Foundations of Data Science for Biologists

More data wrangling

BIO 724D

02-OCT-2023

Instructors: Greg Wray, Paul Magwene, Jesse Granger

Revisiting data types and introducing factors

Revisiting data types and structures

Base R provides six types of atomic vectors, the most fundamental data type

- Only two are essential: **float**, **string**
- Two more are useful and very common: **integer**, **logical**
- Two more are very rarely used in data science: **complex**, **byte**

Base R provides several data structures, each with distinct uses

- Container for tabular data: **data frame**
- General-purpose 1-dimensional container: **list**
- Look-up table, hash table, or associative array: **named list**
- 2- and 3-dimensional homogenous arrays: **array**, **matrix**
- Vector for categorical data: **factor**

Revisiting indexing

Indexing = mechanism to refer to a specific subset of a data object

Square bracket `obj[x]`

Integer argument; returns an *object* (a list if indexing a data frame)

Double square bracket `obj[[x]]`

Integer argument; returns *values* (an atomic vector if indexing a data frame)

Dollar sign `obj$x`

String argument; returns item (from list) or column (from data frame) as *values*

Tidyverse `func(df, x)`

String argument; typically returns another data frame

Introducing factors

Factors are categorical data vectors (homogenous, but different from atomic vectors)

- Categories are variables with a defined set of possible values, usually just a few

In R, categories are called **levels**

- By default, levels are not ordered (e.g., herbivore, carnivore, omnivore)
- Optionally, levels can be ordered (e.g., egg, larva, pupa, adult)

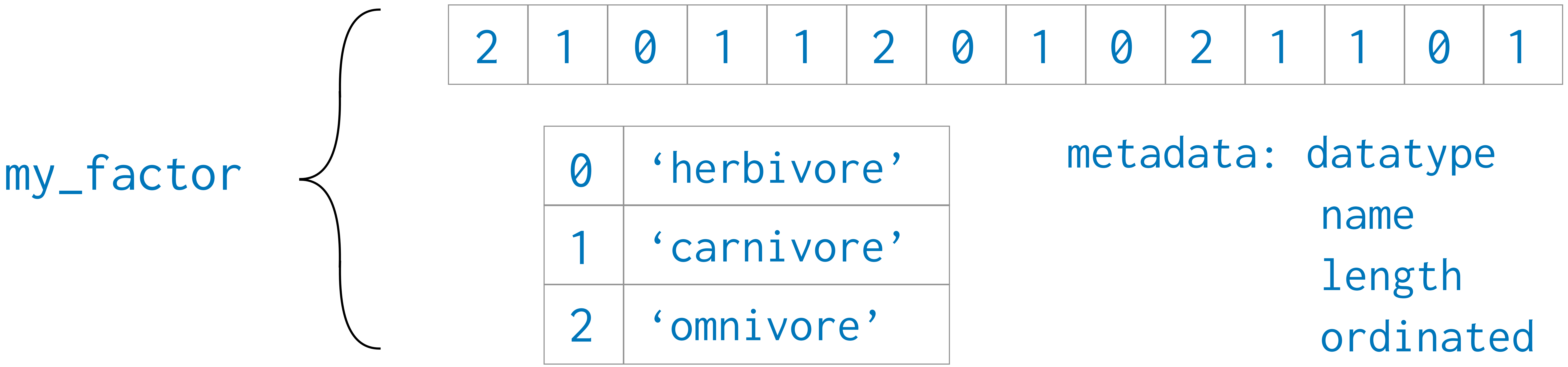
Why use factors?

- Memory-efficient: occupies less space than a character vector
- Computationally efficient: sort, group_by, and other operations are *much* faster
- Minimizes data entry errors: reports any non-standard values
- Required for some functions and 3rd-party packages

How do factors work?

Factors are structures with two data components, plus metadata

- Integer vector that stores the data
- Look-up table that maps integers to strings (optionally ordinated)



my_factor

2	1	0	1	1	2	0	1	0	2	1	1	0	1

0	‘herbivore’
1	‘carnivore’
2	‘omnivore’

```
metadata: datatype
          name
          length
          ordinated
```

Factors provide the best of both worlds

- For humans: displayed as strings, thus simple to understand
- For computer: manipulated as integers, thus very memory- and processor-efficient

Creating factors

Simplest method: pass a vector of strings to the `factor()` function

```
> my_fac <- factor(c("egg", "adult", "larva", "egg", "adult", "egg", "adult"))
> my_fac
[1] egg, adult, larva, egg, adult, egg, adult
Levels: adult egg larva
```

Learn about your new factor

```
> type_of(my_fac)
[1] "integer"
> class(my_fac)
[1] “factor”
> levels(my_fac)
[1] “adult”   "egg"   "larva"
```

Creating factors, continued

For more control, explicitly define the levels

```
> my_levels <- c("egg", “larva", “pupa”, "adult")
> my_values <- c("egg", "adult", "larva", "egg", "adult", "egg", "adult")
> my_fac <- factor(my_values, levels = my_levels)
> my_fac
[1] egg, adult, larva, egg, adult, egg, adult
Levels: egg larva pupa adult
```

To make it an ordered factor

```
> my_fac <- factor(my_values, levels = my_levels, ordered = TRUE)
> my_fac
[1] egg, adult, larva, egg, adult, egg, adult
Levels: egg < larva < pupa < adult
```

Modifying factors after they are created

To add a new level after a factor object has been created:

```
> levels(my_fac) <- c(levels(my_fac), "senescent")
> levels(my_fac)
[1] "egg"       "larva"     "pupa"      "adult"     "senescent"
```

To convert an unordered factor object to ordered:

```
> my_fac <- as.ordered(my_fac)
> my_fac
[1] egg, adult, larva, egg, adult, egg, adult, NA
Levels: egg < larva < adult
```

See the forcats package for additional functions that simplify working with factors

Utilizing factors

Once you have a factor column in a data frame, you can:

- Use `group_by()` to compute summary statistics by category
- Use `ggplot()` to display data by category as facets or distinct colors, shapes, etc.

You can also carry out logic and set operations using factors:

- Create a logical vector for presence / absence of a category
  - Example: `“adult”, "egg", "larva"` becomes `FALSE, TRUE, FALSE`
- Create a new factor vector with fewer levels, providing hierarchical grouping
  - Example: `“adult”, "egg", "larva"` becomes `"adult", "subadult”, "subadult"`
- Create new factor based on union, intersection, etc. of multiple factors
  - Example: female penguins on Briscoe Island with body mass > 3000g

Factoring and pivoting

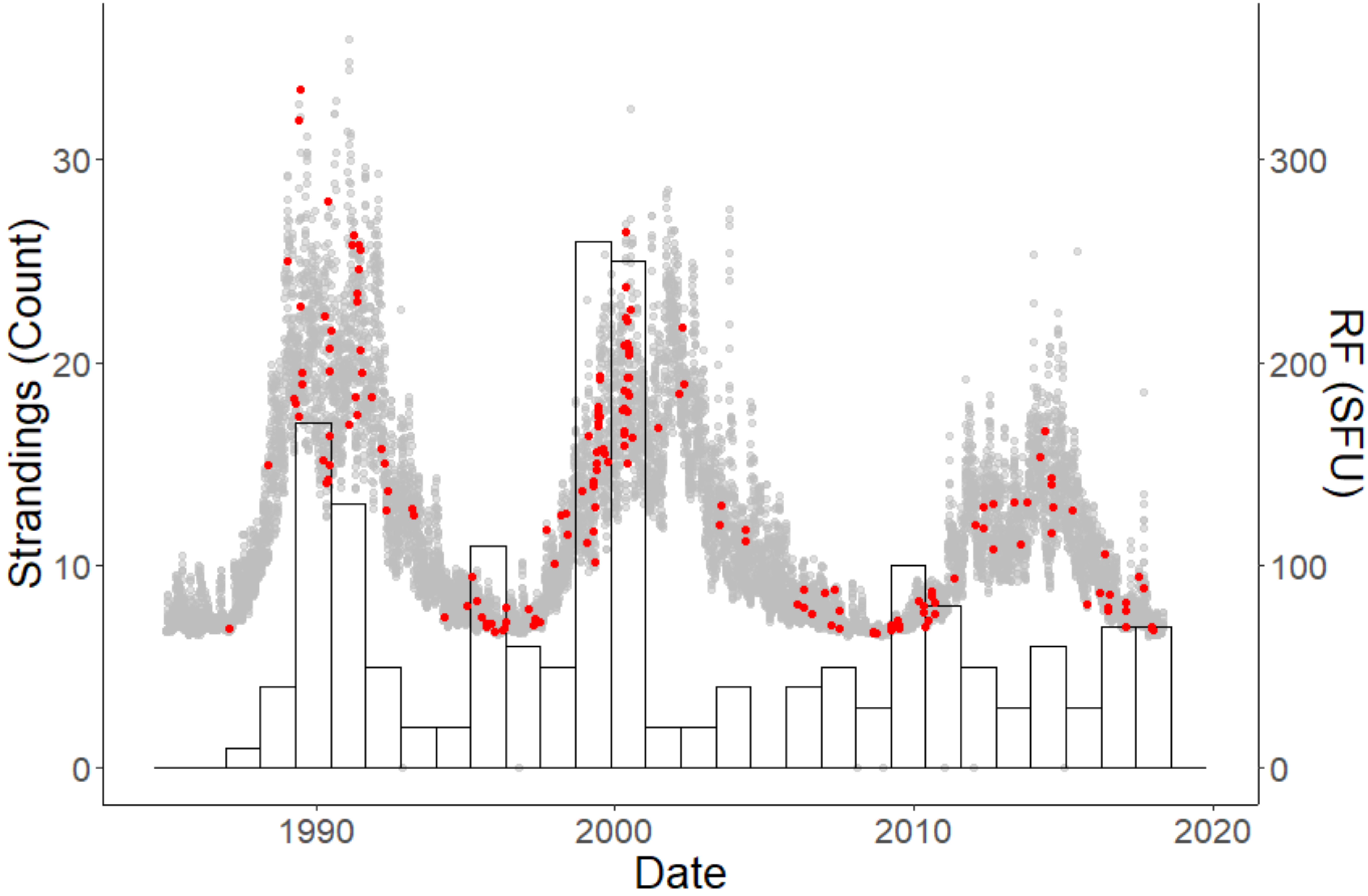
Refer to “In class: Factoring and Pivoting” on the class wiki page for this week

Joins and date objects

Where are we going?

Goals:

- Review how to JOIN datasets
- Practice dealing with DATE objects
- Make a cool figure



Getting Started

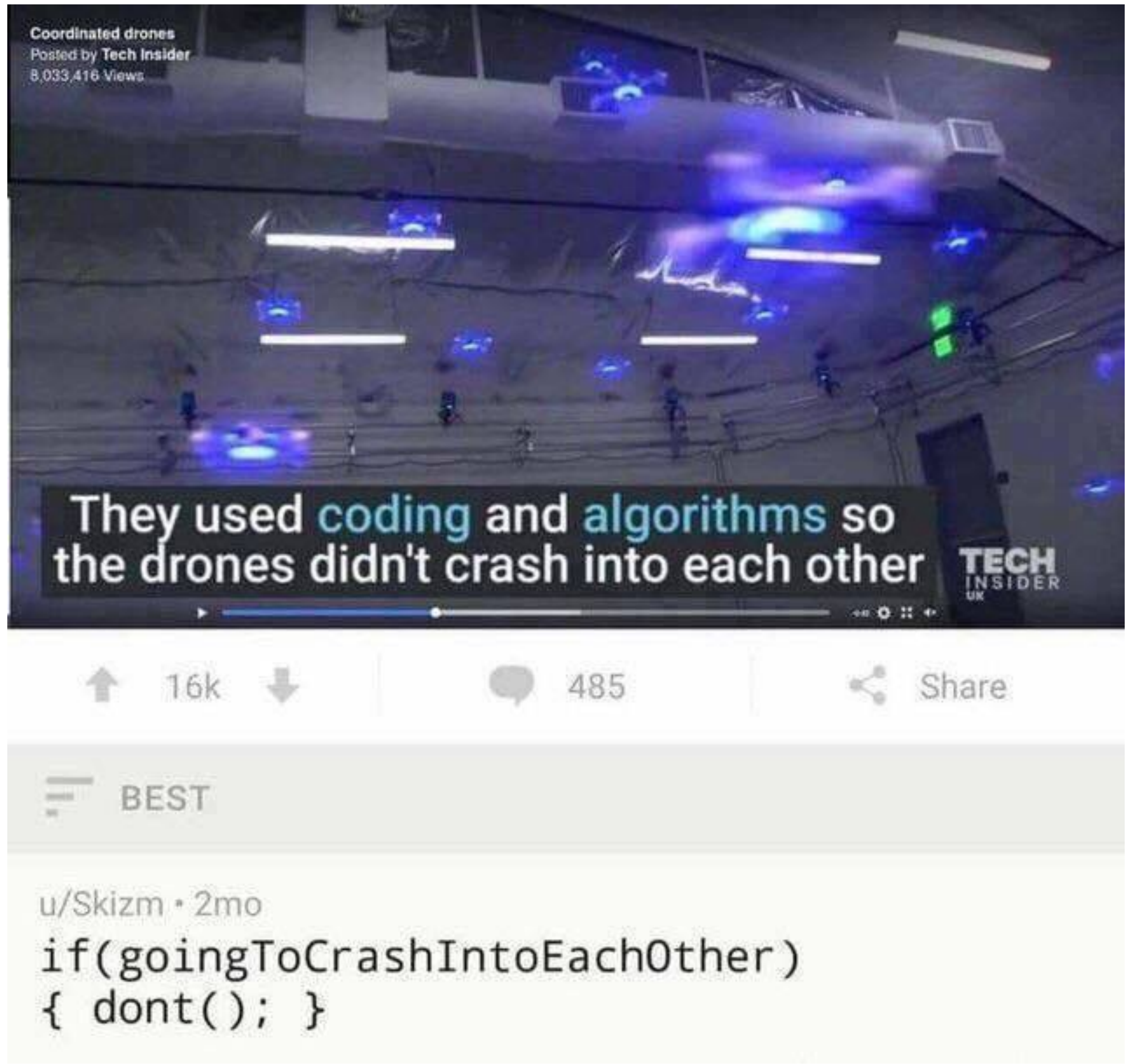
In a Quarto Document:

1) Import these two datasets from the website:

- “RF Data for Class.csv”
- “Whale Strandings for Class MODIFIED.csv”

2) Install (if needed) and load the packages “lubridate” and “tidyverse”

3) Open both the RF and the Stranding datasets, we will go over the columns together

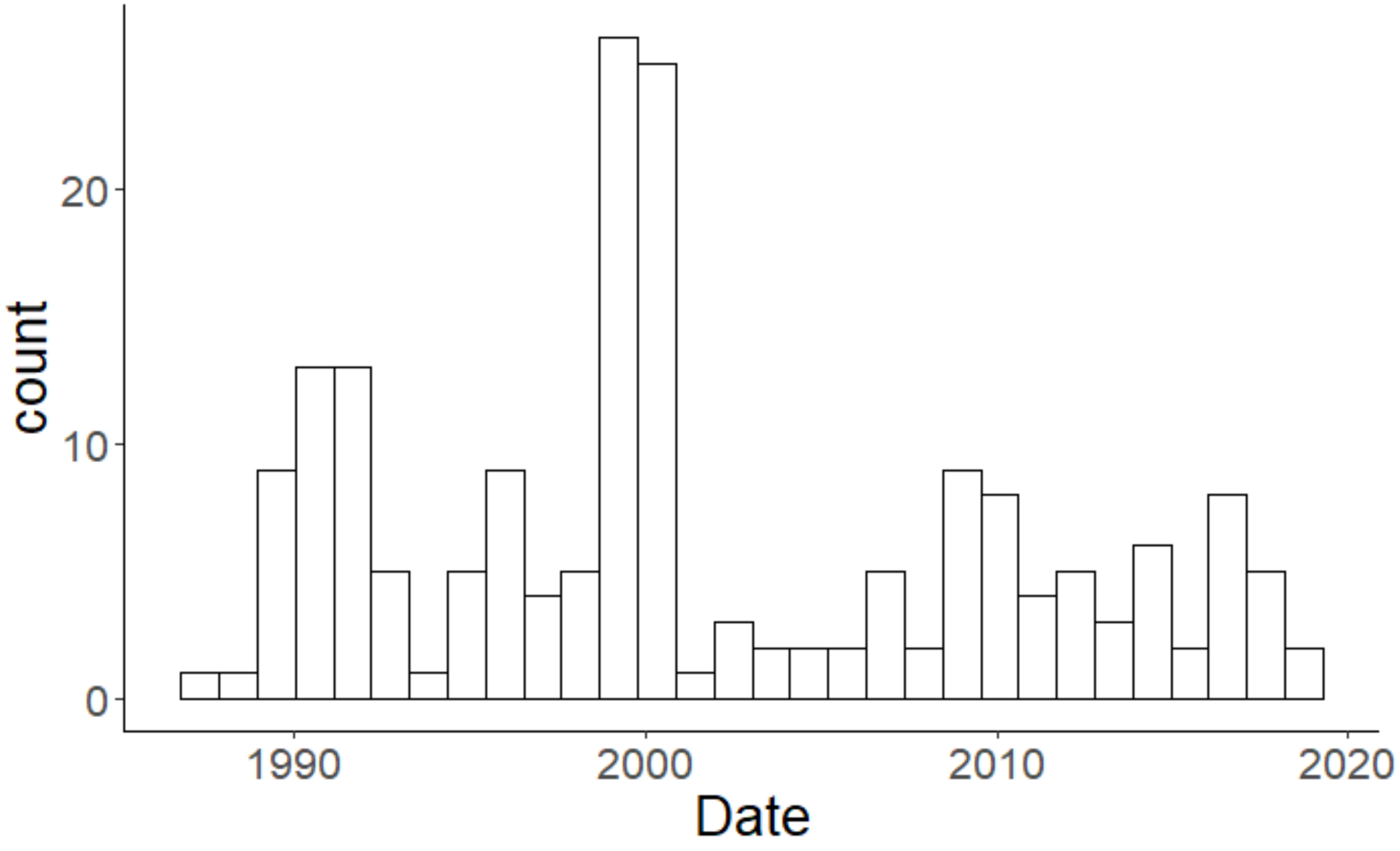


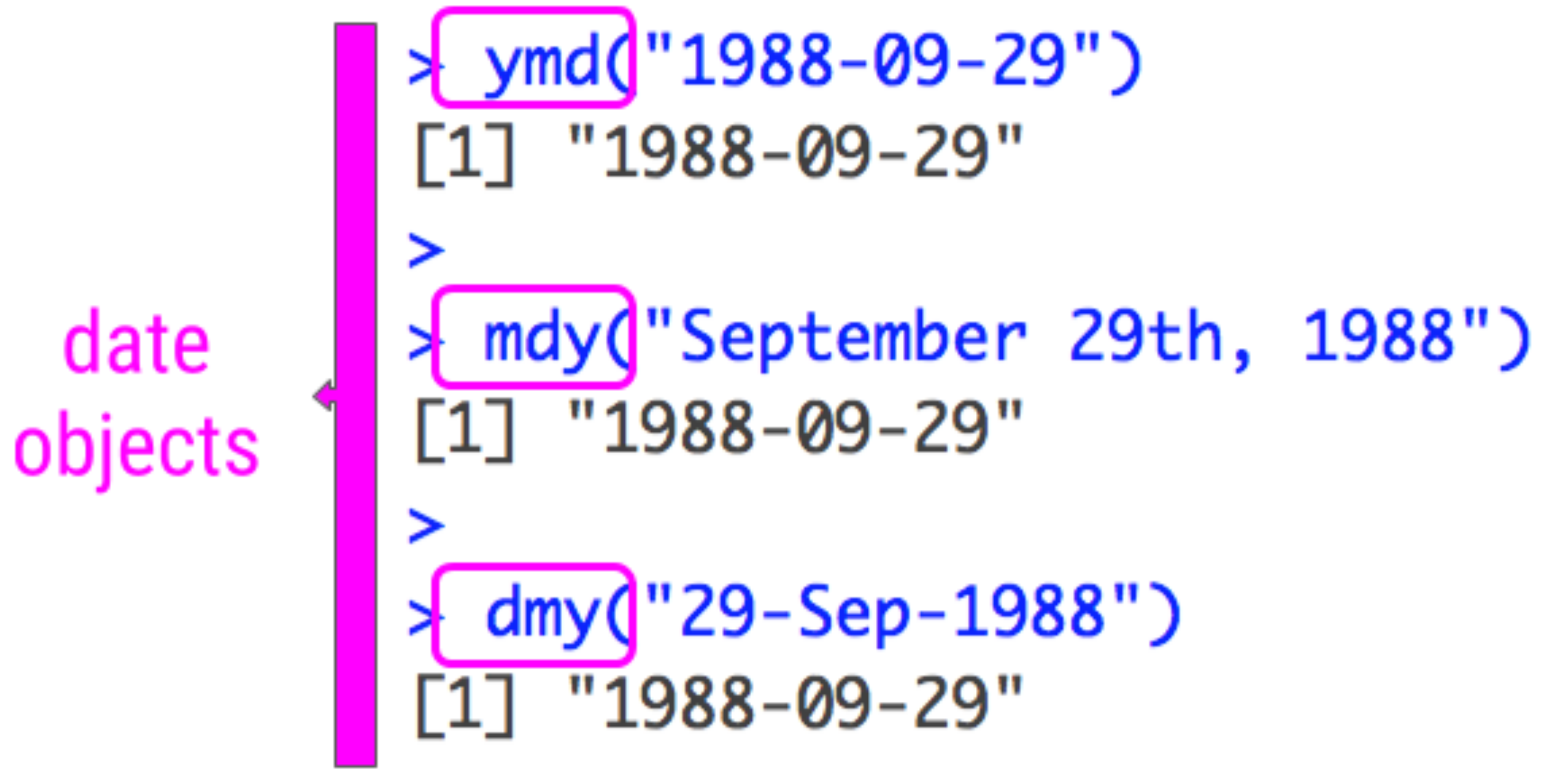
u/Skizm • 2mo

```
if(goingToCrashIntoEachOther)
{ dont(); }
```

Figure One—Stranding Histogram

- Try to make a histogram of the Stranding data using geom_histogram(aes(Date))
- What went wrong? Interpret the error message.
- Making a date object: Use mutate() and mdy() to make the “Date” column in the Stranding dataframe a date object
- Now, remake the stranding histogram. Use bins=30, make the fill transparent and the outline of the bins black. Set all text to size=20

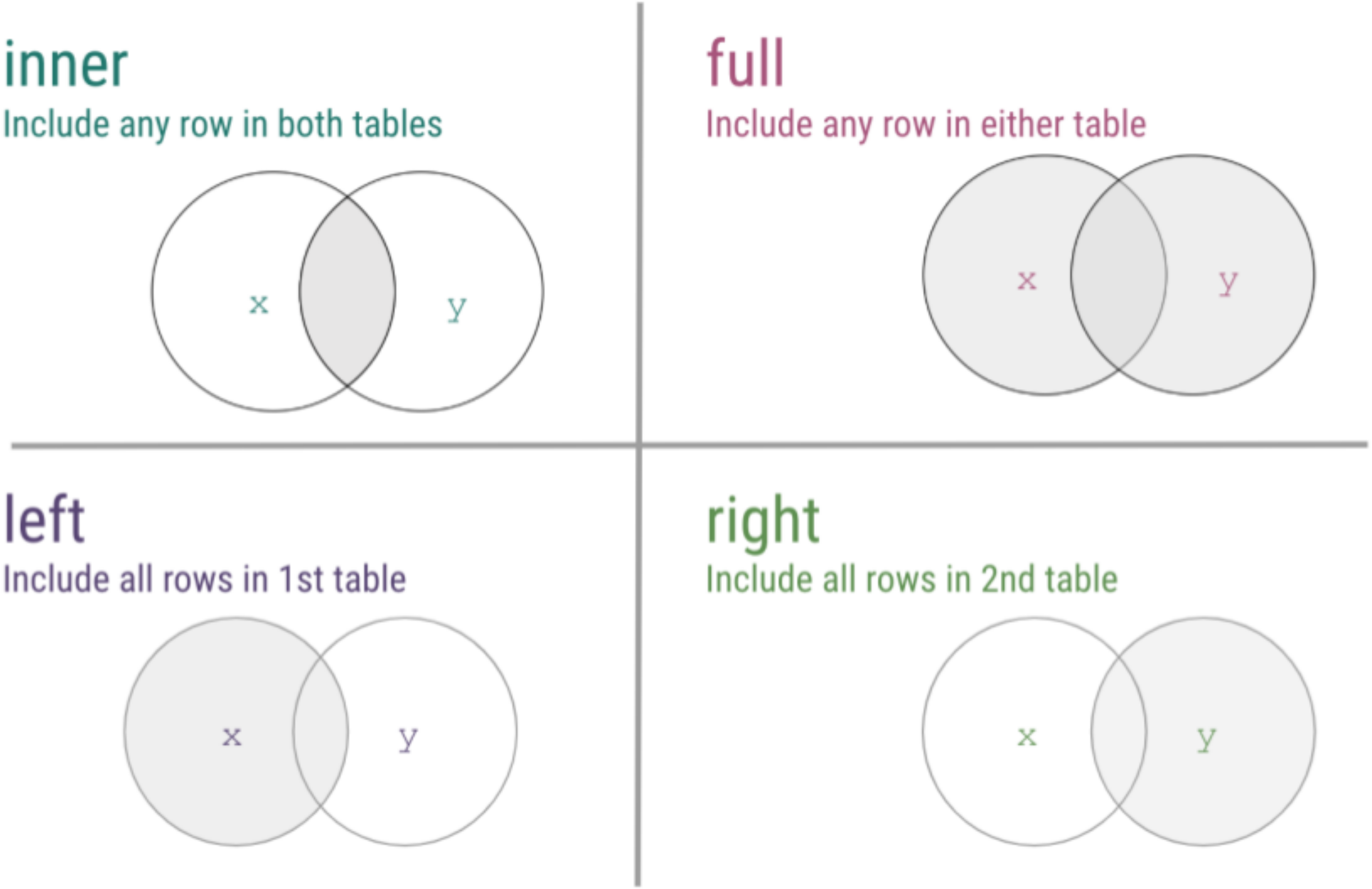




```
> ymd("1988-09-29")
[1] "1988-09-29"
>
> mdy("September 29th, 1988")
[1] "1988-09-29"
>
> dmy("29-Sep-1988")
[1] "1988-09-29"
```

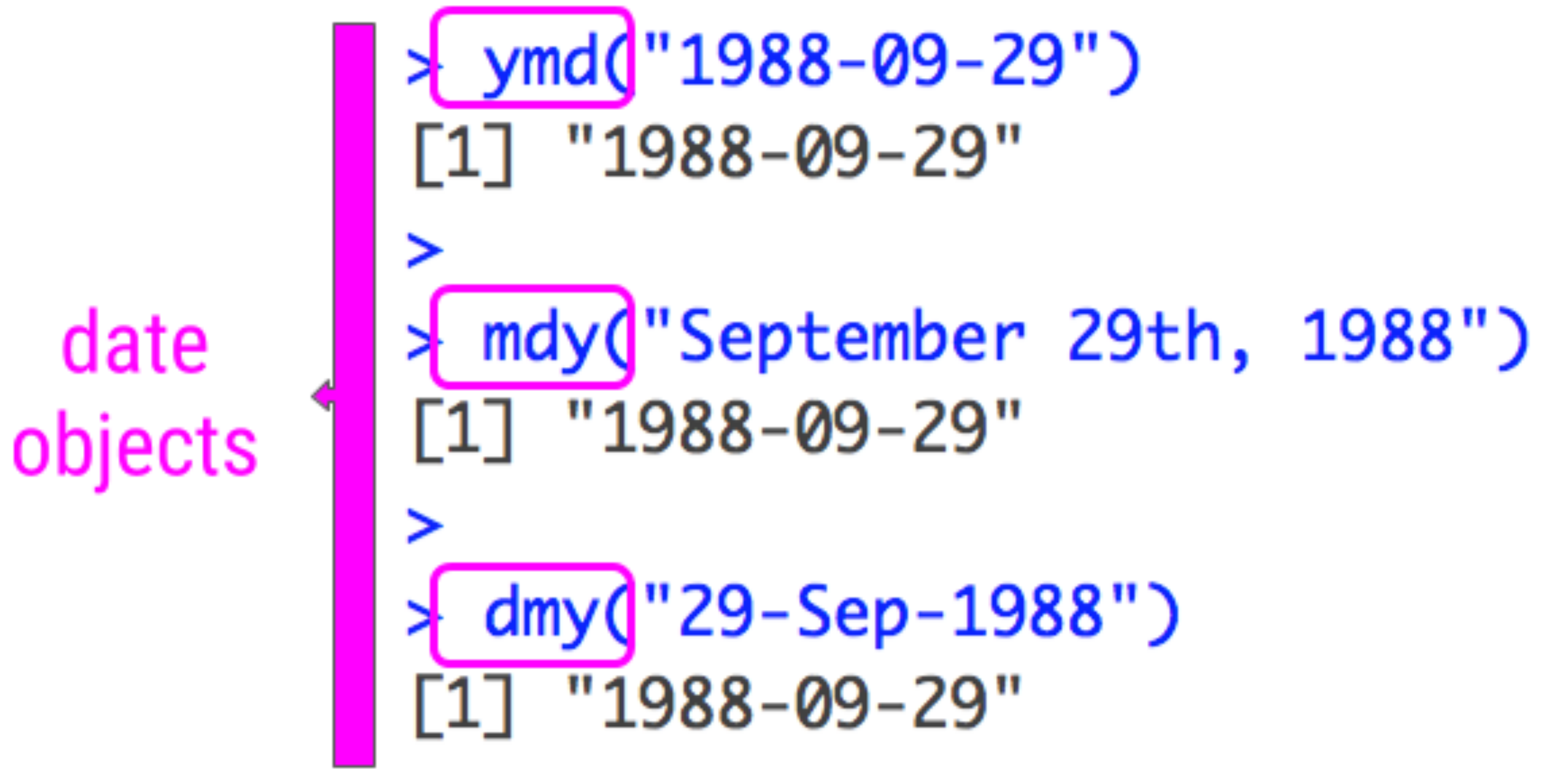
Joins

- Which kind of join should we use to get the RF on the days the whales stranded?
- Which column should we use to join by?
- Let's give it a shot. Use left_join() to try and merge the RF and Stranding data.
- What went wrong? Interpret the error message

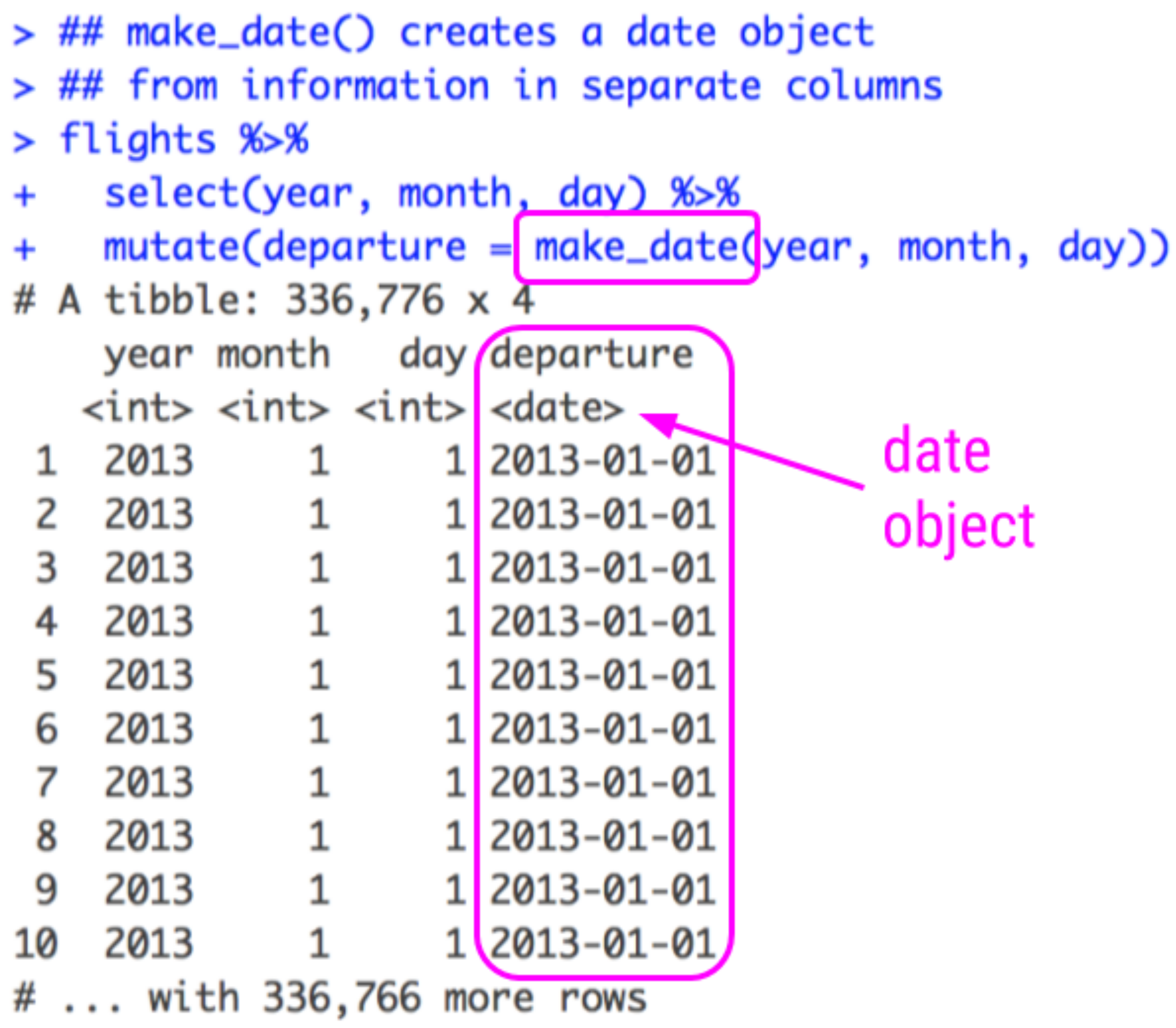

inner
Include any row in both tables
x
y
full
Include any row in either table
x
y
left
Include all rows in 1st table
x
y
right
Include all rows in 2nd table
x
y

Date Objects Take Two

- Use mutate() and make_date() to make the “Date” column in the RF dataframe a date object
- Now, retry the left_join()



```
> ymd("1988-09-29")
[1] "1988-09-29"
>
> mdy("September 29th, 1988")
[1] "1988-09-29"
>
> dmy("29-Sep-1988")
[1] "1988-09-29"
```



```
> ## make_date() creates a date object
> ## from information in separate columns
> flights %>%
+   select(year, month, day) %>%
+   mutate(departure = make_date(year, month, day))
# A tibble: 336,776 x 4
    year month   day departure
   <int> <int> <int> <date>
 1  2013     1     1 2013-01-01
 2  2013     1     1 2013-01-01
 3  2013     1     1 2013-01-01
 4  2013     1     1 2013-01-01
 5  2013     1     1 2013-01-01
 6  2013     1     1 2013-01-01
 7  2013     1     1 2013-01-01
 8  2013     1     1 2013-01-01
 9  2013     1     1 2013-01-01
10  2013     1     1 2013-01-01
# ... with 336,766 more rows
```

Figure Two—The RF Scatterplot

- Use the newly joined dataframe to make a scatterplot of the RF on the days when the whales stranded. Use geom_jitter(aes(Date,RF)
- What went wrong? Interpret the error message
- Remake the scatterplot, and make the points red, and the text size=20

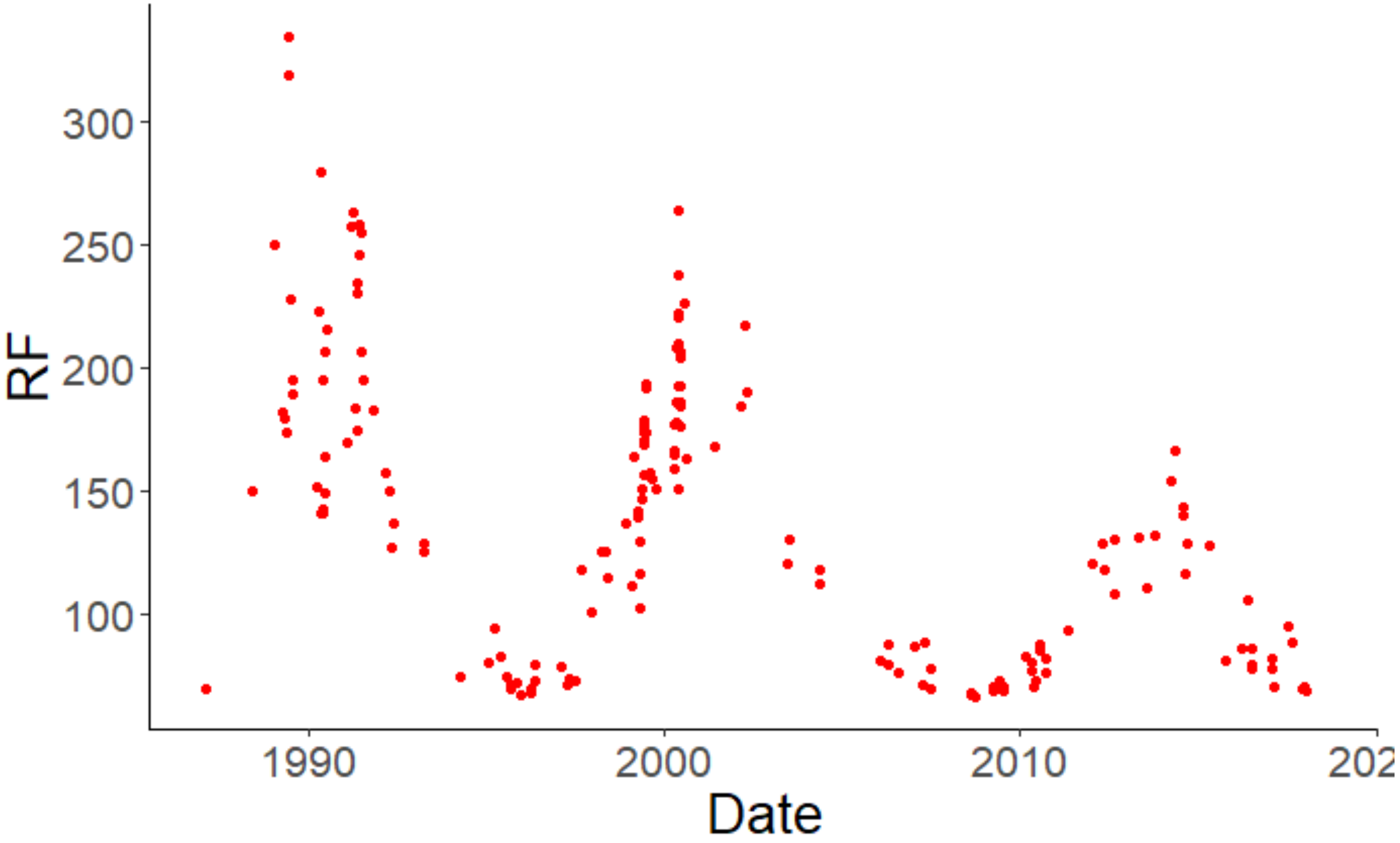
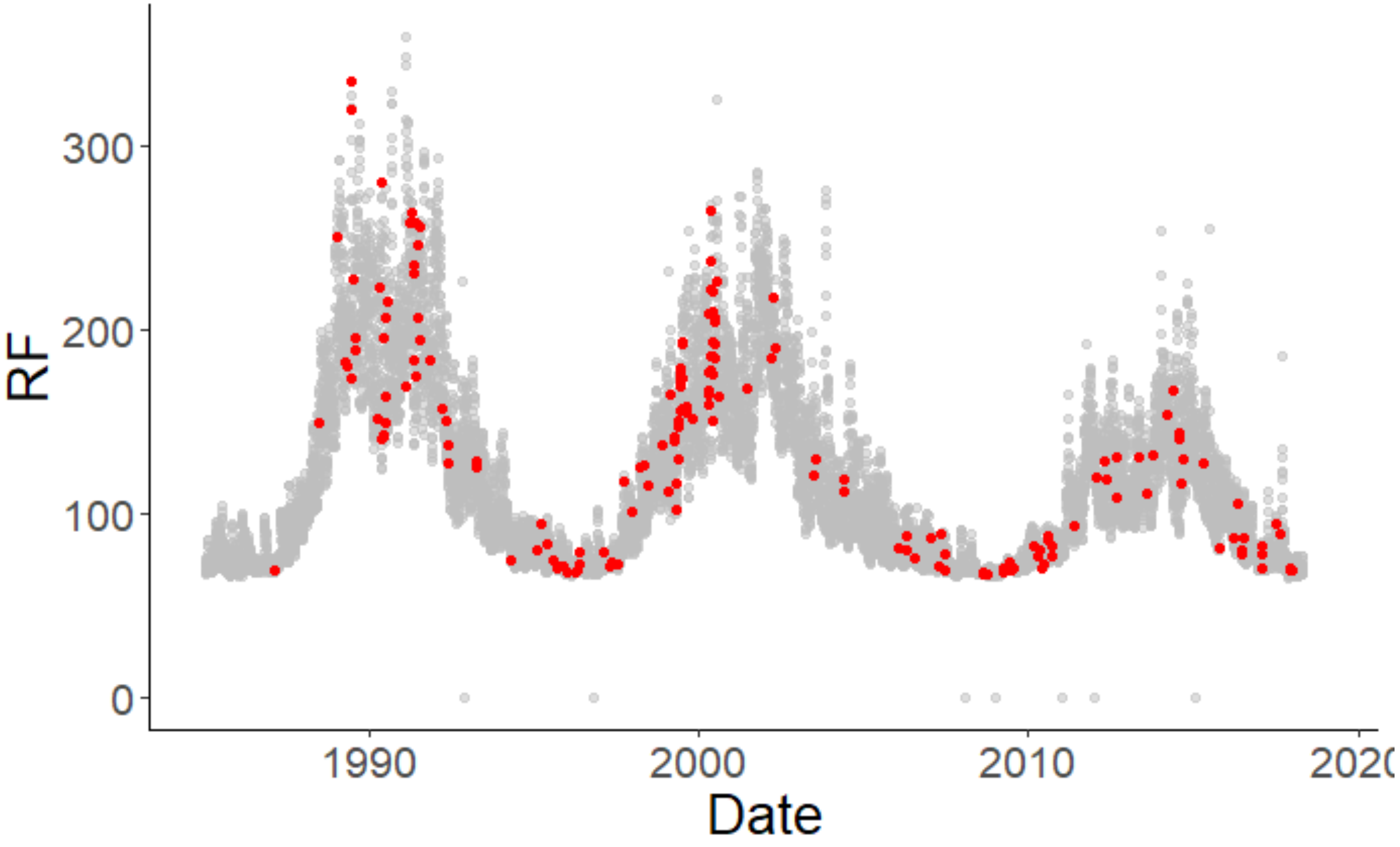


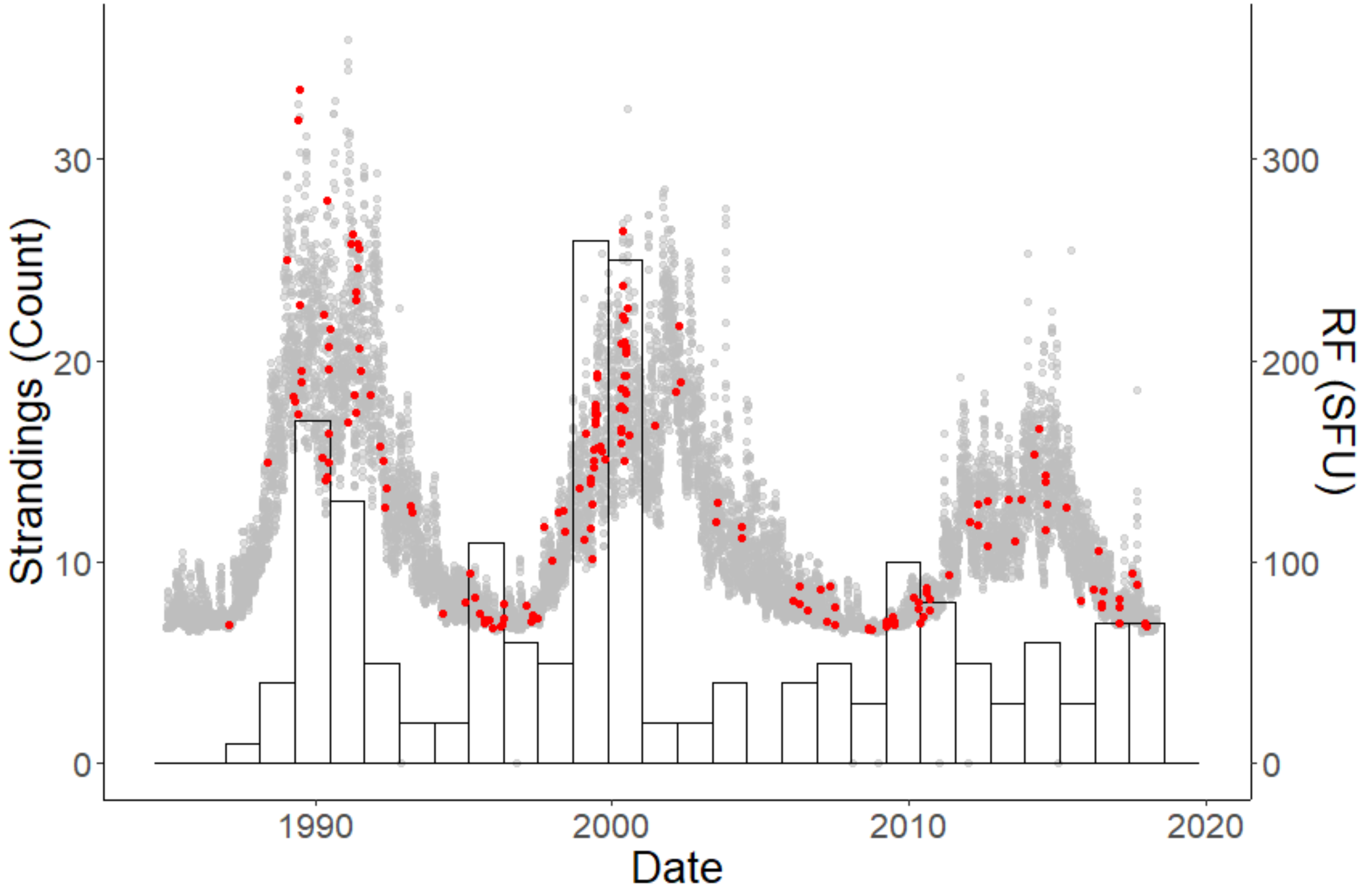
Figure Three—Plotting from two dataframes

- Use the original RF dataframe to add scatterplot with the RF levels across all dates. Use color=grey and alpha=0.5
- Things to think about: Where should you specify which dataframe you are using for each layer? Does the order matter when you add the layers?



Final Figure—Adding a second axis

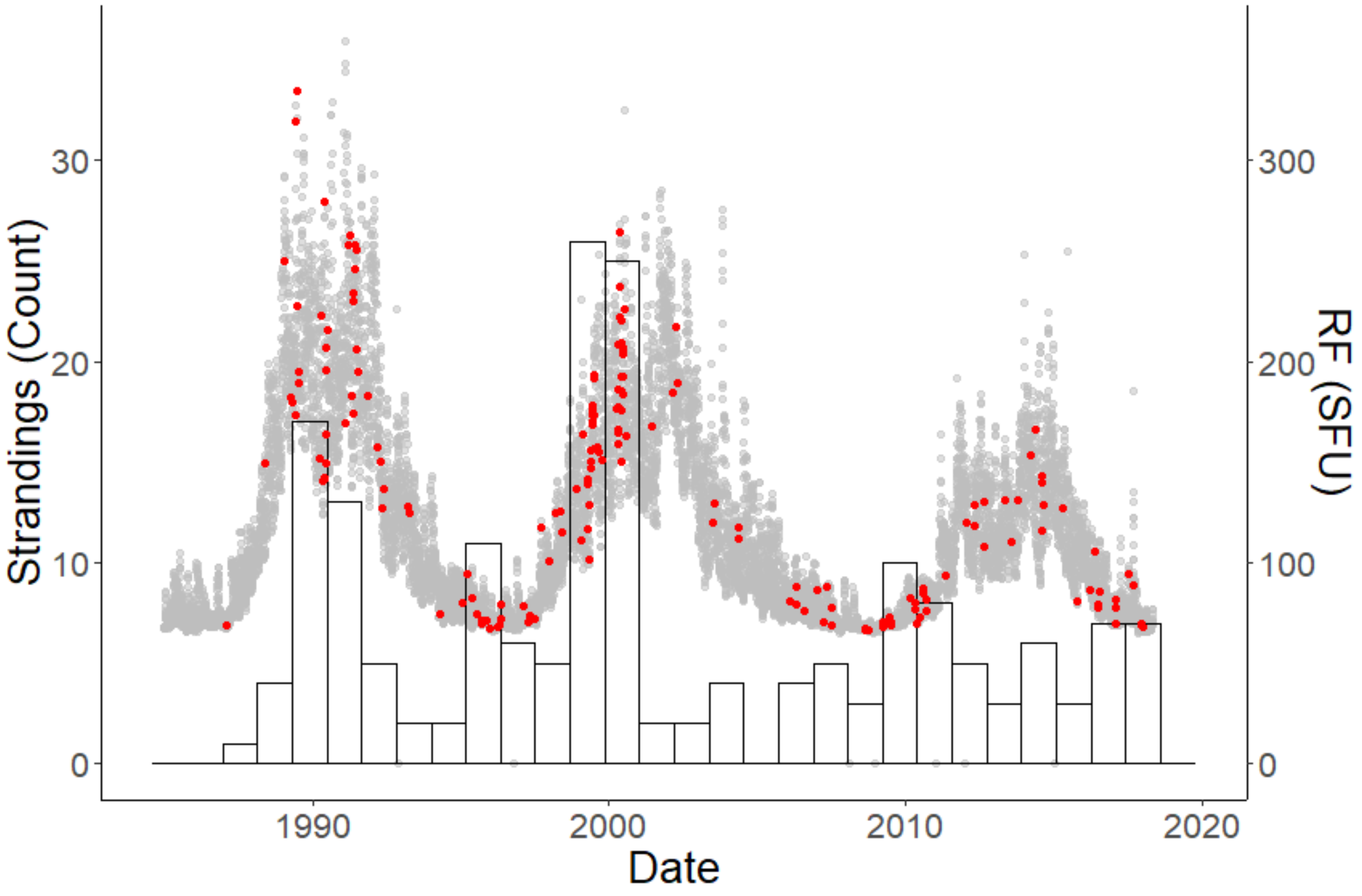
- Add a histogram of the strandings layer to your previous figure. What are some issues you notice?
- Adding a second axis in ggplot is not trivial. In fact, it is nearly impossible to add a second axis that is not a scaled version of the first axis. Why might this be?



Final Figure—Adding a second axis

```
coeff=10

ggplot()+
 theme_classic()+
 theme(text=element_text(size=20))+
 geom_jitter(data=RF.df, aes(Date, RF/coeff),
   alpha=0.5,color="grey")+
 geom_histogram(data=df,aes(Date), bins=30,
   fill="transparent", color="black")+
 geom_jitter(data=df, aes(Date, RF/coeff),
   color=“red")+
 scale_y_continuous(name="Strandings (Count)",
   sec.axis=sec_axis(~.*coeff, name="RF (SFU)"))
```


Strandings (Count)
Date
RF (SFU)