Biostats week 1: Getting started in R

Outline

This packet introduces R software, including:

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. Typing data into R
8. Bringing in data from the web
9. Working with packages
10. Annotating your code

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.

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

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

2. Assigning values to variables

Typically in public health, social work, public policy, 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:

```
# 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 this semester.

- Assign the value of 15 to the word days
- Assign the value of 3 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. Typically these are variables that are *continuous*, which means they can take any value within a range of values. If we assign a number with values to the right of the decimal to a variable called a, R will classify a as *numeric*. We can use the class command to determine what type of data a variable is.

```
# 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 without decimal places 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. Note that truncation is not the same as rounding! Truncation cuts off everything after the decimal place.

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

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

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

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

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

```
# type the variable name to see what is currently saved
# in the variable
a
```

```
## [1] 4.5
```

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

```
# make 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 Jenine, Kinne, Harris
fname <- "Jenine"
mname <- "Kinne"
lname <- "Harris"

# check the class
class(fname)
```

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

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

```
## [1] "Jenine Kinne Harris"
```

```r
# extract the first three letters of lname
substr(lname, start=1, stop=3)
```

```
## [1] "Har"
```

NOTE: Some letters and words are already used by R and will cause some confusion if used as variable names. For example, the uppercase T and F are used by R as shorthand for TRUE and FALSE so are not useful as variable names. In addition, variable names should be words or abbreviations that are easily understood by most people who would read your code.

4. Vectors and lists

A *vector* is a set 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 `numbers` below has 4 components: 1, 2, 3, 4.

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

# print vector numbers
numbers
```

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

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

# print vector logicalVec
logicalVec
```

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

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

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

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

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

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

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

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

```
# subtract 1 from each element and then divide by 5
(numbers - 1)/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 `numbers`, use the assignment arrow. For example, add three to `numbers` and then divide the results by 10:

```
numbers <- numbers + 3
numbers
```

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

```
numbers <- numbers/10
numbers
```

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

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

```
# back to the original vector numbers
numbers <- c(1, 2, 3, 4)
```

```
# add 3 and divide by 10 (remember PEMDAS!)
numbers <- (numbers + 3) / 10
numbers
```

```
## [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:

```
# create a vector called oddNums with members 1, 3, 5, 7
oddNums <- c(1, 3, 5, 7)

# create a list called mylist
# put vector oddNums in the list
mylist <- list(fruit = "blueberries", age = 9.26,
               mynumbers = oddNums, mygoal = TRUE)

# print the list
mylist
```

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

```
# check the data type of the list
class(mylist)
```

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

You try it!

Write the R commands to create a vector that includes three numbers: the number representing the day, the number representing the month, and the number representing the 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:

```
# create a matrix called jellyBeans
jellyBeans <- 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
jellyBeans                                 #print matrix jellyBeans
```

```
##      [,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 oldest and youngest kids 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:

```
# label the rows and columns of numMatrix
dimnames(jellyBeans)=list(
  c("oldest","youngest"),                    #row names
  c("red", "green","yellow")                 #column names
)
jellyBeans                                   #print matrix jellyBeans
```

```
##          red green yellow
## oldest     1     2      3
## youngest   4     5      6
```

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

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, social work, and social policy are very large, so entering data by hand is not a good solution. R can read data files saved in almost any format. Open a text editor (notepad in Windows, cocoa for Mac) and type the following course activities and their percentages.

activity, percentage peer reviews, 9 DataCamp, 6 challenges, 30 book club, 10 statistics in the wild, 10 professionalism, 5 final exam, 30

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

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

```
# bring in data set from computer
courseActivities  <- read.table(file.choose(), sep = ",", header = TRUE)
```

In the read.table command:

- the `sep = ","` tells R the values 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 courseActivities. This is your data set, or `data.frame` in R parlance. Click on it one time to see the data in your command window.

Use some other commands to examine the data:

```
# learn about your data set
class(courseActivities)             #tells you what class courseActivities is
```

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

```
class(courseActivities$activity)    #tells you the class of activity variable
```

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

```
class(courseActivities$percentage)  #tells you the class of the percentage variable
```

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

```
names(courseActivities)             #gives you the variable names in the data frame
```

```
## [1] "activity"   "percentage"
```

```
summary(courseActivities)           #summarizes the contents of the data frame
```

```
##                    activity   percentage
##  book club             :1   Min.   : 5.00
##  challenges            :1   1st Qu.: 7.50
##  DataCamp              :1   Median :10.00
##  final exam            :1   Mean   :14.29
##  peer reviews          :1   3rd Qu.:20.00
##  professionalism       :1   Max.   :30.00
##  statistics in the wild:1
```

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. This is a traditional data set and what you will use for almost all of the data management and analysis you do. For this course and most of your work in R, "data frame" will be used as a synonym for "data set"

`factor`: The factor class is used when a variable is categorical, for example, marital status often is measured using the categories of married, single, divorced, widowed. These are the *categories* of marital status. Variables with *categories* are *categorical* and are stored as *factors* in R.

Often character variables are classified as factors when reading in data frames. This can be changed using the `stringsAsFactors = FALSE` option in the read.table command, like this:

```
# read in data
courseActivities <- read.table(file.choose(), sep=",",
                               header = TRUE, stringsAsFactors = FALSE)

# get data type for activity variable
class(courseActivities$activity)
```

```
##    activity           percentage
##  Length:7           Min.   : 5.00
##  Class :character   1st Qu.: 7.50
##  Mode  :character   Median :10.00
##                     Mean   :14.29
##                     3rd Qu.:20.00
##                     Max.   :30.00
```

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

```
# subsetting from the courseActivities data frame
courseActivities[1,]                                # show the whole first row
```

```
##       activity percentage
## 1 peer reviews          9
```

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

```
## [1] 9
```

```
courseActivities$activity[courseActivities$percentage == 10]  # show activities worth 10%
```

```
## [1] "book club"              "statistics in the wild"
```

```
courseActivities$activity[courseActivities$percentage < 30]   # show activities worth < 30%
```

```
## [1] "peer reviews"           "DataCamp"
## [3] "book club"              "statistics in the wild"
## [5] "professionalism"
```

```
# save the drinks with <30% to a new data frame
# use summary to explore the new data frame
# choose rows with less than 30 to save
# then leave columns blank to save all columns
courseActivitiesLo <- courseActivities[courseActivities$percentage < 30, ]
summary(courseActivitiesLo)
```

```
##    activity           percentage
##  Length:5           Min.   : 5
##  Class :character   1st Qu.: 6
##  Mode  :character   Median : 9
##                     Mean   : 8
##                     3rd Qu.:10
##                     Max.   :10
```

```r
# add a new row to the courseActivitiesLo data set for a new
# optional activity for extra credit called code formatting
# make the new activity worth 2 percent
courseActivitiesLo <- rbind(courseActivitiesLo, c("code formatting", 2))
summary(courseActivitiesLo)
```

```
##    activity          percentage
##  Length:6           Length:6
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

```r
# write the new data frame to a data file so you can
# use it later
# will save in current directory where code is located
write.table(x = courseActivitiesLo,
            file = "courseActivitiesLo.txt",
            sep = ",", col.names = TRUE)
```

7. Typing data directly into R

Instead of creating a data set outside R and bringing it in, you can enter data directly into R using vectors or matrices. To enter the class activity data directly into R:

- Create an activity vector and a percentage vector
- Use the `data.frame` command to combine them into a data frame

Be sure that the vectors are both in the same order. For example, `book club` is the fourth member of the activity vector. The percentage for book club should be the fourth member of the percentage vector, too. This ensures that the vectors will be matched up correctly when combined.

```
# make an activity vector and a percentage vector
activity <- c("peer reviews","DataCamp",
           "challenges","book club",
           "statistics in the wild","professionalism",
           "final exam")
percentage <- c(9, 6, 30, 10, 10, 5, 30)

# combine them into a data frame
classWeights <- data.frame(activity, percentage)

# check it out
summary(classWeights)
```

```
##                    activity   percentage
##  book club             :1   Min.   : 5.00
##  challenges            :1   1st Qu.: 7.50
##  DataCamp              :1   Median :10.00
##  final exam            :1   Mean   :14.29
##  peer reviews          :1   3rd Qu.:20.00
##  professionalism       :1   Max.   :30.00
##  statistics in the wild:1
```

8. Bringing in data from the web

Not all operating systems support opening data directly from an online file. For those that do, you will need to know the URL of the data and what type of file it is (.sav, .xlsx, .csv, etc.). See if yours is one that brings in data by running the following commands:

```
# bring in the data saved at http://tinyurl.com/foblgbt
# it is a csv file
# the data examines happiness among LGBT adults
lgbt <- read.csv('http://tinyurl.com/foblgbt')
summary(lgbt)
```

```
##      happy               lgbt       agecat
##  0      : 178   bisexual   :479   18-24:135
##  1      :1012   gay        :398   25-34:259
##  Refused:   7   lesbian    :277   35-44:157
##                 transgender: 43   45-54:248
##                                   55-64:244
##                                   65-74:120
##                                   75+  : 34
##                          educat                          ethm
##  Bachelor's degree or higher:602   2+ Races, Non-Hispanic: 51
##  High school                :133   Black, Non-Hispanic   : 83
##  Less than high school      : 35   Hispanic              :129
##  Some college               :427   Other, Non-Hispanic   : 33
##                                    White, Non-Hispanic   :901
##
##
##              out                          aware                   accept
##  Hasn't come up:160   All or most of them :667   A lot           :217
##  18            :108   None to some of them:525   None or a little:216
##  20            : 80   Refused             :  5   Refused         : 10
##  16            : 75                              Some            :754
##  22            : 65
##  21            : 62
##  (Other)       :647
##         localaccept             work                 oftvote
##  A lot        :369               :426   Always          :695
##  None at all  : 46   Accepting    :656   Nearly always   :292
##  Only a little:265   Not accepting:111   Part of the time: 82
##  Refused      : 13   Refused      :  4   Refused         :  2
##  Some         :504                       Seldom          :126
##
##
```

If this does not work for your OS, you may need to copy and paste the URL into a browser to download the data and save it to your computer. After downloading, you can open the file using `file.choose` or by entering the file address on your computer into the `read.csv` command.

9. 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. 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 have to install it. Use the `Tools` menu to find 'install packages' or use an R command to install ggplot2:

```
# install the ggplot package for graphing
install.packages('ggplot2')

# Note: sometimes install commands need the location of the package, add repos = "http://cran.rstudio.c
```

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. If you use R commands instead of the Tools menu, add a `#` in front of the `install.packages` command after you use it the first time so that it is treated as a comment and not code to run.

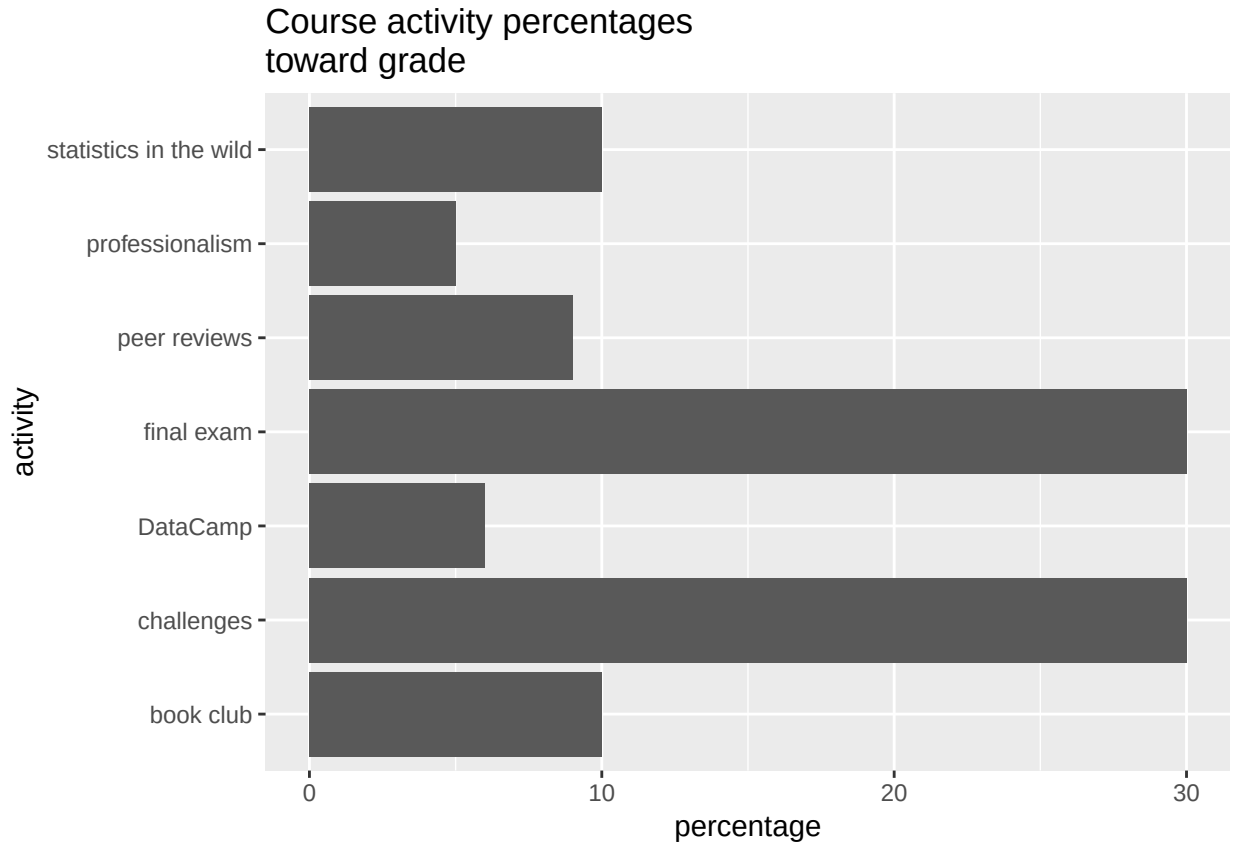
Once you install `ggplot2`, you have to open it. Unlike installing, you will have to open the package *every time* you want to use it. This is similar to other programs you use on your computer. When you first get a new program like R or Microsoft Word, you only install it once, but you have to open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

```
## Warning: package 'ggplot2' was built under R version 3.5.1
```

```
# open the ggplot2 package to use it
library(ggplot2)
```

Once a package is open, you can use its functions. The ggplot2 package has functions for making graphs. Try making a graph of the percentages for activities in this class by typing in (or copy/paste) these commands:

```
# graph the data frame classWeights
# put activity on the x-axis, percentage on the y-axis
# give the graph a title
ggplot(data = classWeights,
       aes(x = activity, y = percentage)) +
         geom_bar(stat = "identity") + coord_flip() +
         ggtitle("Course activity percentages\ntoward grade")
```



Your first R graph (for this class anyway)! The ggplot commands are very flexible and can make some fabulous visualizations, but they are also really complicated....we will learn a lot more about how these commands work later.

10. Annotating your code

Throughout this packet you may have noticed comments above and to the right of the R commands denoted by a leading `#`. These comments are brief statements that describe what your code does. While the best practice for coding is to write code that does not need many comments to explain it, this cannot always be done. So, part of this course will be to work toward both of these goals:

- writing clear code that does not need comments, and
- including clear comments where needed so that anyone can use your code

For example, if I were creating a data set that included the names, ages in years, and favorite color of my relatives, I could:

- make the data file name "jenines_relatives.txt"
- make the name variable called "name"
- make the age variable called "ageYears"
- make the favorite color variable called "faveColor"
- make the categories of the faveColor variable clear as "red", "yellow", "blue", "pink"...

Using clear code and clear comments will help you produce *reproducible research*, which is one of the most important characteristics of good science.

Challenge 1

Examine the distribution of color in a bag of Skittles or M&Ms. First, get a bag of candy and open a new R Script file in RStudio. Pour your candy onto a napkin or display it some other way so that you can easily count the number of candies in each color. Try not to eat your data!

You may choose to complete the coder **or** hacker version of the challenge. They are both worth the same amount of points. The coder version covers the material learned in class. The hacker version adds an additional challenge for you to figure out on your own.

BEFORE THE NEXT CLASS MEETING Upload the file containing your annotated commands into Blackboard as an attachment (see video for how: https://www.youtube.com/watch?v=Pv3cDy9gIp0).

Coder version:

1. Make a vector containing the colors of the candies in your bag (e.g., red, blue, yellow)
2. Make a vector containing the number of candies for each color
3. Create a data frame from the two vectors
4. Print a summary of the data
5. If the summary indicates the color vector is `factor` type, change it to `character` type
6. Print the colors with fewer than the median number of Skittles/M&Ms
7. Print the first row of your data frame
8. Make a bar graph of the data showing number of candies on the y-axis and color on the x-axis
9. Make a new data frame that includes only the colors with more than the median value and print a summary of the new data frame
10. Make a bar graph of the new data frame showing number of candies per color
11. Annotate your commands so someone outside the class could follow your work

Hacker version:

Complete the coder version. Edit the `ggplot` commands for both bar graphs so that the bars are displayed in order from tallest to shortest. Add `theme_minimal()` to change the look of the graph.