Week 3: Data Tables

Ellen Bledsoe

2025-02-04

2-dimensional Data and the `tidyverse`

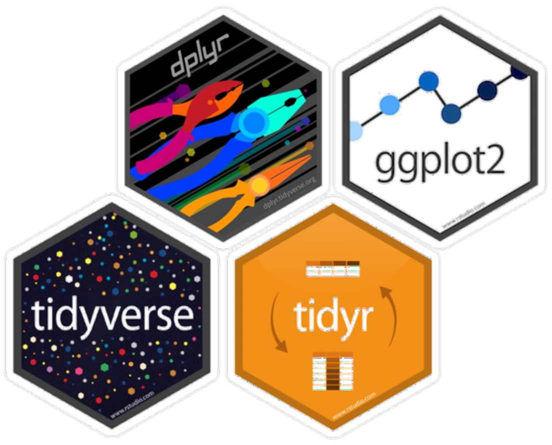
The `tidyverse`: What is it?

Different programming languages have different syntax (language structure). The `tidyverse` is a package (more accurately, a set of packages) offered in R that all have similar goals and a unified syntax designed to work particularly well with 2-dimensional data (data with rows and column). We call these 2-dimension data structures "data frames" in R.

Until now, all of the coding we have done is in the original R language, which is often called "base R." The syntax in the `tidyverse` is often pretty different from base R. Both are useful, and many people often combine them, which is why we start with base R.

Wait, what is a package??

Packages are one of the neatest things about working in an open-source environment like R! They contains bits of code (often in the form of functions) that can be reused, making them a core component of reproducible data science. Anyone can develop a package, and there are thousands of them doing all sorts of things.



Explore the `tidyverse`

If you want to learn more about the tidyverse, head over to www.tidyverse.org and browse the site. Below is a brief summary of *some* of the packages I think you might find the most useful.

- `tidyr`: creating data that is consistent in form/shape
- `dplyr`: creating data that is clean, easily wrangled, and summarized
- `ggplot2`: publication-worthy plots using The Grammar of Graphics
- `tibble`: data frames but better!
- `readr`: fast and friendly ways to read data into R
- `stringr`: easy manipulation of strings (character data)
- `lubridate`: easy manipulation of time and date values

Using `dplyr` and `readr`

Download and install

In most scenarios, you will need to download a package from the internet onto your computer before you can use it in RStudio. However, with Posit Cloud, I've already done this step for you!

For future reference, though:

- when using RStudio on your own computer (not on Posit Cloud), you usually only need to go through this process once until you update R
- we use the function `install.packages()` to download the package

```r
# download and install dplyr and readr
# to run the line of code, remove the # in front of the line below and run this chunk
# install.packages("dplyr")
# install.packages("readr")
```

Load into R

Any time we open R/RStudio and want to use functions from a package, we need to "load" the package. We use the `library()` function to do this.

```r
# load the tidyverse (tell RStudio we want to use this package in this session)
library(readr)
library(dplyr)
```

Set-Up

We are going to download to files into Posit Cloud for us to work with this week. Go ahead and run this code chunk. You should see new CSV files show up in the Files tab.

```r
# for the lesson
download.file("https://ndownloader.figshare.com/files/2292172", "surveys.csv")
download.file("https://ndownloader.figshare.com/files/3299474", "plots.csv")
download.file("https://ndownloader.figshare.com/files/3299483", "species.csv")

# for the assignment
download.file("http://www.datacarpentry.org/semester-biology/data/shrub-volume-data.csv", "shrub-volume
```

We've already talked about some of the benefits of CSV file. They work very nicely in R.

Click on `species.csv` and select View File. If we look at one of these files, we can see that it is plain text, so any program can read it. This makes it *interoperable*, which is an important tenant of reproducibility.

The first row is the header row, with different column headers separated by commas. All of the other rows are the data, again with different columns separated by commas. Hence the name "comma separated values."

Loading and Viewing the Data

We load these into R using a function from the `readr` package called `read_csv()`.

```
surveys <- read_csv("surveys.csv")
```

```
## Rows: 35549 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): species_id, sex
## dbl (7): record_id, month, day, year, plot_id, hindfoot_length, weight
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

```
species <- read_csv("species.csv")
```

```
## Rows: 54 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): species_id, genus, species, taxa
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

```
plots <- read_csv("plots.csv")
```

```
## Rows: 24 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): plot_type
## dbl (1): plot_id
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

The `tidyverse` (in this case, the `read_csv()` function) converts 2D data into something called a tibble! For our intents and purposes, it is basically the same as a data frame (and I'll probably call it a data frame, in reality).

We have three tables here:

- `surveys`: main table, one row for each rodent captured, date on date, location, species ID, sex, and size
- `species`: Latin species names for each species ID and general taxonic information
- `plots`: information on the experimental manipulations at the site

A few things to note about these tables:

- Good tabular data structure; one table per type of data
- Tables can be linked together to combine information.
- Each row contains a single record (single observation or data point)
- Each column or field contains a single attribute or type of information

We can explore the data frames in the Environment tab or through some functions.

```r
str(species)
```

```
## spc_tbl_ [54 x 4] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ species_id: chr [1:54] "AB" "AH" "AS" "BA" ...
##  $ genus     : chr [1:54] "Amphispiza" "Ammospermophilus" "Ammodramus" "Baiomys" ...
##  $ species   : chr [1:54] "bilineata" "harrisi" "savannarum" "taylori" ...
##  $ taxa      : chr [1:54] "Bird" "Rodent" "Bird" "Rodent" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   species_id = col_character(),
##   ..   genus = col_character(),
##   ..   species = col_character(),
##   ..   taxa = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```r
names(species)
```

```
## [1] "species_id" "genus"      "species"    "taxa"
```

```r
head(species)
```

```
## # A tibble: 6 x 4
##   species_id genus            species         taxa
##   <chr>      <chr>            <chr>           <chr>
## 1 AB         Amphispiza       bilineata       Bird
## 2 AH         Ammospermophilus harrisi         Rodent
## 3 AS         Ammodramus       savannarum      Bird
## 4 BA         Baiomys          taylori         Rodent
## 5 CB         Campylorhynchus  brunneicapillus Bird
## 6 CM         Calamospiza      melanocorys     Bird
```

```r
glimpse(species) # from dplyr
```

```
## Rows: 54
## Columns: 4
## $ species_id <chr> "AB", "AH", "AS", "BA", "CB", "CM", "CQ", "CS", "CT", "CU",~
## $ genus      <chr> "Amphispiza", "Ammospermophilus", "Ammodramus", "Baiomys", ~
## $ species    <chr> "bilineata", "harrisi", "savannarum", "taylori", "brunneica~
## $ taxa       <chr> "Bird", "Rodent", "Bird", "Rodent", "Bird", "Bird", "Bird",~
```

Subsetting in base R

Before we jump into using the `tidyverse`, let's briefly explore how we would subset 2D data using base R. As you might recall, we use `[]` in base R to specify that we want a smaller part of the data.

With data frames, we need to specify 2-dimensions of the data (row and column).

```
# the order is [row, column]
species[2, 1]
```

```
## # A tibble: 1 x 1
##   species_id
##   <chr>
## 1 AH
```

If you want to retain *all* rows or *all* columns, you can leave that space blank.

```
# return first 6 rows and all columns
species[1:6, ]
```

```
## # A tibble: 6 x 4
##   species_id genus            species          taxa
##   <chr>      <chr>            <chr>            <chr>
## 1 AB         Amphispiza       bilineata        Bird
## 2 AH         Ammospermophilus harrisi          Rodent
## 3 AS         Ammodramus       savannarum       Bird
## 4 BA         Baiomys          taylori          Rodent
## 5 CB         Campylorhynchus  brunneicapillus  Bird
## 6 CM         Calamospiza      melanocorys      Bird
```

```
# return all values in the first column
species[ , 1]
```

```
## # A tibble: 54 x 1
##    species_id
##    <chr>
##  1 AB
##  2 AH
##  3 AS
##  4 BA
##  5 CB
##  6 CM
##  7 CQ
##  8 CS
##  9 CT
## 10 CU
## # i 44 more rows
```

There is a special way to pull out a single column from a data frame and have it be treated as a vector (1-dimensional data). We use a special operator, `$`.

```
species$species_id
```

```
##  [1] "AB" "AH" "AS" "BA" "CB" "CM" "CQ" "CS" "CT" "CU" "CV" "DM" "DO" "DS" "DX"
## [16] "EO" "GS" "NL" "NX" "OL" "OT" "OX" "PB" "PC" "PE" "PF" "PG" "PH" "PI" "PL"
## [31] "PM" "PP" "PU" "PX" "RF" "RM" "RO" "RX" "SA" "SB" "SC" "SF" "SH" "SO" "SS"
## [46] "ST" "SU" "SX" "UL" "UP" "UR" "US" "ZL" "ZM"
```

Intro to `dplyr`

`dplyr` (pronounced D-ply-R) is a modern data manipulation library for R that uses the syntax of the `tidyverse`. It tends to be a bit more intuitive than using base R, especially ask tasks become more complicated.

`select()`ing columns

Let's use our first function, `select()`. Select allows us to pick out specific columns from our data. You can use names or their position in the data frame.

The first argument in the function is the data frame. Any following arguments are the columns we want to select.

```
# select one column
select(surveys, year)
```

```
## # A tibble: 35,549 x 1
##     year
##    <dbl>
##  1  1977
##  2  1977
##  3  1977
##  4  1977
##  5  1977
##  6  1977
##  7  1977
##  8  1977
##  9  1977
## 10  1977
## # i 35,539 more rows
```

```
# select multiple columns (in any order)
select(surveys, year, month, day)
```

```
## # A tibble: 35,549 x 3
##     year month   day
##    <dbl> <dbl> <dbl>
##  1  1977     7    16
##  2  1977     7    16
##  3  1977     7    16
##  4  1977     7    16
##  5  1977     7    16
##  6  1977     7    16
##  7  1977     7    16
##  8  1977     7    16
##  9  1977     7    16
## 10  1977     7    16
## # i 35,539 more rows
```

```
select(surveys, month, day, year)
```

```
## # A tibble: 35,549 x 3
##    month   day  year
##    <dbl> <dbl> <dbl>
##  1     7    16  1977
##  2     7    16  1977
##  3     7    16  1977
##  4     7    16  1977
##  5     7    16  1977
##  6     7    16  1977
##  7     7    16  1977
##  8     7    16  1977
##  9     7    16  1977
## 10     7    16  1977
## # i 35,539 more rows
```

```
# select a consecutive columns
select(surveys, month:year)
```

```
## # A tibble: 35,549 x 3
##    month   day  year
##    <dbl> <dbl> <dbl>
##  1     7    16  1977
##  2     7    16  1977
##  3     7    16  1977
##  4     7    16  1977
##  5     7    16  1977
##  6     7    16  1977
##  7     7    16  1977
##  8     7    16  1977
##  9     7    16  1977
## 10     7    16  1977
## # i 35,539 more rows
```

```
# remove columns
select(surveys, -weight)
```

```
## # A tibble: 35,549 x 8
##    record_id month   day  year plot_id species_id sex   hindfoot_length
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>
##  1         1     7    16  1977       2 NL         M                  32
##  2         2     7    16  1977       3 NL         M                  33
##  3         3     7    16  1977       2 DM         F                  37
##  4         4     7    16  1977       7 DM         M                  36
##   5          5     7    16  1977       3 DM         M                    35
##   6          6     7    16  1977       1 PF         M                    14
##   7          7     7    16  1977       2 PE         F                    NA
##   8          8     7    16  1977       1 DM         M                    37
##   9          9     7    16  1977       1 DM         F                    34
##  10         10     7    16  1977       6 PF         F                    20
## # i 35,539 more rows
```

It is important to remember that the computer interprets everything literally. We need to tell the function the **exact** names of the columns.

Let's Practice!

Get started with your Assignment. After the set-up, work on Question 1a-b.

Creating new columns with `mutate()`

Sometimes our data doesn't have our data in exactly the format we want. For example, we might want our hindfoot data in cm instead of mm.

The `dplyr` function called `mutate()` lets us create a new column.

The first part of the argument in the mutate function (before the `=`) is the name of the new column we want to create (or, sometimes, the name of a column we want to overwrite). After the `=` is what we want the new column to contain.

```r
mutate(surveys, hindfoot_length_cm = hindfoot_length / 10)
```

```
## # A tibble: 35,549 x 10
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1         1     7    16  1977       2 NL         M                  32     NA
##  2         2     7    16  1977       3 NL         M                  33     NA
##  3         3     7    16  1977       2 DM         F                  37     NA
##  4         4     7    16  1977       7 DM         M                  36     NA
##  5         5     7    16  1977       3 DM         M                    35     NA
##  6         6     7    16  1977       1 PF         M                    14     NA
##  7         7     7    16  1977       2 PE         F                    NA     NA
##  8         8     7    16  1977       1 DM         M                    37     NA
##  9         9     7    16  1977       1 DM         F                    34     NA
## 10        10     7    16  1977       6 PF         F                    20     NA
## # i 35,539 more rows
## # i 1 more variable: hindfoot_length_cm <dbl>
```

If we look at the surveys object, will it contain the new column?

To store the results of these functions for later, use we need to assign them to a new object or overwrite the existing object.

```
surveys_cm <- mutate(surveys, hindfoot_length_cm = hindfoot_length / 10)
```

Sorting data with arrange()

We can sort the data in the table using the arrange() function. Let's sort the surveys table by weight.

```
arrange(surveys, weight)
```

```
## # A tibble: 35,549 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1       218     9    13  1977       1 PF         M                  13      4
##  2      4052     4     5  1981       3 PF         F                  15      4
##  3      4290     4     6  1981       4 PF         <NA>               NA      4
##  4      5346     2    22  1982      21 PF         F                  14      4
##  5      7084    11    22  1982       3 PF         F                  16      4
##  6      8736    12     8  1983      19 RM         M                  17      4
##  7      9790     1    19  1985      16 RM         F                  16      4
##  8      9794     1    19  1985      24 RM         M                  16      4
##  9      9799     1    19  1985      19 RM         M                  16      4
## 10      9823     1    19  1985      23 RM         M                  16      4
## # i 35,539 more rows
```

We can see that the rows are now in order from the smallest weight to the largest.

We can reverse the order of the sort by "wrapping" weight in another function: desc() for "descending"

```
arrange(surveys, desc(weight))
```

```
## # A tibble: 35,549 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1     33049    11    17  2001      12 NL         M                  33    280
##  2     12871     5    28  1987       2 NL         M                  32    278
##  3     15459     1    11  1989       9 NL         M                  36    275
##  4      2133    10    25  1979       2 NL         F                  33    274
##  5     12729     4    26  1987       2 NL         M                  32    270
##  6     13114     7    26  1987       2 NL         M                  NA    269
##   7      30175     1     8  2000        2 NL         M                    34    265
##   8       4962    11    22  1981       12 NL         F                    NA    264
##   9      12602     4     6  1987        2 NL         M                    34    260
##  10      13025     7     1  1987        2 NL         M                    33    260
## # i 35,539 more rows
```

We can also sort by multiple columns. Perhaps we want to sort first by `plot_id` and then by the date.

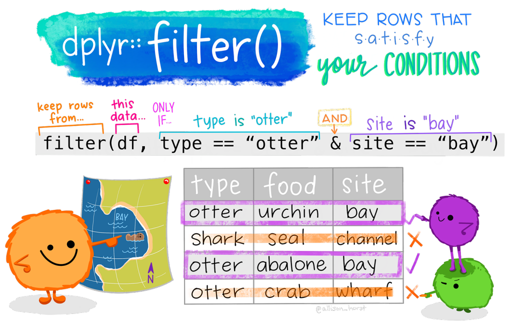
```r
arrange(surveys, plot_id, year, month, day)
```

```
## # A tibble: 35,549 x 9
##     record_id month   day  year plot_id species_id sex   hindfoot_length weight
##         <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##   1         6     7    16  1977       1 PF         M                  14     NA
##   2         8     7    16  1977       1 DM         M                  37     NA
##   3         9     7    16  1977       1 DM         F                  34     NA
##   4        78     8    19  1977       1 PF         M                  16      9
##   5        80     8    19  1977       1 DS         M                  48     NA
##   6       218     9    13  1977       1 PF         M                  13      4
##   7       222     9    13  1977       1 DS         M                  52     NA
##   8       239     9    13  1977       1 DS         M                  48     NA
##   9       263    10    16  1977       1 DM         M                  37     40
##  10       270    10    16  1977       1 DM         F                  36     38
## # i 35,539 more rows
```

Let's Practice

Work on Question 1c-d.

`filter()`ing rows



```
filter(df, type == "otter" & site == "bay")
```

type	food	site
otter	urchin	bay
shark	seal	channel
otter	abalone	bay
otter	crab	wharf

The `filter()` function allows you filter rows by certain conditions.

We start with the data frame, then we set a condition that has to be met. Let's say we want only rows for the species ID "DS."

To set a condition, we start with the name of the column the want to filter based upon, species_id. We then use `==` to set the condition.

Our condition is that we want rows with the value "DS" in the species_id column. "DS" here is a string (character data), not a variable or a column name, so we enclose it in quotation marks.

```r
# base R
# surveys[surveys$species_id == "DS", ]

# dplyr
filter(surveys, species_id == "DS")
```

```
## # A tibble: 2,504 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1        11     7    16  1977       5 DS         F                  53     NA
##  2        17     7    16  1977       3 DS         F                  48     NA
##  3        20     7    17  1977      11 DS         F                  48     NA
##  4        30     7    17  1977      10 DS         F                  52     NA
##  5        42     7    18  1977      18 DS         F                  46     NA
##  6        58     7    18  1977      12 DS         M                  45     NA
##  7        73     8    19  1977       3 DS         F                  44     NA
##  8        76     8    19  1977       9 DS         F                  47     NA
##  9        80     8    19  1977       1 DS         M                  48     NA
## 10        91     8    20  1977      11 DS         F                  50     NA
## # i 2,494 more rows
```

Like with vectors, we can have a condition that is "not equal to" using "!=" Perhaps we want the data for all species except "DS."

```r
filter(surveys, species_id != "DS")
```

```
## # A tibble: 32,282 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1         1     7    16  1977       2 NL         M                  32     NA
##  2         2     7    16  1977       3 NL         M                  33     NA
##  3         3     7    16  1977       2 DM         F                  37     NA
##  4         4     7    16  1977       7 DM         M                  36     NA
##  5         5     7    16  1977       3 DM         M                  35     NA
##  6         6     7    16  1977       1 PF         M                  14     NA
##  7         7     7    16  1977       2 PE         F                  NA     NA
##  8         8     7    16  1977       1 DM         M                  37     NA
##  9         9     7    16  1977       1 DM         F                  34     NA
## 10        10     7    16  1977       6 PF         F                  20     NA
## # i 32,272 more rows
```

We can also filter on multiple conditions at once.

In computing, we combine conditions in two ways: "and" & "or"

Using "and" means that all of the conditions must be true. Do this in `dplyr`, we can add arguments separated by commas or use the `&` symbol.

To get the data on species "DS" for the year 1995:

```r
# same thing
filter(surveys, species_id == "DS", year > 1995)
```

```
## # A tibble: 22 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1     26304     7     9  1997       2 DS         F                  50    113
##  2     26571     7    29  1997       2 DS         F                  49    118
##  3     26792     9    27  1997       2 DS         F                  49    113
##  4     26980    10    25  1997       2 DS         F                  50    108
##  5     27163    11    22  1997       2 DS         F                  50    103
##  6     27306    12    28  1997       2 DS         F                  51    111
##  7     27426     1    31  1998       2 DS         F                  51    122
##  8     27553     3     1  1998       2 DS         M                  50     NA
##  9     27588     3     1  1998       1 DS         M                  46     NA
## 10     28013     6    27  1998       2 DS         F                  51    106
## # i 12 more rows
```

```
filter(surveys, species_id == "DS" & year > 1995)
```

```
## # A tibble: 22 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1     26304     7     9  1997       2 DS         F                  50    113
##  2     26571     7    29  1997       2 DS         F                  49    118
##  3     26792     9    27  1997       2 DS         F                  49    113
##  4     26980    10    25  1997       2 DS         F                  50    108
##  5     27163    11    22  1997       2 DS         F                  50    103
##  6     27306    12    28  1997       2 DS         F                  51    111
##  7     27426     1    31  1998       2 DS         F                  51    122
##  8     27553     3     1  1998       2 DS         M                  50     NA
##  9     27588     3     1  1998       1 DS         M                  46     NA
## 10     28013     6    27  1998       2 DS         F                  51    106
## # i 12 more rows
```

This approach is mostly useful for building more complex conditions.

When we want rows for which one or more of the conditions are met, we use “or”, which is the `|` symbol.

```
# compare to the code above
filter(surveys, species_id == "DS" | year > 1995)
```

```
## # A tibble: 14,817 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1        11     7    16  1977       5 DS         F                  53     NA
##  2        17     7    16  1977       3 DS         F                  48     NA
##  3        20     7    17  1977      11 DS         F                  48     NA
##  4        30     7    17  1977      10 DS         F                  52     NA
##  5        42     7    18  1977      18 DS         F                  46     NA
##  6        58     7    18  1977      12 DS         M                  45     NA
##  7        73     8    19  1977       3 DS         F                  44     NA
##  8        76     8    19  1977       9 DS         F                  47     NA
##  9        80     8    19  1977       1 DS         M                  48     NA
## 10        91     8    20  1977      11 DS         F                  50     NA
## # i 14,807 more rows
```

```r
# if we want multiple options from the same column, need to use "or"
filter(surveys, species_id == "DS" | species_id == "DM" | species_id == "DO")
```

```
## # A tibble: 16,127 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1         3     7    16  1977       2 DM         F                  37     NA
##  2         4     7    16  1977       7 DM         M                  36     NA
##  3         5     7    16  1977       3 DM         M                  35     NA
##  4         8     7    16  1977       1 DM         M                  37     NA
##  5         9     7    16  1977       1 DM         F                  34     NA
##  6        11     7    16  1977       5 DS         F                  53     NA
##  7        12     7    16  1977       7 DM         M                  38     NA
##  8        13     7    16  1977       3 DM         M                  35     NA
##  9        14     7    16  1977       8 DM         <NA>               NA     NA
## 10        15     7    16  1977       6 DM         F                  36     NA
## # i 16,117 more rows
```

Let's Practice

Work on Question 1e-g.

`filter()`ing null values

One of the common tasks we use `filter` for is removing null values from data. Based on what we learned before, it's natural to think that we do this by using the condition `weight != NA`.

```r
filter(surveys, weight != NA)
```

Why didn't that work? Null values like `NA` are special. Instead, we use a special function, `is.na()`.

```r
filter(surveys, is.na(weight))
```

```
## # A tibble: 3,266 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1         1     7    16  1977       2 NL         M                  32     NA
##  2         2     7    16  1977       3 NL         M                  33     NA
##  3         3     7    16  1977       2 DM         F                  37     NA
##  4         4     7    16  1977       7 DM         M                  36     NA
##  5         5     7    16  1977       3 DM         M                  35     NA
##  6         6     7    16  1977       1 PF         M                  14     NA
##  7         7     7    16  1977       2 PE         F                  NA     NA
##  8         8     7    16  1977       1 DM         M                  37     NA
##  9         9     7    16  1977       1 DM         F                  34     NA
## 10        10     7    16  1977       6 PF         F                  20     NA
## # i 3,256 more rows
```

To remove null values, we combine this with `!` for "not"

```r
filter(surveys, !is.na(weight))
```

```
## # A tibble: 32,283 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1        63     8    19  1977       3 DM         M                  35     40
##  2        64     8    19  1977       7 DM         M                  37     48
##  3        65     8    19  1977       4 DM         F                  34     29
##  4        66     8    19  1977       4 DM         F                  35     46
##  5        67     8    19  1977       7 DM         M                  35     36
##  6        68     8    19  1977       8 DO         F                  32     52
##  7        69     8    19  1977       2 PF         M                  15      8
##  8        70     8    19  1977       3 OX         F                  21     22
##  9        71     8    19  1977       7 DM         F                  36     35
## 10        74     8    19  1977       8 PF         M                  12      7
## # i 32,273 more rows
```

It is common to combine a null filter with other conditions using "and." For example, we might want all of the data on a species that contains weights.

```r
filter(surveys, species_id == "DS" & !is.na(weight))
```

```
## # A tibble: 2,344 x 9
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1       357    11    12  1977       9 DS         F                  50    117
##  2       362    11    12  1977       1 DS         F                  51    121
##  3       367    11    12  1977      20 DS         M                  51    115
##  4       377    11    12  1977       9 DS         F                  48    120
##  5       381    11    13  1977      17 DS         F                  48    118
##  6       383    11    13  1977      11 DS         F                  52    126
##  7       385    11    13  1977      17 DS         M                  50    132
##  8       389    11    13  1977      14 DS         F                  NA    113
##  9       392    11    13  1977      11 DS         F                  53    122
## 10       394    11    13  1977       4 DS         F                  48    107
## # i 2,334 more rows
```

Let's Practice

Work on the last of Question 1, Shrub Volume.