R Lesson for Julie

Contents

Hi Julie! Welcome to R-Notebooks! Get ready for some **R**ad leaRning!

R-notebooks are different than normal R because they allow you to generate reports that have "chunks" of R-code and will display the result directly below the code. Don't believe me? Press "Run" and click "Run All", all of your R "Chunks" will run! Press the 'Preview' button to the right of the save button! An HTML version of this document will appear.

Okay, let's get some learning in da house!

To start. let's learn about the the **BEST** package for Data importing, cleaning & manipulating. This package is called the "tidyverse" package. It is technically a package of packages. when you load up this package you are loading up 10+ packages - Which are all vital for setting up date for whatever analyses you want to run. Look, maybe don't follow my advice, but it's literally the first thing I do every time I create an R-notebook. it's so vital.

Let's Load it up, and we get to see your first "R-Chunk"!

Load tidyverse

Btw, see where I put that "#" above? That's me creating a header, when you create reports, they turn into large headings so people know what you are doing. Now look below! That grey part is called an "R-chunk"!

you can create it by clicking the "insert" drop down menu and then clicking "R".

```
#Within here is where we enter R code!!!
#See how the "#" now works as commenting things out?!?! That's because it is the same as working in an
 
#So, let's install the "tidyverse" package
#install.packages("tidyverse") # <- this code installs packages, the package you're referencing needs t

library(tidyverse) # <- this loads the package for the session! You need to do this every session!


#If you want to run this chink you can press the "play button" on the upper right hand of the box.
```

Now, Would you look at that?! We're out of R again! Now I can write whatever notes I want. Btw, in this white space you're actually able to use another language called "Markdown". If you learn Markdown you can utilize it to make your reports ***CRAZY PRETTY*** (I just *italicized* and **bolded** "Crazy pretty"" there. Press "Preview" and you'll see it in action!)

Reading in SPSS data through the haven package

Alright, Let's load up your data. I noticed you gave me a **.sav** file (which is is workable, but definitely NOT the ideal file type). To load stastical software data files like these (i.e ones that aren't .csv files) We use a special package called the "haven" package. So we are going to install and load that library hear too.

From the haven package we can use the function *read_spss()* and we'll use this within the r-chunk.

Usually when importinf data you need to designate the filepath… but we're smart and we, instead, use something called "R-projects" (an ".Rproj" file). For this lesson I created an .Rproj file called "Julie.Rproj" this designates a filepath for this R-session. We don't need to use setwd() when we have r-projects! It makes collaborating a breeze!

Alright! Let's do it! Another R-chunk is below.

```
#install.packages("haven")
library(haven)

julies_data <- read_spss("Judicial_Decision_Making_in_Civil_and_Domestic_Courts.sav")

#Hooray! we loaded in your data!
```

Now, look at your "Global Environment". An 'object' called "julies_data" should be there. It will tell you how many observations and variables are in the dataset. Click on it to view it.

A note on the "read_spss()" function

Although R is amazing and can read SPSS data with a breeze… I want you to be aware that it may give you problems from time to time. The haven package uses a format called "labelled" when reading in SPSS files. This is because it keeps the variable name and the associated label name for the variable (Look at the data set, the SPSS labels are retained!). Many functions and packages work well with this variable type… but not all… and it may cause problems in the future, so be aware. Whenever possible, get your data in ".csv"" format. It is the best best *best* format to read in data (also, use the "read_csv()"" function from tidyverse whenever reading in .csv files… NOT read.csv()… There are reasons why!).

tidyverse in action

The tidyverse "verbs"

Okay. This is where the tidyverse gets amazing. Now we can summarize, manipulate, filter, and select variable within our dataset. tidyverse uses a series of "verbs" which are functions that are easy to work with.

The main verbs you will be working with are:

- select()
- filter()
- mutate()
- arrange()
- summarize()

These verbs use a different method for idenitifying variables within your dataset. Remember how you had to use the "$" to work with variables in a dataset? NOT ANYMORE YOU DON'T! these verbs have a very specific way of looking at your data. The first "argument" (arguments = commands within the functions) for all of these verbs is the dataset you are working with. Then, you can just tick away with variable names and so forth.

Let's Go through each of our verbs.

select()

select() "selects" *variables* in a dataset. Let's try it out! You can also choose what **NOT** to select using a "-" before the variable name.

' Here's the format: *select(**data frame**, all, of, the, variables, you, want, -or, -dont, -want, separated, by, commas, or, get, everything())*

Looking at your dataset, I want only certain variables. Let's bring your dataset down to 3 variables. We'll take: "Condition", "GenderParent" & "Q74". Remember the first argument is the dataset we're working with.

```
#Select variables
select_example <- select(julies_data, Condition, GenderParent, Q74)

#De-select variables
deselect_example <- select(julies_data, -Condition, -GenderParent, -Q74)

#Select also is a great way to order variables. Let's bring Q74 to the front and then you can use the "
order_example <- select(julies_data, Q74, everything())
```

Look at your global environment, you have 'objects' called "select_example", "deselect_example", & "order_example" now. It only has the variables listed. See how that worked?

No dollar signs, no quotations. **tidyverse is smart**, it knows exactly what you are trying to do.

There are a lot of shortcuts for this to. I'd really advise you to read on them… Here, I'll help. The R-chunk below is how you get information on packages and functions. you just use a "?" and then write the funcion name.

```
?select # <- An informational sheet will pop up in the help window.
```

```
## starting httpd help server ... done
```

filter()

filter() “filters” the the rows (aka observations) we specifically want.

I noticed you have seperate conditions. Let’s filter and create a few different datasets based on those.

Again, the first argument is your dataframe. Then we have to tell it how to filter. This requires the use of “logical operators” Which I will list below.

Your logical operators are:

- ‘ == Which means “Is exactly equal to”
- ‘ != Which means “Is NOT equal to”
- ‘ > Which means “Is greater than”
- ‘ < Which means “Is less than”
- ‘ >= Which means “Is greater than OR equal to”
- ‘ <= Which means “Is less than OR equal to”
- ‘ is.na() Which means “Contains an NA value”… Missing data is handled differently. I’ll try to show an example below.
- ‘ !is.na() Which means “Does NOT contain an NA value”
- ‘ %% - is called a “modulo”, which means “Has a remainder of” (It’s not used as much, but is actually very userful)

You need these to tell tidyverse how to filter your variables.

Just a note notice how “=” is not a logical operator, yeah, it’s get’s confusing, but never use a single equal sign in a filter function.

Let’s put a few of these of these to the test in the R-chunk below!

```r
#Let's get only condition 1
cond_1 <- filter(julies_data, Condition == 1)

#Let's get everything EXCEPT conition 1
cond_not_1 <- filter(julies_data, Condition != 1)

#Condition greater than 5
cond_great_5 <- filter(julies_data, Condition > 5)

#condition less than OR equal to 8
cond_lesseq_8 <- filter(julies_data, Condition <= 8)

#Want to filter by text values? Easy! Just use quotes! I'll filter on the "DebtRationale" variable for
text_filter <- filter(julies_data, DebtRationale == "they collected it together")

#Let's filter by NA values... The weird one. We'll see how many didn't write a response for the "cONGRU
NAs_only <- filter(julies_data, is.na(cONGRUITY)) # <- the variable goes with the "is.na()" function

#Let's see how many are NOT NA in the "cONGRUITY" variable.
no_NAs <- filter(julies_data, !is.na(cONGRUITY))

#FYI text variables don't take on NA values. so if you want to filter or text variables that don't have
no_miss_text <- filter(julies_data, DebtRationale != "")
all_miss_text <- filter(julies_data, DebtRationale == "")
```

That's the filter function! Just like the select function, there is a lot more… For example, if you use a "&", then you can say things like "filter(data, var > 3 & var < 7)" which will give you values between 3 and 7. Or the "|" is equivalant to "or" which can be used like "filter(data, var == 9 | var == 15) will give you rows that the values are 9 or 15.

mutate()

mutate() "mutates" variables to get you something interesting. You can use this to rename variables, factor variables, do simple math calculations, etc. etc. Here, you'll be using the "=" sign. The "=" sign is telling the mutate function you want to create a new variable.

Here's the basic format for mutate():

- mutate(**dataframe**, NewVariable = OldVariable mutation instructions)

you're creating a new variable and giving the function instructions of what that new variable should be. It's best showed by example.

To start, I am going to create a subset of your data so it'll be easier. I'll call it "small_data".

```
#creating subset of data
small_data <- select(julies_data, Condition, CongruityDad, CongruityMom, JCheerful, JIntelligent)

#To start, this si a great way to rename variables, let's take those J off Jintelligent & JCheerful...
small_data <- mutate(small_data, Intelligent = JIntelligent, Cheerful = JCheerful) # <- btw, you can mu

#Lets create a new variable called "MomDad" where we add the mom and dad congruity scores. we'll keep i
small_data <- mutate(small_data, MomDad = CongruityDad + CongruityMom) #Simple addition

#Now let's get the difference of JCheerful and JIntelligent we'll call it "DiffCI"
small_data <- mutate(small_data, DiffCI = JCheerful - JIntelligent)

#Multiplication and division work too!
small_data <- mutate(small_data, multCI = JCheerful * JIntelligent)
small_data <- mutate(small_data, divCI = JCheerful / JIntelligent)

#Wanna exponentiate something? Just use "**" Let's square something!
small_data <- mutate(small_data, square_cheer = JCheerful**2)

#Then Cube it!
small_data <- mutate(small_data, cube_cheer = JCheerful**3)

#Okay, How about get a z-score? Now we're getting complex! remember, Z-score = (X - mean) / sd. This ca
#We'll get each observations z-score for the intelligent variable
small_data <- mutate(small_data, zintelligent = (JIntelligent - mean(JIntelligent, na.rm = TRUE) / sd(J

#Notice How I used the mean() and sd() functions in the last example. read up on those using ?mean and
```

arrange()

arrange() "arranges" your data (Are you getting the theme? tidyverse NEVER tricks you!). It's not too complex, but you can arrange on as many variables as you want. It may not be super useful for you now, but it is extremely useful when merging datasets together. Sometimes variables need to be sorted on the variable(s) you are merging on so make sure to make that.

Again, arrange() works like your other tidyverse functions. Here's the general format:

- arrange(**dataset**, variables, to, sort, by)

It automatically sorts in ascending order (1,2,3,4,5) But, if you to put it in descending order (5,4,3,2,1) then use desc() within arrange:

- arrange(**dataset**, desc(variables), desc(to), desc(descend), desc(by))

NOTE there is a hierarchy, first it will arrange by the first variable, then the second, then 3rd and so on... So make sure you know what order works best!

Don't forget! If you want to actually want to arrange your dataset you still need to use the assignment operator (<-), otherwise it just shows you the arranged dataset rather than actually arranging it!

Let's take a look!

```
#Let's start by putting the "Condition" variable in descending order
julies_data <- arrange(julies_data, desc(Condition))

#alright, let's arrange it without descending it
julies_data <- arrange(julies_data, Condition)

#Now let's organize by desc(Condition) & ConditionType!
julies_data <- arrange(julies_data, desc(Condition), ConditionType)

#And then let's just sort it so it's ascending on both... Back to normal.
julies_data <- arrange(julies_data, Condition, ConditionType)
```

summarize()

Finally, the summarize() this summarizes variables for you. we can get means, standard deviation, median, minimum, maximum and so on.

There are a lot of summarizing functions, but here are the most essential.

- mean = mean(variable, na.rm = TRUE) <- *unless you have no missing data, you need that na.rm = TRUE argument for this function to work*
- standard deviation = sd(variable, na.rm = TRUE)
- median = median(variable. na.rm = TRUE)
- minimum = min(variable, na.rm = TRUE)
- max = max(variable, na.rm = TRUE)
- Interquartile range = IQR(variable, na.rm = TRUE)
- N of dataset: n() <- because you specified you dataset in the verb alread, this just stays blank.

The general format is:

- summarize(**dataframe**, new_name = summaryfunction())

For this command, you can choose to save them into a new object using the assignment operator(<-) if you want to utilize it later on. Otherwise, if you want to just view the summary, simply don't assign it to an object and it'll show it to you.

This function is extremely great when you run multiple arguments through it, separated by commas. You can view the means, standard deviations, medians, and so forth of a variable. I will show an example of this below.

Let's do this! I'll show it in multiple R-chunks.

```r
#Let's get the mean of our "JCheerful"

summarize(julies_data, mean = mean(JCheerful, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  3.69
```

```r
#Look below! That's the mean!
```

```r
#Okay, the median
summarize(julies_data, median = median(JCheerful, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   median
##    <dbl>
## 1      4
```

```r
#How about the Min, Median, Max, and let's get the range (Max - Min)


summarize(julies_data,
                       Minimum = min(JCheerful, na.rm = TRUE),
                       Median =  median(JCheerful, na.rm = TRUE),
                       Max = max(JCheerful, na.rm = TRUE),
                       Range = max(JCheerful, na.rm = TRUE) - min(JCheerful, na.rm = TRUE)
          )
```

```
## # A tibble: 1 x 4
##   Minimum Median   Max Range
##     <dbl>  <dbl> <dbl> <dbl>
## 1       1      4     6     5
```

```r
#Okay, final one, let's just do everything. including the n() of our dataset!
#You can even save it to an object and display it just by referencing that object. I'll do that here

summarize_julies_data <- summarize(julies_data,
                                               N = n(),
                                               Mean = mean(JCheerful, na.rm = TRUE),
                                               Minimum = min(JCheerful, na.rm = TRUE),
                                               Median =  median(JCheerful, na.rm = TRUE),
                                               Max = max(JCheerful, na.rm = TRUE),
                                               IQR = IQR(JCheerful, na.rm = TRUE),
                                               Range = max(JCheerful, na.rm = TRUE) - min(JCheerful, na
                                  )

#Then just call it whenevsies
summarize_julies_data
```

```
## # A tibble: 1 x 7
##       N  Mean Minimum Median   Max   IQR Range
##   <int> <dbl>   <dbl>  <dbl> <dbl> <dbl> <dbl>
## 1   587  3.69       1      4     6     1     5
```

The "tm" package

I experimented with the tm package a little bit. Let's see if I can help…

Load library

```r
#install.packages("tm")
library(tm)
```

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

## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

select() variables of interest

…Q74?

```r
#Select Q74
Q74 <- select(julies_data, Q74)

#Let's remove NA's
Q74 <- filter(Q74, Q74 != "")
```

Remove punctuations (stringr)

```r
#install.packages("stringr")
library(stringr)

#Remove punctuaction (Good for matching)
Q74 <- str_remove_all(Q74, "[!.'@#$%^&*()_+=,?><\"\\\\]")
```

Change object class for tm functions

```r
#From tm package
tm_doc <- PlainTextDocument(Q74)

#Check class, should be "PlainTextDocument"
class(tm_doc)
```

```
## [1] "PlainTextDocument" "TextDocument"
```

Frequency count

```
#Frequency count
termfreq <- termFreq(tm_doc, control = list(stopwords = TRUE)) #default stopword, removes words like "t
```

Convert back to dataframe

```
#Convert counts to dataframe
termfreq <- as_data_frame(termfreq)
```

turn rownames into columns (easier for analyses)

```
termfreq <- rownames_to_column(termfreq)
```

select()... a special way to rename

```
#FYI select also lets you rename variables also. It's equivalent to mutating and selecting.

termfreq <- select(termfreq, word = "rowname", count = "value")
```

arrange to see which words are used the most

```
#descending by count
termfreq <- arrange(termfreq, desc(count))
```

let's look! using head()

head() allows you to look at the top few rows of data. by default it chooses the top 6, but we can change that!

```
head(termfreq, n = 20) #n = integer, is how many rows you eant displayed!
```

```
## # A tibble: 20 x 2
##    word      count
##    <chr>     <int>
##  1 children    297
##  2 parents     262
##  3 decisions   192
##  4 decision    155
##  5 making      142
##  6 john        129
##  7 kids        125
##  8 make        124
##  9 nicole      119
## 10 say         116
## 11 parent      109
## 12 one          88
## 13 will         67
## 14 think        61
## 15 able         52
## 16 equal        52
## 17 lives        50
## 18 time         49
## 19 together     47
## 20 best         46
```

Wow… there are some useless words here… filter()!

I'm just gonna filter some of those words that seem pointless here in the top 20 (i.e. peoples names)

```
termfreq_filt <- filter(termfreq, word != "john",
                                  word != "nicole"
                        )
```

```
termfreq_filt <- arrange(termfreq_filt, desc(count))
head(termfreq_filt, n = 20)
```

```
## # A tibble: 20 x 2
##    word       count
##    <chr>      <int>
##  1 children     297
##  2 parents      262
##  3 decisions    192
##  4 decision     155
##  5 making       142
##  6 kids         125
##  7 make         124
##  8 say          116
##  9 parent       109
## 10 one           88
## 11 will          67
## 12 think         61
## 13 able          52
## 14 equal         52
## 15 lives         50
## 16 time          49
## 17 together      47
## 18 best          46
## 19 full          46
## 20 childrens     44
```

mutate() & combine duplicates

Notice how children and childrens likely mean the same thing? same with decision and decisions AND parent/parents… lets combine these. and then filter it so we only have one

…This is going to be a little weird but I can explain it to you later.

But I want to bring up "pipes" (%>%). These are, again, part of your tidyverse package. With these, we can chain all of our "verbs" and only reference our dataset once. They are very VERY efficient, useful and make code easier to read. If we have time, I can cover these too. But see if you can fgure out how I am using them to my advantage.

```r
#Take value of "childrens" & "children"
childrens <- termfreq_filt %>%
  filter(word == "childrens") %>%
  select(childrens = "count")

children <- termfreq_filt %>%
  filter(word == "children") %>%
  select(children = "count")

#convert to matrix (for calculations)
childrens <- as.matrix(childrens)
children  <- as.matrix(children)

#add to get value
children_final <- children + childrens

#remove to prevent mess-up
rm(children); rm(childrens)

#Same but for decision/decisions

#Take value of "decision/decisions"
decision <- termfreq_filt %>%
  filter(word == "decision") %>%
  select(decision = "count")

decisions <- termfreq_filt %>%
  filter(word == "decisions") %>%
  select(childrens = "count")

#convert to matrix (for calculations)
decisions <- as.matrix(decision)
decision  <- as.matrix(decisions)

#add to get value
decision_final <- decision + decisions

#remove so to prevent mess-up
rm(decision); rm(decisions)

#Same but for parents/parent

#Take value of "parents/parent"

parent <- termfreq_filt %>%
  filter(word == "parent") %>%
  select(parent = "count")

parents <- termfreq_filt %>%
  filter(word == "parents") %>%
  select(parents = "count")

#convert to matrix (for calculations)
```

```r
parent <- as.matrix(parent)
parents <- as.matrix(parents)

#add to get value
parents_final <- parent + parents

#remove so to prevent mess-up
rm(parent); rm(parents)


#mutate variables so the values are combined and the count is the summation of both words.
termfreq_filt <- termfreq_filt %>%
  mutate(
         count = ifelse(word == "children", children_final, count),
         count = ifelse(word == "decision", decision_final, count),
         count = ifelse(word == "parents",  parents_final,  count)
         ) %>%
#Filter out duplicates
  filter(word != "childrens",
         word != "decisions",
         word != "parent")
```

Check it again

```r
termfreq_filt <- arrange(termfreq_filt, desc(count))
head(termfreq_filt, n = 20)
```

```
## # A tibble: 20 x 2
##    word      count
##    <chr>     <int>
##  1 parents     371
##  2 children    341
##  3 decision    310
##  4 making      142
##  5 kids        125
##  6 make        124
##  7 say         116
##  8 one          88
##  9 will         67
## 10 think        61
## 11 able         52
## 12 equal        52
## 13 lives        50
## 14 time         49
## 15 together     47
## 16 best         46
## 17 full         46
## 18 fair         44
## 19 still        42
## 20 joint        41
```