

Data Handling in R

Here we introduce a few key basics of handling data in R, using RStudio. In the process, you will also get experience with installing and loading packages. Specifically, we will:

1. import data from an excel file as both csv and txt files
2. install the *dplyr* package in R
3. in R, we will use *dplyr* to select, filter, arrange, rename, mutate, and summarise data

Notice the changes in fonts above? Throughout this course, packages will be named in *italics*, and commands will be in Mono font. And remember, things learned in this course are cumulative. For example, having learned how to install and load a package here, we won't teach those steps in other sessions, but simply instruct you to install and load.

We will work with data on baby names, available from the Social Security Administration, using in part code at <http://www.rforexcelusers.com/combine-delimited-files-r/>. Here each state's records (1910-1915) for babies born in each year are listed, where the minimum number for each name = 5. The data are available as separate .txt files or all in a zip file on the course web page <http://sciences.ucf.edu/biology/d4lab/wp-content/uploads/sites/125/2017/01/SSA-babynames.zip>

1. Make a folder on your desktop (for now) and unzip the file into that folder.
2. Set this folder as your working directory in R. Use the menu at the top: Session... Set Working Directory... Choose Directory...

How much data do we have? How many MB? _____

Working through all 51 data sets to get them to match up, etc. is *not something we should do in Excel*. Instead let's use apply functions and *dplyr* in R do the work.

We need to get all those .txt files into R.

3. Now make an object that is a list of file names (all the .txt files)

```
file_names <- list.files(getwd())
```

4. Now we use `lapply` to repeatedly use the `read.csv` function for all `file_names`

```
files <- lapply(file_names, read.csv, header=F, stringsAsFactors = F)
babynames <- do.call(rbind,files)
# This binds rows (rbind) to combine all 52 data sets into one data set called
"babynames"
```

Wait for the Red Stop Sign to go away (it means the computer is working – if you click on that icon it will stop the process). We now have a dataframe called `babynames`, that combined all 51 state (+DC) .txt files.

How many rows of data do we have? _____ How many columns? _____

5. We have not yet named the variables. To see the first few rows , enter:

```
head(babynames)
```

6. So let's add some headings:

```
names(babynames) <- c("state", "sex", "year", "name", "count")
```

7. Now let's ensure that R works on babynames whenever we run a command below. Enter

```
attach(babynames) # This says "use this file below"; to undo, use detach
```

8. So now we have some data to play with, using `dplyr` . So let's install and load `dplyr` . Installing can be done in 2 ways: using RStudio menus or commands. In RStudio, click on Packages (top of bottom-right window). If `dplyr` is in the list, click on the box (or type `library("dplyr")` in the Console. If it's not in the list, click Install at the top of the Packages window and type `dplyr` in the pop-up window. That implements the `"install.packages("dplyr")` command you could have typed in the Console.
9. Now click on the blue word "dplyr" to see what R documentation looks like. A very good strategy is to first read vignettes – for an English translation of what packages do. Lots of other info is here too. Plan to spend time perusing Instructions when you first learn a package. And when learning details or troubleshooting, browse the web – many web sites are out there for help (some are listed in this course's R Resources web site). And at the bottom of Help screens are often helpful examples. And plan to run thru commands using example data sets provided.

For now, let's plunge into some main `dplyr` commands.

Filter: We use filter to select a subset of **rows** in a dataframe. For example, you can filter out for a name (maybe your name, though this US-based and historical list may not capture all names.

11. Here is an example – modify it as you see fit:

```
filter(babynames, name == "Pedro")
```

Which yields 1738 entries (any state and year in which there are ≥ 5 babies named Pedro in the SSA data. But notice in the upper-right window – we still only have a dataframe named "babynames" – one called Pedro does not yet exist. Let's try this:

```
Pedro <- filter(babynames, name == "Pedro") # to make a dataframe named "Pedro"
```

Now let's set up some graphs, eventually, using a name of your choice. So filter first for that name. Examples below continue to use the Pedro data (substitute your own name if you like).

11. We'll lump West Coast states together by filtering:

```
West <- filter(Pedro, state == c("CA", "OR", "WA")) # this makes a concatenated  
[c()] list
```

12. Now do the same for states in the Northeastern US: ME, NH, VT, MA, CT, RI, NY, PA

13. And make another for the "fly-over" central US: ND, SD, NE, KS, OK, TX

Arrange: This re-sorts data for your specified variable, much like you may do in Excel with Data... Sort. Our data are arranged by state, and we'll arrange them instead by year and state:

```
Westyearlya <- arrange(West, year)
```

Look at those data (click on West in the upper right box); they are now sorted by year first. And you may see >1 count in the same year if >1 state has entries that year. So let's sum for each year so that we can plot the annual number of babies named Pedro born each year. We do this by a grouped `summarise` command.

Group by: This is another way to re-arrange data that IDs groups for later calculations. For example, we get the same result as `arrange` above **BUT** you'll see that `group_by` permits calculations for years (because groups are assigned) when `arrange` does not (that just sorts).

```
Westyearlyg <- group_by(West, year)
```

Summarise: calculates column values for a data frame (or part of one), such as total, mean, etc. As promised we'll do that here per year in each region's data, but first let's use it overall to see how it works:

```
allPedros <- summarise(Pedro, allPeds = sum(count)) # to find out how many  
Pedros there have been.
```

14. Now summarise the West yearly data for real annual totals. First try this with the arranged data above:

```
annualWest <- summarise(Westyearlya, total = sum(count))
```

We get only the one total for all years (not grouped by year).

15. But if we use the grouped data above:

```
annualWest <- summarise(Westyearlyg, total = sum(count))
```

We get exactly what we want – a total for each year. Repeat that command to get grouped sums for the other two regions.

And now we can graph results. We first tell R to stack the graphs vertically so you can compare trends:

```
par(mfrow=c(3,1)) # specifies 3 rows and 1 column to stack 3 graphs for easy comparisons. "mfrow" seems to mean "multi function rows."
```

[To re-set back to one graph per window, just use `par(mfrow=c(1,1))`]

16. Now make plots. The first one is requested by:

```
plot(annualWest$total ~ annualWest$year, type="b")
```

Repeat that basic command for the other two regions to see all 3 graphs, stacked in order.

You should now be able to graph a name's trend through time in the West coast, Northeastern states, and the Central states. Do you see differences in the trends? Did the coasts lead the "fly-over states" in trends?

Clearly, *we have just scratched the surface of dplyr*. For example, you should learn about running multiple commands together in a "pipe" by separating them with a `%>%` flag. Good resources for getting better with dplyr include:

- <http://blog.rstudio.org/2014/01/17/introducing-dplyr/>
- <http://www.r-bloggers.com/using-r-quickly-calculating-summary-statistics-with-dplyr/>
- http://www.cookbook-r.com/Manipulating_data/Summarizing_data/
- <http://rpubs.com/justmarkham/dplyr-tutorial>

Others abound – remember, when using R, the web is your friend... For example, you should explore using **mutate** and the **join** set of commands on your own. *Hint, Hint.*

17. Finally, save your R script and data files because you will use this some more for this week's homework! If not, you can reconstruct it all using information above.