Data Wrangling and Visualization 101

for BCS 206

Fall 2019

Contents

1 Goals for next two weeks

Over the next two weeks, we will introduce you to the basics of data visualization in R (Jaeger group; taught by Florian) and MatLab (Haefner and Mitchell groups; taught by Sabya). The topics we aim to cover are the following:

- Data wrangling: Turning the data into the form you need (*dplyr*)
- Data visualization:
  - General principles
  - How to plot in R (*ggplot*, *plotly*)
- Documenting your code:
  - R Markdown / MatLab notebook
- Introduction to confidence intervals

We only have a relatively short time, so we will focus on learning what tools are available and on *examples* of use (rather than an in-depth tutorial). There are great online tutorials and cheat sheets that contain further information. Specifically:

- **By Friday 10/30:**
  - Send your initial data (e.g., excel, matlab, or csv file) to both Sabya and Florian (you can use Slack). If you won't have your own data by then (which might well happen), that's ok. In that case, please ask your PI whether they have an old data set with data similar to the one that you would be analyzing.
- Monday 11/2:
  - **Prepare before class:** You will receive a data set ahead of class (described below). Load it into Matlab/R and familiarize yourself with its structure. Quick primers are available online for both R (https://rstudio.cloud/learn/primers/1) and Matlab (**SABYA**), as are tutorials on how to load data. Go through them *before* class.
  - In class, we will use that data to illustrate how we can visualize our data at various levels of summarization. <!–

- Whole class together: briefly introduce data
- DV = continuous or discretized continuous outcome
- IVs:
  - 1 predictor that is underlyingly continuous but only discrete steps have been sampled from during experiment
  - 1 predictor that is two-way categorical condition
- Split into R/Matlab groups:
- scatter plot:
- start with simple scatter plot, ignoring condition.
- add color
- deal with point overlap (jitter, transparency)
- add data summary (violin, boxplot)
- increase summarization (point range with confidence interval [do not explain yet how CI is derived])
- Rejoin at about 10am into whole class to discuss pros and cons of the different levels of summary. //–>
- Wednesday 11/4:
  - **Prepare before class:** Load your group's data. For at least one subject in your data, try to repeat the different plots we've introduced on Monday for your own data. You will be asked to present your efforts in class (to go through your script while sharing your screen). It's ok to get stuck, but please use Slack to ask for help prior to class.
  - In class, we will also go through problems/errors you might have encountered while trying to create visualizations of your data. <!–

- Split into R/Matlab groups:
- debug and go through problems with students. (have them present and then explain/help)
- in particular, there might be issues with getting the data into the right format or basic R/Matlab questions.
- challenges could also come up because there's data from multiple subjects.
- SABYA: we could also try to cover legends etc. here if time permits. Then we would use the final session (a week later) to talk about what to do with repeated measures data (why we're summarizing

the data down to the level of one data point per subject per condition when plotting CIs for repeated measures data, such as the "average subject" for your group.)

- Rejoin at about 10am into whole class to discuss what's working with the figures and what's still missing. //–>
- Monday 11/9:
  - Summarizing variability in your data—and thus the researcher's *uncertainty* about the central tendencies in the data
  - Introduction to standard deviations, standard errors, and confidence intervals
- Wednesday 11/11:
  - Preparing your visualizations for presentation (captions, axis titles, legends, and other annotations)
  - Saving your visualizations (format, dimensions)

2 Preliminaries

2.1 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

2.2 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.

3 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.

3.1 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 conditions (A, B, C), for which we have extracted correctness (1 = correct; 0 = incorrect) and reaction times (RT):

```
summary(d)
```

```
##  condition     trial          subject        correct             RT
##  A:2688    Min.   : 1.00   1      : 192   Min.   :0.0000   Min.   : 180.1
##  B:2688    1st Qu.:16.75   2      : 192   1st Qu.:0.0000   1st Qu.: 394.4
##  C:2688    Median :32.50   3      : 192   Median :1.0000   Median : 555.4
##            Mean   :32.50   4      : 192   Mean   :0.6308   Mean   : 748.9
##            3rd Qu.:48.25   5      : 192   3rd Qu.:1.0000   3rd Qu.: 986.5
##            Max.   :64.00   6      : 192   Max.   :1.0000   Max.   :3162.5
##                            (Other):6912
```

```
glimpse(d)
```

```
## Observations: 8,064
## Variables: 5
## $ condition <fct> A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, ...
## $ trial     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ subject   <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17...
## $ correct   <int> 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, ...
## $ RT        <dbl> 568.801, 357.301, 476.414, 352.467, 522.844, 328.874, 381...
```

3.2 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       0.631
```

Or just for condition A:

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

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.496
```

3.3 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

3.4 Putting it together: Wrangling through pipes

Remember how we got the mean proportion correct for just Condition A?

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

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.496
```

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

```
d %>%
  filter(condition == "A") %>%
  summarise(meanCorrect = mean(correct))
```

```
## # A tibble: 1 x 1
##   meanCorrect
##         <dbl>
## 1       0.496
```

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:

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

```
## # A tibble: 3 x 2
##   condition meanCorrect
##   <fct>           <dbl>
## 1 A               0.496
## 2 B               0.583
## 3 C               0.814
```

3.5 Exercises

How can we:

- View the entire data set? (*View*)
- Calculate the by-subject averages for all three conditions? (*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:

```
## Source: local data frame [42 x 3]
## Groups: <by row>
##
## # A tibble: 42 x 3
##    subject gender   age
##    <fct>   <chr>  <dbl>
##  1 1       female    21
##  2 2       female    21
##  3 3       male      21
##  4 4       male      18
##  5 5       female    19
##  6 6       female    19
##  7 7       male      20
##  8 8       female    20
##  9 9       male      19
## 10 10      female    20
## # ... with 32 more rows
```

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

```
## Source: local data frame [42 x 3]
## Groups: <by row>
##
## # A tibble: 42 x 3
##    subject gender   age
```

```
##    <fct>   <chr>  <dbl>
##  1 1       female    21
##  2 2       female    21
##  3 3       male      21
##  4 4       male      18
##  5 5       female    19
##  6 6       female    19
##  7 7       male      20
##  8 8       female    20
##  9 9       male      19
## 10 10      female    20
## # ... with 32 more rows
```

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

```
## # A tibble: 8,064 x 7
##    condition trial subject correct    RT gender   age
##    <fct>     <int> <fct>     <int> <dbl> <chr>  <dbl>
##  1 A             1 1             1  569. female    21
##  2 A             1 2             0  357. female    21
##  3 A             1 3             1  476. male      21
##  4 A             1 4             0  352. male      18
##  5 A             1 5             0  523. female    19
##  6 A             1 6             1  329. female    19
##  7 A             1 7             1  381. male      20
##  8 A             1 8             1  480. female    20
##  9 A             1 9             0  380. male      19
## 10 A             1 10            1  409. female    20
## # ... with 8,054 more rows
```

4 Data visualization

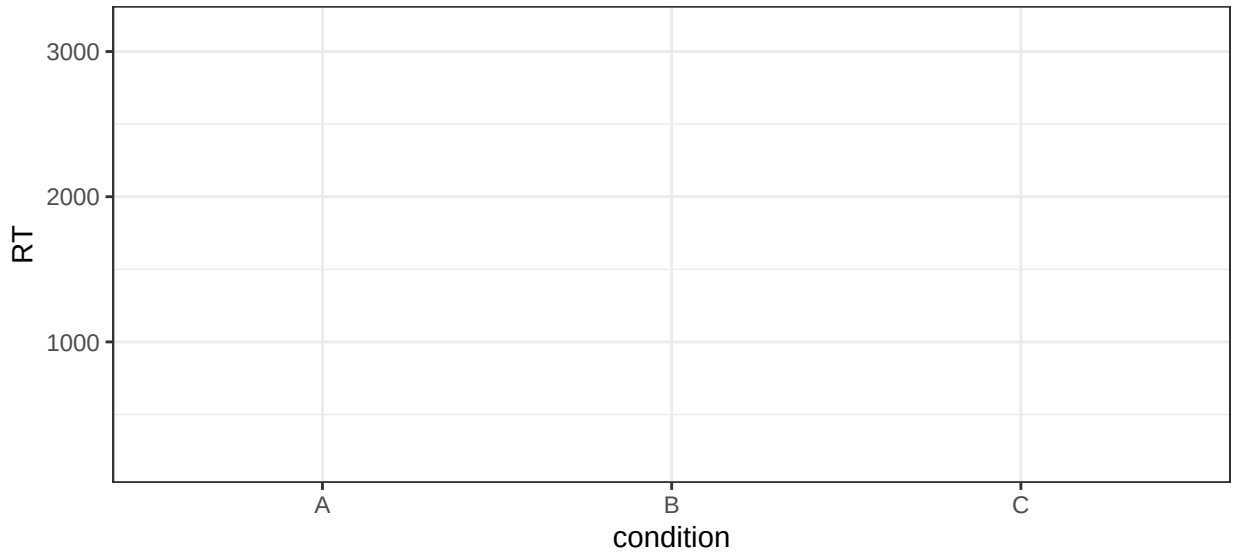
The two main libraries in R we will be using for visualization are *ggplot2* and *plotly*. Ggplot2 provides a grammar of graphics approach to plotting. Plotly let's us interact with our data. In particular, *ggplotly()* wrapped around a ggplot2 figure let's us interact with that figure.

4.1 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 = condition,
      y = RT))
```



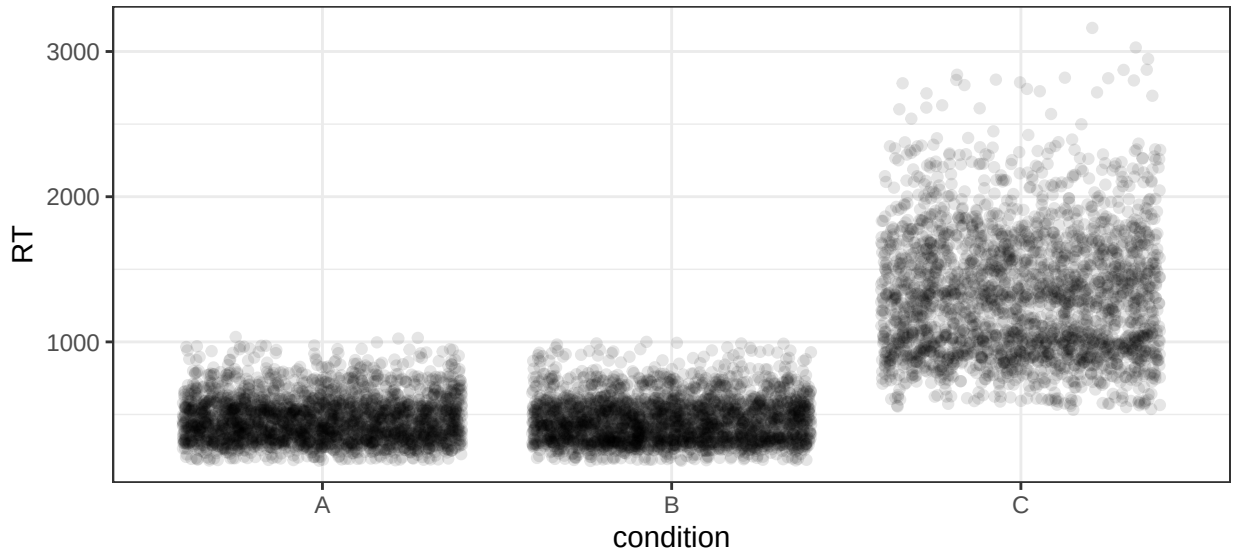

```
# or equivalently and shorter:
# ggplot(
#   d,
#   aes(
#       x = condition,
#       y = RT))
```

4.1.1 Adding geometric components (geoms)

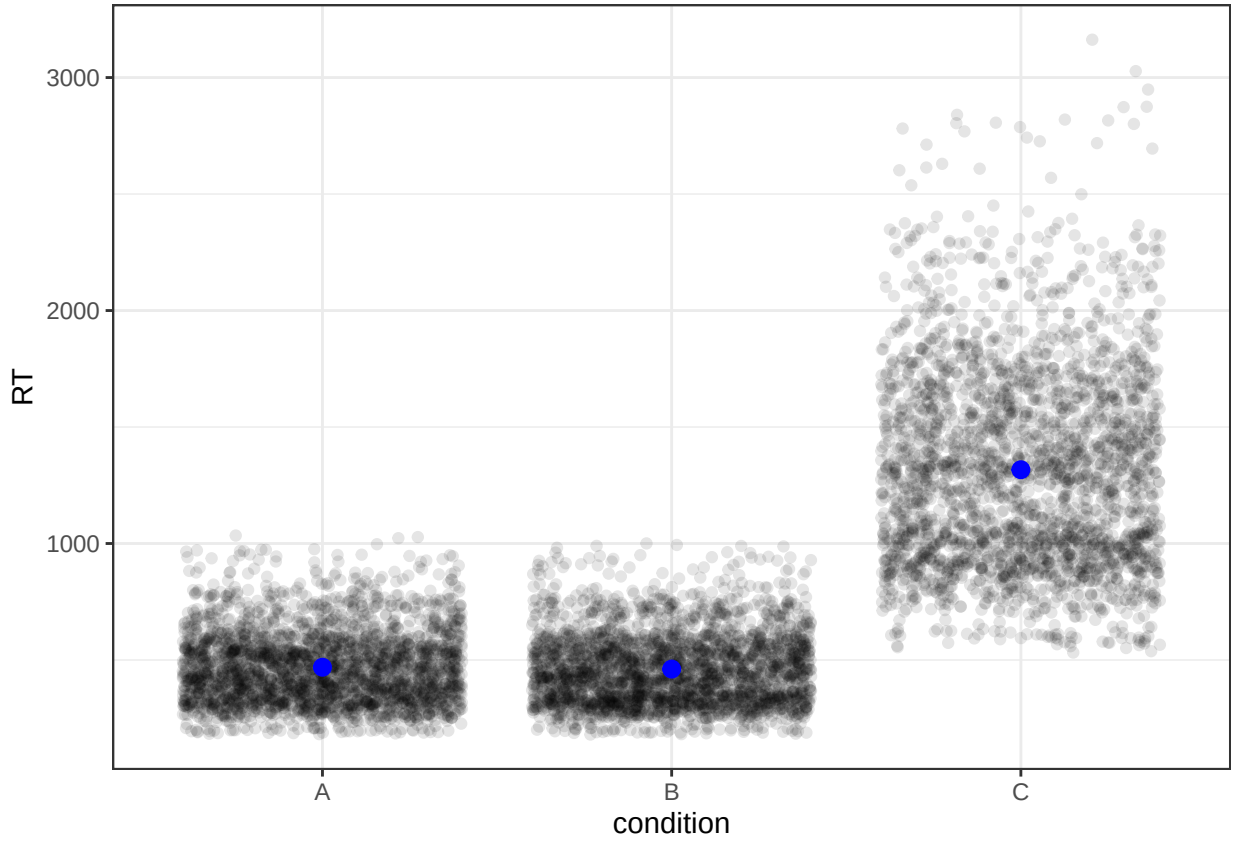
Notice that this by itself only returns an empty plot. That's the case because we have not yet specified how we want the abstract properties of the graph to be expressed visually. That's achieved by specifying *geom*s (for geometrics), such as points (*geom_point*), lines (*geom_line*), histograms (*geom_histogram*), 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 "+". We can also further explicitly specify any unused aesthetical properties of any geom. For example, to plot all the RTs for all three conditions with some transparency so that we see whether points cluster, plus some jittering along the x-axis (for the same reason):

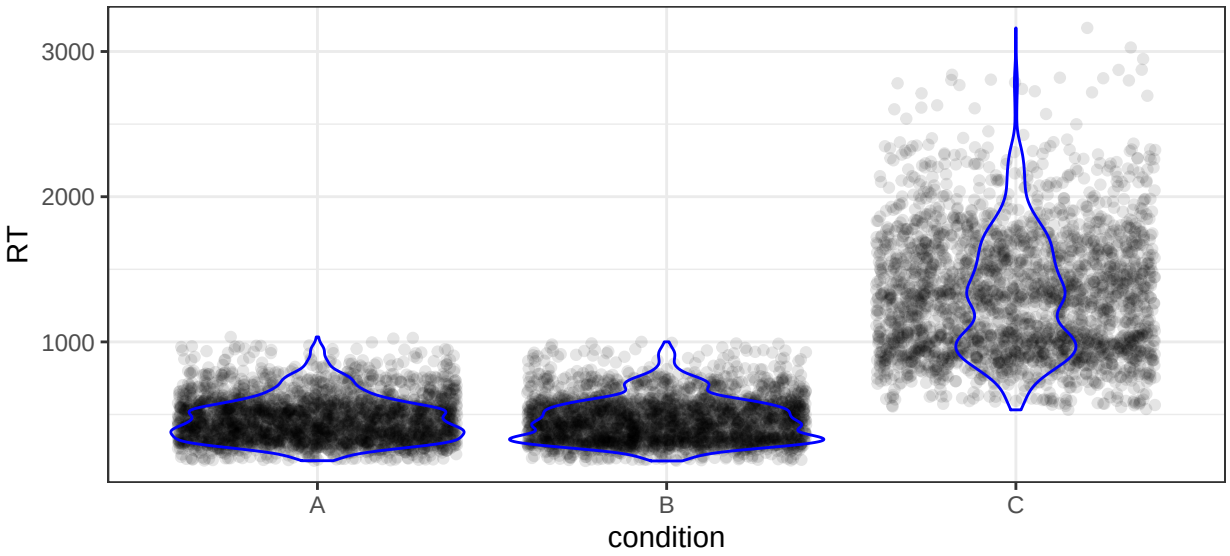
```
p = ggplot(
  data = d,
  mapping =
    aes(
      x = condition,
      y = RT)) +
  geom_point(alpha = .1, position = position_jitter())

plot(p)
```



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:

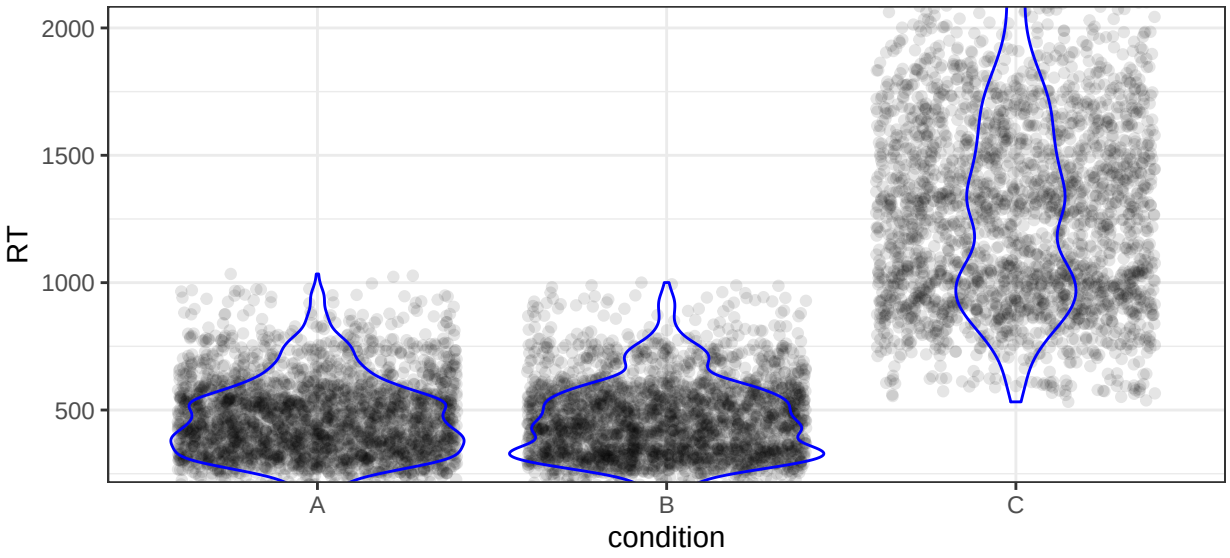


Alternatively, we could add a violin plot (essentially a mirrored density distribution, in this case displayed vertically on top of the points):

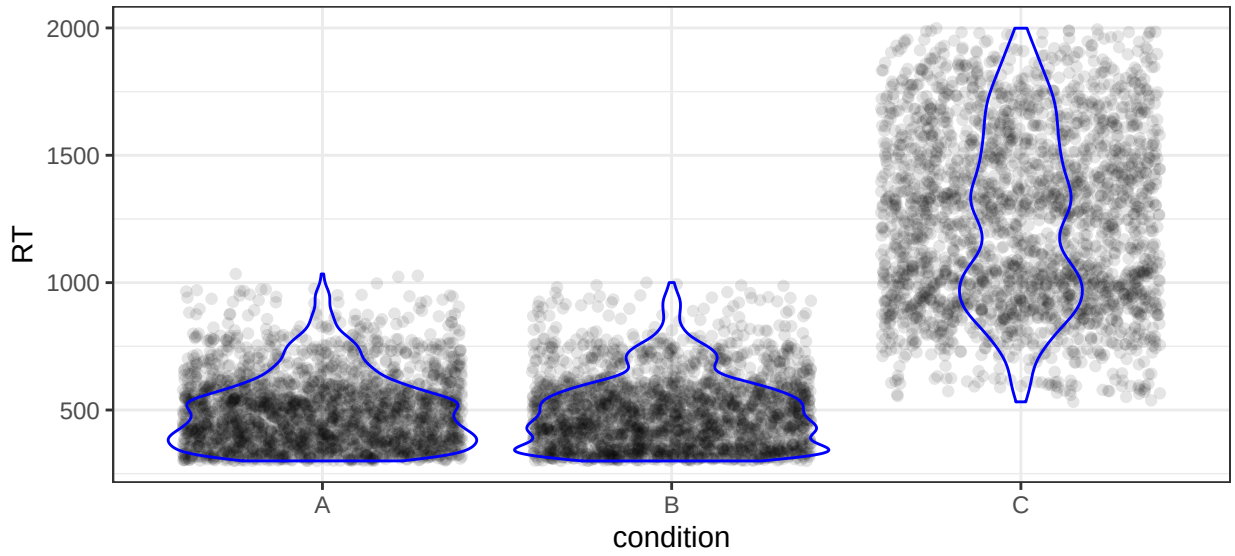


4.1.2 Scales and coordinate systems

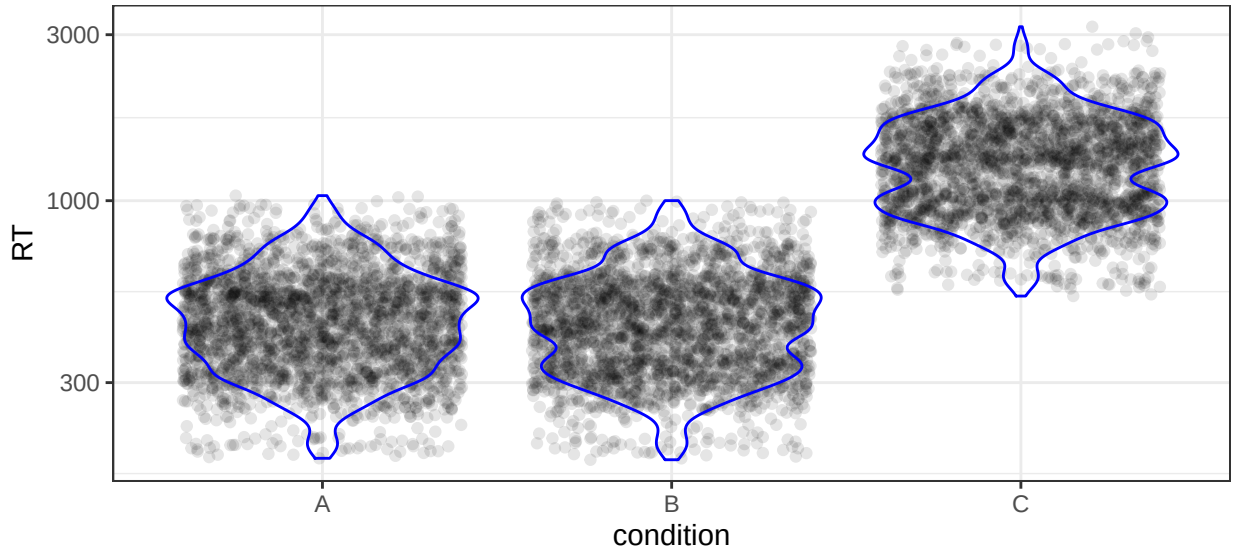
Sometimes we don't want to see all of the data, or we want to zoom into some ranges of our data. We can do so by explicitly specifying the x- and y-limits of our coordinate system:



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):

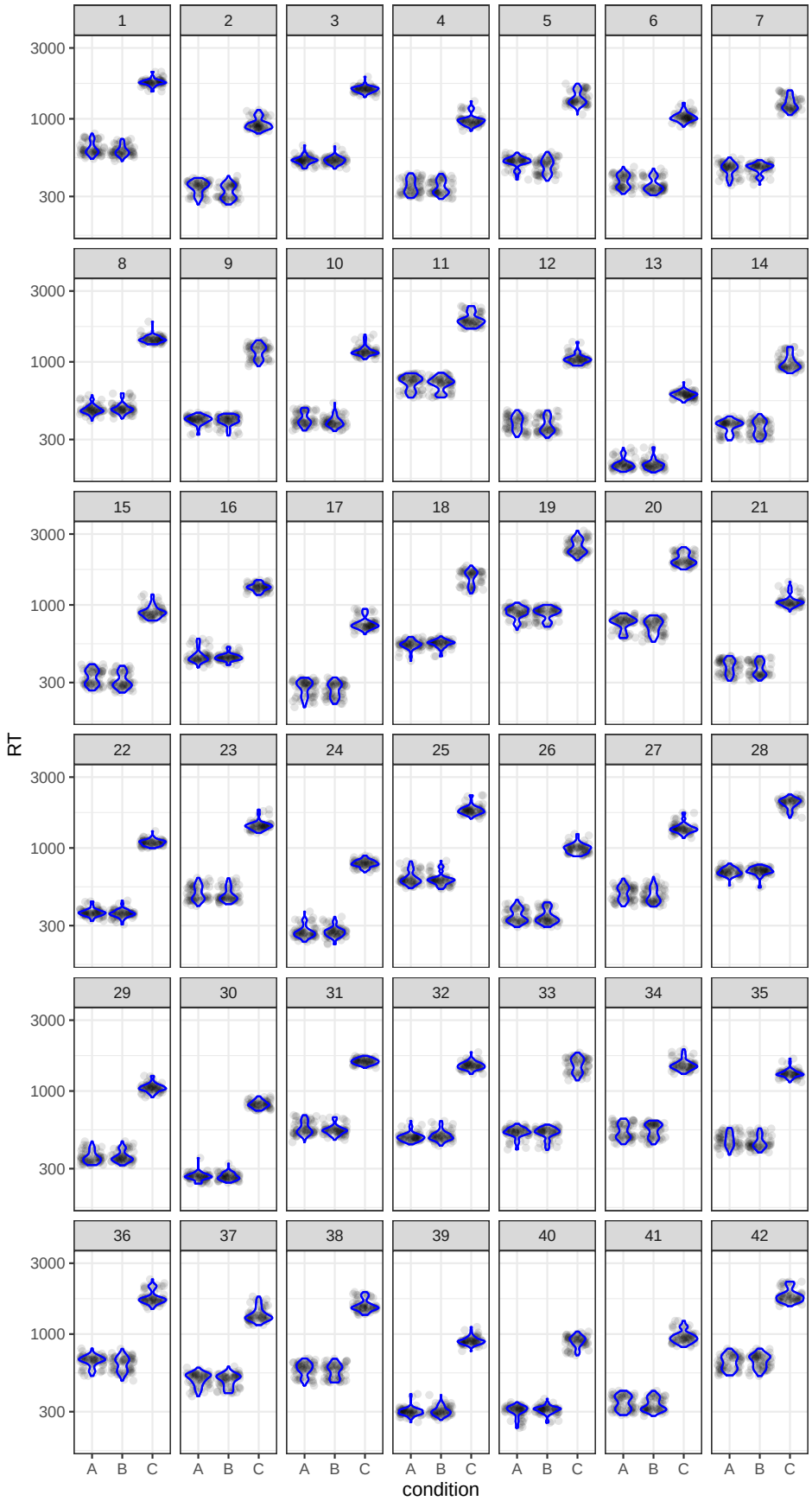


Since reaction times often have distributions that are more lognormal, rather than normal, let's update our original plot to use a log-transformed y-axis:

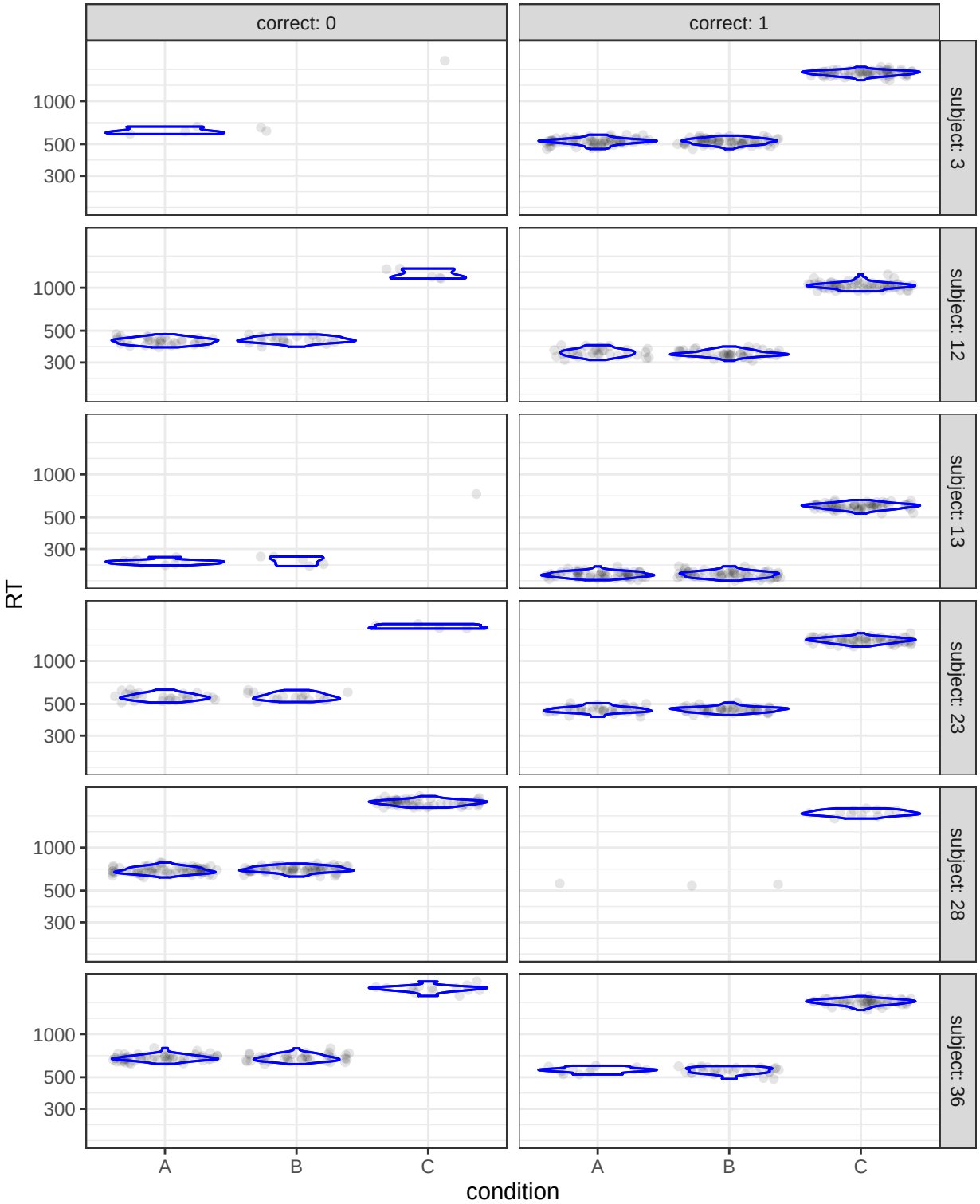


4.1.3 Facets

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
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
3000
1000
300
RT
A
B
C
condition

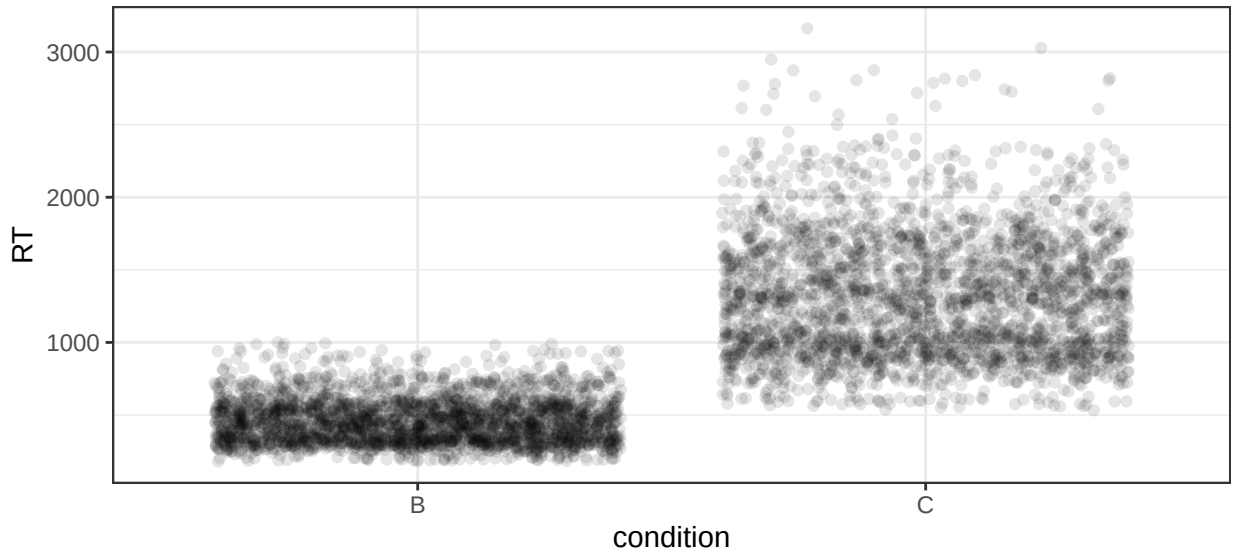
Or we could show by-subject RTs in two columns, separately for false and correct answers. Here, we do so after first sampling 6 random subjects (since the plot would otherwise be rather large).



4.2 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(condition != "A") %>%
  ggplot(
    aes(
      x = condition,
      y = RT)) +
  geom_point(alpha = .1, position = position_jitter())
```



4.3 Exercises

- Plot a histogram of the RTs by condition. 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 conditions. (*geom_histogram*)
- Plot the average proportion of correct answers by condition as a pointrange.
- Do the same, but first average by subject and condition, 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 conditions. (*coord_polar*)

5 Session info

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 3.6.2 (2019-12-12)
##  os       macOS High Sierra 10.13.6
##  system   x86_64, darwin15.6.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
```

```
##  tz       America/New_York
##  date     2020-03-24
##
## - Packages ------------------------------------------------------------------
##  package      * version date       lib source
##  acepack        1.4.1   2016-10-29 [1] CRAN (R 3.6.0)
##  assertthat     0.2.1   2019-03-21 [1] CRAN (R 3.6.0)
##  backports      1.1.5   2019-10-02 [1] CRAN (R 3.6.0)
##  base64enc      0.1-3   2015-07-28 [1] CRAN (R 3.6.0)
##  broom          0.5.5   2020-02-29 [1] CRAN (R 3.6.2)
##  callr          3.4.2   2020-02-12 [1] CRAN (R 3.6.0)
##  cellranger     1.1.0   2016-07-27 [1] CRAN (R 3.6.0)
##  checkmate      2.0.0   2020-02-06 [1] CRAN (R 3.6.0)
##  cli            2.0.2   2020-02-28 [1] CRAN (R 3.6.0)
##  cluster        2.1.0   2019-06-19 [1] CRAN (R 3.6.2)
##  codetools      0.2-16  2018-12-24 [1] CRAN (R 3.6.2)
##  colorspace     1.4-1   2019-03-18 [1] CRAN (R 3.6.0)
##  cowplot      * 1.0.0   2019-07-11 [1] CRAN (R 3.6.0)
##  crayon         1.3.4   2017-09-16 [1] CRAN (R 3.6.0)
##  data.table     1.12.8  2019-12-09 [1] CRAN (R 3.6.0)
##  DBI            1.1.0   2019-12-15 [1] CRAN (R 3.6.0)
##  dbplyr         1.4.2   2019-06-17 [1] CRAN (R 3.6.0)
##  desc           1.2.0   2018-05-01 [1] CRAN (R 3.6.0)
##  devtools       2.2.2   2020-02-17 [1] CRAN (R 3.6.0)
##  digest         0.6.25  2020-02-23 [1] CRAN (R 3.6.2)
##  dplyr        * 0.8.5   2020-03-07 [1] CRAN (R 3.6.0)
##  ellipsis       0.3.0   2019-09-20 [1] CRAN (R 3.6.0)
##  evaluate       0.14    2019-05-28 [1] CRAN (R 3.6.0)
##  fansi          0.4.1   2020-01-08 [1] CRAN (R 3.6.0)
##  farver         2.0.3   2020-01-16 [1] CRAN (R 3.6.0)
##  forcats      * 0.5.0   2020-03-01 [1] CRAN (R 3.6.2)
##  foreign        0.8-76  2020-03-03 [1] CRAN (R 3.6.0)
##  Formula        1.2-3   2018-05-03 [1] CRAN (R 3.6.0)
##  fs             1.3.2   2020-03-05 [1] CRAN (R 3.6.0)
##  generics       0.0.2   2018-11-29 [1] CRAN (R 3.6.0)
##  ggplot2      * 3.3.0   2020-03-05 [1] CRAN (R 3.6.0)
##  glue           1.3.2   2020-03-12 [1] CRAN (R 3.6.0)
##  gridExtra      2.3     2017-09-09 [1] CRAN (R 3.6.0)
##  gtable         0.3.0   2019-03-25 [1] CRAN (R 3.6.0)
##  haven          2.2.0   2019-11-08 [1] CRAN (R 3.6.0)
##  Hmisc          4.3-1   2020-02-07 [1] CRAN (R 3.6.0)
##  hms            0.5.3   2020-01-08 [1] CRAN (R 3.6.0)
##  htmlTable      1.13.3  2019-12-04 [1] CRAN (R 3.6.0)
##  htmltools      0.4.0   2019-10-04 [1] CRAN (R 3.6.0)
##  htmlwidgets    1.5.1   2019-10-08 [1] CRAN (R 3.6.0)
##  httr           1.4.1   2019-08-05 [1] CRAN (R 3.6.0)
##  jpeg           0.1-8.1 2019-10-24 [1] CRAN (R 3.6.0)
##  jsonlite       1.6.1   2020-02-02 [1] CRAN (R 3.6.0)
##  knitr          1.28    2020-02-06 [1] CRAN (R 3.6.0)
##  labeling       0.3     2014-08-23 [1] CRAN (R 3.6.0)
##  lattice        0.20-40 2020-02-19 [1] CRAN (R 3.6.0)
##  latticeExtra   0.6-29  2019-12-19 [1] CRAN (R 3.6.0)
##  lazyeval       0.2.2   2019-03-15 [1] CRAN (R 3.6.0)
##  lifecycle      0.2.0   2020-03-06 [1] CRAN (R 3.6.0)
```

```
##  lubridate      1.7.4   2018-04-11 [1] CRAN (R 3.6.0)
##  magrittr     * 1.5     2014-11-22 [1] CRAN (R 3.6.0)
##  Matrix         1.2-18  2019-11-27 [1] CRAN (R 3.6.2)
##  memoise        1.1.0   2017-04-21 [1] CRAN (R 3.6.0)
##  mgcv           1.8-31  2019-11-09 [1] CRAN (R 3.6.2)
##  modelr         0.1.6   2020-02-22 [1] CRAN (R 3.6.2)
##  munsell        0.5.0   2018-06-12 [1] CRAN (R 3.6.0)
##  nlme           3.1-145 2020-03-04 [1] CRAN (R 3.6.0)
##  nnet           7.3-13  2020-02-25 [1] CRAN (R 3.6.0)
##  openxlsx     * 4.1.4   2019-12-06 [1] CRAN (R 3.6.0)
##  pillar         1.4.3   2019-12-20 [1] CRAN (R 3.6.0)
##  pkgbuild       1.0.6   2019-10-09 [1] CRAN (R 3.6.0)
##  pkgconfig      2.0.3   2019-09-22 [1] CRAN (R 3.6.0)
##  pkgload        1.0.2   2018-10-29 [1] CRAN (R 3.6.0)
##  plotly       * 4.9.2   2020-02-12 [1] CRAN (R 3.6.0)
##  png            0.1-7   2013-12-03 [1] CRAN (R 3.6.0)
##  prettyunits    1.1.1   2020-01-24 [1] CRAN (R 3.6.0)
##  processx       3.4.2   2020-02-09 [1] CRAN (R 3.6.0)
##  ps             1.3.2   2020-02-13 [1] CRAN (R 3.6.0)
##  purrr        * 0.3.3   2019-10-18 [1] CRAN (R 3.6.0)
##  R.matlab     * 3.6.2   2018-09-27 [1] CRAN (R 3.6.0)
##  R.methodsS3    1.8.0   2020-02-14 [1] CRAN (R 3.6.0)
##  R.oo           1.23.0  2019-11-03 [1] CRAN (R 3.6.0)
##  R.utils        2.9.2   2019-12-08 [1] CRAN (R 3.6.0)
##  R6             2.4.1   2019-11-12 [1] CRAN (R 3.6.0)
##  RColorBrewer   1.1-2   2014-12-07 [1] CRAN (R 3.6.0)
##  Rcpp           1.0.4   2020-03-17 [1] CRAN (R 3.6.0)
##  readr        * 1.3.1   2018-12-21 [1] CRAN (R 3.6.0)
##  readxl       * 1.3.1   2019-03-13 [1] CRAN (R 3.6.0)
##  rematch        1.0.1   2016-04-21 [1] CRAN (R 3.6.0)
##  remotes        2.1.1   2020-02-15 [1] CRAN (R 3.6.0)
##  reprex         0.3.0   2019-05-16 [1] CRAN (R 3.6.0)
##  rlang          0.4.5   2020-03-01 [1] CRAN (R 3.6.0)
##  rmarkdown      2.1     2020-01-20 [1] CRAN (R 3.6.0)
##  rpart          4.1-15  2019-04-12 [1] CRAN (R 3.6.2)
##  rprojroot      1.3-2   2018-01-03 [1] CRAN (R 3.6.0)
##  rstudioapi     0.11    2020-02-07 [1] CRAN (R 3.6.0)
##  rvest          0.3.5   2019-11-08 [1] CRAN (R 3.6.0)
##  scales         1.1.0   2019-11-18 [1] CRAN (R 3.6.0)
##  sessioninfo    1.1.1   2018-11-05 [1] CRAN (R 3.6.0)
##  stringi        1.4.6   2020-02-17 [1] CRAN (R 3.6.0)
##  stringr      * 1.4.0   2019-02-10 [1] CRAN (R 3.6.0)
##  survival       3.1-11  2020-03-07 [1] CRAN (R 3.6.0)
##  testthat       2.3.2   2020-03-02 [1] CRAN (R 3.6.0)
##  tibble       * 2.1.3   2019-06-06 [1] CRAN (R 3.6.0)
##  tidyr        * 1.0.2   2020-01-24 [1] CRAN (R 3.6.0)
##  tidyselect     1.0.0   2020-01-27 [1] CRAN (R 3.6.0)
##  tidyverse    * 1.3.0   2019-11-21 [1] CRAN (R 3.6.0)
##  usethis        1.5.1   2019-07-04 [1] CRAN (R 3.6.0)
##  utf8           1.1.4   2018-05-24 [1] CRAN (R 3.6.0)
##  vctrs          0.2.4   2020-03-10 [1] CRAN (R 3.6.0)
##  viridis      * 0.5.1   2018-03-29 [1] CRAN (R 3.6.0)
##  viridisLite  * 0.3.0   2018-02-01 [1] CRAN (R 3.6.0)
##  withr          2.1.2   2018-03-15 [1] CRAN (R 3.6.0)
```

```
##  xfun           0.12    2020-01-13 [1] CRAN (R 3.6.0)
##  xml2           1.2.5   2020-03-11 [1] CRAN (R 3.6.0)
##  yaml           2.2.1   2020-02-01 [1] CRAN (R 3.6.0)
##  zip            2.0.4   2019-09-01 [1] CRAN (R 3.6.0)
##
## [1] /Library/Frameworks/R.framework/Versions/3.6/Resources/library
```