Stat 240 - Lab 02

Dr. Lloyd T. Elliott

January 19, 2022

The goal of this document is to walk you through data types, making a few plots, and doing some basic dataset manipulations in *R*. You should submit the solutions to all of the lab questions to crowdmark by Wednesday January 26 at 6pm. You'll have a chance to ask TAs questions about the work during those tutorials. You should run all the code in this document to get going, but you only need to hand in pieces tagged with **Problem** and **Question #, (... points)**.

1 Getting Started

Open *RStudio* (review the previous lectures in case you don't have *RStudio* yet). Great! Now we need to know which directory we are in. Try this command:

```
getwd()
```

When you later use commands to save and load files, this command will let you know what directory you are in. To change the working directory use this command (or browse using the 'Files' tab and click 'More | Set as Working Directory':

```
# on a mac use these slashes
setwd("/Users/lell/some folder")
# or on windows use the other slashes
setwd("c:\\home\\somewhere\\somewhere else")
# on linux it may be like this:
setwd("/home/lell/")
```

Note that the format depends on the operating system. See the result from *getwd()* as an example.

In this section we will go over some basic commands in the console. Try some arithmatic, first some basic math

```
2+2
```

```
[1] 4
```

```
answer = 4-3*5+3*(1+2)
```

If you want the value of 'answer' you just type in the object name or put the whole thing in brackets:

```
(answer = 4-3*5+3*(1+2))
```

```
[1] -2
```

```
answer
```

```
[1] -2
```

Note that you can use ';' and merge several commands on one line. So we could have written something like:

```
2+2;answer = 4-3*5+3*(1+2);answer<2
```

```
[1] 4
```

```
[1] TRUE
```

On that note, run these lines of code to see why I prefer "=" to "<-":

```
w<-2
w=1; (w<-2)
w=1; (w< -2)
```

If you've seen *R* code before, you may already know that both '=' and '<-' are used to assign variables. We will stick with '=' for this course even though the code in the text looks nicer by having '<-'. The reason is that '=' provides a 'local scope' when used inside parenthesis, and *R* allows named arguments. If you use '`lm(data <- data, formula <- z~x+y)`' then *R* will create variables called *data* and *formula* instead of just calling *lm* with parameters *data* and *formula* assigned as indicated.

Now find the volume of a sphere with radius 4:

```
radius = 4
volume = 4/3*pi*radius^3
```

We can do more complex math by making radius a vector of values to see how volume changes with radius:

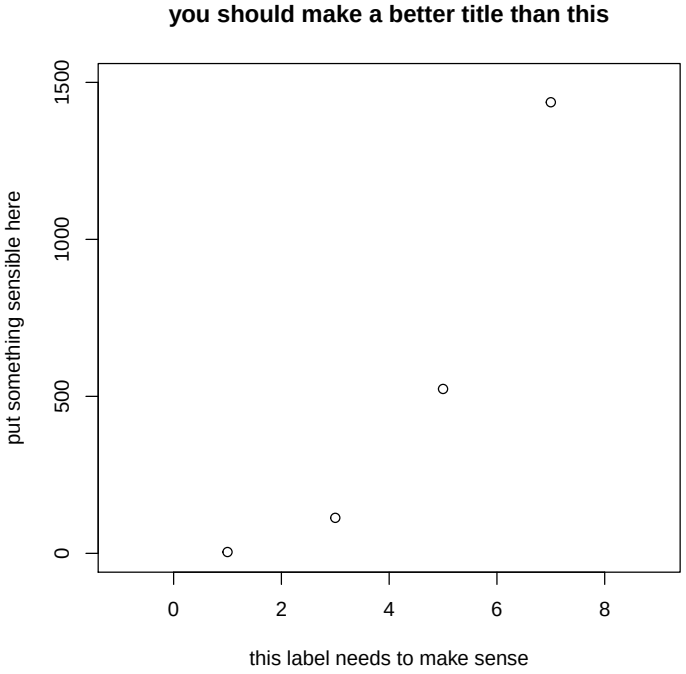
```
radius = c(1,3,5,7)
volume = 4/3*pi*radius^3
```

Plot the results but make sure that you change the plot titles. These are some basic plotting options:

- 'main' puts a title on the top of the plot,
- 'xlab' labels the x axis,
- 'ylab' labels the y axis.
- 'ylim' sets the limits of the y axis
- 'xlim' sets the limits of the x axis

Then the plot command works like this:

```
radius = c(1,3,5,7)
volume = 4/3*pi*radius^3
plot(radius,volume,main="you should make a better title than this",
     xlab="this label needs to make sense",
     ylab="put something sensible here",
     ylim=c(0,1500),xlim=c(-1,9))
```



You can make the plot look a lot nicer if you change the options / add more options:

- 'col' changes the colour. I usually give it a number but you can spell it out
- 'lwd' is the line width. If I change this from the default I usually give it a value of 3.
- 'type' has 4 possible values $\{l, p, n, b\}$
- 'lty' is the type of line (line style). Give it a numeric value.
- 'cex' changes the size of the dots.

Problem 1

Let's try this out but first let's tell R to put 4 plots on one figure. Specifically you should make one plot (using the last plot command from the previous section) with each of the options for the 'type' parameter listed above. Use the *par* command to do this (look it up in the help tab). Each plot should also have a different main title clearly specifying which option you used and what it stands

for. Each plot should have different values of *col* and *lwd*. Make sure that you label the x and y axes appropriately. **Question 1a, (6 points)**: Provide a *.pdf* of this plot. **Question 1b, (1 point)**: provide the *R* code you used to make this plot.

1.1 Nicer plots

Those plots do not look great. Instead we should use a much finer grid of values for 'radius'. You can define a set of numbers with any kind of separation using the 'seq' command. The following are all equivalent to the vector of values that we defined for radius.

```
radius = seq(1,7,by = 2)
radius = seq(from = 1,to = 7,by = 2)
radius = seq(from = 1,to = 7,length = 4)
radius = seq(length = 4,from = 1,to = 7)
```

In other words you can name the inputs and put them in any order or you can put them in the specific order in which inputs are usually entered. This is true for all functions in *R*.

Problem 2

Make a new figure with 1 column and 1 row, then define 'radius') as a vector of length 10,000. These small increments will make the plot look like a smooth curve instead of a choppy set of lines. By now all of your plots should be properly labelled. Full points are only for well made plots. Have you had this error yet? 'Error in plot.new() : figure margins too large' If so you need to make the plot window larger in *RStudio*.

Question 2a, (4 points): Provide a *.pdf* of this new plot. **Question 2b, (1 point)**: provide the *R* code you used to make it. **Question 2c, (1 bonus point)**: make a version of this plot using the *R* package *ggplot2* (`https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf`) and provide a *.pdf* of the plot. You may need to read about installing the *ggplot2* package using the command *install.packages*.

1.2 Adding more lines or points to a plot

Going back to our plot of radius *vs* volume for a sphere, we may also want to include some other information that varies with radius.

When I make one plot after the other it makes a new plot (rendering it in a new frame in the 'Plots' tab—you can switch between the figures in the 'Plots' tab by clicking on the back and forth arrows:

```
#plot about volume
plot(radius,volume,
   main="you should make a better title than this",
   xlab="this label needs to make sense",
   ylab="put something sensible here")
#plot about Area that will replace the Volume plot
plot(radius,pi*radius^2,
```

```
   main="you should make a better title than this",
   xlab="this label needs to make sense",
   ylab="put something sensible here")
```

Instead I want to make a plot and add 'lines' or 'points'

```
#volume
plot(radius,volume,
   main="you should make a better title than this",
   xlab="this label needs to make sense",
   ylab="put something sensible here",type='l')
#Area
lines(1:10,pi*c(1:10)^2,col=2,lty = 2)
#Surface area
points(radius,4*pi*radius^2,col=3)
```

Note that I don't add any axis commands like main or ylim since I'm just adding to the plot. Also notice how the line for area stretched beyond the xlimits. When you originally make the plot you need to set the limits. Sometimes people use the *type="n"* to set up axes for bigger variables, then use points or lines to layer the data in a specific order.

To clarify what is happening we should include a legend. To add a legend give it the *x, y* coordinates of the desired location of the legend, and all of the line type or point arguments that you used when plotting. Labels are entered in the order corresponding to the line arguments *lty, pch, ...* Giving a point type (or line type) an *'NA'* argument omits the line or point. Below I use *pch=c(NA,NA,1)* as arguments to *legend* because the first two legend items were lines only and didn't have point markers plotted.

```
#volume
plot(radius,volume,
     main="you should make a better title than this",
     xlab="this label needs to make sense",
     ylab="put something sensible here",type='l')
#Area
lines(1:10,pi*c(1:10)^2,col=2,lty = 2)
#Surface area
points(radius,4*pi*radius^2,col=3)
legend(x=3,y=1000,col=1:3,pch=c(NA,NA,1),lty = c(1,2,NA),
       c("Volume","Area","Surface Area"))
# Note that I give the pch and lty values of "NA"
# when I don't want a point style or line type.
```

Above I gave the x and y coordinates of where I wanted the legend but you could also use words for x and not give any value for y:

```
legend(x="topleft",col=1:3,pch=c(NA,NA,1),lty = c(1,2,NA),
       c("Volume","Area","Surface Area"))
```

Problem 3

Question 3a, (5 points) Make a plot of 2^x and x^2 and upload the *.pdf*. Include a legend so we know which is which. Both lines should have different colours and

line types. (Hint 1: *R* uses the notation *exp(x)* instead of 'e^x'. Hint 2: you need to make sure that your plot is clear and that both things on the plot are visible so choose a reasonable range of values.) **Question 3b, (1 point)** provide the code used to make this plot.

2 Loading a csv file and dealing with data types

Use the data file "pokemon_2019.csv" provided along with this lab. You may need to use the *setwd* and *getwd* commands at this time in order to load the file. Load the file using *read.table* or *read.csv*.

```
poke = read.csv(file = "pokemon_2019.csv", ... )
```

Now look at the variable named 'poke'. You can show the first few lines using this command:

```
head(poke)
```

The data is in a format that combines rows of text with rows of numbers. That makes it a data frame. Sometimes *R* gets confused about what data type is in a particular column. That also limits what math you can perform and will eventually give you a big headache when things fail. When that time comes try commands like 'as.matrix', 'as.numeric' and the likes. Matrices and vectors only contain numbers, data.frames can contain anything. We'll get to data type and errors related to them later. Let's take a look at another kind of plot that will count up the pokemon types.

```
plot(poke[,"Type_1"], horiz=TRUE,las=2)
```

Note that I rotated the plot using *horiz=TRUE* and rotated the labels using *las=2*. I also extracted the variable that I want by putting its name in as the column delimiter. Sometimes we wish to look at subsets of the data. Recall array indicing using logical vectors that we saw in Lecture 2. Look at the differences between these two commands:

```
poke[,"HP"]>200
poke[poke[,"HP"]>200,"HP"]
```

You can also get the specific pokemon with those high HP values:

```
which(poke[,"HP"]>200) # these are the row numbers
poke[poke[,"HP"]>200,] #These are the full data rows
```

You can combine logical statements by separating them with & or |. Here '[...]' is an array slicing operation.

Problem 4

Question 4a, (2 points): Find the names of the Pokémon which are taller then 2 meters and are Legendary. (Hint: it's usually wise to first extract several columns of data to make sure your code does what you think it does.) Provide

these names with one name per line. **Question 4b, (6 points):** Let's extract two body styles of Pokemon. We'll restrict this question to just pokemon with two body styles: 'head_arms' and 'serpentine_body'. Plot the Attack *vs* Defense of any Pokemon that has those body style (with 'head_arms' on the x axis and 'serpentine_body' on the y axis). Upload a *.pdf* of this plot. Hint: it's usually wise to first extract several columns of data to make sure your code does what you think it does. Here especially as you are using compound criteria.

2.1 Adding data and breaking R

Now let's add a new Pokemon type. Call it your own name and make up some values which are reasonable compared to whatever else you see in the plot. Here's one way to do that by copying a row of data and then replacing values. There are other ways to do this. I will replace all values by 'NA's to avoid making mistakes.

```
pokenew = poke[1,]*NA
```

Change the new pokemon name to your first name or some pokemon hybrid variation thereof in *pokenew*. Change the pokemon type in Type_1 to 'student'.

```
pokenew$Name = "Charmandavar"
pokenew$Type_1 = "student"
```

Note that I could have called on the specific column using

```
pokenew[,"Name"] = "Lloyd" # You can use any name here.
```

but data frames are also special types of lists where you can call specific variables using the $ notation above. Now we want to attach this new row of data to the original 'poke' *data.frame* by binding it as a new row using *rbind*. You could eventually bind together some other variable as a column using *cbind*.

```
pokemonextra = rbind(pokenew,poke)
```

What did the 'rbind' command do? Look at the differences between *data.frames* 'poke' and 'pokenew' Try these commands on those variables: *head*, *dim*, *summary*.

```
dim(poke)
dim(pokenew)
summary(poke)
summary(pokenew)
head(poke)
head(pokenew)
```

At least pokenew has the right columns. Now try remaking the original horizontal plot of pokemon types

```
plot(pokemonextra[,"Type_1"], las=2)
```

The error that you've just encountered is that there are a specific set of known pokemon types and you've just broken that by adding an invalid type to the list. The original *data.frame* 'poke' doesn't know how to deal with the new 'student' type. We actually need to rebuild the list of acceptable values for that variable. That's an advanced concept but we may as well deal with it now since we will run into all sorts of error messages along the way. A categorical variable is called a 'factor' in *R*. The original "poke" was well made but in adding a row we broke it. These diagnostic tools can be reused with other data types. First, I will see if it's still a data.frame. Then, I will check if the Type variable is numeric. Finally I will check if it is still a factor and then fix the problem. It would be especially useful for you to know how to solve this type of problem.

```
is.data.frame(poke)
is.data.frame(pokemonextra)
is.numeric(poke[,"Type_1"])
is.numeric(pokemonextra[,"Type_1"])
is.factor(poke[,"Type_1"])
is.factor(pokemonextra[,"Type_1"])
```

To fix this problem let's look at the first few rows of 'Type_1' in the old and new matrices

```
poke[1:5,"Type_1"]
pokemonextra[1:5,"Type_1"]
```

The original is a factor and it tells us how many levels are in that factor. The second is just a bunch of strings of characters. We need to turn 'pokemonextra' into a factor where we explicitely tell it to use all of the unique values as factor levels.

We can find out the unique levels of the factor in the order in which they appear:

```
unique(poke[,"Type_1"])
```

And converting it into a factor with the right levels:

```
pokemonextra[,"Type_1"] = factor(pokemonextra[,"Type_1"])
pokemonextra[1:5,"Type_1"]
```

On a related note you can convert between data types like this:

```
as.numeric(poke[,"Type_1"])
factor(poke[,"Attack"])
# you don't need to give the input argument 'levels' to factor
# but if you do it will give the levels nicer names.
```

We can now also compare the two *data.frames* by looking at the levels in the factor.

```
levels(pokemonextra[,"Type_1"])
levels(poke[,"Type_1"])
```

Problem 5

Question 5a, (4 points): Remake the original horizontal pokemon plot with your new pokemonextra data frame, except only include pokemon taller than 1 meter and weighing more than 1kg. Hint: If your title is too long use `\n` to split your text into two lines.

3 Make sure you can do this

In future weeks we will need this, make sure you can install packages and load libraries this without error messages. We only need to install a package once. Thereafter we just need to call on the library any time we wish to use it.

```
install.packages("RSQLite")
library(RSQLite)
```

Problem 6

Control flow and functions are a basic aspect of every computer language. In *R* you can use loops with the following syntax: *for (A in B) { C }*. Here, if 'A' is a variable name, and 'B' is a vector and 'C' is an expression, then the expression will be evaluated once for each element of the vector 'B' (and the variable 'A' may be used in the expression 'C' and the value of 'A' will walk through all elements of the vector 'B'). You can define and call functions with the syntax 'A = function(B) { C; return(D); }'. In this code, 'A' is now a function that takes one argument, and 'C' and 'D' are expressions that may refer to 'B'. You can call the function 'A' using code like 'A(7)'. Here's an example:

```
A = function(B) { C = B + 7; return(C) }
A(22)
```

```
[1] 29
```

Does it make sense to you that *A(22)* is 29? What would *A(22)* be if we changed the text *return(C)* to *return(C*2)*? Or to *return(B)*? **Question 6a, (3 points):** Fibonacci numbers. The first two Fibonacci numbers are both 1 ($f_1 = f_2 = 1$). The n-th Fibonnaci number f_n (for $n > 2$) is $f_n = f_{n-1} + f_{n-2}$. Write R code to compute the sum of the first 20 Fibonacci numbers. You may use recursive functions, or you may use Cramer's rule for linear equations. Provide the sum of the first 20 Fibonacci numbers. **Question 6b, (1 point):** Provide the R code you used to compute the previous answer. **Question 6c, (1 bonus point)**: Provide an estimate of the logarithm of the sum of the first one million Fibonacci numbers, in scientific notation with 4 significant figures.