Tidy Hurricanes

Analyzing Hurricanes with Tidyverse Tools

Gaston Sanchez

Contents

About

In this manuscript I describe various examples for how to use *Tidyverse* tools to perform an exploratory analysis of storms and hurricanes in the North Atlantic.

I am assuming that you have some experience working with R. I'm also assuming that you have both R or RStudio installed in your computer. If this is not the case, you can take a look at **Breaking the Ice with R**

https://www.gastonsanchez.com/R-ice-breaker

My Series of R Tutorials

This document is part of a series of texts that I've written about Programming and Data Analysis in R:

- **Breaking the Ice with R: Getting Started with R and RStudio**

https://www.gastonsanchez.com/R-ice-breaker

- **R Coding Basics: An Introduction to Practical Programming in R**

https://www.gastonsanchez.com/R-coding-basics

- **Rolling Dice: Simulating Games of Chance in R**

https://www.gastonsanchez.com/R-rolling-dice

Donation

As a Data Science and Statistics educator, I love to share the work I do. Each month I spend dozens of hours curating learning materials like this resource. If you find any value and usefulness in it, please consider making a one-time donation—via paypal—in any amount (e.g. the amount you would spend inviting me a cup of coffee or any other drink). Your support really matters.

License

Part I

Introduction

Chapter 1

Intro

The main dish in this book has to do with working with tabular data (data arranged in rows and columns). The main reason to focus on tables is because tabular data is the most ubiquitous format in which data is handled for most types of analysis. And even if your raw data is not in tabular format, sooner or later, you'll be handling data in tabular format in most data analysis projects.

We'll begin our discussion about data tables from the "high level" (scientist's point of view), especially how to get to know your data (e.g. univariate, bivariate, multivariate analysis).

Prerequisites

We are assuming that you already have some knowledge under your belt.

You will better understand (and hopefully enjoy) the book if you've taken one or more courses on the following subjects:

- introduction to statistics
- programming or scripting
- previous data analysis eperience (basic level)

Software

We are assuming that you have installed R and RStudio in your computer.

If that's not the case, then follow the instructions to download and install them:

1) Download and Install R:
    - https://cloud.r-project.org/
2) Download and Install RStudio (Desktop free version)
    - https://rstudio.com/products/rstudio/download/

Both R and RStudio are free, and available for Mac (OS X), Windows, and Linux (e.g. Ubuntu, Fedora, Debian).

1.0.1 Installing some packages

We want you to get your hands dirty analyzing data as quick as possible, and we believe that the best way to do this is by working on a case study. To keep things moderately simple, we use one of the data sets that comes in `"dplyr"`, one the most popular R packages for manipulating tables. The main reason to start in this mode, is to avoid having to worry about data importing issues, which we cover later in the book. The other reason is to have data that is already clean and ready to be analyzed. You will also have time to learn tools and skills for cleaning data sets in subsequent chapters.

We are assuming that you already installed the package `"tidyverse"`. If that's not the case then **run on R's console** the command below (do NOT include this command in any `Rmd` file):

```
# don't include this command in any Rmd file
# don't worry too much if you get a warning message
install.packages("tidyverse")
```

Remember that you only need to install a package once! After a package has been installed in your machine, there is no need to call `install.packages()` again on the same package. What you should always invoke, in order to use the functions in a package, is the `library()` function:

```
# you should include this command in your Rmd file(s)
library(tidyverse)
```

Chapter 2

Tropical Storms

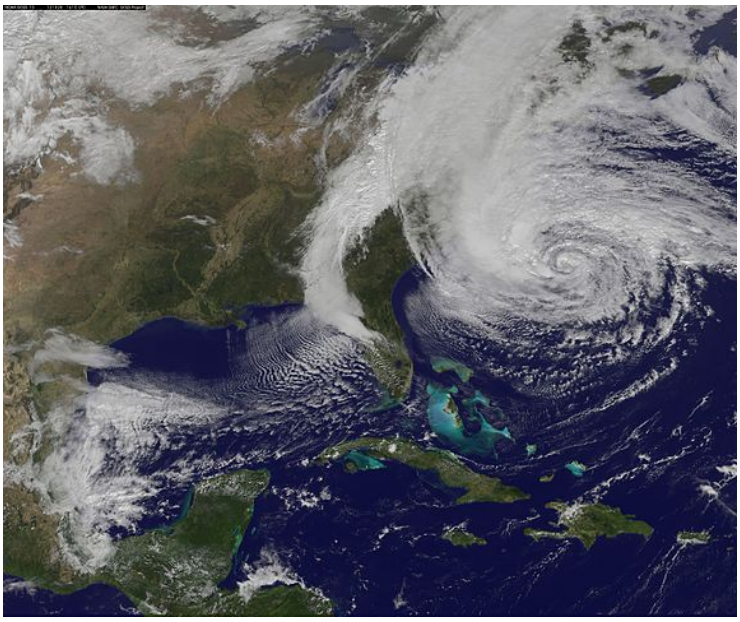


Figure 2.1: NASA satellite image of hurricane Sandy, 2012 (source: wikimedia commons)

> "Hurricane Sandy (unofficially referred to as Superstorm Sandy) was the deadliest and most destructive, as well as the strongest, hurricane of the 2012 Atlantic hurricane season. Inflicting nearly $70 billion (2012 USD) in damage, it was the second-costliest hurricane on record in the United States until surpassed by Hurricanes Harvey and Maria in 2017.

https://en.wikipedia.org/wiki/Hurricane_Sandy

2.1 A little bit about Hurricanes

NOAA SciJinks has a webpage about basic information on hurricanes.

https://scijinks.gov/hurricane/

- Hurricanes are the most violent storms on Earth.
- People call these storms by other names, such as typhoons or cyclones, depending on where they occur.
- The scientific term for all these storms is tropical cyclone.
- Only tropical cyclones that form over the Atlantic Ocean or eastern Pacific Ocean are called "hurricanes."
- A tropical cyclone is a rotating low-pressure weather system that has organized thunderstorms but no fronts.
- Hurricanes are tropical cyclones whose sustained winds have reached 74 mph.
- At this point the hurricane reaches category 1 on the Saffir-Simpson Hurricane Wind Scale,
- Saffir-Simpson Hurricane Wind Scale is a 1 to 5 rating based on a hurricane's sustained wind speed:
  - cateory 1: 74-95 mph; 64-82 kt; 119-153 km/h
  - cateory 2: 96-110 mph; 83-95 kt; 154-177 km/h
  - cateory 3: 111-129 mph; 96-112 kt; 178-208 km/h
  - cateory 4: 130-156 mph; 113-136 kt; 209-251 km/h
  - cateory 5: 157 mph or higher; 137 kt or higher; 252 km/h or higher
- Major hurricanes are defined as Category 3, 4, and 5 storms.
- The official Atlantic hurricane season runs from June through November, but occasionally storms form outside those months.
- September is the most active month (where most of the hurricanes occur), followed by August, and October (based on 1851 to 2015 data).
- A typical year has 12 named storms, including six hurricanes of which three become major hurricanes (category 3, 4, and 5).
- No hurricanes made U.S. landfall before June and after November during the period studied (1851 to 2015 data).

Chapter 3

Data Set `storms`

In this first module, you will have your first contact with tabular data (i.e. data arranged in rows and columns) which is the most common format in which data is handled for data analysis.

3.1 Atlantic Hurricane Data

In order to have a gentle introduction, we are going to use a data set that comes in one the most popular R packages for manipulation of tables: `"dplyr"`. The main reason to start in this mode, is to avoid having to worry about data importing issues, which we cover later in the course. The other reason is to have data that is already clean and ready to be analyzed.

3.1.1 Data `storms`

The package `"dplyr"` contains a dataset called `storms` which is a subset of the *NOAA Atlantic hurricane database best track data*. This database is one of several data sets available in the National Hurricane Center (NHC) Data Archive, which is part of the National Oceanic and Atmospheric Administration (NOAA). In case you are curious about the specifications and format of this type of data, you can visit the following link:

http://www.nhc.noaa.gov/data/#hurdat

The data `storms` includes the positions and attributes of 198 tropical storms, measured every six hours during the lifetime of a storm.

When you type the name of the data object, you would get something like this:

```
storms
```

```
## # A tibble: 11,859 x 13
```

```
##    name   year month   day  hour   lat  long status        categ~1  wind press~2
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>         <ord>   <int>   <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical dep~ -1         25    1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical dep~ -1         25    1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical dep~ -1         25    1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical dep~ -1         25    1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical dep~ -1         25    1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical dep~ -1         25    1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical dep~ -1         25    1011
##  8 Amy    1975     6    28    18  34   -77   tropical dep~ -1         30    1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical sto~ 0          35    1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical sto~ 0          40    1002
## # ... with 11,849 more rows, 2 more variables:
## #   tropicalstorm_force_diameter <int>, hurricane_force_diameter <int>, and
## #   abbreviated variable names 1: category, 2: pressure
```

What's going on in the above output?

- `storms` is a **tibble** object, which is one of the data objects in R that handles data in tabular format.
- tibbles are not a native R object—they come from the homonym package `"tibble"`—instead they are a modern version of data frames

The way tibbles are *printed* is very interesting.

- the number of rows that are displayed is limited to 10;
- depending on the width of the printing space, you will only see a few columns shown to fit such width.
- underneath the name of each column there is a three letter abbreviation inside angle brackets
- this abbreviation indicates the *data type* used by R to store the values. For
  - `<chr>` stands for *character* data
  - `<dbl>` means *double* (i.e. real numbers or numbers with decimal digits)
  - `<int>` means *integer* (numbers with no decimal digits)
  - `<ord>` indicates an *ordinal* `factor` which is how R handles categorical data

3.1.2 `storms` Documentation

You can find a more technical description of `storms` by taking a peek at its manual (or help) documentation. All you need to do is run this command:

```
?storms
```

Take a look at the manual (or *help*) documentation and find the description of the variables in data `storms`

3.2 General Inspection

When dealing with a data table, especially for the first time, I like to do a quick inspection of the *general structure* of the data, meaning the number of rows and columns, the name and data-type of each column, and some times also to quickly inspect a few rows either at the top or at the bottom of the table. To do all these things there is a handful of functions:

- `str()`: to get a summary of the table's structure
- `dim()`: to get the dimensions (number of rows and columns)
- `nrow()`: to get just the number of rows
- `ncol()`: to get just the number of columns
- `names()`: to get the column names; there's also `colnames()`
- `head()`: to look at a few first rows
- `tail()`: to look at a few last rows

Structure:

```
str(storms, vec.len = 1)
```

```
## tibble [11,859 x 13] (S3: tbl_df/tbl/data.frame)
##  $ name                        : chr [1:11859] "Amy" ...
##  $ year                        : num [1:11859] 1975 ...
##  $ month                       : num [1:11859] 6 6 ...
##  $ day                         : int [1:11859] 27 27 ...
##  $ hour                        : num [1:11859] 0 6 ...
##  $ lat                         : num [1:11859] 27.5 28.5 ...
##  $ long                        : num [1:11859] -79 -79 ...
##  $ status                      : chr [1:11859] "tropical depression" ...
##  $ category                    : Ord.factor w/ 7 levels "-1"<"0"<"1"<"2"<..: 1 1 ...
##  $ wind                        : int [1:11859] 25 25 ...
##  $ pressure                    : int [1:11859] 1013 1013 ...
##  $ tropicalstorm_force_diameter: int [1:11859] NA NA ...
##  $ hurricane_force_diameter    : int [1:11859] NA NA ...
```

Dimensions: number of rows and columns

```
dim(storms)
nrow(storms)
ncol(storms)
```

3.3 Basic Inspection of year

As we just saw, when you type `storms`, R displays the first 10 rows, which belong to storm Amy in 1975:

```
storms
```

From this output, we know that the data contains at least one storm from 1975. We also know, from the manual documentation of `storms`, that there are supposed to be 198 storms. But we don't know for what years. So in a more or less arbitrary way, let's begin inspecting `storms` by focusing on column `year`. Our first exploratory question is:

What years have the data been collected for?

There are several ways in R to manipulate a column from a tabular object. Using `"dplyr"`, there are two basic kinds of functions to extract variables: `pull()` and `select()`.

Figure 3.1: Extracting a column with dplyr functions "pull" and "select"

Let's do a sanity check of years. We can use the function `pull()` that *pulls* or extracts an entire column. Because there are 10010 elements in `years`, let's also use `unique()` to find out the set of year values in the data. First we pull the year, and then we identify unique occurrences:

```
unique(pull(storms, year))
```

```
##  [1] 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989
## [16] 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004
## [31] 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
## [46] 2020
```

The same can be accomplished with `select()`. The difference with `pull()` is in the way the output is handled by `select()`, which returns output in a table format:

```
unique(select(storms, year))
```

```
## # A tibble: 46 x 1
##     year
```

```
##    <dbl>
##  1  1975
##  2  1976
##  3  1977
##  4  1978
##  5  1979
##  6  1980
##  7  1981
##  8  1982
##  9  1983
## 10  1984
## # ... with 36 more rows
```

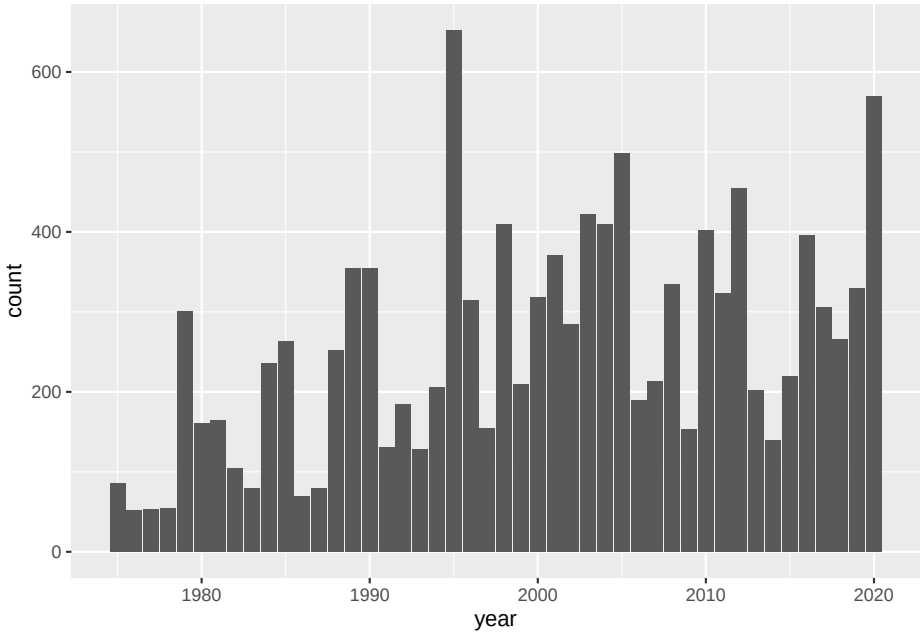
Based on the previous answers, we can see that `storms` has records during a 41-year period since 1975 to 2015.

3.3.1 Barplot of `year` values

Let's keep using the values in column `year` to obtain our first visualization with `"ggplot2"` functions. You could certainly begin a visual exploration of other variables, but we think `year` is a good place to start because it's a numeric variable, measured on a discrete scale, and this is a good candidate to use barcharts (the most popular type of graphic).

`"ggplot2"` comes with a large number of functions to create almost any type of chart. Luckily for us, it already comes with predefined functions to graph barcharts. The syntax may seem a bit scary for beginners, but you will see that it follows a logical structure. Here's the code to make a barplot of values in `year`:

```
# barchart of year values
ggplot(data = storms) +
  geom_bar(aes(x = year))
```



How does the previous command work?

- First, we always call the `ggplot()` function, typically indicating the name of the table to be used with the `data` argument.
- Then, we add more components, or **layers**, using the plus `+` operator.
- In this case we are adding just one layer: a `geom_bar()` component which is the geometric object for bars.
- To tell `ggplot()` that `year` is the column in `data = storms` to be used for the x-axis, we **map** `x = year` inside the `aes()` function which stands for *aesthetic* mapping.

We should clarify that the meaning of "aesthetic" as used by `"ggplot2"` does not mean beautiful or pretty, instead it conserves its etymological meaning of *perception*. Simply put, `aes()` is the function that you use to tell `ggplot()` which variables of a `data` object will be mapped as visual attributes of graphical elements.

3.4 Your Turn

- Use `pull()`, `select()`, and `unique()` to inspect the values in column `month`
- Try to use `sort()` in order to arrange the unique values of `month`

- Does the unique month values make sense? Are there months for which there seem to be no recorded storm data?
- Use `"ggplot2"` functions to graph a barchart for the values in column `month`.
- Look at the cheatsheet for ggplot and locate the information for `geom_bar()`. Find out how to specify: border `color`, `fill` color. Also, see what happens when you specify `alpha = 0.5`.
- Look at the cheatsheet for ggplot and locate the information for background *Themes*, e.g. `theme_bw()`. Find out how to add theme `theme_classic()` to the previous barchart.
- Look at the cheatsheet for ggplot and locate the information for *Labels*. Find out how to add a title with `ggtitle()` as well as with `labs()` to one of your previous barcharts.
- Create a barchart of `month` values. Does the plot make sense?

Part II

Analyzing Storms in 1975

Chapter 4

Storms in 1975

Let's focus on those storms recorded in 1975. How do we select them? Computationally, this operation involves a logical condition: `year == 1975`. This condition means that, from all the available year values, we get those that match 1975. This is done via `"dplyr"` function `filter()`

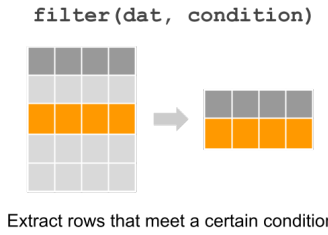


Figure 4.1: Extracting a row with dplyr function "filter"

First, let's create a subset `storms75` by *filtering* those rows with `year` equal to 1975:

```
storms75 <- filter(storms, year == 1975)
storms75
```

```
## # A tibble: 86 x 13
##    name   year month   day  hour   lat  long status        categ~1  wind press~2
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>         <ord>   <int>   <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical dep~ -1         25    1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical dep~ -1         25    1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical dep~ -1         25    1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical dep~ -1         25    1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical dep~ -1         25    1012
```

```
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical dep~ -1         25    1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical dep~ -1         25    1011
##  8 Amy    1975     6    28    18  34   -77   tropical dep~ -1         30    1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical sto~ 0          35    1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical sto~ 0          40    1002
## # ... with 76 more rows, 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>, and abbreviated variable names 1: category,
## #   2: pressure
```

Once we have the set of storms that occurred in 1975, one possible question to ask is what `unique()` storms happened in that year:

```
unique(pull(storms75, name))
```

```
## [1] "Amy"      "Caroline" "Doris"
```

From the returned output, there are only three unique storms recorded in 1975.

A similar result can be obtained with `distinct()`, the difference being the way in which the output is returned, in this case under the format of a tibble:

```
distinct(storms75, name)
```

```
## # A tibble: 3 x 1
##   name
##   <chr>
## 1 Amy
## 2 Caroline
## 3 Doris
```

Now that we know there are three storms for 1975, it would be nice to count the number of rows or records for each of them. `"dplyr"` allows us to do this with `count()`, passing the name of the table, and then the name of the column for which we want to get the counts or frequencies:

```
count(storms75, name)
```

```
## # A tibble: 3 x 2
##   name         n
##   <chr>    <int>
## 1 Amy         30
## 2 Caroline    33
## 3 Doris       23
```

4.1 Visualizing 1975 data

Let's play a bit with those storms from 1975. More specifically, let's visually explore the values of columns `wind` and `pressure`.

```
storms75
```

```
## # A tibble: 86 x 13
##    name   year month   day  hour   lat  long status        categ~1  wind press~2
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>         <ord>   <int>   <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical dep~ -1         25    1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical dep~ -1         25    1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical dep~ -1         25    1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical dep~ -1         25    1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical dep~ -1         25    1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical dep~ -1         25    1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical dep~ -1         25    1011
##  8 Amy    1975     6    28    18  34   -77   tropical dep~ -1         30    1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical sto~ 0          35    1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical sto~ 0          40    1002
## # ... with 76 more rows, 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>, and abbreviated variable names 1: category,
## #   2: pressure
```

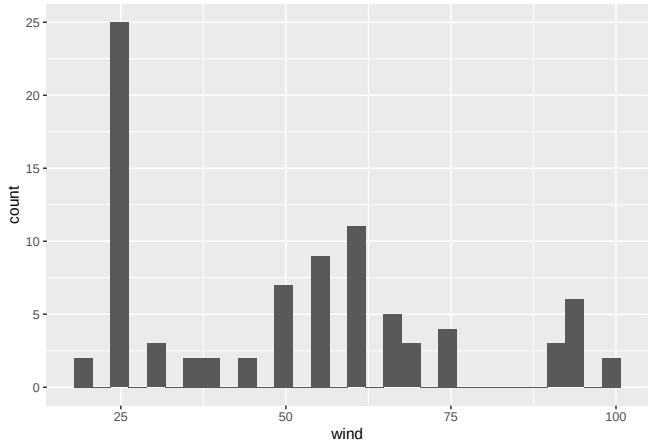
What type of visual display can we use to graph `wind` speed? The answer to this question is based by determining which type of variable `wind` is. You would agree with us in saying that `wind` is a quantitative variable. So one graphing option can be either a histogram or a boxplot, which are statistical charts to visualize the distribution of quantitative variables.

4.1.1 Histograms

Let's begin with a histogram. The associated `geom_()` function to plot a histogram is `geom_histogram()`. We are going to show you a synatx of `ggplot()` slightly different from the one we used for the barcharts. Carefully review the following code:

```
ggplot(data = storms75, aes(x = wind)) +
  geom_histogram()
```

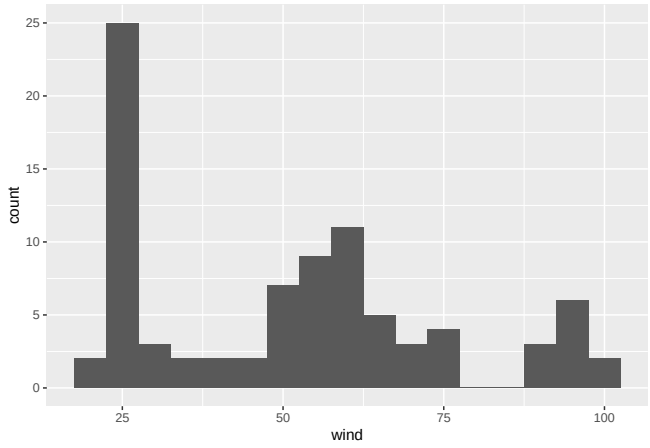
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



You should notice now that `aes()` is an argument of `ggplot()`, and not anymore an argument of the geometric-object function `geom_histogram()`. While this may be a bit confusing when learning about `"ggplot2"`, it is a very flexible and powerful behavior of `aes()`. Again, the important part of `aes()` is to understand that this function allows you to tell `ggplot()` which variables in your `data` table are used as visual attributes of the corresponding geometric elements forming the plot.

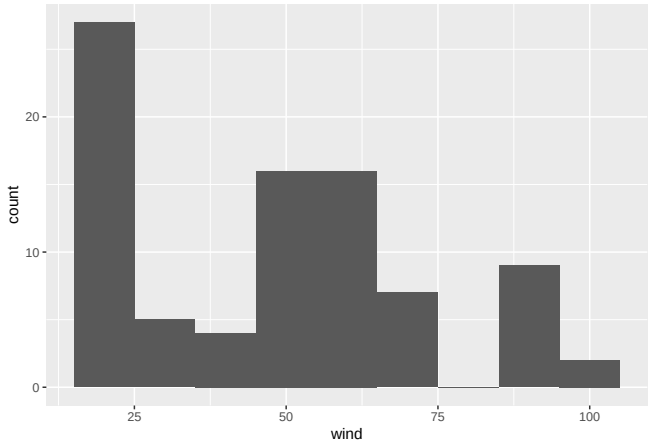
We can change the default argument `binwidth` to get another version of the histogram, for example a bin-width of 5 units (i.e. width of 5 knots):

```
ggplot(data = storms75, aes(x = wind)) +
  geom_histogram(binwidth = 5)
```



or a bin-width of 10:

```
ggplot(data = storms75, aes(x = wind)) +
  geom_histogram(binwidth = 10)
```



Now, let's reflect on what's going on in each of the histograms. Do they make sense? How do we interpret each figure?

4.1.2 Boxplots

While `ggplot()` does what we ask it to do, the displays may not be the most useful, or meaningful. Why? Think what exactly it is that we are plotting. In 1975, there are three storms:

```
unique(pull(storms75, name))
```

```
## [1] "Amy"      "Caroline" "Doris"
```

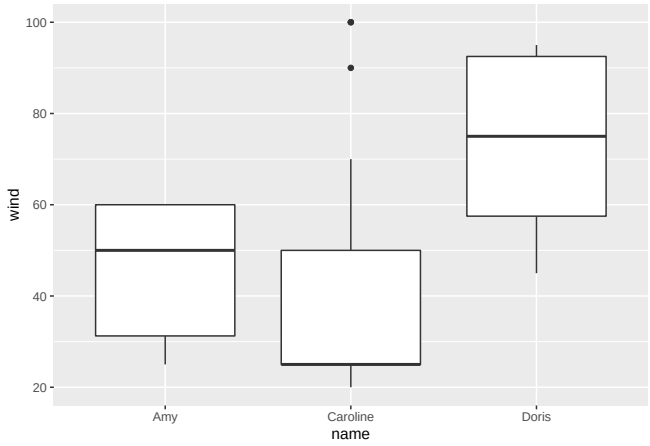
But the histograms are not differentiating between any of those three storms. Rather, the visualization is just giving us a general view of the `wind` values, from the low 20's to the high 90's, or to be more precise:

```
summary(pull(storms75, wind))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   20.00   25.00   52.50   50.87   65.00  100.00
```

However, we don't really know if all three storms have the same minimum `wind` speed, or the same maximum `wind` speed. The good news is that we can tell `ggplot()` to take into account each different storm name. But now let's use boxplots via `geom_boxplot()`, mapping `name` to the x-axis, and `wind` to the y-axis.

```
ggplot(data = storms75, aes(x = name, y = wind)) +
  geom_boxplot()
```

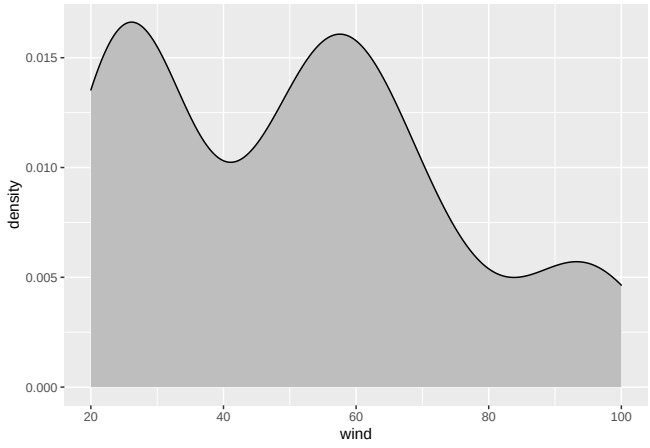


Note how different the distribution of wind speed is in each storm. We can get an alternative plot with density curves thanks to the `geom_density()` function. The syntax in this case is different. Let's first do it without separating storms, and then we do it taking into account the storm names.

4.1.3 Density Curves

Here's the command that plots a density curve of wind, without distinguishing between storms. Observe also how the argument `fill` is *set* to color `'gray'`:

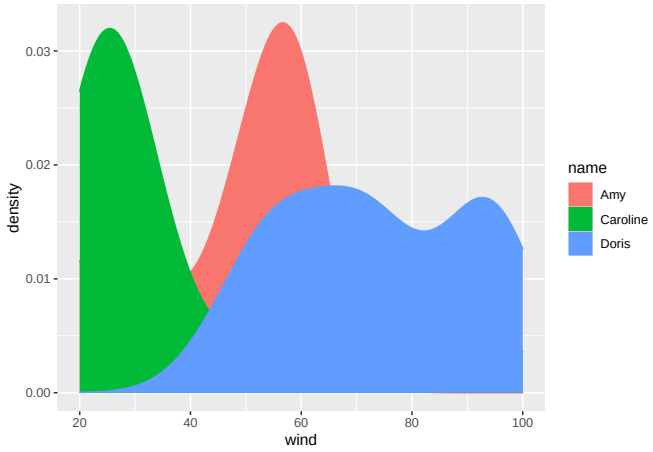
```
ggplot(data = storms75, aes(x = wind)) +
  geom_density(fill = 'gray')
```



As you can tell, the density curve looks like the profile of a roller coaster, or like the silhouette of three mountain peaks. Is this a pattern followed by wind speed in all storms? Or is it just an artifact due to the fact that we are plotting data without taking into consideration the context of `storms75`?

Let's replot density of wind, but now distinguishing between each storm. We do this by mapping `name` to the `color` argument:

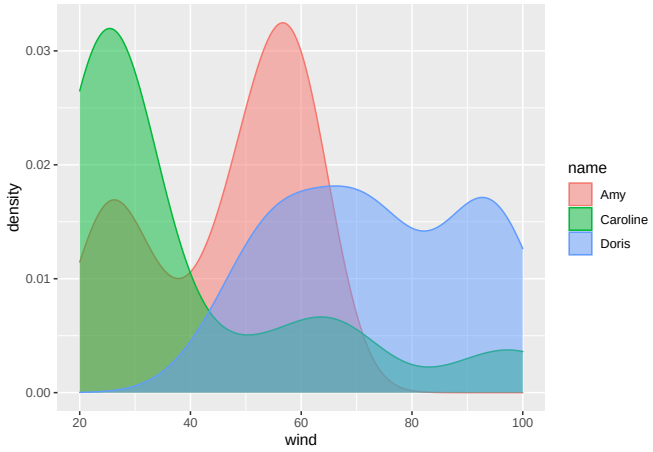
```
ggplot(data = storms75, aes(x = wind, color = name)) +
  geom_density(aes(fill = name))
```



Aha! Now things look more interesting: the roller coast shape of the first call to `geom_density()` turned out to be an artificial pattern. As you can tell from the above plot, each storm has its own different density curve.

To get a better visualization, we can take the previous command and add a bit of transparency to the colors, this is done with the argument `alpha` inside `geom_density()`. Note how arguments are specified inside `geom_density()`: we **map** `name` to the color-fill attribute of the curve inside `aes()`, but we **set** `alpha = 0.5` outside `aes()`:

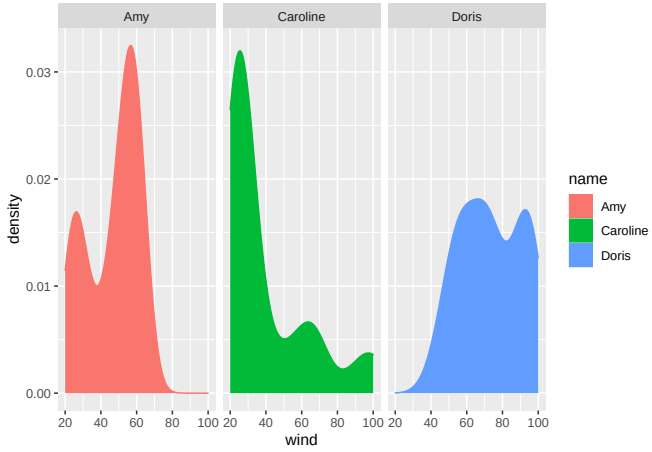
```
ggplot(data = storms75, aes(x = wind, color = name)) +
  geom_density(aes(fill = name), alpha = 0.5)
```



4.1.4 Facets

We are going to take advantage of this graphic to introduce another cool feature of `"ggplot2"` that allows us to split data based on categorical or discrete variables, in order to produce separated frames called **facets**. Here's the previous command—without `alpha` transparency—adding a new layer given by `facet_wrap()` taking into account the `name` of the storms:

```
ggplot(data = storms75, aes(x = wind, color = name)) +
  geom_density(aes(fill = name)) +
  facet_wrap(~ name)
```



In this command we are faceting by `name`. We obtain a facet for each unique category of `name`. In other words, we get separated density curves, one for each storm. The syntax inside `facet_wrap()` uses the tilde `~` operator which is the *formula* operator in R. Basically, the command `~ name` tells `ggplot()` to create facets based on the values of `name`.

Chapter 5

Storm Amy (1975)

Let's focus on storm `Amy` in 1975, subsetting `storms75` to filter out just the rows of Amy into its own table

```
amy75 <- filter(storms75, name == "Amy")
amy75
```

```
## # A tibble: 30 x 13
##    name   year month   day  hour   lat  long status        categ~1  wind press~2
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>         <ord>   <int>   <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical dep~ -1         25    1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical dep~ -1         25    1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical dep~ -1         25    1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical dep~ -1         25    1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical dep~ -1         25    1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical dep~ -1         25    1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical dep~ -1         25    1011
##  8 Amy    1975     6    28    18  34   -77   tropical dep~ -1         30    1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical sto~ 0          35    1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical sto~ 0          40    1002
## # ... with 20 more rows, 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>, and abbreviated variable names 1: category,
## #   2: pressure
```

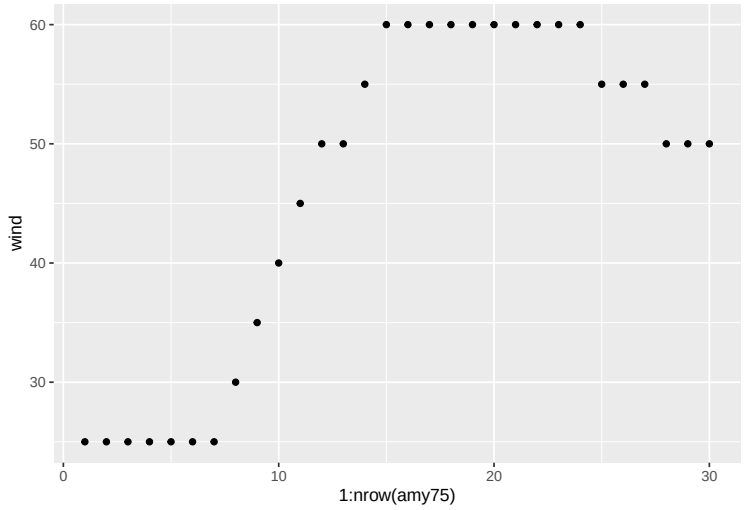
5.1 Exploring `wind`

Let's keep exploring `wind` but now let's do it chronologically, that is, graphing the wind values in the order that they were recorded (recall storms are tracked every six hours).

We begin with a scatterplot using `geom_point()`, and we specify that the x-axis

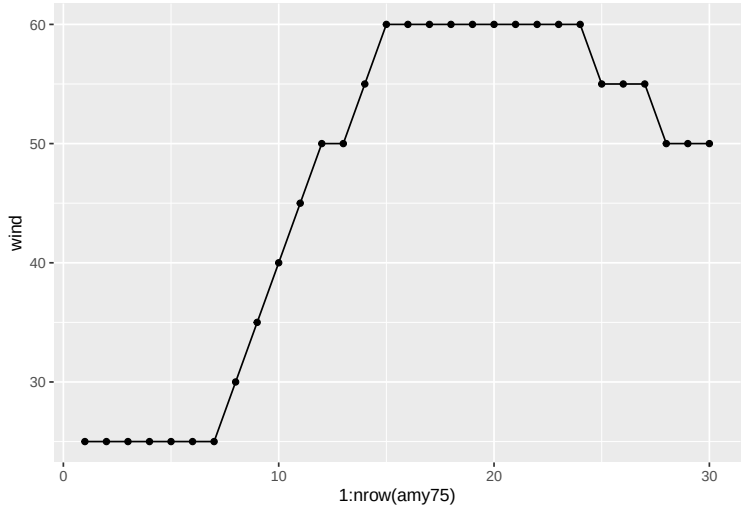
should use a numeric sequence `1:nrow(amy75)` from the first row till the last row of `amy75`, and `wind` for y-axis:

```
ggplot(data = amy75, aes(x = 1:nrow(amy75), y = wind)) +
  geom_point()
```



Because the x-axis denotes progression over time, we can connect the dots with a line. A simple way to do this is by adding another layer to our plot, this time with `geom_line()`

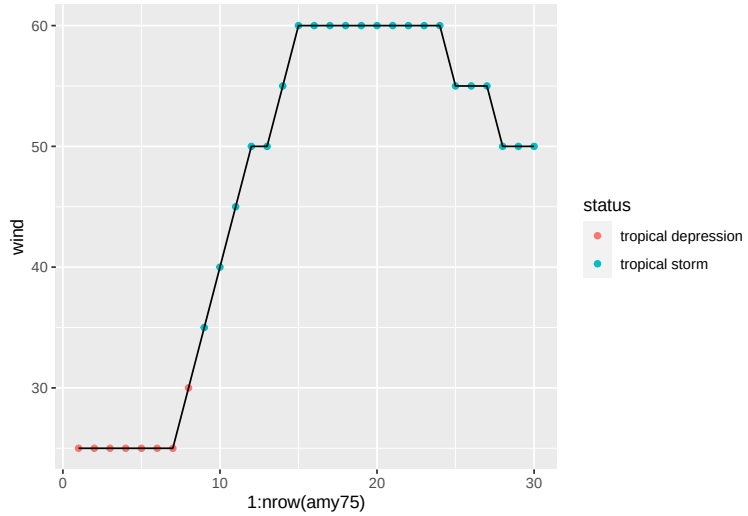
```
ggplot(data = amy75, aes(x = 1:nrow(amy75), y = wind)) +
  geom_point() +
  geom_line()
```



As you can tell, Amy started to being recorded with wind speed of 25 knots, and then after (7 x 6) = 42 hours, its speed kept increasing to 30, 35, 40, and so on until reaching its maximum speed of 60 knots that lasted 54 hours (9 x 6).

At this point, we can ask about the `status` of Amy along its lifetime. One option is to map `status` to the `color` attribute of points:

```
ggplot(data = amy75, aes(x = 1:nrow(amy75), y = wind)) +
  geom_point(aes(color = status)) +
  geom_line()
```

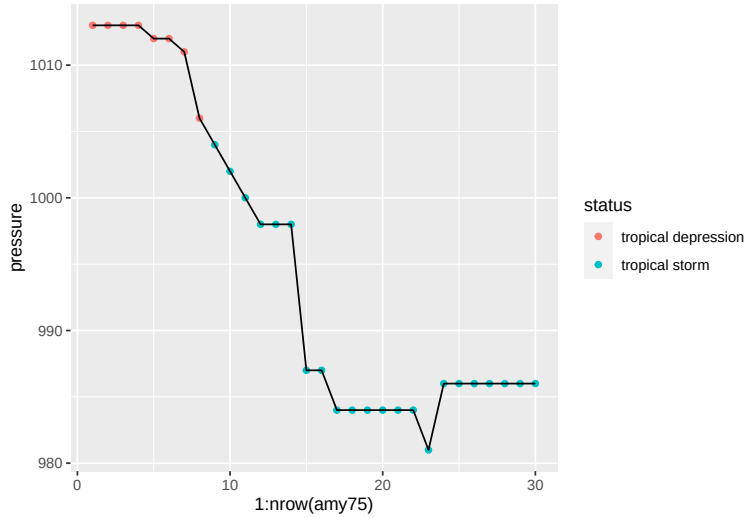


We see that Amy started as a tropical depression, and then became a tropical storm, but never became a hurricane. For a storm to reach hurricane status, of category 1, it must have one-minute maximum sustained winds of at least 64 kn (33 m/s; 74 mph; 119 km/h).

5.2 Exploring pressure

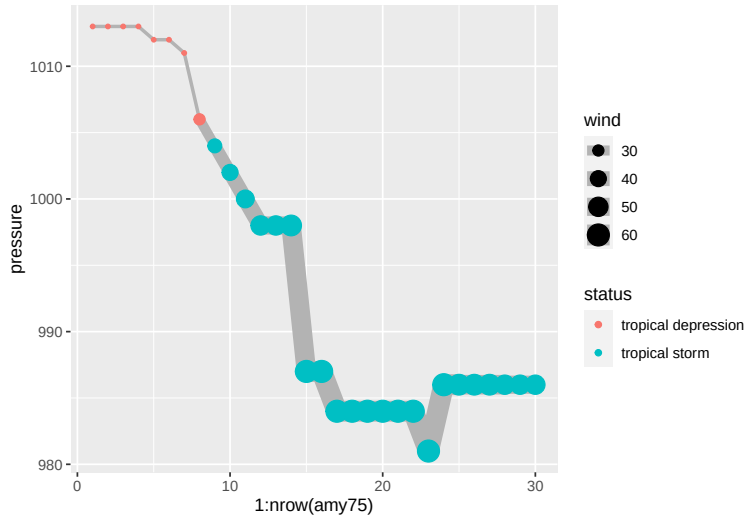
What about the pressure values of Amy? We can produce a similar scatterplot with a line connecting the dots:

```
ggplot(data = amy75, aes(x = 1:nrow(amy75), y = pressure)) +
  geom_point(aes(color = status)) +
  geom_line()
```



As an exploratory exercise, we can also play with the size of points, the size (width) of lines, colors, etc. Here's one suggestion graphing `pressure` and taking into account the `wind` speed reflected in the size of points and line segments:

```
ggplot(data = amy75, aes(x = 1:nrow(amy75), y = pressure)) +
  geom_line(aes(size = wind), lineend = "round", color = 'gray70') +
  geom_point(aes(size = wind, color = status))
```



If you know a little bit about storms, you know that there's actually an association between `wind` and `pressure`. But let's pretend for a second that we don't know much about tropical storms, hurricanes, and things like that. By looking at the previous chart, this should allows us to guess that something is going on between the `pressure` of a storm and its `wind` speed. As Amy becomes stronger,

with higher winds, its pressure levels drop accordingly, suggesting a negative correlation, which is confirmed when we compute this statistic:

```
summarise(amy75, cor(wind, pressure))
```

```
## # A tibble: 1 x 1
##   `cor(wind, pressure)`
##                   <dbl>
## 1                -0.956
```

5.3 Your Turn

- Repeat the previous exploratory steps but now with storms from year 1980.
- Try to find out how to specify a logical condition to filter various years: for example, storms from years 1975, 1976, and 1977.
- Try to find out how to specify a logical condition to filter storms from year 1975 with `wind` values less than 100.
- Use `"dplyr"` functions/commands to create a table (e.g. tibble) `storm_names_1980s` containing the name and year of storms recorded during the 1980s (i.e. from 1980 to 1989).
- Create boxplots of pressure, for storms in 1980. You can also try graphing *violins* (`geom_violin()`) instead of boxplots (`geom_boxplot()`).
- Use `"ggplot2"` functions to make a single scatterplot of `wind` and `pressure` for all storms. Use `category` to add color to the dots.
- Use `"ggplot2"` functions to make a scatterplot of `wind` and `pressure` for all storms, facetting by month, and using `category` to differentiate by color.
- Use `"ggplot2"` functions to make a scatterplot of `wind` and `pressure` for all storms, but now create facets based on `month`. Feel free to add some amount of `alpha` transparency to the color of dots.

Chapter 6

Summarizing 1975 Data

We've been working with the table `storms75`, which was obtained by *filtering* those rows with `year` equal to 1975:

```
storms75 <- filter(storms, year == 1975)
storms75
```

```
## # A tibble: 86 x 13
##    name   year month   day  hour   lat  long status        categ~1  wind press~2
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>         <ord>   <int>   <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical dep~ -1         25    1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical dep~ -1         25    1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical dep~ -1         25    1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical dep~ -1         25    1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical dep~ -1         25    1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical dep~ -1         25    1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical dep~ -1         25    1011
##  8 Amy    1975     6    28    18  34   -77   tropical dep~ -1         30    1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical sto~ 0          35    1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical sto~ 0          40    1002
## # ... with 76 more rows, 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>, and abbreviated variable names 1: category,
## #   2: pressure
```

6.1 Group-by Operations

Another common task when exploring data has to do with computations applied on certain groups or categories of data. `"dplyr"` provides the function `group_by()` which takes a data table, and we specify the column(s) on which rows will be grouped by:

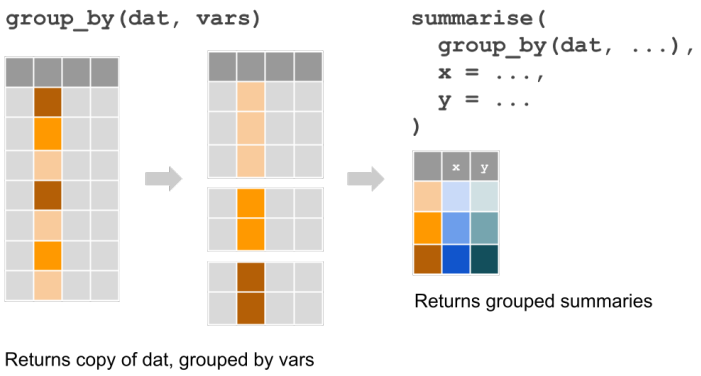


Figure 6.1: Group-by operations

For example, we may be interested in calculating the average `wind` speed and average `pressure` of each storm in 1975. First we need to group by `name`, and then we use `summarise()` to indicate that we want to get the `mean()` of `wind` and `pressure`, like this:

```
summarise(
  group_by(storms75, name),
  avg_wind = mean(wind),
  avg_pressure = mean(pressure)
)
```

```
## # A tibble: 3 x 3
##   name     avg_wind avg_pressure
##   <chr>       <dbl>        <dbl>
## 1 Amy          46.5         995.
## 2 Caroline     38.9        1002.
## 3 Doris        73.7         983.
```

Sometimes, you'll find convenient to assign the output into its own table:

```
avg_wind_pressure_75 <- summarise(
  group_by(storms75, name),
  avg_wind = mean(wind),
  avg_pressure = mean(pressure)
)

avg_wind_pressure_75
```

```
## # A tibble: 3 x 3
##   name     avg_wind avg_pressure
##   <chr>       <dbl>        <dbl>
```

```
## 1 Amy          46.5         995.
## 2 Caroline     38.9        1002.
## 3 Doris        73.7         983.
```

6.2 Arrange operations

The table of summary means `avg_wind_pressure_75` is ordered alphabetically by `name`. But perhaps you may want to organize its contents by `avg_wind` or by `avg_pressure`. Let's see how to do this.

Besides `group_by()` operations, another common type of manipulation is the arragement of rows based on the values of one or more columns. In `"dplyr"`, this can easily be achieved with the function `arrange()`. The way this function works is passing the name of the table, and then specifying one or more columns to order rows based on such values.

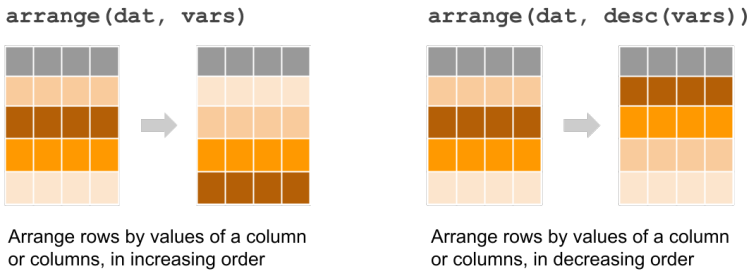


Figure 6.2: Arranging rows

Say you want to arrange the contents of the average summary table, by taking into account the column `avg_wind`:

```
arrange(avg_wind_pressure_75, avg_wind)
```

```
## # A tibble: 3 x 3
##   name     avg_wind avg_pressure
##   <chr>       <dbl>        <dbl>
## 1 Caroline     38.9        1002.
## 2 Amy          46.5         995.
## 3 Doris        73.7         983.
```

Likewise, you can also arrange the averages by `avg_pressure`:

```
arrange(avg_wind_pressure_75, avg_pressure)
```

```
## # A tibble: 3 x 3
##   name     avg_wind avg_pressure
##   <chr>       <dbl>        <dbl>
```

```
## 1 Doris        73.7         983.
## 2 Amy          46.5         995.
## 3 Caroline     38.9        1002.
```

The default behavior of `arrange()` is to organize rows in increasing order. But what if you want to organize rows in decreasing order? No problem, just use the auxiliary function `desc()` to indicate that rows should be arranged decreasingly:

```
arrange(avg_wind_pressure_75, desc(avg_wind))
```

```
## # A tibble: 3 x 3
##   name     avg_wind avg_pressure
##   <chr>       <dbl>        <dbl>
## 1 Doris        73.7         983.
## 2 Amy          46.5         995.
## 3 Caroline     38.9        1002.
```

6.3 Further inspection of 1975 storm Amy

Let's focus on a specific storm, for example storm `Amy` in 1975. For sake of simplicity, we are going to create a table `amy75` containing the values of this storm:

```
amy75 <- filter(storms75, name == "Amy")
amy75
```

```
## # A tibble: 30 x 13
##    name   year month   day  hour   lat  long status        categ~1  wind press~2
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>         <ord>   <int>   <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical dep~ -1         25    1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical dep~ -1         25    1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical dep~ -1         25    1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical dep~ -1         25    1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical dep~ -1         25    1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical dep~ -1         25    1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical dep~ -1         25    1011
##  8 Amy    1975     6    28    18  34   -77   tropical dep~ -1         30    1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical sto~ 0          35    1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical sto~ 0          40    1002
## # ... with 20 more rows, 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>, and abbreviated variable names 1: category,
## #   2: pressure
```

Here's a couple of questions that we could investigate:

- which are the `status` categories for Amy?
- during which months was Amy active? and for how many days?

- what are the basic summary statistics for `wind` and `pressure`?

```
# which are the `status` categories for Amy?
distinct(amy75, status)
```

```
## # A tibble: 2 x 1
##   status
##   <chr>
## 1 tropical depression
## 2 tropical storm
```

```
# during which months was Amy active?
distinct(amy75, month)
```

```
## # A tibble: 2 x 1
##   month
##   <dbl>
## 1     6
## 2     7
```

```
# for how many days was Amy active?
count(distinct(amy75, day))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     8
```

```
# summary statistics for wind
summary(select(amy75, wind))
```

```
##       wind
##  Min.   :25.00
##  1st Qu.:31.25
##  Median :50.00
##  Mean   :46.50
##  3rd Qu.:60.00
##  Max.   :60.00
```

```
# summary statistics for pressure
summary(select(amy75, pressure))
```

```
##     pressure
##  Min.   : 981.0
##  1st Qu.: 986.0
##  Median : 987.0
##  Mean   : 995.1
##  3rd Qu.:1005.5
##  Max.   :1013.0
```

6.3.1 Your Turn

- Use `"dplyr"` functions/commands to create a table (e.g. tibble) `max_wind_pressure_75` containing columns: 1)`name` of storm, 2) `max_wind` maximum wind speed, and 3) `max_pressure` maximum pressure
- Use `"dplyr"` functions/commands to create a table (e.g. tibble) `wind_stats_75` containing columns: 1)`name` of storm, 2) `min_wind` minimum wind speed, 3) `avg_wind` mean wind speed, 4) `med_wind` median wind speed, and 5) `max_wind` maximum wind speed.

Chapter 7

Basic Maps

In the previous chapters, you were introduced to the basics of "dplyr" and "ggplot2", performing various operations on the data storms. Because this data set contains geographical information such as longitude and latitude, we need to take a further step in this module in order to learn about plotting basic geographical maps.

You will need the following packages:

```
library(tidyverse)      # for syntactic manipulation of tables
library(maps)           # for drawing basic geographical maps
```

and the following objects:

```
storms75 <- filter(storms, year == 1975)
```

7.1 Graphing Maps

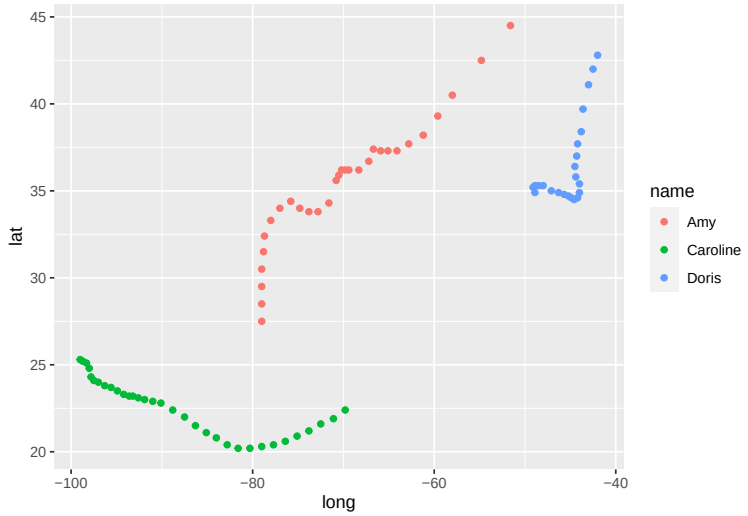
In this part, we give a basic exposure to plotting maps with "ggplot2" and "maps". Keep in mind that there is a wide array of packages for graphing all sorts of maps, and geospatial information. Good resources to look at are:

- *Drawing beautiful maps programmatically with R, sf and ggplot2* by Mel Moreno and Mathieu Basille; https://www.r-spatial.org/r/2018/10/25/ggplot2-sf.html
- *Geocomputation with R* by Robin Lovelace, Jakub Nowosad, and Jannes Muenchow; https://geocompr.robinlovelace.net
- *Making Maps with R* by Eric C. Anderson; https://eriqande.github.io/rep-res-web/lectures/making-maps-with-R.html

7.1.1 Plotting location of storm records

For illustration purposes, we continue using the data frame `storms75`. Having latitude and longitude, we can make a scatterplot to see the location of the storm records. Recall that the `ggplot` function to do this is `geom_point()`. To distinguish each storm, we can color the dots by taking into account the different storm names. This involves *mapping* the column `name` to the `color` attribute:

```
ggplot(data = storms75, aes(x = long, y = lat, color = name)) +
  geom_point()
```



Keep in mind that the previous command can also be written as:

```
# alternative ways to write equivalent commands
ggplot(data = storms75) +
  geom_point(aes(x = long, y = lat, color = name))

ggplot() +
  geom_point(data = storms75, aes(x = long, y = lat, color = name))
```

The above scatterplot is a good starting point to visualize the location of the storm records, but it would be nice to have an actual image of a map. Let's see how to do this in the following subsections.

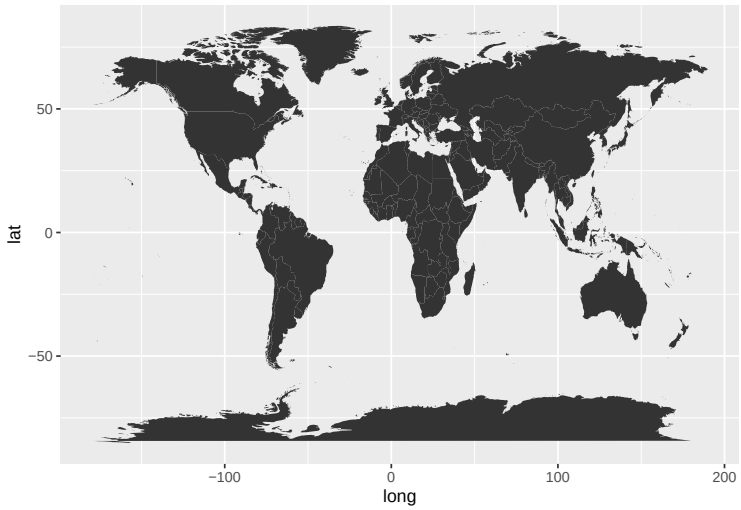
7.1.2 Basic map

One of the oldest ways to plot maps in R is with the package `"maps"`. Nowadays, there are better packages for geospatial data and making maps, but let's not worry about them at this moment.

One rudimentary way to plot a map is by first getting data of the world. `"ggplot2"` provides the function `map_data()` to create the required data ta-

ble with geospatial information of a world map. All you have to do is specify the name of the map provided by the `"maps"` package. In this example, let's use the `"world"` map. Once we have this data, we can use it with `ggplot()` and a `geom_polygon()` layer like this:

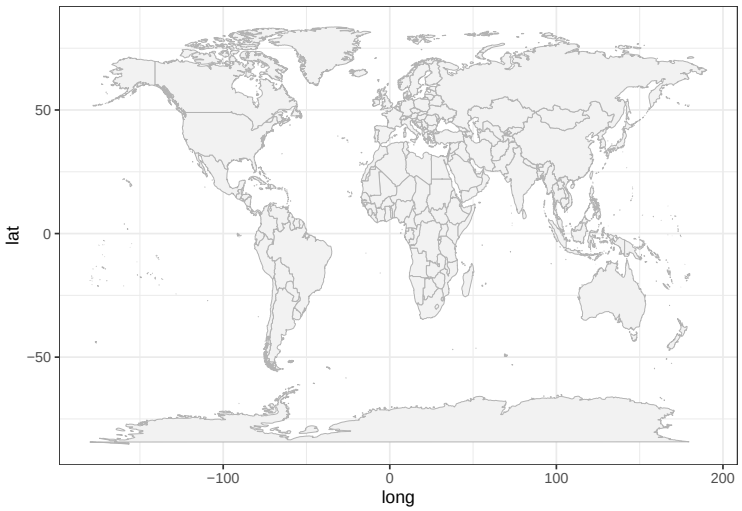
```
# world map data
world_map <- map_data("world")

# a default world map
ggplot() +
  geom_polygon(data = world_map,
               aes(x = long, y = lat, group = group))
```



In the above command, notice how we specify the `data` argument inside `geom_polygon()` instead of inside `ggplot()`. We do this because the data frame `world_map` is used to graph the layer of the map. We still need to add another layer—via `geom_point()`—for the coordinates indicating the position of each storm's record.

To handle the code more easily, let's modify the map, and create a `"ggplot"` object called `gg_world`. We'll use this object as our "canvas" for plotting the storm locations:

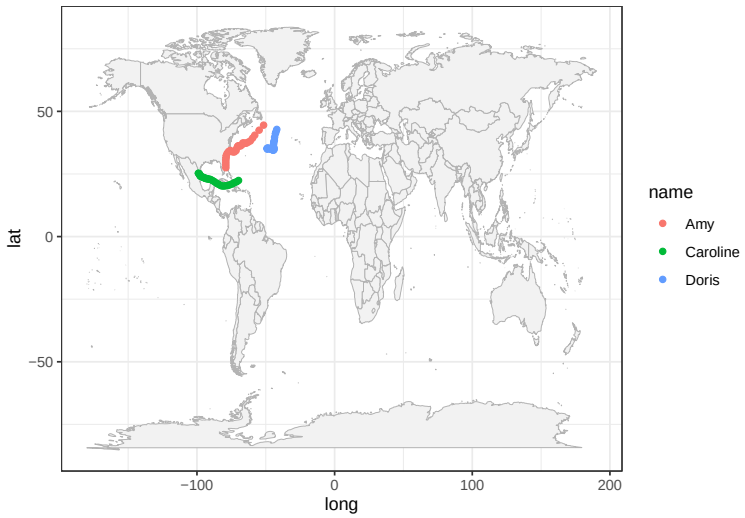
```
# map "canvas" stored as gg_world
gg_world <- ggplot() +
  geom_polygon(data = world_map,
               aes(x = long, y = lat, group = group),
               fill = "gray95", colour = "gray70", size = 0.2) +
  theme_bw()

gg_world
```



7.1.3 Mapping 1975 Storms

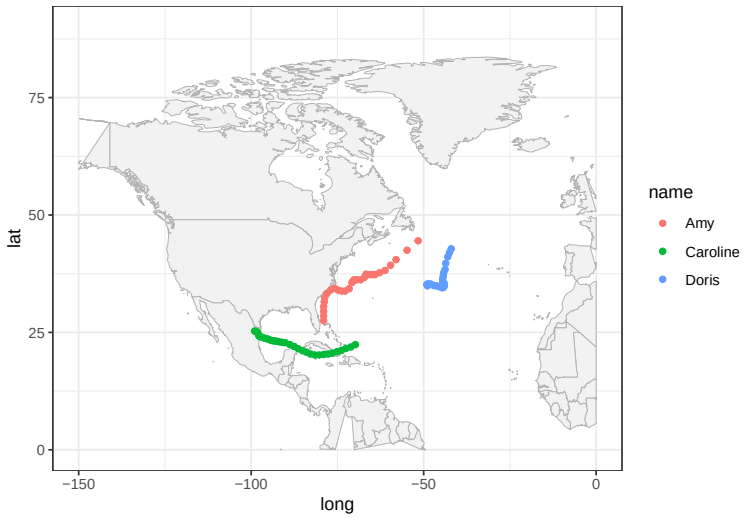
Now that we know how to plot a map with `ggplot()`, we can add the points of the storm records. This is done with `geom_point()`, and specifying `storms75` as the `data` argument inside this function. In other words, we are using two separate data frames. One is `world_map`, used to draw the polygons of the map; the other one is `stomrs75` to graph the dots of each storm. Notice also that there are no inputs provided to the function `ggplot()`.

```
# world map, adding storms in 1975
gg_world +
  geom_point(data = storms75,
             aes(x = long, y = lat, color = name))
```



Because the analyzed hurricanes occurred in the North Atlantic basin, we can focus on that region by modifying the x-and-y axis limits:

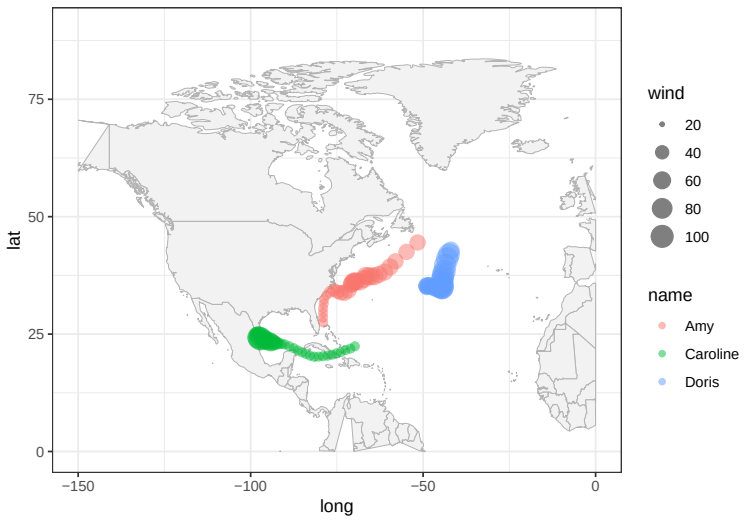
```
# zoom-in
gg_world +
  geom_point(data = storms75,
             aes(x = long, y = lat, color = name)) +
  xlim(c(-150, 0)) +
  ylim(c(0, 90))
```



It's worth mentioning that this zooming-in has a secondary effect of distorting some of the polygons. For example, Alaska seems to get cut in half. Also the polygon of Colombia is incomplete. Ignoring these distortions for now, we can

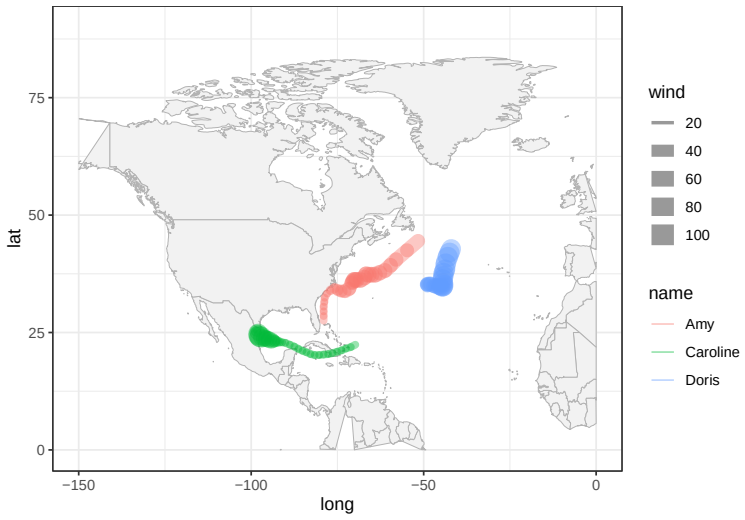
continue exploring things by taking into account more variables. For instance, let's map the `wind` speed to the `size` argument of points.

```
gg_world +
  geom_point(data = storms75,
             aes(x = long, y = lat, color = name, size = wind),
             alpha = 0.5) +
  xlim(c(-150, 0)) +
  ylim(c(0, 90))
```



A very similar appearance can be achieved by replacing `geom_point()` with `geom_path()`:

```
gg_world +
  geom_path(data = storms75,
             aes(x = long, y = lat, color = name, size = wind),
             lineend = "round", alpha = 0.4) +
  xlim(c(-150, 0)) +
  ylim(c(0, 90))
```


75
50
25
0
lat
−150
−100
−50
0
long
wind
20
40
60
80
100
name
Amy
Caroline
Doris

Chapter 8

Less Basic Maps

The code in this chapter requires the following packages:

```
library(tidyverse)          # for syntactic manipulation of tables
library(rnaturalearth)      # world map data from Natural Earth
library(rnaturalearthdata)  # companion package of rnaturalearth
```

and the following table for storms in 1975:

```
storms75 <- filter(storms, year == 1975)
```

8.1 More mapping approaches

Another interesting map graphing approach is by using map-objects from the package `"rnaturalearth"`.

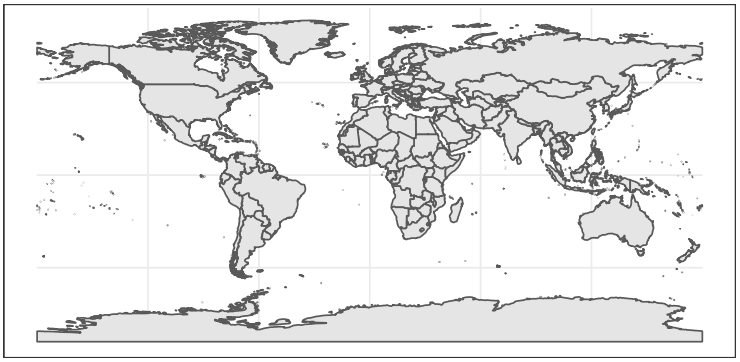
We use the `ne_countries()` function—from `"rnaturalearth"`—to get world country polygons. In the following command, we specify a `medium` scale resolution, and a returned object of class `"sf"` (simple features).

```
# another world data frame
world_df <- ne_countries(scale = "medium", returnclass = "sf")
class(world_df)
```

```
## [1] "sf"         "data.frame"
```

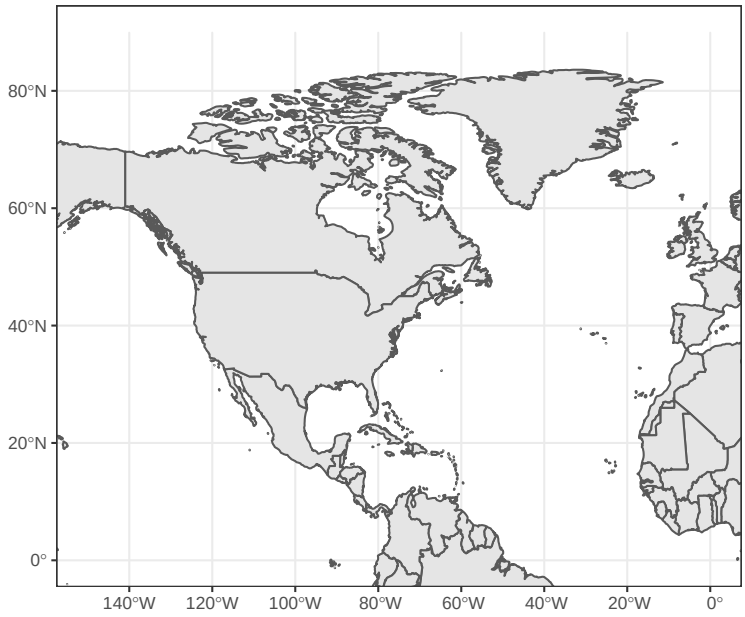
Now we can pass `world_df` to `ggplot()`, and use `geom_sf()` which is the function that allows us to visualize *simple features* objects `"sf"`.

```
# another world map (from "rnaturalearth")
ggplot(data = world_df) +
  geom_sf() +
  theme_bw()
```



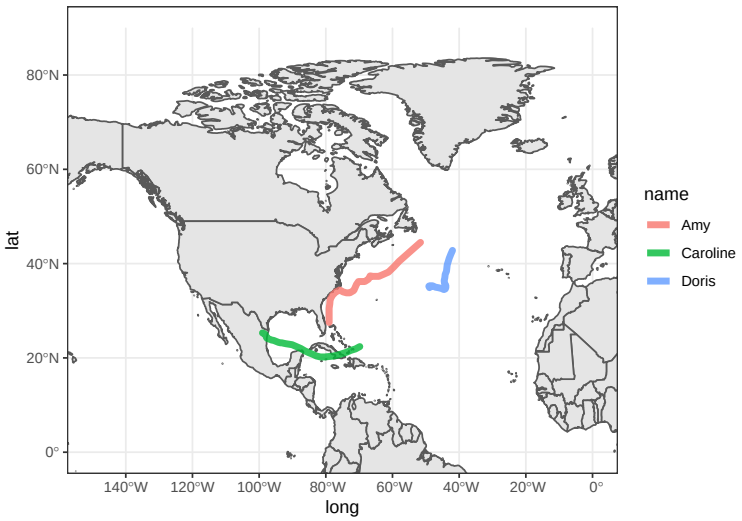
One advantage of using this other mapping approach is that we can zoom-in without having distorted polygons. To focus on a specific region, we set the x-axis and y-axis limits with the `coord_sf()` function. Again, for coding convenience, let's create another `"ggplot"` object

```
# ggplot object to be used as a canvas
gg_world2 <- ggplot(data = world_df) +
  geom_sf() +
  coord_sf(xlim = c(-150, 0), ylim = c(0, 90), expand = TRUE) +
  theme_bw()

gg_world2
```



Now let's add the storms:

```
gg_world2 +
  geom_path(data = storms75,
            aes(x = long, y = lat, color = name),
            lineend = "round", size = 2, alpha = 0.8)
```



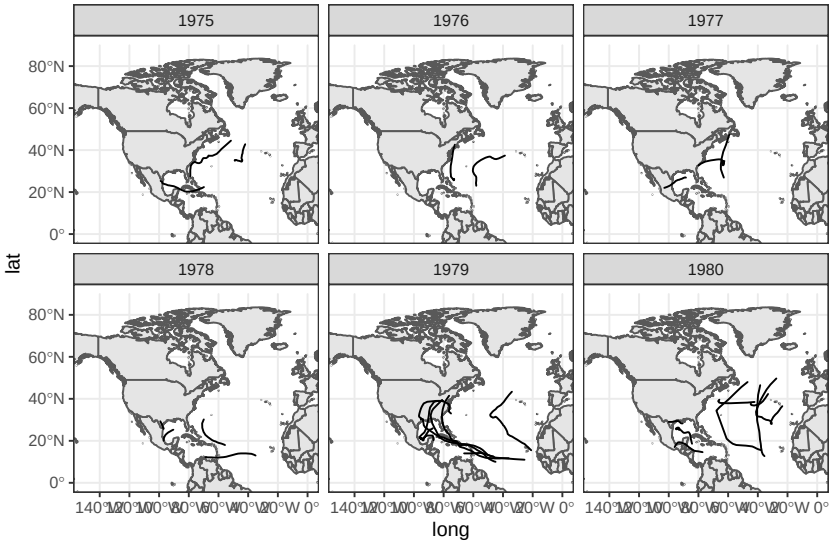
8.1.1 Storms from 1975 to 1980

As a simple experiment, let's graph storms between 1975 and 1980 (six years). First we create a dedicated data table `storms_75_80` to select the rows we are interested in:

```
storms_75_80 <- filter(storms, year %in% 1975:1980)
```

And then we can use `facet_wrap(~ year)` to graph storms by year:

```
gg_world2 +
  geom_path(data = storms_75_80,
            aes(x = long, y = lat, group = name),
            lineend = "round") +
  xlim(c(-150, 0)) +
  ylim(c(0, 90)) +
  facet_wrap(~ year)
```


1975
1976
1977
1978
1979
1980
80°N
60°N
40°N
20°N
0°
lat
140°W 120°W 100°W 80°W 60°W 40°W 20°W 0°
long

Chapter 9

Part 1 Summary

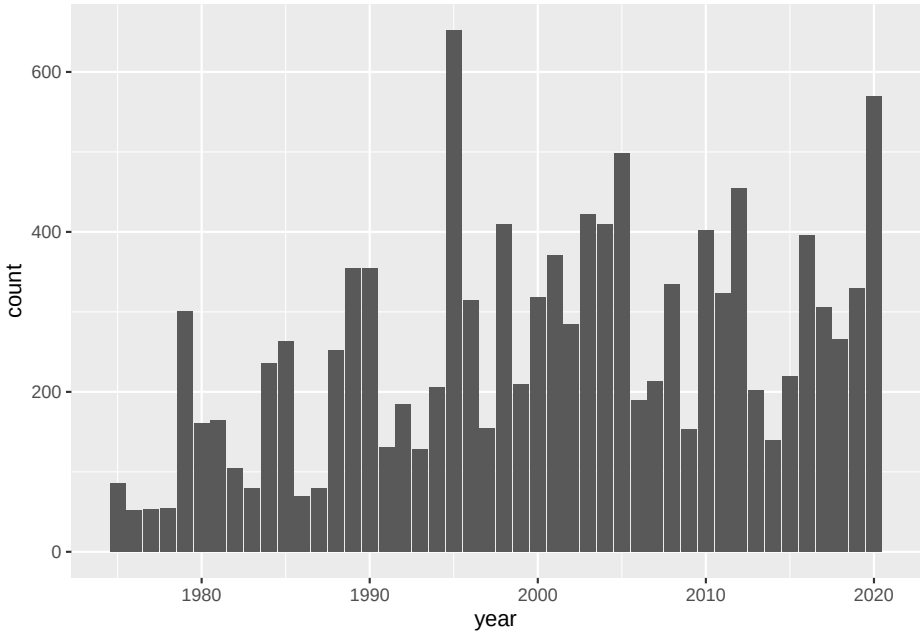
So far, we've covered several functions from `"dplyr"`, as well as some other functions in R:

- functions from `"dplyr"`
  - `pull()` and `select()`
  - `filter()`
  - `group_by()`
  - `arrange()` and `desc()`
  - `count()`, `distinct()`, `summarise()`
- functions in base R
  - `unique()`, `sort()`, `mean()`, `summary()`

9.1 Number of Storms per Year

If you recall, our first ggplot involved a barchart for the values in column `year`

```
ggplot(data = storms) +
  geom_bar(aes(x = year))
```



We discovered that the 41-year period of recorded data from 1975 to 2015. We can take a further step and ask: how many storms are there in each year?

To answer this question, we need to do some data manipulation with `"dplyr"`. Our general recommendation when working with `"dplyr"`'s functions, especially when you are learning about them, is to do computations step by step, deciding which columns you need to use, which rows to consider, which functions to call, and so on.

Think about the columns that we need to select to find the number of unique storms per year. We obviously need `year`, but this column alone it's not enough because for any given storm we have multiple records with the same year. Therefore, we also need column `name`.

For illustration purposes, we are going to build the data manipulation pipeline step by step. As you get more comfortable with `"dplyr"` and other functions, you won't have the need to disect every single command.

A first step is to `select()` variables `year` and `name`:

```
select(storms, year, name)
```

```
## # A tibble: 11,859 x 2
##     year name
##    <dbl> <chr>
##  1  1975 Amy
##  2  1975 Amy
##  3  1975 Amy
```

```
##  4  1975 Amy
##  5  1975 Amy
##  6  1975 Amy
##  7  1975 Amy
##  8  1975 Amy
##  9  1975 Amy
## 10  1975 Amy
## # ... with 11,849 more rows
```

Next, we need to `group_by()` year. At first glance, the previous output and the output below seem identical. But notice the tiny difference: the output below has a second line of text with some relevant information: `# Groups: year [41]`, telling us that the values are grouped by year.

```
group_by(select(storms, year, name), year)
```

```
## # A tibble: 11,859 x 2
## # Groups:   year [46]
##     year name
##    <dbl> <chr>
##  1  1975 Amy
##  2  1975 Amy
##  3  1975 Amy
##  4  1975 Amy
##  5  1975 Amy
##  6  1975 Amy
##  7  1975 Amy
##  8  1975 Amy
##  9  1975 Amy
## 10  1975 Amy
## # ... with 11,849 more rows
```

Then, we identify the `distinct()` values (combination of year-name):

```
distinct(group_by(select(storms, year, name), year))
```

```
## # A tibble: 512 x 2
## # Groups:   year [46]
##     year name
##    <dbl> <chr>
##  1  1975 Amy
##  2  1975 Caroline
##  3  1975 Doris
##  4  1976 Belle
##  5  1976 Gloria
##  6  1977 Anita
##  7  1977 Clara
##  8  1977 Evelyn
```

```
##  9  1978 Amelia
## 10  1978 Bess
## # ... with 502 more rows
```

For convenience purposes, let's assign this table into its own object, which we can call `storms_year_name`

```
storms_year_name <- distinct(group_by(select(storms, year, name), year))
```

Finally, we need to `count()` how many storms are in each year:

```
count(storms_year_name, year)
```

```
## # A tibble: 46 x 2
## # Groups:   year [46]
##     year     n
##    <dbl> <int>
##  1  1975     3
##  2  1976     2
##  3  1977     3
##  4  1978     4
##  5  1979     7
##  6  1980     8
##  7  1981     5
##  8  1982     5
##  9  1983     4
## 10  1984    10
## # ... with 36 more rows
```

All the previous commands can be assembled together with various embedded lines of code:

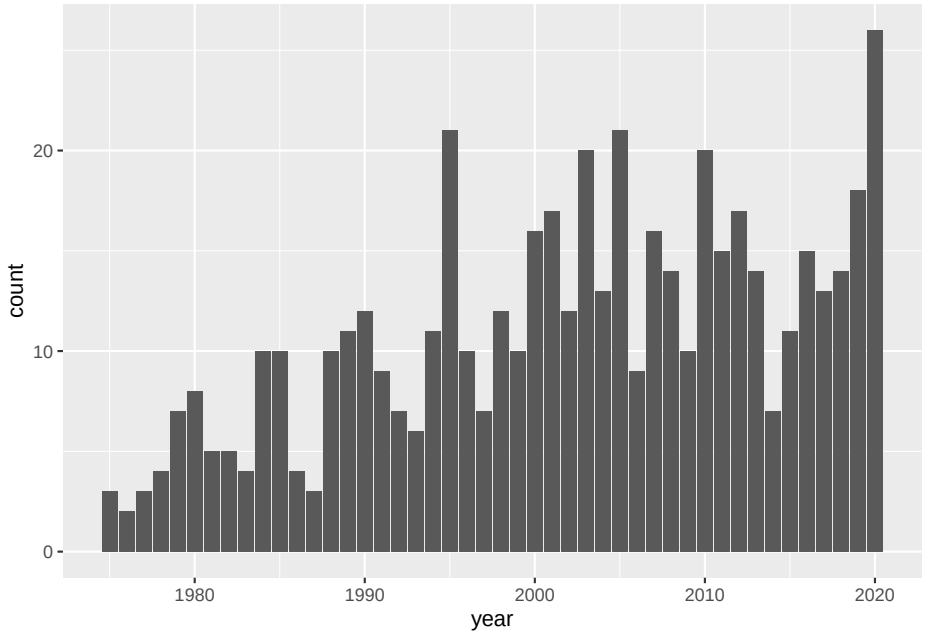
```
storms_per_year <- count(
  distinct(
    group_by(
      select(storms, year, name),
      year)
    )
  )

storms_per_year
```

```
## # A tibble: 46 x 2
## # Groups:   year [46]
##     year     n
##    <dbl> <int>
##  1  1975     3
##  2  1976     2
##  3  1977     3
```

```
##  4  1978     4
##  5  1979     7
##  6  1980     8
##  7  1981     5
##  8  1982     5
##  9  1983     4
## 10  1984    10
## # ... with 36 more rows
```

Now that we have the counts or frequencies, we can make our next barchart. In this case, we will use the table `storms_year_name` as the input data for `ggplot()`:

```
ggplot(data = storms_year_name) +
  geom_bar(aes(x = year))
```



By looking at the chart, there are some fairly tall bars. Although it's hard to see exactly which years have a considerably large number of storms, eyeballing things out it seems that around 1995, 2003, 2005, and 2010 there are 20 or more storms. We can find the actual answer by using `arrange()`, specifying the counts to be shown in descending order—with `desc()`:

```
arrange(storms_per_year, desc(n))
```

```
## # A tibble: 46 x 2
## # Groups:   year [46]
##     year     n
```

```
##    <dbl> <int>
##  1  2020    26
##  2  1995    21
##  3  2005    21
##  4  2003    20
##  5  2010    20
##  6  2019    18
##  7  2001    17
##  8  2012    17
##  9  2000    16
## 10  2007    16
## # ... with 36 more rows
```

As you can tell, in the 41-year period from 1975 to 2015, there are two years, 1995 and 2005, with a maximum number of storms equal to 21.

9.2 Your Turn

- Use `"dplyr"` functions/commands to create a table (e.g. tibble) `storm_records_per_year` containing three columns: 1) `name` of storm, 2) `year` of storm, and 3) `count` for number of recorded valued (of the corresponding storm).
- Use `"dplyr"` functions/commands to create a table (e.g. tibble) `storms_categ5` containing the name and year of those storms of category 5.
- Use `"dplyr"` functions/commands to display a table showing the `status`, `avg_pressure` (average pressure), and `avg_wind` (average wind speed), for each type of storm `category`. This table should contain four columns: 1) `category`, 2) `status`, 3) `avg_pressure`, and 4) `avg_wind`.
- Use `"dplyr"` functions/commands to create a table (e.g. tibble) `max_wind_per_storm` containing three columns: 1) `year` of storm, 2) `name` of storm, and 3) `max_wind` maximum wind speed record (for that storm).
- Use `"dplyr"` functions/commands to create a table (e.g. tibble) `max_wind_per_year` containing three columns: 1) `year` of storm, 2) `name` of storm, and 3) `wind` maximum wind speed record (for that year). Arrange rows by wind speed in decreasing order.