

Entering and cleaning data #3

Odds and ends

Groups for homework #5

- Group 1: Jessy, Kyle, Alyssa, Devin
- Group 2: Kayla, Kathleen, Amanda
- Group 3: Grant, Aeriell, Rebecca
- Group 4: Anastasia, Camron, Nichole
- Group 5: Randy, Ana, Amy

Group projects

Potential ideas:

- Predicting if trapped mosquitoes test positive for West Nile virus
- Investigating temporal and spatial patterns in human West Nile cases within a state (and link to Census data and / or climate information)
- Explore 'omics data for West Nile virus (<https://omics-lhv.org/data/>)
- Genomic epidemiology of West Nile virus in California (<https://github.com/andersen-lab/west-nile>)
- Patterns in papers about West Nile virus

“Products”

Each group will need to generate:

1. The code for their piece of the project (we'll use GitHub to collaborate).
2. A brief report (4 to 5 pages) describing how the group tackled their problem, what pieces were particularly challenging and how they tackled those, what they would do differently if they started fresh, and what interesting things they found in this set of data. Each group must submit a draft of this the last Wednesday of class and submit the final version on the day of presentations.
3. A presentation (12 minutes maximum) that covers the same material as the report. The groups will present these during finals week, in the time period scheduled for a final exam for our course.

Pulling online data

API: “Application Program Interface”

An API provides the rules for software applications to interact. In the case of open data APIs, they provide the rules you need to know to write R code to request and pull data from the organization’s web server into your R session.

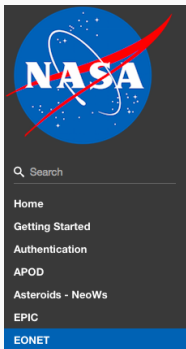
Often, an API can help you avoid downloading all available data, and instead only download the subset you need.

Strategy for using APIs from R:

- Figure out the API rules for HTTP requests
- Write R code to create a request in the proper format
- Send the request using GET or POST HTTP methods
- Once you get back data from the request, parse it into an easier-to-use format if necessary

API documentation

Start by reading any documentation available for the API. This will often give information on what data is available and how to put together requests.



QUERYING BY DATE(S)

Parameter	Type	Default	Description
date	YYYY-MM-DD	Most Recent Available	Retrieve metadata for all imagery available for a given date.
available_dates	string	All Available Dates	Retrieve a listing of all dates with available imagery.
api_key	string	DEMO_KEY	api.nasa.gov key for expanded usage

EXAMPLE QUERIES

https://api.nasa.gov/EPIC/api/v1.0/images.php?api_key=DEMO_KEY

https://api.nasa.gov/EPIC/api/v1.0/images.php?date=2015-10-31&api_key=DEMO_KEY

https://api.nasa.gov/EPIC/api/v1.0/images.php?available_dates&api_key=DEMO_KEY

More examples and usage tips can be found on the [EPIC About Page](#).

Source: <https://api.nasa.gov/api.html#EONET>

Many organizations will require you to get an API key and use this key in each of your API requests. This key allows the organization to control API access, including enforcing rate limits per user. API rate limits restrict how often you can request data (e.g., an hourly limit of 1,000 requests per user for NASA APIs).

You should keep this key private. In particular, make sure you do not include it in code that is posted to GitHub.

Example— `riem` package

The `riem` package, developed by Maelle Salmon and an ROpenSci package, is an excellent and straightforward example of how you can use R to pull open data through a web API.

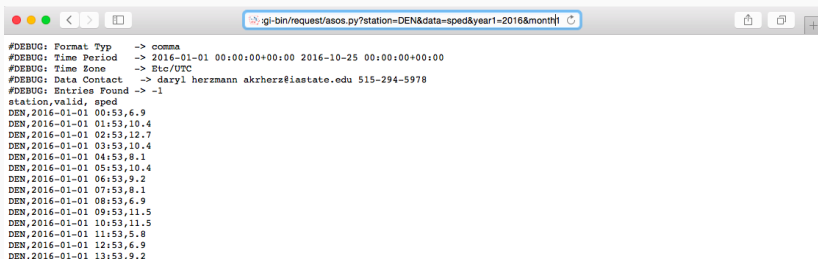
This package allows you to pull weather data from airports around the world directly from the Iowa Environmental Mesonet.

Example— `riem` package

To get a certain set of weather data from the Iowa Environmental Mesonet, you can send an HTTP request specifying a base URL, “`https://mesonet.agron.iastate.edu/cgi-bin/request/asos.py/`”, as well as some parameters describing the subset of dataset you want (e.g., date ranges, weather variables, output format).

Once you know the rules for the names and possible values of these parameters (more on that below), you can submit an HTTP GET request using the `GET` function from the `httr` package.

Example– riem package



```
#DEBUG: Format Typ    -> comma
#DEBUG: Time Period   -> 2016-01-01 00:00:00+00:00 2016-10-25 00:00:00+00:00
#DEBUG: Time Zone     -> Etc/UTC
#DEBUG: Data Contact  -> daryl herzmann akrherz@iastate.edu 515-294-5978
#DEBUG: Entries Found -> -1
station,valid, sped
DEN,2016-01-01 00:53,6.9
DEN,2016-01-01 01:53,10.4
DEN,2016-01-01 02:53,12.7
DEN,2016-01-01 03:53,10.4
DEN,2016-01-01 04:53,8.1
DEN,2016-01-01 05:53,10.4
DEN,2016-01-01 06:53,9.2
DEN,2016-01-01 07:53,8.1
DEN,2016-01-01 08:53,6.9
DEN,2016-01-01 09:53,11.5
DEN,2016-01-01 10:53,11.5
DEN,2016-01-01 11:53,5.8
DEN,2016-01-01 12:53,6.9
DEN,2016-01-01 13:53,9.2
```

https://mesonet.agron.iastate.edu/cgi-bin/request/asos.py?station=DEN&data=sknt&year1=2016&month1=6&day1=1&year2=2016&month2=6&day2=30&tz=America%2FDenver&format=comma&latlon=no&direct=no&report_type=1&report_type=2

Using httr to get data from a webpage

When you are making an HTTP request using the GET or POST functions from the httr package, you can include the key-value pairs for any query parameters as a list object in the query argument of the function.

```
library(httr)
meso_url <- paste0("https://mesonet.agron.iastate.edu/",
                  "cgi-bin/request/asos.py/")
denver <- GET(url = meso_url,
             query = list(station = "DEN", data = "sped",
                          year1 = "2016", month1 = "6",
                          day1 = "1", year2 = "2016",
                          month2 = "6", day2 = "30",
                          tz = "America/Denver",
                          format = "comma"))
```

Using httr to get data from a webpage

The GET call will return a special type of list object with elements that include the url you queried and the content of the page at that url:

```
str(denver, max.level = 1, list.len = 6)
```

```
## List of 10
## $ url          : chr "https://mesonet.agron.iastate.edu/cgi-bin
## $ status_code: int 200
## $ headers      :List of 6
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
## $ all_headers:List of 1
## $ cookies     :'data.frame': 0 obs. of  7 variables:
## $ content      : raw [1:230652] 23 44 45 42 ...
## [list output truncated]
## - attr(*, "class")= chr "response"
```

Using `httr` to get data from a webpage

The `httr` package includes functions to pull out elements of this list object, including:

- `headers`: Pull out the header information
- `content`: Pull out the content returned from the page
- `status_code`: Pull out the status code from the GET request (e.g., 200: okay; 404: not found)

Note: For some fun examples of 404 pages, see <https://www.creativebloq.com/web-design/best-404-pages-812505>

Using httr to get data from a webpage

You can use `content` from `httr` to retrieve the contents of the HTTP request we made. For this particular web data, the requested data is a comma-separated file, so you can convert it to a dataframe with `read_csv`:

```
denver %>% content() %>%  
  read_csv(skip = 5, na = "M") %>%  
  slice(1:3)
```



```
## # A tibble: 3 x 3  
##   station valid          sped  
##   <chr>      <dtm>          <dbl>  
## 1 DEN      2016-06-01 00:00:00    9.2  
## 2 DEN      2016-06-01 00:05:00    9.2  
## 3 DEN      2016-06-01 00:10:00    6.9
```

Example– riem package

The riem package wraps up this whole process, so you can call a single function to get in the data you want from the API:

```
library(riem)
denver_2 <- riem_measures(station = "DEN",
                          date_start = "2016-06-01",
                          date_end = "2016-06-30")
denver_2 %>% slice(1:3)
```

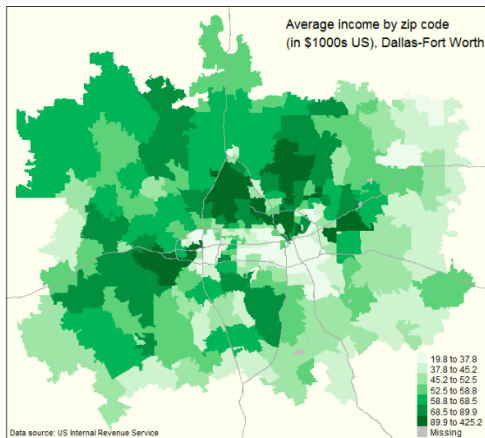


```
## # A tibble: 3 x 24
##   station valid          lon   lat  tmpf  dwpf  relh  drct  sknt
##   <chr>   <dtm>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 DEN     2016-06-01 00:00:00 -105.  39.8   NA    NA    NA    70    7
## 2 DEN     2016-06-01 00:05:00 -105.  39.8   NA    NA    NA    80    8
## 3 DEN     2016-06-01 00:10:00 -105.  39.8   NA    NA    NA    80    9
## # ... with 15 more variables: p01i <dbl>, alti <dbl>, mslp <dbl>,
## #   vsby <dbl>, gust <dbl>, skyc1 <chr>, skyc2 <chr>, skyc3 <chr>,
## #   skyc4 <chr>, skyl1 <dbl>, skyl2 <dbl>, skyl3 <dbl>, skyl4 <dbl>,
## #   wxcodes <chr>, metar <chr>
```

Example R API wrappers

- Location boundaries
 - States
 - Counties
 - Blocks
 - Tracks
 - School districts
 - Congressional districts
- Roads
 - Primary roads
 - Primary and secondary roads
- Water
 - Area-water
 - Linear-water
 - Coastline
- Other
 - Landmarks
 - Military

Example from: Kyle Walker. 2016. “tigris: An R Package to Access and Work with Geographic Data from the US Census Bureau”. The R Journal.



US Census packages

A number of other R packages also help you access and use data from the U.S. Census:

- `acs`: Download, manipulate, and present American Community Survey and Decennial data from the US Census (see “Working with the American Community Survey in R: A Guide to Using the `acs` Package”, a book available free online through the CSU library)
- `USABoundaries`: Historical and contemporary boundaries of the United States of America
- `idbr`: R interface to the US Census Bureau International Data Base API (e.g., populations of other countries)

rOpenSci (<https://ropensci.org>):

“At rOpenSci we are creating packages that allow access to data repositories through the R statistical programming environment that is already a familiar part of the workflow of many scientists. Our tools not only facilitate drawing data into an environment where it can readily be manipulated, but also one in which those analyses and methods can be easily shared, replicated, and extended by other researchers.”

rOpenSci collects a number of packages for tapping into open data for research: <https://ropensci.org/packages>

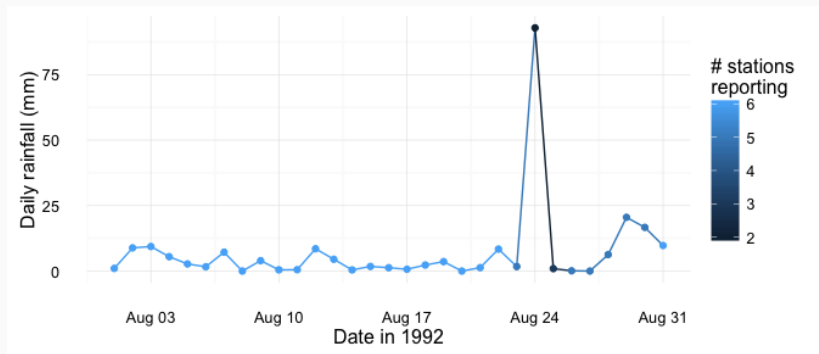
Some examples (all descriptions from rOpenSci):

- **AntWeb**: Access data from the world's largest ant database
- **chromer**: Interact with the chromosome counts database (CCDB)
- **gender**: Encodes gender based on names and dates of birth
- **musmeta**: R Client for Scraping Museum Metadata, including The Metropolitan Museum of Art, the Canadian Science & Technology Museum Corporation, the National Gallery of Art, and the Getty Museum, and more to come.
- **rusda**: Interface to some USDA databases
- **webchem**: Retrieve chemical information from many sources. Currently includes: Chemical Identifier Resolver, ChemSpider, PubChem, and Chemical Translation Service.

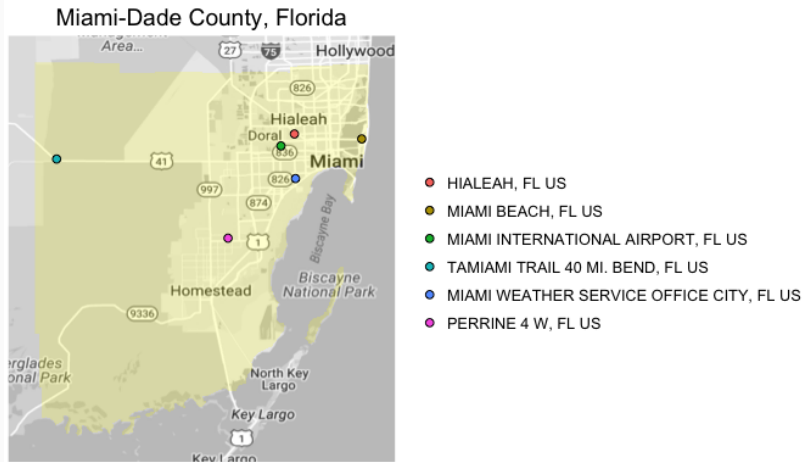
“Access climate data from NOAA, including temperature and precipitation, as well as sea ice cover data, and extreme weather events”

- Buoy data from the National Buoy Data Center
- Historical Observing Metadata Repository (HOMR)— climate station metadata
- National Climatic Data Center weather station data
- Sea ice data
- International Best Track Archive for Climate Stewardship (IBTrACS)— tropical cyclone tracking data
- Severe Weather Data Inventory (SWDI)

The countyweather package wraps the rnoaa package to let you pull and aggregate weather at the county level in the U.S. For example, you can pull all data from Miami during Hurricane Andrew:



When you pull the data for a county, the package also maps the contributing weather stations:



USGS-R Packages

USGS has a very nice collection of R packages that wrap USGS open data

APIs: <https://owi.usgs.gov/R/>

"USGS-R is a community of support for users of the R scientific programming language. USGS-R resources include R training materials, R tools for the retrieval and analysis of USGS data, and support for a growing group of USGS-R developers."

USGS R packages include:

- `dataRetrieval`: Obtain water quality sample data, streamflow data, and metadata directly from either the USGS or EPA
- `EGRET`: Analysis of long-term changes in water quality and streamflow, including the water-quality method Weighted Regressions on Time, Discharge, and Season (WRTDS)
- `laketemps`: Lake temperature data package for Global Lake Temperature Collaboration Project
- `lakeattributes`: Common useful lake attribute data
- `soilmoisturetools`: Tools for soil moisture data retrieval and visualization

Other R API wrappers

Here are some examples of other R packages that facilitate use of an API for open data:

- `twitterR`: Twitter
- `Quandl`: Quandl (financial data)
- `RGoogleAnalytics`: Google Analytics
- `WDI`, `wbstats`: World Bank
- `GuardianR`, `rdian`: The Guardian Media Group
- `blsAPI`: Bureau of Labor Statistics
- `rtimes`: New York Times

Find out more about writing API packages with this vignette for the httr package: <https://cran.r-project.org/web/packages/httr/vignettes/api-packages.html>.

This document includes advice on error handling within R code that accesses data through an open API.

We'll take a break now to do the first part of the In-Course Exercise.

Cleaning very messy data

Hurricane tracking data

One version of Atlantic basin hurricane tracks is available here:
<https://www.nhc.noaa.gov/data/hurdat/hurdat2-1851-2017-050118.txt>.
The data is not in a classic delimited format:

```
AL011851,          UNNAMED,      14,
18510625, 0000,    , HU, 28.0N,  94.8W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 0600,    , HU, 28.0N,  95.4W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 1200,    , HU, 28.0N,  96.0W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 1800,    , HU, 28.1N,  96.5W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 2100, L,  HU, 28.2N,  96.8W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 0000,    , HU, 28.2N,  97.0W,  70, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 0600,    , TS, 28.3N,  97.6W,  60, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 1200,    , TS, 28.4N,  98.3W,  60, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 1800,    , TS, 28.6N,  98.9W,  50, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 0000,    , TS, 29.0N,  99.4W,  50, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 0600,    , TS, 29.5N,  99.8W,  40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 1200,    , TS, 30.0N, 100.0W, 40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 1800,    , TS, 30.5N, 100.1W, 40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510628, 0000,    , TS, 31.0N, 100.2W, 40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
AL021851,          UNNAMED,      1,
18510705, 1200,    , HU, 22.2N,  97.6W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
AL031851,          UNNAMED,      1,
18510710, 1200,    , TS, 12.0N,  60.0W,  50, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
AL041851,          UNNAMED,      49,
18510816, 0000,    , TS, 13.4N,  48.0W,  40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510816, 0600,    , TS, 13.7N,  49.5W,  40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510816, 1200,    , TS, 14.0N,  51.0W,  50, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510816, 1800,    , TS, 14.4N,  52.8W,  50, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510817, 0000,    , TS, 14.9N,  54.6W,  60, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510817, 0600,    , TS, 15.4N,  56.5W,  60, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
-----
```

Hurricane tracking data

This data is formatted in the following way:

- Data for many storms are included in one file.
- Data for a storm starts with a shorter line, with values for the storm ID, name, and number of observations for the storm. These values are comma separated.
- Observations for each storm are longer lines. There are multiple observations for each storm, where each observation gives values like the location and maximum winds for the storm at that time.

Hurricane tracking data

Strategy for reading in very messy data:

1. Read in all lines individually.
2. Use regular expressions to split each line into the elements you'd like to use to fill columns.
3. Write functions and `/` or `map` calls to process lines and use the contents to fill a data frame.
4. Once you have the data in a data frame, do any remaining cleaning to create a data frame that is easy to use to answer research questions.

Hurricane tracking data

Because the data is not nicely formatted, you can't use `read_csv` or similar functions to read it in.

However, the `read_lines` function from `readr` allows you to read a text file in one line at a time. You can then write code and functions to parse the file one line at a time, to turn it into a dataframe you can use.

Note: Base R has `readLines`, which is very similar.

Hurricane tracking data

The `read_lines` function from `readr` will read in lines from a text file directly, without trying to separate into columns. You can use the `n_max` argument to specify the number of lines to read it.

For example, to read in three lines from the hurricane tracking data, you can run:

```
tracks_url <- paste0("http://www.nhc.noaa.gov/data/hurdat/",  
                     "hurdat2-1851-2017-050118.txt")  
hurr_tracks <- read_lines(tracks_url, n_max = 3)  
hurr_tracks  
  
## [1] "AL011851,          UNNAMED,      14,"  
## [2] "18510625, 0000,    , HU, 28.0N,  94.8W,  80, -999, -999, -"  
## [3] "18510625, 0600,    , HU, 28.0N,  95.4W,  80, -999, -999, -"
```

Hurricane tracking data

The data has been read in as a vector, rather than a dataframe:

```
class(hurr_tracks)
```

```
## [1] "character"
```

```
length(hurr_tracks)
```

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

```
hurr_tracks[1]
```

```
## [1] "AL011851, UNNAMED, 14,"
```

Hurricane tracking data

You can use regular expressions to break each line up. For example, you can use `str_split` from the `stringr` package to break the first line of the hurricane track data into its three separate components:

```
library(stringr)
str_split(hurr_tracks[1], pattern = ",")

## [[1]]
## [1] "AL011851"      "                "      "UNNAMED" "      "14"
## [4] ""
```


Hurricane tracking data

You can use this to create a list where each element of the list has the split-up version of a line of the original data. First, read in all of the data:

```
tracks_url <- paste0("http://www.nhc.noaa.gov/data/hurdat/",  
                    "hurdat2-1851-2017-050118.txt")  
hurr_tracks <- read_lines(tracks_url)  
length(hurr_tracks)  
  
## [1] 52151
```

Hurricane tracking data

Next, use `map` with `str_split` to split each line of the data at the commas:

```
library(purrr)
hurr_tracks <- map(hurr_tracks, str_split,
                   pattern = ",", simplify = TRUE)
hurr_tracks[[1]]

##           [,1]           [,2]           [,3]           [,4]
## [1,] "AL011851" "          UNNAMED" "          14" ""
hurr_tracks[[2]][1:2]

## [1] "18510625" " 0000"
```

Hurricane tracking data

Next, you want to split this list into two lists, one with the shorter “meta-data” lines and one with the longer “observation” lines. You can use `map_int` to create a vector with the length of each line. You will later use this to identify which lines are short or long.

```
hurr_lengths <- map_int(hurr_tracks, length)
hurr_lengths[1:17]
```

```
## [1] 4 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 4 21
```

```
unique(hurr_lengths)
```

```
## [1] 4 21
```

Hurricane tracking data

You can use bracket indexing to split the `hurr_tracks` into two lists: one with the shorter lines that start each observation (`hurr_meta`) and one with the storm observations (`hurr_obs`). Use bracket indexing with the `hurr_lengths` vector you just created to make that split.

```
hurr_meta <- hurr_tracks[hurr_lengths == 4]  
hurr_obs <- hurr_tracks[hurr_lengths == 21]
```

Hurricane tracking data

```
hurr_meta[1:3]
```

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

```
##           [,1]           [,2]                [,3]           [,4]
```

```
## [1,] "AL011851" "                UNNAMED" "          14" ""
```

```
##
```

```
## [[2]]
```

```
##           [,1]           [,2]                [,3]           [,4]
```

```
## [1,] "AL021851" "                UNNAMED" "           1" ""
```

```
##
```

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

```
##           [,1]           [,2]                [,3]           [,4]
```

```
## [1,] "AL031851" "                UNNAMED" "           1" ""
```

Hurricane tracking data

```
hurr_obs[1:2]
```

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

```
##      [,1]      [,2]      [,3] [,4]  [,5]      [,6]      [,7]
## [1,] "18510625" " 0000" "  " " HU" " 28.0N" " 94.8W" " 80"
##      [,9]      [,10]     [,11]    [,12]    [,13]    [,14]    [,15]
## [1,] " -999" " -999" " -999" " -999" " -999" " -999" " -999"
##      [,17]     [,18]     [,19]    [,20]    [,21]
## [1,] " -999" " -999" " -999" " -999" ""
##
```

```
## [[2]]
```

```
##      [,1]      [,2]      [,3] [,4]  [,5]      [,6]      [,7]
## [1,] "18510625" " 0600" "  " " HU" " 28.0N" " 95.4W" " 80"
##      [,9]      [,10]     [,11]    [,12]    [,13]    [,14]    [,15]
## [1,] " -999" " -999" " -999" " -999" " -999" " -999" " -999"
##      [,17]     [,18]     [,19]    [,20]    [,21]
## [1,] " -999" " -999" " -999" " -999" ""
```

Hurricane tracking data

Now, you can use `bind_rows` from `dplyr` to change the list of metadata into a dataframe. (You first need to use `as_tibble` with `map` to convert all elements of the list from matrices to dataframes.)

```
library(dplyr); library(tibble)
hurr_meta <- hurr_meta %>%
  map(as_tibble) %>%
  bind_rows()
hurr_meta %>% slice(1:3)
```

```
## # A tibble: 3 x 4
##   V1          V2          V3          V4
##   <chr>      <chr>      <chr>      <chr>
## 1 AL011851 "          UNNAMED" "      14" ""
## 2 AL021851 "          UNNAMED" "       1" ""
## 3 AL031851 "          UNNAMED" "       1" ""
```

Hurricane tracking data

You can clean up the data a bit more.

- First, the fourth column doesn't have any non-missing values, so you can get rid of it:

```
unique(hurr_meta$V4)
```

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

- Second, the second and third columns include a lot of leading whitespace:

```
hurr_meta$V2[1:2]
```

```
## [1] "          UNNAMED" "          UNNAMED"
```

- Last, we want to name the columns.

Hurricane tracking data

```
hurr_meta <- hurr_meta %>%  
  select(-V4) %>%  
  rename(storm_id = V1, storm_name = V2, n_obs = V3) %>%  
  mutate(storm_name = str_trim(storm_name),  
         n_obs = as.numeric(n_obs))  
hurr_meta %>% slice(1:3)
```

```
## # A tibble: 3 x 3  
##   storm_id storm_name n_obs  
##   <chr>      <chr>      <dbl>  
## 1 AL011851 UNNAMED      14  
## 2 AL021851 UNNAMED       1  
## 3 AL031851 UNNAMED       1
```

Hurricane tracking data

Now you can do the same idea with the hurricane observations. First, we'll want to add storm identifiers to that data. The “meta” data includes storm ids and the number of observations per storm. We can take advantage of that to make a `storm_id` vector that will line up with the storm observations.

```
storm_id <- rep(hurr_meta$storm_id, times = hurr_meta$n_obs)
head(storm_id, 3)
```

```
## [1] "AL011851" "AL011851" "AL011851"
```

```
length(storm_id)
```

```
## [1] 50303
```

```
length(hurr_obs)
```

```
## [1] 50303
```

Hurricane tracking data

```
hurr_obs <- hurr_obs %>%  
  map(as_tibble) %>%  
  bind_rows() %>%  
  mutate(storm_id = storm_id)  
hurr_obs %>% select(V1:V2, V5:V6, storm_id) %>% slice(1:3)
```

```
## # A tibble: 3 x 5
```

##	V1	V2	V5	V6	storm_id
##	<chr>	<chr>	<chr>	<chr>	<chr>
## 1	18510625	" 0000"	" 28.0N"	" 94.8W"	AL011851
## 2	18510625	" 0600"	" 28.0N"	" 95.4W"	AL011851
## 3	18510625	" 1200"	" 28.0N"	" 96.0W"	AL011851

We'll take a break now to do the next part of the In-Course Exercise.