Confidence intervals

Florian Jaeger

Fall 2020

Contents

1 Goals of this document

This document is intended to introduce the basics of confidence intervals.

1.1 In preparation for class

Before class, please read the remainder of Section 1. Then watch this great video on confidence intervals by Dr. Nic. Then watch this simulation video by the Khan Academy on the meaning of confidence intervals and what happens to those confidence intervals, if we repeat (replicate) an experiment.

You might prefer to instead (or also) have a manuscript that covers this topic in writing. Cumming's (2013) suggestions for a 'new' statistics is a worthwhile read and has a self-contained section on "Replication, p values, and CIs" that I recommend. You can just read those 2 pages.

1.2 Important concepts and terminology

In the brain and cognitive sciences, the goal of conducting experiments is to advance our understanding of the brain/mind. We aim to test the **predictions** of **theories** or more specific **hypotheses**. These theories typically constitute propositions about causes, and the statistical tests we conduct typically assess the predicted correlations that would result from the hypothesized causal relations. These correlations are

assessed on some finite amount of data from *samples* we've obtained from the underyling *population* of interest.

To illustrate this with an example, remember the experiment on visual crowding from last week. We hypothesized that (H1) performance in a number recognition task increases with increasing size of the number and that (H2) performance decreases in visually crowded displays. We elicited two types of **dependent variables** or **outcomes**: response time and response accuracy. To turn our hypotheses into predictions, we implicitly assumed **linking hypotheses** that link the abstract theoretical construct ("performance") to the specific dependent measures we employ in our experiment: higher performance → faster processing → response times and higher performance → higher accuracy.

To test the resulting predictions, we manipulated both the number size (size 1, 2, 4) and the visual crowding (easy: no crowding, hard: numbers flanking the target number on all sides). Thus size and crowding are our **independent variables** or **predictors**.

When we analyze data from our experiment, we assume that the **observations** we obtain constitute a **sample** of the underlying **population**. We then use this finite sample to draw **inferences** about the population. For example, a really typical question we ask would be whether the means of the population differ between two conditions. Sticking with our example, we might test our hypothesis about the brain/mind by asking whether we can confidently conclude that reaction times are indeed smaller in for high frequency objects, compared to reactions times for high frequency objects. Either way, this constitutes an inference we're aiming to draw about the world, based on the finite sample of observations we collected in our experiment.

Specifically, we will compare the condition **means in our sample**. For example to test (H2), we would compare the mean response time in the easy and hard condition of the experiment. In null hypothesis significance testing (NHST), we do this comparison by **assuming the null hypothesis that there is *no* difference** and then assessing how improbable the actual sample means are under that null hypothesis. If the sample means are too far apart from each other so that their difference is improbable to have happened by chance under the null hypothesis, we **reject the null hypothesis** and speak of a **significant difference**.

But how do we determine how different is "too different", so that we should reject the null hypothesis? For this, we need a measure of our **uncertainty**. Intuitively, this measure should depend on the inherent **random variability** of the data: the more variable the data is—i.e., the more each sample drawn from the underlying population will differ from each other—the less certain we can be that the same absolute difference between two condition means is meaningful (or in NHST terms: "significant"). This is where confidence intervals come in.

1.3 Confidence intervals

Confidence intervals are an effective means to summarize our uncertainty. From Cohen (1994, p. 1002) as quoted on the ReplicationIndex blog:

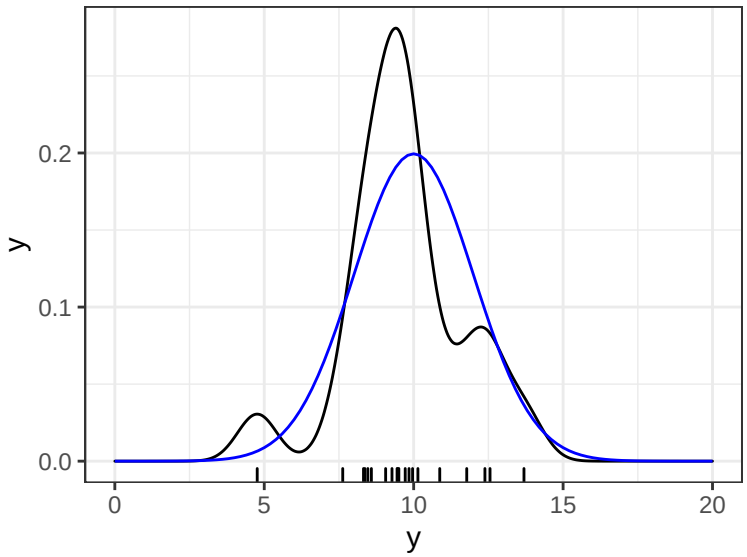
> "Everyone knows" that confidence intervals contain all the information to be found in significance tests and much more. They not only reveal the status of the trivial nil hypothesis but also about the status of non-nil null hypotheses and thus help remind researchers about the possible operation of the crud factor. Yet they are rarely to be found in the literature. I suspect that the main reason they are not reported is that they are so embarrassingly large! But their sheer size should move us toward improving our measurement by seeking to reduce the unreliable and invalid part of the variance in our measures (as Student himself recommended almost a century ago). Also, their width provides us with the analogue of power analysis in significance testing—larger sample sizes reduce the size of confidence intervals as they increase the statistical power of NHST.

Confidence intervals on the one hand are rather intuitive—in that they provide a measure of variability or uncertainty—but the specifics of confidence intervals are **often misunderstood and misinterpreted**. The following summary of common misunderstandings provides a concise overview. The somewhat unintuitive

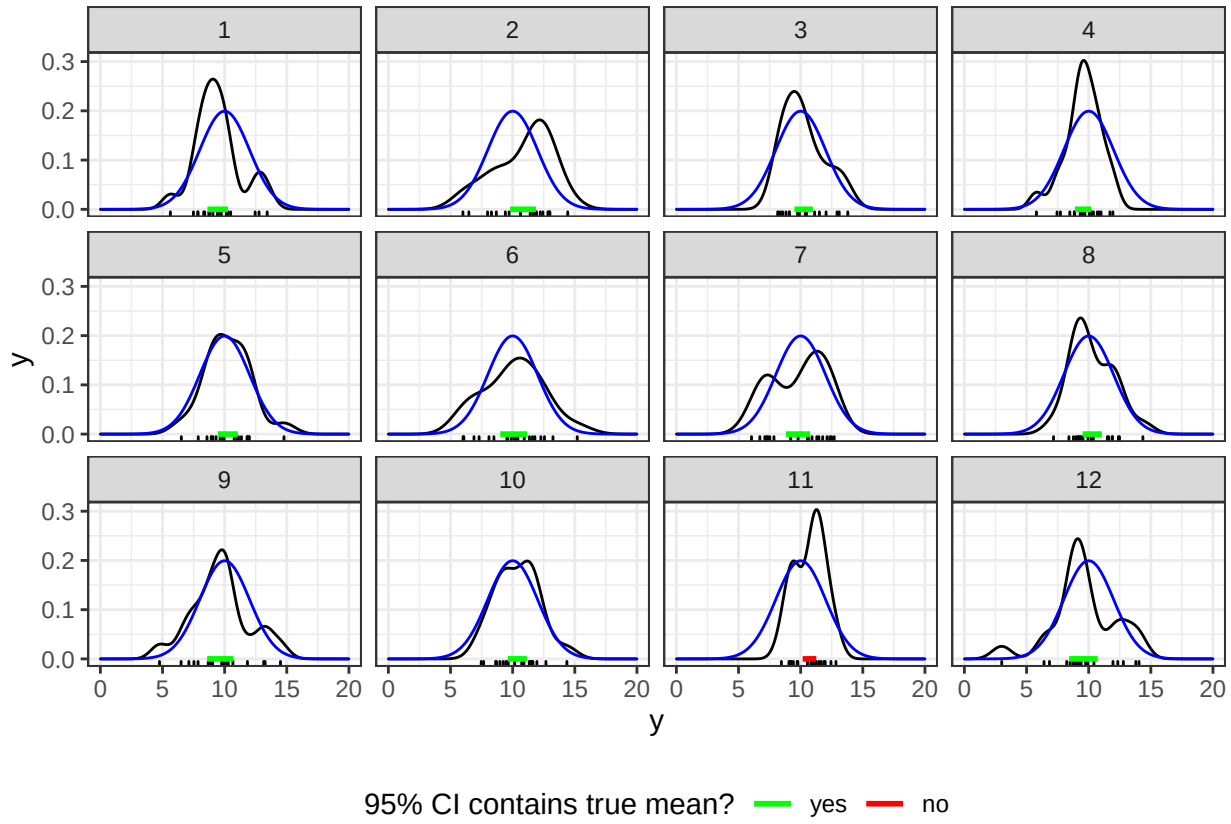
interpretation of CIs is that—provided they are constructed properly—95% of all 95% CIs over repeated samples from a population will contain the true population mean.

2 Illustrating the meaning of confidence intervals

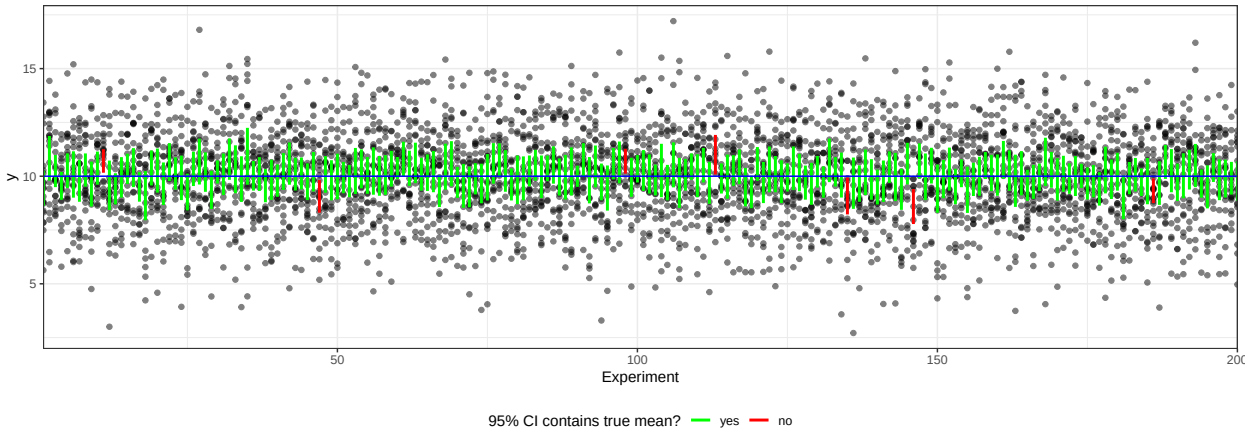
To illustrate the notion of CIs let's go through a simple example. For this purpose, we will choose an example where we know the 'ground truth'—unlike in typical research. That is, we will generate our own data. Specifically, imagine that we have a normally distributed outcome with mean 10 and standard deviation 2. We take 20 samples from that distribution. Here's one instance of this thought experiment. The blue line shows the normal distribution we are sampling from. The tick marks show the 20 data points we sampled. The black line is a density estimate of the 20 data points.



Here are 12 more instances of this thought experiment (the blue line is always the same, as it shows the *population's distribution*). This time I've also plotted a 95% CI based on the (in this case, correct) assumption of a normally distributed outcome. We'll learn below how it was constructed.



To focus on the CIs, let's just plot the 20 observations and the CI for each instance of the experiment. We'll essentially make the x-axis of the previous plot the y-axis of the new plot, and use the x-axis to indicate the different experiments. That way we can plot many more thought experiments next to each other. Let's plot 200 of them. The CI is colored green if it includes the true mean and red if it doesn't.



This begins to give us an intuition as to what it means that 95% of our 95% CIs over repeated experiments are guaranteed to contain the true mean of the data. We see 7 red lines out of 200 (3.5%), so that 96.5% of the 200 CIs we have constructed contain the true mean. If we continue to collect more samples we will find that in the long run 95% of the 95% CIs contain the true mean.

Of course, for a real-life example we won't know the true mean. But we will still know that if we repeat the

experiment over and over again then 95% of the 95% CIs will contain the true mean.

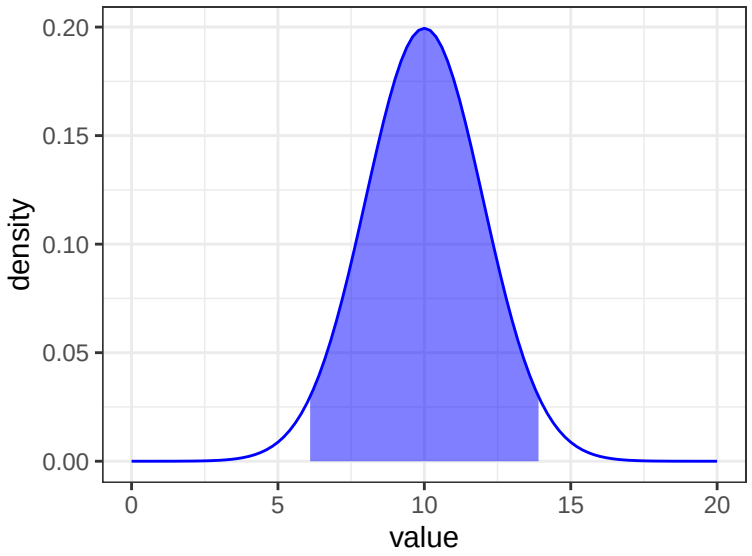
3 How to obtain confidence intervals

3.1 Parametric confidence intervals

When the distribution of a variable is known—or if we are willing to make assumptions about that distributions—there might be a known analytic solution to calculating the confidence intervals. For example, under the assumption that an outcome y are normally distributed around an unknown mean mu the 95% CI around the *sample* mean $mean_y$ is:

$mean_y \pm 1.96 * \frac{SD_y}{\sqrt{n}}$

where SD_y is the *sample's* standard deviation of y—i.e., the standard deviation of y in our observed sample. Why is it ± 1.96? Because 95% of the density of a normal distribution lie between its mean $\pm 1.959964...$ its standard deviation—i.e., between its .025 and .975 quantile. E.g. for our example from above, a normal distribution with mean 10 and standard deviation 2:



This approach to CIs is called parametric because the CIs are obtained under the assumption of a specific parametric distribution. For example, the normal distribution is a parametric distribution, a distribution specified by parameters (the mean and the standard deviation).

Notice two important properties of the CIs defined above, both due to the term $\frac{SD_y}{\sqrt{n}}$ (the so-called **standard error of the sample mean**):

- the more variable the data (SD_y), the larger/wider the CIs
- the more sample data we have, the smaller/more narrow the CIs

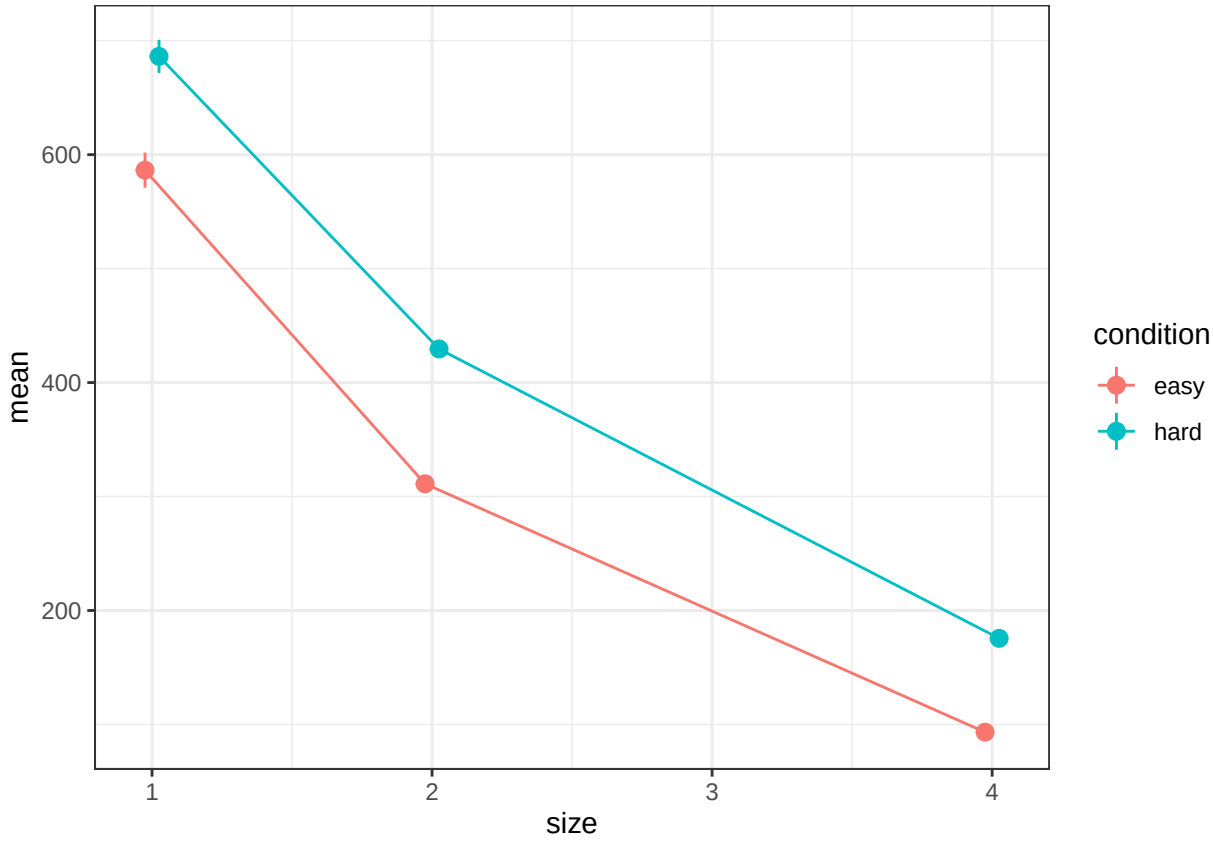
To make this more concrete, consider the experiment from last week. Let's calculate the 95% CIs for the six conditions defined by crossing the three number sizes and two crowding conditions. We'll do so for subject 1.

```
d = read_csv("../../data/example_data.csv")

d.summary =  d %>%
  filter(subject == 1) %>%
```

```
  group_by(size, condition) %>%
  summarise_at(
    .vars = "RT",
    .funs = list(
      "mean" = function(x) mean(x),
      "CI.lower" = function(x) { mean(x) - 1.96 * sd(x) / sqrt(length(x) - 1) },
      "CI.upper" = function(x) { mean(x) + 1.96 * sd(x) / sqrt(length(x) - 1) }))

print(d.summary)
```

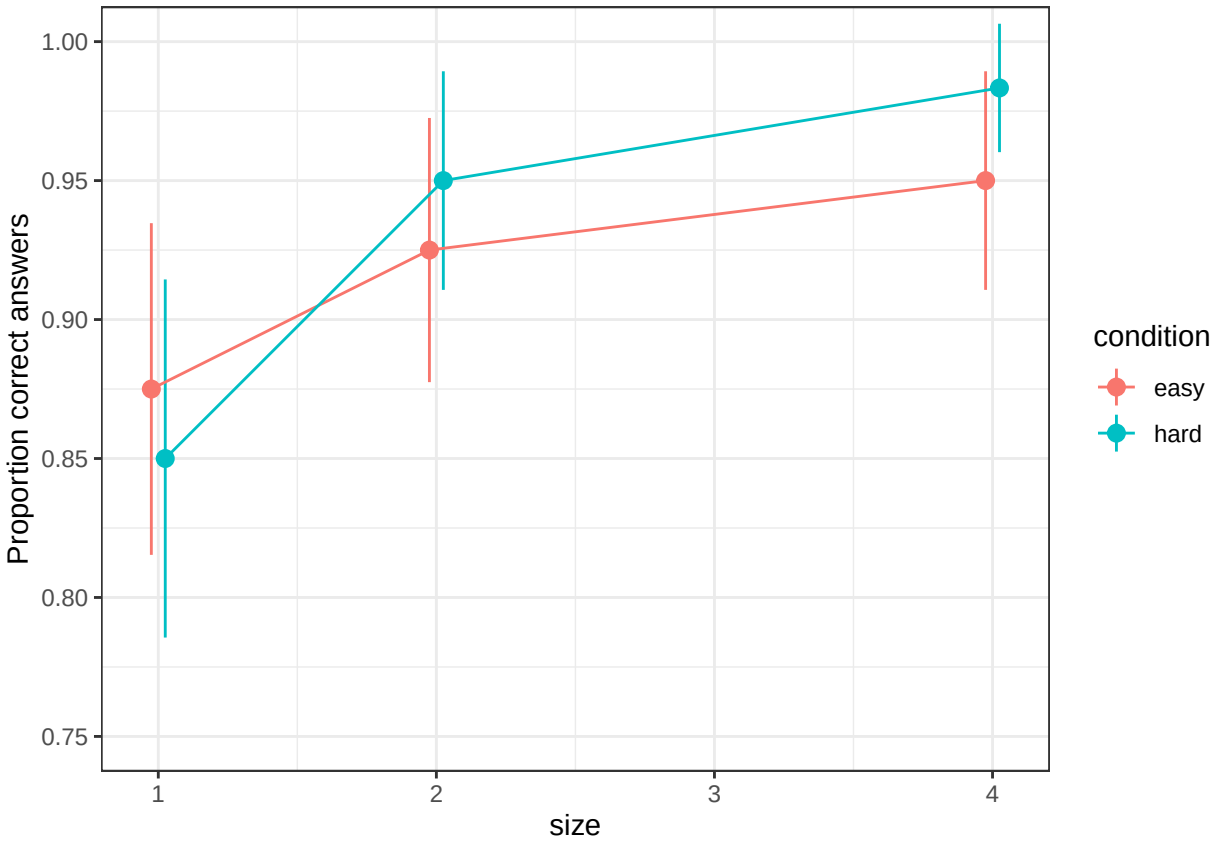
```
## # A tibble: 6 x 5
## # Groups:   size [3]
##    size condition  mean CI.lower CI.upper
##   <dbl> <chr>     <dbl>    <dbl>    <dbl>
## 1     1 easy      586.     571.     602.
## 2     1 hard      686.     672.     701.
## 3     2 easy      311.     305.     317.
## 4     2 hard      429.     422.     437.
## 5     4 easy       93.2     91.1     95.3
## 6     4 hard      176.     172.     179.
```

Or plotted:



3.2 Issues with parametric confidence intervals

One issue with parametric CIs becomes apparent when we make parametric assumptions that are *wrong*. For example, the correctness of the responses in the visual crowding data follows a Bernoulli, rather than Normal, distribution. If we obtain CIs for the mean proportion of correct responses in the six conditions following the procedure we used for the RTs, we get the following:



Note how the 95% CIs include values outside of the possible range for proportions (larger than 1 in this case). This is a clear indication that we cannot safely assume normality for this type of data. There are generally two ways around this issue: either we obtain CIs under assumptions that reflect the actual generative process underlying the data, or we use non-parametric methods to obtain CIs that do not assume a specific parametric form of the distribution. The second approach has the advantage that it works for any type of distribution, which is great since the true distribution of the outcome is typically not known!

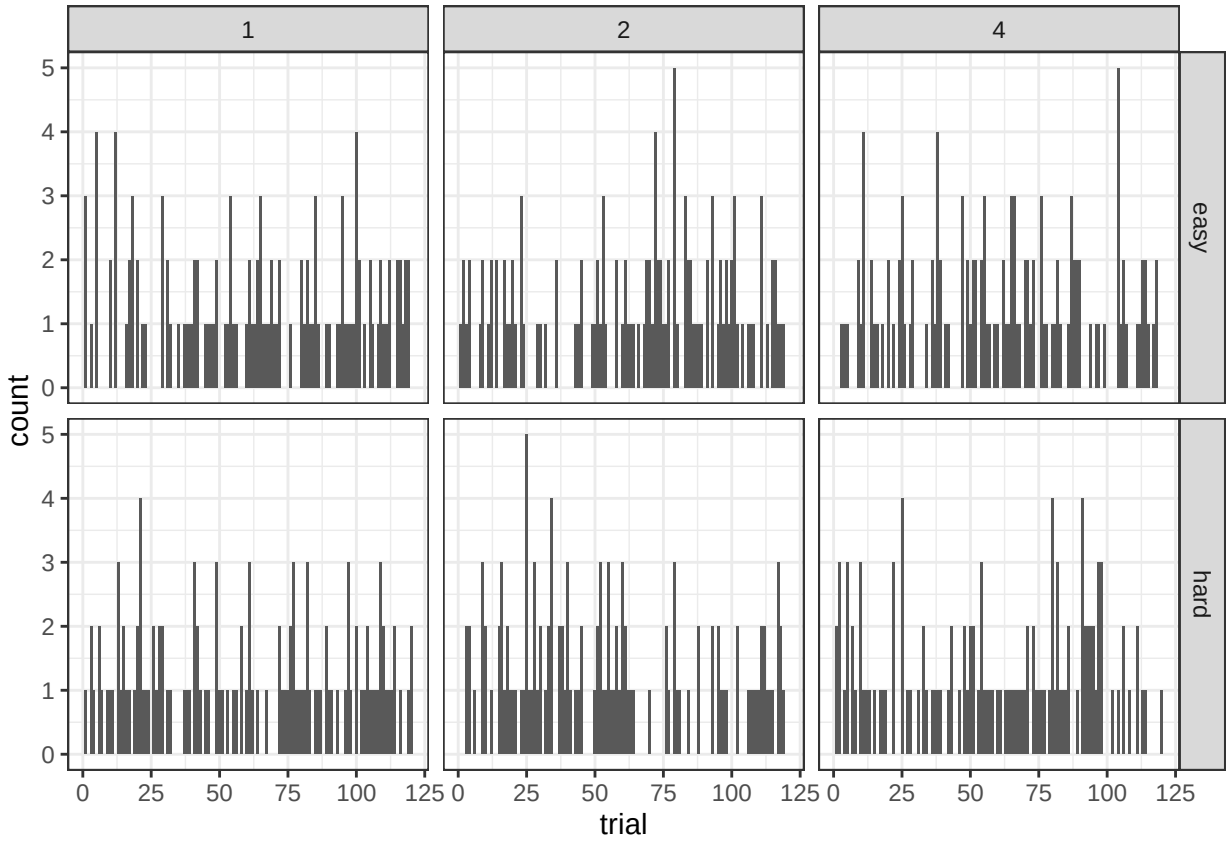
3.3 Non-parametric confidence intervals

One non-parametric method to calculate CIs is the **bootstrap**. There are many different implementations of it, and the details can become quite complicated depending on the type of data you are analyzing. At the core though, the bootstrap is based on a rather intuitive idea. Rather than to make parametric assumptions about the distribution of the data and the resulting expected distribution of the sample means (across samples), we can use the sample data we have to estimate the distribution. Specifically, we can randomly sample with replacement a new sample *from the sample data we have*. For example, for subject 1 from the same experiment as above we resample 720 new trials (balanced in the same way as before) from the 720 trials we have from that subject:

```
## # A tibble: 720 x 6
```

```
## # Groups:   size, condition [6]
##     size condition trial subject correct    RT
##    <dbl> <chr>     <dbl>   <dbl>   <dbl> <dbl>
##  1     1 easy         65       1       1  551.
##  2     1 easy         47       1       1  487.
##  3     1 easy         61       1       1  532.
##  4     1 easy         29       1       1  481.
##  5     1 easy         42       1       1  672.
##  6     1 easy         20       1       1  548.
##  7     1 easy         10       1       0  673.
##  8     1 easy        110       1       1  587.
##  9     1 easy        109       1       1  412.
## 10     1 easy         40       1       1  462.
## # ... with 710 more rows
```

Note that these are not the same trials as before. We can see that by plotting a histogram of the original trial ID in the newly resampled data. We can see that some trials were sampled multiple times, and others were sampled never:



Now we can obtain the condition means from this resampled data. By repeating this process many times, we can thus obtain an estimate of the *distribution of the condition means* across random samples. For example, from 1000 random resamples of the data, we obtain 1000 means for each of the six conditions. Here I'm illustrating this for the mean proportion of correct responses for which we have seen above that the assumption of normality is problematic:

```
## # A tibble: 6,000 x 4
## # Groups:   sample, size [3,000]
##    sample  size condition sample_mean
```

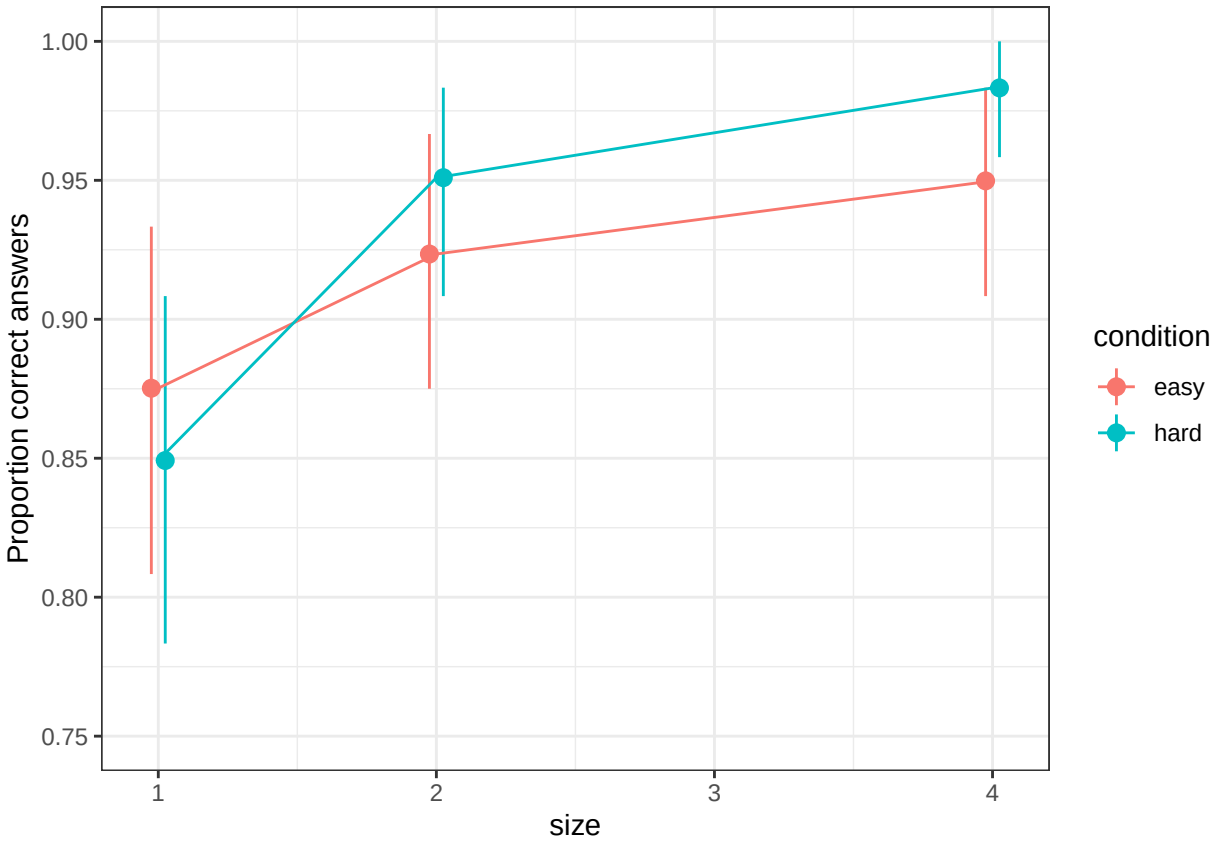
```
##     <int> <dbl> <chr>           <dbl>
##  1      1     1 easy            0.875
##  2      1     1 hard            0.875
##  3      1     2 easy            0.917
##  4      1     2 hard            0.942
##  5      1     4 easy            0.975
##  6      1     4 hard            0.975
##  7      2     1 easy            0.892
##  8      2     1 hard            0.867
##  9      2     2 easy            0.967
## 10      2     2 hard            0.942
## # ... with 5,990 more rows
```

Now we simply ask what the 2.5th to 97.5th quantiles of each of these six distribution of means are. Basically, we determine the range around the mean in which 95% of the resampled means fall. Notice that the CIs based on these quantiles do not (and can never) fall outside of the possible range of the data. Notice also that the CIs are asymmetric around the mean, as we would expect because precisely because they should not include values outside of the [0,1] range:

```
d.sample_summary = d.sample_means %>%
  group_by(size, condition) %>%
  summarise(
    mean = mean(sample_mean),
    CI.lower = quantile(sample_mean, .025),
    CI.upper = quantile(sample_mean, .975)
  )

print(d.sample_summary)
```

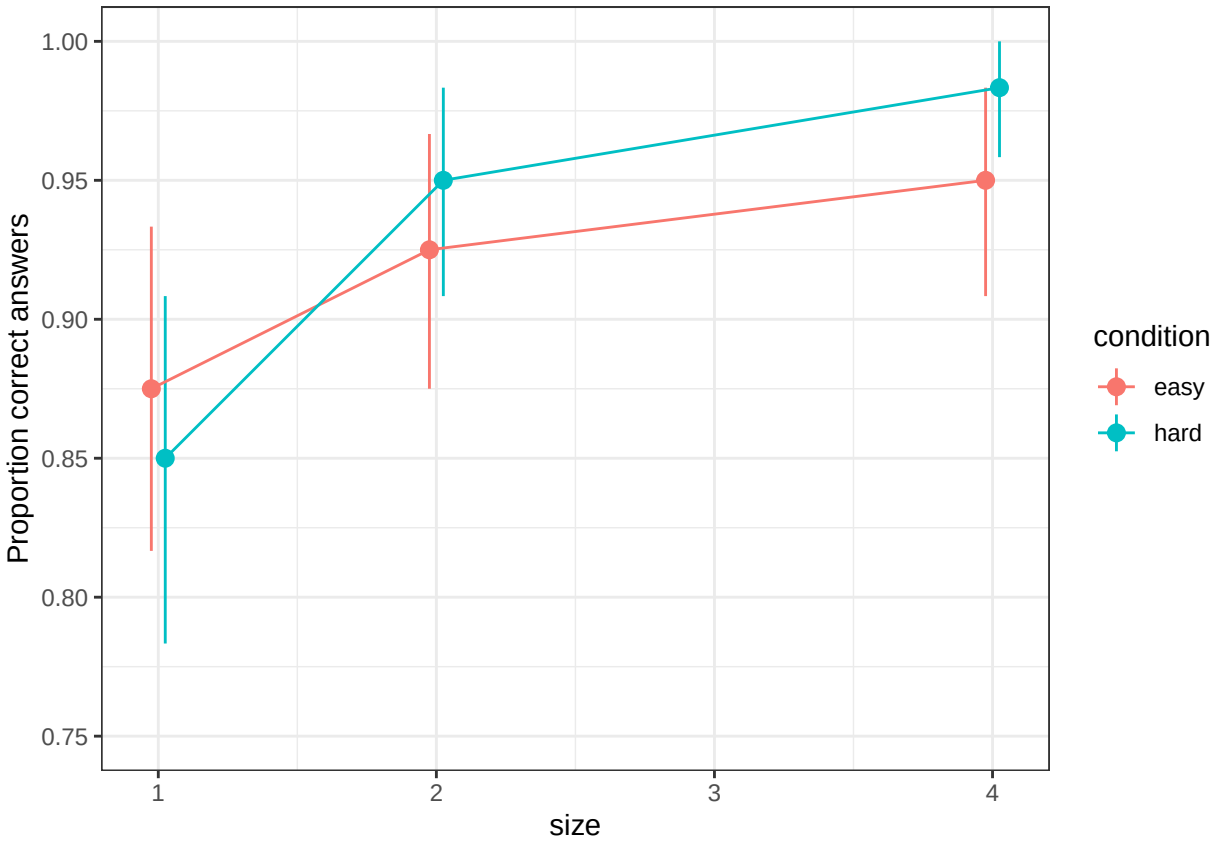
```
## # A tibble: 6 x 5
## # Groups:   size [3]
##    size condition  mean CI.lower CI.upper
##   <dbl> <chr>     <dbl>    <dbl>    <dbl>
## 1     1 easy      0.875    0.808    0.933
## 2     1 hard      0.849    0.783    0.908
## 3     2 easy      0.923    0.875    0.967
## 4     2 hard      0.951    0.908    0.983
## 5     4 easy      0.950    0.908    0.983
## 6     4 hard      0.983    0.958    1
```

```
d.sample_summary %>%
  ggplot(aes(x = size,
             y = mean,
             ymin = CI.lower,
             ymax = CI.upper,
             color = condition)) +
  geom_pointrange(position = position_dodge(.1)) +
  geom_line() +
  scale_y_continuous("Proportion correct answers") +
  coord_cartesian(ylim = c(0.75,1))
```



Compare this to the CIs obtained from the bootstrap function built into ggplot:

```
d %>%
  filter(subject == 1) %>%
  ggplot(aes(x = size,
             y = correct,
             color = condition)) +
  stat_summary(
    fun.data = mean_cl_boot,
    geom = "pointrange",
    position = position_dodge(.1)) +
  stat_summary(
    fun.data = mean_cl_boot,
    geom = "line",
    position = position_dodge(.1)) +
  scale_y_continuous("Proportion correct answers") +
  coord_cartesian(ylim = c(0.75,1))
```



Bootstrap is thus a powerful (and ultimately simple) option for obtained confidence intervals.One important limitation to keep in mind about the bootstrap though is that it is utterly dependent on the representativeness of the sample data. One direct consequence of that is that bootstrapped CIs become more robust the more sample data we have. In the Spring semester, we will return to this and other sampling-based approaches to statistics.

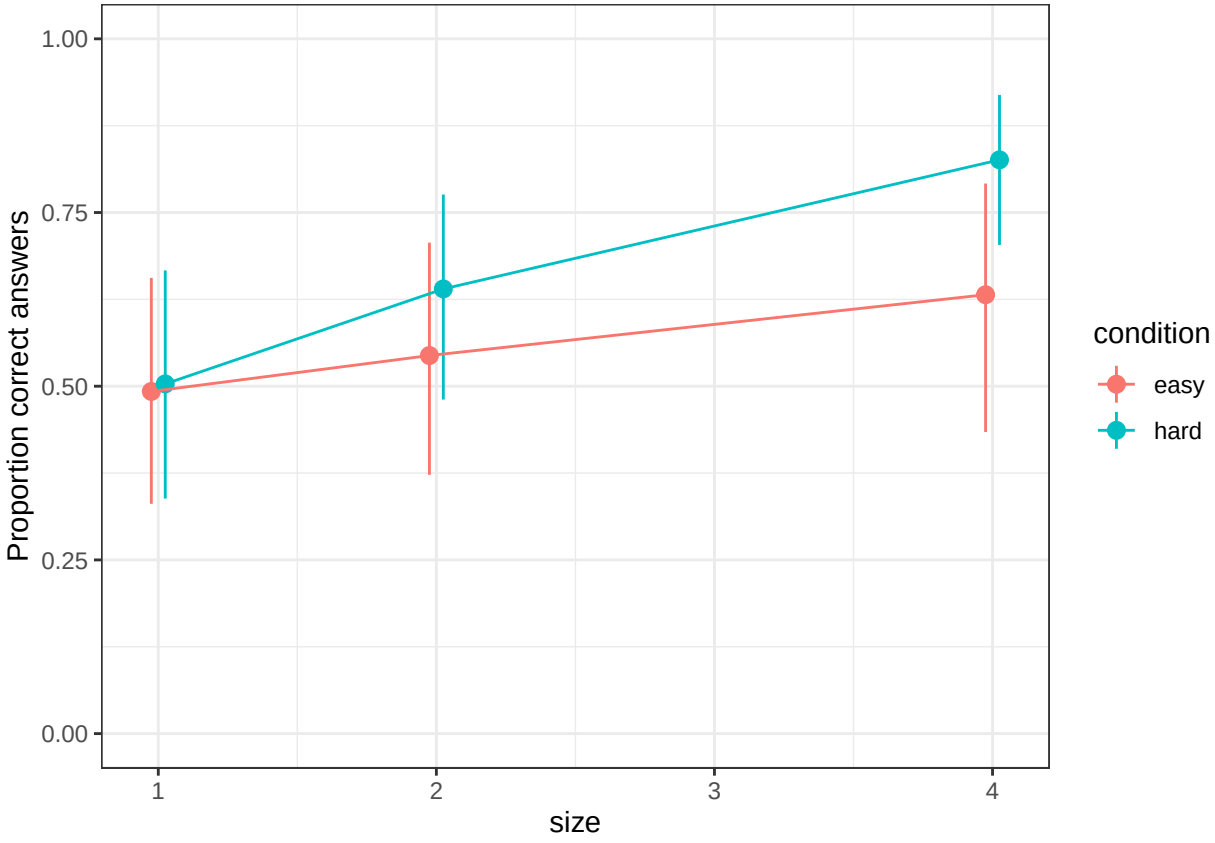
4 Confidence intervals for grouped (repeated measures) data

So far we have obtained CIs only for data from a single subject. When we have repeated measures from multiple subjects, we can still use the approaches discussed here. We should, however, apply them to the *by-subject means* rather than the raw data to obtain visualizations that approximate appropriately conservative (i.e., wide) CIs.

Consider, for example, that we would like to plot the CIs for the same six conditions analyzed above but would like to do so for all subjects together rather than just subject 1. In that case, we should first summarize the data down to one mean for each of the six conditions for each subject, and then obtain CIs for the distribution of these by-subject means:

```
d %>%
  # the next two lines first create by-subject means for each of the six conditions
  group_by(subject, condition, size) %>%
  summarize(correct = mean(correct)) %>%
  # the rest of this code is the same as above
  ggplot(aes(x = size,
             y = correct,
```

```
            color = condition)) +
stat_summary(
  fun.data = mean_cl_boot,
  geom = "pointrange",
  position = position_dodge(.1)) +
stat_summary(
  fun.data = mean_cl_boot,
  geom = "line",
  position = position_dodge(.1)) +
scale_y_continuous("Proportion correct answers") +
coord_cartesian(ylim = c(0,1))
```



5 Confidence intervals around the mean vs. around *differences* in a pair of means

Plots of condition means like the ones we have seen above are very common in scientific papers. They provide intuitive summaries of the raw data. But how do the confidence intervals of the condition means relate to the statistical tests we conduct? Can we infer from a plot of condition means and CIs whether there is a significant effect? The short answer is "kind of" but "not really".

The reason for that is that our statistical tests typically assess hypotheses about *differences between means*, rather than the means themselves. For example, if want to know whether RTs in the the easy (uncrowded) condition were significantly faster than RTs in the hard (crowded) condition we are testing whether we can reject the null hypothesis that the *difference* in RTs between these conditions is 0. It is therefore the CI

of the mean difference between the conditions that tells whether there is an effect. For the conventional sigificance criterion of .05, we would reject the null hypothesis if the 95% CI around the mean difference in RTs between the easy and hard condition does not contain 0.

So unless we plot difference between RTs, rather than the mean RTs of all conditions, the CIs in the plot do not directly map onto the p-values of our statistical tests. That said, especially when we're not dealing with paired data (i.e., if the data aren't grouped) significant differences in the means will cause the CIs around a means to not include the other mean.

6 Session info

```
## - Session info  -------------------------------------------------------------
##  hash: grinning squinting face, flag: Malta, synagogue
##
##  setting  value
##  version  R version 4.1.0 (2021-05-18)
##  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     2021-11-06
##  pandoc   2.14.2 @ /Applications/RStudio.app/Contents/MacOS/pandoc/ (via rmarkdown)
##
## - Packages -----------------------------------------------------------------
##  package      * version date (UTC) lib source
##  assertthat     0.2.1   2019-03-21 [1] CRAN (R 4.1.0)
##  backports      1.3.0   2021-10-27 [1] CRAN (R 4.1.0)
##  base64enc      0.1-3   2015-07-28 [1] CRAN (R 4.1.0)
##  bit            4.0.4   2020-08-04 [1] CRAN (R 4.1.0)
##  bit64          4.0.5   2020-08-30 [1] CRAN (R 4.1.0)
##  boot         * 1.3-28  2021-05-03 [1] CRAN (R 4.1.0)
##  broom          0.7.10  2021-10-31 [1] CRAN (R 4.1.0)
##  cachem         1.0.6   2021-08-19 [1] CRAN (R 4.1.0)
##  callr          3.7.0   2021-04-20 [1] CRAN (R 4.1.0)
##  cellranger     1.1.0   2016-07-27 [1] CRAN (R 4.1.0)
##  checkmate      2.0.0   2020-02-06 [1] CRAN (R 4.1.0)
##  cli            3.1.0   2021-10-27 [1] CRAN (R 4.1.0)
##  cluster        2.1.2   2021-04-17 [1] CRAN (R 4.1.0)
##  colorspace     2.0-2   2021-06-24 [1] CRAN (R 4.1.0)
##  crayon         1.4.2   2021-10-29 [1] CRAN (R 4.1.0)
##  data.table     1.14.2  2021-09-27 [1] CRAN (R 4.1.0)
##  DBI            1.1.1   2021-01-15 [1] CRAN (R 4.1.0)
##  dbplyr         2.1.1   2021-04-06 [1] CRAN (R 4.1.0)
##  desc           1.4.0   2021-09-28 [1] CRAN (R 4.1.0)
##  devtools       2.4.2   2021-06-07 [1] CRAN (R 4.1.0)
##  digest         0.6.28  2021-09-23 [1] CRAN (R 4.1.0)
##  dplyr        * 1.0.7   2021-06-18 [1] CRAN (R 4.1.0)
##  ellipsis       0.3.2   2021-04-29 [1] CRAN (R 4.1.0)
##  evaluate       0.14    2019-05-28 [1] CRAN (R 4.1.0)
##  fansi          0.5.0   2021-05-25 [1] CRAN (R 4.1.0)
```

```
##  farver          2.1.0   2021-02-28 [1] CRAN (R 4.1.0)
##  fastmap         1.1.0   2021-01-25 [1] CRAN (R 4.1.0)
##  forcats       * 0.5.1   2021-01-27 [1] CRAN (R 4.1.0)
##  foreign         0.8-81  2020-12-22 [1] CRAN (R 4.1.0)
##  Formula         1.2-4   2020-10-16 [1] CRAN (R 4.1.0)
##  fs              1.5.0   2020-07-31 [1] CRAN (R 4.1.0)
##  generics        0.1.1   2021-10-25 [1] CRAN (R 4.1.0)
##  ggplot2       * 3.3.5   2021-06-25 [1] CRAN (R 4.1.0)
##  glue            1.4.2   2020-08-27 [1] CRAN (R 4.1.0)
##  gridExtra       2.3     2017-09-09 [1] CRAN (R 4.1.0)
##  gtable          0.3.0   2019-03-25 [1] CRAN (R 4.1.0)
##  haven           2.4.3   2021-08-04 [1] CRAN (R 4.1.0)
##  Hmisc           4.6-0   2021-10-07 [1] CRAN (R 4.1.0)
##  hms             1.1.1   2021-09-26 [1] CRAN (R 4.1.0)
##  htmlTable       2.3.0   2021-10-12 [1] CRAN (R 4.1.0)
##  htmltools       0.5.2   2021-08-25 [1] CRAN (R 4.1.0)
##  htmlwidgets     1.5.4   2021-09-08 [1] CRAN (R 4.1.0)
##  httr            1.4.2   2020-07-20 [1] CRAN (R 4.1.0)
##  jpeg            0.1-9   2021-07-24 [1] CRAN (R 4.1.0)
##  jsonlite        1.7.2   2020-12-09 [1] CRAN (R 4.1.0)
##  knitr           1.36    2021-09-29 [1] CRAN (R 4.1.0)
##  labeling        0.4.2   2020-10-20 [1] CRAN (R 4.1.0)
##  lattice         0.20-45 2021-09-22 [1] CRAN (R 4.1.0)
##  latticeExtra    0.6-29  2019-12-19 [1] CRAN (R 4.1.0)
##  lifecycle       1.0.1   2021-09-24 [1] CRAN (R 4.1.0)
##  lubridate       1.8.0   2021-10-07 [1] CRAN (R 4.1.0)
##  magrittr      * 2.0.1   2020-11-17 [1] CRAN (R 4.1.0)
##  Matrix          1.3-4   2021-06-01 [1] CRAN (R 4.1.0)
##  memoise         2.0.0   2021-01-26 [1] CRAN (R 4.1.0)
##  modelr          0.1.8   2020-05-19 [1] CRAN (R 4.1.0)
##  munsell         0.5.0   2018-06-12 [1] CRAN (R 4.1.0)
##  nnet            7.3-16  2021-05-03 [1] CRAN (R 4.1.0)
##  pillar          1.6.4   2021-10-18 [1] CRAN (R 4.1.0)
##  pkgbuild        1.2.0   2020-12-15 [1] CRAN (R 4.1.0)
##  pkgconfig       2.0.3   2019-09-22 [1] CRAN (R 4.1.0)
##  pkgload         1.2.3   2021-10-13 [1] CRAN (R 4.1.0)
##  png             0.1-7   2013-12-03 [1] CRAN (R 4.1.0)
##  prettyunits     1.1.1   2020-01-24 [1] CRAN (R 4.1.0)
##  processx        3.5.2   2021-04-30 [1] CRAN (R 4.1.0)
##  ps              1.6.0   2021-02-28 [1] CRAN (R 4.1.0)
##  purrr         * 0.3.4   2020-04-17 [1] CRAN (R 4.1.0)
##  R6              2.5.1   2021-08-19 [1] CRAN (R 4.1.0)
##  RColorBrewer    1.1-2   2014-12-07 [1] CRAN (R 4.1.0)
##  Rcpp            1.0.7   2021-07-07 [1] CRAN (R 4.1.0)
##  readr         * 2.0.2   2021-09-27 [1] CRAN (R 4.1.0)
##  readxl          1.3.1   2019-03-13 [1] CRAN (R 4.1.0)
##  remotes         2.4.1   2021-09-29 [1] CRAN (R 4.1.0)
##  reprex          2.0.1   2021-08-05 [1] CRAN (R 4.1.0)
##  rlang           0.4.12  2021-10-18 [1] CRAN (R 4.1.0)
##  rmarkdown       2.11    2021-09-14 [1] CRAN (R 4.1.0)
##  rpart           4.1-15  2019-04-12 [1] CRAN (R 4.1.0)
##  rprojroot       2.0.2   2020-11-15 [1] CRAN (R 4.1.0)
##  rstudioapi      0.13    2020-11-12 [1] CRAN (R 4.1.0)
##  rvest           1.0.2   2021-10-16 [1] CRAN (R 4.1.0)
```

```
##  scales         1.1.1   2020-05-11 [1] CRAN (R 4.1.0)
##  sessioninfo    1.2.1   2021-11-02 [1] CRAN (R 4.1.0)
##  stringi        1.7.5   2021-10-04 [1] CRAN (R 4.1.0)
##  stringr      * 1.4.0   2019-02-10 [1] CRAN (R 4.1.0)
##  survival       3.2-13  2021-08-24 [1] CRAN (R 4.1.0)
##  testthat       3.1.0   2021-10-04 [1] CRAN (R 4.1.0)
##  tibble       * 3.1.5   2021-09-30 [1] CRAN (R 4.1.0)
##  tidyr        * 1.1.4   2021-09-27 [1] CRAN (R 4.1.0)
##  tidyselect     1.1.1   2021-04-30 [1] CRAN (R 4.1.0)
##  tidyverse    * 1.3.1   2021-04-15 [1] CRAN (R 4.1.0)
##  tzdb           0.2.0   2021-10-27 [1] CRAN (R 4.1.0)
##  usethis        2.1.3   2021-10-27 [1] CRAN (R 4.1.0)
##  utf8           1.2.2   2021-07-24 [1] CRAN (R 4.1.0)
##  vctrs          0.3.8   2021-04-29 [1] CRAN (R 4.1.0)
##  vroom          1.5.5   2021-09-14 [1] CRAN (R 4.1.0)
##  withr          2.4.2   2021-04-18 [1] CRAN (R 4.1.0)
##  xfun           0.27    2021-10-18 [1] CRAN (R 4.1.0)
##  xml2           1.3.2   2020-04-23 [1] CRAN (R 4.1.0)
##  yaml           2.2.1   2020-02-01 [1] CRAN (R 4.1.0)
##
##  [1] /Library/Frameworks/R.framework/Versions/4.1/Resources/library
##
## ------------------------------------------------------------------------------
```