Intro to R – 2019 Summer Workshop – CoN University of Cincinnati

Joshua Lambert, PhD, MS, MS

June 19th, 2019

Welcome!

Welcome to the 1st Annual Intro to R Workshop in the College of Nursing at the University of Cincinnati. You might ask, "Why R?" That is a good question and one that should be answered by the end of this workshop. In short, R offers a free, flexible, cutting edge, and easy to learn programming language. R can be used to run your statistical analysis, visualize and tabulate your data, harness the 10,000 packages to simplify 1000's of lines of code in one command, and create great looking research documents. Learning R is impowering and is a super useful for a career in research or number crunching. In this workshop I will introduce you to R and R studio as well how to install packages, import data, perform simple analysis, and create complex tables all with just a few lines of code.

R and R Studio

First navigate to r-project.org. "R is a **free** software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS." Here you can learn more about the R project and the many statisticians and researchers who maintain its components. You can download base R from the Comprehensive R Archive Network (CRAN). Install base R and go to the next step.

Next navigate to rstuido.com. "RStudio makes R easier to use. It includes a code editor, debugging & visualization tools." RStudio is **free** and is used by everyone from NASA to EBay to Journalist for the Washington Post. R and RStudio is for everyone and when coupled together create a powerful language that can take your research to another level. While Rstuio is a buisness it makes it's money from other more advanced pieces of software that only buisnesses are likely to need. Many of paid software that RStudio has is free to Academics (like us!). To download and install RStudio navigate to https://www.RStudio.com/products/RStudio/download/#download. Here, find your version (windows, mac, or linux) under "Installers for Supported Platforms". If you are on windows, click the "Windows 7+" version. Install RStudio and then you will be ready to go.

Next, open RStudio. You won't need to open base R but you can if you'd like. RStudio does everything (and more) that base R does so you really don't have to use base R.

The last piece of software you will need to install are packages. There are literally more than 10,000 packages for R (and one is mine). They range in their capabilities but can really allow you to do complex and useful things in just a few lines of code. The nice thing is the maintainers of these packages don't charge (aka **free**) to use their pacakges and the CRAN host these for free as well so they are easy to download right within RStudio. More about these later.

Are you confused yet? This is a lot of software right? Lots of moving parts? Heres an hospital room as an analogy to help (maybe). Think of base R as the core components (floor, doors, windows,roof, bed), R Studio as a front end (computer, heart monitor, IV machine) that allows you to easily work with those core components, and the packages as the add on features (nice tv with patient education, fancy scoring mechanisms for diagnosis, robots to help with various actions). You really only need the core components but the front end and add on features really make the experience much better. The best thing is: the core components, front end, and add ons are all **free**.

The Basics of R and RStudio

First, open RStudio. The icon can likely be found on your desktop (or applications bar) or your start menu (applications folder).

RStudio screen

At the very top are menu items (file, edit, code, view, plots, seesion, project, build, tools, and help). These can be useful depending on what your doing but aren't required for this tutorial.

Next, right below the menu at the top is a bar with a white page and green plus, folder icon, and some greyed out icons that look to be save buttons. Also THere are some other icons that we won't be using for right now. This bar is partially meant to open a blank script window (scripts are key for using R).

Saving your work and projects.

Before we jump in to R, lets talk about saving our work and keeping things organized. When you start R, R will attempt to import your workspace from your last session. If you didn't have one, or didn't save it, you won't have any files, or variables pulled in by defult. To assure that you are working with files pertinent to your project you should likely start an R project. You can do this by going "File>New Project…". R projects are nice, because it keeps all the files, and work contained to the project you are working on and doesn't get different project files and variables confused. I like to start a project for every analysis or idea I start.

Working Directory

R always has a designated working directory. This WD is where R will store and find datasets, figures, and tables that you might want to access on your computer's hard drive. You can set your WD by going to "Session>Set Working Directory>Choose Directory …". When you start an R project, your working directory becomes the folder that the project is saved to.

R Scripts Window

Go ahead and create a blank script (select the white page with the green plus on the left and then select "R Script"). This will open up a blank R script where you can write down all your fancy R code! You should now see 4 windows in RStudio. Above the R script window go ahead and click the save icon and save this blank R script to a folder of your choosing.

Working Environment and History Window

On the top right is the Working Environment, History, and Connections Window. We will only be talking about Environment, and History today. When you open RStudio, all the previous datasets, objects, tables, and figures that you made before previously closing out will be stored in your working environment (global envirnment) assuming that you saved it before you closed. Think of this as Word showing (when you first start it) a list of your previous copied items or saved files. This working environment is very useful and allows you to access all the fun things you previously created. Did you spend all that time creating that table and now you want to access it? No problem, just go to your working environment (on the right top of RStudio), find it in the list, and open it. WHen you first start RStudio for the first time, you won't see an items there. Or if you didn't save your last environment, nothing may be there. The History tab shows a list of all the commands you had previously typed into the console (lower left window).

Console Window

The console is where all the action happens. The R *language* is an interpreted language instead of a complied language (like SPSS and SAS). This means things happen one at a time vs. all at once. So, what ever command you type into the R console (lower left corner of RStudio) will run. If many lines of R code are run at one time, they will run one at a time in order. SPSS, and SAS consider the whole code first and then run. This really isn't important here, but I just want you to know that if you type things into the Console and click "enter" R will run it. For instance lets run the following code one line at at time. You can just type these into the console and click enter.

```
1
1+1
2
x=1
x
x=x+1
x
x+1

x==3
x>3
x<3
```

Files, Plots, Packages, Help, and Viewer Window

This window has a lot going on, so try and keep up! Its in the bottom right. The files tab shows a mystic and secret transmitision of next weeks lottery numbers. Just kidding…it shows the files that are currently in your working directory. By default, R will assume you want your working directory to the default. You can set the default under Tools>Global Options>General. The Plots tab will show plots that you create in the R console. The Package tab shows all the packages you currently have installed. Out of the box, only a the core packages are installed. You can install more by clicking the "Install" button under the Packages tab. More about this later. The Viewer tab won't be covered today.

R packages

I want to introduce and install two R packages before we get knee deep in R code (BREATHING INTENSIFIES). I guess this would be a good time to tell you why RStudio is so great. Back in the old days, us R users really only had R scripts and the R console. They were two seperate floating windows and it really wasn't very nice to look at or use for any long duration. When RStudio came along (2010) it began to tidy up a number of the clunkier things about R and make the user experience much more friendly. While RStudio is great, the old R environment can still be used and a lot of what makes RStudio so great is that it alows you to point and click to get R code in your console rather than writing it all out. Many of the buttons, windows, and tabs in RStudio are just nice visualizations of what some R code could tell you in the console. Luckily, RStudio created an Environment that made visuallizing what you were doing in the console more diguestable. **TLDR:The R Console is where it all happens**. Ok, enough of my ranting and now onto R packages.

In the bottom right under the packages tab you will see a "Install" button. Lets click that. This pop up will allow you to search CRAN (remember that from above) for all (10,000+) the publically hosted packages. You must be connected to the internet for this to work. Most of these package names are not descriptive so you usually have to know what you are looking for. For our tutorial we want to install "tableone" and "nhanesA". You can type each of these in the second box (with a space sepearating them) and click "Install".

Make sure to click install. A bunch of crazy stuff will start to happen in the console now. For maybe a while, so in the words of let's get up and stretch our arms, go to the potty, and get a snack.

Ok, now that that our break is over, let's take a second to talk about what those packages just brought us. Still within the packages tab go to the hourglass on the right side. Search for "nhanes". You will see our nhanesA package with it's descritpion and version. Click on the blue "nhanesA" under name. This will take us over to the Help tab, and there we can see all the functions within the nhanesA package as well as a user guide. Lets click on nhanesSearch. You will see that this page shows us an overview of this function, what it does, and how to use it. Usually, at the bottom, you can find examples.

To use this function, we must first load it. You can do this by typing `library(nhanesA)`. You must load the package library prior to using the functions that are within it. Luckily, you only have to do this once per RStudio or R session.

Now that we have the package loaded lets try the following code:

```r
install.packages(nhanesA)
library(nhanesA)
nhSearch=nhanesSearch(search_terms = "bladder")
View(nhSearch)
```

The first line loads the library. The second line runs the `nhanesSearch` function with an argument "bladder" for search_terms and assigns the value from that function to the variable nhSearch. If you got an error, please make sure you are connect to the internet and try again. I could have easily let nhSearch be X, or y, or temp if I wanted to. The third line `View(nhSearch)` will open the value of `nhSearch` in a new tab above the console so you can "View" the contents.

Import Data (and download)

Importing data into R can be done via the R console. R has built in capablity to import `txt`, and `csv` files. If you'd like to import `xlx`, `xlxs`, `sav`, or some other file format, an R pakage will likely need to be used. To import a `csv` file via the R console type:

```r
data<-read.csv("C:/Users/Josh/file.csv",header = FALSE)
```

Here, `C:/Users/Josh/file.csv` is the file's location on your computer. If you'd like to choose where your file comes from, rather than writing it all out you can use the `file.choose()` command instead. For instance:

```r
data<-read.csv(file.choose(),header = FALSE)
```

If neither of these options are your cup of tea, in RStudio you can import data via the Working Environment Window. In the Environment tab, you will see the "Import Dataset" button. Clicking it will show you a list of options to select. I prefrer this method when I'm not sure how to write the code out, because when you take this route, Rstudio automatically wrties the accompanying R code to the console for you. So you can then take that code and paste it into your R Script for later use (if you'd like).

When you are done importing the data, you can view it by typing `View(data)` in the console.

Simple Analysis: The T-Test and Chi-Squared Test

Now that we have some basics completed, lets talk about the fun stuff! STATISTICS! How do I use R to complete a T-Test? Chi-Square Test? Linear regression?

To go through these, I will use an example dataset called `mtcars`. The `mtcars` data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973-74 models). The data is already in R by default. To view it just type: `View(mtcars)`. To learn more about the mtcars data, we can type `help(mtcars)`.

T-Test

As we can see, column 9 is the transmission type. I hypothesis is that the average MPG for automatic transmission is different from the average MPG for manual. Lets use a T-Test to test this hypothesis! In R that is really easy. All we need is the data from the two columns that contain this information. To access it we can type `mtcars$mpg` (or `mtcars[,1]`), and `mtcars$am` (or `mtcars[,9]`). Now, lets look at the help document for the `t.test` function to learn how we can complete a `t.test`. To do this type `help(t.test)` into the console and click enter. Because, or groups (auto and manual) are not contained in two seperate lists of data we must get it into one. This isn't too bad using the `subset` function. To do this type: `subset(mtcars,am==1,select=mpg)` and `subset(mtcars,am==0,select=mpg)`. These give us only the `0` and `1` data for the `mpg` variables. Now onto the T-test.

To conduct the t-test we can run:

```r
auto=subset(mtcars,am==0,select=mpg)
manual=subset(mtcars,am==1,select=mpg)

tt<-t.test(x = auto, y=manual, alternative = "two.sided",mu = 0,paired = FALSE,
           var.equal = FALSE,conf.level = 0.95)
tt
```

What if I wanted something abit different?

```r
tt_v2<-t.test(x = auto, y=manual, alternative = "two.sided",mu = 0,paired = FALSE,
              var.equal = TRUE,conf.level = 0.95)
tt_v2
```

Chi-Square Test

Now that we have some skillz to pay the billz lets see if we can put them to work completeing a Chi-Square test! Lets look at the help page for the Chi-Square test. Well what is the function anyway? If you don't know what the function is in R for a certain test, you can either google it or use the `help.search` functon. Type `help.search("chi square test")` and view a number of results related to what you wrote. Selecting the `stats::chisq.test` result leads us to the right help page. To test if the frequence of engine types is independent of the frequence of transmission types (basically is there a relationship between having an automatic or manual and the type of engine) which are both categorical variables (R calls these factors) we can do a chi-square test. To complete the chi-square test we can run the following code:

```r
help(mtcars)
mtcars$am
mtcars$vs
tab<-table(Engine=mtcars$vs,Transmission=mtcars$am)
tab
chisq.test(tab)
```

An Example from the National Health and Nutrition Examination Survey(NHANES) dataset.

Finally, I'd like to give you a glimpse into the depth and complexity of R. You really have a lot at your fingertips if you choose to learn R. From downloading large datases, creating awesome graphs, making powerpoints, creating word documents, to writing web applications the R language is really powerful and has a great community to boot. For this example we will need two R packages: `nhanesr`, and `tableone`. We downloaded `nhanesr` earlier so lets download `tableone` with the `install.packages('tableone')` command in console. Next load the packages with the following code:

```r
library(nhanesA)
library(tableone)
```

For this example, we will explore the National Health and Nutrition Examination Survey(NHANES) datasets. "The National Health and Nutrition Examination Survey (NHANES) is a program of studies designed to assess the health and nutritional status of adults and children in the United States. The survey is unique in that it combines interviews and physical examinations." More specifically, lets explore the relationship between having the flu and your levels of nutrients in your blood. Now, obviously I didn't pull that out of thin air. I already knew the types of things NHANES captures, but you can find a lot of measurements in NHANES data. I encourage you to search. To identify the datasets where NHANES captures that information lets conduct a search like we did before for "bladder". This time lets look for `flu`, and `vitamin`. We can do that with the following code:

```r
flu_nhSearch=nhanesSearch(search_terms = "flu") #search for the the term flu
View(flu_nhSearch)
vit_nhSearch=nhanesSearchTableNames("VIT",details = TRUE)
View(vit_nhSearch)
```

Merging files by column name

Next, we can actually allow the `nhanesA` package to go and get the files that we would like. For our example we will go and get the `HSQ_D` and `VITAEC_D` files as they have questions about the flu, common cold, and vitamin levels. We can get those files and merge the two files by subject ID (SEQN) using the following code:

```r
flu_data=nhanes("HSQ_D")
vit_data=nhanes("VITAEC_D")

full_data=merge(flu_data,vit_data,by = "SEQN")
```

Tables, Summaries, and Histograms.

Now lets look at the variables we are interested in using the `table`, `summary`, and `hist` functions in R. `HSQ520` is "Did {you/SP} have flu, pneumonia, or ear infections that started during those 30 days?" And `HSQ500` is "Did {you/SP} have a head cold or chest cold that started during those 30 days?" Both `LBXVIA` and `LBXVIE` measure Vitamin A and E levels in the blood of the subjects, measured in ug/dL.

```r
table(full_data$HSQ520) #1 is yes, and 2 is no. Everything else is missing
table(full_data$HSQ500) #1 is yes, and 2 is no. Everything else is missing

#LBXVIA is the micronutrient level of Vitamin A (ug/dL) in the blood higher is more
```

```r
summary(full_data$LBXVIA)
hist(full_data$LBXVIA)

#LBXVIE is the micronutrient level of Vitamin E (ug/dL) in the blood higher is more
summary(full_data$LBXVIE)
hist(full_data$LBXVIE)
```

T-Test

It looks like `HSQ500` has more individuals who actually had the event. Rare events are hard to analyze as we don't have as much information about them. For this reason we will choose the head or chest cold variable (`HSQ500`) over the flu variable (`HSQ520`). A basic research question that could come from this data is: "Is the average Vitamin A levels different for those that had the cold compared to those who did not?" We can do that in R with a few straight forward lines of code. First we need to orgainize our data so we can analyze it.

```r
#creating new variables to use with t.test for vitamin A
cold_vita<-subset(full_data,HSQ500==1,LBXVIA)
nocold_vita<-subset(full_data,HSQ500==2,LBXVIA)
tt_nhanes_vita=t.test(cold_vita,nocold_vita) #t test happens here.

tt_nhanes_vita #this prints out the result.

#what about VitE?
cold_vite<-subset(full_data,HSQ500==1,LBXVIE)
nocold_vite<-subset(full_data,HSQ500==2,LBXVIE)
tt_nhanes_vite=t.test(cold_vite,nocold_vite) #t-test
tt_nhanes_vite
```

Creating a new variable

Next, lets look at doing a simple linear regression and plotting some boxplots. To do this, we first need recognize that our outcome will be the vitamin levels, and our independent variable will be whether they had a cold or not. We will first create a new variable indicating whether the patient had a cold.

```r
full_data$newvar=NA
full_data$newvar[full_data$HSQ500==1]="Cold"
full_data$newvar[full_data$HSQ500==2]="No Cold"
full_data$newvar=factor(full_data$newvar,levels = c("No Cold","Cold"))
table(full_data$newvar)
```

Linear Regression, diagnostic plots, and graphs

Next, we peform the linear regresssion, look at the output, and plot some diagnostic plots. All with just three lines of R code.

```r
#vitamin E
lm_fit_vite<-lm(formula = "LBXVIE~newvar",data=full_data)
summary(lm_fit_vite)
plot(lm_fit_vita)

#vitamin A
```

```
lm_fit_vita<-lm(formula = "LBXVIA~newvar",data=full_data)
summary(lm_fit_vita)
plot(lm_fit_vita)
```

Now for some boxplots:

```
#vitamin E
plot(x = full_data$newvar,y = full_data$LBXVIE, main="Boxplot Vitamin E by Cold Status",
     xlab="", ylab="Vitamin E")
```

We can also plot the scatterplot for VitE and VitA:

```
#vitamin E and A scatterplot
plot(x = full_data$LBXVIA,y = full_data$LBXVIE, main="Scatterplot for Vitamin E and A",
     xlab="Vitamin A", ylab="Vitamin E")
```

Summary tables for your data

Finally, I will discuss how you can make tables using an R package calleed `tableone`. This package is really great and allows you to make tables quickly. The code below will show you how to create, and output your own tables using the `tableone` package.

```
#tableone
library(tableone)

#assure variables are numerical variables in R
full_data[,grep(x = colnames(full_data),pattern = "LBX")]<-
  lapply(full_data[,grep(x = colnames(full_data),pattern = "LBX")],as.numeric)

#create table of VitE and VitA by Cold Status.
tabone=CreateTableOne(vars = c("LBXVIE","LBXVIA"),data = full_data,strata = "newvar")
tabone

#Get all variables that include LBX in their name
vars<-colnames(full_data)[grep(x = colnames(full_data),pattern = "LBX")]
tabone_a=CreateTableOne(vars = vars,data = full_data,strata = "newvar")
tabone_a

#write file to a csv file in your working directory.
write.csv(print(tabone_a),file = "tableone.csv")
```