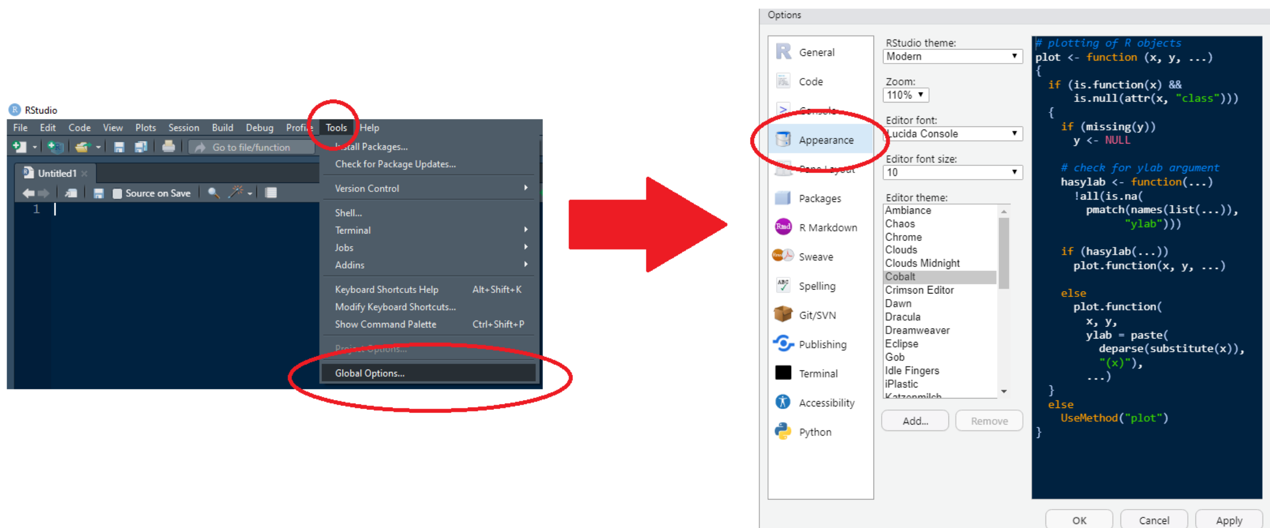
David Abugaber's quick-n-dirty guide to getting started with R

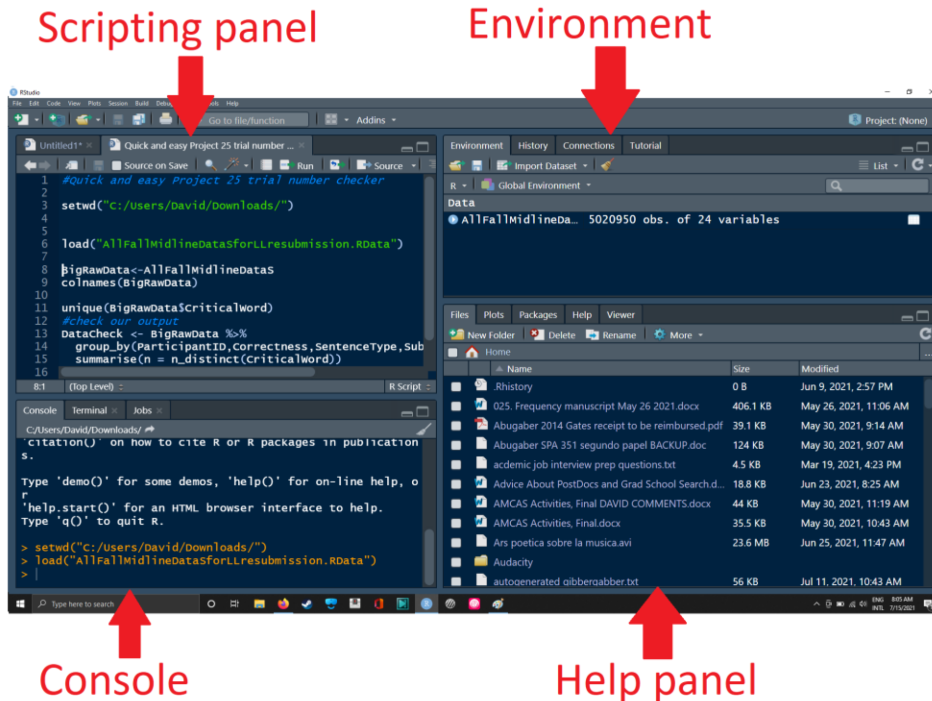
Last edited: 7.15.2021

Module 1: (80% of what you need for reading and running others' code]

1. Installing R and R studio for the first time (do it in this order!)
    a. Download and install base R, the scripting language itself:
        i. http://ftp.ussg.iu.edu/CRAN/
            1. If link above doesn't work, find another mirror:
                a. https://cran.r-project.org/mirrors.html
    b. Download and install RStudio, which makes working with R 100x easier:
        i. https://www.rstudio.com/products/rstudio/download/
    c. After RStudio is installed, change the appearance to an eyeball-friendly color scheme:
        i. Tools -> Global Options -> Appearance
            1. David likes Cobalt, but to each their own :)

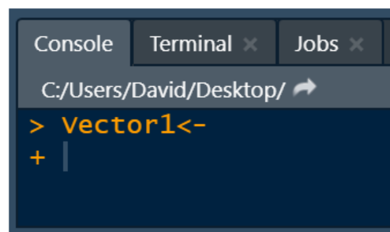


2. Introduction to the RStudio interface
    a. Consists of four panels
        i. Scripting panel (where you'll be actually typing and manipulating code)
        ii. Environment (where your R objects are listed as you load/create them)
        iii. Console (where the output of your commands is actually shown)
        iv. Help panel (where you can browse function help files, see lists of loaded packages, see plots you've generated, and see the files in your working directory)

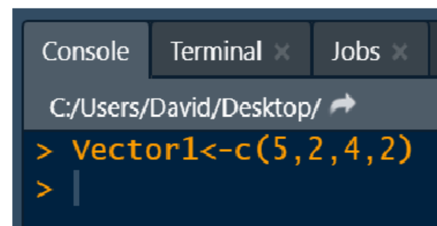


3. How to run a command in R
    a. Write something into your scripting panel
    b. Select what you want to run
        i. If you only want to run part of the line, highlight the exact part you want to run
        ii. If you want to run several lines at one, highlight them all
        iii. Otherwise, easiest method is to click anywhere on the line you want to run
    c. Hit “**Ctrl + ENTER**”
    d. Now some output will "print” to your console log – either some valid output or (if R didn't like what you wrote) an error message.
    e. Careful! If you run an incomplete command, R will be stuck as it waits for more input.
        i. You can tell it's stuck because there's a “+” instead of “>” in last line of console.
        ii. Resolve this by clicking on console and hitting Esc key

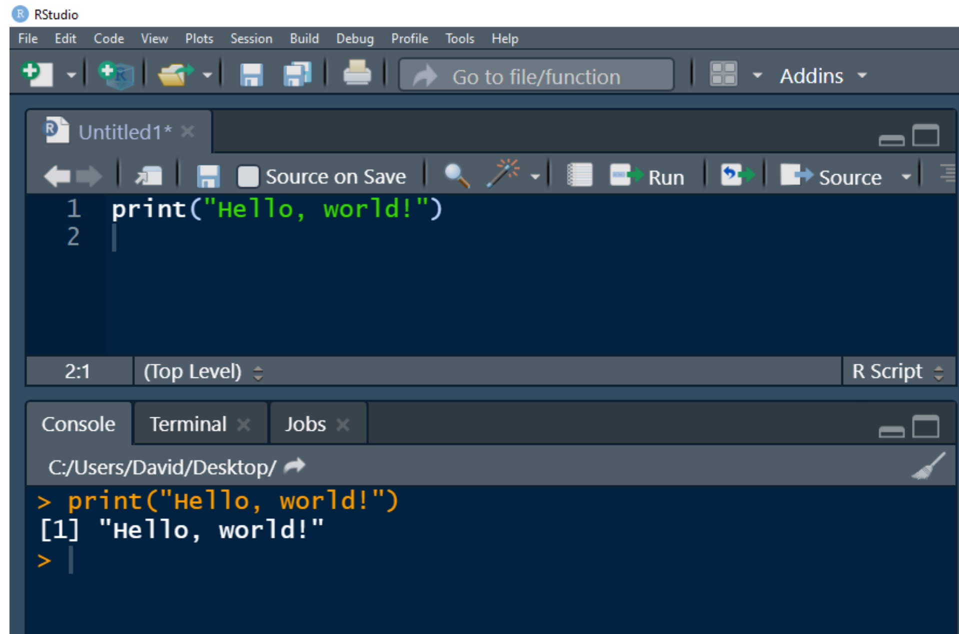
Console is stuck



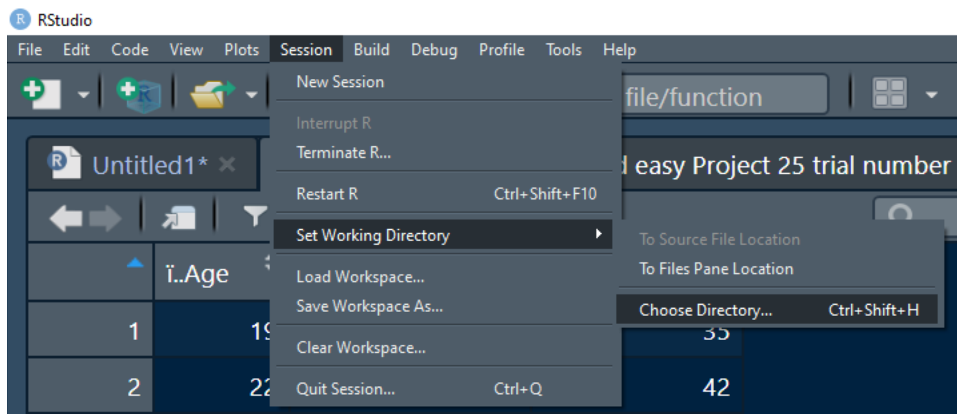
Console is ready for more input



    f. Try running the following command:
        i. **print(“Hello, world!")**



4. Setting your working directory
    a. Your working directory is the folder on your operating system that R actually interfaces with. It will read and write files from this location.
    b. Easiest way to set is via RStudio's Session > Set Working Directory > Choose Directory...



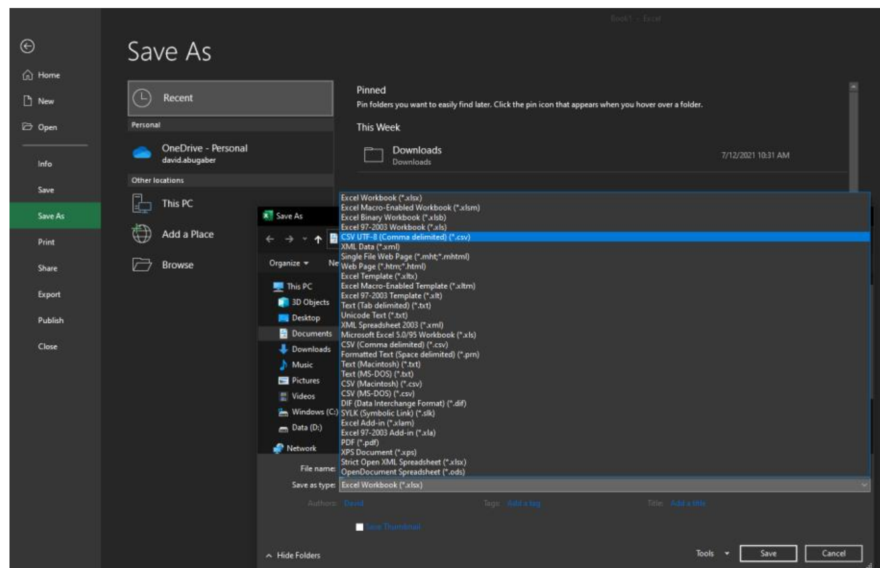
    c. After you've done this, do yourself a favor and copy-and-paste the setwd(...) console output to your script so that if you ever run this script again you can set the working directory with one button press instead of five.

```
setwd("C:/Users/David/Desktop")
```

Copy the "setwd(...)" console output to your script so that you never have to point-and-click again :)

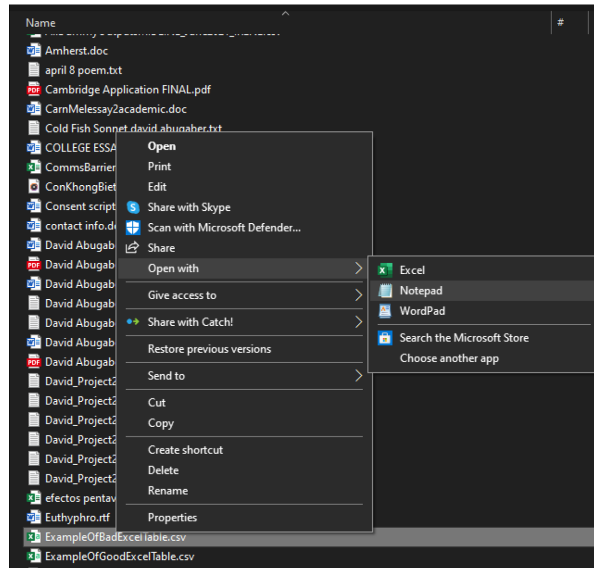
d.

5. Formatting data in Microsoft Excel for R to read
    a. Make sure that...
        i. There are no spaces or punctuation in the headers of your data table
        ii. There are no cells with values in them, that aren't part of the table
        iii. All your data are in a single tab
    b. Best practice is to save as a .CSV (“comma separated values") table
        i. In Excel, navigate to Save as... > Browse > Save as type: “CSV”
            1. Don't choose “CSV UTF-8" because it adds weird characters like “ï..” to your column headers



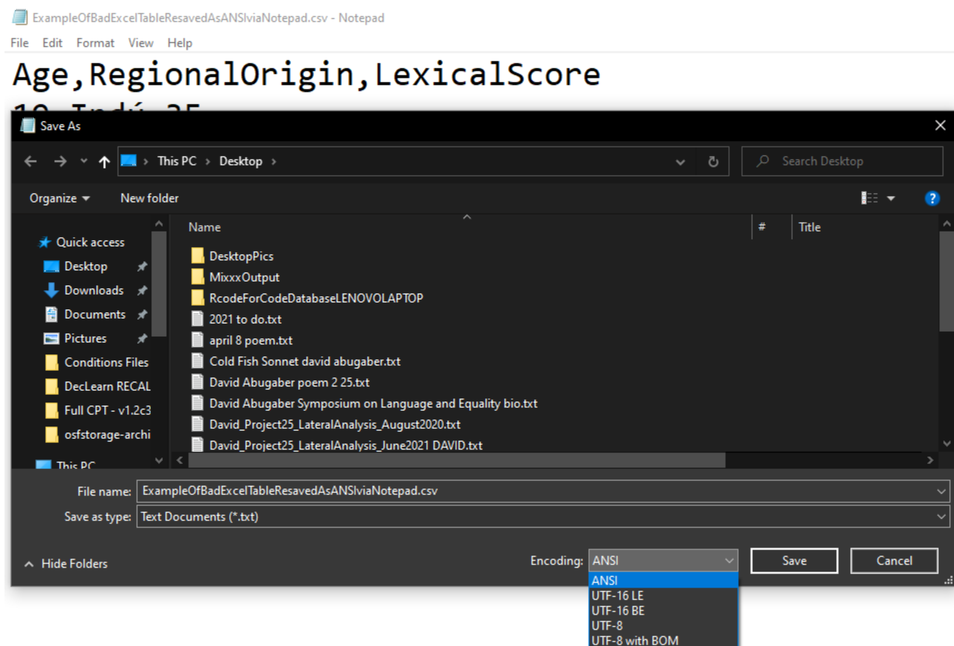
    c. If your data have foreign-language characters (é, á, ñ, etc.) then you will have to take an additional step to save the data using the write character encoding. Otherwise, special characters will print funny (example: ú will print as Ãº).

i. As of 7.15.2021, choosing “ANSI” as the encoding method through Microsoft Excel *still* produces errors, so as a workaround open the data using “Notepad” (or some equivalent simple text editor)



ii. Then, save the file using ANSI encoding

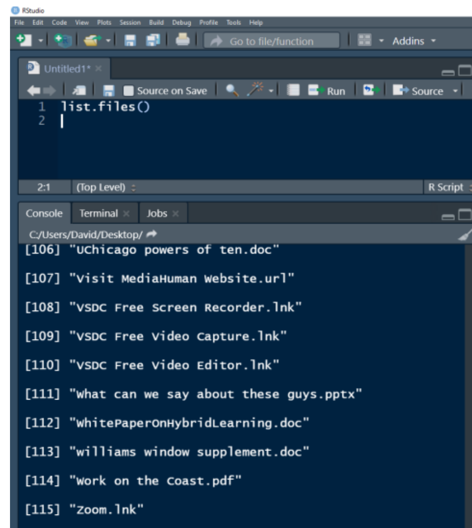
1. Save As > for “Encoding” option choose ANSI



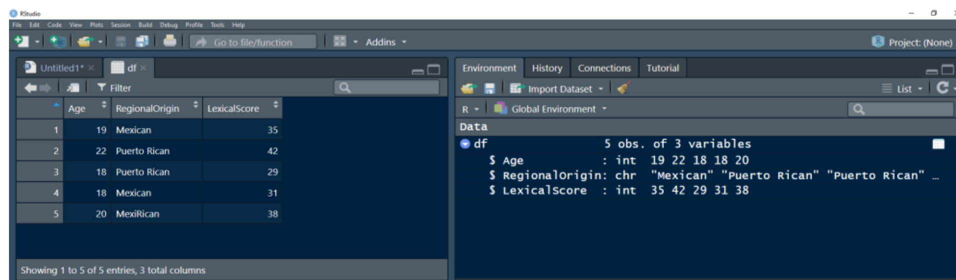
6. Getting a list of files in your directory
   a. R is case sensitive, and if you don't remember the filename with 100% accuracy, then R won't recognize the files you refer to.
   b. Instead of trying to memorize your file names exactly, you can get an exact list of the files in your directory printed into your console log as follows...

i. Run the command "**list.files()**" (without the quotation marks).

1. Remember: write it into the scripting pane, click anywhere on the line you just wrote, and then hit **Ctrl + Enter**



7. Loading your data into R
   a. Now that you remember your filename exactly, you can read in a .CSV of your data by running a command like the following.
      i. ObjectName <- read.csv("FilenameHere.csv")
         1. This is essentially saying "define 'ObjectName' as the output of the function 'read.csv'"
   b. If your data doesn't have headers, you can add a header=FALSE argument. Make sure that FALSE is in all capital leters
      i. ObjectName <- read.csv("FilenameHere.csv", header=FALSE)
8. Viewing your data in R
   a. Easiest way is to click on the object name in your Environment. R will open a new tab that shows the data. Note that unlike in Excel, you can't edit it directly here.



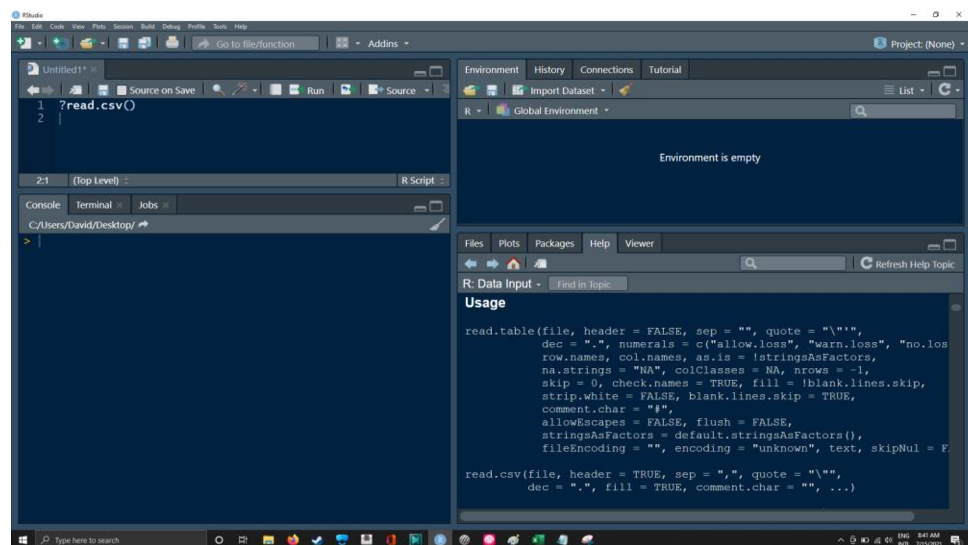
   b. You can also run the following command:
      i. View(DataframeNameHere)
   c. If you just want a quick summary of the columns in the data, you can click on the little arrow-in-a-circle next to the dataframe name in your environment to pull down a list of the variables in the data
   d. For a full description of the contents, you can run the following command:

i. **summary(DataframeNameHere)**

e. To print the first few lines of the dataframe to your console log (easy way to get a glance at the contents without opening a new tab, or without forcing R to load the entire dataframe), you can run the following command:

i. **head(DataframeNameHere)**

9. Looking up the Help fyie for for an R function
    a. Sometimes you might be unsure of how exactly to write/customize the function you want. For instance, in the read.csv() function above, maybe you forgot how to load data without automatically interpreting the first row of the table as a set of column names.
        i. Besides Google, the easiest way to quickly look up a function is by running a line that consists of: **?** (question mark) + **function name** + **()** (dummy parentheses)
            1. Examples:
                a. ?list.files()
                b. ?read.csv()
                c. ?print()
        ii. As illustrated below, the documentation for that function will now appear in the Help panel



10. Basic operations in R
    a. Commenting out
        i. Write “ #” to the left of whatever you want to comment out
    b. Creating an object. You can use either = or <-
        i. a = 5
        ii. a <- 5
    c. Multiplication and division
        i. b <- 2*a
        ii. result.1 <- a*b
        iii. b <- a/2
    d. Boolean logicals (TRUE, FALSE)
        i. 3 == 3 #double-equal-sign means “is this equal to that?”

    ii. 3 != 4 #exclamation+equal sign means “is it *not* equal to that?”
    iii. 3 < 2 #greater than
    iv. !(3 < 2) #NOT greater than

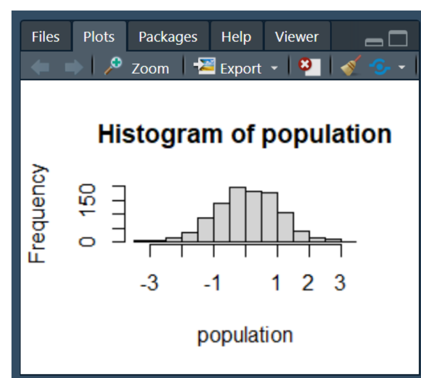
e. Vectors (lists of numbers or text strings)
    i. c1 <- c(4,2,6,3,7,4,1,9,1,6) #the function "c” concatenates
    ii. c2 <- 4,2,6,3,7,4,1,9,1,6 #don't forget the parentheses!
    iii. c3 <- 4 2 6 3 7 4 1 9 1 6 #don't forget the commas!
    iv. d <- c(0,2,0,c1) #you can make lists of lists!

f. Matrices (2x2 tables with raw values and very little to no meta-data)
    i. z <- matrix(1:12, ncol = 3, nrow = 4, byrow = TRUE)
    ii. t(z) #transpose a matrix
    iii. z*z #multiply the matrix by its self

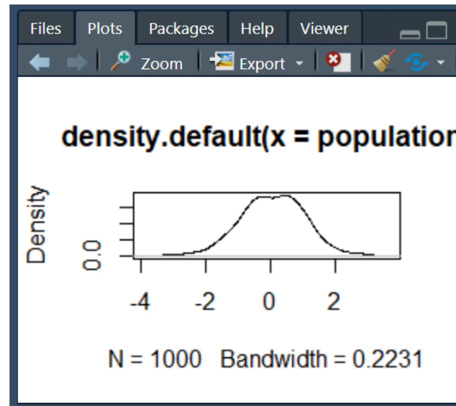
g. Basic built-in functions
    i. mean(c1) #where c1 is the vector we defined earlier
    ii. median(c1)
    iii. var(c1) #gives the variance (square of the standard deviation)
    iv. sd(c1) #gives the standard deviation
    v. sum(c1)

h. Building a normal distribution
    1. First, define the parameters
        a. N <- 1000 #population size
        b. MeanofPop <- 0
        c. SDofPop <- 1
    2. Now build it using the **rnorm()** function!
        a. population<- rnorm(N, MeanofPop, SDofPop)

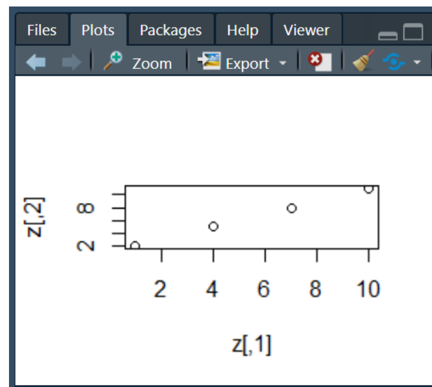
11. How to create a quick and dirty plot
    a. For all of these examples, use the help function -- example: ?hist(), ?plot() -- to get a list of customizable parameters like Title, Axis Labels, etc.
    b. Quick-n-dirty histogram
        i. In this example, the object “population” is a vector (list) of 1,000 numbers with a mean of 0 and a standard deviation of 1. See “building a normal distribution” above.
        ii. hist(population)



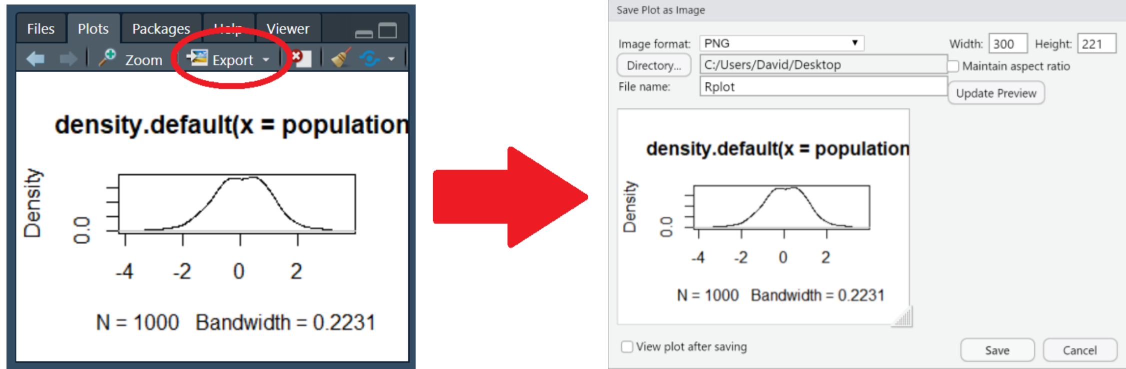
c. Quick-n-dirty kernel density plot
   i. plot(density(population))   #Note the two nested functions: plot() and density()



d. Quick and dirty scatterplot
   i. Uses our example matrix from above (see Matrices section above)
   ii. To rebuild the matrix:
      1. z <- matrix(1:12, ncol = 3, nrow = 4, byrow = TRUE)
   iii. Here is the plotting code:
      1. plot(z)



e. How to save plot
   i. Right-click anywhere on the plot and hit “Save as…”
      1. Re-size the Help/Plot pane before saving to re-size the plot
   ii. To specify more parameters, use the dialogue box under the “Export” menu



12. Congrats! You have completed Module 1, which is designed for follks that are reading and running other people's code but not writing their own. You get 1 gold sticker :)

Module 2

13. Working with dataframes in R
    a. To convert a matrix into a dataframe (necessary for certain functions
        i. DataframeName <- as.data.frame(MatrixName)
    b. To build a dataframe from scratch
        i. In this example, we define three vectors (ID, Group, Value) separately and them put them together into a dataframe
            1. ID<-c(seq(1:10),seq(1:10))
            2. Group<-c(rep("Group1",10),rep("Group2",10))
            3. Value<-c(rnorm(10,mean=10,sd=2),rnorm(10,mean=5,sd=1))
            4. OurDataframe<-data.frame(#this is the dataframe-making function
            5. subjectID = ID,
            6. Condition = Group,
            7. DV = Value)
    c. To add an existing vector as a new variable in a dataframe
        i. DataframeName$NewVariableName <- YourList
            1. Ojo: the number of elements in the list must be the same as the number of rows in your dataframe!
    d. To combine two dataframes
        i. By binding rows together
            1. BigDataframe<-rbind(Dataframe1,Dataframe2)
        ii. By binding columns together
            1. BigDataframe<-cbind(Dataframe1,Dataframe2)
    e. To reformat a dataframe variable so that it's...
        i. A text string
            1. DataframeName$Variable <- as.character(DataframeName$Variable)
        ii. A numeric value
            1. DataframeName$Variable <- as.numeric(DataframeName$Variable)
        iii. A "factor" (necessary for certain analysis functions

1. DataframeName$Variable <- as.function(DataframeName$Variable)

f. How to specify specific values from a dataframe/vector
   i. To specify a variable within a dataframe
      1. DataframeName$VectorName
   ii. To get the Nth value from a vector (list). Replace N with a number as appropriate
      1. VectorName[N]
   iii. To get the Nth value from a variable within a dataframe
      1. DataframeName$VectorName[N]
   iv. To get the Nth value from the Mth row in a dataframe
      1. DataframeName[M,N]
         a. Remember, R's syntax goes "(row, column")
            i. Very counterintuitive because we're used to "(X,Y)"!
   v. To get values that match a given condition
      1. DataframeName[which(DataframeName$Variable == "DesiredValue"),]
         a. Another option for this:
            i. subset(DataframeName,Variable=="DesiredValue")
   vi. To get values that match two (or more) conditions
      1. DataframeName[which(DataframeName$Variable == "DesiredValue" & DataframeName$Variable2 == "DesiredValue2"),]
   vii. To pull out specific columns by name
      1. DataframeName[,c("Variable1","Variable2")]
   viii. The following screenshot gives some simple examples of subsetting data based on row/column number, value, or variable name

```
> df
  Age RegionalOrigin LexicalScore
1  19        Mexican           35
2  22   Puerto Rican           42
3  18   Puerto Rican           29
4  18        Mexican           31
5  20      MexiRican           38
> df$Age
[1] 19 22 18 18 20
> df$Age[3]
[1] 18
> df[2,1]
[1] 22
> df[1,2]
[1] "Mexican"
> df[,2:3]
  RegionalOrigin LexicalScore
1        Mexican           35
2   Puerto Rican           42
3   Puerto Rican           29
4        Mexican           31
5      MexiRican           38
> df[1:2,]
  Age RegionalOrigin LexicalScore
1  19        Mexican           35
2  22   Puerto Rican           42
```

```
> df[which(df$RegionalOrigin=="Mexican"),]
  Age RegionalOrigin LexicalScore
1  19        Mexican           35
4  18        Mexican           31
> df[which(df$Age<20),]
  Age RegionalOrigin LexicalScore
1  19        Mexican           35
3  18   Puerto Rican           29
4  18        Mexican           31
> df[which(df$RegionalOrigin=="Puerto Rican" & df$LexicalScore<30),]
  Age RegionalOrigin LexicalScore
3  18   Puerto Rican           29
> df$Age[which(df$RegionalOrigin=="Puerto Rican" & df$LexicalScore<30)]
[1] 18
> df[,c("Age","LexicalScore")]
  Age LexicalScore
1  19           35
2  22           42
3  18           29
4  18           31
5  20           38
```

14. Quick introduction to loops
   a. How to run a "for" loop
      i. Saves time when you need to repeat an operation several times
      ii. To translate intuitively: "for each element in this list, do this..."
      iii. The letter that you use as the "counter" is arbitrary.
      iv. Here is an example that takes the sum of a vector by iteratively adding each element to a running total
         1. Vector1<-c(5,2,4,2)   #create an example Vector

        2. Loop.Sum<-0 # set up any necessary variables outside the loop
        3. #Start running the "for" loop. This example uses "i" as the counter
        4. for (i in 1:length(Vector1)) {
        5. Loop.Sum <- Vector1[i] + Loop.Sum
        6. } #curly bracket indicates end of loop
    v. Careful! If you run a line that includes the loop-start opening curly bracket "{" without the loop-end closing curly bracket "}," then the console will be stuck because R will be expecting more input:
        1. Note that the console log ends in "+" instead of in ">" like normal
        2. Click anywhere in the console and hit "Esc" to resolve this.

```
Console  Terminal  Jobs
C:/Users/David/Desktop/
> for (i in 1:length(Vector1)) {
+     Loop.Sum <- Vector1[i] + Loop.Sum
+ |
```

    vi. You can hide the inner contents of the "for" loop by clicking on the little arrows to the left of the loop start / end

```
22 #build for loop
23 for (i in 1:length(Vector1)) {
24     Loop.Sum <- Vector1[i] + Loop.Sum
25 }
26
```

```
22 #build for loop
23 for (i in 1:length(Vector1)) {...}
26
27 Loop.Sum
```

b. How to write a "while" loop
    i. Similar to a "for" loop, but runs infinitely until a given condition is met.
    ii. To translate intuitively: "while X is the case, do this"
    iii. In this condition, we create the Fibonacci numbers (where each number is the sum of the previous two numbers) until we get to the number 50
        1. a <- 0
        2. b <- 1
        3. while (b < 50) {
        4. print(b)
        5. step1 <- a + b
        6. a <- b
        7. b <- step1
        8. }
c. You can nest loops within loops. For instance, in this screenshot, the loop that uses the counter "g" is nested in the loop with the counter "f", which is in turn nested in the loop with counter "l."
    i. This means that, for each iteration of the loop with counter g, the loop with counter f will run in full. In turn, each iteration of this loop with counter f involves running the loop with counter g.
        1. Why would you ever want to do this? Imagine you want to run some script separately for each of N participants, separately for Session 1 and Session 2, and separately for Condition X and Condition Y. Writing a

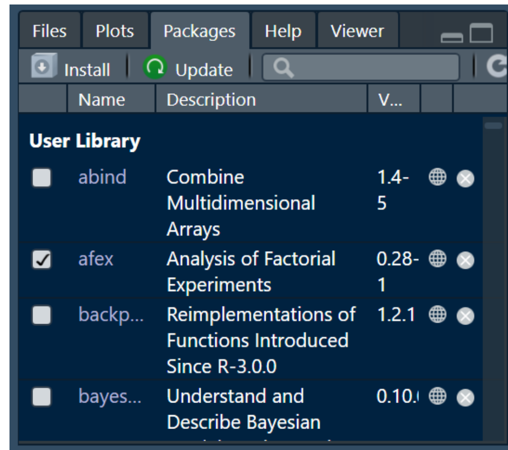
nested loop means that you only have to write the desired script one time instead of Nx2x2 times!

```
101   #from this stage below, the code iteratively takes subsets of the matrix dataset
102   moltenExampleBACKUP1<-moltenExample
103
104   for(l in 1:length(LanguagesForNow)){
105
106     LanguageForNow<-LanguagesForNow[l]
107     moltenExampleBACKUP2<-moltenExampleBACKUP1[which(moltenExampleBACKUP1$EnglishOrSpa
108
109
110     for(f in 1:length(ConditionsForNow)){
111
112       ConditionForNow<-ConditionsForNow[f]
113       moltenExampleBACKUP3<-moltenExampleBACKUP2[which(moltenExampleBACKUP2$condition=
114
115
116       for(g in 1:length(AnalyzeSMorNonSMLoopVector)){
117         AnalyzeSMorNonSM<-AnalyzeSMorNonSMLoopVector[g]
118         moltenExampleBACKUP4<-moltenExampleBACKUP3[which(moltenExampleBACKUP3$Subjecti
119
```

15. How to write a conditional statement
    a. Follows the same general blabla(CONDITION){ INNER CONTENTS } syntax as the loops. Note that the "else{}" statement is entirely optional
        i. if (9 < 10) {
        ii. print("Yes")
        iii. }
        iv. else {
        v. print("No")
        vi. }
    b. If you are only trying to change the values for one variable in a dataframe, you're better off using the ifelse() function. See ?ifelse()
        i. Example syntax. Arguments go: ifelse(CONDITION, IF YES, IF NO)
        ii. DataframeName$IsParticipantAMinor <- ifelse(DataframeName$ParticipantAge<18, "yes!", "no!")
16. How to save data externally
    a. To save any R object, in a format that only R can read:
        i. save(ObjectName,file="DesiredFilename.RData")
        ii. Now can be re-imported using load("DesiredFilename.RData")
    b. To save data as an Excel-friendly table:
        i. write.csv(DataframeName,file="DesiredFilename.csv")
            1. Run ?write.csv() for a list of all the optional parameters
            2. Now can be re-read using read.csv("DesiredFilename.csv")
    c. To write the console log to an external text file...
        i. Run this line of code to start shunting console output to file
            1. sink("DesiredFilename.txt")
        ii. When done, run this line of code to STOP writing console output to file:
            1. sink()
        iii. Ojo: the sink function is a little annoying to work with because it's hard to follow what the code is doing when you can't see the console log in real time. Use judiciously!
17. List of built-in R functions
    a. For a full list see http://www.sr.bham.ac.uk/~ajrs/R/r-function_list.html

i. builtins() # List all built-in functions
ii. ?NA # Help page on handling of missing data values
iii. abs(x) # The absolute value of "x"
iv. append() # Add elements to a vector
v. c(x) # A generic function which combines its arguments
vi. cbind() # Combine vectors by row/column (cf. "paste" in Unix)
vii. diff(x) # Returns suitably lagged and iterated differences
viii. identical() # Test if 2 objects are *exactly* equal
ix. jitter() # Add a small amount of noise to a numeric vector
x. length(x) # Return no. of elements in vector x
xi. paste(x) # Concatenate vectors after converting to character
xii. range(x) # Returns the minimum and maximum of x
xiii. rep(1,5) # Repeat the number 1 five times
xiv. rev(x) # List the elements of "x" in reverse order
xv. seq(1,10,0.4) # Generate a sequence (1 -> 10, spaced by 0.4)
xvi. sign(x) # Returns the signs of the elements of x
xvii. sort(x) # Sort the vector x
xviii. order(x) # list sorted element numbers of x
xix. unique(x) # Remove duplicate entries from vector

18. How to load and install a package
    a. First step is to download the package to your computer.
        i. **install.packages("PackageNameHere")**
            1. You only ever need to do this step once!
    b. Second step is to load the package into your library
        i. **library(PackageNameHere)**
            1. You need to re-do this anytime you restart R
    c. NOTE: sometimes packages can conflict with each other. For instance, if you load Package X, then the behavior of Function Y will be different than if you hadn't loaded that package.
        i. If this is a big source of stress for you, you can install and load the "conflicted" package so that R will automatically tell you if/when such conflicts arise
        ii. Alternately, you can specify, for a given function, which package you want to use to run it
            1. EXAMPLE:
                a. Instead of just "FunctionName(…)", you can run PackageName::FunctionName(…)
    d. You can get a list of the packages that are currently loaded by going to the "Packages" tab under the Help panel



Welcome to Module 4, about **data wrangling**

19. Long format to wide format
    a. To come...
20. Wide format to long format
    a. To come...
21. Matching up values across two dataframes
    a. To come...
22. Getting simple summary statistics with the dplyr package.
    a. To come...

Welcome to Module 5, about more advanced data visualization

Welcome to Module 6, about **statistics**

23. How to run a correlation between two sets of numbers
    a. Install and load the APA package
        i. install.packages(“apa”)
        ii. library(apa)
    b. Run the correlation itself:
        i. apa(cor.test(Dataframe$Variable1,Dataframe$Variable2),format="text")
    c. Output looks like this:
        i. "r(47) = -.45, p = .001"
24. How to format mixed model results into APA-style table

Mixed model results

The results of our mixed model analyses are summarized in Tab

Table #: *Full Results from Mixed Models for Phrase Structure and Subject-Verb Agreeme*

Effects and Interactions	Phrase Structure 300 – 500 ms			600 – 1000 ms			S 300 – 500 m	
	F	*df*	*p*	*F*	*df*	*p*	*F*	*df*
Grammaticality	0.32	1, 302.38	.575	4.86	1, 314.81	.028*	0.26	1, 331.97
Source Attribution	1.63	1, 1147.85	.202	0.50	1, 1200.76	.482	0.88	1, 1013.40
Grammaticality * Source Attribution	0.69	1, 1257.08	.406	5.18	1, 1252.41	.023*	4.63	1, 1224.54

Note. For all analyses, our final model included random intercepts for items and for participants. *

Now, the ANOVA-style table using Sattherwaite's method

```
## Type III Analysis of Variance Table with Satterthwaite's method
##                             Sum Sq Mean Sq NumDF   DenDF F value Pr(>F)
## Correctness.e               52.673  52.673     1  124.44  0.8682 0.3533
## BroadSM.e                   83.134  83.134     1 1074.35  1.3703 0.2420
## Correctness.e:BroadSM.e     12.887  12.887     1 1277.51  0.2124 0.6450
```

F value is rounded to 2 decimal places, WITH a leading zero: example: 0.32
df is actually two values, for example: 1, 302.38 (also rounded to two values)
p value does NOT show the leading zero; rounded to three decimal places: .047.
IF it's less than .05 put an asterisk
IF it's less than .01, put two asterisks

25.
26. More to come!