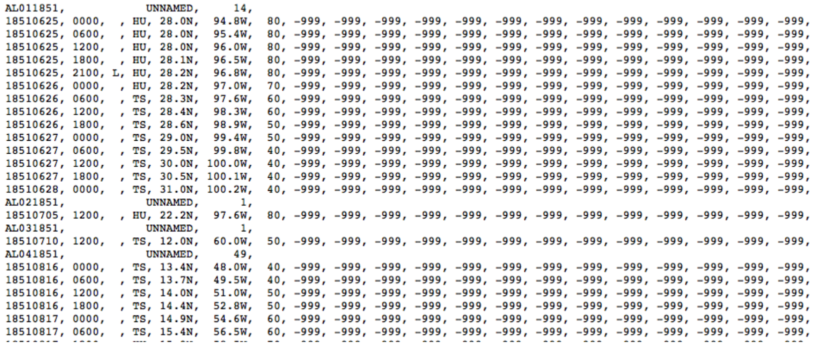
Entering and cleaning data #3

Cleaning very messy data

Hurricane tracking data

One version of Atlantic basin hurricane tracks is available here: `http://www.nhc.noaa.gov/data/hurdat/hurdat2-1851-2015-070616.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,
18510625, 0600,  , HU, 28.0N,  95.4W,  80, -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,
18510625, 1800,  , HU, 28.1N,  96.5W,  80, -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,
18510626, 0000,  , HU, 28.2N,  97.0W,  70, -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,
18510626, 1200,  , TS, 28.4N,  98.3W,  60, -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,
18510627, 0000,  , TS, 29.0N,  99.4W,  50, -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,
18510627, 1200,  , TS, 30.0N, 100.0W,  40, -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,
18510628, 0000,  , TS, 31.0N, 100.2W,  40, -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,
AL031851,            UNNAMED,      1,
18510710, 1200,  , TS, 12.0N,  60.0W,  50, -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,
18510816, 0600,  , TS, 13.7N,  49.5W,  40, -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,
18510816, 1800,  , TS, 14.4N,  52.8W,  50, -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,
18510817, 0600,  , TS, 15.4N,  56.5W,  60, -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, loops, or `apply` 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 `readLines` function 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.

Hurricane tracking data

The `readLines` function will read in lines from a text file directly, without trying to separate into columns. You can use the `n` 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:

```r
tracks_url <- paste0("http://www.nhc.noaa.gov/data/hurdat/",
                     "hurdat2-1851-2015-070616.txt")
hurr_tracks <- readLines(tracks_url, n = 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/",
                     "hurdat2-1851-2015-070616.txt")
hurr_tracks <- readLines(tracks_url)
length(hurr_tracks)
```

```
## [1] 50919
```

Hurricane tracking data

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

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

```
##  [1]  4 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

```r
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 lapply to convert all elements of the list from matrices to dataframes.)

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

```
## # A tibble: 3 × 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 × 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] 49105
```

```
length(hurr_obs)
```

```
## [1] 49105
```

Hurricane tracking data

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

```
## # A tibble: 3 × 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
```

Hurricane tracking data

To finish, you just need to clean up the data. Now that the data is in a dataframe, this process is inline with what you've been doing with `dplyr` and related packages.

The "README" file for the hurricane tracking data is useful at this point:

`http://www.nhc.noaa.gov/data/hurdat/hurdat2-format-atlantic.pdf`

Hurricane tracking data

First, say you only want some of the columns for a study you are doing. You can use `select` to clean up the dataframe by limiting it to columns you need.

If you only need date, time, storm status, location (latitude and longitude), maximum sustained winds, and minimum pressure, then you can run:

```
hurr_obs <- hurr_obs %>%
  select(V1, V2, V4:V8, storm_id) %>%
  rename(date = V1, time = V2, status = V4, latitude = V5,
         longitude = V6, wind = V7, pressure = V8)
hurr_obs %>% slice(1:3) %>%
  select(date, time, status, latitude, longitude)
```

```
## # A tibble: 3 × 5
##       date  time status latitude longitude
##      <chr> <chr>  <chr>    <chr>     <chr>
## 1 18510625  0000     HU    28.0N     94.8W
## 2 18510625  0600     HU    28.0N     95.4W
```

Hurricane tracking data

Next, the first two columns give the date and time. You can `unite` these and then convert them to a Date-time class.

```r
library(tidyr)
library(lubridate)
hurr_obs <- hurr_obs %>%
  unite(date_time, date, time) %>%
  mutate(date_time = ymd_hm(date_time))
hurr_obs %>% slice(1:3) %>%
  select(date_time, status, latitude, longitude)
```

```
## # A tibble: 3 × 4
##             date_time status latitude longitude
##                <dttm>  <chr>    <chr>     <chr>
## 1 1851-06-25 00:00:00     HU    28.0N     94.8W
## 2 1851-06-25 06:00:00     HU    28.0N     95.4W
## 3 1851-06-25 12:00:00     HU    28.0N     96.0W
```

Hurricane tracking data

Next, you can change status to a factor and give the levels more meaningful names:

```
unique(hurr_obs$status)
```

```
## [1] " HU" " TS" " EX" " TD" " LO" " DB" " SD" " SS" " WV"
```

```
storm_levels <- c("TD", "TS", "HU", "EX",
                  "SD", "SS", "LO", "WV", "DB")
storm_labels <- c("Tropical depression", "Tropical storm",
                  "Hurricane", "Extratropical cyclone",
                  "Subtropical depression",
                  "Subtropical storm", "Other low",
                  "Tropical wave", "Disturbance")
hurr_obs <- hurr_obs %>%
  mutate(status = factor(str_trim(status),
                         levels = storm_levels,
                         labels = storm_labels))
```

Hurricane tracking data

Now, you can clean up the latitude and longitude. Ultimately, we'll want numeric values for those so we can use them for mapping. You can use regular expressions to separate the numeric and non-numeric parts of these columns. For example:

```
head(str_extract(hurr_obs$latitude, "[A-Z]"))
```

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

```
head(str_extract(hurr_obs$latitude, "[^A-Z]+"))
```

```
## [1] " 28.0" " 28.0" " 28.0" " 28.1" " 28.2" " 28.2"
```

Hurricane tracking data

Use this idea to split the numeric latitude from the direction of that latitude:

```r
hurr_obs <- hurr_obs %>%
  mutate(lat_dir = str_extract(latitude, "[A-Z]"),
         latitude = as.numeric(str_extract(latitude,
                                           "[^A-Z]+")),
         lon_dir = str_extract(longitude, "[A-Z]"),
         longitude = as.numeric(str_extract(longitude,
                                            "[^A-Z]+")))
```

Hurricane tracking data

Now these elements are in separate columns:

```
hurr_obs %>%
  select(latitude, lat_dir, longitude, lon_dir) %>%
  slice(1:2)
```

```
## # A tibble: 2 × 4
##   latitude lat_dir longitude lon_dir
##      <dbl>   <chr>     <dbl>   <chr>
## 1       28       N      94.8       W
## 2       28       N      95.4       W
```

```
unique(hurr_obs$lat_dir)
```

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

```
unique(hurr_obs$lon_dir)
```

```
## [1] "W" "E"
```

Hurricane tracking data

If we're looking at US impacts, we probably only need observations from the western hemisphere, so let's filter out other values:

```
hurr_obs <- hurr_obs %>%
  filter(lon_dir == "W")
```

Hurricane tracking data

Next, clean up the wind column:

```
unique(hurr_obs$wind)[1:5]
```

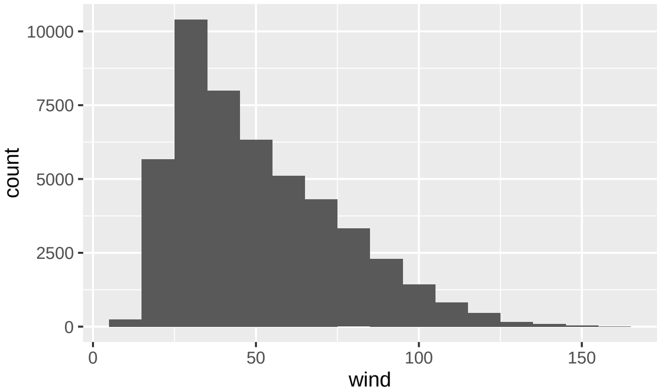
```
## [1] "  80" "  70" "  60" "  50" "  40"
```

```
hurr_obs <- hurr_obs %>%
  mutate(wind = ifelse(wind == " -99", NA,
                       as.numeric(wind)))
```

Hurricane tracking data

Check the cleaned measurements:

```
library(ggplot2)
ggplot(hurr_obs, aes(x = wind)) +
  geom_histogram(binwidth = 10)
```



Hurricane tracking data

Clean and check air pressure measurements in the same way:

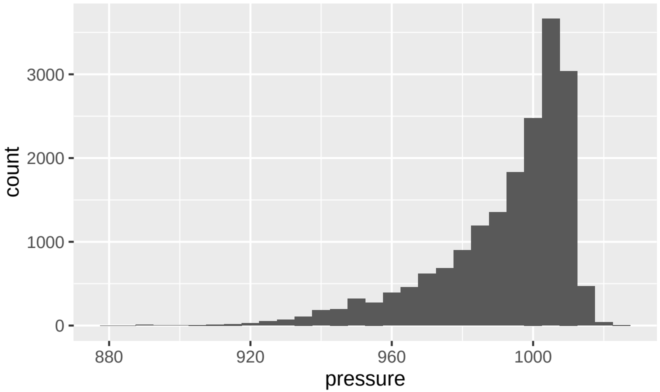
```
head(unique(hurr_obs$pressure))
```

```
## [1] " -999" "  961" "  924" "  938" "  950" "  997"
```

```
hurr_obs <- hurr_obs %>%
  mutate(pressure = ifelse(pressure == " -999", NA,
                           as.numeric(pressure)))
```

Hurricane tracking data

```
ggplot(hurr_obs, aes(x = pressure)) +
  geom_histogram(binwidth = 5)
```



Hurricane tracking data

Check some of the very low pressure measurements:

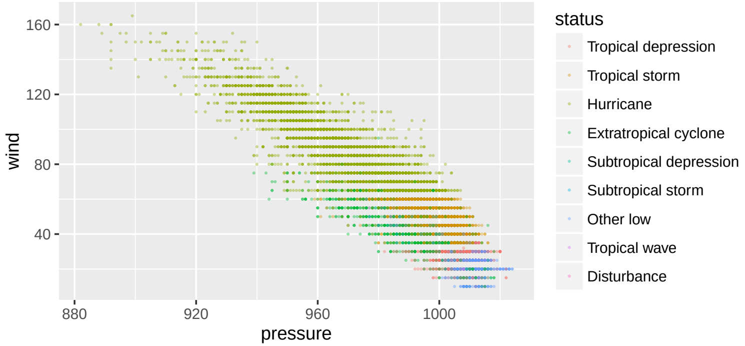
```r
hurr_obs %>% arrange(pressure) %>%
  select(date_time, wind, pressure) %>% slice(1:5)
```

```
## # A tibble: 5 × 3
##              date_time  wind pressure
##                 <dttm> <dbl>    <dbl>
## 1 2005-10-19 12:00:00   160      882
## 2 1988-09-14 00:00:00   160      888
## 3 1988-09-14 06:00:00   155      889
## 4 1935-09-03 00:00:00   160      892
## 5 1935-09-03 02:00:00   160      892
```

Hurricane tracking data

Explore pressure versus wind speed, by storm status:

```r
ggplot(hurr_obs, aes(x = pressure, y = wind,
                     color = status)) +
  geom_point(size = 0.2, alpha = 0.4)
```



Hurricane tracking data

Next, we want to map storms by decade. Add hurricane decade:

```r
hurr_obs <- hurr_obs %>%
  mutate(decade = substring(year(date_time), 1, 3),
         decade = paste0(decade, "0s"))
unique(hurr_obs$decade)
```

```
##  [1] "1850s" "1860s" "1870s" "1880s" "1890s" "1900s" "1910s"
##  [9] "1930s" "1940s" "1950s" "1960s" "1970s" "1980s" "1990s"
## [17] "2010s"
```

Add logical for whether the storm was ever category 5:

```r
hurr_obs <- hurr_obs %>%
  group_by(storm_id) %>%
  mutate(cat_5 = max(wind) >= 137) %>%
  ungroup()
```

Hurricane tracking data

To map the hurricane tracks, you need a base map to add the tracks to. Pull data to map hurricane-prone states:

```
east_states <- c("florida", "georgia", "south carolina",
                 "north carolina", "virginia", "maryland",
                 "delaware", "new jersey", "new york",
                 "connecticut", "massachusetts",
                 "rhode island", "vermont", "new hampshire",
                 "maine", "pennsylvania", "west virginia",
                 "tennessee", "kentucky", "alabama",
                 "arkansas", "texas", "mississippi",
                 "louisiana")
east_us <- map_data("state", region = east_states)
```

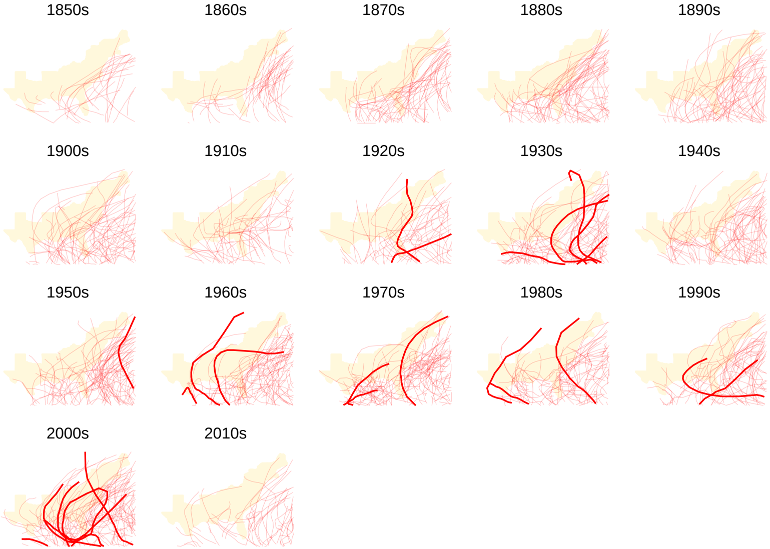
Hurricane tracking data

Plot tracks over a map of hurricane-prone states. Add thicker lines for storms that were category 5 at least once in their history.

```r
ggplot(east_us, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "cornsilk", color = "cornsilk") +
  theme_void() +
  xlim(c(-108, -65)) + ylim(c(23, 48)) +
  geom_path(data = hurr_obs,
            aes(x = -longitude, y = latitude,
                group = storm_id),
            color = "red", alpha = 0.2, size = 0.2) +
  geom_path(data = filter(hurr_obs, cat_5),
            aes(x = -longitude, y = latitude,
                group = storm_id),
            color = "red") +
  facet_wrap(~ decade)
```

Hurricane tracking data

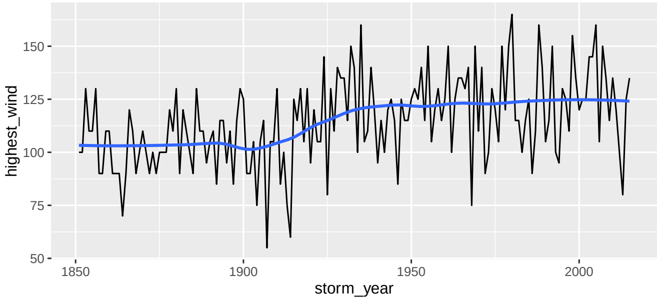
Check trends in maximum wind recorded in any observation each year:



Hurricane tracking data

Maximum wind observed each year:

```r
hurr_obs %>%
  mutate(storm_year = year(date_time)) %>%
  group_by(storm_year) %>%
  summarize(highest_wind = max(wind, na.rm = TRUE)) %>%
  ggplot(aes(x = storm_year, y = highest_wind)) +
  geom_line() + geom_smooth(se = FALSE, span = 0.5)
```



Hurricane tracking data

There is an R package named `gender` that predicts whether a name is male or female based on historical data:

Vignette for `gender` package

This package uses one of several databases of names (here, we'll use Social Security Administration data), inputs a year or range of years, and outputs whether a name in that year was more likely female or male.

We can apply a function from this package across all the named storms to see how male / female proportions changed over time.

Hurricane tracking data

First, install the package (as wll as `genderdata`, which is required to use the package). Once you do, you can use `gender` to determine the most common gender associated with a name in a given year or range of years:

```
# install.packages("gender")
# install.packages("genderdata", type = "source",
#                  repos = "http://packages.ropensci.org")
library(gender)
gender("KATRINA", years = 2005)[ , c("name", "gender")]
```

```
## # A tibble: 1 × 2
##      name gender
##     <chr>  <chr>
## 1 KATRINA female
```

Hurricane tracking data

To apply this function across all our storms, it helps if we write a small function that "wraps" the gender function and outputs exactly (and only) what we want, in the format we want:

```r
get_gender <- function(storm_name, storm_year){
  storm_gender <- gender(names = storm_name,
                         years = storm_year,
                         method = "ssa")$gender
  if(length(storm_gender) == 0) storm_gender <- NA
  return(storm_gender)
}
```

Hurricane tracking data

Now we can use mapply with this wrapper function to apply it across all our named storms:

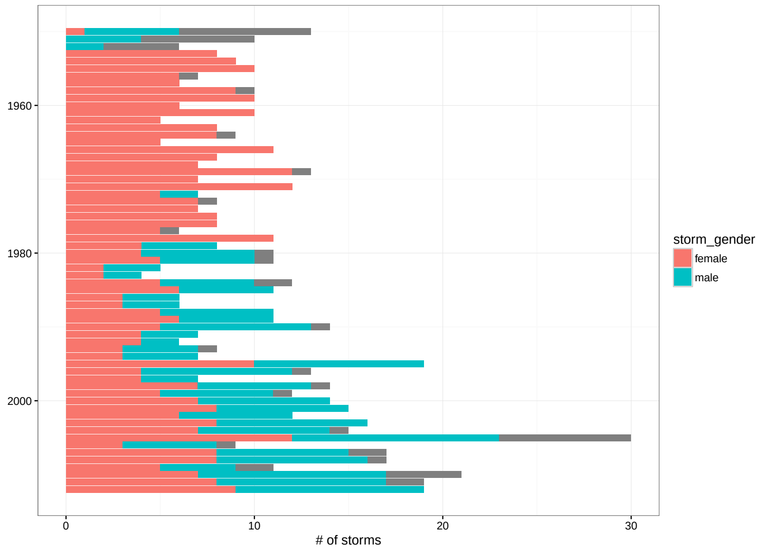
```
hurr_genders <- hurr_meta %>%
  filter(storm_name != "UNNAMED") %>%
  mutate(storm_year = substring(storm_id, 5, 8),
         storm_year = as.numeric(storm_year)) %>%
  filter(1880 <= storm_year & storm_year <= 2012) %>%
  select(storm_name, storm_year, storm_id) %>%
  mutate(storm_gender = mapply(get_gender,
                               storm_name = storm_name,
                               storm_year =
                                 as.numeric(storm_year)))
```

Hurricane tracking data

Now, plot a bar chart with the number of male, female, and unclear storms each year:

```r
hurr_genders %>%
  group_by(storm_year, storm_gender) %>%
  summarize(n = n()) %>%
  ggplot(aes(x = storm_year, y = n, fill = storm_gender)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  scale_x_reverse() +
  theme_bw() +
  xlab("") + ylab("# of storms")
```

Hurricane tracking data



Hurricane tracking data

Next, you can write a function to plot the track for a specific storm. You'll want to be able to call the function by storm name and year, so join in the storm names from the `hurr_meta` dataset. We'll exclude any "UNNAMED" storms.

```
hurr_obs <- hurr_obs %>%
  left_join(hurr_meta, by = "storm_id") %>%
  filter(storm_name != "UNNAMED") %>%
  mutate(storm_year = year(date_time))
```

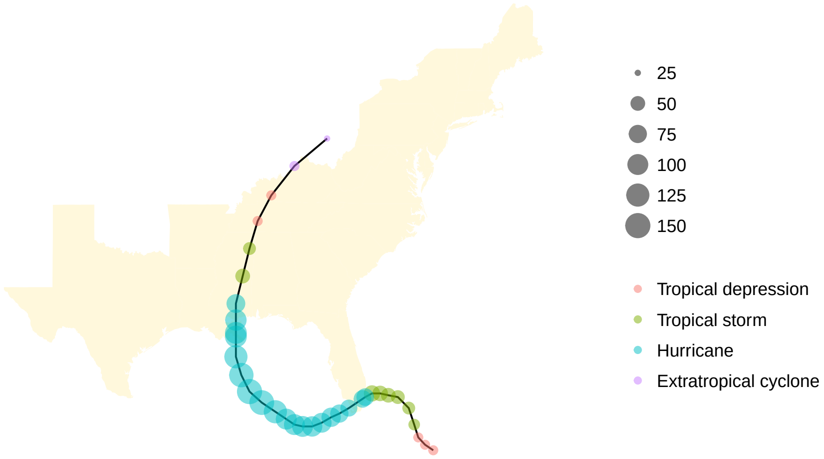
Next, write a function to plot the track for a single storm. Use color to show storm status and size to show wind speed.

Hurricane tracking data

```r
map_track <- function(storm, year, map_data = east_us,
                      hurr_data = hurr_obs){
  to_plot <- hurr_obs %>%
    filter(storm_name == toupper(storm) & storm_year == year)
  out <- ggplot(east_us, aes(x = long, y = lat,
                             group = group)) +
    geom_polygon(fill = "cornsilk") +
    theme_void() +
    xlim(c(-108, -65)) + ylim(c(23, 48)) +
    geom_path(data = to_plot,
              aes(x = -longitude, y = latitude,
                  group = NULL)) +
    geom_point(data = to_plot,
               aes(x = -longitude, y = latitude,
                   group = NULL, color = status,
                   size = wind), alpha = 0.5)
  return(out)
}
```

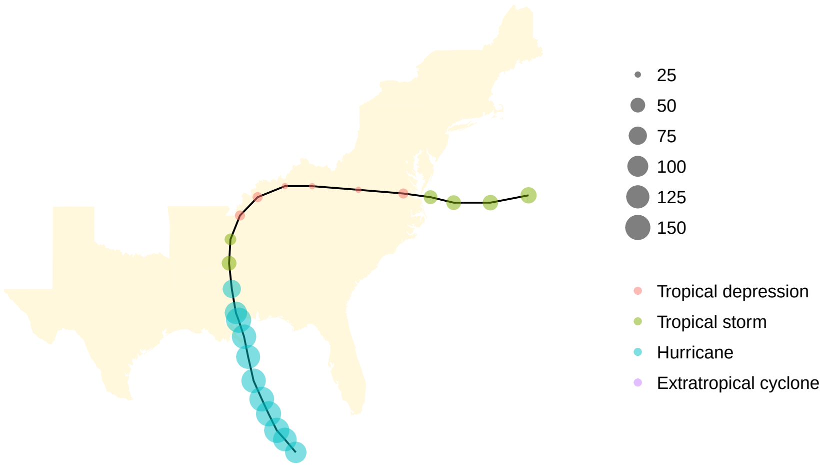
Hurricane tracking data

```r
map_track(storm = "Katrina", year = "2005")
```



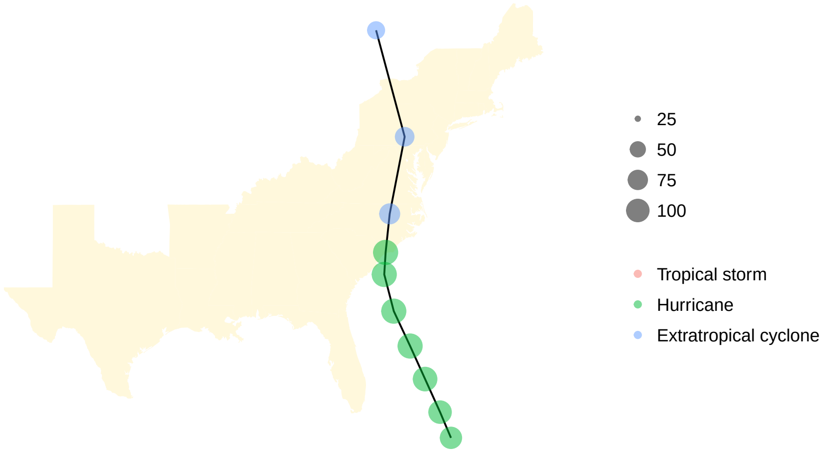
Hurricane tracking data

```
map_track(storm = "Camille", year = "1969")
```



Hurricane tracking data

```
map_track(storm = "Hazel", year = "1954")
```



readLines

You can also write code with `readLines` that will read, check, and clean each line, one line at a time.

```r
con  <- file("~/my_file.txt", open = "r")
while (length(single_line <-
              readLines(con, n = 1,
                        warn = FALSE)) > 0) {

  ## Code to check and clean each line and
  ## then add it to "cleaned" data frame.
  ## Run operations on `single_line`.

  }
close(con)
```

This can be particularly useful if you're cleaning a very big file, especially if there are many lines you don't want to keep.

Pulling online data

APIs

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.

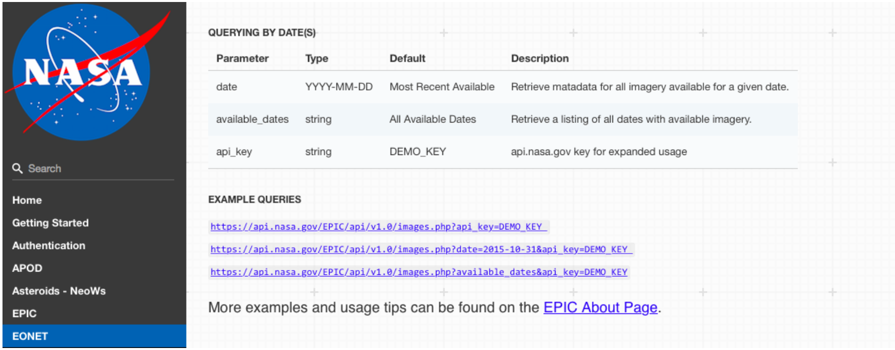
APIs

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 matadata 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`

API key

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.

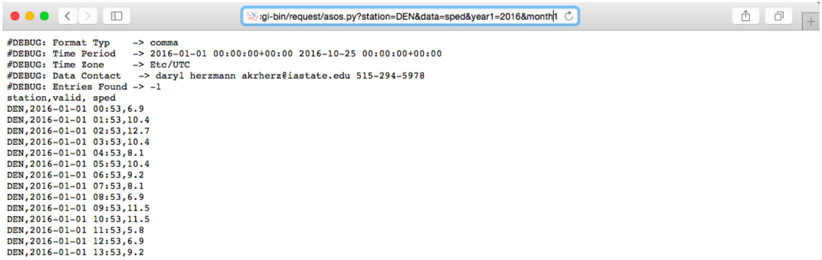
This package allows you to pull weather data from airports around the world directly from the Iowa Environmental Mesonet.

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


:gi-bin/request/asos.py?station=DEN&data=sped&year1=2016&month1

```
#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
```

Example– `riem` package

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` argurment 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"))
```

Example– riem package

You can then use content from httr to retrieve the contents of the HTTP request. 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() %>%
  readr::read_csv(skip = 5, na = "M") %>%
  slice(1:3)
```

```
## # A tibble: 3 × 3
##   station               valid  sped
##     <chr>              <dttm> <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 R API wrappers

rOpenSci

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

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
- `musemeta`: 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.

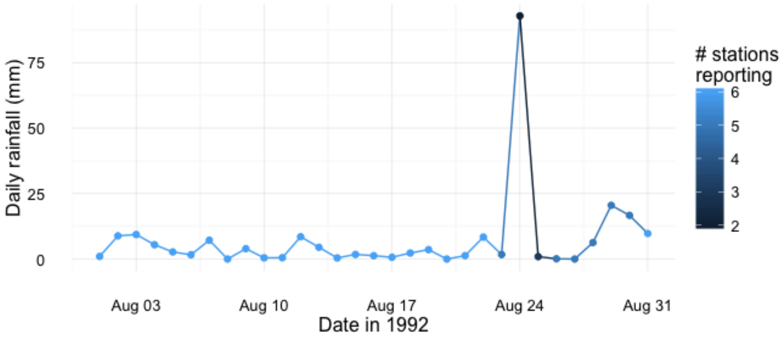
RNOAA

"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)

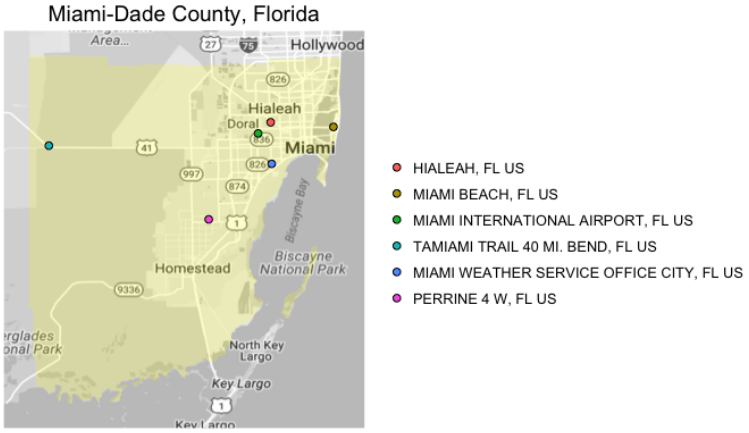
countyweather

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:



countyweather

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

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

US Census packages

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

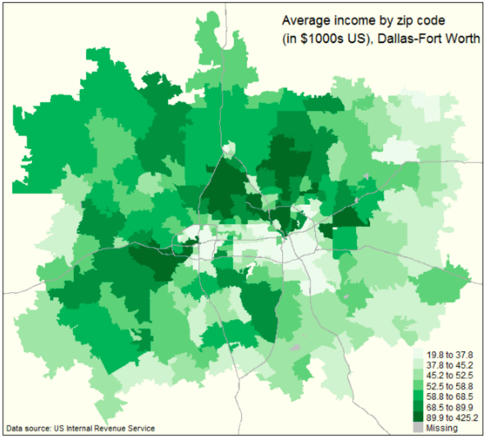
- `tigris`: Download and use Census TIGER/Line shapefiles in R
- `acs`: Download, manipulate, and present American Community Survey and Decennial data from the US Census
- `USABoundaries`: Historical and contemporary boundaries of the United States of America
- `idbr`: R interface to the US Census Bureau International Data Base API

tigris package

- 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

TIGRIS PACKAGE

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


Average income by zip code
(in $1000s US), Dallas-Fort Worth
19.8 to 37.8
37.8 to 45.2
45.2 to 52.5
52.5 to 58.8
58.8 to 68.5
68.5 to 89.9
89.9 to 425.2
Missing
Data source: US Internal Revenue Service

Other R API wrappers

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

- `twitteR`: 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

R and APIs

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.

Parsing webpages

Parsing webpages

You can also use R to pull and clean web-based data that is not accessible through a web API or as an online flat file.

In this case, the strategy is:

- Pull in the full web page file (often in HTML or XML)
- Parse or clean the file within R (e.g., with regular expressions)

RVEST

The rvest package should be the first thing you try if you need to pull and parse data from a webpage that is not a flat file.

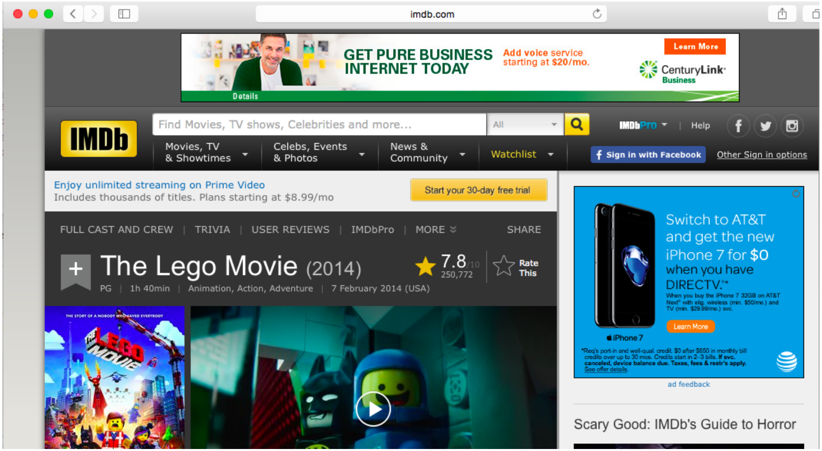
This package allows you to read an HTML or XML file and pull out a certain element. Here is a very simple example of this parsing (this and later examples are from rvest documentation):

```r
library(rvest)
read_html("<html><title>Hi<title></html>")
```

```
## {xml_document}
## <html>
## [1] <head>\n  <title>Hi<title/></title>\n</head>
```

RVEST

If you have an HTML or XML page you want to pull data from, you'll first need to read the page:


imdb.com
GET PURE BUSINESS INTERNET TODAY
Add voice service starting at $20/mo.
Learn More
CenturyLink Business
Details
IMDb
Find Movies, TV shows, Celebrities and more...
All
IMDbPro
Help
Movies, TV & Showtimes
Celebs, Events & Photos
News & Community
Watchlist
Sign in with Facebook
Other Sign in options
Enjoy unlimited streaming on Prime Video
Includes thousands of titles. Plans starting at $8.99/mo
Start your 30-day free trial
FULL CAST AND CREW | TRIVIA | USER REVIEWS | IMDbPro | MORE
SHARE
The Lego Movie (2014)
7.8/10
250,772
Rate This
PG | 1h 40min | Animation, Action, Adventure | 7 February 2014 (USA)
THE STORY OF A NOBODY WHO SAVED EVERYBODY
THE LEGO MOVIE
Switch to AT&T and get the new iPhone 7 for $0 when you have DIRECTV.*
Learn More
iPhone 7
ad feedback
Scary Good: IMDb's Guide to Horror

RVEST

```r
library(rvest)
lego_movie <- read_html("http://www.imdb.com/title/tt1490017/")
lego_movie
```

```
## {xml_document}
## <html xmlns:og="http://ogp.me/ns#" xmlns:fb="http://www.faceb
## [1] <head>\n  <meta charset="utf-8"/>\n  <meta http-equiv="X-
## [2] <body id="styleguide-v2" class="fixed">\n<script><![CDATA
```

RVEST

Then you can use `html_nodes` and `html_text` to pull and parse just the elements you want:

```
rating_node <- lego_movie %>% html_nodes("strong span")
rating_node
```

```
## {xml_nodeset (1)}
## [1] <span itemprop="ratingValue">7.8</span>
```

```
rating <- rating_node %>%
    html_text() %>% as.numeric()
rating
```

```
## [1] 7.8
```

RVEST

You can pull and parse tables:

```
lego_movie %>%
  html_nodes("table") %>% `[[`(1) %>%
  html_table() %>% select(X2) %>% slice(2:8)
```

```
##                  X2
## 1       Will Arnett
## 2 Elizabeth Banks
## 3       Craig Berry
## 4       Alison Brie
## 5     David Burrows
## 6 Anthony Daniels
## 7       Charlie Day
```

RVEST

The only tricky part of this is figuring out which CSS selector you can use to pull a specific element of a webpage.

You can use "Selectorgadget" to help with this. Read the vignette for that tool here: `ftp://cran.r-project.org/pub/R/web/packages/rvest/vignettes/selectorgadget.html`

RVEST

```
cities <- c("denver", "boulder", "fort-collins")

kitchen_addresses <- c()
for(i in 1:length(cities)){
  restaurant_url <- paste0("http://thekitchen.com/the-kitchen-",
                            cities[i])
  restaurant_page <- read_html(restaurant_url)
  address <- restaurant_page %>%
    html_nodes("p:nth-child(2)") %>% html_text()
  kitchen_addresses[i] <- address[1]
}
kitchen_addresses
```

```
## [1] "1530 16th Street (Entrance on Wazee Street)\nDenver, CO
## [2] "1039 Pearl St.,\nBoulder, CO 80302"
## [3] "100 North College Avenue\nFort Collins, CO 80524"
```

RVEST

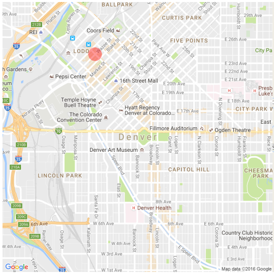
```
library(ggmap)
library(stringr)
kitchen_latlons <- geocode(kitchen_addresses)
```

rvest

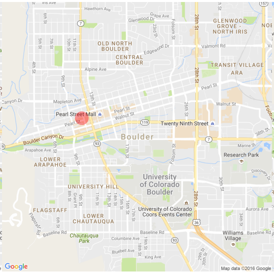
```
for(i in 1:length(cities)){
  city_map <- get_map(paste(gsub("-", " ", cities[i]),
                            "colorado"),
                      zoom = 14,
                      maptype = "roadmap")
  city_map <- ggmap(city_map) +
    geom_point(data = kitchen_latlons[i, ],
               aes(x = lon, y = lat),
               color = "red", size = 4, alpha = 0.4) +
    theme_void() +
    ggtitle(paste("The Kitchen in",
                  str_to_title(gsub("-", " ", cities[i]))))
  print(city_map)
}
```

RVEST

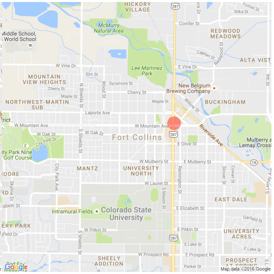
The Kitchen in Denver



The Kitchen in Boulder



The Kitchen in Fort Collins


Fort Collins
Colorado State University
New Belgium Brewing Company
OLD TOWN
Map data ©2016 Google