Not-so-standardized effect sizes

Versions of Cohen's *d* for dependent data

Ian Hussey

06 Mai, 2024

Contents

Hypothetical study

Imagine that, like Balcetis & Dunning (2010), you want to answer a very simple question: how much do people prefer chocolate over poop?

- You have participants rate pictures of chocolate and pictures of poop from negative (1) to positive (7).
- How should you analyse this data to compare the mean evaluations of chocolate or poop? Should the test/effect size be independent or dependent?
- What version of Cohen's d should you use to calculate the effect size?
- More mundanely, what implementation of this version of Cohen's d should you use - i.e., which R package?

The specific choice of which version of Cohen's d won't affect the results that much... right? Studies and meta-analyses never make much distinction between them, so they must all be quite similar, right?

And since different packages implement the same math, they won't produce different results, right? Right??

Let's explore this in simulations.

```
# Dependencies
# N.B.: to ensure full computational reproducibility, R version 4.3.3 should be used.

library(groundhog)

groundhog_day = "2024-04-07"

packages = c("effectsize","faux","janitor",
             "rstatix", "effsize",
             "psych", "MBESS", "lsr",
```

```
              "metafor", "esc", "esci",
              "dplyr", "tidyr", "tibble",
              "forcats", "ggplot2", "stringr")

groundhog.library(packages, groundhog_day)
```

Functions

I have gathered many popular versions of Cohen's d, both dependent and independent, and their implementations. Some of these I am still working to understand what label/name is appropriate for each package's version of Cohen's d.

```
cohens_d_s_metafor <- function(data){
  # nb always applies hedges correction
  require(dplyr)
  require(tibble)
  require(metafor)

  summaries <- data |>
    group_by(timepoint) |>
    summarize(mean = mean(score),
              sd = sd(score),
              n = n()) |>
    pivot_wider(names_from = "timepoint",
                values_from = c("mean", "sd", "n"))

  fit <- escalc(measure = "SMD",
                m1i = mean_stimulus2,
                sd1i = sd_stimulus2,
                n1i = n_stimulus2,
                m2i = mean_stimulus1,
                sd2i = sd_stimulus1,
                n2i = n_stimulus1,
                data = summaries,
                append = FALSE)

  res <-
    tibble(estimate = fit$yi,
           ci_lower = fit$yi - sqrt(fit$vi)*1.96,
           ci_upper = fit$yi + sqrt(fit$vi)*1.96)
  return(res)
}


cohens_d_s_esc <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(esc)

  summaries <- data |>
    group_by(timepoint) |>
    summarize(mean = mean(score),
              sd = sd(score),
```

```
              n = n())

  fit <- esc::esc_mean_sd(grp1m  = summaries$mean[summaries$timepoint == "stimulus2"],
                          grp1sd = summaries$sd[summaries$timepoint == "stimulus2"],
                          grp1n  = summaries$n[summaries$timepoint == "stimulus2"],
                          grp2m  = summaries$mean[summaries$timepoint == "stimulus1"],
                          grp2sd = summaries$sd[summaries$timepoint == "stimulus1"],
                          grp2n  = summaries$n[summaries$timepoint == "stimulus1"],
                          es.type = ifelse(hedges_correction, "g", "d"))

  res <-
    tibble(estimate = fit$es,
           ci_lower = fit$ci.lo,
           ci_upper = fit$ci.hi)

  return(res)
}


cohens_d_z_esc <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(esc)

  summaries <- data |>
    group_by(timepoint) |>
    summarize(mean = mean(score),
              sd = sd(score),
              n = n())

  r <- data |>
    pivot_wider(names_from = "timepoint",
                values_from = "score") |>
    dplyr::select(-id) |>
    cor_test()

  fit <- esc::esc_mean_sd(grp1m  = summaries$mean[summaries$timepoint == "stimulus2"],
                          grp1sd = summaries$sd[summaries$timepoint == "stimulus2"],
                          grp1n  = summaries$n[summaries$timepoint == "stimulus2"],
                          grp2m  = summaries$mean[summaries$timepoint == "stimulus1"],
                          grp2sd = summaries$sd[summaries$timepoint == "stimulus1"],
                          grp2n  = summaries$n[summaries$timepoint == "stimulus1"],
                          r = r$cor,
                          es.type = ifelse(hedges_correction, "g", "d"))

  res <-
    tibble(estimate = fit$es,
           ci_lower = fit$ci.lo,
           ci_upper = fit$ci.hi)

  return(res)
}
```

```r
cohens_d_s_rstatix <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(rstatix)

  fit <- rstatix::cohens_d(formula = score ~ timepoint,
                           data = data,
                           comparisons = list(c("stimulus1", "stimulus2")),
                           ref.group = "stimulus2",
                           paired = FALSE,
                           hedges.correction = hedges_correction,
                           ci = TRUE,
                           ci.type = "bca",
                           nboot = 2000)

  res <-
    tibble(estimate = fit$effsize,
           ci_lower = fit$conf.low,
           ci_upper = fit$conf.high)

  return(res)
}


cohens_d_z_rstatix <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(rstatix)

  fit <- rstatix::cohens_d(formula = score ~ timepoint,
                           data = data,
                           comparisons = list(c("stimulus1", "stimulus2")),
                           ref.group = "stimulus2",
                           paired = TRUE,
                           hedges.correction = hedges_correction,
                           ci = TRUE,
                           ci.type = "bca",
                           nboot = 2000)

  res <-
    tibble(estimate = fit$effsize,
           ci_lower = fit$conf.low,
           ci_upper = fit$conf.high)

  return(res)
}


# esci used to have a within subjects d in 2020 but now doesn't? and its not in the git history
cohens_d_s_esci <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(esci)
```

```
  summaries <- data |>
    group_by(timepoint) |>
    summarize(mean = mean(score),
              sd = sd(score),
              n = n())

  fit <- esci::CI_smd_ind_contrast(means = summaries$mean,
                                   sds = summaries$sd,
                                   ns = summaries$n,
                                   contrast = c(+1, -1),
                                   conf_level = 0.95,
                                   assume_equal_variance = FALSE,
                                   correct_bias = hedges_correction)

  res <-
    tibble(estimate = fit$effect_size,
           ci_lower = fit$LL,
           ci_upper = fit$UL)

  return(res)
}


cohens_d_s_effsize <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(effsize)

  fit <- effsize::cohen.d(score ~ timepoint,
                          paired = FALSE,
                          pooled = TRUE,
                          hedges.correction = hedges_correction,
                          data = data)

  res <-
    tibble(estimate = fit$estimate,
           ci_lower = fit$conf.int[1],
           ci_upper = fit$conf.int[2])

  return(res)
}


cohens_d_dep_effsize <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(effsize)

  fit <- effsize::cohen.d(score ~ timepoint | Subject(id),
                          paired = TRUE,
                          pooled = TRUE,
                          hedges.correction = hedges_correction,
                          data = data)
```

```
  res <-
    tibble(estimate = fit$estimate,
           ci_lower = fit$conf.int[1],
           ci_upper = fit$conf.int[2])

  return(res)
}



cohens_d_s_psych <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(tibble)
  require(psych)

  fit <- psych::cohen.d(score ~ timepoint,
                        data = data)

  res <-
    tibble(estimate = ifelse(hedges_correction, fit$hedges.g[2]*-1, fit$cohen.d[2]*-1),
           ci_lower = ifelse(hedges_correction, min(fit$hedges.g*-1), min(fit$cohen.d*-1)),
           ci_upper = ifelse(hedges_correction, max(fit$hedges.g*-1), max(fit$cohen.d*-1)))

  return(res)
}



cohens_d_s_mbess <- function(data, hedges_correction = TRUE){
  require(dplyr)
  require(MBESS)

  d <- MBESS::smd(Group.1 = data |> filter(timepoint == "stimulus2") |> pull(score),
                  Group.2 = data |> filter(timepoint == "stimulus1") |> pull(score),
                  Unbiased = hedges_correction)

  t <- t.test(score ~ timepoint, data = data)$statistic

  cis <- ci.smd(ncp = t,
                #smd = d,
                n.1 = data |> filter(timepoint == "stimulus2") |> pull(score) |> length(),
                n.2 = data |> filter(timepoint == "stimulus1") |> pull(score) |> length())

  res <-
    data.frame(estimate = cis$smd,
               ci_lower = cis$Lower.Conf.Limit.smd,
               ci_upper = cis$Upper.Conf.Limit.smd)

  return(res)
}


cohens_d_s_lsr <- function(data){
```

```r
  require(dplyr)
  require(tibble)
  require(lsr)

  # note that lsr::cohenD returns the absolute value of cohen's d, ie always positive values. fix this
  mean_stimulus1 <- data |> dplyr::filter(timepoint == "stimulus1") |> summarize(mean = mean(score))
  mean_stimulus2 <- data |> dplyr::filter(timepoint == "stimulus2") |> summarize(mean = mean(score))

  d <- lsr::cohensD(score ~ timepoint,
                    data = data)

  res <-
    tibble(estimate = as.numeric(ifelse(mean_stimulus1 < mean_stimulus2, d, d * -1)),
           ci_lower = NA,
           ci_upper = NA)

  return(res)
}


cohens_d_dep_lsr <- function(data){
  require(dplyr)
  require(tibble)
  require(lsr)

  # note that lsr::cohenD returns the absolute value of cohen's d, ie always positive values. fix this
  mean_stimulus1 <- data |> dplyr::filter(timepoint == "stimulus1") |> summarize(mean = mean(score))
  mean_stimulus2 <- data |> dplyr::filter(timepoint == "stimulus2") |> summarize(mean = mean(score))

  d <- lsr::cohensD(data |> dplyr::filter(timepoint == "stimulus1") |> pull(score),
                    data |> dplyr::filter(timepoint == "stimulus2") |> pull(score),
                    method = "paired")

  res <-
    tibble(estimate = as.numeric(ifelse(mean_stimulus1 < mean_stimulus2, d, d * -1)),
           ci_lower = NA,
           ci_upper = NA)

  return(res)
}


# assumes that 'data' contains columns named score (numeric) and timepoint (factor with two levels, sti
# assumes a two sided Student's t test with alpha = .05
# by default, assumes correlation between timepoints is 0. larger values will narrow the CIs but leave
d_t_dependent <- function(data, hedges_correction = TRUE, r = 0) { # assumes a correlation of 0 for sim
  require(dplyr)

  fit <- t.test(data$score[data$timepoint == "stimulus2"],
                data$score[data$timepoint == "stimulus1"],
                paired = TRUE)

  res <-
```

```
    data.frame(t_stat = fit$statistic["t"],
               df = fit$parameter["df"],
               r = r) |>
    mutate(n = df + 1, # for dependent t test, n = df + 1
           dt_estimate = t_stat / sqrt(n), # for dependent t test, d_t = t / sqrt(n)
           dt_estimate = ifelse(hedges_correction,
                                dt_estimate * (1 - (3 / (4 * n - 9))),
                                dt_estimate),
           dt_se = sqrt((2 * (1 - r)) / n + (dt_estimate ^ 2) / (2 * n)), # default assumes a correlati
           ci_lower = dt_estimate - (1.96 * dt_se),
           ci_upper = dt_estimate + (1.96 * dt_se),
           hedges_correction = hedges_correction) |>
    dplyr::select(hedges_correction, estimate = dt_estimate, ci_lower, ci_upper)

  return(res)
}


# cohen's d from a Student's t test (two sided, alpha = .05)
# if a Welches' t test, the conversion of df to N is imstimulus1cise, affecting the CIs.
# assumes that 'data' contains columns named score (numeric) and timepoint (factor with two levels)
d_t_independent <- function(data, hedges_correction = TRUE){
  require(dplyr)

  fit <- t.test(score ~ timepoint,
                data = data,
                paired = FALSE,
                var.equal = TRUE)

  res <-
    data.frame(t_stat = fit$statistic["t"],
               df = fit$parameter["df"]) |>
    mutate(n1 = data |> filter(timepoint == "stimulus2") |> nrow(),
           n2 = data |> filter(timepoint == "stimulus1") |> nrow(),
           dt_estimate = t_stat * sqrt(1/n1 + 1/n2), # d = t * sqrt(1/n1 + 1/n2) - from lakens 2013 equ
           dt_estimate = ifelse(hedges_correction,
                                dt_estimate * (1 - (3 / (4 * (n1+n2) - 9))),
                                dt_estimate),
           dt_se = sqrt((n1 + n2) / (n1 * n2) + (dt_estimate^2) / (2 * (n1 + n2 - 2))),
           ci_lower = dt_estimate - (1.96 * dt_se),
           ci_upper = dt_estimate + (1.96 * dt_se),
           hedges_correction = hedges_correction) |>
    dplyr::select(hedges_correction, estimate = dt_estimate, ci_lower, ci_upper)

  return(res)
}


multiple_cohens_ds_for_dependent_data <- function(data, hedges_correction = TRUE){

  # Check if 'data' is a dataframe or tibble
  if (!is.data.frame(data) && !is_tibble(data)) {
    stop("The 'data' argument must be a dataframe or tibble.")
```

```
}

# Check for 'score' column and its type
if (!"score" %in% names(data) || !is.numeric(data$score)) {
  stop("The 'data' must contain a numeric column named 'score'.")
}

# Check for 'timepoint' column, its type, and number of levels
if (!"timepoint" %in% names(data)) {
  stop("The 'data' must contain a column named 'timepoint'.")
}
if (!is.factor(data$timepoint) && !is.character(data$timepoint)) {
  stop("The 'timepoint' column must be of type factor or character.")
}

# some pivots below assume only these columns are stimulus1sent
data <- data |>
  select(id, timepoint, score)

# Convert 'timepoint' to factor if it's not already
data$timepoint <- as.factor(data$timepoint)

# Check that 'timepoint' has exactly two levels
if (nlevels(data$timepoint) != 2) {
  stop("The 'timepoint' column must have exactly two levels.")
}

results <-
  bind_rows(
    effectsize::cohens_d(score ~ timepoint, data = data, pooled_sd = TRUE, adjust = hedges_correction
      as_tibble() |>
      mutate(type = "d_s",
             implementation = "{effectsize}",
             hedges_correction = hedges_correction) |>
      dplyr::select(implementation, type, hedges_correction, estimate = Hedges_g, ci_lower = CI_low,

    cohens_d_s_psych(data = data, hedges_correction = hedges_correction) |>
      as_tibble() |>
      mutate(type = "d_s",
             implementation = "{psych}",
             hedges_correction = hedges_correction) |>
      dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

    cohens_d_s_mbess(data = data, hedges_correction = hedges_correction) |>
      mutate(type = "d_s",
             implementation = "{MBESS}",
             hedges_correction = hedges_correction) |>
      dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

    cohens_d_s_effsize(data = data, hedges_correction = hedges_correction) |>
      mutate(type = "d_s",
             implementation = "{effsize}",
             hedges_correction = hedges_correction) |>
```

```
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_s_rstatix(data = data, hedges_correction = hedges_correction) |>
    mutate(type = "d_s",
           implementation = "{rstatix}",
           hedges_correction = hedges_correction) |>
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_s_metafor(data = data) |>
    mutate(type = "d_s",
           implementation = "{metafor}",
           hedges_correction = TRUE) |> # nb always applies hedges correction
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_z_rstatix(data = data, hedges_correction = hedges_correction) |>
    mutate(type = "d_z",
           implementation = "{rstatix}",
           hedges_correction = hedges_correction) |>
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_s_esc(data = data, hedges_correction = hedges_correction) |>
    mutate(type = "d_s",
           implementation = "{esc}",
           hedges_correction = hedges_correction) |>
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_z_esc(data = data, hedges_correction = hedges_correction) |>
    mutate(type = "d_??? (dep)",
           implementation = "{esc}",
           hedges_correction = hedges_correction) |>
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_s_esci(data = data, hedges_correction = hedges_correction) |>
    mutate(type = "d_??? (indep)",
           implementation = "{esci}",
           hedges_correction = hedges_correction) |>
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_dep_effsize(data = data, hedges_correction = hedges_correction) |>
    mutate(type = "d_??? (dep)",
           implementation = "{effsize}",
           hedges_correction = hedges_correction) |>
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_s_lsr(data = data) |>
    mutate(type = "d_s",
           implementation = "{lsr}",
           hedges_correction = FALSE) |> # lsr doesn't have an option for hedges corrections
    dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

  cohens_d_dep_lsr(data = data) |>
    mutate(type = "d_z",
           implementation = "{lsr}",
```

```
          hedges_correction = FALSE) |> # lsr doesn't have an option for hedges corrections
  dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

# this provides nearly identical results to d_s under most conditions
# effectsize::cohens_d(score ~ timepoint, data = data, pooled_sd = FALSE, adjust = hedges_correct
#   as_tibble() |>
#   mutate(type = "d_s_nonpooled",
#          implementation = "{effectsize}",
#          hedges_correction = hedges_correction) |>
#   dplyr::select(implementation, type, hedges_correction, estimate = Hedges_g, ci_lower = CI_low

effectsize::repeated_measures_d(score ~ timepoint | id, data = data, method = "d", adjust = hedge
  as_tibble() |>
  mutate(type = "d_s_withinCIs",
         implementation = "{effectsize}",
         hedges_correction = hedges_correction) |>
  dplyr::select(implementation, type, hedges_correction, estimate = Cohens_d, ci_lower = CI_low,

effectsize::repeated_measures_d(score ~ timepoint | id, data = data, method = "rm", adjust = hedg
  as_tibble() |>
  mutate(type = "d_rm",
         implementation = "{effectsize}",
         hedges_correction = hedges_correction) |>
  dplyr::select(implementation, type, hedges_correction, estimate = d_rm, ci_lower = CI_low, ci_u

effectsize::repeated_measures_d(score ~ timepoint | id, data = data, method = "av", adjust = hedg
  as_tibble() |>
  mutate(type = "d_av",
         implementation = "{effectsize}",
         hedges_correction = hedges_correction) |>
  dplyr::select(implementation, type, hedges_correction, estimate = d_av, ci_lower = CI_low, ci_u

effectsize::repeated_measures_d(score ~ timepoint | id, data = data, method = "b", adjust = hedge
  as_tibble() |>
  mutate(type = "d_b",
         implementation = "{effectsize}",
         hedges_correction = hedges_correction) |>
  dplyr::select(implementation, type, hedges_correction, estimate = Beckers_d, ci_lower = CI_low,

# use timepoint stimulus2's SD rather than stimulus1.
# this is useful to include as not all comparisons are stimulus1-stimulus2, some are the same par
data |>
  mutate(timepoint = fct_relevel(timepoint, "stimulus1", "stimulus2")) |>
  effectsize::repeated_measures_d(score ~ timepoint | id, data = _, method = "b", adjust = hedges_
  as_tibble() |>
  mutate(type = "d_b (alt)",
         implementation = "{effectsize}",
         hedges_correction = hedges_correction,
         estimate = Beckers_d * -1,
         ci_lower = CI_high * -1,
         ci_upper = CI_low * -1) |>
  dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),
```

```
    effectsize::repeated_measures_d(score ~ timepoint | id, data = data, method = "z", adjust = hedges
      as_tibble() |>
      mutate(type = "d_z",
             implementation = "{effectsize}",
             hedges_correction = hedges_correction) |>
      dplyr::select(implementation, type, hedges_correction, estimate = d_z, ci_lower = CI_low, ci_up

    # this provides the same estimate as d_s
    d_t_independent(data = data, hedges_correction = hedges_correction) |>
      as_tibble() |>
      mutate(type = "d_t (independent)",
             implementation = "[custom]",
             hedges_correction = hedges_correction) |>
      dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper),

    # this provides the same estimate as d_z with slightly wider CIs
    d_t_dependent(data = data, hedges_correction = hedges_correction) |>
      as_tibble() |>
      mutate(type = "d_t (dependent, r = 0)",
             implementation = "[custom]",
             hedges_correction = hedges_correction) |>
      dplyr::select(implementation, type, hedges_correction, estimate, ci_lower, ci_upper)
  )
  return(results)
}


plot_different_dependent_cohens_ds <- function(parameters){
  data_for_simulations <-
    faux::rnorm_multi(n = parameters$simulation_n,
                      mu = c(stimulus1 = parameters$mean_stimulus1, stimulus2 = parameters$mean_stimulu
                      sd = c(parameters$sd_stimulus1, parameters$sd_stimulus2),
                      r = matrix(c(1, parameters$r_stimulus1_stimulus2,
                                   parameters$r_stimulus1_stimulus2, 1),
                                 ncol = 2)) |>
    rownames_to_column(var = "id") |>
    pivot_longer(cols = -id,
                 names_to = "timepoint",
                 values_to = "score") |>
    mutate(timepoint = fct_relevel(timepoint, "stimulus2", "stimulus1")) # ensure that factor levels ar

  simulation_results <- data_for_simulations |>
    multiple_cohens_ds_for_dependent_data() |>
    select(-hedges_correction) |>
    mutate(ci_width = ci_upper - ci_lower,
           sig = ifelse((ci_lower > 0 & ci_upper > 0) |
                          (ci_lower < 0 & ci_upper < 0) |
                          (is.na(ci_lower) & is.na(ci_upper)), TRUE, FALSE))

  plot <- ggplot(simulation_results, aes(paste(type, implementation), estimate, color = sig)) +
    geom_hline(yintercept = 0, linetype = "dashed") +
    geom_linerange(aes(ymin = ci_lower, ymax = ci_upper), size = 0.8) +
    geom_point(position = position_dodge(width = 0.5), size = 1.8) +
```

```
    scale_y_continuous(breaks = scales::pretty_breaks()) +
    coord_flip() +
    theme_linedraw() +
    xlab("") +
    ylab("Cohen's d") +
    scale_color_viridis_d(begin = 0.2, end = 0.5, option = "D") +
    theme(legend.position = "none",
          panel.grid.minor = element_blank())

  return(plot)
}
```

Simulate data

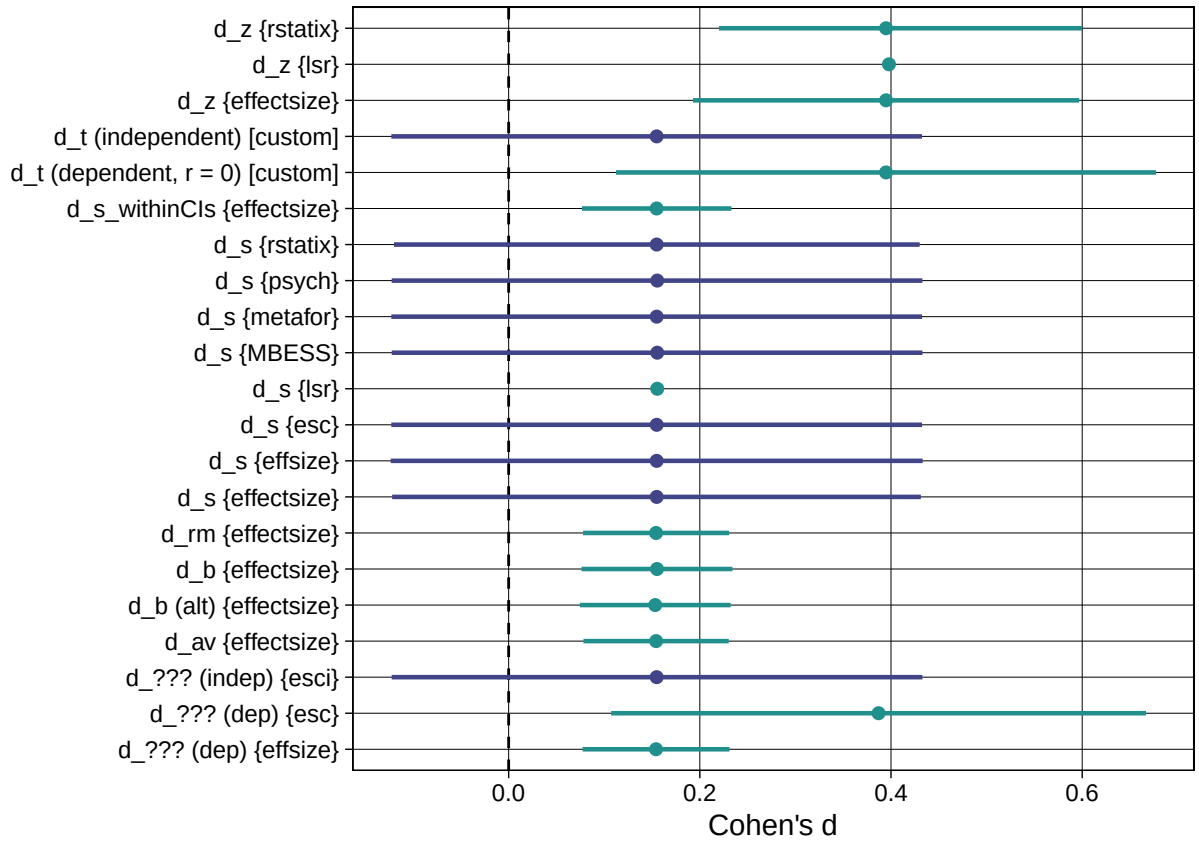
Simulation 1

Small effect size, equal variances, very large correlation between ratings.

```
set.seed(42)

parameters <-
  data.frame(simulation_n = 100,
             mean_stimulus1 = 0,
             mean_stimulus2 = 0.2,
             sd_stimulus1 = 1,
             sd_stimulus2 = 1,
             r_stimulus1_stimulus2 = 0.9)

parameters |>
  pivot_longer(cols = everything(),
               names_to = "parameter",
               values_to = "value") |>
  knitr::kable()
```

parameter	value
simulation_n	100.0
mean_stimulus1	0.0
mean_stimulus2	0.2
sd_stimulus1	1.0
sd_stimulus2	1.0
r_stimulus1_stimulus2	0.9

```
plot_different_dependent_cohens_ds(parameters)
```



Simulation 2

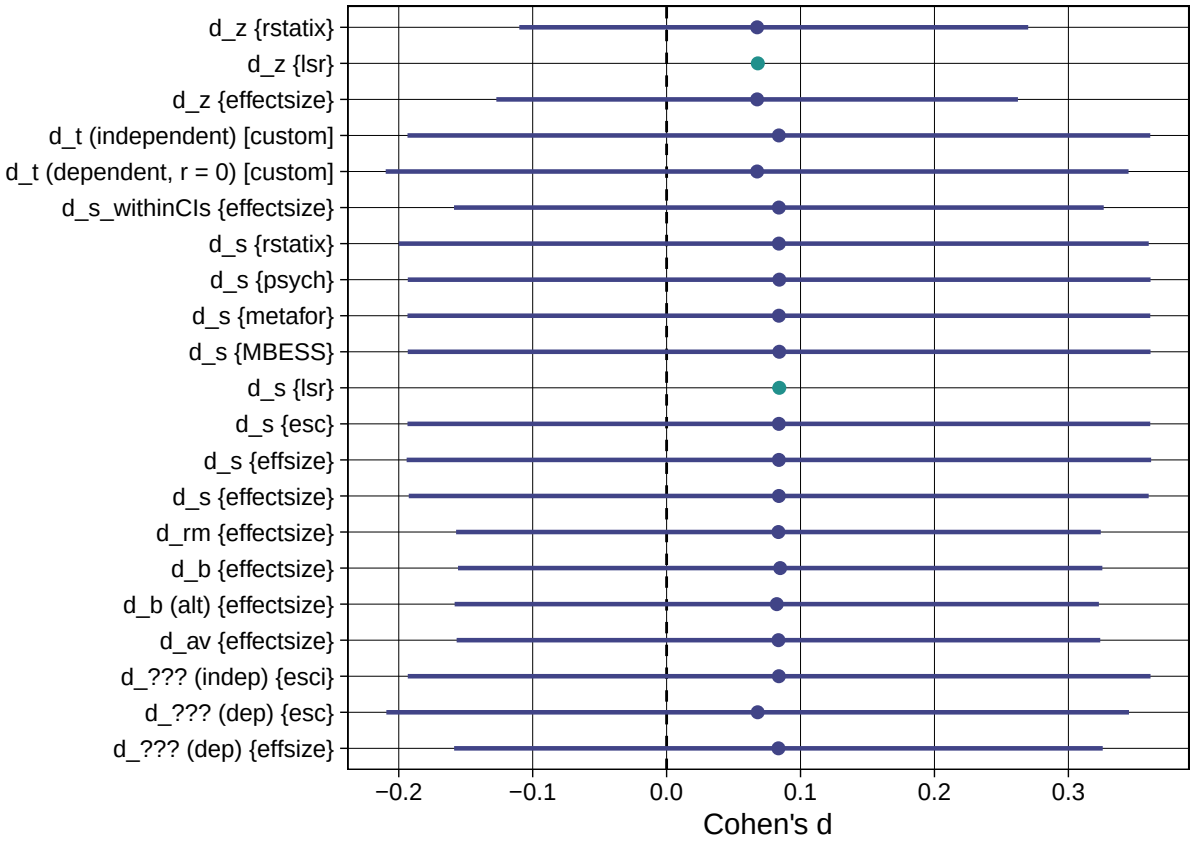
Small effect size, equal variances, small correlation between ratings.

```
set.seed(42)

parameters <-
  data.frame(simulation_n = 100,
             mean_stimulus1 = 0,
             mean_stimulus2 = 0.2,
             sd_stimulus1 = 1,
             sd_stimulus2 = 1,
             r_stimulus1_stimulus2 = 0.1)

parameters |>
  pivot_longer(cols = everything(),
               names_to = "parameter",
               values_to = "value") |>
  knitr::kable()
```

parameter	value
simulation_n	100.0
mean_stimulus1	0.0
mean_stimulus2	0.2
sd_stimulus1	1.0
sd_stimulus2	1.0

parameter	value
r_stimulus1_stimulus2	0.1

```
plot_different_dependent_cohens_ds(parameters)
```



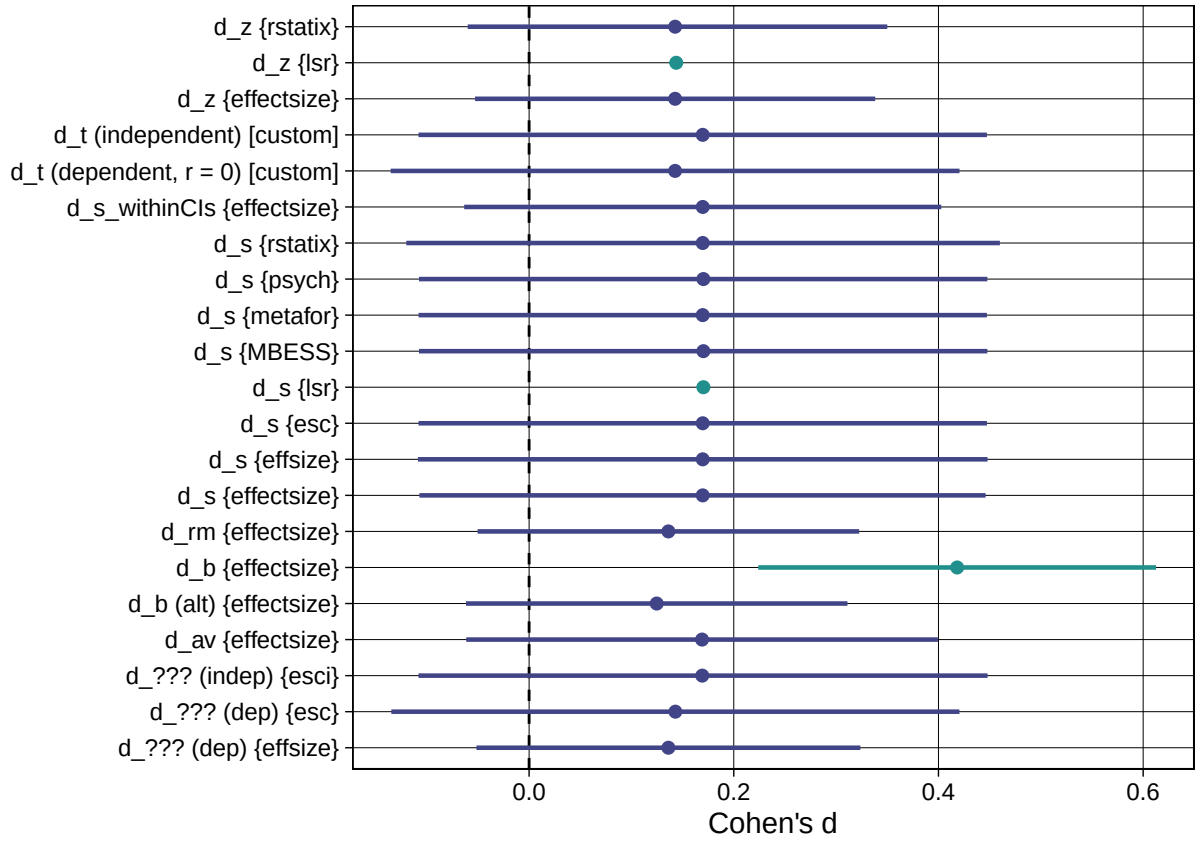
Simulation 3

Small effect size, unequal variances, large correlation between ratings.

```
set.seed(42)

parameters <-
  data.frame(simulation_n = 100,
             mean_stimulus1 = 0,
             mean_stimulus2 = 0.2,
             sd_stimulus1 = 0.5,
             sd_stimulus2 = 1.5,
             r_stimulus1_stimulus2 = 0.5)

parameters |>
  pivot_longer(cols = everything(),
               names_to = "parameter",
               values_to = "value") |>
  knitr::kable()
```

parameter	value
simulation_n	100.0
mean_stimulus1	0.0
mean_stimulus2	0.2
sd_stimulus1	0.5
sd_stimulus2	1.5
r_stimulus1_stimulus2	0.5

```
plot_different_dependent_cohens_ds(parameters)
```



Simulation 4

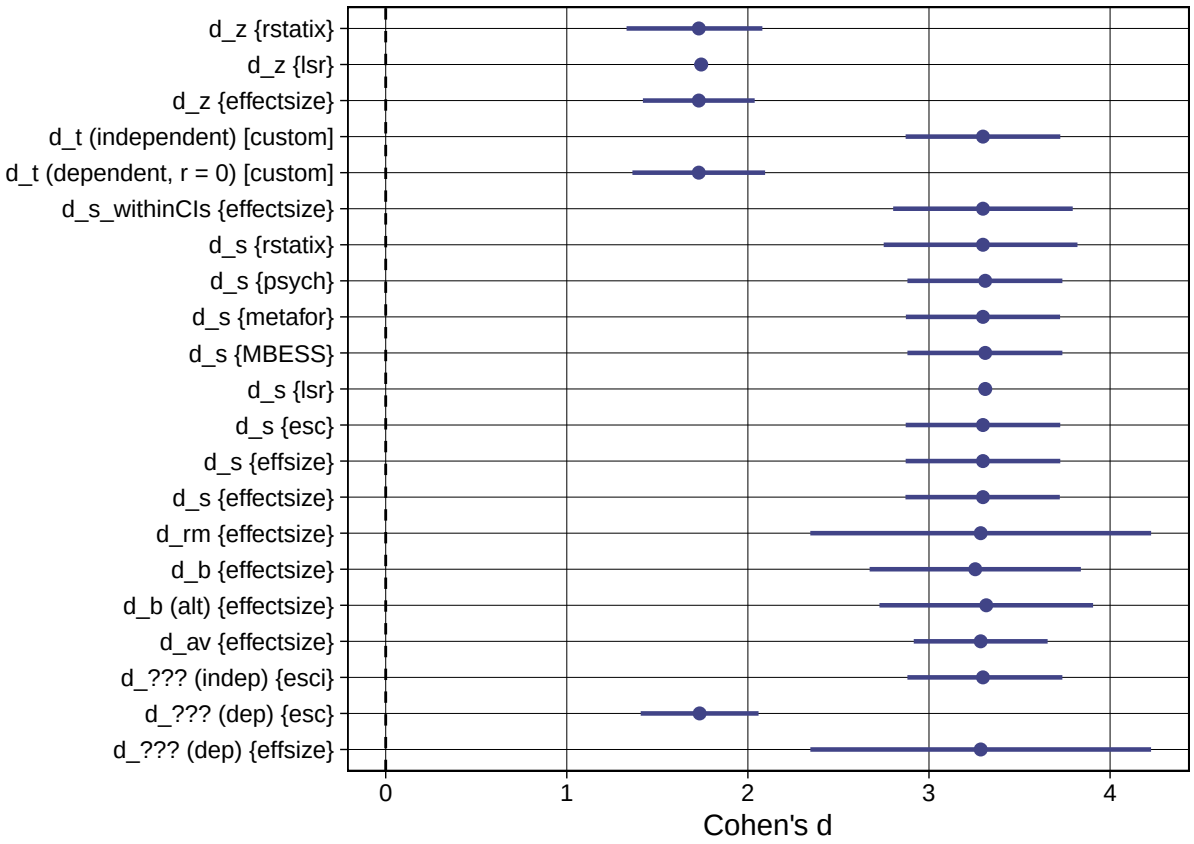
Plausible parameters for preference between chocolate and poop on a 1 to 7 Likert scale.

```
set.seed(42)

parameters <-
  data.frame(simulation_n = 100,
             mean_stimulus1 = 1.5,
             mean_stimulus2 = 6.5,
             sd_stimulus1 = 1.5,
             sd_stimulus2 = 1.5,
             r_stimulus1_stimulus2 = -0.75)

parameters |>
```

```
  pivot_longer(cols = everything(),
               names_to = "parameter",
               values_to = "value") |>
  knitr::kable()
```

parameter	value
simulation_n	100.00
mean_stimulus1	1.50
mean_stimulus2	6.50
sd_stimulus1	1.50
sd_stimulus2	1.50
r_stimulus1_stimulus2	-0.75

```
plot_different_dependent_cohens_ds(parameters)
```



Session info

```
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] stringr_1.5.1       ggplot2_3.5.0       forcats_1.0.0
##  [4] tibble_3.2.1        tidyr_1.3.1         dplyr_1.1.4
##  [7] esci_1.0.2          esc_0.5.1           metafor_4.6-0
## [10] numDeriv_2016.8-1.1 metadat_1.2-0       Matrix_1.6-5
## [13] lsr_0.5.2           MBESS_4.9.3         psych_2.4.3
## [16] effsize_0.8.1       rstatix_0.7.2       janitor_2.2.0
## [19] faux_1.2.1          effectsize_0.8.7    groundhog_3.2.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4         xfun_0.43            bayestestR_0.13.2
##  [4] contfrac_1.1-12      elliptic_1.4-0       insight_0.19.10
##  [7] lattice_0.22-6       mathjaxr_1.6-0       vctrs_0.6.5
## [10] tools_4.3.2          generics_0.1.3       parallel_4.3.2
## [13] datawizard_0.10.0    fansi_1.0.6          highr_0.10
## [16] pkgconfig_2.0.3      distributional_0.4.0 lifecycle_1.0.4
## [19] farver_2.1.1         compiler_4.3.2       munsell_0.5.1
## [22] mnormt_2.1.1         PDQutils_0.1.6       carData_3.0-5
## [25] snakecase_0.11.1     htmltools_0.5.8.1    yaml_2.3.8
## [28] mnonr_1.0.0          pillar_1.9.0         car_3.1-2
## [31] MASS_7.3-60.0.1      hypergeo_1.2-13      boot_1.3-30
## [34] abind_1.4-5          nlme_3.1-164         tidyselect_1.2.1
## [37] digest_0.6.35        stringi_1.8.3        purrr_1.0.2
## [40] fastmap_1.1.1        grid_4.3.2           colorspace_2.1-0
## [43] cli_3.6.2            magrittr_2.0.3       utf8_1.2.4
## [46] orthopolynom_1.0-6.1 broom_1.0.5          withr_3.0.0
## [49] sadists_0.2.5        scales_1.3.0         backports_1.4.1
## [52] lubridate_1.9.3      timechange_0.3.0     rmarkdown_2.26
## [55] statpsych_1.5.0      deSolve_1.40         moments_0.14.1
## [58] evaluate_0.23        knitr_1.46           parameters_0.21.6
## [61] ggdist_3.3.2         viridisLite_0.4.2    rlang_1.1.3
## [64] Rcpp_1.0.12          polynom_1.4-1        glue_1.7.0
## [67] rstudioapi_0.16.0    R6_2.5.1
```