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, -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, 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:

```
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  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 `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.

```
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:

```
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]
```

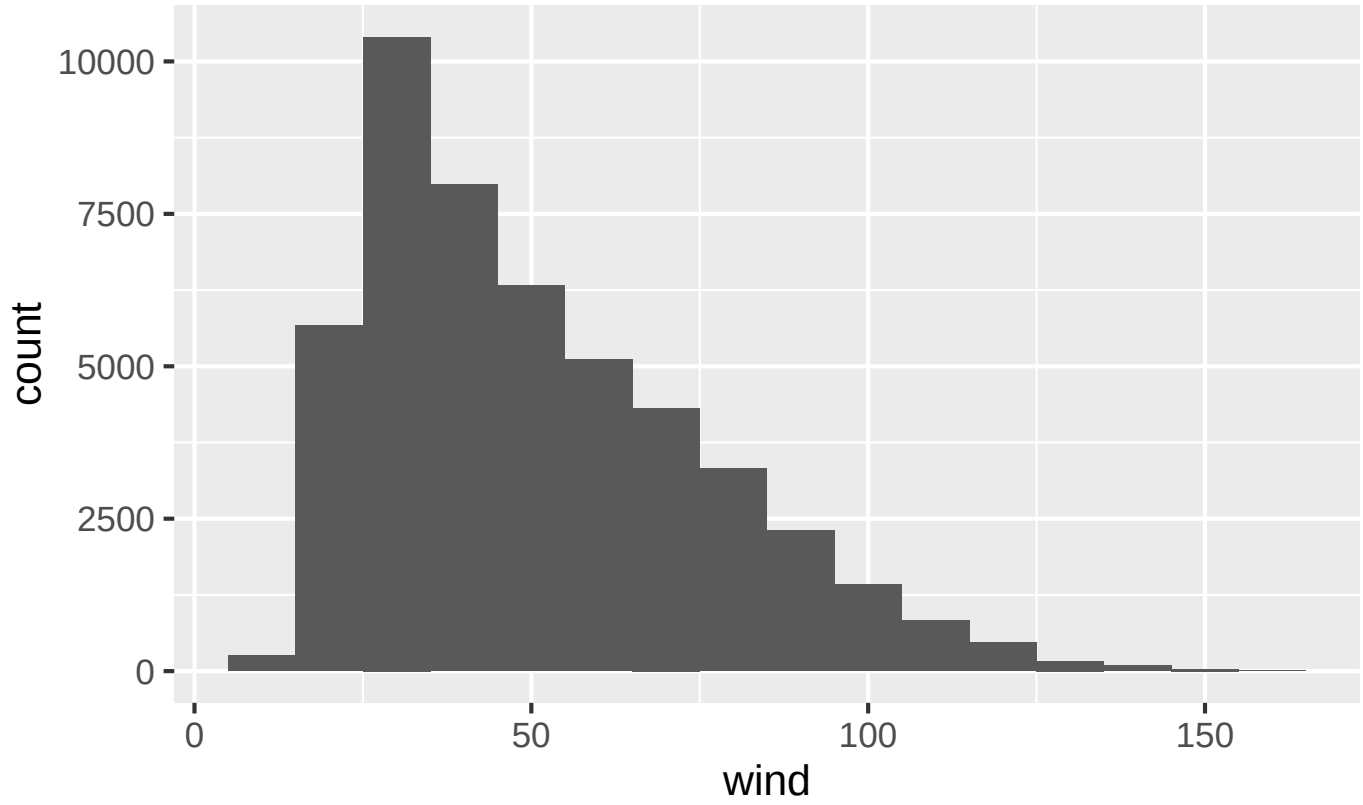
```
## [1] "  80" "  70" "  60" "  50" "  40"
```

```
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:

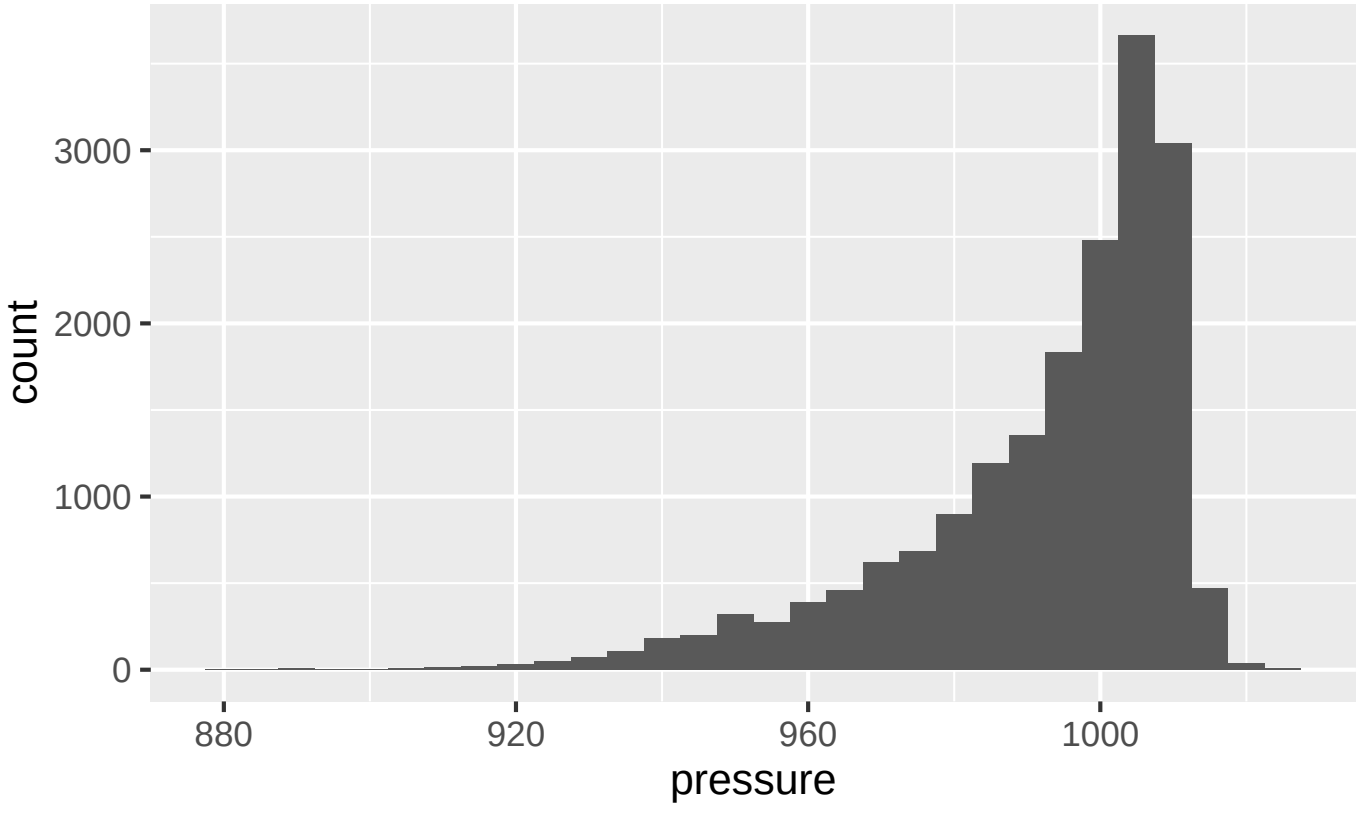
```
head(unique(hurr_obs$pressure))
```

```
## [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:

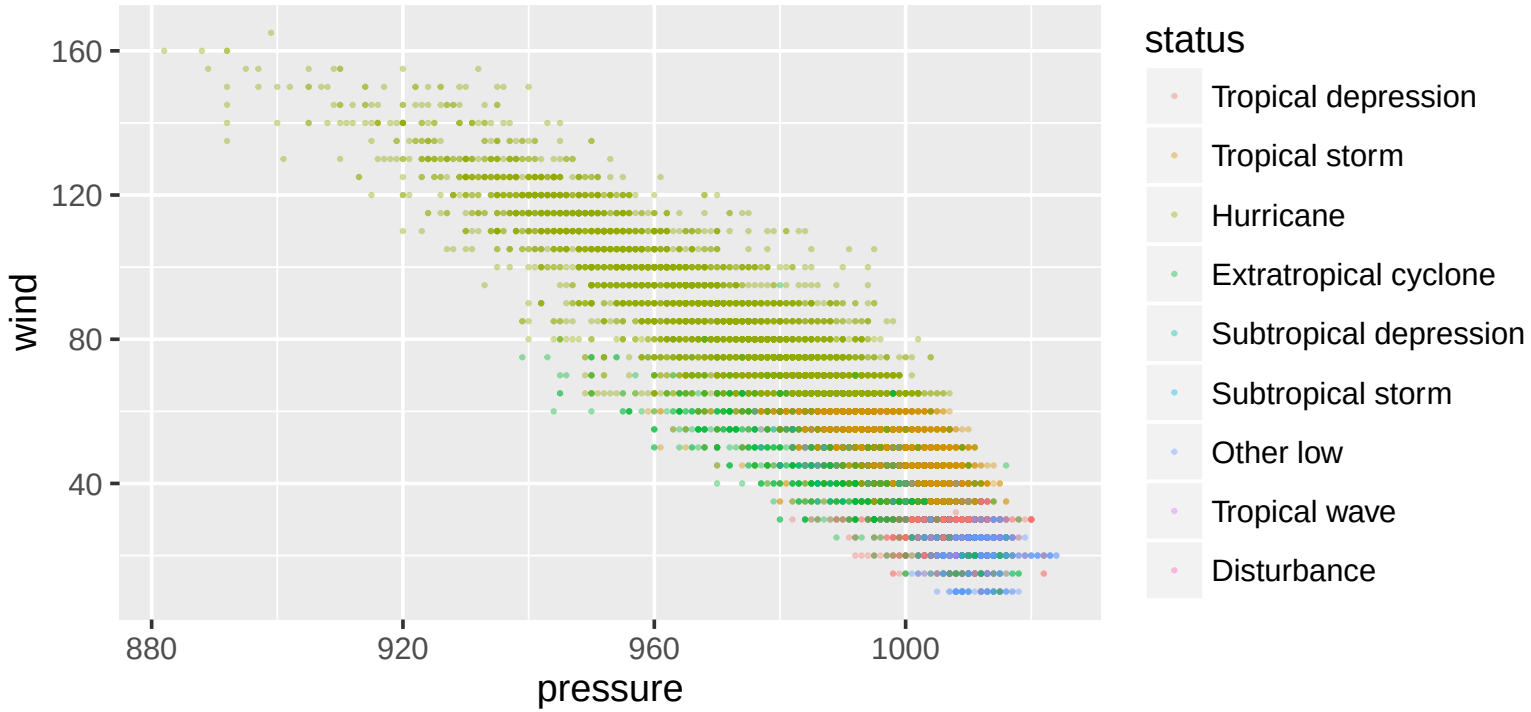
```
hurr_obs %>% arrange(pressure) %>%
  select(date_time, wind, pressure) %>% slice(1:5)
```

```
## # 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:

```
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:

```
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:

```
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)
```

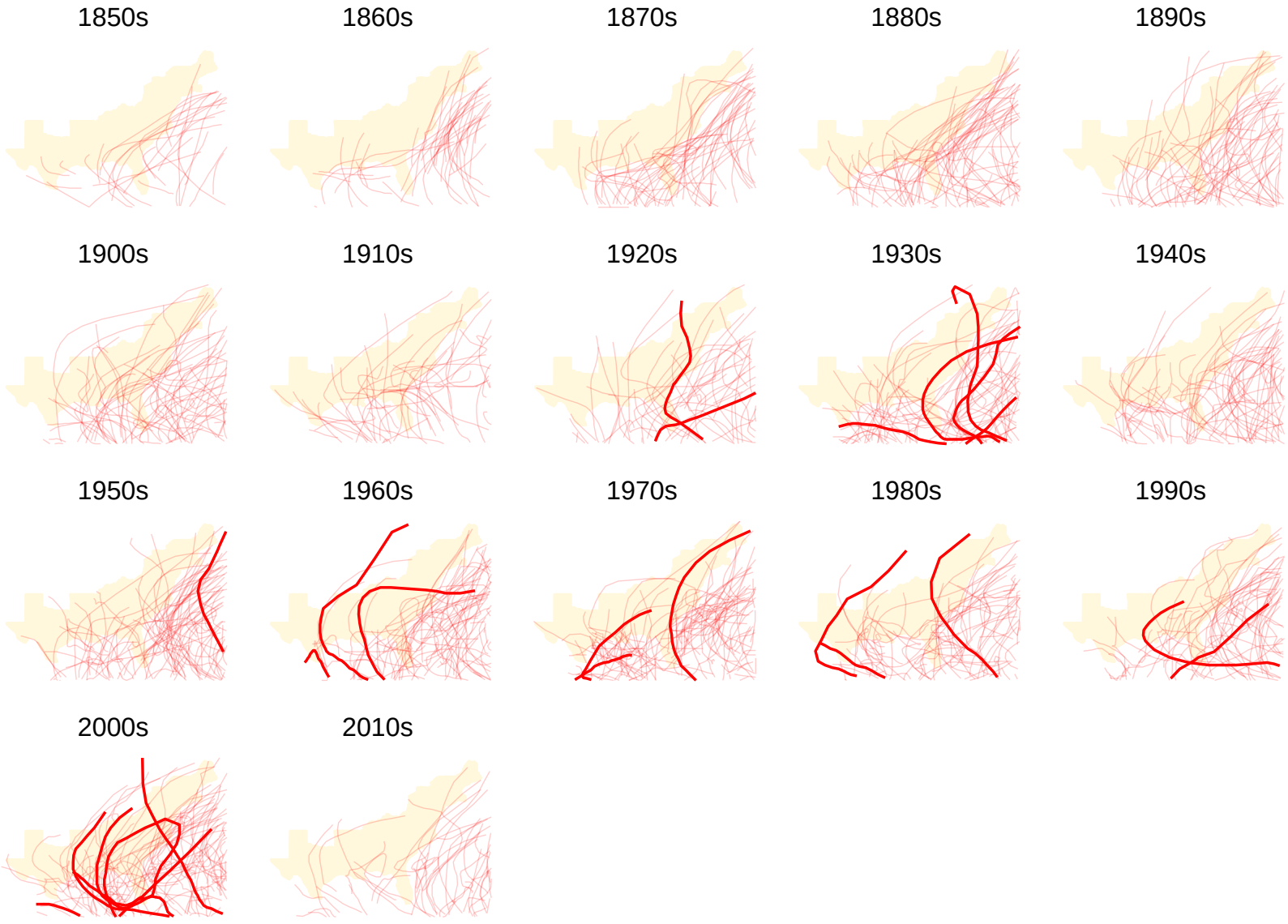
Hurricane tracking data

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

```
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

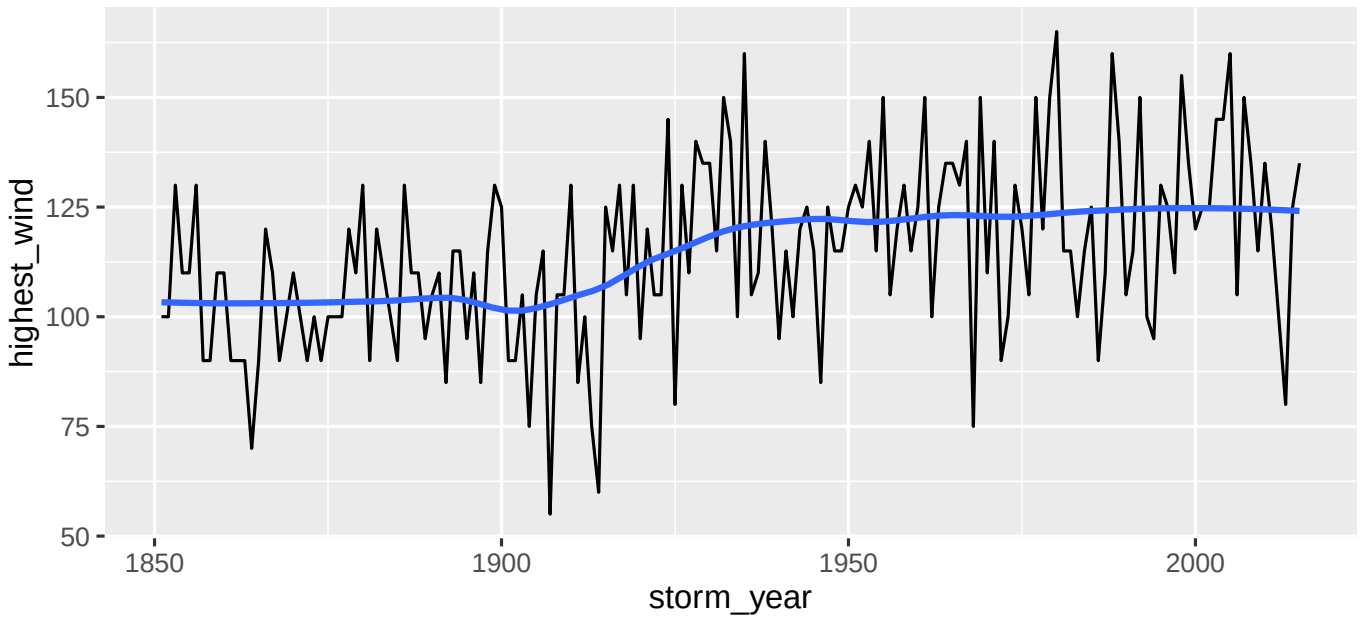
Check trends in maximum wind recorded in any observation each year:



Hurricane tracking data

Maximum wind observed each year:

```
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")
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:

```
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:

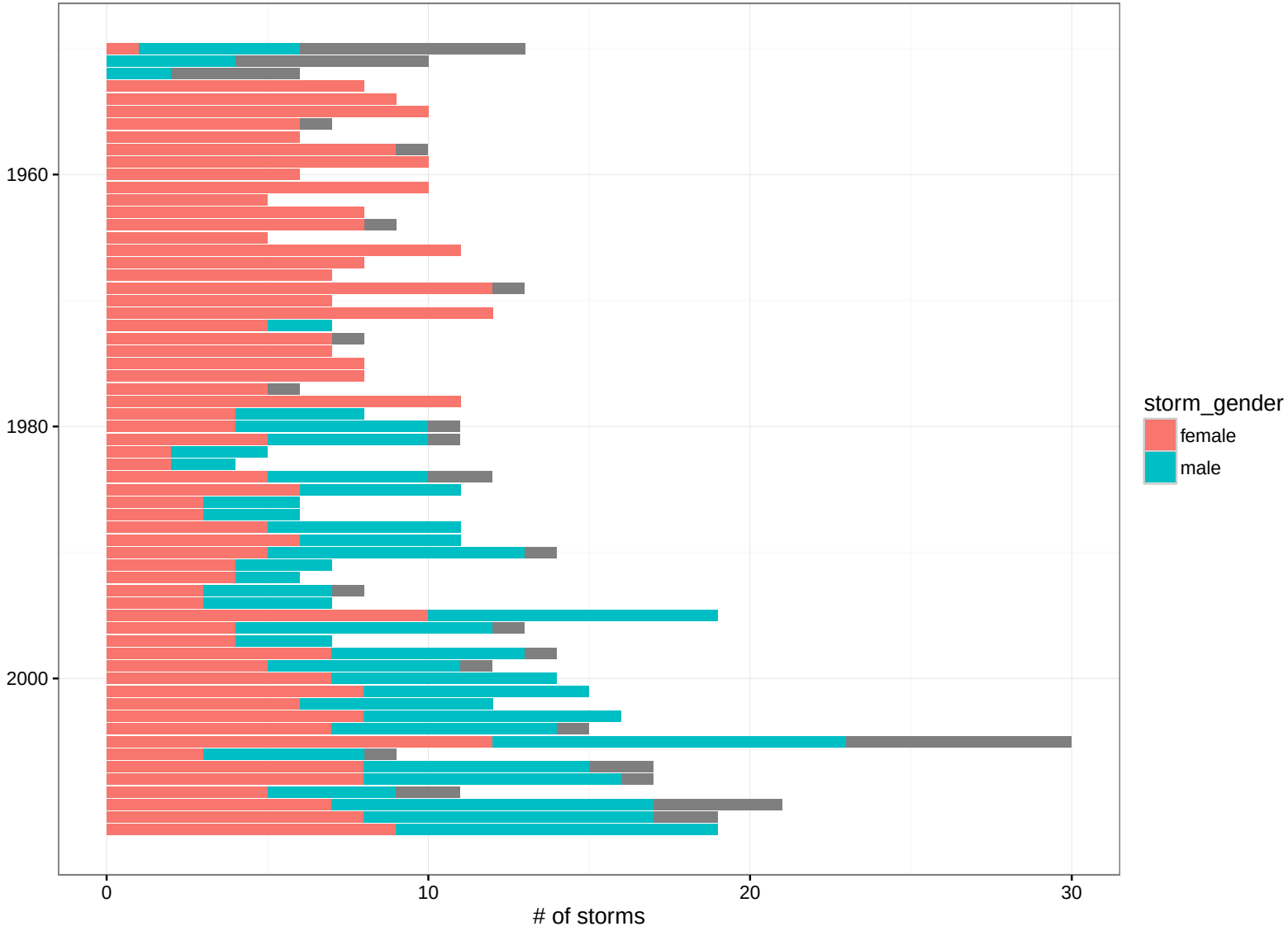
```
hurr_genders <- hurr_meta %>%
  filter(storm_name != "UNNAMED") %>%
  mutate(storm_year = substring(storm_id, 5, 8),
         storm_year = as.numeric(storm_year)) %>%
  filter(1880 <= storm_year & storm_year <= 2012) %>%
  select(storm_name, storm_year, storm_id) %>%
  mutate(storm_gender = mapply(get_gender,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

```
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))
```

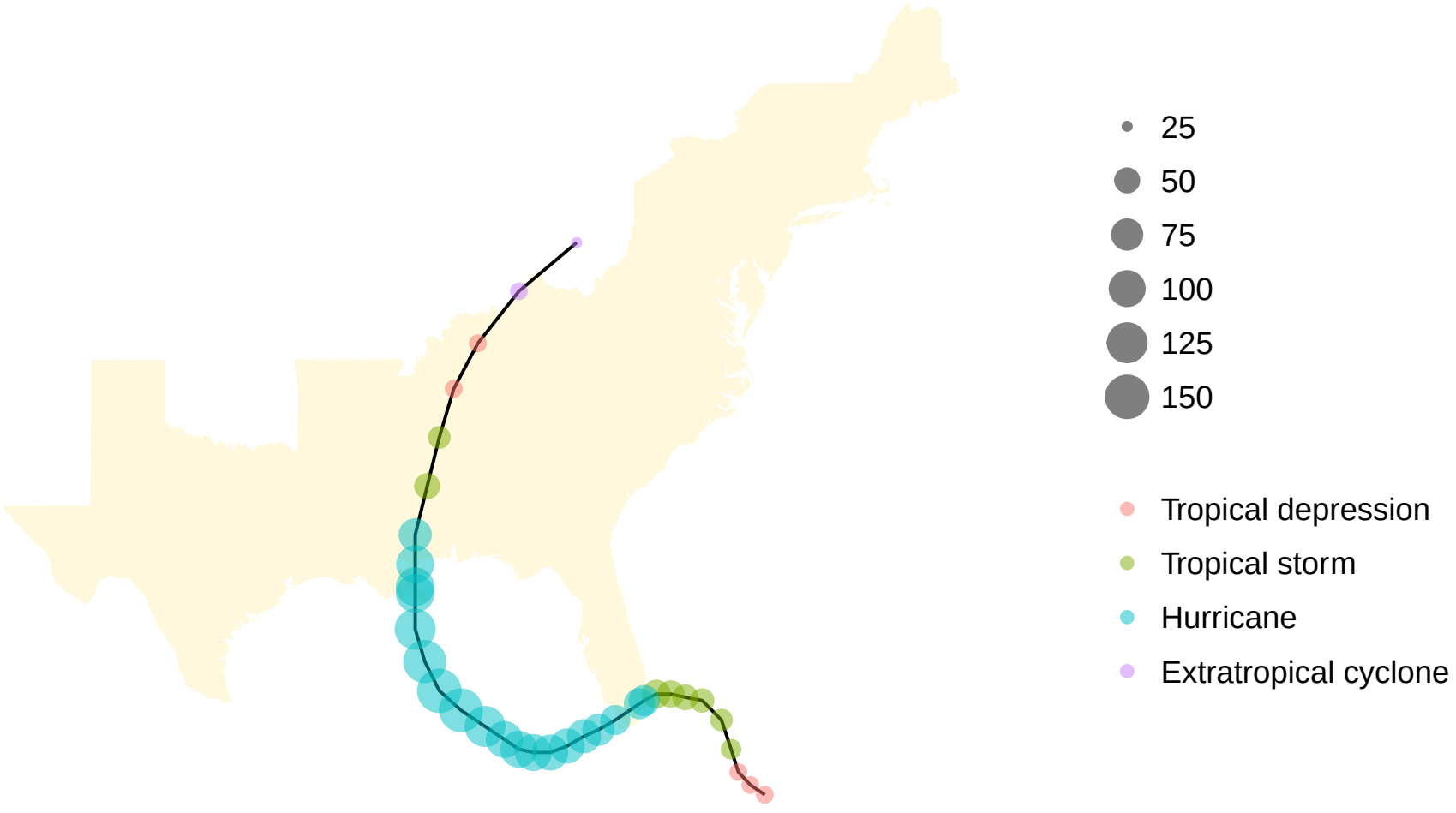
Next, write a function to plot the track for a single storm. Use color to show storm status and size to show wind speed.

Hurricane tracking data

```
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)
}
```

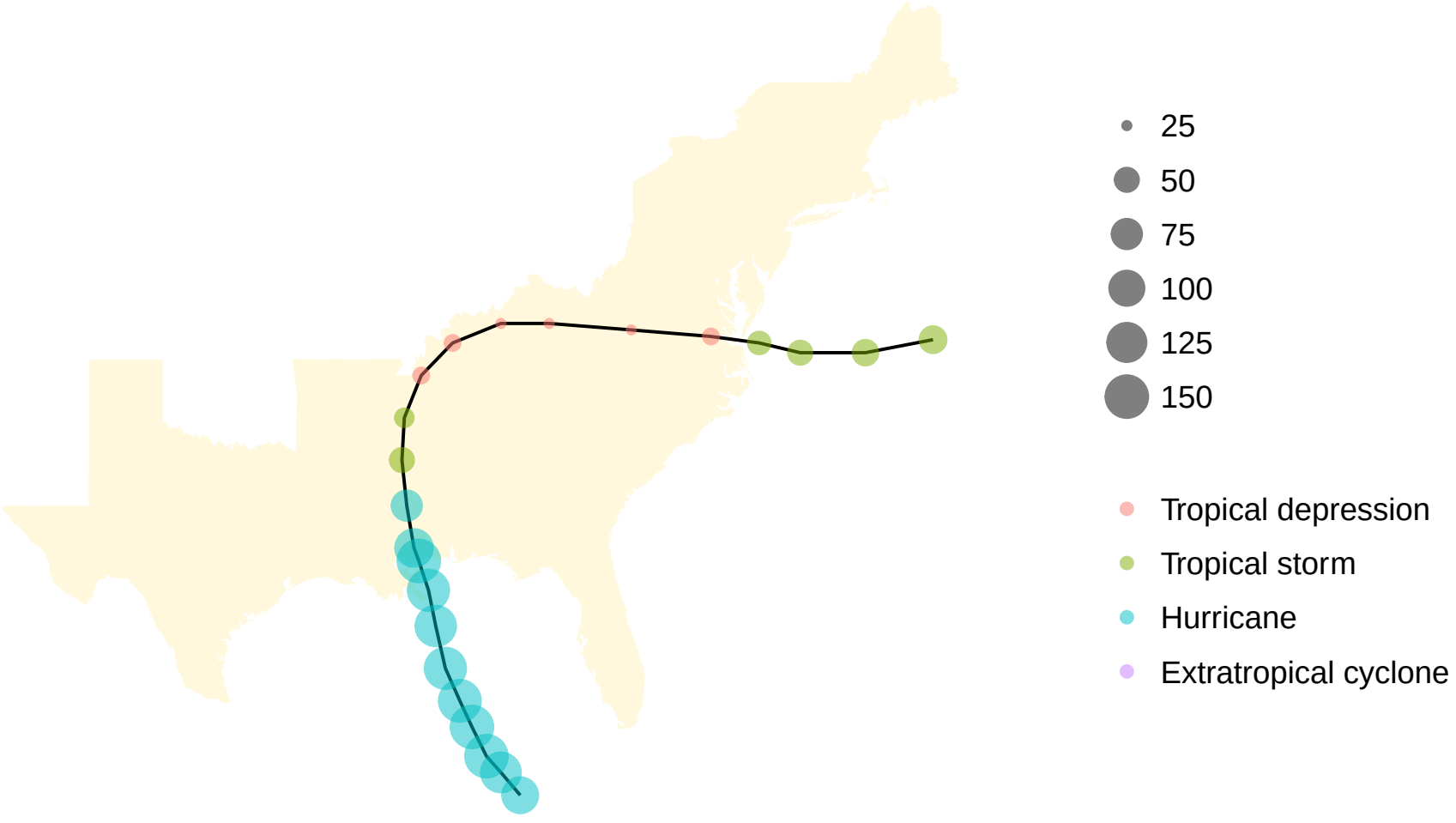
Hurricane tracking data

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



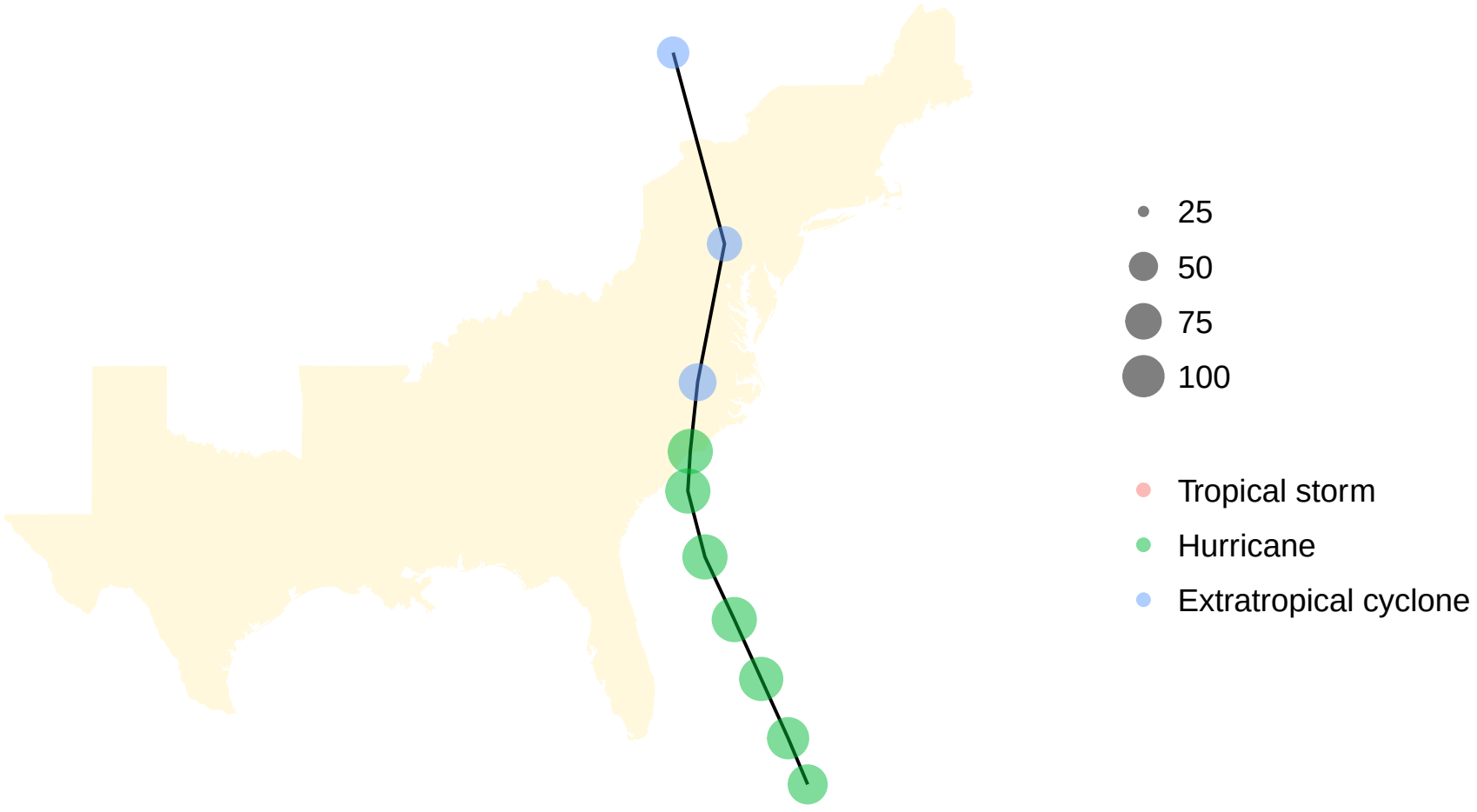
Hurricane tracking data

```
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.

```
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

Example R packages for APIs

ROpenSci

Parsing webpages