyHSD(left.anova)
```

This is a good chance to talk about the limitations of this simplified approach. We are throwing out a *lot* of information, by averaging not only over the number of visits but also the tone frequencies.