Data Wrangling and Visualization 101 in R

for BCS 206

Fall 2020

Contents

1 Preliminaries

1.1 Version control

RStudio makes version control, data backup, and data sharing easy (e.g., via Github.com). To use it, download and install git on your computer. Get a free github.com or bitbucket.com account. You only have to do this once.

Then, for each project, create a new project in RStudio and link it to the remote repository (select "Create project" > "Version control"). You will have to enter a URL for the remote repository, which you get, for example, at github.com under the repository's main page by clicking the "Clone or download button".

For step by step instructions, see:

- Setting up RStudio for version control
- RStudio help on version control

- Reverting a file to an earlier version

1.2 Reproducibility and literate coding

R and RStudio support reproducibility oriented literate coding via Sweave and Knitr: lab books, presentations, and papers can weave/knit together data, code, and text. The document you share contains the code needed to create its outputs (figures, tables, etc.). This is achieved by combining latex or R markdown with R code (or, for that matter, code from other programming languages). For an excellent video-based introduction, see this tutorial on R markdown. *This document is R markdown compiled with RStudio's knitr.

2 Data wrangling

The *R* libraries *dplyr* provide us with efficient ways to transform ('wrangle') our data tables. The library *magrittr* let's us concatenate these operations in transparent and easy to read code.

2.1 An example data set

We will illustrate the use of *dplyr* with the following data from an experiment with a 2AFC task in three within-subject letter sizes (A, B, C), for which we have extracted correctness (1 = correct; 0 = incorrect) and reaction times (RT):

```
summary(d)
```

```
##  size     condition       trial           subject        correct
##  1:4800   easy:7200   Min.   :  1.00   1      :1440   Min.   :0.0000
##  2:4800   hard:7200   1st Qu.: 60.75   2      :1440   1st Qu.:0.0000
##  4:4800               Median :120.50   3      :1440   Median :1.0000
##                       Mean   :120.50   4      :1440   Mean   :0.6227
##                       3rd Qu.:180.25   5      :1440   3rd Qu.:1.0000
##                       Max.   :240.00   6      :1440   Max.   :1.0000
##                                        (Other):5760
##        RT
##  Min.   : 27.72
##  1st Qu.:151.84
##  Median :255.62
##  Mean   :277.00
##  3rd Qu.:376.28
##  Max.   :891.47
##
```

```
glimpse(d)
```

```
## Rows: 14,400
## Columns: 6
## $ size      <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ condition <fct> easy, easy, easy, easy, easy, easy, easy, easy, easy, eas...
## $ trial     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
## $ subject   <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9,...
## $ correct   <int> 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, ...
## $ RT        <dbl> 403.021, 248.336, 314.841, 204.182, 274.689, 427.762, 312...
```

2.2 Dplyr's verbs

Dplyr has 'verbs' like filter, select, summarize, mutate, transmute, etc. to let use conduct operations on our data, and reshape the data frame into the format we need. We can use dplyr, for example, to calculate the proportion correct answers in our experiment by using *summarise*.

```
summarise(d, meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.623
```

Or just for letter size 1:

```
d.A = filter(d, size == "1")
summarise(d.A, meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.536
```

2.3 Maggritr's pipes

Here we will use only of the 'pipes' magrittr provides:

- x %>% f: takes x and hands it to the function f on the right, as f's first argument
- x %<>% f1 %>% f2 %>% etc.: takes x hands it to f1, takes the output of f1 and hands it to f2, etc. And since the first pipe was %<>% (rather than just %>%), the final result will be written back into x.

Figure 1: Magritt's pipe

Figure 2: Magrittr's pipe

2.4 Putting it together: Wrangling through pipes

Remember how we got the mean proportion correct for just letter size 1?

```
d.A = filter(d, size == "1")
summarise(d.A, meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.536
```

This is inelegant and hard to read. Pipes let us make this more transparent:

```
d %>%
  filter(size == "1") %>%
  summarise(meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.536
```

And this advantage becomes even clearer, the more operationgs we concatenate. For example, *group_by* is an elegant operator that tells the pipes to conduct all subsequent operations for each of the groups (and then put all the separate outcomes back together into a single data frame). So if we want the proportion correct for all groups:

```
d %>%
  group_by(size) %>%
  summarise(meanCorrect = mean(correct))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   size  meanCorrect
##   <fct>       <dbl>
## 1 1           0.536
## 2 2           0.598
## 3 4           0.734
```

2.5 Exercises

How can we:

- View the entire data set? (*View*)
- Calculate the by-subject averages for all three letter sizes? (*group_by*, *summarise*)
- Calculate the by-subject standard deviations around those averages? (*group_by*, *summarise*)
- Attach this information (the averages and SDs) to each row of the present data.frame? (*group_by*, *mutate*)
- Determine whether RTs were on average faster for correct, as compared to incorrect, trials?
- Add a column for log-transformed RTs to the data set?
- Remove the old column for raw RTs? (*select*)
- Sort the data by log-transformed reaction times? (*arrange*)

Say we further have an additional data frame with information about our subjects:

```
## # A tibble: 10 x 3
## # Rowwise:
##    subject gender   age
##    <fct>   <fct>  <dbl>
##  1 1       female    22
##  2 2       female    19
##  3 3       female    21
##  4 4       female    20
##  5 5       male      19
##  6 6       male      21
##  7 7       male      20
##  8 8       female    20
##  9 9       female    21
## 10 10      female    17
```

- How can we join the information from the two data sources together? (*left_join*)

```
## # A tibble: 10 x 3
## # Rowwise:
##    subject gender   age
##    <fct>   <fct>  <dbl>
##  1 1       female    22
##  2 2       female    19
##  3 3       female    21
##  4 4       female    20
##  5 5       male      19
##  6 6       male      21
##  7 7       male      20
##  8 8       female    20
##  9 9       female    21
## 10 10      female    17
```

```
## Joining, by = "subject"
```

```
## # A tibble: 14,400 x 8
##    size  condition trial subject correct    RT gender   age
##    <fct> <fct>     <int> <fct>     <int> <dbl> <fct>  <dbl>
##  1 1     easy          1 1             1  403. female    22
##  2 1     easy          1 2             0  248. female    19
##  3 1     easy          1 3             1  315. female    21
##  4 1     easy          1 4             1  204. female    20
##  5 1     easy          1 5             0  275. male      19
##  6 1     easy          1 6             1  428. male      21
##  7 1     easy          1 7             1  313. male      20
##  8 1     easy          1 8             0  473. female    20
##  9 1     easy          1 9             1  311. female    21
## 10 1     easy          1 10            1  375. female    17
## # ... with 14,390 more rows
```

3 Data visualization

The two main libraries in R we will be using for visualization are *ggplot2* and *plotly*. Ggplot2 provides a grammar of graphics approach to plotting. Plotly let's us interact with our data. In particular, *ggplotly()*

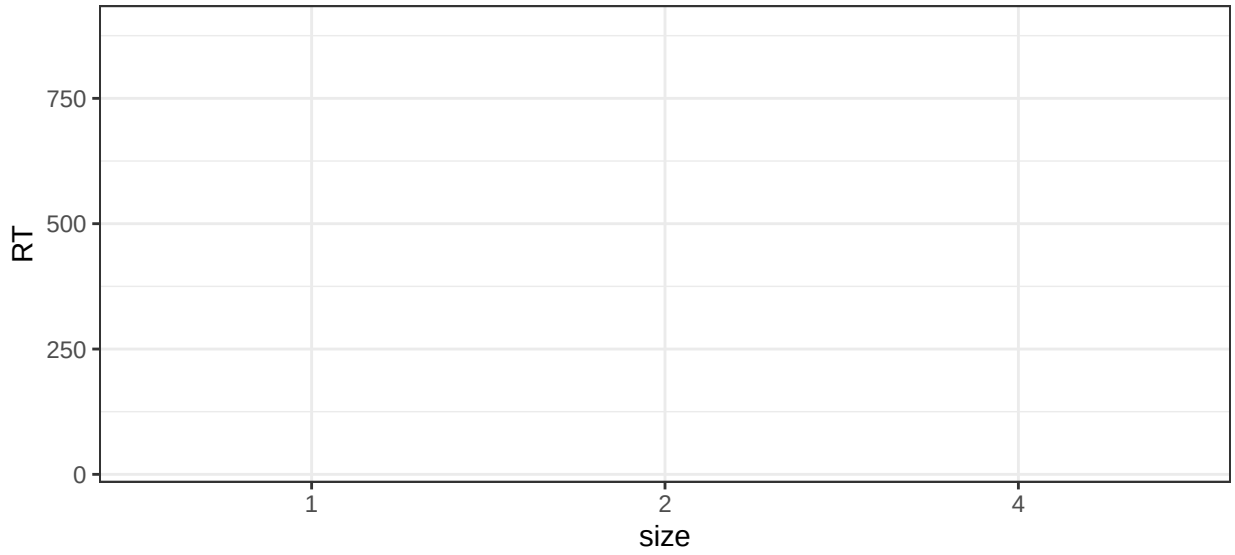
wrapped around a ggplot2 figure let's us interact with that figure.

3.1 Ggplot2's components (aesthetic mappings)

In order to plot in ggplot2, we need to understand the way it thinks about visualization. There are excellent online course that explain all of this, so I focus on the basics.

At the heart of a plot is a mapping between properties of your data (i.e., column in your data frame) and abstract properties of the plot (such as x- or y-coordinates, color, fill, transparency (alpha), linetype, shape, or label information). If we call the function ggplot() in order to create a figure, we specify two arguments: the name of the data frame we want to work with, and the mapping. The latter is done through a helpful function called *aes()*—for aesthetics:

```
ggplot(
  data = d,
  mapping =
    aes(
      x = size,
      y = RT))
```



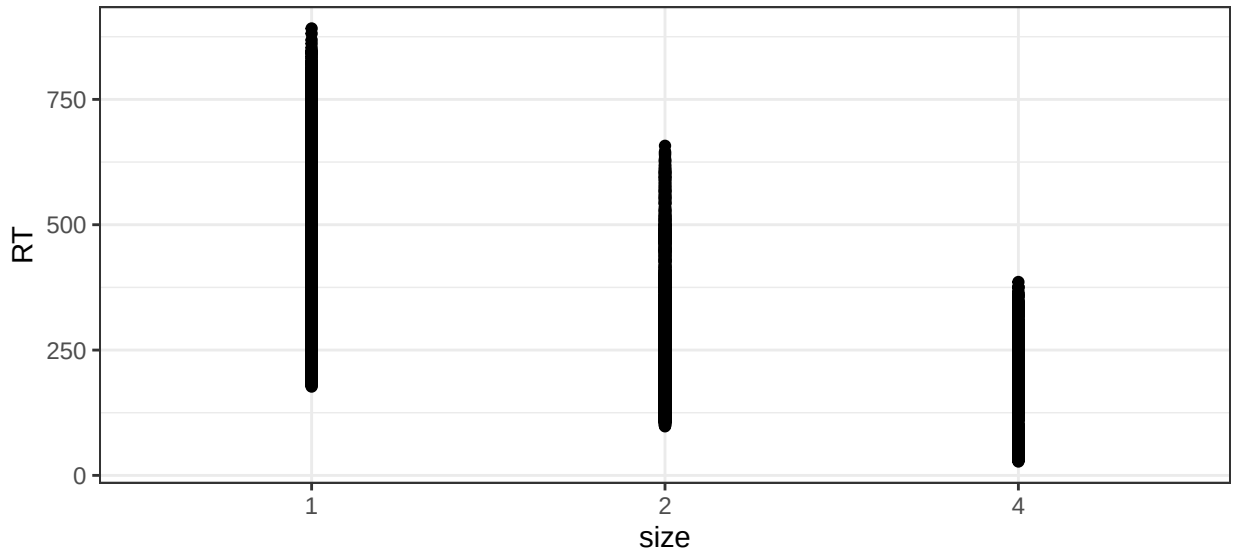

```
# or equivalently and shorter:
# ggplot(
#   d,
#   aes(
#       x = size,
#       y = RT))
```

3.1.1 Adding geometric components (geoms)

Notice that this by itself only returns an empty plot. That's the case because we have not yet specified how we want the abstract properties of the graph to be expressed visually. That's achieved by specifying *geoms* (for geometrics), such as points (*geom_point*), lines (*geom_line*), histograms (*geom_histogram*), lineranges (*geom_linerange*), and many similar functions (I take it you're getting th hang for the naming scheme …). You can find all of them on the ggplot2 cheatsheet.

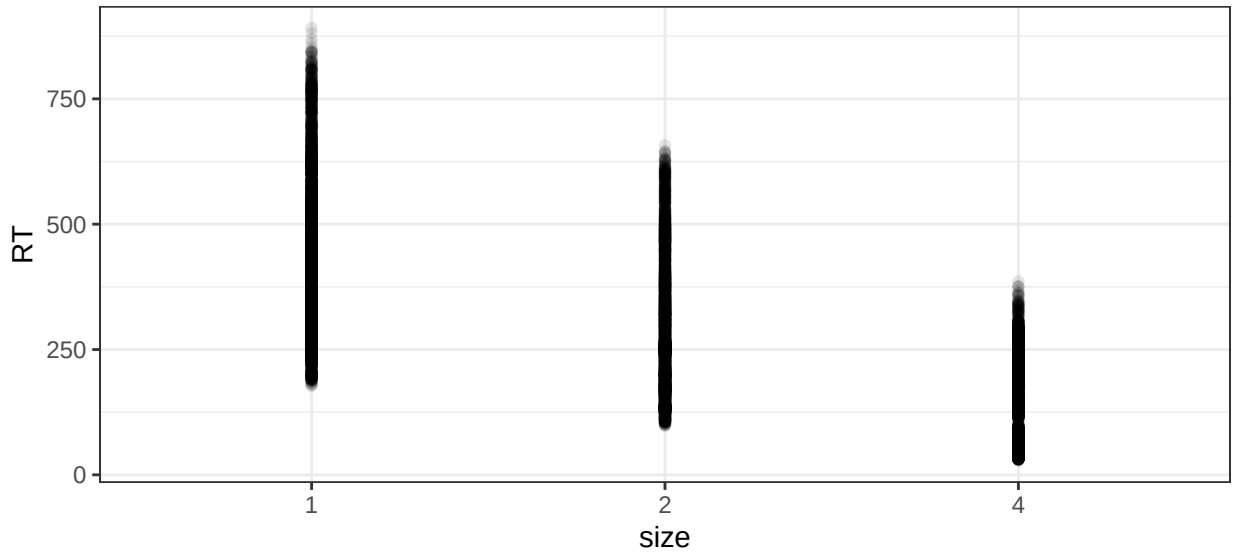
We add such components to a plot with "+".

```
p = ggplot(
  data = d,
  mapping =
    aes(
      x = size,
      y = RT)) +
  geom_point()

plot(p)
```



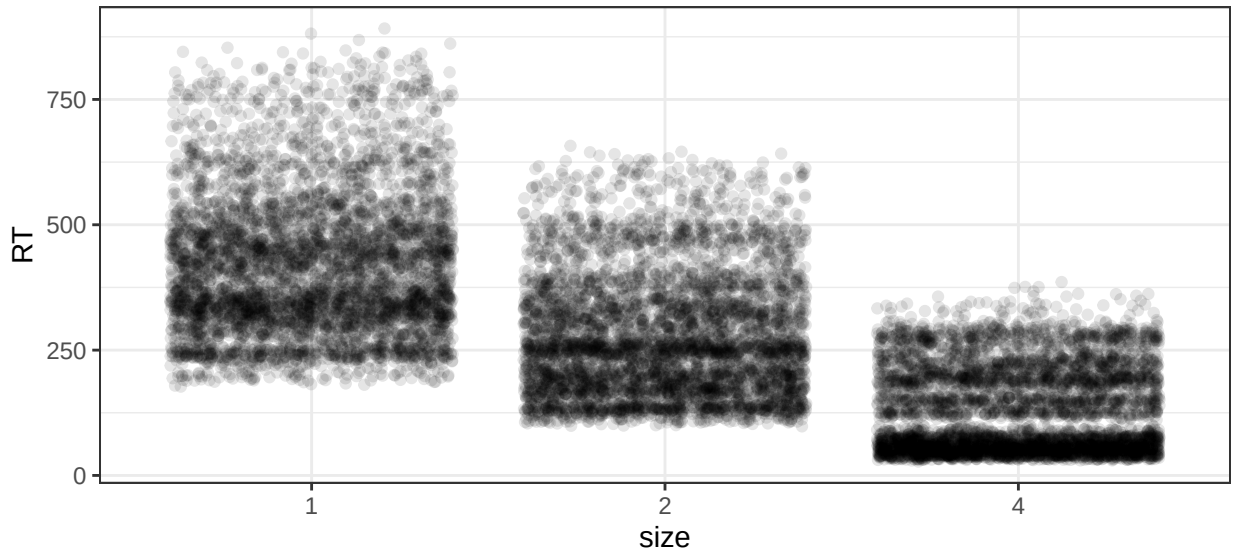
Notice that this plot is not particularly helpful! There is a lot of points and they are all overlaid, making it hard to get a clear idea of the data. One way we can improve this plot is to add transparency to the points. Another way is to jitter the points along the x-axis. Both approaches help convey information about the distribution of RTs at each letter size, and sometimes—especially, when we have a lot of data—we might want to combine both approaches. For example, here's the same plot with just transparency added. Since we have quite a bit of data, it still does not convey much information:

```
p = ggplot(
  data = d,
  mapping =
    aes(
      x = size,
      y = RT)) +
  geom_point(alpha = .1)

plot(p)
```



And here is the plot with some additional jitter added along the x-axis. Note that we intentionally avoid jitter along the y-axis:

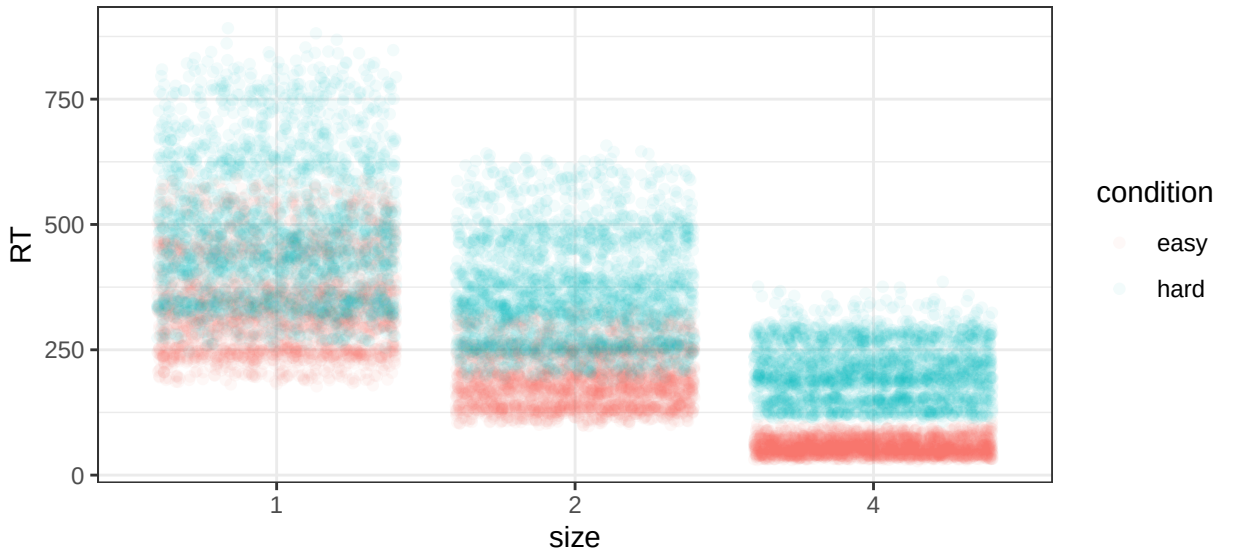
```
p = ggplot(
  data = d,
  mapping =
    aes(
      x = size,
      y = RT)) +
  geom_point(alpha = .1, position = position_jitter(height = 0))

plot(p)
```



That's much better already, giving us a much clearer idea of how the data is distributed! We are now in a position to think about how to include information about condition in this plot. There are a few ways to do this. For example, we could use different point shapes or colors for the easy and hard condition. Here, we do the latter:

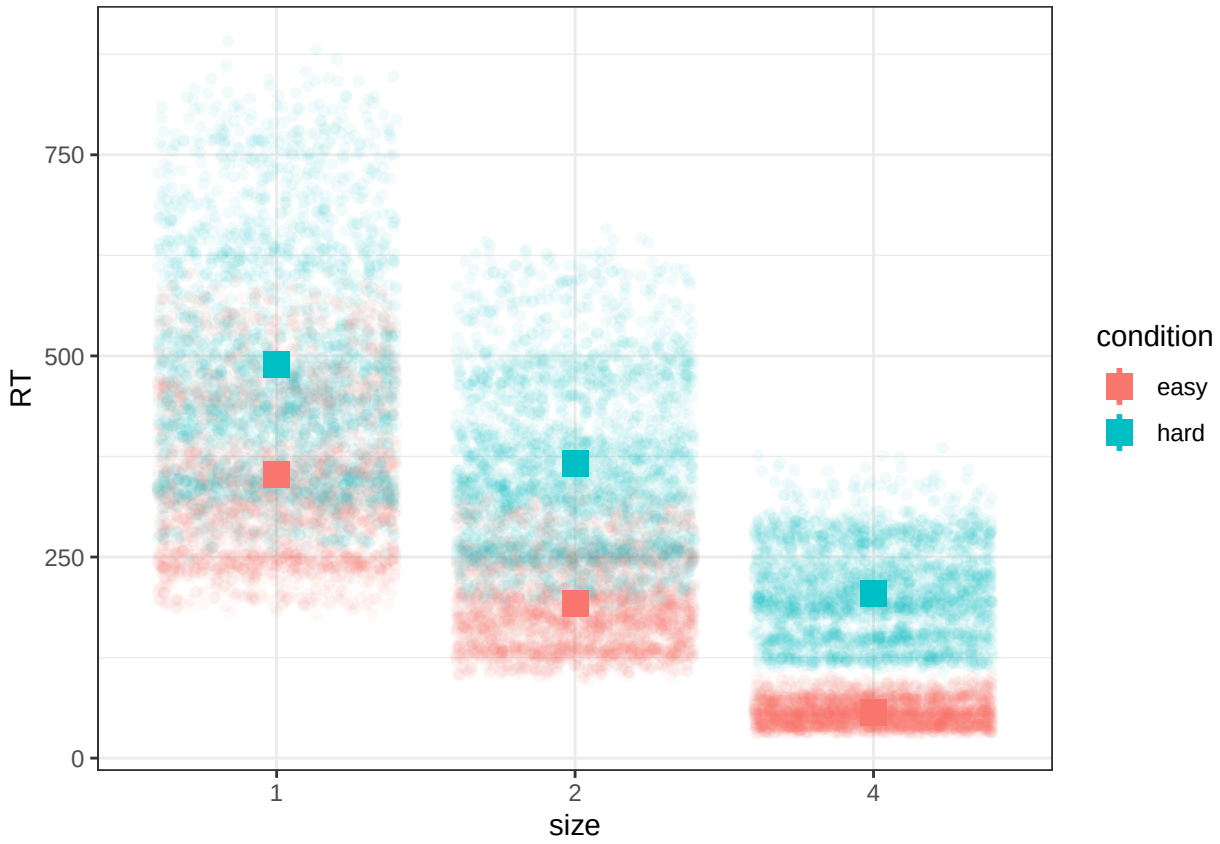
```
p = ggplot(
  data = d,
```

```
  mapping =
    aes(
      x = size,
      y = RT,
      color = condition)) +
  geom_point(alpha = .05, position = position_jitter(height = 0))

plot(p)
```

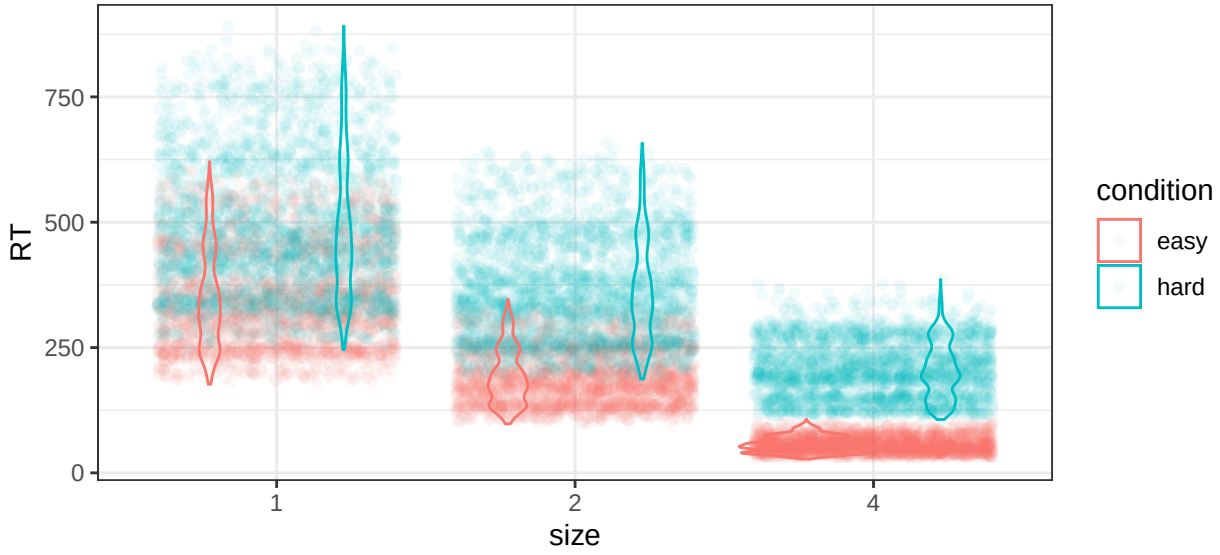


3.1.2 Visualizing data summaries

We could also summarize the data and plot a bootstrapped 95% confidence interval as a pointrange. In this case, we're specifying a statistical summary of the data and, as part of that, specify through which type of geom we would like it to be expressed:

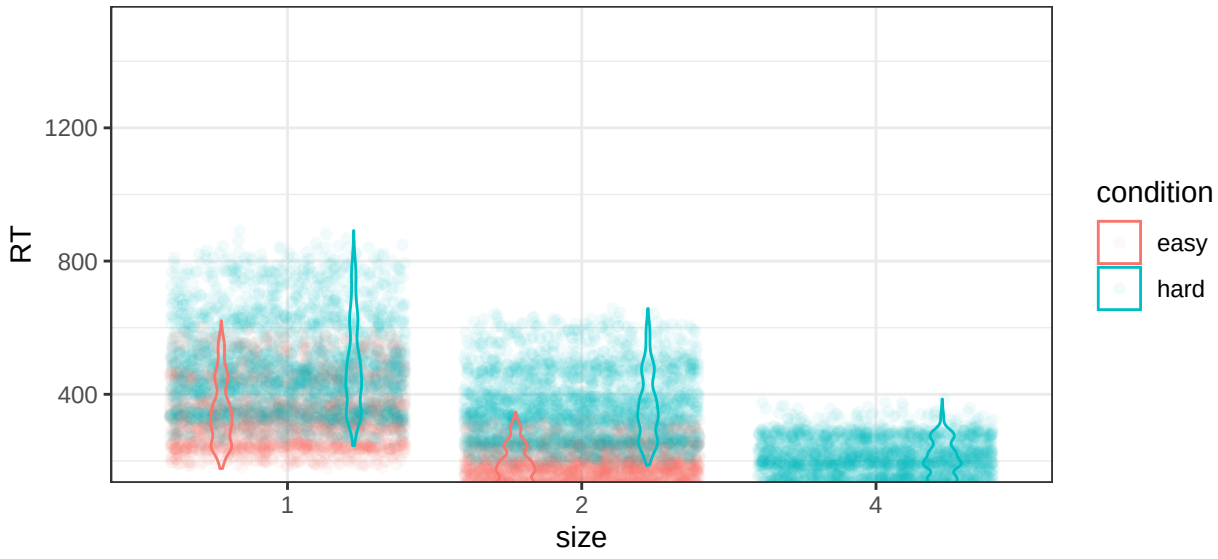


Alternatively, we could add a violin (essentially a mirrored density distribution):

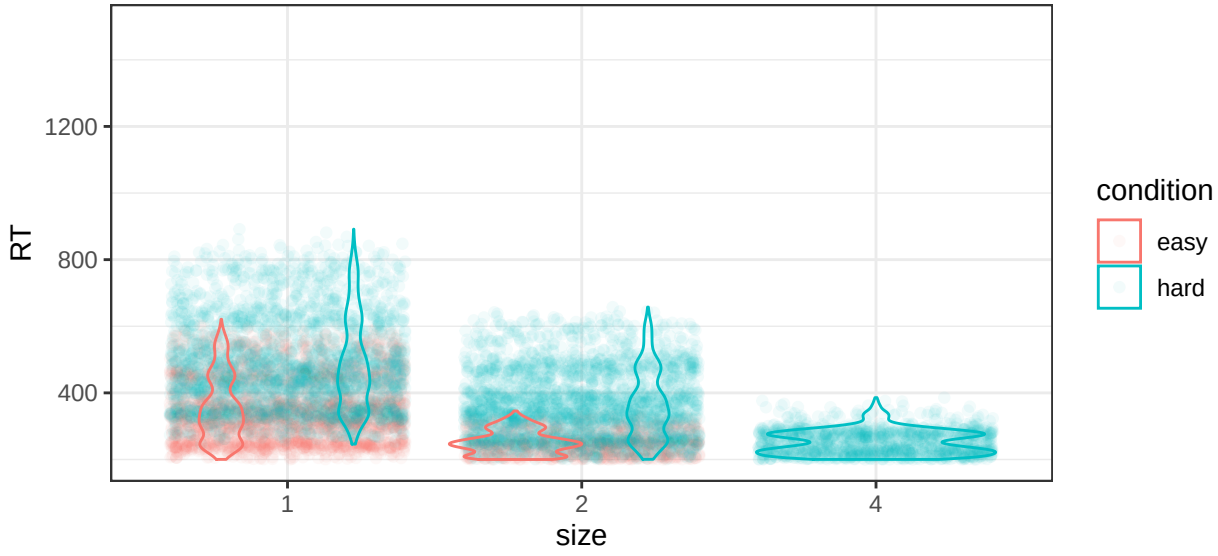


3.1.3 Scales and coordinate systems

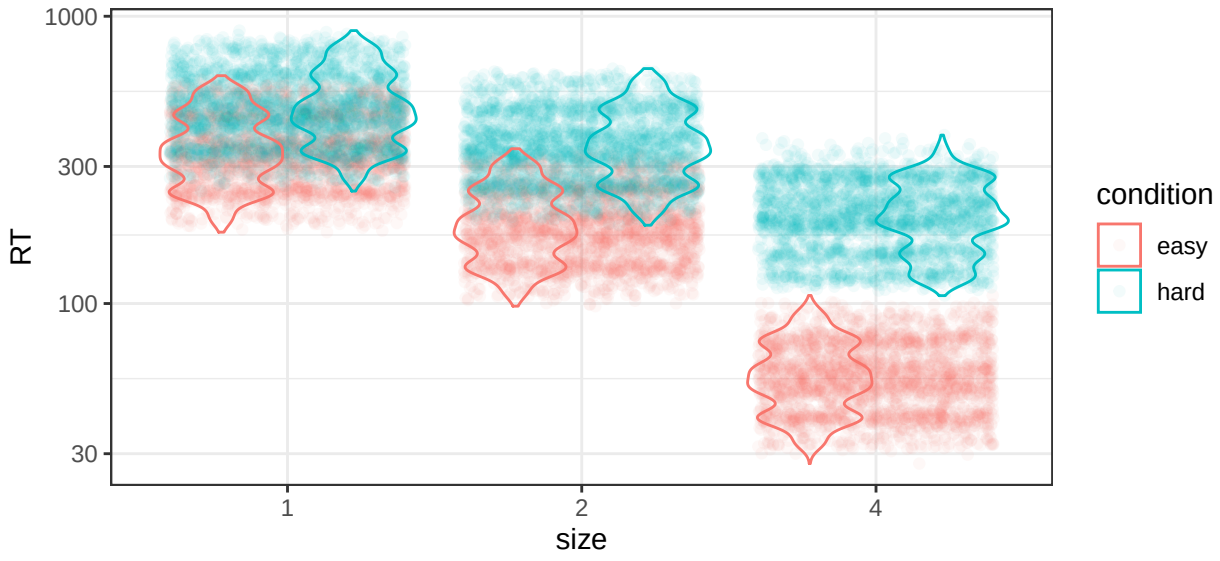
Sometimes we don't want to see all of the data, or we want to zoom into some ranges of our data. We can do so by explicitly specifying the x- and y-limits of our coordinate system:



Note that this zooms into parts of our data without excluding any data (e.g., from the calculation of the violins, which have the same shape as above). If we want to exclude data, transform data or in other ways change the way the aesthetical mappings are interpreted, this is achieved through *scales*. For example, the following *ex*cludes all RTs below 300 and above 2000. Note how that changes the violin plots (as it should: they estimate the ditribution of RTs):

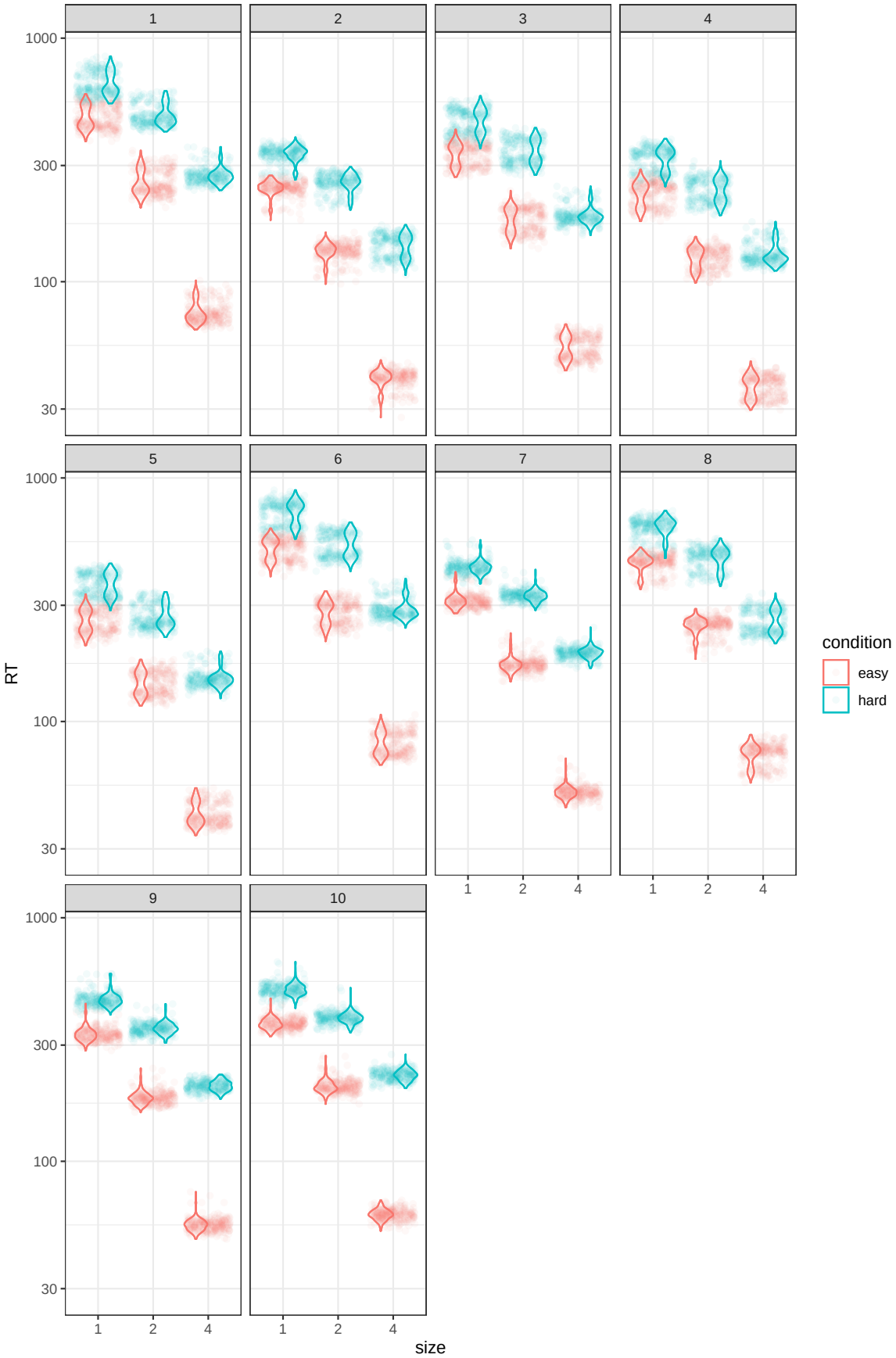


Since reaction times often have distributions that are more lognormal, rather than normal, let's update our original plot to use a log-transformed y-axis:

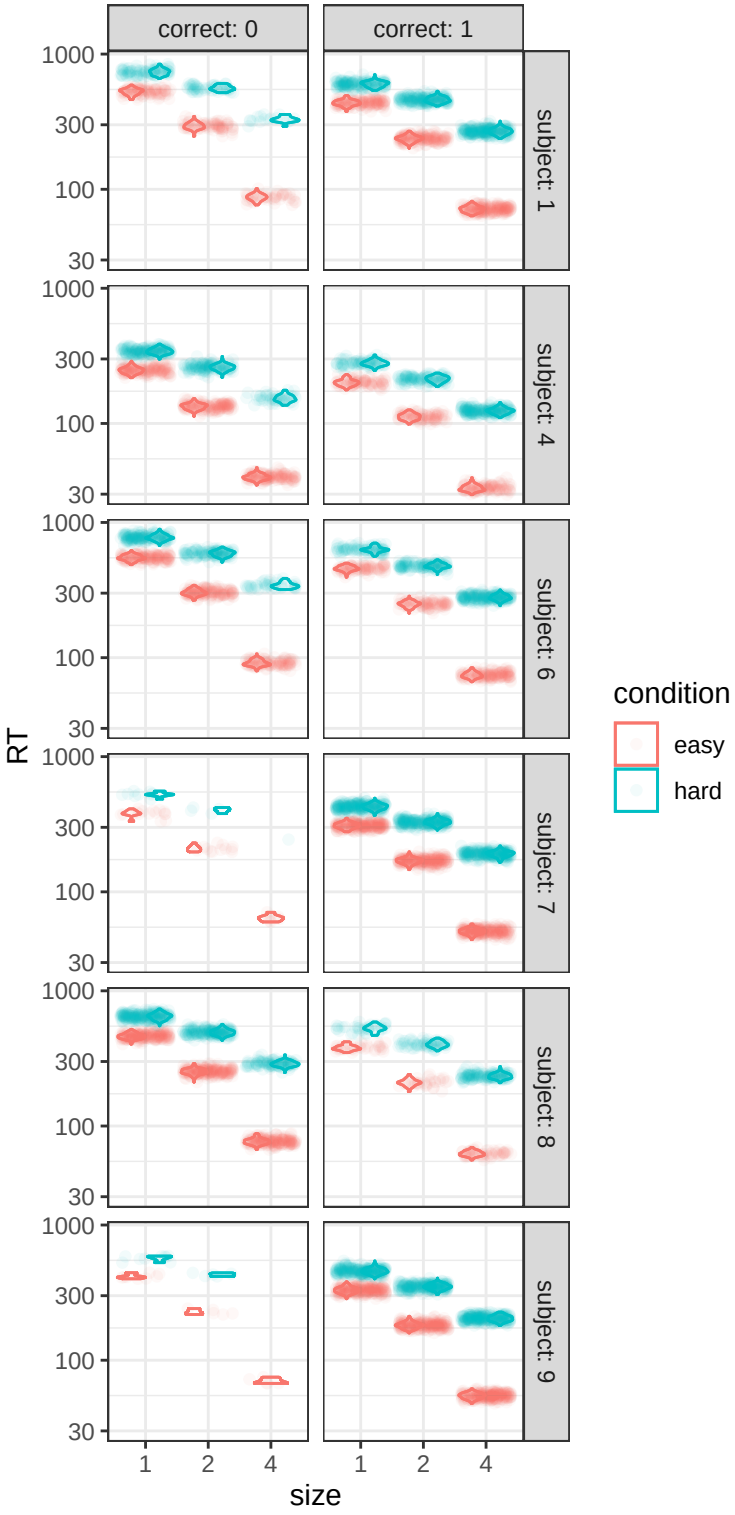


3.1.4 Facets

If we want to have separate panels conditional on another variable, we can do so through *facet*s. There are two major facet functions, *facet_wrap* (to have panels conditional on one variable) and *facet_grid* (conditional on two variables). For example, we can have separate panels for each subject:


1
2
3
4
5
6
7
8
9
10
1000
300
100
30
RT
1
2
4
size
condition
easy
hard

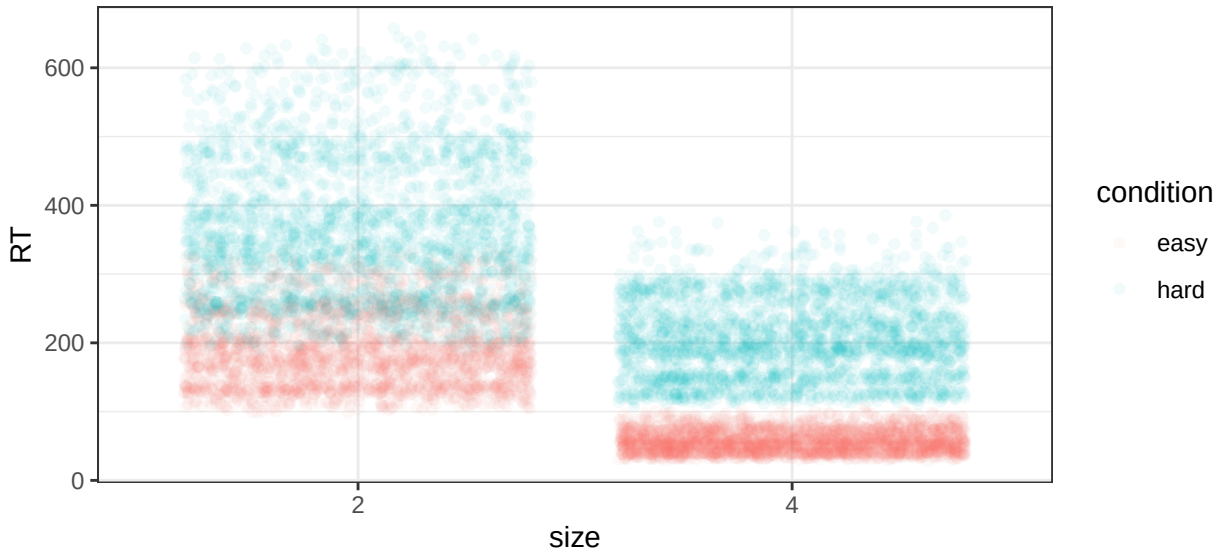
Or we could show by-subject RTs in two columns, separately for false and correct answers. Here, we do so after first sampling 6 random subjects (since the plot would otherwise be rather large).



3.2 Pipes (again)

Of course, we can use pipes to pipe the data frame into the plotting function, optionally after first piping the data through some additional *dplyr* operations (since the output of that entire pipe is again a data frame):

```
d %>%
  filter(size != "1") %>%
  ggplot(
    aes(
      x = size,
      y = RT,
      color = condition)) +
  geom_point(alpha = .05, position = position_jitter())
```



3.3 Exercises

- Plot a histogram of the RTs by letter size. Make on version where you plots the histograms in different facets, and another version where you have only one facet and use fill color to distinguish between letter sizes. (*geom_histogram*)
- Plot the average proportion of correct answers by letter size as a pointrange.
- Do the same, but first average by subject and letter size, and then plot the average (and confidence interval) of those by-subject averages of correct responses.
- Try to make a pie chart that shows the proportion correct for the three letter sizes. (*coord_polar*)

4 TBA

- making x-axis have right units
- axis labels
- panel grid, etc.

5 Session info

```
## - Session info ---------------------------------------------------------------
```

```
##  setting  value
##  version  R version 4.0.2 (2020-06-22)
##  os       macOS High Sierra 10.13.6
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       America/New_York
##  date     2020-10-25
##
## - Packages -------------------------------------------------------------------
##  package      * version    date       lib source
##  assertthat     0.2.1      2019-03-21 [1] CRAN (R 4.0.2)
##  backports      1.1.9      2020-08-24 [1] CRAN (R 4.0.2)
##  base64enc      0.1-3      2015-07-28 [1] CRAN (R 4.0.2)
##  blob           1.2.1      2020-01-20 [1] CRAN (R 4.0.2)
##  broom          0.7.0      2020-07-09 [1] CRAN (R 4.0.2)
##  callr          3.4.4      2020-09-07 [1] CRAN (R 4.0.2)
##  cellranger     1.1.0      2016-07-27 [1] CRAN (R 4.0.2)
##  checkmate      2.0.0      2020-02-06 [1] CRAN (R 4.0.2)
##  cli            2.0.2      2020-02-28 [1] CRAN (R 4.0.2)
##  cluster        2.1.0      2019-06-19 [1] CRAN (R 4.0.2)
##  colorspace     1.4-1      2019-03-18 [1] CRAN (R 4.0.2)
##  cowplot      * 1.1.0      2020-09-08 [1] CRAN (R 4.0.2)
##  crayon         1.3.4      2017-09-16 [1] CRAN (R 4.0.2)
##  data.table     1.13.0     2020-07-24 [1] CRAN (R 4.0.2)
##  DBI            1.1.0      2019-12-15 [1] CRAN (R 4.0.2)
##  dbplyr         1.4.4      2020-05-27 [1] CRAN (R 4.0.2)
##  desc           1.2.0      2018-05-01 [1] CRAN (R 4.0.2)
##  devtools       2.3.1      2020-07-21 [1] CRAN (R 4.0.2)
##  digest         0.6.25     2020-02-23 [1] CRAN (R 4.0.2)
##  dplyr        * 1.0.2      2020-08-18 [1] CRAN (R 4.0.2)
##  ellipsis       0.3.1      2020-05-15 [1] CRAN (R 4.0.2)
##  evaluate       0.14       2019-05-28 [1] CRAN (R 4.0.1)
##  fansi          0.4.1      2020-01-08 [1] CRAN (R 4.0.2)
##  farver         2.0.3      2020-01-16 [1] CRAN (R 4.0.2)
##  forcats      * 0.5.0      2020-03-01 [1] CRAN (R 4.0.2)
##  foreign        0.8-80     2020-05-24 [1] CRAN (R 4.0.2)
##  Formula        1.2-3      2018-05-03 [1] CRAN (R 4.0.2)
##  fs             1.5.0      2020-07-31 [1] CRAN (R 4.0.2)
##  generics       0.0.2      2018-11-29 [1] CRAN (R 4.0.2)
##  ggplot2      * 3.3.2      2020-06-19 [1] CRAN (R 4.0.2)
##  glue           1.4.2      2020-08-27 [1] CRAN (R 4.0.2)
##  gridExtra      2.3        2017-09-09 [1] CRAN (R 4.0.2)
##  gtable         0.3.0      2019-03-25 [1] CRAN (R 4.0.2)
##  haven          2.3.1      2020-06-01 [1] CRAN (R 4.0.2)
##  Hmisc          4.4-1      2020-08-10 [1] CRAN (R 4.0.2)
##  hms            0.5.3      2020-01-08 [1] CRAN (R 4.0.2)
##  htmlTable      2.0.1      2020-07-05 [1] CRAN (R 4.0.2)
##  htmltools      0.5.0      2020-06-16 [1] CRAN (R 4.0.2)
##  htmlwidgets    1.5.1      2019-10-08 [1] CRAN (R 4.0.2)
##  httr           1.4.2      2020-07-20 [1] CRAN (R 4.0.2)
##  jpeg           0.1-8.1    2019-10-24 [1] CRAN (R 4.0.2)
```

```
##  jsonlite       1.7.1      2020-09-07 [1] CRAN (R 4.0.2)
##  knitr          1.29.4     2020-08-23 [1] Github (yihui/knitr@bd64f2e)
##  labeling       0.3        2014-08-23 [1] CRAN (R 4.0.2)
##  lattice        0.20-41    2020-04-02 [1] CRAN (R 4.0.2)
##  latticeExtra   0.6-29     2019-12-19 [1] CRAN (R 4.0.2)
##  lazyeval       0.2.2      2019-03-15 [1] CRAN (R 4.0.2)
##  lifecycle      0.2.0      2020-03-06 [1] CRAN (R 4.0.2)
##  lubridate      1.7.9      2020-06-08 [1] CRAN (R 4.0.2)
##  magrittr     * 1.5        2014-11-22 [1] CRAN (R 4.0.2)
##  Matrix         1.2-18     2019-11-27 [1] CRAN (R 4.0.2)
##  memoise        1.1.0      2017-04-21 [1] CRAN (R 4.0.2)
##  modelr         0.1.8      2020-05-19 [1] CRAN (R 4.0.2)
##  munsell        0.5.0      2018-06-12 [1] CRAN (R 4.0.2)
##  nnet           7.3-14     2020-04-26 [1] CRAN (R 4.0.2)
##  pillar         1.4.6      2020-07-10 [1] CRAN (R 4.0.2)
##  pkgbuild       1.1.0.9000 2020-08-06 [1] Github (r-lib/pkgbuild@3a87bd9)
##  pkgconfig      2.0.3      2019-09-22 [1] CRAN (R 4.0.2)
##  pkgload        1.1.0      2020-05-29 [1] CRAN (R 4.0.2)
##  plotly       * 4.9.2.1    2020-04-04 [1] CRAN (R 4.0.2)
##  png            0.1-7      2013-12-03 [1] CRAN (R 4.0.2)
##  prettyunits    1.1.1      2020-01-24 [1] CRAN (R 4.0.2)
##  processx       3.4.4      2020-09-03 [1] CRAN (R 4.0.2)
##  ps             1.3.4      2020-08-11 [1] CRAN (R 4.0.2)
##  purrr        * 0.3.4      2020-04-17 [1] CRAN (R 4.0.2)
##  R.matlab     * 3.6.2      2018-09-27 [1] CRAN (R 4.0.2)
##  R.methodsS3    1.8.1      2020-08-26 [1] CRAN (R 4.0.2)
##  R.oo           1.24.0     2020-08-26 [1] CRAN (R 4.0.2)
##  R.utils        2.10.1     2020-08-26 [1] CRAN (R 4.0.2)
##  R6             2.4.1      2019-11-12 [1] CRAN (R 4.0.2)
##  RColorBrewer   1.1-2      2014-12-07 [1] CRAN (R 4.0.2)
##  Rcpp           1.0.5      2020-07-06 [1] CRAN (R 4.0.2)
##  readr        * 1.3.1      2018-12-21 [1] CRAN (R 4.0.2)
##  readxl       * 1.3.1      2019-03-13 [1] CRAN (R 4.0.2)
##  remotes        2.2.0      2020-07-21 [1] CRAN (R 4.0.2)
##  reprex         0.3.0      2019-05-16 [1] CRAN (R 4.0.2)
##  rlang          0.4.7      2020-07-09 [1] CRAN (R 4.0.2)
##  rmarkdown      2.3        2020-06-18 [1] CRAN (R 4.0.2)
##  rpart          4.1-15     2019-04-12 [1] CRAN (R 4.0.2)
##  rprojroot      1.3-2      2018-01-03 [1] CRAN (R 4.0.2)
##  rstudioapi     0.11       2020-02-07 [1] CRAN (R 4.0.2)
##  rvest          0.3.6      2020-07-25 [1] CRAN (R 4.0.2)
##  scales         1.1.1      2020-05-11 [1] CRAN (R 4.0.2)
##  sessioninfo    1.1.1      2018-11-05 [1] CRAN (R 4.0.2)
##  stringi        1.4.6      2020-02-17 [1] CRAN (R 4.0.2)
##  stringr      * 1.4.0      2019-02-10 [1] CRAN (R 4.0.2)
##  survival       3.2-3      2020-06-13 [1] CRAN (R 4.0.2)
##  testthat       2.3.2      2020-03-02 [1] CRAN (R 4.0.2)
##  tibble       * 3.0.3      2020-07-10 [1] CRAN (R 4.0.2)
##  tidyr        * 1.1.2      2020-08-27 [1] CRAN (R 4.0.2)
##  tidyselect     1.1.0      2020-05-11 [1] CRAN (R 4.0.2)
##  tidyverse    * 1.3.0      2019-11-21 [1] CRAN (R 4.0.2)
##  usethis        1.6.1      2020-04-29 [1] CRAN (R 4.0.2)
##  utf8           1.1.4      2018-05-24 [1] CRAN (R 4.0.2)
##  vctrs          0.3.4      2020-08-29 [1] CRAN (R 4.0.2)
```

```
##  viridisLite    0.3.0      2018-02-01 [1] CRAN (R 4.0.1)
##  withr          2.2.0      2020-04-20 [1] CRAN (R 4.0.2)
##  xfun           0.17       2020-09-09 [1] CRAN (R 4.0.2)
##  xml2           1.3.2      2020-04-23 [1] CRAN (R 4.0.2)
##  yaml           2.2.1      2020-02-01 [1] CRAN (R 4.0.2)
##
## [1] /Library/Frameworks/R.framework/Versions/4.0/Resources/library
```