# Data wrangling in R

*Andrew W. Park & John M. Drake*

## Introduction

This is the third in a series of five exercises that constitute *Training Module 1: Introduction to Scientific Programming*, taught through the IDEAS PhD program at the University of Georgia Odum School of Ecology in conjunction with the Center for the Ecology of Infectious Diseases.

This exercise explores methods of data wrangling, which is essentially the good practices associated with storing and manipulating data. This module uses the following libraries:

```r
library(tidyverse)
library(magrittr)
library(dplyr)
library(stringr)
library(GGally)
library(maptools)
library(ggmap)
library(maps)
```

## Case study

Lyme disease is a tick-borne emerging infectious disease in the US dating back to the 1970s and caused by a bacteria called *Borrelia burgdorferi.* It is thought to infect hundreds of thousands of people each year, though not all cases get reported. The distribution of cases of Lyme across the US is incompletely understood to this day. We'll be working with three distinct data sets

- The US census data (population size) 2000-2014 by county 'pop.csv'
- The CDC public-use data set on Lyme disease cases 2000-2015 by county 'lyme.csv'
- The PRISM data set, which contains commonly-used climate variables, 2000-2015, by county 'climate.csv'

Our ultimate research goal is to better understand the relationship between climate, population size and Lyme disease cases. Our scientific programming goals are to

- Import the data sets
- Convert data to the `tidy data` format
- Identify and manipulate text strings with the `regexp` language
- Merge data sets
- Visualize geographic variables as a map

In the subsequent module, we'll continue to work with these data and develop more good techniques to support hypothesis generation.

## Importing data

The Lyme disease data is relatively simple to import because the CDC maintains the data as a csv file (this data is provided to you on the workshop web page, but for your records it is available here: https://www.cdc.gov/lyme/stats/). We're going to use the `read_csv` command for loading all these data sets (not `read.csv`). The `read_csv` will create tibble versions of data frames, which retain all the good things about data frames, but not some of the less good things (more here on tibbles: https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html)

Similarly, the Census U.S. Intercensal County Population Data, 1970-2014 (also provided to you) is available from the National Bureau of Economic Research as a csv file ( http://www.nber.org/data/census-intercensal-county-population.html)

The PRISM data for total rainfall and average temperature, is available for overlapping years, 2000-2015 (http://www.prism.oregonstate.edu/). Please note: your instructors have obtained and formatted this data from PRISM in advance, as it involves some time consuming steps that are beyond the scope of this workshop (please ask in a break if you're interested in learning about this).

*Task 1: Read in all three csv files as tibble data frames. For consistency with these notes, we'll assign their dataframes to be called "ld", "pop" and "prism", resectively.*

## Converting data to the `tidy data` format

Currently, only the PRISM data conforms to the concept of `tidy data`:

- Each variable has its own column
- Each observation has its own row
- Each value has its own cell

This is a highly recommended format to store data, and you can read more about why here: http://www.jstatsoft.org/v59/i10/paper. Unfortunately, it is not a standard way of storing data. Fortunately, there are tools within the R programming environment that can help us convert data to the tidy format.

### A note about FIPS codes

You'll note that there is a column called `fips`. This is a number that uniquely defines a county in the US. The first 1 or 2 digits refer to the state (e.g. 6=California, 13=Georgia). The following 1-3 digits identify the counties of that state, numbered in alphabetical order, usually using only odd numbers (e.g. 59 is Clarke county, where we are now). The full FIPS code for Clarke county is 13059 (we 'pad' the county code with zeros to ensure it is three digits in total - but we don't do that with states, which can be 1 or 2 digits). The format may seem a little quirky, but the system works very well in uniquely identifying counties in the US and is one of the common ways that infectious disease, climate and demographic data are organized in this country.

### Worked example: Census data `pop`

*Task 2: By inspecting the 'pop' data, and talking with your neighbors and instructors, articulate in which way(s) these data fail to conform to the `tidy data` format?*

The following code chunk manipulates the `pop` data frame to the `tidy data` format, and does one or two other useful things along the way. It uses three functions from the `dplyr` package: *select*, *gather*, and *mutate*. In addition, it uses `str_replace_all`, which is a function of the `stringr` packages that replaces characters (text) with other characters. The basic call is

```
str_replace_all(x,y,z)
```

where `x` is where R should look to perform these replacements, `y` is the pattern to be replaced, and `z` is the replacing pattern. The syntax to identify/find character strings is known as `regexp` language (short for *regular expression*). It's beyond the scope of this workshop to learn the language in full, but you'll note that the code chunk below can remove text by replacing a string with "". It also finds characters that begin with "0" by using "^0" (here the ^ means 'start of character string').

*Task 3: Before running the code, read it to see if you can work out or guess what each line is doing. Before running a line of code, remind yourself of the current format of `pop` by typing it's name in the console window and hitting return. Then run each line, one by one, and after each line has been executed, look again at `pop`* *to see what it did. Once you're clear about what each line is doing, add a comment line above each code line to remind you, in your own words, what the code does.*

```
pop %<>% select(fips,starts_with("pop2"))
pop %<>% gather(starts_with("pop2"),key="str_year",value="size") %>% na.omit
pop %<>% mutate(year=str_replace_all(str_year,"pop",""))
pop %<>% mutate(year=as.integer(year))
pop %<>% mutate(fips=str_replace_all(fips,"^0",""))
pop %<>% mutate(fips=as.integer(fips))
```

The code is now in `tidy data` format. Arguably we could remove state-wide population summaries too, but we can do that when we combine data sets (*How would you do remove state-wide summaries at this stage?*). Also we could remove the character column 'str_year' now that we've converted year to integer (*How would you do this?*), but it may be useful to retain it.

### Over to you: Lyme disease data

The CDC Lyme disease data is also not currently in `tidy data` format.

*Task 4: Write a code chunk to convert the Lyme disease data to `tidy data` format.*

Hints:

- Draw on all your resources, especially what you've already learned and your instructors.
- You are going to want a column that has the full FIPS codes for each county, as explained above. You should write a function that takes two arguments: state code and county code. The function should return a 4 or 5 digit FIPS code (integer data type). For county codes that are 1 or 2 digits, you'll need to 'pad' them with the appropriate number of zeros (this will require an if-else flow control). You can determine how long a character string is using the str_length function of the `stringr` package (e.g. str_length("disease")=7). As you might expect, the command to paste two character strings together is `paste` (remember, you can get help with ?paste). To apply your function to every cell you can use the `rowwise` function which is part of the `dplyr` package (as used in the presentation example).
- Use 'rename' to rename the columns "STNAME" and "CTYNAME" with "state" and "county", respectively (useful for a later join-operation for mapping)

## Combining data sets

Base R has a `merge` function that works well in combining data frames, and we encourage you to familiarize yourself with it (?merge). The `dplyr` package has a set of `join` functions that operate on tibbles. This is the method we are using in this workshop.

*Task 5: Join the Lyme disease data frame and PRISM data frame together to form a new data frame, and in such a way that it retains county-year combinations for which we have both disease and clime data.*

Hint: revisit the presentation on joins and perhaps even the link to superheroes, if you need more information (and ask instructors)

*Task 6: Write a line of code to additionally combine the demographic data with the Lyme disease and climate data.*

## Obtaining summary information

We may want to obtain an overview of a large data frame. For this purpose, we'll use the `summarize` function of the `dplyr` package (note: base R can perform similar duties with the `aggregate` function).

The accompanying presentation demonstrates `summarize` by obtaining the average test scores for students according to their major. Building on this example:

*Task 7: Write two lines of code that create two new data frames: (1) to determine how many cases of Lyme disease were reported each year, (2) the average number of cases in each state - averaged across county and year. What was the worst year? Which three states have been most impacted on average?*

Hints:

- we previously applied a `rowwise` operation to the Lyme disease data set `ld` - it is wise to `ungroup` it early in the pipe (e.g. start `my_dataframe <- ld %>% ungroup %>% ...`)
- when you use `summarize` you can define the new column name (recall 'grades' example of 'avScore') - otherwise R will give it a default name
- you can include the `arrange` function at the end of your pipe to put the data in a meaningful order (e.g. `...  %>% arrange(avScore)` or `...%>% arrange(desc(avScore))`)

## Saving data frames as objects and files

We've come a long way since we read in the raw data on Lyme disease, demography and climate. We'll save the combined data frame you created in readiness for a subsequent module

*Task 8: use `save` to create an Rda file of the data frame and use `write_csv` to create a csv file of the same (remember to try ?save and ?write_csv in the console if you're not sure how these functions work).*

## Using FIPS and built-in mapping tools to visualize geographic data

R has ever-increasing capabilities to produce high-quality maps and sophisticated geographical data analysis. For an introductory scientific programming workshop we can't go into all those details - but we can illustrate some simple mapping ideas that can help us understand and visualize our spatial data.

ggplot has US mapping objects that we can take advantage of.

*Task 9: Add annotations to the following lines of code with comment lines to explain what they are achieving. Note: in the code line with "group_by(ld.prism.pop)" you need to replace the data frame in parentheses with the name of your data frame in which you combined Lyme disease, climate and demography data (Task 6)*

```r
#get map data for US counties and states
county_map <- map_data("county")
state_map <- map_data("state")

## ANNOTATE FROM HERE
ag.fips <- group_by(ld.prism.pop,fips)
ld.16y<-summarize(ag.fips,all.cases=sum(cases))
ld.16y<-left_join(select(ld.prism,c(state,county,fips)),ld.16y)
ld.16y<-distinct(ld.16y)
ld.16y %<>% rename(region=state,subregion=county)
ld.16y$subregion<-str_replace_all(ld.16y$subregion," County","")
ld.16y$region<-tolower(ld.16y$region)
ld.16y$subregion<-tolower(ld.16y$subregion)
ld.16y %<>% mutate(log10cases=log10(1+all.cases))
map.ld.16y<-left_join(county_map,ld.16y)
ggplot(map.ld.16y)+geom_point(aes(long,lat,color=log10cases),size=0.1) +
  scale_colour_gradientn(colours=rev(rainbow(4)))
```