Towards reproducibility in small-N treatment research in aphasiology: a tutorialy

Robert Cavanaugh[1], Yina Quique[2], Alexander Swiderski[1], Lydia Kalhoff[3], Lauren Terhorst[1], Julie Wambaugh[3], William D. Hula[4], & William S. Evans[1]

[1] University of Pittsburgh
[2] Northwestern University
[3] University of Utah
[4] VA Pittsburgh Healthcare System

Abstract

Purpose: Small-N studies are the dominant study design supporting evidence-based treatment studies in communication sciences and disorders, and specifically in research on aphasia and related disorders. However, there is little guidance on conducting reproducible analysis of such studies, which has implications for scientific review, rigor, and replication.
Methods: This tutorial demonstrates how to implement reproducible analyses of small-N designs by reanalyzing data from Wambaugh et al. (2017), a single-case experimental design study with 20 individuals with post-stroke apraxia of speech and aphasia receiving Sound Production Treatment. A comparison and discussion of the strengths and weaknesses of small-N effect sizes is provided so that researchers can make informed decisions about how to best characterize treatment effects for their own work.
Results: Tutorial code demonstrates how to implement the following effect sizes: standardized mean difference, Proportion of Maximal Gain, Tau-U, and mixed-effects models at the individual and group level in the statistical language R. Data and code are publicly available as a resource for students, researchers, and clinicians.
Conclusion: This tutorial demonstrates how researchers in aphasia and related disorders can conduct reproducible analysis of small-N studies, the dominant intervention design in the field. We also demonstrate how properties of different approaches to statistical analysis can affect the interpretation and replication of small-N studies. This article may serve as a template for conducting reproducible analyses of the small-N designs common to aphasia and related disorders.

Setup

Load packages and functions

```r
library(here)            # for locating files
library(tidyverse)       # data wrangling
library(SingleCaseES)    # calculating SMD, Tau-U
library(lme4)            # frequentist mixed-effects models
library(emmeans)         # estimating effect sizes from lme4
library(brms)            # bayesian mixed-effects models
library(tidybayes)       # estimating effect sizes from brms
library(ggdist)          # Visualizing posterior distributions

# set a seed for reproducibility
set.seed(42)
```

Read in data

Note that the current setup uses RStudio R projects (https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects). One of the features of R projects is that the working directory is automatically set to the project root (the folder with the .Rproj). A discussion of R projects can be found at https://www.tidyverse.org/blog/2017/12/workflow-vs-script/. In this case `here("study-data")` refers to the /study-data folder inside the project.

```r
# create a list of files
files <- list.files(
                 here("study-data"), # look in the study-data folder
                 full.names = TRUE,  # use the full paths of the files
                 pattern = ".csv",   # only read in .csv files
                 recursive = TRUE)   # include files within subfolders

# read in the files and combine them together
# map_df takes a function, in this case read_csv().
# show_col_types suppresses output sinc we're reading in many files
df <- files %>%
  map_dfr(read_csv, show_col_types = FALSE)
```

Preview the data

```r
head(df)
```

```
## # A tibble: 6 x 11
```

Correspondence concerning this article should be addressed to Robert Cavanaugh. E-mail: rob.cavanaugh@pitt.edu

```
##   participant condition phoneme itemType phase    session item  trials spt2017
##   <chr>       <chr>     <chr>   <chr>    <chr>      <dbl> <chr>  <dbl> <chr>
## 1 P1          blocked   pr      tx       baseline       1 pr-1      10 pre
## 2 P1          blocked   pr      tx       baseline       1 pr-12     10 pre
## 3 P1          blocked   pr      tx       baseline       1 pr-4      10 pre
## 4 P1          blocked   pr      tx       baseline       1 pr-15     10 pre
## 5 P1          blocked   pr      tx       baseline       1 pr-5      10 pre
## 6 P1          blocked   pr      tx       baseline       1 pr-7      10 pre
## # ... with 2 more variables: response <dbl>, n_baselines <dbl>
```

Variable	Description
participant	de-identified participant ID
condition	probe schedule (blocked or random)
phoneme	target_phoneme
itemType	item condition (treatment or generalization)
phase	treatment phase
session	session number from Wambaugh 2017
item	item identifier
trials	number of items in the list (per phoneme)
spt2017	phase used to calcualte effect sizes in Wambaugh et al., 2017
response	accuracy of participant response
n_baselines	Number of baseline sessions

Case example: Participant 10

Filter data for Participant 10

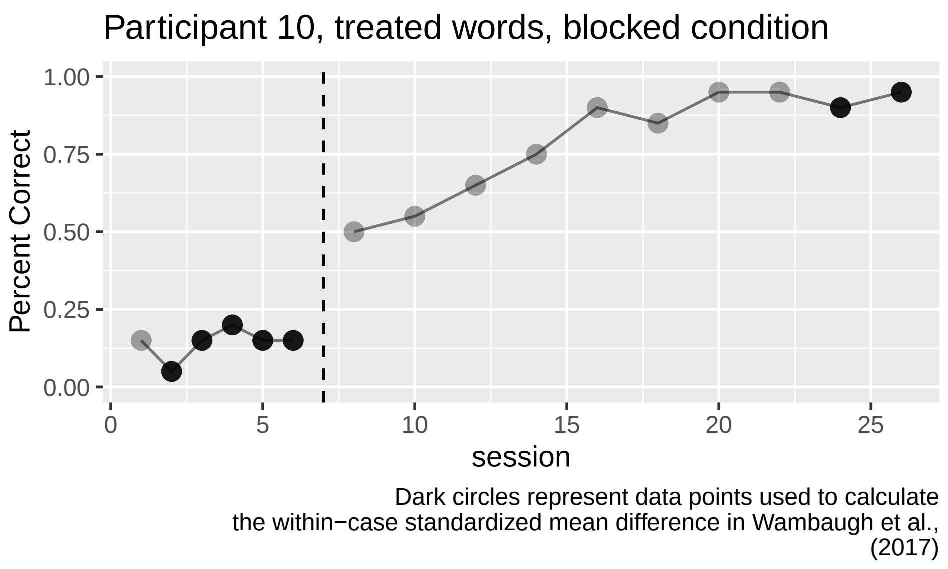
Starting from the entire dataset, filter for participant 10, treated items, and the blocked condition. Then to calculate session-level data (the number of correct responses per session), group by session, and use the summarize function to calculate the number of correct responses per session. The `group_by` function also includes phase and spt2017 because we want to keep these variables in the summary data frame, but their addition doesn't affect grouping. The .groups argument removes the grouping after summarize.

```r
P10 <- df %>%
  filter(participant == "P10",
         itemType == "tx",
         condition == "blocked") %>%
  group_by(session, phase, spt2017) %>%
  summarize(sum_correct = sum(response), .groups = "drop")
```

Plot performance over time

Plotting data from participant (also Figure 1.). First, we select only the baseline and treatment phases (ignoring the washout and maintenance phases for the purpose of this paper). The we create a dummy variable reflecting whether or not the session was included in the SMD/PMG calculations. Finally, the {ggplot2} package. A recent primer on {ggplot2} for researchers unfamilar with R can be found here: https://doi.org/10.1177/25152459221074654

```
P10 %>%
  filter(phase == "baseline" | phase == "treatment") %>%
  mutate(Measure = factor(
    ifelse(!is.na(spt2017), "include", "exclude"),
           levels = c("exclude", "include"))) %>%
  ggplot(aes(x = session, y = sum_correct/20, group = phase)) +
  geom_point(aes(alpha = Measure), size = 3) +
  geom_line(alpha = 0.5) +
  geom_vline(aes(xintercept = 7), linetype = "dashed") +
  scale_x_continuous(breaks = seq(0,30,5)) +
  ylim(0, 1) +
  scale_alpha_discrete(range = c(0.35, 0.9)) +
  labs(title = "Participant 10, treated words, blocked condition",
       caption = "Dark circles represent data points used to calculate
       the within-case standardized mean difference in Wambaugh et al.,
       (2017)",
       y="Percent Correct") +
  guides(alpha = "none")
```



Within-case standardized mean difference

There are any number of ways to calculate the within case standardized mean difference using R code. In this example, we have used the `SMD()` function from the established package {SingleCaseES} by James Pustejovsky because it includes additional functions that may be of interest to researchers in aphasiology.

Note that the bias_correct argument is set to `FALSE` to match what is typically done in aphasia research, though aphasia researchers may benefit from using the bias correction for small sample sizes as it can reduce procedural sensitivities of the within-case standardized mean difference.

Additionally, we do not show all information returned by the function, which also includes a 95% confidence interval, as it is not clear that this confidence interval applies to the the d_{BR} modification of the original within-case standardized mean difference.

```
A = P10 %>% filter(spt2017 == "pre") %>% pull(sum_correct)
B = P10 %>% filter(spt2017 == "post") %>% pull(sum_correct)

SMD(A_data = A,
    B_data = B,
    bias_correct = FALSE # not typically used in aphasia research
    )$Est
```

```
## [1] 14.33207
```

To calculate d_{BR} for all participants and conditions in the Wambaugh et al, (2017) study, we created a custom function which can be found in the R/effect-size-functions.R file.

Proportion of potential maximal gain

There is no R package that includes a function to calculate PMG to our knowledge. However, creating such a function is relatively straightforward. A function that calculates PMG similar to the SMD() function from the {SingleCaseES} package might take the following form, with an additional argument for the number of items treated (nitems). The function calculates the mean of the A phase and B phase, and then calculates and returns the PMG value from the same data as d_{BR} above.

```
# the function is named PMG and takes 3 arguments:
# vectors of the a_data and b_data, and
# a single number indicating how many items were treated
PMG <- function(a_data, b_data, nitems){
  mean_a <- mean(a_data) # calculate mean of a_data
  mean_b <- mean(b_data) # calculate mean of b_data
  pmg <- (mean_b-mean_a)/(nitems-mean_a) # calculate PMG
  return(pmg) # return the PMG value.
}

PMG(a_data = A, b_data = B, nitems = 20)
```

```
## [1] 0.9127907
```

To calculate PMG for all participants and conditions in the Wambaugh et al, (2017) study, we created a custom function which can be found in the R/effect-size-functions.R file.

Tau-U

The Tau-U family of effect sizes (and a number of other non-overlap measures) can be calculated using the {SingleCaseES} package. In this case, we use all data summarized in the P10 dataframe (and not just the data used to calculate d_{BR}).

First, we estimate the trend line during the baseline phase, which can be generated by creating a simple linear model using the `lm()` function. The model includes the number of correct responses as the dependent variable and the session number as the independent variable. The session coefficient reflects the slope during the baseline phase. The `coef()` function simple extracts the model coefficients.

```
P10 %>%
    filter(phase == "baseline") %>%
    lm(data = ., sum_correct~session) %>%
    coef()
```

```
## (Intercept)     session
##    2.133333    0.200000
```

The `Tau()` and `Tau_U()` functions take the same data structure as the `SMD()` and `PMG()` functions above. I

Using the conservative benchmark of 0.33 recommended by Lee and Cherney (2018), we would calculate Tau~A VS. B~ as the slope of the baseline phase is only 0.2. To calculate Tau-U~A VS. B~, we can use the `Tau()` function.

```
A = P10 %>% filter(phase == "baseline") %>% pull(sum_correct)
B = P10 %>% filter(phase == "treatment") %>% pull(sum_correct)

Tau(A_data = A, B_data = B)
```

```
##    ES Est         SE CI_lower CI_upper
## 1 Tau   1 0.02710291        1        1
```

However, if we had elected to correct for baseline trends and use Tau-U~A VS. B - TREND A~, we can use the similar `Tau_U()` function.

```
Tau_U(A_data = A, B_data = B)
```

```
##      ES  Est
## 1 Tau-U 0.95
```

Mixed-effects model-based effect sizes

The mixed-effects model example for participant 10 uses item-level data, so we need to create a new dataframe for this model. The model formula is based on a structure from Huitema & McKean (2000). We recommend the reader read Huitema & McKean for a clear description and justifcation for this model structure.

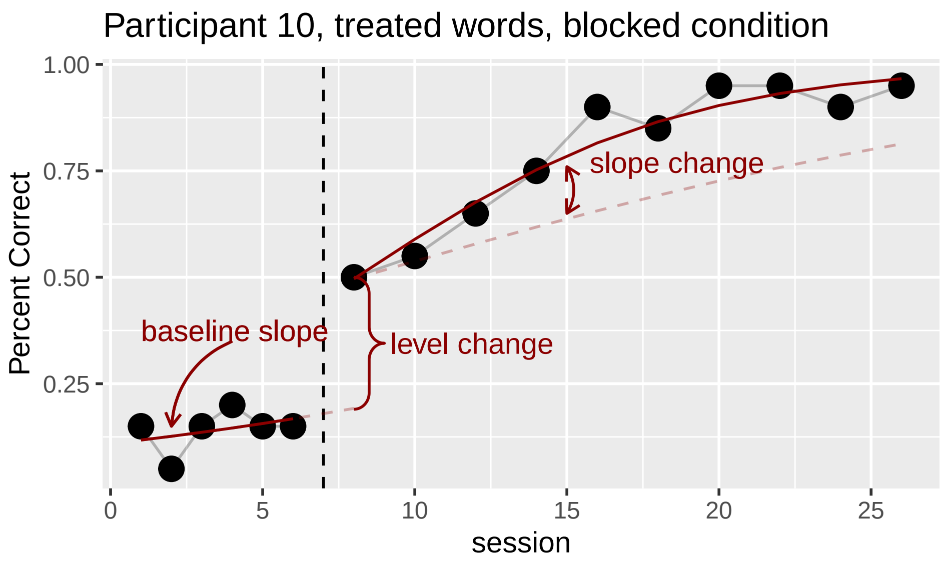
$$Y_t = \beta_0 + \beta_1 T_t + \beta_2 D_t + \beta_3[T_t - (n_1 + 1)]D_t + \epsilon_t$$

After selecting data from participant 10, the coefficients are created by:

- setting `baseline_slope` equal to the session variable
- `level_change` is a dummy variable, 0 during baseline and 1 during treatment
- `slope_change` is created by subtracting the number of baselines plus 2 from the baseline slope value, and then multiplying the result with the `level_change` variable. Typically, if probing every session, the formula calls for subtracting the number of baselines plus 1. However, because Wambaugh et al., (2017) used intermittent probing schedules, and probed every other treatment session starting at the second, we need to add 2 to the number of baselines to ensure that the slope change variable starts at 0 on the first recorded treatment probe.

```
P10 <- df %>%
  filter(participant == "P10",
         condition == "blocked",
         itemType == "tx",
         phase == "baseline" | phase == "treatment") %>%
  mutate(baseline_slope = session,
         level_change = ifelse(phase == "baseline", 0, 1),
         slope_change = (baseline_slope - (6+2))*level_change,
         level_change = as.factor(level_change))
```

Figure 2. visualizes each parameter in this model structure. The code can be found in the .Rmd file, and is omitted from the pdf due to its length.



The following shows how we arrived at the final model for P10

1. First, we fit the maximal random effects structure. However, the model did not converge.

```
mod1 <-
    glmer(response ~ baseline_slope + level_change + slope_change +
            (1 + baseline_slope + level_change + slope_change | item),
          data = P10,
          family = binomial)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.192497 (tol = 0.002, component 1)
```

2. Second, we tried specifying a different optimizer, following recommendations that can be found at https://bbolker.github.io/mixedmodels-misc/glmmFAQ.html#convergence-warnings.

Because the model structure is pre-determined, we tried a different optimizer, which we have had more success with in past studies. This removed the convergence warning.

```
mod1 <-
    glmer(response ~ baseline_slope + level_change + slope_change +
            (1 + baseline_slope + level_change + slope_change | item),
          data = P10,
          family = binomial,
          control = glmerControl(optimizer="bobyqa"))
```

We can now examine the model summary:

```
summary(mod1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: response ~ baseline_slope + level_change + slope_change + (1 +
##     baseline_slope + level_change + slope_change | item)
##    Data: P10
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##    249.5    302.3   -110.8    221.5      306
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.2995 -0.2462  0.0202  0.1563  3.6534
##
## Random effects:
##  Groups Name           Variance Std.Dev. Corr
##  item   (Intercept)    4.5354   2.1296
##         baseline_slope 0.3358   0.5795   -0.75
##         level_change1  5.6514   2.3773    0.34 -0.01
##          slope_change   0.8646   0.9298    0.66 -0.97 -0.14
## Number of obs: 320, groups:  item, 20
##
## Fixed effects:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -2.70900    1.33242  -2.033    0.042 *
## baseline_slope  0.01743    0.33249   0.052    0.958
## level_change1   2.50785    1.69431   1.480    0.139
## slope_change    0.39216    0.39651   0.989    0.323
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) bsln_s lvl_c1
## baselin_slp -0.861
## level_chng1  0.535 -0.723
## slope_chang  0.761 -0.916  0.499
```

We note that in this case, further reducing the random effects structure often returns a significant result for the level_change parameter, demonstrating how our choice of random effect structure can influence the statistical significance of model parameters.

Calculating an overall effect size for this participant requires contrasting performance either at the end of treatment with and without the level change and slope change parameters, or contrasting performance at the end of treatment with performance at the end of baseline. The former option assumes that any baseline trend would have continued throughout the treatment phase in the absence of treatment, is typically more conservative.

While there is a small, empirical baseline slope in this data, it may be reasonable to consider that this slope is largely driven by lower performance on the second probe session, and that performance in baseline sessions 3-6 are stable, and therefore estimate the difference in performance from the end of baseline to the end of treatment. Criteria for such decisions should ideally be made a-priori if possible.

1. First, we generate the marginal means for each combination of baseline slope, level change, and slope change.

```
# setup marginal means
#
marginal_means = emmeans(
                       object = mod1,
                       specs = c("baseline_slope", "level_change", "slope_change"),
                       at = list(
                         baseline_slope = c(7, 26),
                         level_change = c("0", "1"),
                         slope_change = c(0, 19)
                       )
```

```
                     )

marginal_means
```

```
##  baseline_slope level_change slope_change  emmean   SE  df asymp.LCL asymp.UCL
##               7 0                       0 -2.5870 1.36 Inf    -5.252    0.0782
##              26 0                       0 -2.2559 7.53 Inf   -17.009   12.4976
##               7 1                       0 -0.0792 1.20 Inf    -2.427    2.2684
##              26 1                       0  0.2520 6.39 Inf   -12.263   12.7672
##               7 0                      19  4.8641 6.46 Inf    -7.801   17.5290
##              26 0                      19  5.1952 3.06 Inf    -0.802   11.1928
##               7 1                      19  7.3719 7.35 Inf    -7.041   21.7846
##              26 1                      19  7.7030 2.49 Inf     2.819   12.5873
##
## Results are given on the logit (not the response) scale.
## Confidence level used: 0.95
```

2. This returns a table of all possible comparisons, and we are only interested in contrasting the first row (beginning of treatment) with the last row (end of treatment). After selecting these two rows, we can then contrast their estimates.

```
# code to select first and last rows
# The 1 indicates that the row should be selected
A = c(1, 0, 0, 0, 0, 0, 0, 0)
B = c(0, 0, 0, 0, 0, 0, 0, 1)

# contrast the marginal means
# infer argument returns a confidence interval and p value if
# both are set to TRUE.
contrast(marginal_means,
     method = list("Unadjusted effect size" = B-A),
     infer = c(TRUE, TRUE))
```

```
##  contrast               estimate  SE  df asymp.LCL asymp.UCL z.ratio p.value
##  Unadjusted effect size     10.3 3.1 Inf      4.22      16.4   3.322  0.0009
##
## Results are given on the log odds ratio (not the response) scale.
## Confidence level used: 0.95
```

We could also make the more conservative assumption that any baseline trend continues by choosing the second row where baseline slope is set to the last treatment session.

```
# code to select first and last rows
# The 1 indicates that the row should be selected
A = c(0, 1, 0, 0, 0, 0, 0, 0)
B = c(0, 0, 0, 0, 0, 0, 0, 1)
```

```
# contrast the marginal means
# infer argument returns a confidence interval and p value if
# both are set to TRUE.
contrast(marginal_means,
     method = list("Unadjusted effect size" = B-A),
     infer = c(TRUE, TRUE))
```

```
##  contrast               estimate   SE  df asymp.LCL asymp.UCL z.ratio p.value
##  Unadjusted effect size     9.96 8.51 Inf     -6.71      26.6   1.171  0.2417
##
## Results are given on the log odds ratio (not the response) scale.
## Confidence level used: 0.95
```

Notice that there is much greater uncertainty in this contrast, and as a result the p-value is no longer significant.

Group-level model. We can extend this individual model to all participants, still focusing on treated items in the blocked condition. First, we create a new dataframe that includes all participants and then repeat the model

```
df_glmm <- df %>%
  filter(phase == "baseline" | phase == "treatment",
         condition == "blocked",
         itemType == "tx") %>%
  mutate(baseline_slope = session,
         level_change = ifelse(phase == "baseline", 0, 1),
         slope_change = (baseline_slope - (n_baselines+2))*level_change,
         level_change = as.factor(level_change))
```

Then we can start again with a relatively maximal random effect structures, noting that we could also include random slopes for items. However, it is unlikely that such a model structure could be supported by the data. In this case we have chosen to include the most theoretically important random effects (Matsucheck, 2018) that we expect to be supported by the data.

The model takes a little longer to run, but returns a convergence warning

```
mod2 <-
 glmer(response ~ baseline_slope + level_change + slope_change +
       (1 + baseline_slope + level_change + slope_change | participant) +
       (1|item),
           data = df_glmm,
           family = binomial)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.0786858 (tol = 0.002, component 1)
```

Again, we change the optimizer.

```
mod2 <-
 glmer(response ~ baseline_slope + level_change + slope_change +
       (1 + baseline_slope + level_change + slope_change | participant) +
       (1|item),
           data = df_glmm,
           family = binomial,
       control = glmerControl(optimizer = "bobyqa"))
```

Since the model appears to have converged, we can examine the model results

```
summary(mod2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: response ~ baseline_slope + level_change + slope_change + (1 +
##     baseline_slope + level_change + slope_change | participant) +
##     (1 | item)
##    Data: df_glmm
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   5652.2   5755.8  -2811.1   5622.2     7385
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -8.1144 -0.3551 -0.1630  0.3241 13.5510
##
## Random effects:
##  Groups      Name           Variance Std.Dev. Corr
##  item        (Intercept)    1.611099 1.26929
##  participant (Intercept)    0.404805 0.63624
##              baseline_slope 0.002648 0.05146   0.41
##              level_change1  2.390503 1.54613   0.18  0.86
##              slope_change   0.007304 0.08546  -0.78 -0.54 -0.52
## Number of obs: 7400, groups:  item, 322; participant, 20
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -3.44259    0.20649 -16.672  < 2e-16 ***
## baseline_slope  0.07624    0.01883   4.050 5.13e-05 ***
## level_change1   0.85952    0.39754   2.162 0.030611 *
## slope_change    0.09124    0.02582   3.534 0.000409 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Correlation of Fixed Effects:
##             (Intr) bsln_s lvl_c1
## baselin_slp -0.161
## level_chng1  0.096  0.310
## slope_chang -0.192 -0.666 -0.363
```

The summary table shows us there there are statistically reliable effects for all three parameters: a small but reliable trend at baseline, a fairly substantial level change on average, and an increase in slope from baseline that is slightly more than double the initial average trend. Additionally, performance at baseline is predicted to be low, just 3%. We also note that there there is much more variation in the level change paramter between participants relative to the baseline slope and slope change parameters. Finally, the correlation of fixed effects shows a positive association between individuals baseline trend and their level change, but a negative association between individuals baseline trend and slope change and level change.

We can calculate an overall effect size using the same approach as the individual model. In this case, we assume the median number of baseline sessions (11) and treatment sessions (20)

```
# setup marginal means
#
marginal_means = emmeans(
                      object = mod2,
                      specs = c("baseline_slope", "level_change", "slope_change"),
                      at = list(
                        baseline_slope = c(11, 31),
                        level_change = c("0", "1"),
                        slope_change = c(0, 20)
                      )
                    )

marginal_means
```

```
##  baseline_slope level_change slope_change  emmean    SE  df asymp.LCL asymp.UCL
##              11 0                       0 -2.6040 0.268 Inf   -3.1289  -2.07905
##              31 0                       0 -1.0792 0.587 Inf   -2.2293   0.07091
##              11 1                       0 -1.7444 0.545 Inf   -2.8119  -0.67695
##              31 1                       0 -0.2197 0.814 Inf   -1.8145   1.37515
##              11 0                      20 -0.7791 0.393 Inf   -1.5503  -0.00791
##              31 0                      20  0.7457 0.410 Inf   -0.0587   1.54996
##              11 1                      20  0.0804 0.480 Inf   -0.8613   1.02214
##              31 1                      20  1.6052 0.581 Inf    0.4669   2.74347
##
## Results are given on the logit (not the response) scale.
## Confidence level used: 0.95
```

Because the baseline trend, on average, was statistically reliable, we calculated an overall effect size assuming that it would have continued in the absense of treatment.

```
# code to select first and last rows
# The 1 indicates that the row should be selected
A = c(0, 1, 0, 0, 0, 0, 0, 0)
B = c(0, 0, 0, 0, 0, 0, 0, 1)

# contrast the marginal means
# infer argument returns a confidence interval and p value if
# both are set to TRUE.
contrast(marginal_means,
     method = list("Unadjusted effect size" = B-A),
     infer = c(TRUE, TRUE))
```

```
##  contrast               estimate    SE  df asymp.LCL asymp.UCL z.ratio p.value
##  Unadjusted effect size     2.68 0.525 Inf      1.66      3.71   5.112  <.0001
##
## Results are given on the log odds ratio (not the response) scale.
## Confidence level used: 0.95
```

This results in a statistically reliable group effect size of 2.7 logits. Given that the group model suggests a starting place of only around 3%, this indicates a gain of about 29 percentage points on average can be attributed to the level and slope changes. we can calculate this by running `plogis(-3.44 + 2.68)-plogis(3.44)`. However, we're not aware of a straightforward method of estimating individual effect sizes and confidence intervals using the frequentist approach.

Bayesian Mixed effects models

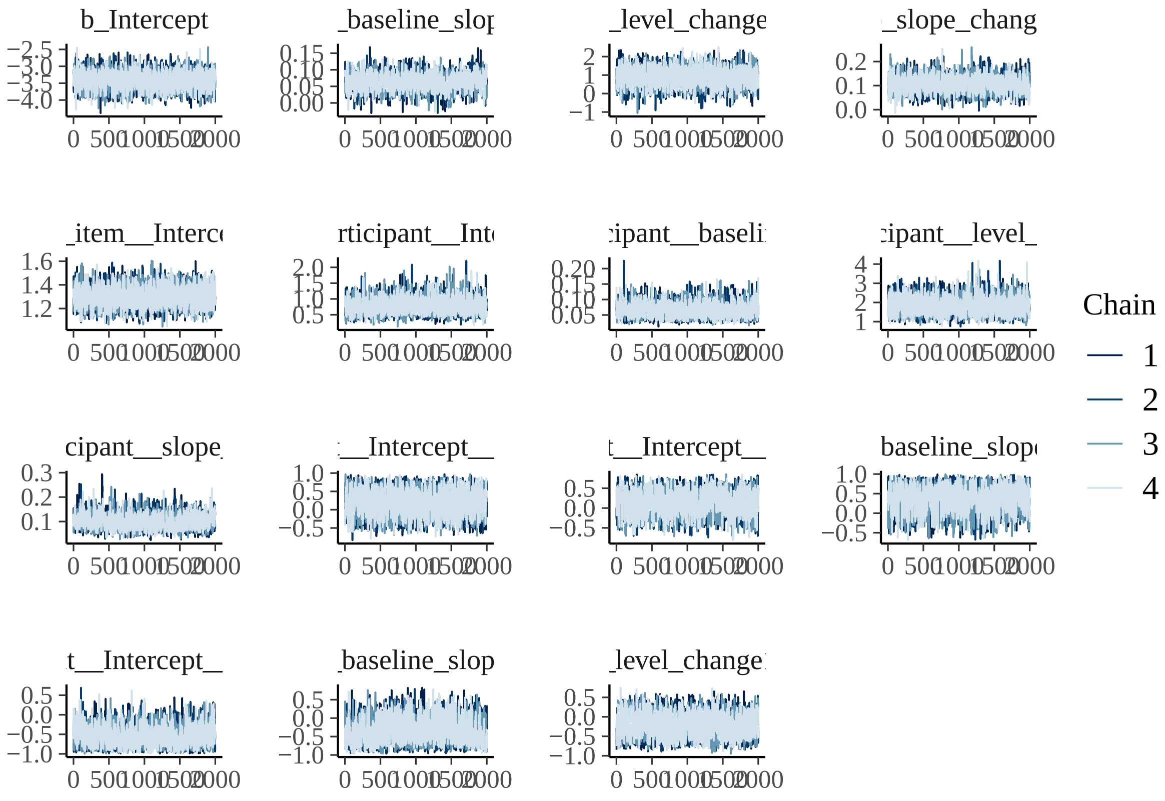
Bayesian mixed-effects models can be used in the same fashion as model 2 above to obtain both group and individual effect size estimates. First, a group-level model is estimated.

```
mod3 <-
 brm(response ~ 0 + Intercept +
     baseline_slope + level_change + slope_change +
     (1 + baseline_slope + level_change + slope_change | participant) +
     (1|item),
             data = df_glmm,
             family = bernoulli(),
             iter = 3000,
             warmup = 1000,
             chains = 4,
             seed = 42,
             prior = c(
               prior(normal(-1, 2), class = b, coef = Intercept),
               prior(normal(0, 2.5), class = b)
             ),
             # extra arguments, see rmd file
             cores = 4,
             file = "models/group_brm",
             file_refit = "on_change")
```

We can check several aspects of model fit and convergence.

1. Check that the chains have converged. They should look like "hairy catepillars" with no discernable patterns.

```
brms::mcmc_plot(mod3, type = "trace")
```



2. Check that the model can successfully re-estimate the data. ALEX PUT DESCRIPTION HERE.

```
y = mod3$data$response
yrep = posterior_predict(mod3)
mean_ppc = mean(apply(yrep, 1, mean) > mean(y))
sd_ppc = mean(apply(yrep, 1, sd) > sd(y))
kurtosis_ppc = mean(apply(yrep, 1, e1071::kurtosis) > e1071::kurtosis(y))

print(c(mean_ppc, sd_ppc, kurtosis_ppc))
```

```
## [1] 0.506625 0.506625 0.482625
```

3. Rhat statistic should be < 1.05, ideally < 1.01

```
max(rhat(mod3))
```

```
## [1] 1.003422
```

We can preview the model results using `summary()` again. Notably, the model estimates are largely similar to the frequentist model.

```
summary(mod3)
```

```
##  Family: bernoulli
##   Links: mu = logit
## Formula: response ~ 0 + Intercept + baseline_slope + level_change + slope_change + (1 + 
##    Data: df_glmm (Number of observations: 7400)
##   Draws: 4 chains, each with iter = 3000; warmup = 1000; thin = 1;
##          total post-warmup draws = 8000
##
## Group-Level Effects:
## ~item (Number of levels: 322)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     1.30      0.08     1.15     1.46 1.00     2366     4243
##
## ~participant (Number of levels: 20)
##                                   Estimate Est.Error l-95% CI u-95% CI Rhat
## sd(Intercept)                         0.77      0.23     0.40     1.28 1.00
## sd(baseline_slope)                    0.06      0.02     0.03     0.11 1.00
## sd(level_change1)                     1.80      0.40     1.12     2.70 1.00
## sd(slope_change)                      0.10      0.03     0.05     0.16 1.00
## cor(Intercept,baseline_slope)         0.21      0.33    -0.45     0.79 1.00
## cor(Intercept,level_change1)          0.15      0.28    -0.43     0.65 1.00
## cor(baseline_slope,level_change1)     0.36      0.32    -0.32     0.87 1.00
## cor(Intercept,slope_change)          -0.56      0.25    -0.93     0.02 1.00
## cor(baseline_slope,slope_change)     -0.37      0.31    -0.85     0.33 1.00
## cor(level_change1,slope_change)      -0.22      0.28    -0.72     0.35 1.00
##                                   Bulk_ESS Tail_ESS
## sd(Intercept)                         2463     3834
## sd(baseline_slope)                    2559     3731
## sd(level_change1)                     3090     4799
## sd(slope_change)                      1180     2898
## cor(Intercept,baseline_slope)         1883     3701
## cor(Intercept,level_change1)          1310     2675
## cor(baseline_slope,level_change1)      915     1814
## cor(Intercept,slope_change)           1053     2559
## cor(baseline_slope,slope_change)       954     1958
## cor(level_change1,slope_change)       1657     2921
##
## Population-Level Effects:
##                Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept         -3.41      0.24    -3.89    -2.96 1.00     2118     3857
## baseline_slope     0.06      0.02     0.02     0.11 1.00     1309     2171
## level_change1      0.90      0.45    -0.01     1.78 1.00     2587     3970
## slope_change       0.11      0.03     0.05     0.17 1.00     1582     2628
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

```
df_contrasts <- mod3$data %>%
    group_by(level_change, participant) %>%
    mutate(last_session = max(baseline_slope)) %>%
    filter(baseline_slope == last_session) %>%
    select(-response) %>%
    distinct()

es_logit = df_contrasts %>%
    add_linpred_draws(mod3) %>%
    ungroup() %>%
    mutate(timepoint = ifelse(level_change == 0, "entry", "exit")) %>%
    select(timepoint, item, .draw, .linpred, participant) %>%
    pivot_wider(names_from = "timepoint", values_from = .linpred) %>%
    mutate(ES = exit-entry) %>%
    group_by(participant) %>%
    point_interval(ES)
```

Examine the results:

```
head(es_logit, 20)
```

```
## # A tibble: 20 x 7
##    participant    ES .lower .upper .width .point .interval
##    <chr>       <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
##  1 P1           3.34   2.36   4.34   0.95 median qi
##  2 P10          6.35   5.13   7.78   0.95 median qi
##  3 P11          4.99   3.69   6.52   0.95 median qi
##  4 P12          3.73   2.90   4.58   0.95 median qi
##  5 P13          6.54   5.43   7.72   0.95 median qi
##  6 P14          4.85   3.91   5.86   0.95 median qi
##  7 P15          3.39   2.13   4.77   0.95 median qi
##  8 P16          7.22   5.27   9.73   0.95 median qi
##  9 P17          5.75   4.07   8.12   0.95 median qi
## 10 P18         1.46  0.342   2.66   0.95 median qi
## 11 P19         7.46  6.14    9.00   0.95 median qi
## 12 P2          8.31  6.65   10.3    0.95 median qi
## 13 P20         4.42  3.44    5.52   0.95 median qi
## 14 P3          6.40  5.20    7.74   0.95 median qi
## 15 P4          6.27  5.17    7.55   0.95 median qi
## 16 P5          3.61  2.35    5.03   0.95 median qi
## 17 P6          4.76  3.48    6.26   0.95 median qi
## 18 P7          2.57  1.73    3.41   0.95 median qi
## 19 P8          4.15  3.19    5.18   0.95 median qi
## 20 P9          2.29  1.43    3.21   0.95 median qi
```

```
devtools::session_info()
```

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.1.2 (2021-11-01)
##  os       macOS Big Sur 10.16
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       America/New_York
##  date     2022-05-04
##  pandoc   2.17.1.1 @ /Applications/RStudio.app/Contents/MacOS/quarto/bin/ (via rmarkdown
##
## - Packages -------------------------------------------------------------------
##  package        * version    date (UTC) lib source
##  abind            1.4-5      2016-07-21 [1] CRAN (R 4.1.0)
##  arrayhelpers     1.1-0      2020-02-04 [1] CRAN (R 4.1.0)
##  assertthat       0.2.1      2019-03-21 [1] CRAN (R 4.1.0)
##  backports        1.4.1      2021-12-13 [1] CRAN (R 4.1.0)
##  base64enc        0.1-3      2015-07-28 [1] CRAN (R 4.1.0)
##  bayesplot        1.9.0      2022-03-10 [1] CRAN (R 4.1.2)
##  bayestestR       0.11.5.1   2021-12-04 [1] https://easystats.r-universe.dev (R 4.1.2)
##  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)
##  bookdown         0.25       2022-03-16 [1] CRAN (R 4.1.2)
##  boot             1.3-28     2021-05-03 [1] CRAN (R 4.1.2)
##  bridgesampling   1.1-2      2021-04-16 [1] CRAN (R 4.1.0)
##  brio             1.1.3      2021-11-30 [1] CRAN (R 4.1.0)
##  brms           * 2.17.2     2022-04-26 [1] Github (paul-buerkner/brms@abf65ea)
##  Brobdingnag      1.2-7      2022-02-03 [1] CRAN (R 4.1.2)
##  broom            0.7.12     2022-01-28 [1] CRAN (R 4.1.2)
##  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        2.1.0     2022-04-21 [1] CRAN (R 4.1.2)
##  class            7.3-19    2021-05-03 [1] CRAN (R 4.1.2)
##  cli              3.3.0     2022-04-25 [1] CRAN (R 4.1.2)
##  coda             0.19-4    2020-09-30 [1] CRAN (R 4.1.0)
##  codetools        0.2-18    2020-11-04 [1] CRAN (R 4.1.2)
##  colorspace       2.0-3     2022-02-21 [1] CRAN (R 4.1.2)
##  colourpicker     1.1.1     2021-10-04 [1] CRAN (R 4.1.0)
##  crayon           1.5.1     2022-03-26 [1] CRAN (R 4.1.2)
##  crosstalk        1.2.0     2021-11-04 [1] CRAN (R 4.1.0)
##  curl             4.3.2     2021-06-23 [1] CRAN (R 4.1.0)
##  datawizard       0.4.0.5   2022-04-12 [1] https://easystats.r-universe.dev (R 4.1.3)
##  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             1.4.1     2022-03-06 [1] CRAN (R 4.1.2)
##  devtools         2.4.3     2021-11-30 [1] CRAN (R 4.1.0)
##  digest           0.6.29    2021-12-01 [1] CRAN (R 4.1.0)
##  distributional   0.3.0     2022-01-05 [1] CRAN (R 4.1.2)
##  dplyr          * 1.0.8     2022-02-08 [1] CRAN (R 4.1.2)
##  DT               0.22      2022-03-28 [1] CRAN (R 4.1.2)
##  dygraphs         1.1.1.6   2018-07-11 [1] CRAN (R 4.1.0)
##  e1071            1.7-9     2021-09-16 [1] CRAN (R 4.1.0)
##  effectsize       0.5.0.10  2021-12-06 [1] https://easystats.r-universe.dev (R 4.1.2)
##  ellipsis         0.3.2     2021-04-29 [1] CRAN (R 4.1.0)
##  emmeans        * 1.7.1-1   2021-11-29 [1] CRAN (R 4.1.0)
##  estimability     1.3       2018-02-11 [1] CRAN (R 4.1.0)
##  evaluate         0.15      2022-02-18 [1] CRAN (R 4.1.2)
##  fansi            1.0.3     2022-03-24 [1] CRAN (R 4.1.2)
##  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)
##  fs               1.5.2     2021-12-08 [1] CRAN (R 4.1.0)
##  gamm4            0.2-6     2020-04-03 [1] CRAN (R 4.1.0)
##  generics         0.1.2     2022-01-31 [1] CRAN (R 4.1.2)
##  ggbrace          0.1.0     2022-05-02 [1] Github (nicolash2/ggbrace@51cc034)
##  ggdist         * 3.1.1     2022-02-27 [1] CRAN (R 4.1.2)
##  ggplot2        * 3.3.5     2021-06-25 [1] CRAN (R 4.1.0)
##  ggridges         0.5.3     2021-01-08 [1] CRAN (R 4.1.0)
##  glue             1.6.2     2022-02-24 [1] CRAN (R 4.1.2)
##  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)
##  gtools           3.9.2     2021-06-06 [1] CRAN (R 4.1.0)
##  haven            2.4.3     2021-08-04 [1] CRAN (R 4.1.0)
##  here           * 1.0.1     2020-12-13 [1] CRAN (R 4.1.0)
##  highr            0.9        2021-04-16 [1] CRAN (R 4.1.0)
##  hms              1.1.1      2021-09-26 [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)
##  httpuv           1.6.5      2022-01-05 [1] CRAN (R 4.1.2)
##  httr             1.4.2      2020-07-20 [1] CRAN (R 4.1.0)
##  igraph           1.3.1      2022-04-20 [1] CRAN (R 4.1.2)
##  inline           0.3.19     2021-05-31 [1] CRAN (R 4.1.0)
##  insight          0.17.0     2022-03-29 [1] CRAN (R 4.1.2)
##  jsonlite         1.8.0      2022-02-22 [1] CRAN (R 4.1.2)
##  kableExtra     * 1.3.4      2021-02-20 [1] CRAN (R 4.1.0)
##  knitr            1.37       2021-12-16 [1] CRAN (R 4.1.0)
##  labeling         0.4.2      2020-10-20 [1] CRAN (R 4.1.0)
##  later            1.3.0      2021-08-18 [1] CRAN (R 4.1.0)
##  lattice          0.20-45    2021-09-22 [1] CRAN (R 4.1.2)
##  lifecycle        1.0.1      2021-09-24 [1] CRAN (R 4.1.0)
##  lme4           * 1.1-28     2022-02-05 [1] CRAN (R 4.1.2)
##  loo              2.5.1      2022-03-24 [1] CRAN (R 4.1.2)
##  lubridate        1.8.0      2021-10-07 [1] CRAN (R 4.1.0)
##  magick           2.7.3      2021-08-18 [1] CRAN (R 4.1.0)
##  magrittr         2.0.3      2022-03-30 [1] CRAN (R 4.1.2)
##  markdown         1.1        2019-08-07 [1] CRAN (R 4.1.0)
##  MASS             7.3-54     2021-05-03 [1] CRAN (R 4.1.2)
##  Matrix         * 1.3-4      2021-06-01 [1] CRAN (R 4.1.2)
##  matrixStats      0.62.0     2022-04-19 [1] CRAN (R 4.1.2)
##  memoise          2.0.1      2021-11-26 [1] CRAN (R 4.1.0)
##  mgcv             1.8-38     2021-10-06 [1] CRAN (R 4.1.2)
##  mime             0.12       2021-09-28 [1] CRAN (R 4.1.0)
##  miniUI           0.1.1.1    2018-05-18 [1] CRAN (R 4.1.0)
##  minqa            1.2.4      2014-10-09 [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)
##  mvtnorm          1.1-3      2021-10-08 [1] CRAN (R 4.1.0)
##  nlme             3.1-155    2022-01-13 [1] CRAN (R 4.1.2)
##  nloptr           2.0.0      2022-01-26 [1] CRAN (R 4.1.2)
##  papaja         * 0.1.0.9999 2022-03-23 [1] Github (crsh/papaja@5e28378)
##  parameters       0.15.0.1   2021-12-06 [1] https://easystats.r-universe.dev (R 4.1.2)
##  performance      0.9.0.1    2022-04-06 [1] https://easystats.r-universe.dev (R 4.1.3)
##  pillar           1.7.0      2022-02-01 [1] CRAN (R 4.1.2)
##  pkgbuild         1.3.1      2021-12-20 [1] CRAN (R 4.1.0)
##  pkgconfig        2.0.3      2019-09-22 [1] CRAN (R 4.1.0)
##  pkgload          1.2.4      2021-11-30 [1] CRAN (R 4.1.0)
##  plyr             1.8.7      2022-03-24 [1] CRAN (R 4.1.2)
##  posterior        1.2.1      2022-03-07 [1] CRAN (R 4.1.2)
##  prettyunits      1.1.1      2020-01-24 [1] CRAN (R 4.1.0)
##  processx         3.5.3     2022-03-25 [1] CRAN (R 4.1.2)
##  projpred         2.0.2     2020-10-28 [1] CRAN (R 4.1.0)
##  promises         1.2.0.1   2021-02-11 [1] CRAN (R 4.1.0)
##  proxy            0.4-26    2021-06-07 [1] CRAN (R 4.1.0)
##  ps               1.7.0     2022-04-23 [1] CRAN (R 4.1.2)
##  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)
##  Rcpp           * 1.0.8.3   2022-03-17 [1] CRAN (R 4.1.2)
##  RcppParallel     5.1.5     2022-01-05 [1] CRAN (R 4.1.2)
##  readr          * 2.1.1     2021-11-30 [1] CRAN (R 4.1.0)
##  readxl           1.3.1     2019-03-13 [1] CRAN (R 4.1.0)
##  remotes          2.4.2     2021-11-30 [1] CRAN (R 4.1.0)
##  reprex           2.0.1     2021-08-05 [1] CRAN (R 4.1.0)
##  reshape2         1.4.4     2020-04-09 [1] CRAN (R 4.1.0)
##  rlang            1.0.2     2022-03-04 [1] CRAN (R 4.1.2)
##  rmarkdown        2.13      2022-03-10 [1] CRAN (R 4.1.2)
##  rprojroot        2.0.3     2022-04-02 [1] CRAN (R 4.1.2)
##  rstan            2.26.11   2022-04-23 [1] local
##  rstantools       2.2.0     2022-04-08 [1] CRAN (R 4.1.2)
##  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           1.2.0     2022-04-13 [1] CRAN (R 4.1.2)
##  sessioninfo      1.2.2     2021-12-06 [1] CRAN (R 4.1.2)
##  shiny            1.7.1     2021-10-02 [1] CRAN (R 4.1.0)
##  shinyjs          2.1.0     2021-12-23 [1] CRAN (R 4.1.0)
##  shinystan        2.6.0     2022-03-03 [1] CRAN (R 4.1.2)
##  shinythemes      1.2.0     2021-01-25 [1] CRAN (R 4.1.0)
##  SingleCaseES   * 0.5.0     2021-09-30 [1] CRAN (R 4.1.0)
##  StanHeaders      2.26.11   2022-04-23 [1] local
##  stringi          1.7.6     2021-11-29 [1] CRAN (R 4.1.0)
##  stringr        * 1.4.0     2019-02-10 [1] CRAN (R 4.1.0)
##  svglite          2.0.0     2021-02-20 [1] CRAN (R 4.1.0)
##  svUnit           1.0.6     2021-04-19 [1] CRAN (R 4.1.0)
##  systemfonts      1.0.2     2021-05-11 [1] CRAN (R 4.1.0)
##  tensorA          0.36.2    2020-11-19 [1] CRAN (R 4.1.0)
##  testthat         3.1.3     2022-03-29 [1] CRAN (R 4.1.2)
##  threejs          0.3.3     2020-01-21 [1] CRAN (R 4.1.0)
##  tibble         * 3.1.6     2021-11-07 [1] CRAN (R 4.1.0)
##  tidybayes      * 3.0.1     2021-08-22 [1] CRAN (R 4.1.0)
##  tidyr          * 1.2.0     2022-02-01 [1] CRAN (R 4.1.2)
##  tidyselect       1.1.2     2022-02-21 [1] CRAN (R 4.1.2)
##  tidyverse      * 1.3.1     2021-04-15 [1] CRAN (R 4.1.0)
##  tinylabels     * 0.2.3     2022-02-06 [1] CRAN (R 4.1.2)
##  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)
##  V8              4.1.0     2022-02-06 [1] CRAN (R 4.1.2)
##  vctrs           0.4.1     2022-04-13 [1] CRAN (R 4.1.2)
##  viridisLite     0.4.0     2021-04-13 [1] CRAN (R 4.1.0)
##  vroom           1.5.7     2021-11-30 [1] CRAN (R 4.1.0)
##  webshot         0.5.2     2019-11-22 [1] CRAN (R 4.1.0)
##  withr           2.5.0     2022-03-03 [1] CRAN (R 4.1.2)
##  xfun            0.30      2022-03-02 [1] CRAN (R 4.1.2)
##  xml2            1.3.3     2021-11-30 [1] CRAN (R 4.1.0)
##  xtable          1.8-4     2019-04-21 [1] CRAN (R 4.1.0)
##  xts             0.12.1    2020-09-09 [1] CRAN (R 4.1.0)
##  yaml            2.3.5     2022-02-21 [1] CRAN (R 4.1.2)
##  zoo             1.8-10    2022-04-15 [1] CRAN (R 4.1.2)
##
##  [1] /Library/Frameworks/R.framework/Versions/4.1/Resources/library
##
## ------------------------------------------------------------------------------
```