M8161: Intro To R - Day 1

Mike Gore
8/8/2022

M8161: Intro To R - Day 1
* Why Use R?
* Getting Packages
* Loading R Packages
* Loading R Packages
* Getting Help
* Package Vignettes
* Function Documentation
* Resources From The R Community
* R Basics
* Functions and Objects
* Objects
* Data Structures
* Data Frames/Tibbles
* Vectors
* Matrices
* Indexing
* Reading Data In To R
* Get Raw Data
* Tidy Data
* Tidyverse vs Base R
* Additional Resources
* Subsetting A Tibble/Data Frame
* Base R Subsetting
* Tidyverse Subsetting
* read_*() functions
* Tidyverse Subsetting
* filter() and slice()
* filter() OR slice()
* Adding/Editing Variables
* Long vs Wide Data
* Tidyverse Pipes
* Exercise 1

Why Use R?

R is a crucial tool for all platforms, or languages, that allow you to be experimentative. Some others you might have run across include Python, Command Line/Linux, and the "Bill" program (Office of Field Studies of the Bureau).

* Free and open source software
* Cross platform
* Extensive statistical and graphical capabilities
* Extensive ecosystem of packages (CRAN, Bioconductor, GitHub, Gitlab, Bitbucket, etc.)
* Active community
* Many packages for data wrangling (including tidyverse)
* Many packages for data visualization (including ggplot2, ggmap, leaflet, plotly, etc.)
* Ease to run automatic reports (knitr/Sweave, markdown, etc.)
* Ease to run automatic/semi-automatic data analysis (batch mode)

Opening R / R Studio

You have the option of working in the original R, or R Studio. **I highly recommend you work in R Studio**. R is built on top of R, so it has all the functionality of R, but gives you a more user friendly interface for R and other programming/editing.

Once you open R Studio, you should have a screen that looks like:

If you have trouble loading R Studio or R Studio, check out this video.

R Packages

Much of the power of R comes from packages that are written by R users and made available to the broader R community. There are 19000+ packages that become useful, addressing all areas of science and analysis, including those relevant to biostatistics and data.

Installing R Packages

R packages are stored on a repository on our computer's drive.

```
install.packages("package")
```

Loading R Packages

The R packages you need to use on a very given R session need to be loaded for that session. We'll circle back to R packages later.

```
library(tidverse)
```

```
## [1] "Attaching core tidyverse packages"
## [2] "-------------------------------------- tidyverse 1.2.1 --"
## [3] "v ggplot2 3.2.1     v purrr   0.3.4"
## [4] "v tibble  3.0.1     v dplyr   1.0.0"
## [5] "v tidyr   1.1.0     v stringr 1.4.0"
## [6] "v readr   1.3.1     v forcats 0.5.0"
## [7] "-------------------- Conflicts ------------------------"
## [8] "x dplyr::filter() masks stats::filter()"
## [9] "x dplyr::lag()    masks stats::lag()"
## [10] "Warning message:"
## [11] "package 'ggplot2' was built under R 3.6.1 but R 3.6.2 is 3.6.0 or 3.6.1 and tidyverse 1.3.0 needs 3.6.2\n"
```

Getting Help

Help pop-ups/vignettes can be a lot of assist to get help.

Package Vignettes

Most packages have a vignette that has certain information such as what functions are available in the package, how the functions are used, and examples of functions.

```
vignette("dplyr")
```

Function Documentation

Help pop-ups in R are a lot more cryptic than the information that has to use the function, details about arguments that can be passed to the function, and examples of how to use the function.

```
?read_csv
```

Resources From The R Community

There are a lot of resources that are available online that provide information on how to use R and use R. Personally, I use Google and Stackoverflow a lot. Stackoverflow is a forum for questions and answers on programming and related topics. Frequently in troubleshooting, I find my functions for doing something I want to actually work. In addition, when it comes to learning R/programming, there are many resources out there. I personally like DataCamp, but there are a lot of free resources. Hadley Wickham's book, R for Data Science, is a great resource (available online at https://r4ds.had.co.nz/ and also available in https://www.amazon.com/R-Data-Science-Import-Tidy-Transform/dp/1491910399/ref=sr_1_1?dchild=1&keywords=r+for+data+science&qid=1596195651&sr=8-1 and also on reserve by the Science Library).

R Basics

Functions and Objects

Everything in R is an object stored under the functions name followed by a set of parentheses, which often accept one or more arguments to the function.

```
log(x)
```

```
## [1] 0
```

Here's a R user created (not the assignment operator - a combination of the less that sign/dash) function that stores and lists data1. Note, the input of x1 is assigned to the value of variable "input".

```
check <- x1
```

```
check
```

```
## [1] 3
```

And we can see where this is by calling result.

```
result
```

```
## [1] 3
```

Object Classes

Every object you create in R is assigned to a particular class. Some of the object classes you'll encounter frequently include...

* character
* numeric/integer
* logical

We can use the `class()` function to check the class of any object...

```
class(check)
```

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

Try running the following by and an example of the significance of different classes of data...

```
1 + 1
1 + "1"
```

```
## Error:
## non-numeric argument to binary operator
```

Data Structures

Some common data structures

* Vectors
* Matrix
* Array
* Data Frame/Tibble

Vectors

Vectors are one-dimensional, and are created with the combine function `c()`...

```
name <- c("a")
name
```

```
## [1] "a"
```

```
name <- c("dog", "dog", "cat")
name
```

```
## [1] "dog" "dog" "cat"
```

```
name <- c(1)
name
```

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

```
x <- c(1, 2, 4, 5, 6, 7, 8, 9)
```

Matrices

Matrices are vectors that are wrapped into two dimensions.

```
name_matrix <- matrix(name, ncol = 1)
```

```
name_matrix
```

```
##      [,1]
## [1,] "dog"
## [2,] "dog"
## [3,] "cat"
```

Data Frames/Tibbles

Unlike matrices, data frames are two-dimensional data. Columns, if a column, differences in classes and data frames is that the data within a frame can be different classes, but the data within a column must be of the same class.

```
name_df <- data.frame("name" = c("a","b","c"),
                      "number" = c(1,2,3), "dog", "animal", "orange",
                      "color" = c("r", "b", "y"), "x")
```

```
name_df
```

```
##   name number dog ...1 color x
## 1    a      1 dog  ...  r 1 1
## 2    b      2 dog  ...  b 2 2
## 3    c      3 dog  ...  y 3 3
```

```
name_df$x[1,]; dim(name_df)
```

```
## [1] 1 1
## [1] 3 6
```

Lastly one of the dimensions back just returns all of the entries along that margin...

```
name_df$x[1,]
```

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

```
name_df[1,]
```

```
##   name number dog ...1 color x
## 1    a      1 dog  ...  r 1 1
```

```
name_df[,1]
```

```
## [1] "a" "b" "c"
```

Lastly one of the dimensions back just returns all of the entries along that margin...

```
name_df[c(1,2)]
```

```
##   name number
## 1    a      1
## 2    b      2
## 3    c      3
```

And the 1, 2 can be used as an alternative way to select the contents of and individual column/variable from a data frame or tibble...

```
name_df$name
```

```
## [1] "a" "b" "c"
```

```
name_df[c("name", "color")]
```

```
##   name color
## 1    a     r
## 2    b     b
## 3    c     y
```

Additional Resources

A lot of the time, you need to search online for something or to speak with some of the topics of R, you can try these videos...

* R Objects and Functions (1 hr/vid)
https://www.youtube.com/watch?v=ZdW7dKK8zdc&t=2s&list=PLqbS7AVVErFiWDOAVrhvRz8uZ1ioybkba&index=2
* R Data Structures Vectors (1 min)
https://www.youtube.com/watch?v=1KHVNnG3Fb0&list=PLqbS7AVVErFiWDOAVrhvRz8uZ1ioybkba&index=3
* R Data Structures Matrices (1 min)
https://www.youtube.com/watch?v=2-Ed7PNqqoA&list=PLqbS7AVVErFiWDOAVrhvRz8uZ1ioybkba&index=4
* R Data Structures Lists and Data Frames (1 min)
https://www.youtube.com/watch?v=-0ZJ-Ed7ZuU&list=PLqbS7AVVErFiWDOAVrhvRz8uZ1ioybkba&index=5

Tidyverse vs Base R

The tidverse is a set of packages that have become very popular in a new dialect of the R language. The packages within the tidyverse are written for R, to make it a lot more readable and intuitive, and put an emphasis on working with data in a "tidy" format.

Tidy Data

Tidy data has several characteristics...

* Each variable forms a column
* Each observation forms a row
* Each cell is a single measurement
* Each type of observational unit forms a table

Tidy data is a framework that allows for easy manipulation, modeling and visualization, and it works well with the tidyverse.

Reading Data In To R

Data is probably the first crux of most people's use of R (if not some point involve reading data in from outside of R). It could be a simple file from excel/google sheets, or maybe something you put together by hand or something that has to be wrangled into tidy format. Data you have to R typically should...

* be in a tidy format
* be in the appropriate directory
* be named something descriptive but not too long/short - ideal is summing up data

Get Raw Data

The data we're going to have to work with for a class example and the homework are the directory, M8161_R_Data. Download that folder from Google Drive and place it in your R working directory. Make sure it's unzipped and the folder is named "M8161_R_Data".

The data dictionary we're going to have to use for the R code in class/homework/quizzes/exams is FY14 Jail Jurisdictions.xlsx. This is a file that I created based off of the data dictionary that the Bureau of Justice Statistics provided for their Jail Inmates in Year Ending Year 2014 data. The data dictionary is in tab one of the file. The data set from a comprehensive survey by Schwarzer et al.(2014), and was used for the fortnightly on one of the example homework assignments. The data set is in tab two of the file. The data set is in the Stata format, and I've exported out the dataset to a pseudo-text file (Stata's Code file) but concatenating all of each unit in one line. Which is a fixed format.

We're going to first work with the fixed-unitized dataset, which is inside the homework folder of M8161_R_Data. Before we read in it, let's see how many lines it contains...

```
wc -l "./M8161_R_Data/Homework Data/Fortnightly Data/Fortnightly Data Fixed Unitized.txt"
```

```
## 1000 ./M8161_R_Data/Homework Data/Fortnightly Data/Fortnightly Data Fixed Unitized.txt
```

Does it look like there's a header? Let's use the `head` command to see what the first lines look like...

```
head -n 10 "./M8161_R_Data/Homework Data/Fortnightly Data/Fortnightly Data Fixed Unitized.txt"
```

```
## 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000002 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000003 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000004 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000005 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000006 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000007 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000008 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000009 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000010 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
```

Does it look like there's a header? Let's use the `head` command to see what the first lines look like...

```
cat "./M8161_R_Data/Homework Data/Fortnightly Data/Fortnightly Data Fixed Unitized.txt"
```

```
## 0000000001 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000002 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
## 0000000003 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
##
```

```
head(raw_count_sub_long)
```

```
## # A tibble: 6 × 3
##   Group   name    value
##   <chr>   <chr>   <dbl>
## 1 DA00006 Otu0001     0
## 2 DA00006 Otu0002     1
## 3 DA00006 Otu0003   323
## 4 DA00006 Otu0004   539
## 5 DA00006 Otu0005     1
## 6 DA00006 Otu0006     0
```

OR

```
raw_count_sub_long <- raw_count_sub %>%
  pivot_longer(cols = starts_with("Otu"))
```

```
head(raw_count_sub_long)
```

```
## # A tibble: 6 × 3
##   Group   name    value
##   <chr>   <chr>   <dbl>
## 1 DA00006 Otu0001     0
## 2 DA00006 Otu0002     1
## 3 DA00006 Otu0003   323
## 4 DA00006 Otu0004   539
## 5 DA00006 Otu0005     1
## 6 DA00006 Otu0006     0
```

And the same thing, but getting some more informative names for the new columns…

```
raw_count_sub_long <- raw_count_sub %>%
  pivot_longer(cols = !Group, names_to = "OTU", values_to = "Count")
```

```
head(raw_count_sub_long)
```

```
## # A tibble: 6 × 3
##   Group   OTU     Count
##   <chr>   <chr>   <dbl>
## 1 DA00006 Otu0001     0
## 2 DA00006 Otu0002     1
## 3 DA00006 Otu0003   323
## 4 DA00006 Otu0004   539
## 5 DA00006 Otu0005     1
## 6 DA00006 Otu0006     0
```

Changing Variable Names

`rename()`

Group is maybe not the best name for that original column. We can rename it…

```
raw_count_sub_long <- raw_count_sub_long %>%
  rename(Sample = Group)
```

```
head(raw_count_sub_long)
```

```
## # A tibble: 6 × 3
##   Sample  OTU     Count
##   <chr>   <chr>   <dbl>
## 1 DA00006 Otu0001     0
## 2 DA00006 Otu0002     1
## 3 DA00006 Otu0003   323
## 4 DA00006 Otu0004   539
## 5 DA00006 Otu0005     1
## 6 DA00006 Otu0006     0
```

Grouping Data

`group_by()`

Now we'll group the tibble by Sample and calculate the total reads for each sample. We'll add these as a new column.

```
raw_count_sub_long <- raw_count_sub_long %>%
  group_by(Sample) %>%
  mutate("Total_Reads" = sum(Count)) %>%
  ungroup()
```

```
head(raw_count_sub_long)
```

```
## # A tibble: 6 × 4
##   Sample  OTU     Count Total_Reads
##   <chr>   <chr>   <dbl>       <dbl>
## 1 DA00006 Otu0001     0         881
## 2 DA00006 Otu0002     1         881
## 3 DA00006 Otu0003   323         881
## 4 DA00006 Otu0004   539         881
## 5 DA00006 Otu0005     1         881
## 6 DA00006 Otu0006     0         881
```

And with these counts, we can get the proportions of each OTU in each sample, and then multiply by 1000000 to get the reads per million value.

```
raw_count_sub_long <- raw_count_sub_long %>%
  mutate("cpm" = Count/Total_Reads*1000000)
```

```
head(raw_count_sub_long)
```

```
## # A tibble: 6 × 5
##   Sample  OTU     Count Total_Reads     cpm
##   <chr>   <chr>   <dbl>       <dbl>   <dbl>
## 1 DA00006 Otu0001     0         881      0
## 2 DA00006 Otu0002     1         881   1135.
## 3 DA00006 Otu0003   323         881 366629.
## 4 DA00006 Otu0004   539         881 611805.
## 5 DA00006 Otu0005     1         881   1135.
## 6 DA00006 Otu0006     0         881      0
```

Exercise 2

Transform these cpm values to get the log (ln) cpm. Note that the log of 0 is undefined. Since log(1) is 0, we can add 1 to each value before taking the log so you don't get any values that aren't defined. Call the new column "lcpm".

Editing Text

`str_replace()`

As an exercise in modifying character strings, let's say we wanted to remove the first two zeros following the "Otu" in the OTU column. Since our otu names only go up to 10, we don't really need these.

```
raw_count_sub_long %>%
  mutate("OTU" = str_replace(OTU, pattern = "(Otu){00}(..)", replacement = "\\1\\2"))
```

```
## # A tibble: 100 × 6
##    Sample  OTU   Count Total_Reads     cpm  lcpm
##    <chr>   <chr> <dbl>       <dbl>   <dbl> <dbl>
##  1 DA00006 Otu01     0         881       0     0
##  2 DA00006 Otu02     1         881    1135.  7.04
##  3 DA00006 Otu03   323         881 366629.  12.8
##  4 DA00006 Otu04   539         881 611805.  13.3
##  5 DA00006 Otu05     1         881    1135.  7.04
##  6 DA00006 Otu06     0         881       0     0
##  7 DA00006 Otu07    17         881   19296.  9.87
##  8 DA00006 Otu08     0         881       0     0
##  9 DA00006 Otu09     0         881       0     0
## 10 DA00006 Otu10     0         881       0     0
## # ℹ 90 more rows
```

```
head(raw_count_sub_long)
```

```
## # A tibble: 6 × 6
##   Sample  OTU     Count Total_Reads     cpm  lcpm
##   <chr>   <chr>   <dbl>       <dbl>   <dbl> <dbl>
## 1 DA00006 Otu0001     0         881       0     0
## 2 DA00006 Otu0002     1         881    1135.  7.04
## 3 DA00006 Otu0003   323         881 366629.  12.8
## 4 DA00006 Otu0004   539         881 611805.  13.3
## 5 DA00006 Otu0005     1         881    1135.  7.04
## 6 DA00006 Otu0006     0         881       0     0
```

Before Thurs

The file 'vik/vik_abundance.csv' in the 'M8161_R_files' directory stores (partial) data from a large study investigating ocean microbial communities. The values in the table represent tp means, which are derived from the raw counts. However, these values do not account for differing sequencing depths across samples. The relevant total sequence read counts for each sample needed to normalize for sequence read counts are stored in the 2nd row in this file.

- First, read in the data and store the sequence read information in one tibble and the OTU counts (tp means) in another.
- Next, reformat the data in each of these tibbles to get them into long format, with columns named "sample_site" & "seq_reads" for the sequence reads tibble, and columns named "sample_site" & "tp_mean" for the OTU count tibble.
- Finally, the values in the "contig" column of the tibble containing the OTU counts consist of fields separated by "_". These identifiers are longer than they need to be, with the first four of the eight fields providing enough information to uniquely identify each OTU. So, let's reduce these down to just the first 4 fields (everything before"_length"). Also, rename this column to "OTU".

See below for previews of what these tibbles should look like, along with expected dimensions of each…

```
view(vik_seq_cts_long)
```

	sample_site	seq_reads
1	St08_oxy	5252248
2	St07_oxy	11327318
3	St17_scm	6666728
4	St17_oxy	10379407
5	St16_scm	7677440
6	St16_oxy	16844671
7	St14_scm	8128816
8	St14_oxy	10644521
9	St08_scm	7198305
10	St07_scm_col-0	8676581

```
view(vik_otu_cts_long)
```

	OTU	sample_site	tm_mean
1	St07_omz1_NODE_1001	St08_oxy	0.000000
2	St07_omz1_NODE_1001	St07_oxy	0.000000
3	St07_omz1_NODE_1001	St17_scm	0.000000
4	St07_omz1_NODE_1001	St17_oxy	0.000000
5	St07_omz1_NODE_1001	St16_scm	0.000000
6	St07_omz1_NODE_1001	St16_oxy	0.000000
7	St07_omz1_NODE_1001	St14_scm	0.000000
8	St07_omz1_NODE_1001	St14_oxy	0.000000
9	St07_omz1_NODE_1001	St08_scm	0.000000
10	St07_omz1_NODE_1001	St07_scm_col-0	0.000000
11	St07_omz1_NODE_1001	St14_uomzD	2.526239
12	St07_omz1_NODE_1001	St08_uomzD	0.000000
13	St07_omz1_NODE_1001	St18_uomzD	2.121984
14	St07_omz1_NODE_1001	St14_omz	5.448524
15	St07_omz1_NODE_1001	St07_omz1	4.105292

The dimensions of these tibbles should be…

```
dim(vik_seq_cts_long)
```

```
## [1] 22  2
```

```
dim(vik_otu_cts_long)
```

```
## [1] 22000     3
```