Advanced Data Analysis Class Preparation

Kim Johnson

I am looking forward to our upcoming *Advanced Data Analysis* class! We will jump right into using R on the first day of class, so please make sure you are ready by following the installation instructions below before our first class session and completing the R tutorial below if you have never used R before. This class does not have a prerequisite of R so you may need to learn some of things you will need to do to manage and analyze data in R as you go along. This is the typical scenario for data analysts and very much models the real-world setting. We will have plenty of heavier R users in the class so we will all be able to help each other should we get stuck. If you run into any problems during the tutorial, please email me: kijohnson@wustl.edu or Kyle Pitzer kyleapitzer@wustl.edu

Installing R & RStudio

The class will be conducted using the R statistical programming environment. It is free and available for most modern operating systems. You will need to have access to a laptop where you can install programs. For the class you will want to install R and RStudio.

R can be downloaded from https://cran.r-project.org. Once R has been installed, then install the RStudio IDE, available at https://www.rstudio.com. An IDE is an interactive development environment, which in this case makes R much easier to use.

Open RStudio and make sure that everything has installed correctly. There should be a window open on the left hand side of the RStudio screen that says "Console" in small bold print in the top left corner. R is running within this console window.

Check to see if R is working properly by typing in the following commands (shown in shading) at the R > prompt. Press Enter after typing each command to get the results shown:

```r
2+2
```

```
## [1] 4
```

```r
(4+6)/2
```

```
## [1] 5
```

```r
10^2
```

```
## [1] 100
```

```r
a <- 3
a
```

```
## [1] 3
```

Help with learning R (beyond the tutorial below)

There are ample sites that provide help learning R if you are a beginner. One that I found particularly useful as I was starting to learn R is called Swirl at http://swirlstats.com/. Click on the 'Learn' tab to get started with learning R in R.

R tutorial

If you have used R in the past, you can skip this. If you have never used R before or have very minimal R experience, follow along with this tutorial to gain a few foundational skills:

1. Using R as a calculator
2. Assigning values to variables
3. Data types
4. Vectors and lists
5. Matrices
6. Loading data into R
7. Working with packages

1. Using R as a calculator

R can be used to add, subtract, multiply, divide, exponentiate, and more. In using R as a calculator be sure to remember the *order of operations*: P-E-M-D-A-S or Parentheses, Exponents, Mulitplication & Division, Addition & Subtraction.

For example, note the different results obtained from these examples:

```
10+10/2^2
```

```
## [1] 12.5
```

```
(10+10)/2^2
```

```
## [1] 5
```

```
((10+10)/2)^2
```

```
## [1] 100
```

The R command for square root is sqrt(), like this:

```
sqrt(5+1+3+7)
```

```
## [1] 4
```

You try it!

Solve the equations below using R with the appropriate mathematical symbols so the calculations are done in the proper order:

$$\frac{6+10+9+8+12}{5}$$

$$\frac{(6-9)^2+(10-9)^2+(9-9)^2+(8-9)^2+(12-9)^2}{5-1}$$

$$\sqrt{\frac{(6-9)^2+(10-9)^2+(9-9)^2+(8-9)^2+(12-9)^2}{5-1}}$$

2. Assigning values to variables

Typically in public health, social work, and other fields, we measure or observe characteristics of people or organizations or other entities. These measurements are called *variables*. In R values are assigned variable names using an arrow <-. For example:

```r
#Assign the value of 12 to a variable called months
months <- 12

#Use the variable months in an expression
2*months
```

```
## [1] 24
```

You try it!

Write R commands to calculate the number of hours this class meets over the next 15 weeks.

- Assign the value of 15 to the word days
- Assign the value of 2 to the word hours
- Multiply days and hours

3. Data types in R

- Numeric
- Integer
- Logical (or Boolean)
- Character (or String)

The *numeric* data type in R is the default for numbers with decimal places that we can use in calculations. If we assign a number with values to the right of the decimal to a variable called a, a will be *numeric*. We can use the class command to determine what type of data a variable is.

```r
#Assign the value of 4.5 to a variable called a
a <- 4.5

#Use the class command to determine the data type of a
class(a)
```

```
## [1] "numeric"
```

The *integer* data type is similar to numeric but does NOT have decimal places. When a number is assigned to a variable name the default type is numeric. To change the variable type to integer, use the R command *as.integer*. The *as.integer* command can also be used to truncate numeric data that has decimal places.

```r
#Assign the value of 4 to a variable called b
b <- as.integer(4)

#Use the class command to determine the data type of b
class(b)
```

```
## [1] "integer"
```

```r
#use the as.integer command to truncate the variable a
as.integer(a)
```

```
## [1] 4
```

The *logical* data type includes the values of TRUE and FALSE and is often created when values are compared.

```r
#make a variables c and d with values of 6 and 8
c <- 6
d <- 8

#is c larger than d? store answer in variable e
e = c > d

#print e
e
```

```
## [1] FALSE
```

```r
#determine the data type of e
class(e)
```

```
## [1] "logical"
```

The *character* data type includes letters or words. The paste command puts character variables together (concatenates). The substring command can be used to extract part of a character-type variable.

```r
#make variables fname, mname, lname with values of Kimberly, Jean, Johnson
fname <- "Kimberly"
mname <- "Jean"
```

```r
lname <- "Johnson"

#check the class
class(fname)
```

```
## [1] "character"
```

```r
#assign fname mname lname to a new variable called full and print the variable
full <- paste(fname,mname,lname)
full
```

```
## [1] "Kimberly Jean Johnson"
```

```r
#extract the first three letters of lname
substr(lname, start=1, stop=7)
```

```
## [1] "Johnson"
```

NOTE: Some letters and names are already used by R and will cause some confusion if used as variable names. For example, the uppercase T or F is used by R as shorthand for TRUE or FALSE so are not useful as variable names. When possible, use words and abbreviations that are not common mathematical terms.

You try it!

Assign each part of your name to a character variable. Concatenate your name and assign the full name to a new character variable. Extract characters 2 through 6 from your full name variable.

4. Vectors and lists

A *vector* is a sete of data elements that are the same type (numeric, logical, etc). Each entry in a vector is called a *member* or *component* of the vector. For example, the vector x below has 4 components: 1, 2, 3, 4.

```
x <- c(1,2,3,4)                   #create numeric vector x
x                                 #print vector x
```

```
## [1] 1 2 3 4
```

```
y <- c(T, F, F, T)                #create logical vector y
y                                 #print vector y
```

```
## [1]  TRUE FALSE FALSE  TRUE
```

Vectors can be combined, added to, subtracted from, subsetted, and other operations.

```
x + 3                             #add 3 to each element in the x vector
```

```
## [1] 4 5 6 7
```

```
x + c(1,2,3,4)                    #add 1 to the first element of x, add 2 to the second element, etc
```

```
## [1] 2 4 6 8
```

```
x*5                               #multiply each element of x by 5
```

```
## [1]  5 10 15 20
```

```
x[c(T,F)]                         #remove every other element of vector x
```

```
## [1] 1 3
```

```
(x-1)/5                           #subtract 1 from each element and then divide by 5
```

```
## [1] 0.0 0.2 0.4 0.6
```

If you want to assign the new values of your vector to the vector name of *x*, use the assignment arrow. For example, add three to x and then divide the results by 10:

```
x <- x + 3
x
```

```
## [1] 4 5 6 7
```

```
x <- x/10
x
```

```
## [1] 0.4 0.5 0.6 0.7
```

The results show the original vector x with 3 added to each value and the result of that addition divided by 10. You could also do it in one step:

```
x <- c(1,2,3,4)                   #back to the original vector x
x <- (x+3)/10                     #add 3 and divide by 10
x
```

```
## [1] 0.4 0.5 0.6 0.7
```

Finally, the *list* data type is similar to a vector, but can include entries of different types, like this:

```
a <- c(1, 3, 5, 7)
mylist <- list(fruit="blueberries", age=9.26, mynumbers=a, mygoal=TRUE)
mylist
```

```
## $fruit
## [1] "blueberries"
##
## $age
## [1] 9.26
##
## $mynumbers
## [1] 1 3 5 7
##
## $mygoal
## [1] TRUE
```

```r
class(mylist)
```

```
## [1] "list"
```

You try it!

Write the R commands to create a vector that includes the day, month, year of your birthday. Subtract 2 from each element of your vector, divide each resulting element by 10, and remove the middle number. Use as few steps as you can to get the final answer.

5. Matrices

In addition to the *vector* format, R also uses the *matrix* format to store information. A matrix is information, or data elements, stored in a rectangular format with rows and columns. We can perform operations on matrices like we did with vectors.

The R command for producing a matrix is, surprisingly, matrix. The command has options for specifying the number of rows and columns, like this:

```
z = matrix(c(1, 2, 3, 4, 5, 6), #the data in the matrix
           nrow=2,              #number of rows
           ncol=3,              #number of columns
           byrow=TRUE)          #fill the matrix by rows
z                               #print matrix z
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Say your matrix is the number of red, green, and yellow jelly beans that your son and daughter received in their easter baskets. You can name the columns and rows so you remember which is which by using the dimnames command, like this:

```
dimnames(z)=list(
  c("son","daughter"),          #row names
  c("red", "green","yellow")    #column names
)
z                               #print matrix z
```

```
##          red green yellow
## son        1     2      3
## daughter   4     5      6
```

Now you can find specific pieces of data in your matrix, like the number of red jelly beans your son received.

You try it!

Choose two people you know well (friends, family members) and create a matrix that includes the years you were born, the years you graduated high school, and the years you graduated college. With 3 people and 3 numbers, the matrix will have nine entries. Add names to the columns and rows. Print the matrix.

6. Loading data into R

So far we have just been entering information directly into R. Many of the data sets we use in public health and social work are very large, so entering by hand is not a good solution. R can read data files saved in almost any format. Open a text editor (notepad in Windows, TextEdit for Mac) and type the following favorite Starbucks items with calories. *Note if you use TextEdit, you may need to make the file plain text before saving under the format pull down menu.

drink,calories
iced coffee,0
iced vanilla latte,130.5
cinnamon frappuccino,350
chai latte,220
mocha,290
cappucino,120

Save the file with a file name and location you remember.

Type the following command into R and navigate to the starb data set file you just created when the window opens:

```
starb <- read.table(file.choose(), sep=",", header = TRUE)
```

In the read.table command:

- the *sep=","* tells R the items in the data are separated by a comma
- the *header=TRUE* tells R there is a row at the top with variable names in it

In the top right window of RStudio you should now see an entry called starb. This is your data set. Click on it one time to see the data in your command window.

Use some other commands to examine the data:

```
class(starb)                    #tells you what starb is
```

```
## [1] "data.frame"
```

```
class(starb$drink)              #tells you what class the drink variable in starb is
```

```
## [1] "factor"
```

```
class(starb$calories)           #tells you what class the calories variable in starb is
```

```
## [1] "numeric"
```

```
names(starb)                    #gives you the variable names in the starb data set
```

```
## [1] "drink"    "calories"
```

```
summary(starb)                  #summarizes the starb data set
```

```
##                   drink      calories
##  cappucino           :1   Min.   :  0.0
##  chai latte          :1   1st Qu.:122.6
##  cinnamon frappuccino:1   Median :175.2
##  iced coffee         :1   Mean   :185.1
##  iced vanilla latte  :1   3rd Qu.:272.5
##  mocha               :1   Max.   :350.0
```

You may have noticed some classes you are not yet familiar with:

data frame: The data frame class is similar to the matrix class but is structured with *variables* as columns and *observations* as rows *factor*: The factor class is used when a variable has categories, for example, marital status often is measured using the categories of married, single, divorced, widowed. These are the *categories* of marital status. Marital status is a *categorical* variable! Variables with *categories* are *categorical* and are stored as *factors* in R. Often the class of factor is assigned to character variables by default. This can be changed using an extra option in the read.table command, like this:

```r
starb <- read.table(file.choose(), sep=",", header = TRUE, stringsAsFactors = FALSE)
class(starb$drink)
```

```
## [1] "character"
```

Subsetting is used frequently to isolate specific data in a data frame, so it is useful to try subsetting a few ways:

```r
starb[1,]                                  #show the whole first row
```

```
##         drink calories
## 1 iced coffee        0
```

```r
starb[1,2]                                 #show the first row second column
```

```
## [1] 0
```

```r
starb$drink[starb$calories==0]             #show starb drinks with 0 calories
```

```
## [1] "iced coffee"
```

```r
starb$drink[starb$calories<200]            #show starb drinks with <200 cals
```

```
## [1] "iced coffee"        "iced vanilla latte" "cappucino"
```

You try it!

Create a text file with two variables in it: first name, age in years. Think of at least five people you know and list their names and ages in the file. Put one person per line and separate the name and age with a comma. Be sure to add a header row with the variable names in it, also separated by a comma. Save the data file and bring the data file into R. Use the summarize command to find the mean age of the people in your data.

7. Working with packages

The basic R functions included with R can do a lot, but not everything. Additional functions are included in *packages* developed by researchers and others around the world and contributed to this open-source platform. We will use many of these packages throughout the semester. One that we will use a lot for graphing is *ggplot2*. To use a package, you will have to install it. Use the R command to install ggplot2:

```
install.packages('ggplot2', repos="http://cran.rstudio.com/")
```

```
## Installing package into '/Users/kijohnson/Library/R/3.5/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/0j/tv_4zb197t14n6c9qd14231sb1r76t/T//RtmpSDidcc/downloaded_packages
```

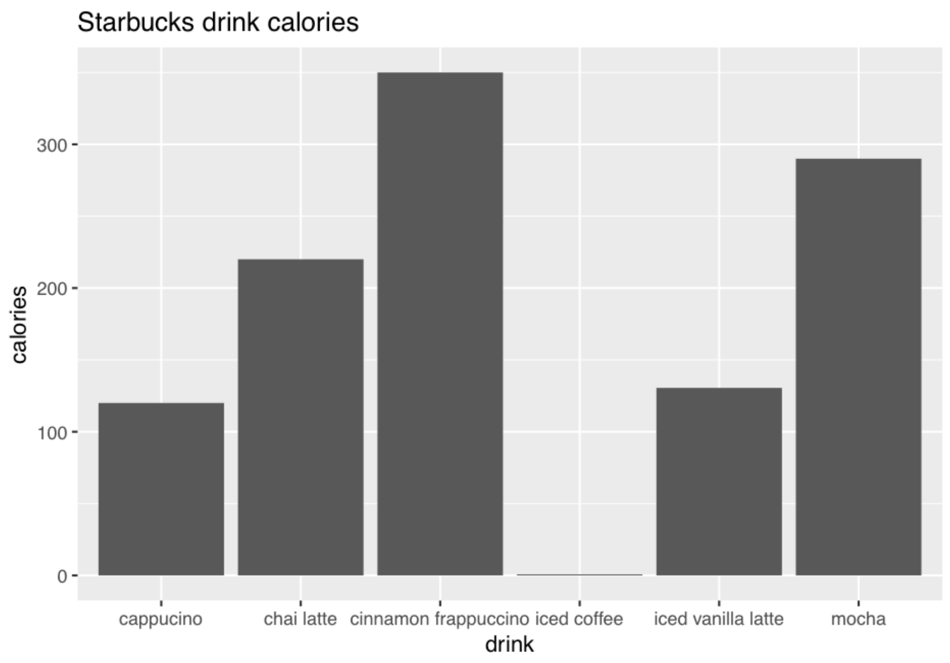
Please note that you only have to install a package ONE TIME. Reinstalling it every time you use R will take you a lot of processing power and slow you down.

Once you install ggplot2, you have to open it. Unlike installing, you will have to open the package *every time* you want to use it. This is similar to other programs you use on your computer. When you first get a new program like R, you only install it once, but you have to open it every time you use it. Use the *library* command to open the package:

```
library(ggplot2)
```

Once a package is open, you can use its functions. Try making a graph of the calories in the Starbucks data with these commands:

```
ggplot(starb, aes(x=drink, y=calories))+
         geom_bar(stat="identity")+
         ggtitle("Starbucks drink calories")
```



The ggplot commands are complicated, we will work on ggplot commands a lot over the semester!

If you want more R before we begin class, check out this awesome set of slides at https://stats.idre.ucla.edu/r/seminars/intro/ and browse through the pages at Quick-R http://www.statmethods.net/. See you soon!