Scraping Web Data

Unit 3 - Lab 5

Directions: Follow along with the slides and answer the questions in **BOLDED** font in your journal.

# The web as a data source

* The internet contains huge amounts of information.
* Gathering this information in an automated fashion is referred to as *scraping data*.
* The difficulty of scraping data from the web varies quite a bit.
* So let's start with something fairly easy.
* Scraping data is usually done in two parts:
* Step 1: Gather the information from the web.
* Step 2: Clean it up and make it something useful.

# Our first web scraper

* Our first task will be to scrape the data contained [on this website](http://web.ohmage.org/mobilize/resources/ids/data/mountains.html).
* Click on the link.
* Notice that the data doesn't look so different from what data already looks like in RStudio.
* **Briefly describe what the data on the website is about.**
* Since data on the web can change all the time, we typically write *web scrapers* as scripts.
* Which lets us just re-run all the code in the script to access the latest data.
* Open a new script by clicking: File -> New File -> R Script.
* Write each new line of code into the script.

# Getting started.

* To get started, we first notice that:
* We have a webpage.
* It contains a table of data.
* To access this data, we're going to need some functions that RStudio doesn't come with when it's first installed.
* Specifically, we need the XML package, which includes functions on reading HTML tables.
* To get started, *TYPE* the [url of the website we want to scrape](http://web.ohmage.org/mobilize/resources/ids/data/mountains.html) as an object in R (Don't copy & paste):

data\_url <- "http://web.ohmage.org/mobilize/  
 resources/ids/data/mountains.html"

# Loading packages

* Try running the following code (It probably won't work but that's ok):

readHTMLTable(data\_url)

* R is popular among data scientists because it's easy to create and share new functions.
* We want to have RStudio scrape an HTML table, which as you might have found by running the code above, isn't something RStudio can do by itself.
* To load the *package* that will let us easily scrape HTML tables, run:

library(XML)

# The library() function

* The library() function is how we can add new functions for R to use.
* We loaded the files in the XML package.
* Run the following line of code now:

readHTMLTable(data\_url)

* **How is running the code readHTMLTable(data\_url) different after loading the XML package than before?**

# Saving tables

* The readHTMLTable() function will scrape *EVERY* table that is on a website.
* This means we'll, typically, need to sift through the different tables to find the one we're interested in.
* Run the following to save all of the scraped tables as an object called tables:

tables <- readHTMLTable(data\_url)

* Since our site contains only a single table, we'll run the following to save the table as a data object called mountains:

mountains <- tables[[1]]

# Multiple tables

* When a website contains multiple tables, we'll replace the 1 in tables[[1]] with a 2, 3, and so on until we find the table we're interested in.
* **What happens when you run tables[[2]]?**
* **Since our data only has a single table why does tables[[2]] return the output that it does?**

# Looking at our new data

* Now that we've scraped our data from the web, our next job is to clean it up.
* Type the following to view the data:

View(mountains)

names(mountains)

* Then answer the following questions:
* **Is something wrong with the variable names?**
* **Do the values for each variable seem reasonable?**
* **What do the variables long and lat tell us about our data?**

# Cleaning our data (variable names)

* One of the most common problems with scraped data is poorly formatted variable names.
* Which are at least easy to fix.
* If you think you have better names for the data's variables, use the following example code to change them:

names(mountains) <- c("new\_name1",   
 "new\_name2",  
 ...,   
 "new\_name10")

* Note: Don't use new\_name1 ... try fixing the names that printed when you ran names(mountains) Cleaning our data (variable classes) ====================================
* After inspecting our variable names, we want to make sure that *categorical variables* (or *factors*) are composed of categories and that *numerical variables* are composed of numbers.
* Type the following to determine if R thinks that our numbers for mountain elevation (in feet) are actually numbers:

mean(~elev\_ft, data=mountains)

* If you get an error, then R doesn't understand that elev\_ft is a numerical variable.

# Data structure

* R will tell you what type of variable it considers each variable.
* Use the *structure* function to find out what types of variables R thinks your data contains:

str(mountains)

* Here we see the problem.
* R thinks that our numerical variables (elev\_ft, prominence\_ft, rank, etc.) are actually Factors or *categorical variables*.

# Changing the script

* Why is it better to sometimes use a script? Because it's easier to make changes to your code by using it.
* View your data and determine which variables are *categorical* (i.e. *factors*) and which should be numerical.
* Next, write down the variable types and save them as a vector called var\_types.
* To get you started, the first 3 varibles are *factors* (categorical) and the 4th is *numeric*. So we write:

var\_types <- c("character", "character", "character", "numeric", ...)

* Finish the above code with the remaining variables types & run it.

# Making some changes

* If you have been writing your code in a script you can just scroll back now and update it with the following:

tables <- readHTMLTable(data\_url,   
 colClasses=var\_types)

* The rest of your code in the script can remain the same which saves you from having to write the same code over again:
* **Use your previous code to create your mountains data.**
* **When you're finished, find and report the mean elevation (in feet) of the mountains**

# Saving your data

* Now that you've scraped and cleaned your data, you'll want to save it by running:

save(mountains, file="mountains.rda")

* **Check the *Files* pane in Rstudio to make sure your saved data shows up. In which directory can you find it?**
* Finally, run the following code to put your mountains data to good use:

make\_map(lat, long, data=mountains)