Stat 33A - Lecture 3

February 24, 2020

Announcements

Homework 2 will be posted later today.

Review

Implicit Coercion

Last time we saw that R can automatically convert between ("coerce") types in one direction:

```
logical -> integer -> numeric -> complex -> character
```

```
class(5.1)
```

```
## [1] "numeric"
```

```
x = "hello"
class(x)
```

```
## [1] "character"
```

```
TRUE + 5
```

```
## [1] 6
```

```
c(6, 8, 1)
```

```
## [1] 6 8 1
```

```
c("hi", "hello")
```

```
## [1] "hi"    "hello"
```

```
c(7, "hi")
```

```
## [1] "7"  "hi"
```

Lists

In a vector, every element has to have the same type.

A list is a container for elements with *different* types.

If you use elements that can't be coerced to a common type, `c()` creates a list instead:

```
class(sin)
```

```
## [1] "function"
```

```
class(c(5, sin))
```

```
## [1] "list"
```

You can also directly create a list with the `list()` function:

```
list(5, "hi", 6.1)
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] "hi"
##
## [[3]]
## [1] 6.1
```

```
x = list(5, 6)

class(x)
```

```
## [1] "list"
```

```
# Lists print with [[ to indicate element positions.
x
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 6
```

```
# List elements can have names. Named elements print with $ instead:
list(a = 1, 2)
```

```
## $a
## [1] 1
##
## [[2]]
## [1] 2
```

Vectorized operations don't work for lists:

```
c(1, 2) + c(3, 4) # ok
```

```
## [1] 4 6
```

```
list(1, 2) + list(3, 4) # not ok
```

```
## Error in list(1, 2) + list(3, 4): non-numeric argument to binary operator
```

Projects and Files

Setting Up A Data Analysis

1. Create a project directory.
2. Download the data to the project directory.
3. Read the data into R.

For this class, think of each assignment as a new project.

The File System Tree

In order to read a data set, you need to tell R where it is on your computer.

Your computer's file system is like a tree.

The root is at `/` (OS X, Linux) or `C:/` (Windows).

Each directory is a branch.

Paths

The "path" to a file is the sequence of directories it's in, separated by forward slashes `/`.

For example, the file `dinosaur.csv`, in the directory `data,` in the directory `storage`, in the root directory:

`/storage/data/dinosaur.csv`

Windows traditionally uses backslashes instead. You can still use forward slashes in R.

An "absolute path" is one that starts from the root directory.

Paths in R

Absolute paths can be infuriatingly long to type.

You can set a "working directory" in R and then use "relative paths" that start from the working directory.

Use `getwd()` to check the working directory:

```
getwd()
```

```
## [1] "/home/nick/university/teach/stat33ab/stat33a/lectures/02.24"
```

```
# Check the PDF version of the notes if you want to see the output on
# my computer.
```

The output on your computer will be different, because you have different files!

PITFALL: RStudio maintains the working directory for your console independently of the working directory for each Rmd notebook. So:

- Running `getwd()` from the notebook with `Ctrl + Enter` displays the NOTEBOOK'S working directory.
- Typing `getwd()` in the "Console" window and pressing `Enter` displays the CONSOLE's working directory.

In the console, you can use `setwd()` with a path to set the working directory.

In the notebook, if you use `setwd()` it only lasts for **that chunk** and is then reset:

```
setwd("/home/nick/university/teach/stat33ab/stat33b")

getwd()
```

```
## [1] "/home/nick/university/teach/stat33ab/stat33b"
```

So in subsequent chunks it looks like you didn't call `setwd()`:

```
getwd()
```

```
## [1] "/home/nick/university/teach/stat33ab/stat33a/lectures/02.24"
```

Why does RStudio do this? It is a bad practice to include `setwd()` in your notebooks, because people you share the notebook with, like your colleagues, instructor, or employer, might not have the same directories on their computer as the ones you have on your computer. The next section has more details about this.

By default, RStudio does the right thing and sets the notebook's working directory to the place where the notebook is saved. Then you can use relative paths (see below) to load and save files from the notebook.

If you really want to set the working directory in a notebook, it is possible to override RStudio. See https://yihui.org/knitr/options/ for details.

Use `list.files()` to list files in a directory:

```
list.files()
```

```
## [1] "data"         "notes.pdf"    "notes.Rmd"    "rsession.txt"
```

A "relative path" is one that starts from the working directory.

```
# List the files in the "data" directory:
list.files("data")
```

```
##  [1] "airline"           "baywheels"         "census"
##  [4] "college_scorecard" "craigslist"        "datasaurus"
##  [7] "dc_bikes"          "digits"            "dogs"
## [10] "duncan"            "gapminder"         "kickstarter"
## [13] "mystery"           "simpson"           "stocks.sqlite"
## [16] "suppliers.sqlite"  "ucd_catalog"       "us_shapes"
## [19] "volerup.tsv"
```

```
# The relative path "data" stands for the absolute path
#
# "/home/nick/university/teach/stat33ab/stat33a/sandbox/data"
#
# So it saves a lot of typing!
```

If you see `character(0)` as the output from `list.files()`, that means one of:

- The path you provided is incorrect.
- The path you provided leads to a file, not a directory.
- There are no files in the directory.

For example, if we make a deliberate typo:

```
list.files("dat") # no files, the path is incorrect
```

```
## character(0)
```

The path `..` is a shortcut for the directory above:

```
getwd()
```

```
## [1] "/home/nick/university/teach/stat33ab/stat33a/lectures/02.24"
```

```
list.files("..") # what's in the directory above
```

```
## [1] "01.27" "02.03" "02.24"
```

```
# You can use .. more than once:
list.files("../..") # what's in the directory two levels above
```

```
## [1] "docs"     "hw"       "labs"     "lectures" "private"  "quizzes"  "sandbox"
```

```
# You can also use .. with regular directory names:
list.files("../../stat33b") # two levels above, then back down
```

```
## character(0)
```

The path ~ means your personal directory:

```
# `~` is `/home/nick` on my computer.
#
```

```
# It will be different on your computer.

list.files("~") # lists files in /home/nick
```

```
##  [1] "archive"             "Documents"           "fa18_141a_evals.pdf"
##  [4] "garden"              "market"              "nltk_data"
##  [7] "TIMESHEET.csv"       "TODO.md"             "university"
## [10] "workshop"            "yard"                "Zotero"
```

You can convert a relative path to an absolute path with `normalizePath()`:

```
# Where is `~`?
normalizePath("~")
```

```
## [1] "/home/nick"
```

Reproducible Analyses

Plan ahead so that other people can run your code and reproduce your results.

Good habits:

- Putting your notebook(s) and data in the project directory.
- Using paths relative to the project directory.

Bad habits:

- Calling `setwd()` in R notebooks and scripts.
- Using absolute paths.

It's okay to use `setwd()` in the *R console* to set the working directory to your project directory.

Working with Tabular Data

Data Frames

In statistics, tabular data usually has: * Observations as rows * Features as columns

R's data structure for tabular data is the "data frame".

For the next few lectures, we'll use the Dogs Data Set, from:

`https://informationisbeautiful.net/visualizations/best-in-show-whats-the-top-data-dog/`

Also posted on the bCourse in RDS format.

You can read an RDS file with `readRDS`:

```
dogs = readRDS("data/dogs/dogs_full.rds")
```

Usually not a good idea to print an unfamiliar data set:

```
# In the Rmd version of the notes, I've set eval = FALSE here so that
# this chunk doesn't actually run or print anything.

dogs # don't do this with unfamiliar data sets
```

Printing a large data set will also slow down knitting for R notebooks!

Instead, inspect the data set with functions.

We already saw one function for inspecting data:

```r
class(dogs) # note it is a data.frame
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Use `head()` to print the first 6 rows (or elements):

```r
head(dogs, 4)
```

```
## # A tibble: 4 x 18
##   breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##   <chr> <fct>   <dbl>          <int>      <int>         <dbl>             <int>
## 1 Bord~ herd~    3.64             45         39         20143                 1
## 2 Bord~ terr~    3.61             80         61         22638                30
## 3 Brit~ spor~    3.54             30         30         22589                19
## 4 Cair~ terr~    3.53             59         48         21992                35
## # ... with 11 more variables: longevity <dbl>, ailments <int>, price <dbl>,
## #   food_cost <dbl>, grooming <fct>, kids <fct>, megarank_kids <int>,
## #   megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

Use `tail()` for the last 6:

```r
tail(dogs)
```

```
## # A tibble: 6 x 18
##   breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##   <chr> <fct>   <dbl>          <int>      <int>         <dbl>             <int>
## 1 Vizs~ spor~      NA             37         NA            NA                25
## 2 Weim~ spor~      NA             32         NA            NA                21
## 3 Wels~ terr~      NA             99         NA            NA                53
## 4 Wire~ terr~      NA            100         NA            NA                51
## 5 Wire~ spor~      NA             92         NA            NA                46
## 6 Xolo~ non-~      NA            155         NA            NA                NA
## # ... with 11 more variables: longevity <dbl>, ailments <int>, price <dbl>,
## #   food_cost <dbl>, grooming <fct>, kids <fct>, megarank_kids <int>,
## #   megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

Use `dim()` to print the dimensions:

```r
dim(dogs)
```

```
## [1] 172  18
```

Alternatively, use `ncol()` and `nrow()`:

```r
ncol(dogs)
```

```
## [1] 18
```

```r
nrow(dogs)
```

```
## [1] 172
```

Use `names()` to print the column (or element) names:

```r
names(dogs)
```

```
##  [1] "breed"             "group"             "datadog"
##  [4] "popularity_all"    "popularity"        "lifetime_cost"
##  [7] "intelligence_rank" "longevity"         "ailments"
## [10] "price"             "food_cost"         "grooming"
## [13] "kids"              "megarank_kids"     "megarank"
```

```
## [16] "size"              "weight"            "height"
```

Use `rownames()` to print the row names:

```
rownames(dogs)
```

```
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
##  [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
##  [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"
##  [37] "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"
##  [49] "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"
##  [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72"
##  [73] "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"
##  [85] "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"
##  [97] "97"  "98"  "99"  "100" "101" "102" "103" "104" "105" "106" "107" "108"
## [109] "109" "110" "111" "112" "113" "114" "115" "116" "117" "118" "119" "120"
## [121] "121" "122" "123" "124" "125" "126" "127" "128" "129" "130" "131" "132"
## [133] "133" "134" "135" "136" "137" "138" "139" "140" "141" "142" "143" "144"
## [145] "145" "146" "147" "148" "149" "150" "151" "152" "153" "154" "155" "156"
## [157] "157" "158" "159" "160" "161" "162" "163" "164" "165" "166" "167" "168"
## [169] "169" "170" "171" "172"
```

Use `str()` to print a structural summary:

```
str(dogs)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    172 obs. of  18 variables:
##  $ breed            : chr  "Border Collie" "Border Terrier" "Brittany" "Cairn Terrier" ...
##  $ group            : Factor w/ 7 levels "herding","hound",..: 1 5 4 5 4 4 4 6 1 1 ...
##  $ datadog          : num  3.64 3.61 3.54 3.53 3.34 3.33 3.3 3.26 3.25 3.22 ...
##  $ popularity_all   : int  45 80 30 59 130 63 27 38 60 20 ...
##  $ popularity       : int  39 61 30 48 81 51 27 33 49 20 ...
##  $ lifetime_cost    : num  20143 22638 22589 21992 20224 ...
##  $ intelligence_rank: int  1 30 19 35 31 18 20 8 10 6 ...
##  $ longevity        : num  12.5 14 12.9 13.8 12.5 ...
##  $ ailments         : int  2 0 0 2 1 0 2 5 1 5 ...
##  $ price            : num  623 833 618 435 750 800 465 740 530 465 ...
##  $ food_cost        : num  324 324 466 324 324 324 674 324 466 405 ...
##  $ grooming         : Factor w/ 3 levels "daily","weekly",..: 2 2 2 2 2 2 2 2 2 1 ...
##  $ kids             : Factor w/ 3 levels "high","medium",..: 3 1 2 1 1 1 1 2 3 1 ...
##  $ megarank_kids    : int  1 2 3 4 5 6 7 8 9 11 ...
##  $ megarank         : int  29 1 11 2 4 5 6 22 52 8 ...
##  $ size             : Factor w/ 3 levels "large","medium",..: 2 3 2 3 2 2 3 3 2 3 ...
##  $ weight           : num  NA 13.5 35 14 NA 30 25 NA NA 22 ...
##  $ height           : num  20 NA 19 10 18 16 14.5 9.5 18.5 14.5 ...
```

Use `summary()` to print a statistical summary:

```
summary(dogs)
```

```
##     breed                    group       datadog      popularity_all
##  Length:172         herding     :25   Min.   :0.990   Min.   :  1.00
##  Class :character   hound       :26   1st Qu.:2.185   1st Qu.: 43.75
##  Mode  :character   non-sporting:19   Median :2.710   Median : 87.50
##                     sporting    :28   Mean   :2.604   Mean   : 87.12
##                     terrier     :28   3rd Qu.:3.035   3rd Qu.:130.25
##                     toy         :19   Max.   :3.640   Max.   :173.00
##                     working     :27   NA's   :85
```

```
##    popularity   lifetime_cost   intelligence_rank   longevity
##  Min.   : 1.0   Min.   :12653   Min.   : 1.00     Min.   : 6.29
##  1st Qu.:22.5   1st Qu.:17816   1st Qu.:27.00     1st Qu.: 9.70
##  Median :44.0   Median :20087   Median :42.00     Median :11.29
##  Mean   :44.0   Mean   :19820   Mean   :40.92     Mean   :10.96
##  3rd Qu.:65.5   3rd Qu.:21798   3rd Qu.:54.25     3rd Qu.:12.37
##  Max.   :87.0   Max.   :26686   Max.   :80.00     Max.   :16.50
##  NA's   :85     NA's   :81      NA's   :40        NA's   :37
##     ailments         price          food_cost         grooming      kids
##  Min.   :0.000   Min.   : 283.0   Min.   : 270.0   daily  :23   high  :67
##  1st Qu.:0.000   1st Qu.: 587.2   1st Qu.: 324.0   weekly :88   medium:35
##  Median :1.000   Median : 795.0   Median : 466.0   monthly: 1   low   :10
##  Mean   :1.216   Mean   : 876.8   Mean   : 489.6   NA's   :60   NA's  :60
##  3rd Qu.:2.000   3rd Qu.:1042.2   3rd Qu.: 466.0
##  Max.   :9.000   Max.   :3460.0   Max.   :1349.0
##  NA's   :24      NA's   :26       NA's   :85
##  megarank_kids      megarank         size        weight           height
##  Min.   : 1.00   Min.   : 1.00   large :54   Min.   :  5.00   Min.   : 5.00
##  1st Qu.:22.50   1st Qu.:22.50   medium:60   1st Qu.: 17.50   1st Qu.:14.00
##  Median :44.00   Median :44.00   small :58   Median : 35.00   Median :19.00
##  Mean   :43.95   Mean   :43.94               Mean   : 44.97   Mean   :19.09
##  3rd Qu.:65.50   3rd Qu.:65.50               3rd Qu.: 62.50   3rd Qu.:24.12
##  Max.   :87.00   Max.   :87.00               Max.   :175.00   Max.   :32.00
##  NA's   :85      NA's   :85                  NA's   :86       NA's   :13
```

In a data frame:

- Every column must be the same length.
- Every row must be the same length.
- Each column must have homogeneous elements (like a vector).
- Each row may have heterogeneous elements (like a list).

_This is where the lecture ended.

I've included some additional notes to help you get started on the next homework assignment. These will also be covered in the lab and in the next lecture._

R Packages

The Comprehensive R Archive Network (CRAN) is a repository of user-contributed packages for R.

You can install packages from CRAN with `install.packages()`:

```
install.packages("dplyr")
```

For maintaining your packages, there are also functions:

- `remove.packages()` to remove a package
- `update.packages()` to update ALL packages
- `installed.packages()` to list installed packages

You can load an installed package into your R session with the `library()` function:

```
library(dplyr)
```

The process:

- Use `install.packages()` to install a package the **first time** you want to use the package, or if you want to update just one package to the latest version.
- Each time you start R, use `library()` to load the packages you want to use. This includes right after installing a package.

dplyr

The `dplyr` package provides functions for working with data frames.

We'll use `dplyr` for now, and learn about R's built-in tools later.

Cheat sheet:

```
https://github.com/rstudio/cheatsheets/raw/master/
    data-transformation.pdf
```

Use `slice()` to choose rows by position:

```
slice(dogs, 1) # row 1
```

```
## # A tibble: 1 x 18
##   breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##   <chr> <fct>   <dbl>          <int>      <int>         <dbl>            <int>
## 1 Bord~ herd~    3.64             45         39         20143                1
## # ... with 11 more variables: longevity <dbl>, ailments <int>, price <dbl>,
## #   food_cost <dbl>, grooming <fct>, kids <fct>, megarank_kids <int>,
## #   megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

```
slice(dogs, 2) # row 2
```

```
## # A tibble: 1 x 18
##   breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##   <chr> <fct>   <dbl>          <int>      <int>         <dbl>            <int>
## 1 Bord~ terr~    3.61             80         61         22638               30
## # ... with 11 more variables: longevity <dbl>, ailments <int>, price <dbl>,
## #   food_cost <dbl>, grooming <fct>, kids <fct>, megarank_kids <int>,
## #   megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

Use `filter()` to choose rows that satisfy a condition:

```
# Dogs with weight greater than 60:
filter(dogs, weight > 60)
```

```
## # A tibble: 24 x 18
##    breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##    <chr> <fct>   <dbl>          <int>      <int>         <dbl>            <int>
##  1 Germ~ spor~    3.03             15         15         25842               17
##  2 Labr~ spor~    2.97              1          1         21299                7
##  3 Iris~ spor~    2.84             70         56         20323               35
##  4 Ches~ spor~    2.78             46         40         16697               27
##  5 Gord~ spor~    2.73             94         69         19605               34
##  6 Clum~ spor~    2.44            133         82         18084               37
##  7 Gian~ work~    2.38             95         70         26686               28
##  8 Grey~ hound    2.29            140         85         15819               46
##  9 Newf~ work~    2.07             43         37         19351               34
## 10 Rhod~ hound    1.91             44         38         16530               52
## # ... with 14 more rows, and 11 more variables: longevity <dbl>,
## #   ailments <int>, price <dbl>, food_cost <dbl>, grooming <fct>, kids <fct>,
```

```
## #   megarank_kids <int>, megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

```
# Dogs with weight greater than mean weight:
filter(dogs, weight > mean(weight, na.rm = TRUE))
```

```
## # A tibble: 37 x 18
##    breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##    <chr> <fct>   <dbl>          <int>      <int>         <dbl>             <int>
##  1 Sibe~ work~    3.22             16         16         22049                45
##  2 Engl~ spor~    3.09             29         29         21946                13
##  3 Germ~ spor~    3.03             15         15         25842                17
##  4 Poin~ spor~    3.03            115         74         24445                43
##  5 Labr~ spor~    2.97              1          1         21299                 7
##  6 Iris~ spor~    2.84             70         56         20323                35
##  7 Gold~ spor~    2.8               4          4         21447                 4
##  8 Ches~ spor~    2.78             46         40         16697                27
##  9 Gord~ spor~    2.73             94         69         19605                34
## 10 Clum~ spor~    2.44            133         82         18084                37
## # ... with 27 more rows, and 11 more variables: longevity <dbl>,
## #   ailments <int>, price <dbl>, food_cost <dbl>, grooming <fct>, kids <fct>,
## #   megarank_kids <int>, megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

```
# Dogs with breed equal to Bulldog:
filter(dogs, breed == "Bulldog")
```

```
## # A tibble: 1 x 18
##   breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##   <chr> <fct>   <dbl>          <int>      <int>         <dbl>             <int>
## 1 Bull~ non-~    0.99              6          6         13479                78
## # ... with 11 more variables: longevity <dbl>, ailments <int>, price <dbl>,
## #   food_cost <dbl>, grooming <fct>, kids <fct>, megarank_kids <int>,
## #   megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

Use `select()` to choose columns by name or position:

```
# Get datadog column:
select(dogs, datadog)
```

```
## # A tibble: 172 x 1
##    datadog
##      <dbl>
##  1    3.64
##  2    3.61
##  3    3.54
##  4    3.53
##  5    3.34
##  6    3.33
##  7    3.3
##  8    3.26
##  9    3.25
## 10    3.22
## # ... with 162 more rows
```

Use `:` to indicate a range of rows or columns:

```
# Get first 3 rows:
slice(dogs, 1:3)
```

```
## # A tibble: 3 x 18
##   breed group datadog popularity_all popularity lifetime_cost intelligence_ra~
##   <chr> <fct>   <dbl>          <int>      <int>         <dbl>            <int>
## 1 Bord~ herd~    3.64             45         39         20143                1
## 2 Bord~ terr~    3.61             80         61         22638               30
## 3 Brit~ spor~    3.54             30         30         22589               19
## # ... with 11 more variables: longevity <dbl>, ailments <int>, price <dbl>,
## #   food_cost <dbl>, grooming <fct>, kids <fct>, megarank_kids <int>,
## #   megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

```
# Get columns 2 through 4:
select(dogs, 2:4)
```

```
## # A tibble: 172 x 3
##    group    datadog popularity_all
##    <fct>      <dbl>          <int>
##  1 herding     3.64             45
##  2 terrier     3.61             80
##  3 sporting    3.54             30
##  4 terrier     3.53             59
##  5 sporting    3.34            130
##  6 sporting    3.33             63
##  7 sporting    3.3              27
##  8 toy         3.26             38
##  9 herding     3.25             60
## 10 herding     3.22             20
## # ... with 162 more rows
```

```
# Get columns breed through popularity:
select(dogs, breed:popularity)
```

```
## # A tibble: 172 x 5
##    breed                 group    datadog popularity_all popularity
##    <chr>                 <fct>      <dbl>          <int>      <int>
##  1 Border Collie         herding     3.64             45         39
##  2 Border Terrier        terrier     3.61             80         61
##  3 Brittany              sporting    3.54             30         30
##  4 Cairn Terrier         terrier     3.53             59         48
##  5 Welsh Springer Spaniel sporting   3.34            130         81
##  6 English Cocker Spaniel sporting   3.33             63         51
##  7 Cocker Spaniel        sporting    3.3              27         27
##  8 Papillon              toy         3.26             38         33
##  9 Australian Cattle Dog herding     3.25             60         49
## 10 Shetland Sheepdog     herding     3.22             20         20
## # ... with 162 more rows
```

Use - to exclude rows or columns:

```
# Get all columns except breed:
select(dogs, -breed)
```

```
## # A tibble: 172 x 17
##    group datadog popularity_all popularity lifetime_cost intelligence_ra~
##    <fct>   <dbl>          <int>      <int>         <dbl>            <int>
##  1 herd~    3.64             45         39         20143                1
##  2 terr~    3.61             80         61         22638               30
##  3 spor~    3.54             30         30         22589               19
```

```
##  4 terr~    3.53            59         48         21992              35
##  5 spor~    3.34           130         81         20224              31
##  6 spor~    3.33            63         51         18993              18
##  7 spor~    3.3             27         27         24330              20
##  8 toy      3.26            38         33         21001               8
##  9 herd~    3.25            60         49         20395              10
## 10 herd~    3.22            20         20         21006               6
## # ... with 162 more rows, and 11 more variables: longevity <dbl>,
## #   ailments <int>, price <dbl>, food_cost <dbl>, grooming <fct>, kids <fct>,
## #   megarank_kids <int>, megarank <int>, size <fct>, weight <dbl>, height <dbl>
```

```
# Get all rows except 5 through 10:
select(dogs, -(5:10))
```

```
## # A tibble: 172 x 12
##    breed group datadog popularity_all food_cost grooming kids  megarank_kids
##    <chr> <fct>   <dbl>          <int>     <dbl> <fct>    <fct>         <int>
##  1 Bord~ herd~    3.64             45       324 weekly   low               1
##  2 Bord~ terr~    3.61             80       324 weekly   high              2
##  3 Brit~ spor~    3.54             30       466 weekly   medi~             3
##  4 Cair~ terr~    3.53             59       324 weekly   high              4
##  5 Wels~ spor~    3.34            130       324 weekly   high              5
##  6 Engl~ spor~    3.33             63       324 weekly   high              6
##  7 Cock~ spor~    3.3              27       674 weekly   high              7
##  8 Papi~ toy      3.26             38       324 weekly   medi~             8
##  9 Aust~ herd~    3.25             60       466 weekly   low               9
## 10 Shet~ herd~    3.22             20       405 daily    high             11
## # ... with 162 more rows, and 4 more variables: megarank <int>, size <fct>,
## #   weight <dbl>, height <dbl>
```

```
# Note that -5:10 is different from -(5:10) because of order of
# operations!
```

Other useful `dplyr` functions:

- `arrange()` changes the ordering of the rows.
- `mutate()` adds new columns by transforming existing columns.
- `summarise()` reduces multiple rows down to a single value.
- `group_by()` splits rows into groups when summarizing.