Synthesising evidence across studies and planning studies

Ian Hussey

08 Mai, 2024

Contents

```r
# dependencies ----
library(tidyr)
library(dplyr)
library(forcats)
library(readr)
library(purrr)
library(ggplot2)
library(effsize)
library(janitor)
library(tibble)
#library(sn)
library(metafor)
library(parameters)
library(knitr)
library(kableExtra)
library(pwr)
library(ggstance)

# set the seed ----
# for the pseudo random number generator to make results reproducible
set.seed(123)
```

What is a meta-analysis?

Let's start with some imagined summary statistics, and convert them to Cohen's d effect sizes.

`yi` refers to the effect size, and `vi` refers to its variance (note that Standard Error = sqrt(variance)).

```r
mean_intervention <- c(0.68, 0.97, 0.40, 0.48, 0.56, 0.10, -0.10, 0.03)
mean_control      <- c(   0,    0,    0,    0,    0,    0,     0,    0)
sd_intervention   <- c(   1,    1,    1,    1,    1,    1,     1,    1)
sd_control        <- c(   1,    1,    1,    1,    1,    1,     1,    1)
n_intervention    <- c(  20,   10,  100,   37,   50,  450,    50, 1000)
n_control         <- c(  20,   10,  100,   37,   50,  450,    50, 1000)

es <- escalc(measure = "SMD",
             m1i  = mean_intervention,
             m2i  = mean_control,
             sd1i = sd_intervention,
             sd2i = sd_control,
             n1i  = n_intervention,
             n2i  = n_control)

es |>
  as_tibble() |>
  mutate_all(round_half_up, digits = 2) |>
  kable() |>
  kable_classic(full_width = FALSE)
```

yi	vi
0.67	0.11
0.93	0.22
0.40	0.02
0.47	0.06
0.56	0.04
0.10	0.00
-0.10	0.04
0.03	0.00

Fixed-effects meta-analysis

Aka Common-Effects or Equal-Effects.

The simplest form of meta-analysis - although it would not be acceptable to use anywhere these days - is simply the mean effect size.

```r
es |>
  summarize(mean_effect_size = round_half_up(mean(yi), digits = 2)) |>
  kable() |>
  kable_classic(full_width = FALSE)
```

mean_effect_size
0.38

Note that calculating the mean is equivalent to fitting an intercept only fixed-effects model (ie linear regression) with the effect sizes as the DV. The estimate of the intercept is equivalent to the mean effect size.

```
lm(yi ~ 1,
   data = es) |>
  model_parameters()
```

```
## Parameter   | Coefficient |   SE |       95% CI | t(7) |     p
## ----------------------------------------------------------------
## (Intercept) |        0.38 | 0.12 | [0.09, 0.67] | 3.09 | 0.018
```

Why is this not acceptable? Because it ignores the error associated with each effect size. A very simple meta-analysis might use weighted-mean effect sizes and weight them by the total sample size of each study:

```
weighted.mean(x = es$yi, w = n_intervention + n_control) |>
  round_half_up(digits = 2)
```

```
## [1] 0.1
```

Note that the weighted mean effect size is equivalent to an intercept only fixed-effects model (linear regression) with the effect sizes as the DV and the sample sizes as weights. The estimate of the intercept is equivalent to the weighted mean effect size.

```
lm(yi ~ 1,
   weights = n_intervention + n_control,
   data = es) |>
  model_parameters()
```

```
## Parameter   | Coefficient |   SE |        95% CI | t(7) |     p
## -----------------------------------------------------------------
## (Intercept) |        0.10 | 0.06 | [-0.04, 0.25] | 1.70 | 0.133
```

Quite early in the development of meta-analysis methods, people started to weight not by N but by inverse variance of the effect size, on the basis that things other than N can affect the precision of estimation of the effect size. This can be implemented as follows:

```
lm(yi ~ 1,
   weights = 1/vi,
   data = es) |>
  model_parameters()
```

```
## Parameter   | Coefficient |   SE |        95% CI | t(7) |     p
## -----------------------------------------------------------------
## (Intercept) |        0.10 | 0.06 | [-0.04, 0.24] | 1.70 | 0.133
```

The above linear regression produces comparable results as when you fit a 'proper' fixed-effects meta-analysis using the {metafor} package. There are some small differences in the effect size and its 95% CIs that aren't important to understand here. The 'Overall' row reports the meta-analysis results.

```
rma(yi = yi,
    vi = vi,
    method = "FE", # fixed effect model
    data = es) |>
  model_parameters()
```

```
## Meta-analysis using 'metafor'
##
## Parameter | Coefficient |   SE |        95% CI |    z |     p | Weight
## -------------------------------------------------------------------------
## Study 1   |        0.67 | 0.32 | [ 0.03, 1.30] | 2.05 | 0.040 |   9.47
## Study 2   |        0.93 | 0.47 | [ 0.01, 1.85] | 1.97 | 0.048 |   4.51
## Study 3   |        0.40 | 0.14 | [ 0.12, 0.68] | 2.79 | 0.005 |  49.03
## Study 4   |        0.47 | 0.24 | [ 0.01, 0.94] |  2.01 | 0.044 |  17.99
## Study 5   |        0.56 | 0.20 | [ 0.16, 0.96] |  2.73 | 0.006 |  24.07
## Study 6   |        0.10 | 0.07 | [-0.03, 0.23] |  1.50 | 0.134 | 224.72
## Study 7   |       -0.10 | 0.20 | [-0.49, 0.29] | -0.50 | 0.620 |  24.97
## Study 8   |        0.03 | 0.04 | [-0.06, 0.12] |  0.67 | 0.503 | 499.94
## Overall   |        0.10 | 0.03 | [ 0.03, 0.17] |  2.97 | 0.003 |
```

Random-effects meta-analysis

There is a debate about whether Fixed-Effects vs. Random-Effects models should be employed in meta-analysis. Most recommendations come down on the side of Random-Effects, sometimes people recommend reporting the results of both. Briefly: FE models have less plausible assumptions about the differences between studies, but RE models suffer from putting less weight on the sample sizes of individual large studies.

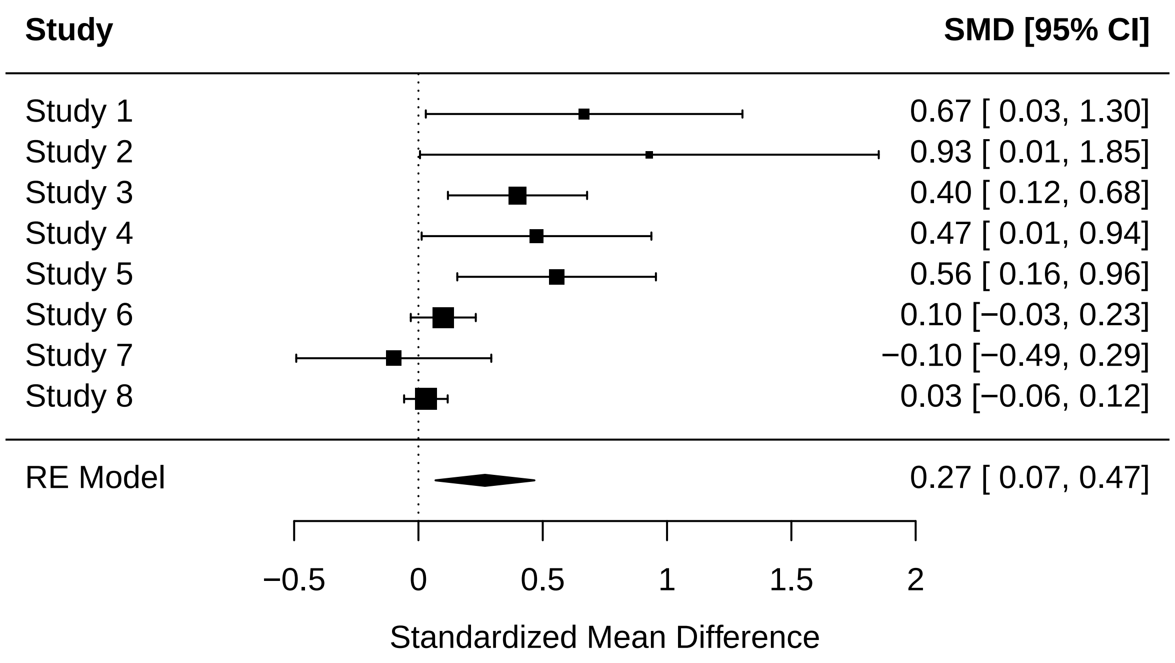
Without getting into Random-Effects models conceptually, its useful to know that Random-Effects meta-analyses can also easily be fitted in {metafor}, and indeed are the default.

```r
fit <-
  rma(yi = yi,
      vi = vi,
      method = "REML", # default random effects model
      data = es)

model_parameters(fit)
```

```
## Meta-analysis using 'metafor'
##
## Parameter | Coefficient |   SE |        95% CI |     z |     p | Weight
## -----------------------------------------------------------------------
## Study 1   |        0.67 | 0.32 | [ 0.03, 1.30] |  2.05 | 0.040 |   9.47
## Study 2   |        0.93 | 0.47 | [ 0.01, 1.85] |  1.97 | 0.048 |   4.51
## Study 3   |        0.40 | 0.14 | [ 0.12, 0.68] |  2.79 | 0.005 |  49.03
## Study 4   |        0.47 | 0.24 | [ 0.01, 0.94] |  2.01 | 0.044 |  17.99
## Study 5   |        0.56 | 0.20 | [ 0.16, 0.96] |  2.73 | 0.006 |  24.07
## Study 6   |        0.10 | 0.07 | [-0.03, 0.23] |  1.50 | 0.134 | 224.72
## Study 7   |       -0.10 | 0.20 | [-0.49, 0.29] | -0.50 | 0.620 |  24.97
## Study 8   |        0.03 | 0.04 | [-0.06, 0.12] |  0.67 | 0.503 | 499.94
## Overall   |        0.27 | 0.10 | [ 0.07, 0.47] |  2.64 | 0.008 |
```

Many meta-analyses also report forest plots, which list the studies, plot the individual and meta-analysis effect sizes and their 95% CIs, and report them numerically too for precision.

```r
forest(fit, header = TRUE)
```

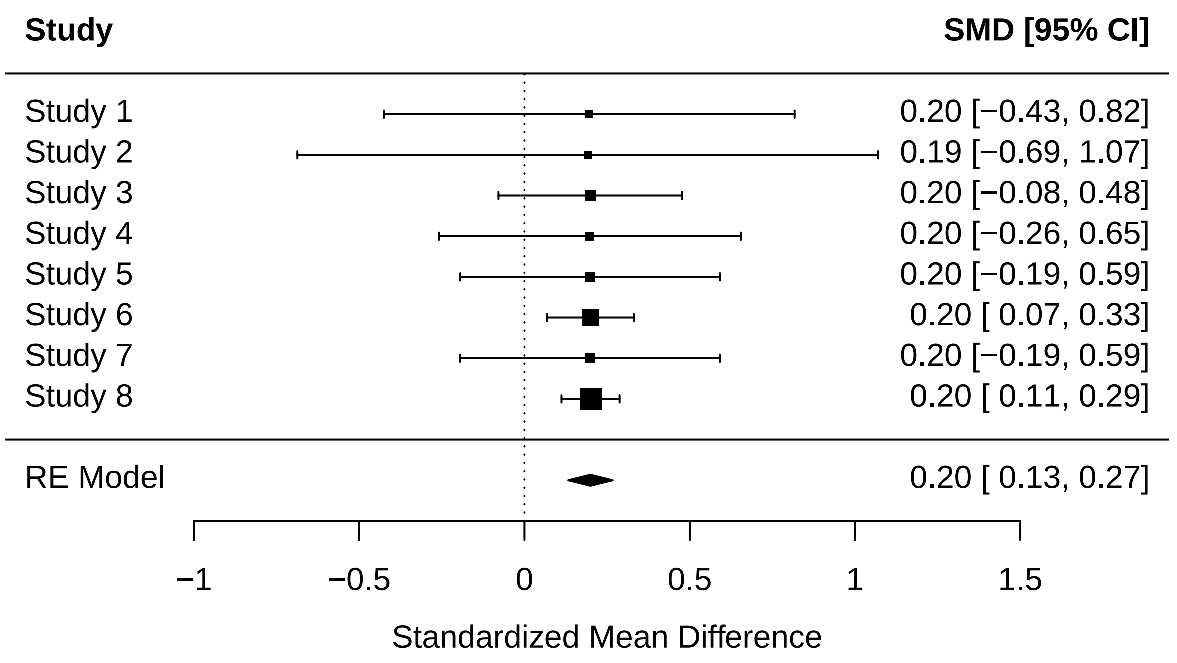


Why you can't just count the proportion of significant *p* values

Studies have different power, so each tests the hypothesis with a different probability of detecting the effect assuming its true. Mixed results can therefore be found even when all have studied the same true hypothesis, even if they happened to all find exactly the same effect size.

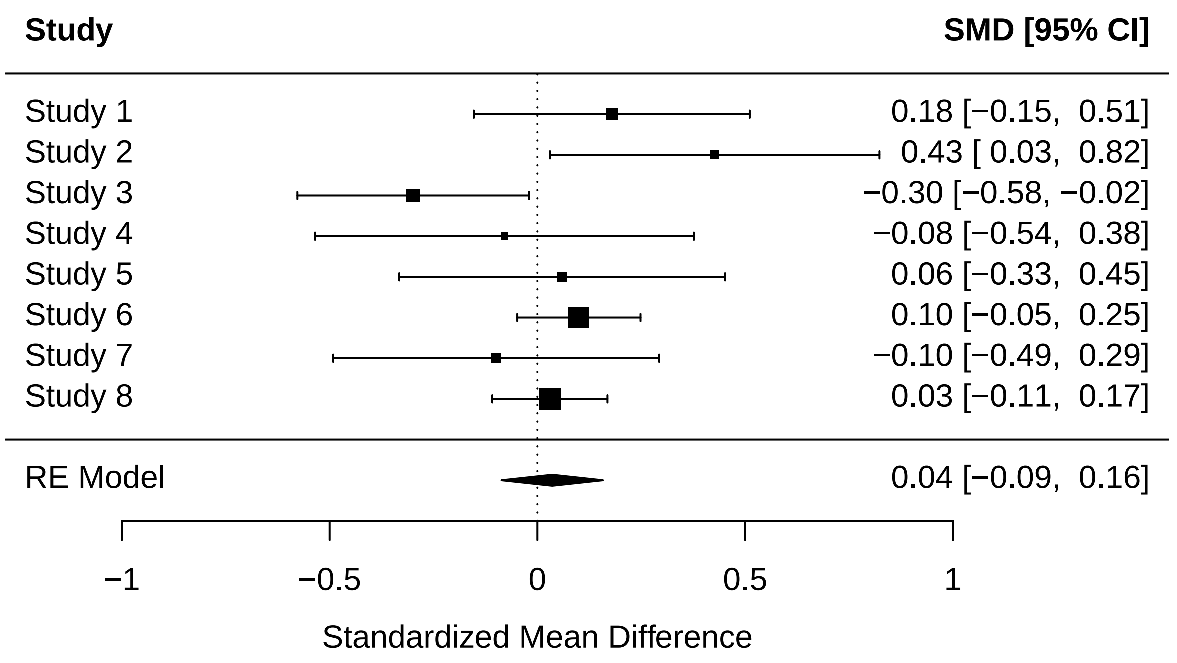
- Counting p values: Only 2/8 studies found significant results [reject H1]
- Observed effect sizes: All studies found Cohen's = 0.2 [accept H1]
- Meta-analysis: Cohen's d = 0.20, 95% CI [0.13, 0.27] [accept H1]

```r
es <- escalc(measure = "SMD",
             m1i  = c( 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2,  0.2),
             m2i  = c(   0,   0,   0,   0,   0,   0,   0,    0),
             sd1i = c(   1,   1,   1,   1,   1,   1,   1,    1),
             sd2i = c(   1,   1,   1,   1,   1,   1,   1,    1),
             n1i  = c(  20,  10, 100,  37,  50, 450,  50, 1000),
             n2i  = c(  20,  10, 100,  37,  50, 450,  50, 1000))

rma(yi     = yi,
    vi     = vi,
    data   = es,
    method = "REML") |>
  forest(header = TRUE)
```



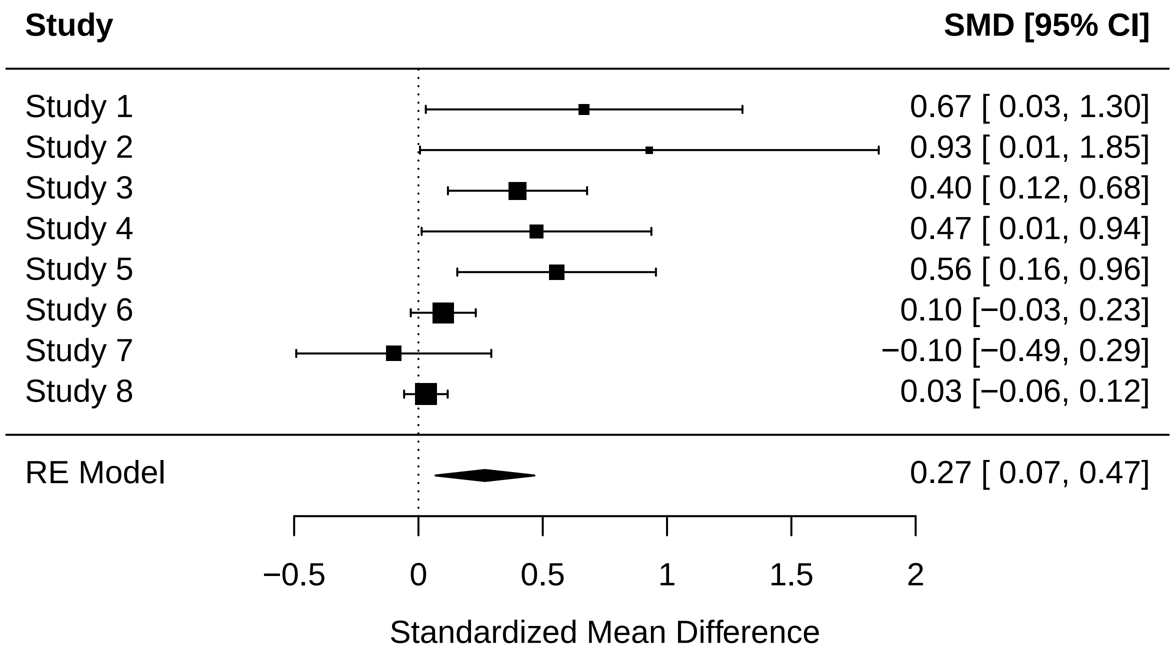
Equally, the population effect might be zero, but some studies might still detect effects. Why might this happen?

```r
es <- escalc(measure = "SMD",
             m1i  = c( 0.18, 0.43, -0.30, -0.08, 0.06, 0.10, -0.10, 0.03),
             m2i  = c(    0,    0,     0,     0,    0,    0,     0,    0),
             sd1i = c(    1,    1,     1,     1,    1,    1,     1,    1),
             sd2i = c(    1,    1,     1,     1,    1,    1,     1,    1),
             n1i  = c(   70,   50,   100,    37,   50,  350,    50,  400),
             n2i  = c(   70,   50,   100,    37,   50,  350,    50,  400))

rma(yi      = yi,
    vi      = vi,
    data    = es,
    method = "REML") |>
  forest(header = TRUE)
```



Of course, the majority of studies might produce significant results and the meta-analysis also produce a significant effect, and we might still have doubts about whether the effect really exists or not. Why might this be?

```
es <- escalc(measure = "SMD",
             m1i  = c( 0.68, 0.97, 0.40, 0.48, 0.56, 0.10, -0.10, 0.03),
             m2i  = c(    0,    0,    0,    0,    0,    0,     0,    0),
             sd1i = c(    1,    1,    1,    1,    1,    1,     1,    1),
             sd2i = c(    1,    1,    1,    1,    1,    1,     1,    1),
             n1i  = c(   20,   10,  100,   37,   50,  450,    50, 1000),
             n2i  = c(   20,   10,  100,   37,   50,  450,    50, 1000))

res <-
  rma(yi     = yi,
      vi     = vi,
      data   = es,
      method = "REML")

forest(res, header = TRUE)
```



```r
# funnel(res, level = c(90, 95, 99), refline = 0, legend = TRUE)
```

Why we can't have nice things

Confusing SD and SE when extracting summary statistics

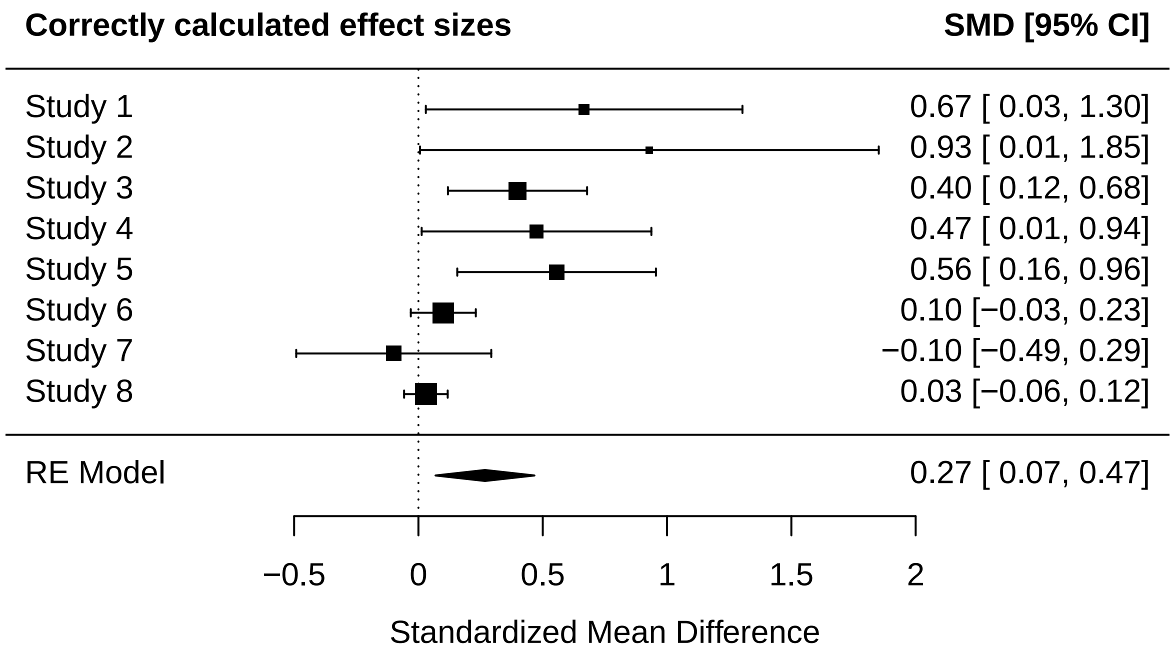
Even articles published in the most prestigious journals and on topics that will impact patient care are highly susceptible to this, e.g., Metaxa & Clarke (2024) "Efficacy of psilocybin for treating symptoms of depression: systematic review and meta-analysis" was highlighted as doing this. It's not even down to unclear labelling in the orignal study: even when they state "numerical data show means (SEM)", as in this case, its often extracted as the SD.

Because SEs are MUCH smaller than SDs, incorrectly using SE causes Cohen's d to be inflated - usually by a lot.

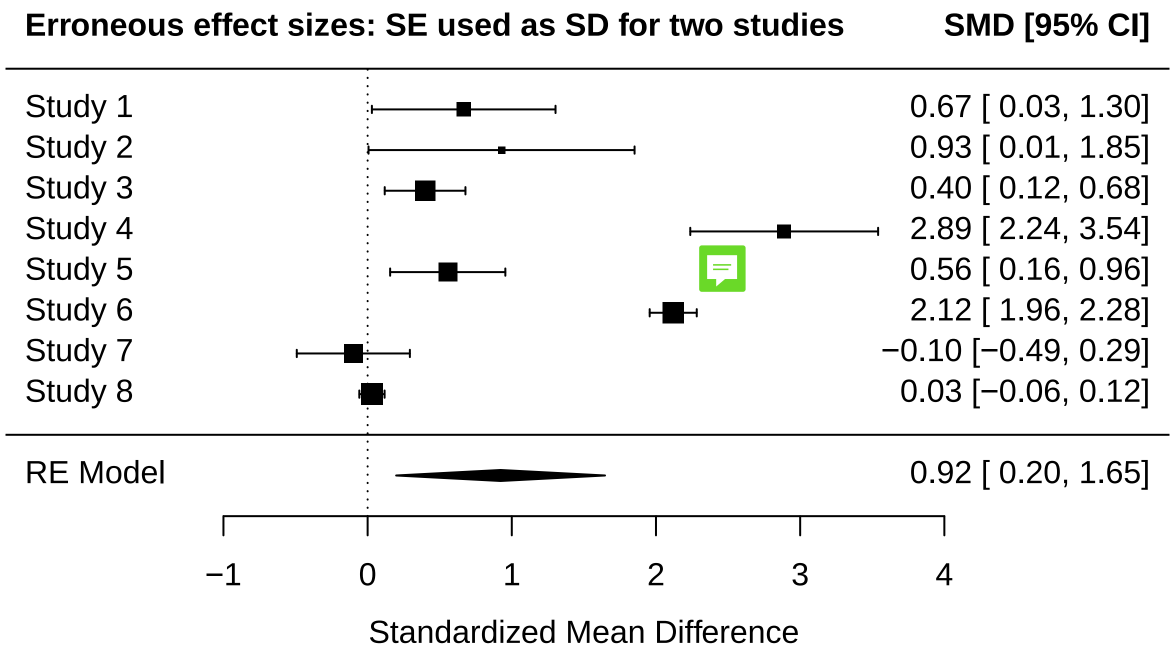
The below demonstrates this. The same summary statistics are used as in previous examples above. Effect sizes, their 95% CIs, and their SEM are then calculated. For two of the studies, the SDs are replaced with the SEs. The meta-analysis then shows this distortion on the individual effect sizes and the meta-analytic effect size.

```r
# summary stats
mean_intervention <- c(0.68, 0.97, 0.40, 0.48, 0.56, 0.10, -0.10, 0.03)
mean_control      <- c(   0,    0,    0,    0,    0,    0,     0,    0)
sd_intervention   <- c(   1,    1,    1,    1,    1,    1,     1,    1)
sd_control        <- c(   1,    1,    1,    1,    1,    1,     1,    1)
n_intervention    <- c(  20,   10,  100,   37,   50,  450,    50, 1000)
n_control         <- c(  20,   10,  100,   37,   50,  450,    50, 1000)
```

```r
dat <-
  tibble(m1i  = mean_intervention,
         m2i  = mean_control,
         sd1i = sd_intervention,
         sd2i = sd_control,
         n1i  = n_intervention,
         n2i  = n_control) |>
  rownames_to_column(var = "study") |>
  # calculate SEs
  mutate(se1i = sd1i/sqrt(n1i),
         se2i = sd2i/sqrt(n2i)) |>
  # replace SDs with SEs for two studies, studies 4 and 6
  mutate(sd1i_error = ifelse(study %in% c("4", "6"), se1i, sd1i),
         sd2i_error = ifelse(study %in% c("4", "6"), se2i, sd2i))

# calculate effect sizes properly
es_without_errors <-
  escalc(measure = "SMD",
         m1i  = dat$m1i,
         m2i  = dat$m2i,
         sd1i = dat$sd1i,
         sd2i = dat$sd2i,
         n1i  = dat$n1i,
         n2i  = dat$n2i)

# calculate effect sizes with SE/SD errors
es_with_errors <-
  escalc(measure = "SMD",
         m1i  = dat$m1i,
         m2i  = dat$m2i,
         sd1i = dat$sd1i_error,
         sd2i = dat$sd2i_error,
         n1i  = dat$n1i,
         n2i  = dat$n2i)

# meta-analyze the correctly calculated effect sizes
fit_correct <-
  rma(yi = yi,
      vi = vi,
      method = "REML",
      data = es_without_errors)

forest(fit_correct,
       header = "Correctly calculated effect sizes")
```



```r
# meta-analyze the erroneously calculated effect sizes
fit_errors <-
  rma(yi = yi,
      vi = vi,
      method = "REML",
      data = es_with_errors)

forest(fit_errors,
       header = "Erroneous effect sizes: SE used as SD for two studies")
```



Making this error for 2 of 8 studies here inflates the effect sizes for those studies to be extremely and implausibly large - Cohen's d > 2. This also greatly increases the meta-analysis effect size.

- *Note that this could be simulated more extensively as an end-of-course assignment. I.e., assuming different true effect sizes and prevalences of misinterpreting SE as SD, what is the proportionate distortion of of meta-effect sizes in the literature? For a given true effect size, what is the probability of observing a true effect size of X in a component study relative to it being a coding error? (e.g., if true effect size is 0.2, what proportion of observed effect sizes of 1.5 are erroneous?)*

Publication bias

What proportion of studies that are conducted are actually published?

Given what we know about publication bias, perhaps we should instead ask: what proportion of studies with significant results are published? And what proportion with non-significant results are published?

It is very hard to know how to interpret and synthesize the published literature without knowing this, because we don't know what is hidden from us.

Several estimates of the prevalence of significant vs non-significant results in the literature exist.

- Sterling (1959) found that 97% of psychology articles reported support for their hypothesis.
- Sterling et al. (1989) later found that this result was nearly unchanged 30 years later (95%).
- Another 20 years later, Fanelli (2010) found it was around 90% (albeit using different journals).

However, all of these estimate estimate the opposite conditional probability: the probability of significance given being published: P(significant | published).

We actually need to know the opposite, the probability of being published given significance: P(published | significant), and the probability of being published given non-significance: P(published | nonsignificant).

Worryingly, there is **very little** research on this. I only know of two studies that provide estimates of this (Franco et al. (2014; 2016):

- P(published | significant) = 57/93 = 0.61
- P(published | nonsignificant) = 11/49 = 0.22

However, registered databases of approved studies are not typical in psychology, so these values are likely to be representative of psychology as a whole.

We can also look to other fields such as medical trials. Both the EU Clinical Trials Register (EUCTR) and the US Food and Drug Administration's (FDA) ClinicalTrials.gov registries make it a **legal requirement** to publish clinical trials within 12 months of their completion. So, perhaps at least in some areas that really matter, and where there is a legal requirement to do so, null results don't sit unpublished? Unfortunately:

- Goldacre et al. (2018) found that only 50% of 7274 EU trials were published within that time frame.
- DeVito, Bacon, & Goldacre (2020) found that only 41% of 4209 US trials were published within that time frame and 64% were published ever.

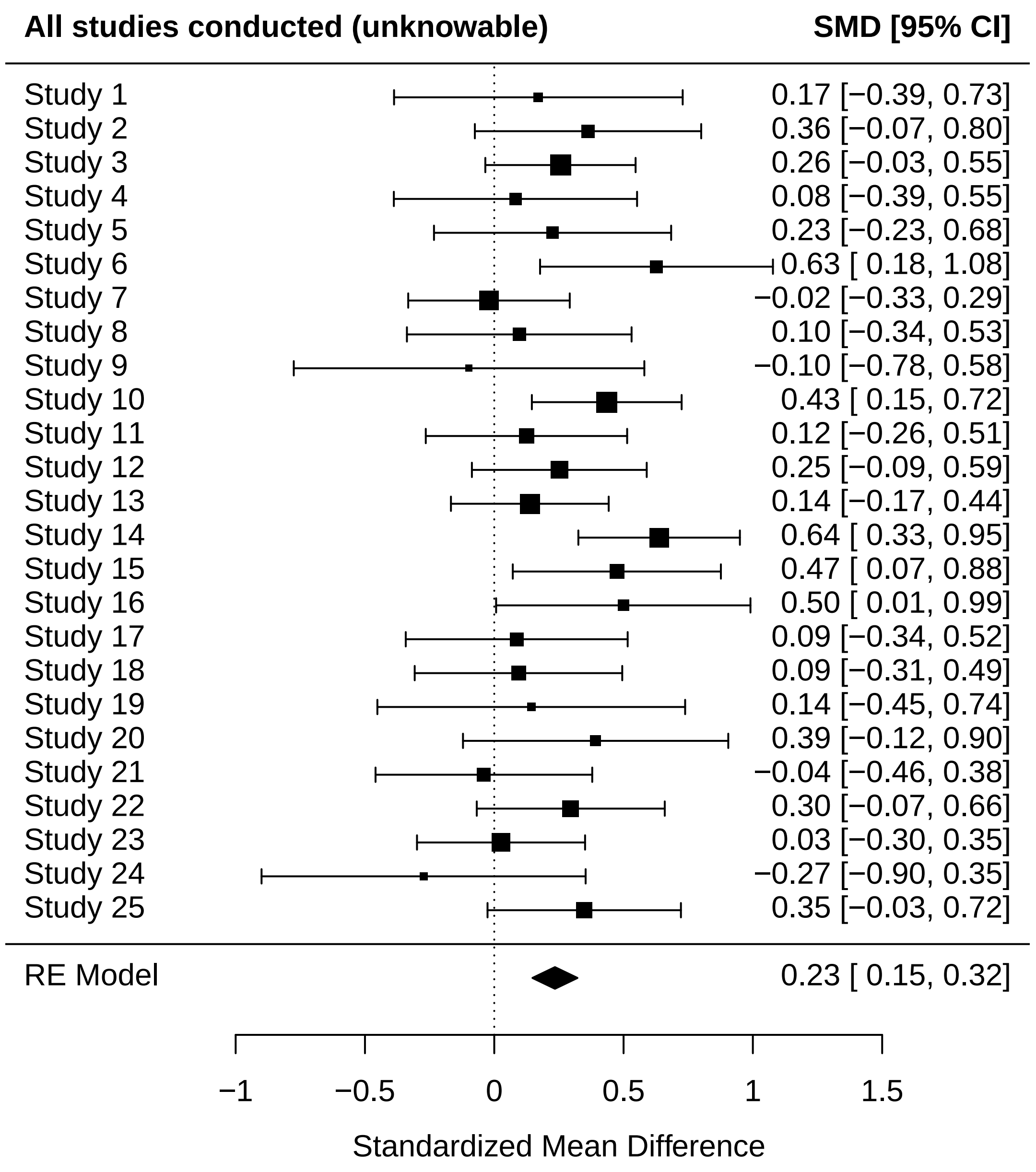
More extreme values for these conditional probabilities might therefore be more realistic for psychology. In my anecdotal experience, they are more like: P(published | significant) = 0.70 and P(published | nonsignificant) = 0.05.

Let's simulate the impact of this on rate of bias for a given literature. In this simulation, each iteration is a given study in the literature, so the number of iterations (25) is much smaller than a typical simulation.

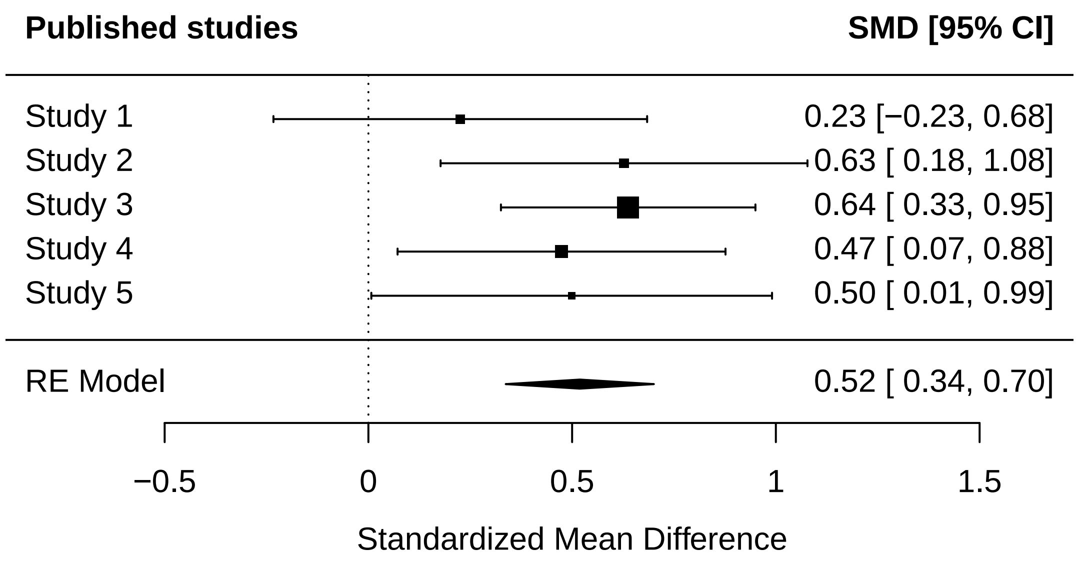
```r
# remove all objects from environment ----
#rm(list = ls())


# dependencies ----
# repeated here for the sake of completeness

library(tidyr)
library(dplyr)
library(tibble)
library(forcats)
library(purrr)
library(ggplot2)
library(knitr)
library(kableExtra)
library(janitor)
library(metafor)


# set the seed ----
# for the pseudo random number generator to make results reproducible
set.seed(46)


# define data generating function ----
generate_data <- function(n_minimum,
                          n_max,
                          mean_control,
                          mean_intervention,
                          sd_control,
                          sd_intervention) {
  require(tibble)
  require(dplyr)
  require(forcats)

  n_per_condition <- runif(n = 1, min = n_minimum, max = n_max)

  data_control <-
    tibble(condition = "control",
           score = rnorm(n = n_per_condition, mean = mean_control, sd = sd_control))

  data_intervention <-
    tibble(condition = "intervention",
           score = rnorm(n = n_per_condition, mean = mean_intervention, sd = sd_intervention))

  data <- bind_rows(data_control,
                    data_intervention) |>
    # control's factor levels must be ordered so that intervention is the first level and control is th
    # this ensures that positive cohen's d values refer to intervention > control and not the other way
    mutate(condition = fct_relevel(condition, "intervention", "control"))

  return(data)
}


# define data analysis function ----
analyse_data <- function(data, probability_sig_published, probability_nonsig_published) {
  require(effsize)
  require(tibble)

  res_n <- data |>
    count()

  res_t_test <- t.test(formula = score ~ condition,
                       data = data,
                       var.equal = FALSE,
                       alternative = "two.sided")

  res_cohens_d <- effsize::cohen.d(formula = score ~ condition,  # new addition: also fit cohen's d
                                   within = FALSE,
                                   data = data)

  res <- tibble(total_n = res_n$n,
                p = res_t_test$p.value,
                cohens_d_estimate = res_cohens_d$estimate,  # new addition: save cohen's d and its 95% 
                cohens_d_ci_lower = res_cohens_d$conf.int["lower"],
                cohens_d_ci_upper = res_cohens_d$conf.int["upper"]) |>
    mutate(cohens_d_se = (cohens_d_ci_upper - cohens_d_ci_lower)/(1.96*2),
           cohens_d_variance = cohens_d_se^2) |> # variance of effect size = its standard error squared
    mutate(
      # define result as (non)significant
      significant = p < .05,
      # generate a random luck probability between 0 and 1
      luck = runif(n = 1, min = 0, max = 1),
      # decide if the result is published or not based on whether:
```

```
      # (a) the result was significant and the luck variable is higher than the probability of signific
      # (b) the result was nonsignificant and the luck variable is higher than the probability of nonsi
      published = ifelse((significant & luck >= (1 - probability_sig_published)) |
                           (!significant & luck >= (1 - probability_nonsig_published)), TRUE, FALSE)
    )

  return(res)
}


# define experiment parameters ----
experiment_parameters_grid <- expand_grid(
  n_minimum = 10,
  n_maximum = 100,
  mean_control = 0,
  mean_intervention = 0.25,
  sd_control = 1,
  sd_intervention = 1,
  probability_sig_published = 0.70, # 0.61 from Franco et al 2014, 2016
  probability_nonsig_published = 0.05, # 0.22 from Franco et al 2014, 2016
  iteration = 1:25 # here iterations are studies, so the number is small relative to a normal simulatio
)


# run simulation ----
simulation <-
  # using the experiment parameters
  experiment_parameters_grid |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data = pmap(list(n_minimum,
                                    n_maximum,
                                    mean_control,
                                    mean_intervention,
                                    sd_control,
                                    sd_intervention),
                               generate_data)) |>

  # apply the analysis function to the generated data using the parameters relevant to analysis
  mutate(analysis_results = pmap(list(generated_data,
                                      probability_sig_published,
                                      probability_nonsig_published),
                                 analyse_data))


# summarise simulation results over the iterations ----
simulation_unnested <- simulation |>
  unnest(analysis_results)
```

```
# meta analysis and forest plot
fit_all <-
  rma(yi     = cohens_d_estimate,
    vi     = cohens_d_variance,
    data   = simulation_unnested,
    method = "REML")

forest(fit_all, header = c("All studies conducted (unknowable)", "SMD [95% CI]"), xlab = "Standardized M
```



```
fit_published <-
  rma(yi      = cohens_d_estimate,
      vi      = cohens_d_variance,
      data    = simulation_unnested |> filter(published == TRUE),
method = "REML")

forest(fit_published, header = c("Published studies", "SMD [95% CI]"), xlab = "Standardized Mean Differe
```



Note that the non-overlap between the confidence intervals between the two meta-analyses imply that the published literature has a significantly higher effect size than the actual studies run.

Remember, however, that (a) our values for the prior probability that (non)significant results are published are not based on any good evidence, and (b) that this only simulates a single literature. These results try to illustrate a point, they don't comprehensively simulate the potential impact of publication bias across a range of conditions.

- *Note that this could be simulated more extensively as an end-of-course assignment. E.g., assuming different prior probabilities and true effect sizes, what is the proportionate distortion of of meta-effect sizes in the literature?*

Alternatives to Cohen's d

How else might we get an intuition for these differences, or the efficacy of treatments generally? Cohen's d is not intuitive, it just has commonly repeated cut-offs for small/medium/large. Non-parametric alternatives do exist which are (a) more robust and (b) have more intuitive interpretations. Confusingly, one of them goes by several names in the literature: Ruscio's A aka the Common Language Effect Size aka the Probability of Superiority aka Area Under the Curve (AUC), and indeed others. This refers to the probability that a randomly selected individual in the intervention condition has a higher score than a randomly selected participant in the control condition. When the control condition functions as the counterfactual of "what would have happened if the intervention had not been given", Ruscio's A can be interpreted as the probability of improvement due to treatment.

The math for approximating Ruscio's A from Cohen's d looks scarier than the implementation:

$A = \Theta\left(\frac{d}{\sqrt{2}}\right)$, where Θ is the cumulative distribution function (CDF) of the standard normal distribution, and d is Cohen's d. This is implemented in R as simply "pnorm(d / sqrt(2)".

```r
# convert Cohen's d to Ruscio's A / Common Language Effect Size / Probability of Superiority
d_to_A <- function(d) {
  A <- pnorm(d / sqrt(2))
  return(A)
}

rma_d_to_A <- function(fit){
  require(janitor)
  paste0("Ruscio's A = ", round_half_up(d_to_A(fit$b), 2),
       ", 95% CI [", round_half_up(d_to_A(fit$ci.lb), 2),
       ", ", round_half_up(d_to_A(fit$ci.ub), 2),
       "]")
}
```

If we had access to all the studies actually conducted - which we don't - we would know that a randomly selected individual in the intervention group will do better than a randomly selected in the control group 57% of the time (Ruscio's A = 0.57, 95% CI [0.54, 0.59]).

If we meta-analyzed just the published literature, and this literature did not suffer from any other form of bias (eg p-hacking, extraction mistakes) - which it probably would - this would suggest that a randomly selected individual in the intervention group will do better than a randomly selected in the control group 64% of the time (Ruscio's A = 0.64, 95% CI [0.59, 0.69]).

So, the published literature would lead you to believe that individuals are twice as likely to improve than they really are (7 percentage points above chance vs. 14 percentage points above chance).

- *Note that the robustness of Ruscio's A relative to Cohen's d to violations of d's assumptions could be simulated more extensively as an end-of-course assignment. E.g., for different true effect sizes, and under different degrees of skew, unbalanced designs, and heterogeneity of variances, how relatively biased and variant are A vs. d?*

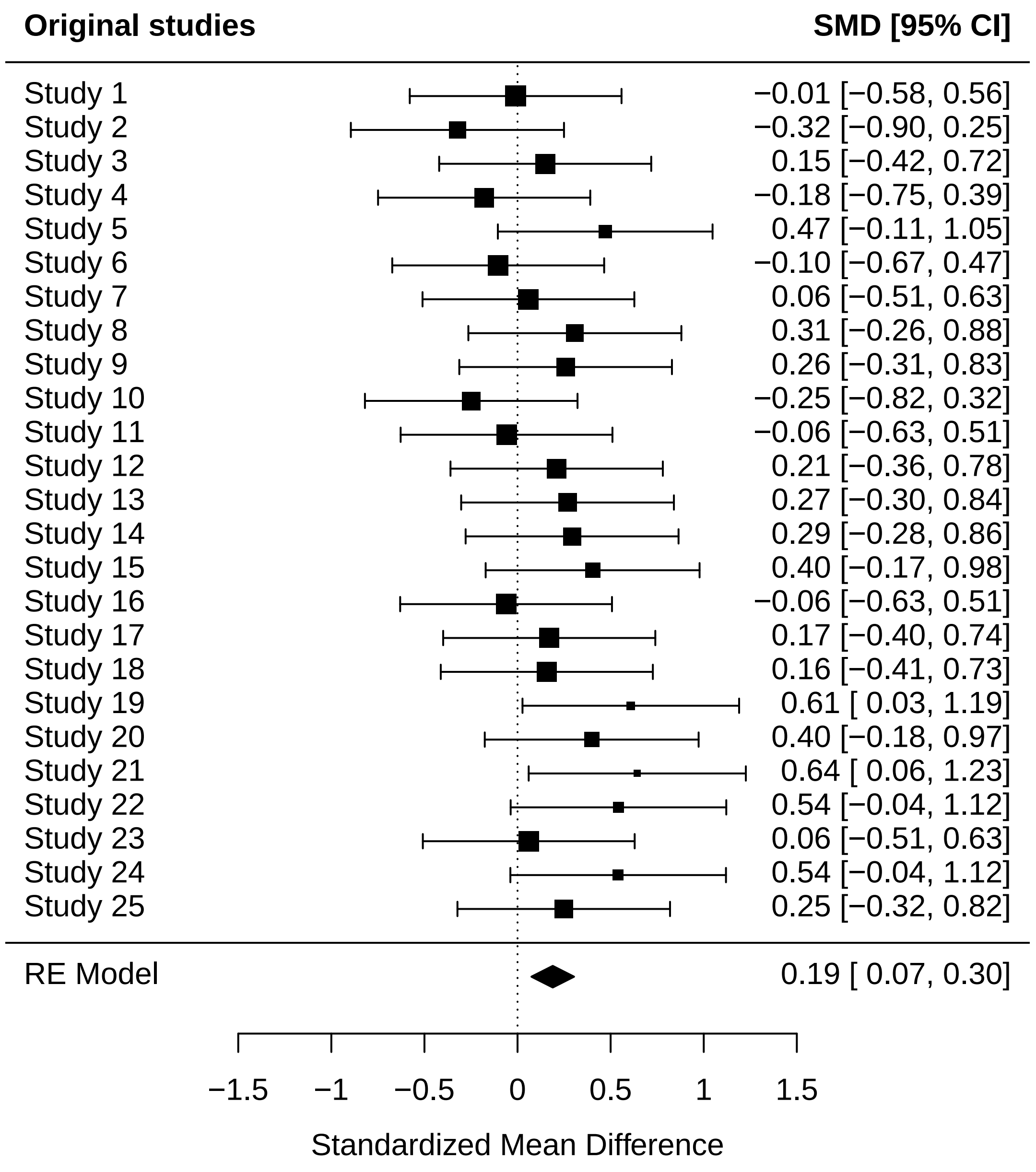
Small pilot/original studies don't help sample size planning

Scenario 1: 50 participants in the original studies

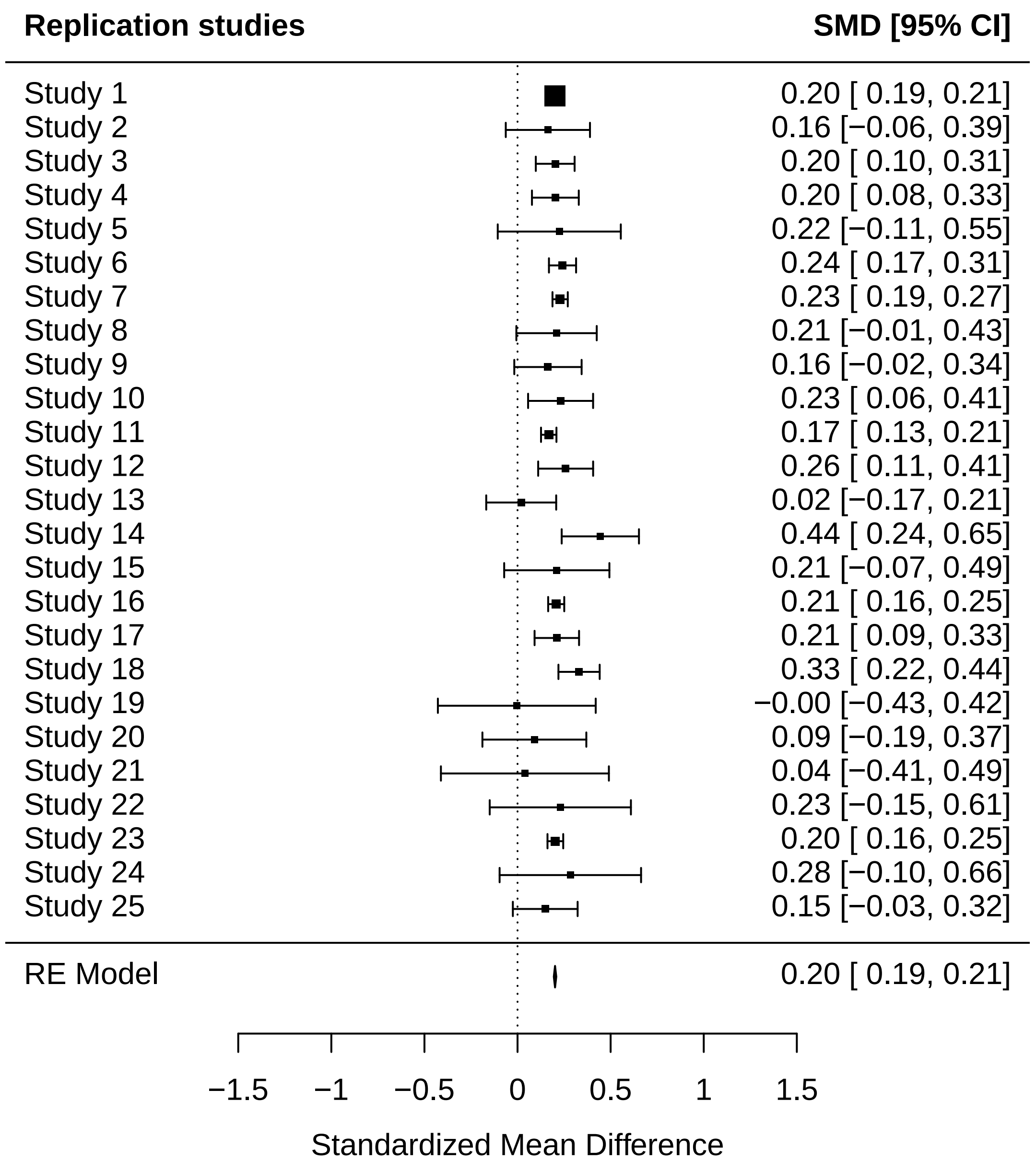
```r
# remove all objects from environment ----
#rm(list = ls())


# dependencies ----
# repeated here for the sake of completeness

library(tidyr)
library(dplyr)
library(tibble)
library(forcats)
library(purrr)
library(ggplot2)
library(knitr)
library(kableExtra)
library(janitor)
library(metafor)
```

```r
# set the seed ----
# for the pseudo random number generator to make results reproducible
set.seed(46)


# define data generating function ----
generate_data <- function(n_per_condition,
                          mean_control,
                          mean_intervention,
                          sd_control,
                          sd_intervention) {
  require(tibble)
  require(dplyr)
  require(forcats)

  data_control <-
    tibble(condition = "control",
           score = rnorm(n = n_per_condition, mean = mean_control, sd = sd_control))

  data_intervention <-
    tibble(condition = "intervention",
           score = rnorm(n = n_per_condition, mean = mean_intervention, sd = sd_intervention))

  data <- bind_rows(data_control,
                    data_intervention) |>
    # control's factor levels must be ordered so that intervention is the first level and control is th
    # this ensures that positive cohen's d values refer to intervention > control and not the other way
    mutate(condition = fct_relevel(condition, "intervention", "control"))

  return(data)
}


# define data analysis function ----
analyse_data <- function(data) {
  require(effsize)
  require(tibble)

  res_n <- data |>
    count()

  res_t_test <- t.test(formula = score ~ condition,
                       data = data,
                       var.equal = FALSE,
                       alternative = "two.sided")

  res_cohens_d <- effsize::cohen.d(formula = score ~ condition,  # new addition: also fit cohen's d
                                   within = FALSE,
                                   data = data)

  res <- tibble(total_n = res_n$n,
                p = res_t_test$p.value,
                cohens_d_estimate = res_cohens_d$estimate,  # new addition: save cohen's d and its 95% 
                cohens_d_ci_lower = res_cohens_d$conf.int["lower"],
                cohens_d_ci_upper = res_cohens_d$conf.int["upper"]) |>
    mutate(cohens_d_se = (cohens_d_ci_upper - cohens_d_ci_lower)/(1.96*2),
           cohens_d_variance = cohens_d_se^2)  # variance of effect size = its standard error squared

  return(res)
}


sample_size_planning <- function(results){
  require(pwr)
  fit_power <-
    pwr.t.test(d = results$cohens_d_estimate,
               power = .80, # .95
               sig.level = 0.05,
               type = "two.sample",
               alternative = "two.sided")

  res <- tibble(n_per_condition_replication = ceiling(fit_power$n))

  return(res)
}


# define experiment parameters ----
experiment_parameters_grid <- expand_grid(
  n_per_condition = 25,
  mean_control = 0,
  mean_intervention = 0.2,
  sd_control = 1,
  sd_intervention = 1,
  iteration = 1:25
)


# run simulation ----
simulation <-
  # using the experiment parameters
  experiment_parameters_grid |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data_original = pmap(list(n_per_condition,
                                             mean_control,
                                             mean_intervention,
                                             sd_control,
                                             sd_intervention),
                                        generate_data)) |>

  # apply the analysis function to the generated data using the parameters relevant to analysis
  mutate(analysis_results_original = pmap(list(generated_data_original),
                                          analyse_data)) |>
```

```r
  mutate(parameters_replication = map(analysis_results_original, sample_size_planning)) |>
  unnest(parameters_replication) |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data_replication = pmap(list(n_per_condition_replication,
                                               mean_control,
                                               mean_intervention,
                                               sd_control,
                                               sd_intervention),
                                          generate_data)) |>

  # apply the analysis function to the generated data using the parameters relevant to analysis
  mutate(analysis_results_replication = pmap(list(generated_data_replication),
                                            analyse_data))

simulation_summary <- simulation |>
  unnest(analysis_results_original) |>
  rename(total_n_original = total_n,
         p_original = p,
         cohens_d_estimate_original = cohens_d_estimate,
         cohens_d_ci_lower_original = cohens_d_ci_lower,
         cohens_d_ci_upper_original = cohens_d_ci_upper,
         cohens_d_se_original = cohens_d_se,
         cohens_d_variance_original = cohens_d_variance) |>

  unnest(analysis_results_replication) |>
  rename(total_n_replication = total_n,
         p_replication = p,
         cohens_d_estimate_replication = cohens_d_estimate,
         cohens_d_ci_lower_replication = cohens_d_ci_lower,
         cohens_d_ci_upper_replication = cohens_d_ci_upper,
         cohens_d_se_replication = cohens_d_se,
         cohens_d_variance_replication = cohens_d_variance)

rma(yi     = cohens_d_estimate_original,
    vi     = cohens_d_variance_original,
    data   = simulation_summary,
    method = "REML") |>
  forest(header = c("Original studies", "SMD [95% CI]"),
         xlab = "Standardized Mean Difference",
         xlim = c(-2.75, +2.75),
         at = c(-1.5, -1, -0.5, 0, 0.5, 1, 1.5))
```



```r
rma(yi     = cohens_d_estimate_replication,
    vi     = cohens_d_variance_replication,
    data   = simulation_summary,
    method = "REML") |>
  forest(header = c("Replication studies", "SMD [95% CI]"),
         xlab = "Standardized Mean Difference",
         xlim = c(-2.75, +2.75),
         at = c(-1.5, -1, -0.5, 0, 0.5, 1, 1.5))
```



```r
simulation_summary |>
  select(total_n_original, total_n_replication) |>
  kable() |>
  kable_classic(full_width = FALSE)
```

total_n_original	total_n_replication
50	304986

50	304
50	1426
50	980
50	144
50	2916
50	9186
50	334
50	474
50	510
50	8984
50	714
50	438
50	368
50	196
50	8288
50	1086
50	1272
50	88
50	200
50	78
50	110
50	8656
50	110
50	512

```r
simulation_summary |>
  summarise(min_n_replications = min(total_n_replication),
            max_n_replications = max(total_n_replication),
            median_n_replications = median(total_n_replication),
            total_n_across_replications = sum(total_n_replication),
            total_n_across_originals_and_replications = sum(total_n_replication) + sum(total_n_original
  pivot_longer(cols = everything()) |>
  kable() |>
  kable_classic(full_width = FALSE)
```

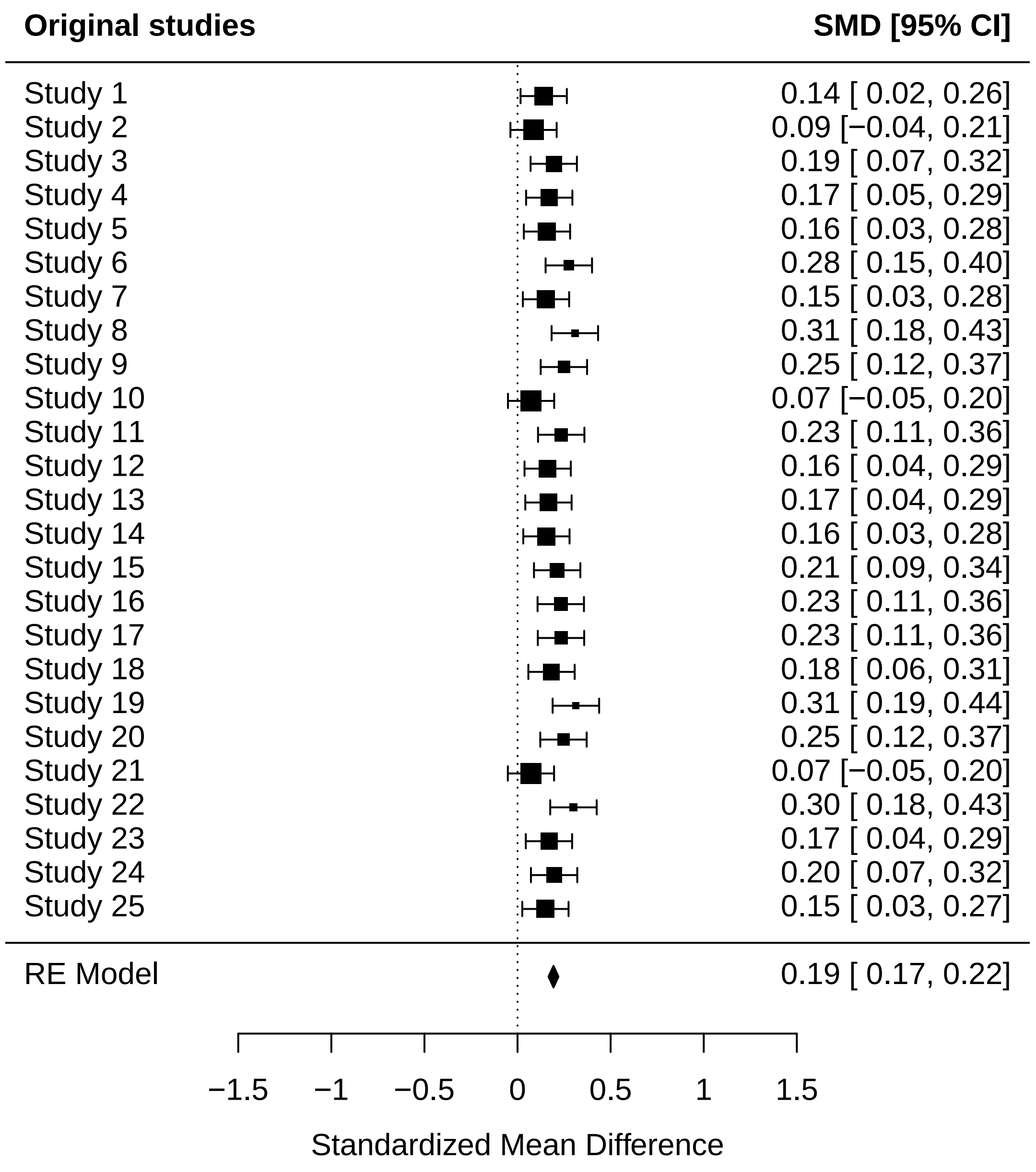
name	value
min_n_replications	78
max_n_replications	304986
median_n_replications	510
total_n_across_replications	352360
total_n_across_originals_and_replications	353610

Scenario 2: 500 participants in the original studies

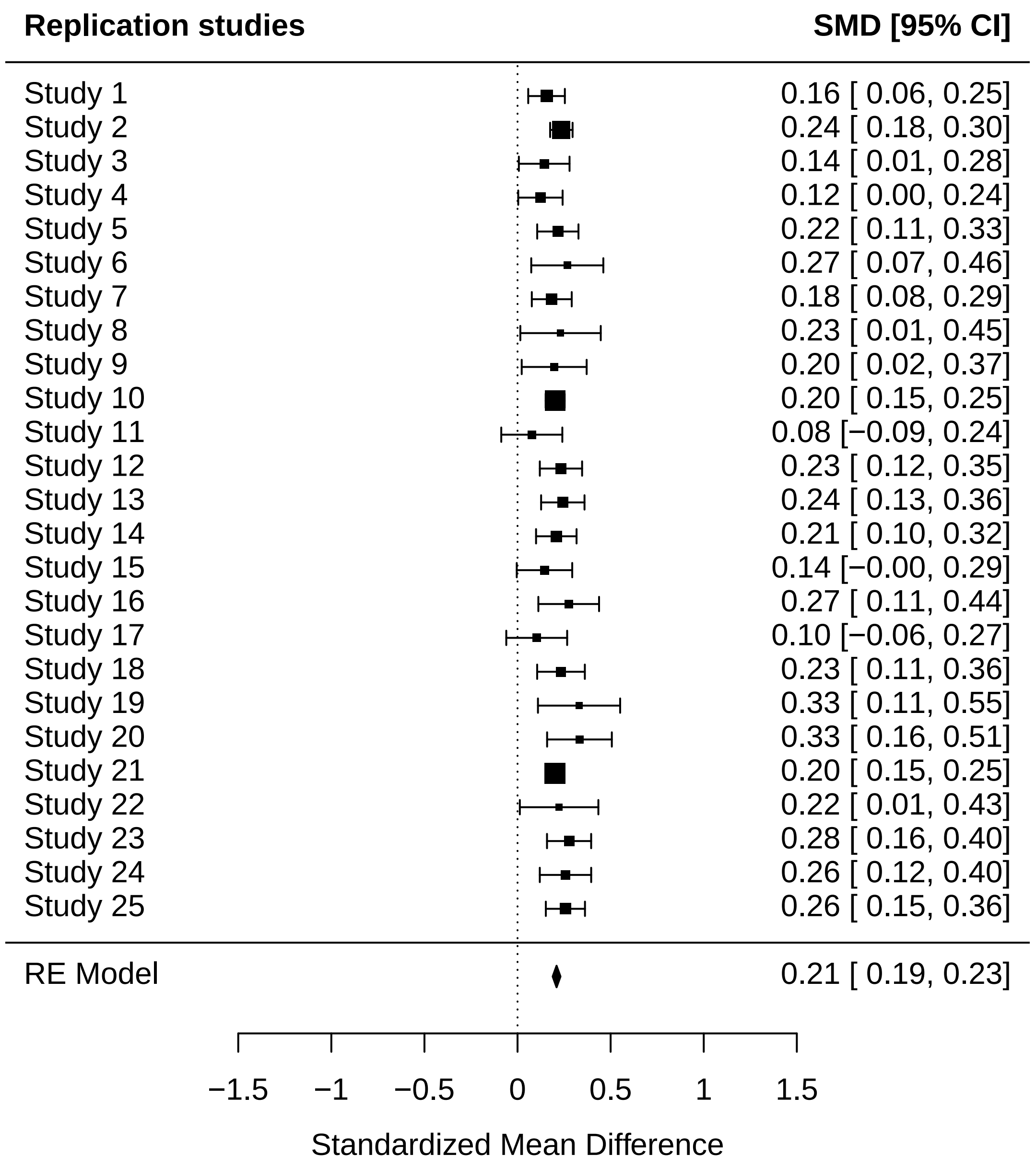
```r
pwr.t.test(d = NULL,
           n = 500, # n per condition
           power = .80, # .95
           sig.level = 0.05,
           type = "two.sample",
           alternative = "two.sided")
```

```
##
##      Two-sample t test power calculation
##
##               n = 500
##               d = 0.1773575
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

```r
set.seed(46)

# define experiment parameters ----
experiment_parameters_grid <- expand_grid(
  n_per_condition = 500,
  mean_control = 0,
  mean_intervention = 0.2,
  sd_control = 1,
  sd_intervention = 1,
  iteration = 1:25
)


# run simulation ----
simulation <-
  # using the experiment parameters
  experiment_parameters_grid |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data_original = pmap(list(n_per_condition,
                                             mean_control,
                                             mean_intervention,
                                             sd_control,
                                             sd_intervention),
                                        generate_data)) |>

  # apply the analysis function to the generated data using the parameters relevant to analysis
  mutate(analysis_results_original = pmap(list(generated_data_original),
                                          analyse_data)) |>

  mutate(parameters_replication = map(analysis_results_original, sample_size_planning)) |>
  unnest(parameters_replication) |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data_replication = pmap(list(n_per_condition_replication,
                                                mean_control,
                                                mean_intervention,
                                                sd_control,
                                                sd_intervention),
                                           generate_data)) |>

  # apply the analysis function to the generated data using the parameters relevant to analysis
  mutate(analysis_results_replication = pmap(list(generated_data_replication),
                                                  analyse_data))

simulation_summary <- simulation |>
  unnest(analysis_results_original) |>
  rename(total_n_original = total_n,
         p_original = p,
         cohens_d_estimate_original = cohens_d_estimate,
         cohens_d_ci_lower_original = cohens_d_ci_lower,
         cohens_d_ci_upper_original = cohens_d_ci_upper,
         cohens_d_se_original = cohens_d_se,
         cohens_d_variance_original = cohens_d_variance) |>

  unnest(analysis_results_replication) |>
  rename(total_n_replication = total_n,
         p_replication = p,
         cohens_d_estimate_replication = cohens_d_estimate,
         cohens_d_ci_lower_replication = cohens_d_ci_lower,
         cohens_d_ci_upper_replication = cohens_d_ci_upper,
         cohens_d_se_replication = cohens_d_se,
         cohens_d_variance_replication = cohens_d_variance)

rma(yi     = cohens_d_estimate_original,
    vi     = cohens_d_variance_original,
    data   = simulation_summary,
    method = "REML") |>
  forest(header = c("Original studies", "SMD [95% CI]"),
         xlab = "Standardized Mean Difference",
         xlim = c(-2.75, +2.75),
         at = c(-1.5, -1, -0.5, 0, 0.5, 1, 1.5))
```



```r
rma(yi     = cohens_d_estimate_replication,
    vi     = cohens_d_variance_replication,
    data   = simulation_summary,
    method = "REML") |>
  forest(header = c("Replication studies", "SMD [95% CI]"),
         xlab = "Standardized Mean Difference",
         xlim = c(-2.75, +2.75),
         at = c(-1.5, -1, -0.5, 0, 0.5, 1, 1.5))
```



```r
simulation_summary |>
  select(total_n_original, total_n_replication) |>
  kable() |>
  kable_classic(full_width = FALSE)
```

total_n_original	total_n_replication
1000	1594

1000	4258
1000	834
1000	1086
1000	1260
1000	416
1000	1346
1000	334
1000	510
1000	5934
1000	574
1000	1202
1000	1142
1000	1308
1000	696
1000	586
1000	578
1000	946
1000	322
1000	518
1000	6042
1000	350
1000	1106
1000	816
1000	1404

```r
simulation_summary |>
  summarise(min_n_replications = min(total_n_replication),
            max_n_replications = max(total_n_replication),
            median_n_replications = median(total_n_replication),
            total_n_across_replications = sum(total_n_replication),
            total_n_across_originals_and_replications = sum(total_n_replication) + sum(total_n_original
  pivot_longer(cols = everything()) |>
  kable() |>
  kable_classic(full_width = FALSE)
```

name	value
min_n_replications	322
max_n_replications	6042
median_n_replications	946
total_n_across_replications	35162
total_n_across_originals_and_replications	60162

Most psychology studies are statstically unfalsifiable

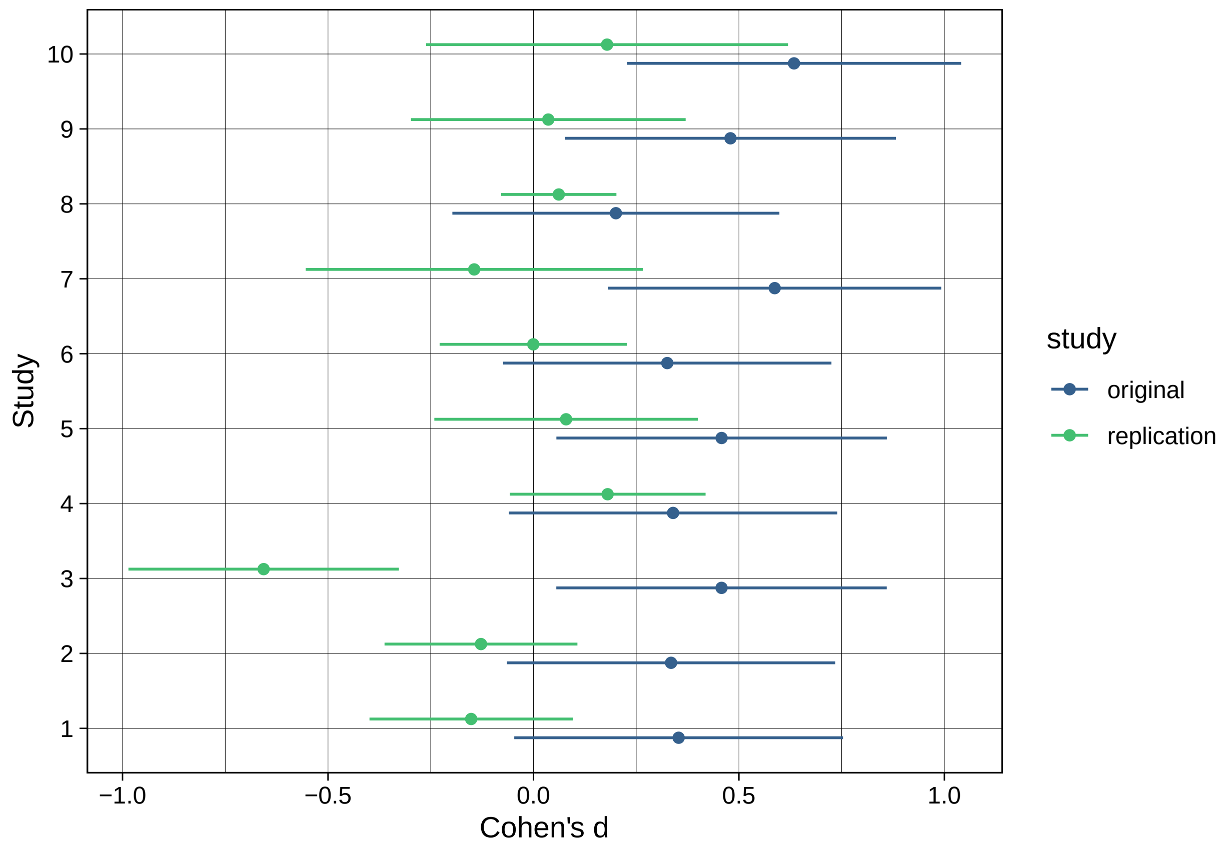
Even when the original studies are consistently hacked (all of them), replication studies cannot reliably detect different results.

Recall what we learned before about how directly comparing two effect sizes.

```r
# remove all objects from environment ----
#rm(list = ls())


# dependencies ----
# repeated here for the sake of completeness

library(tidyr)
library(dplyr)
library(tibble)
library(forcats)
library(purrr)
library(ggplot2)
library(knitr)
library(kableExtra)
library(janitor)
library(metafor)


# set the seed ----
# for the pseudo random number generator to make results reproducible
set.seed(46)


# define data generating function ----
generate_data <- function(n_per_condition,
                          mean_control,
                          mean_intervention,
                          sd_control,
                          sd_intervention) {
  require(tibble)
  require(dplyr)
  require(forcats)

  data_control <-
    tibble(condition = "control",
           score = rnorm(n = n_per_condition, mean = mean_control, sd = sd_control))

  data_intervention <-
    tibble(condition = "intervention",
           score = rnorm(n = n_per_condition, mean = mean_intervention, sd = sd_intervention))

  data <- bind_rows(data_control,
                    data_intervention) |>
    # control's factor levels must be ordered so that intervention is the first level and control is th
    # this ensures that positive cohen's d values refer to intervention > control and not the other way
    mutate(condition = fct_relevel(condition, "intervention", "control"))

  return(data)
}


# define data analysis function ----
```

```
analyse_data <- function(data) {
  require(effsize)
  require(tibble)

  res_n <- data |>
    count()

  res_t_test <- t.test(formula = score ~ condition,
                       data = data,
                       var.equal = FALSE,
                       alternative = "two.sided")

  res_cohens_d <- effsize::cohen.d(formula = score ~ condition,  # new addition: also fit cohen's d
                                   within = FALSE,
                                   data = data)

  res <- tibble(total_n = res_n$n,
                p = res_t_test$p.value,
                cohens_d_estimate = res_cohens_d$estimate,  # new addition: save cohen's d and its 95%
                cohens_d_ci_lower = res_cohens_d$conf.int["lower"],
                cohens_d_ci_upper = res_cohens_d$conf.int["upper"]) |>
    mutate(cohens_d_se = (cohens_d_ci_upper - cohens_d_ci_lower)/(1.96*2),
           cohens_d_variance = cohens_d_se^2)  # variance of effect size = its standard error squared

  return(res)
}


sample_size_planning <- function(results){
  require(pwr)
  fit_power <-
    pwr.t.test(d = results$cohens_d_estimate,
               power = .80, # .95
               sig.level = 0.05,
               type = "two.sample",
               alternative = "two.sided")

  res <- tibble(n_per_condition_replication = ceiling(fit_power$n))

  return(res)
}


# define experiment parameters ----
experiment_parameters_grid <- expand_grid(
  n_per_condition = 50,
  mean_control = 0,
  mean_intervention = 0,
  sd_control = 1,
  sd_intervention = 1,
  iteration = 1:10
) |>
  mutate(mean_intervention_biased = mean_intervention + 0.4)
```

```
# run simulation ----
simulation <-
  # using the experiment parameters
  experiment_parameters_grid |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data_original = pmap(list(n_per_condition,
                                             mean_control,
                                             mean_intervention_biased, # original studies effect size a
                                             sd_control,
                                             sd_intervention),
                                        generate_data)) |>

  # apply the analysis function to the generated data using the parameters relevant to analysis
  mutate(analysis_results_original = pmap(list(generated_data_original),
                                          analyse_data)) |>
  #select(-generated_data_original) |>

  mutate(parameters_replication = map(analysis_results_original, sample_size_planning)) |>
  unnest(parameters_replication) |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data_replication = pmap(list(n_per_condition_replication,
                                                mean_control,
                                                mean_intervention, # replication studies effect size is
                                                sd_control,
                                                sd_intervention),
                                           generate_data)) |>

  # apply the analysis function to the generated data using the parameters relevant to analysis
  mutate(analysis_results_replication = pmap(list(generated_data_replication),
                                             analyse_data))
  #select(-generated_data_replication)


# summarize results
simulation_summary <- simulation |>
  unnest(analysis_results_original) |>
  rename(total_n_original = total_n,
         p_original = p,
         cohens_d_estimate_original = cohens_d_estimate,
         cohens_d_ci_lower_original = cohens_d_ci_lower,
         cohens_d_ci_upper_original = cohens_d_ci_upper,
         cohens_d_se_original = cohens_d_se,
         cohens_d_variance_original = cohens_d_variance) |>

  unnest(analysis_results_replication) |>
  rename(total_n_replication = total_n,
         p_replication = p,
         cohens_d_estimate_replication = cohens_d_estimate,
         cohens_d_ci_lower_replication = cohens_d_ci_lower,
         cohens_d_ci_upper_replication = cohens_d_ci_upper,
         cohens_d_se_replication = cohens_d_se,
         cohens_d_variance_replication = cohens_d_variance)

simulation_summary |>
  select(iteration,
         estimate_original = cohens_d_estimate_original,
         lower_original = cohens_d_ci_lower_original,
         upper_original = cohens_d_ci_upper_original,
         estimate_replication = cohens_d_estimate_replication,
         lower_replication = cohens_d_ci_lower_replication,
         upper_replication = cohens_d_ci_upper_replication) |>
  pivot_longer(cols = -iteration,
               names_to = c("type", "study"),
               names_sep = "_",
               values_to = "estimate") |>
  pivot_wider(names_from = "type",
              values_from = "estimate") |>
  ggplot(aes(estimate, as.factor(iteration), color = study)) +
  geom_linerangeh(aes(xmin = lower, xmax = upper),
                  position = position_dodge(width = 0.5)) +
  geom_point(position = position_dodge(width = 0.5)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7) +
  theme_linedraw() +
  ylab("Study") +
  xlab("Cohen's d")
```



```r
# proportion of original and replication pairs that differ significantly
simulation_summary |>
  mutate(diff = case_when(cohens_d_ci_lower_replication > cohens_d_ci_upper_original ~ TRUE,
                          cohens_d_ci_upper_replication < cohens_d_ci_lower_original ~ TRUE,
                          TRUE ~ FALSE)) |>
  summarize(proportion_diff = mean(diff)) |>
  kable() |>
  kable_classic(full_width = FALSE)
```

proportion_diff
0.1

Session info

```r
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 11 x64 (build 22631)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=German_Switzerland.utf8  LC_CTYPE=German_Switzerland.utf8
## [3] LC_MONETARY=German_Switzerland.utf8 LC_NUMERIC=C
## [5] LC_TIME=German_Switzerland.utf8
##
## time zone: Europe/Zurich
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] ggstance_0.3.7      pwr_1.3-0           kableExtra_1.4.0
##  [4] knitr_1.46          parameters_0.21.6   metafor_4.6-0
##  [7] numDeriv_2016.8-1.1 metadat_1.2-0       Matrix_1.6-5
## [10] tibble_3.2.1        janitor_2.2.0       effsize_0.8.1
## [13] ggplot2_3.5.1       purrr_1.0.2         readr_2.1.5
## [16] forcats_1.0.0       dplyr_1.1.4         tidyr_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.4        generics_0.1.3    xml2_1.3.6        stringi_1.8.4
##  [5] lattice_0.22-6    hms_1.1.3         digest_0.6.35     magrittr_2.0.3
##  [9] evaluate_0.23     grid_4.3.2        timechange_0.3.0  fastmap_1.1.1
## [13] fansi_1.0.6       viridisLite_0.4.2 scales_1.3.0      cli_3.6.2
## [17] rlang_1.1.3       munsell_0.5.1     withr_3.0.0       yaml_2.3.8
## [21] datawizard_0.10.0 tools_4.3.2       tzdb_0.4.0        colorspace_2.1-0
## [25] mathjaxr_1.6-0    bayestestR_0.13.2 vctrs_0.6.5       R6_2.5.1
## [29] lifecycle_1.0.4   lubridate_1.9.3   snakecase_0.11.1  stringr_1.5.1
## [33] insight_0.19.10   pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.5
## [37] glue_1.7.0        systemfonts_1.0.6 highr_0.10        xfun_0.43
## [41] tidyselect_1.2.1  rstudioapi_0.16.0 farver_2.1.1      htmltools_0.5.8.1
## [45] nlme_3.1-164      labeling_0.4.3    svglite_2.1.3     rmarkdown_2.26
## [49] compiler_4.3.2
```