p values, Confidence Intervals, and sequential testing

Ian Hussey

06 Mai, 2024

Contents

Overview of lesson

In a previous lesson, we briefly talked about the "dance of the p-values". Because p values are uniformly distributed under the null hypothesis, when the true effect is null, p values between studies will dance back and forth between all possible values with equal probability (from 0 to 1).

But what happens when we're not looking at independent studies, but adding additional participants to the same study, and reanalyzing repeatedly with a little more data each time?

The figure below illustrates this. The first analysis uses participants 1 to 10 in each group. The second analysis uses participants 1 to 15 in each group. The third uses participants 1 to 20 in each group, etc.

This involves dependencies between the datasets, so the results will be related to one another too.

This lesson simulates such *sequential analysis*, in order to show:

1. How p values 'dance' in sequential analysis
2. Why 'optional stopping' is problematic and a form of p-hacking.
3. The relationship between p values and Confidence Intervals.
4. How the width of Confidence Intervals narrow as sample size increases.
5. How p values, confidence intervals, and statistical power / false positive rates are all interrelated.

```r
knitr::include_graphics("images/sequential analysis illustration.png", dpi = 100)
```

condition	score	id
control	1.9946963377	1
intervention	-0.0425145252	1
control	0.7111425051	2
intervention	-0.3654730678	2
control	0.1854052843	3
intervention	-0.3246292982	3
control	-0.2817650147	4
intervention	0.8473205646	4
control	0.1087755466	5
intervention	0.7594290016	5
control	-1.0857374702	6
intervention	0.2988294445	6
control	-0.9854821582	7
intervention	0.0389536708	7
control	0.0151308601	8
intervention	1.5920320178	8
control	-0.2520458977	9
intervention	1.7607354005	9
control	-1.4657503001	10
intervention	0.8711555278	10
control	-0.9224562385	11
intervention	-0.0474960823	11
control	0.0396024331	12
intervention	0.6078219104	12
control	0.4938201830	13
intervention	0.4365070801	13
control	-1.8282291682	14
intervention	0.7047397268	14
control	0.0914729119	15
intervention	1.5699502564	15
control	0.6707792190	16
intervention	-0.8581000555	16
control	-0.0810780515	17
intervention	0.1922206956	17
control	1.2642410898	18
intervention	-0.7002351938	18
control	-0.7033881930	19
intervention	1.4959561507	19
control	-0.0405781737	20
intervention	-0.3742250181	20

Simulations

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


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

library(tidyr)
library(dplyr)
```

```
##
## Attache Paket: 'dplyr'
```

```
## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     filter, lag
```

```
## Die folgenden Objekte sind maskiert von 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(forcats)
library(readr)
library(purrr)
library(ggplot2)
library(effsize)
library(ggstance)
```

```
## Warning: Paket 'ggstance' wurde unter R Version 4.3.3 erstellt
```

```
##
## Attache Paket: 'ggstance'
```

```
## Die folgenden Objekte sind maskiert von 'package:ggplot2':
##
##     geom_errorbarh, GeomErrorbarh
```

```
library(scales)
```

```
##
## Attache Paket: 'scales'
```

```
## Das folgende Objekt ist maskiert 'package:purrr':
##
##     discard
```

```
## Das folgende Objekt ist maskiert 'package:readr':
##
##     col_factor
```

```
dir.create("plots")
```

```
## Warning in dir.create("plots"): 'plots' existiert bereits
```

```
# define data generating function ----
generate_data <- function(n_control,
                          n_intervention,
                          mean_control,
```

```
                          mean_intervention,
                          sd_control,
                          sd_intervention) {

  data <-
    bind_rows(
      tibble(condition = "control",
             score = rnorm(n = n_control, mean = mean_control, sd = sd_control)),
      tibble(condition = "intervention",
             score = rnorm(n = n_intervention, mean = mean_intervention, sd = sd_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")) |>
    # create a participant id column and then arrange by it, to facilitate sequential analysis
    group_by(condition) |>
    mutate(id = row_number(),
           unique_id = paste(id, condition, sep = "_")) |>
    ungroup() |>
    arrange(id)

  return(data)
}


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

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

  res_cohens_d <- effsize::cohen.d(formula = score ~ condition,
                                   within = FALSE,
                                   data = data)

  res <- tibble(p = res_t_test$p.value,
                cohens_d = res_cohens_d$estimate,
                cohens_d_ci_lower = res_cohens_d$conf.int["lower"],
                cohens_d_ci_upper = res_cohens_d$conf.int["upper"])

  return(res)
}

# # for loop solution
# analyse_data_sequential <- function(data, minimum_n_per_group, additional_n_per_group_per_step) {
#   # calculate the total number of iterations needed
#   n_per_group <- data |>
#     summarize(n_per_group = max(id)) |>
#     pull(n_per_group)
```

```
#
#   iter_count <- ceiling((n_per_group - minimum_n_per_group) / additional_n_per_group_per_step) + 1
#
#   # Initialize an empty data frame for results
#   results <- data.frame()
#
#   # Perform analyses iteratively
#   for (iter in 1:iter_count) {
#     # Calculate the row limit for the current iteration
#     highest_id <- min(minimum_n_per_group + (iter - 1) * additional_n_per_group_per_step, n_per_group
#
#     # Subset the data frame for the current iteration
#     data_subset <- data |>
#       filter(id <= highest_id)
#
#     # Perform the analysis on the subset
#     current_results <- analyse_data(data_subset) |>
#       mutate(sequential_analysis_n_per_group = highest_id)
#
#     # Combine the current results with the overall results
#     results <- rbind(results, current_results)
#   }
#
#   return(results)
# }

# purrr solution, for consistency
analyse_data_sequential <- function(data, minimum_n_per_group, additional_n_per_group_per_step) {
  # calculate the total number of iterations needed
  n_per_group <- data |>
    summarize(n_per_group = max(id)) |>
    pull(n_per_group)

  # create a sequence of n_per_group_current_analysis that defines the highest participant id value tha
  n_per_group_current_analysis <- seq(minimum_n_per_group, n_per_group, by = additional_n_per_group_per

  # if the last element of the n_per_group_current_analysis vector is not the total sample size, append
  # this is useful if the (total n per group - minimum_n_per_group) / additional_n_per_group_per_step h
  if (n_per_group_current_analysis[length(n_per_group_current_analysis)] != n_per_group) {
    n_per_group_current_analysis <- c(n_per_group_current_analysis, n_per_group)  # Ensure the last poi
  }

  analyze_by_subsset <- function(n_per_group_current_analysis) {
    data_subset <- data |>
      filter(id <= n_per_group_current_analysis)

    results_subset <- analyse_data(data_subset) |>
      mutate(sequential_analysis_n_per_group = n_per_group_current_analysis)

    return(results_subset)
  }

  # Use map_dfr to apply the analysis to each subset and combine the results
```

```
  results <- map_dfr(n_per_group_current_analysis, analyze_by_subsset)

  return(results)
}
```

Population null effect

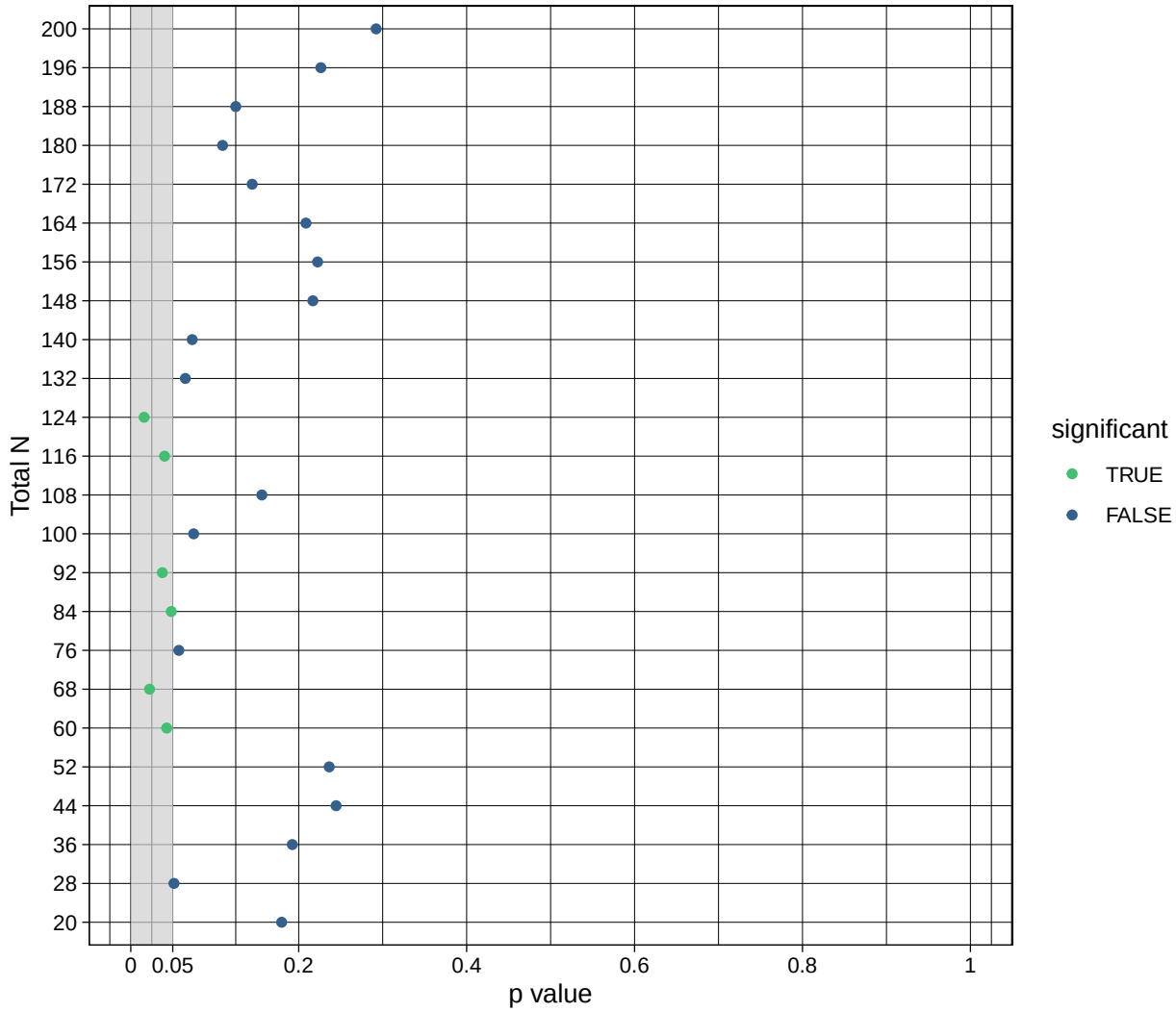
I.e., population Cohen's d = 0.0

```
# set the seed ----
# for the pseudo random number generator to make results reproducible
set.seed(47)

# define experiment parameters ----
experiment_parameters_grid_null <- expand_grid(
  n_control = 100,
  n_intervention = 100, # functions above assume n_control == n_intervention
  mean_control = 0,
  mean_intervention = 0,
  sd_control = 1,
  sd_intervention = 1,
  minimum_n_per_group = 10,
  additional_n_per_group_per_step = 4,
  iteration = 1:1 # note only one iteration as the "do it lots of time" part of this simulation is, aty
)


# run simulation ----
simulation_null <-
  # using the experiment parameters
  experiment_parameters_grid_null |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data = pmap(list(n_control,
                                    n_intervention,
                                    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,
                                      minimum_n_per_group,
                                      additional_n_per_group_per_step),
                                 analyse_data_sequential))


# summarise simulation results over the iterations ----
## estimate power
## ie what proportion of p values are significant (< .05)
simulation_summary_null <- simulation_null |>
  unnest(analysis_results) |>
  mutate(cohens_d_centered = cohens_d - cohens_d,
```

```
          cohens_d_ci_lower_centered = cohens_d_ci_lower - cohens_d,
          cohens_d_ci_upper_centered = cohens_d_ci_upper - cohens_d,
          significant = p < .05)
```

Dance of the p values

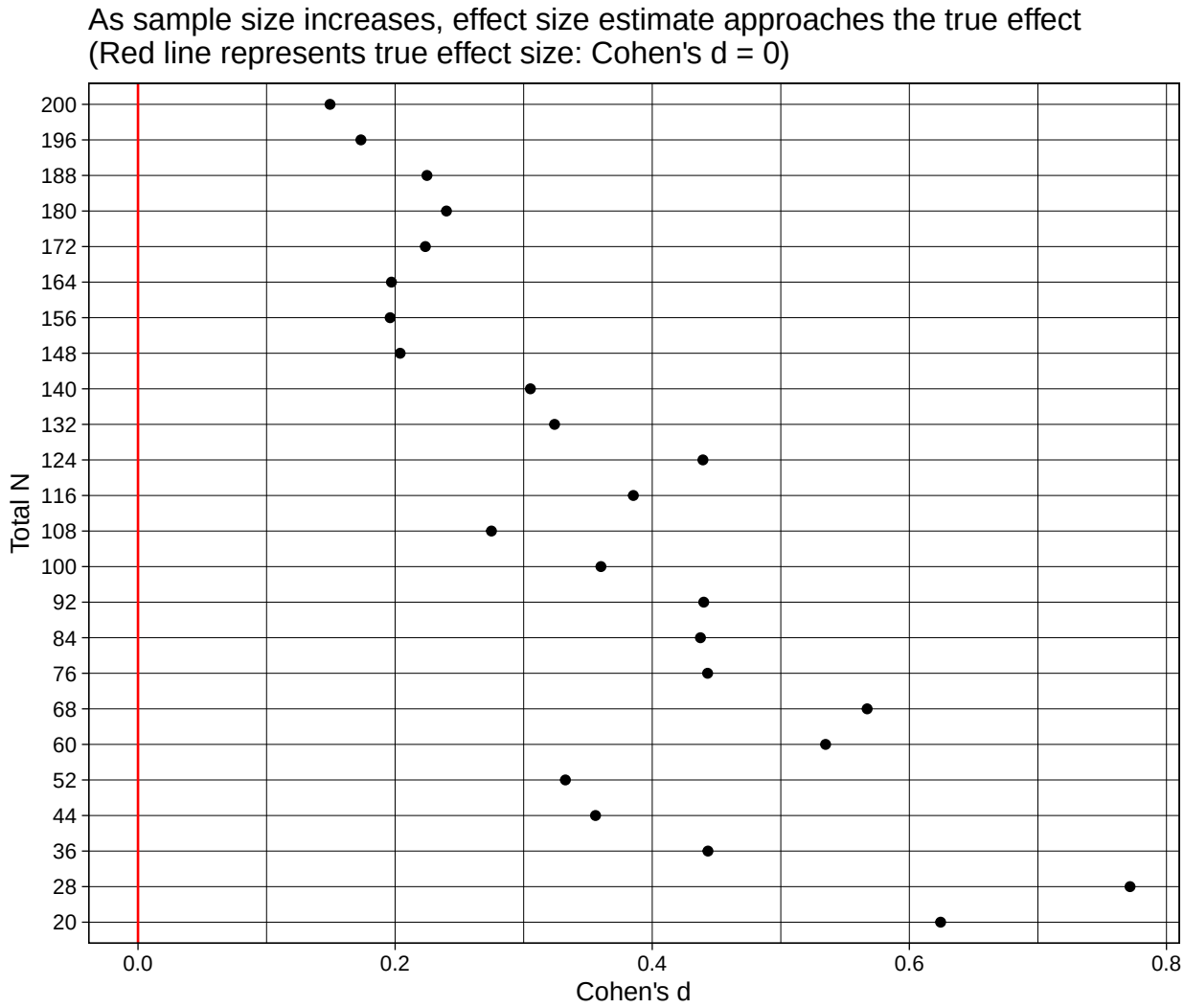
```
p_dance_of_p_values_null <-
  ggplot(simulation_summary_null, aes(p, as.factor(sequential_analysis_n_per_group*2), color = significa
  geom_rect(aes(xmin = 0, xmax = 0.05, ymin = -Inf, ymax = Inf), fill = "grey85", alpha = 0.05, color =
  geom_point(position = position_dodge(width = 0.3)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = c(0, 0.05, 0.2, 0.4, 0.6, 0.8, 1),
                     labels = c(0, 0.05, 0.2, 0.4, 0.6, 0.8, 1)) +
  coord_cartesian(xlim = c(0, 1)) +
  theme_linedraw() +
  xlab("p value") +
  ylab("Total N")

p_dance_of_p_values_null
```



```r
# ggsave(filename = "p_dance_of_p_values_null.pdf",
#        plot = p_dance_of_p_values_null,
#        path = "plots/",
#        device = "pdf",
#        width = 7,
#        height = 6)
```

Dance of the effect size estimates

```r
p_dance_of_effect_sizes_null <-
  ggplot(simulation_summary_null, aes(cohens_d, as.factor(sequential_analysis_n_per_group*2))) +
  geom_vline(xintercept = 0, color = "red") +
  #geom_linerangeh(aes(xmin = cohens_d_ci_lower, xmax = cohens_d_ci_upper), position = position_dodge(w
  geom_point(position = position_dodge(width = 0.3)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = pretty_breaks()) +
  theme_linedraw() +
  xlab("Cohen's d") +
  ylab("Total N") +
  ggtitle("As sample size increases, effect size estimate approaches the true effect\n(Red line represe

p_dance_of_effect_sizes_null
```



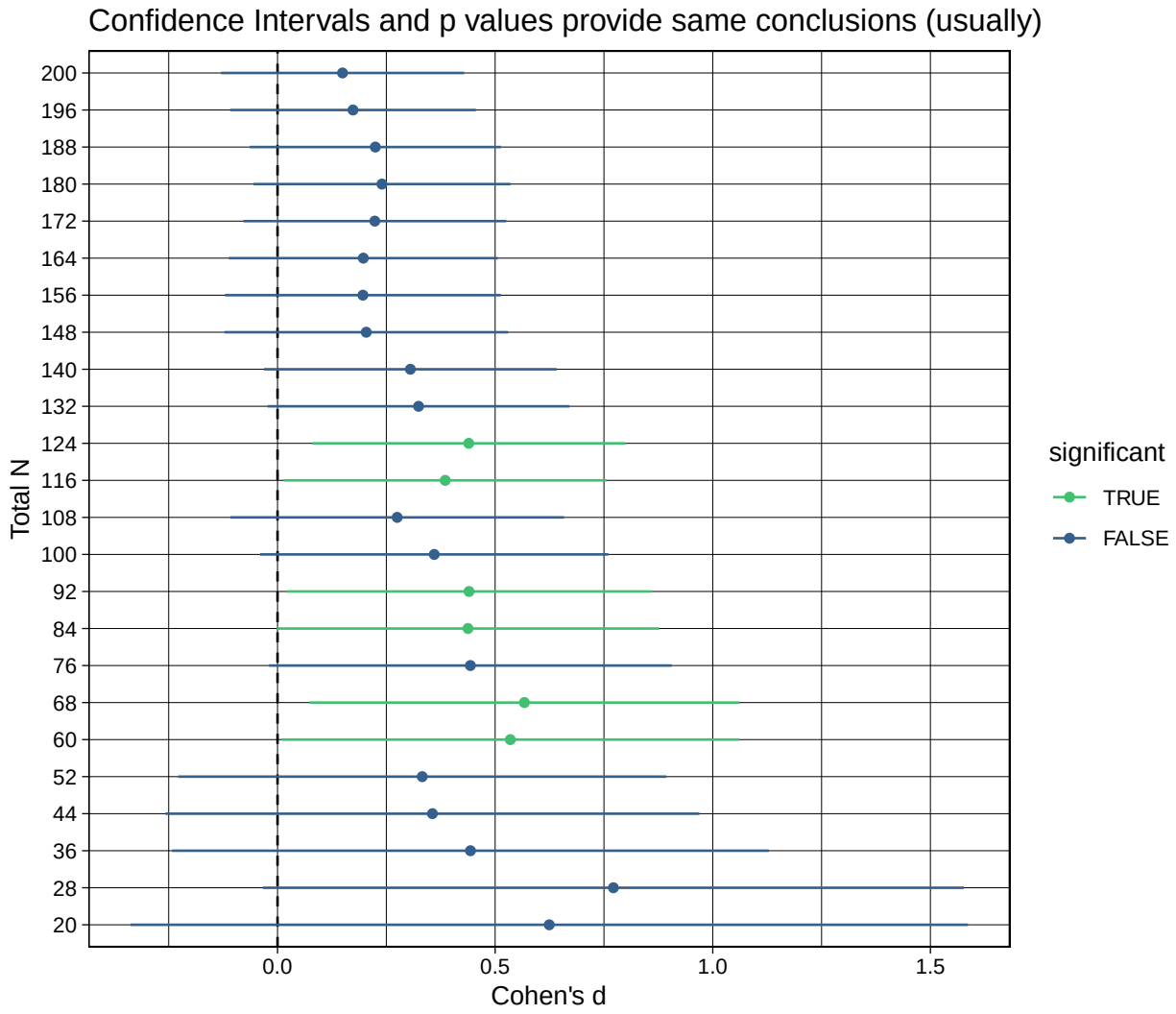

```
# ggsave(filename = "p_dance_of_effect_sizes_null.pdf",
#        plot = p_dance_of_effect_sizes_null,
#        path = "plots/",
#        device = "pdf",
#        width = 7,
#        height = 6)
```

Confidence intervals and p values

```
p_confidence_intervals_p_values_null <-
  ggplot(simulation_summary_null, aes(cohens_d, as.factor(sequential_analysis_n_per_group*2), color = si
  geom_vline(xintercept = 0, linetype = "dashed") +
  geom_linerangeh(aes(xmin = cohens_d_ci_lower, xmax = cohens_d_ci_upper), position = position_dodge(wi
  geom_point(position = position_dodge(width = 0.3)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = pretty_breaks()) +
  theme_linedraw() +
  xlab("Cohen's d") +
  ylab("Total N") +
```

```
  ggtitle("Confidence Intervals and p values provide same conclusions (usually)")

p_confidence_intervals_p_values_null
```

```
## Warning: Using the `size` aesthetic with geom_segment was deprecated in ggplot2 3.4.0.
## i Please use the `linewidth` aesthetic instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```
# ggsave(filename = "p_confidence_intervals_p_values_null.pdf",
#        plot = p_confidence_intervals_p_values_null,
#        path = "plots/",
#        device = "pdf",
#        width = 7,
#        height = 6)
```

- Notice how conclusions change as additional data is collected. The estimate of Cohen's d, and its confidence intervals also become narrower. The effect goes from being statistically non-significant, to significant, back and forth (although note that I have chosen a seed value that produces particularly

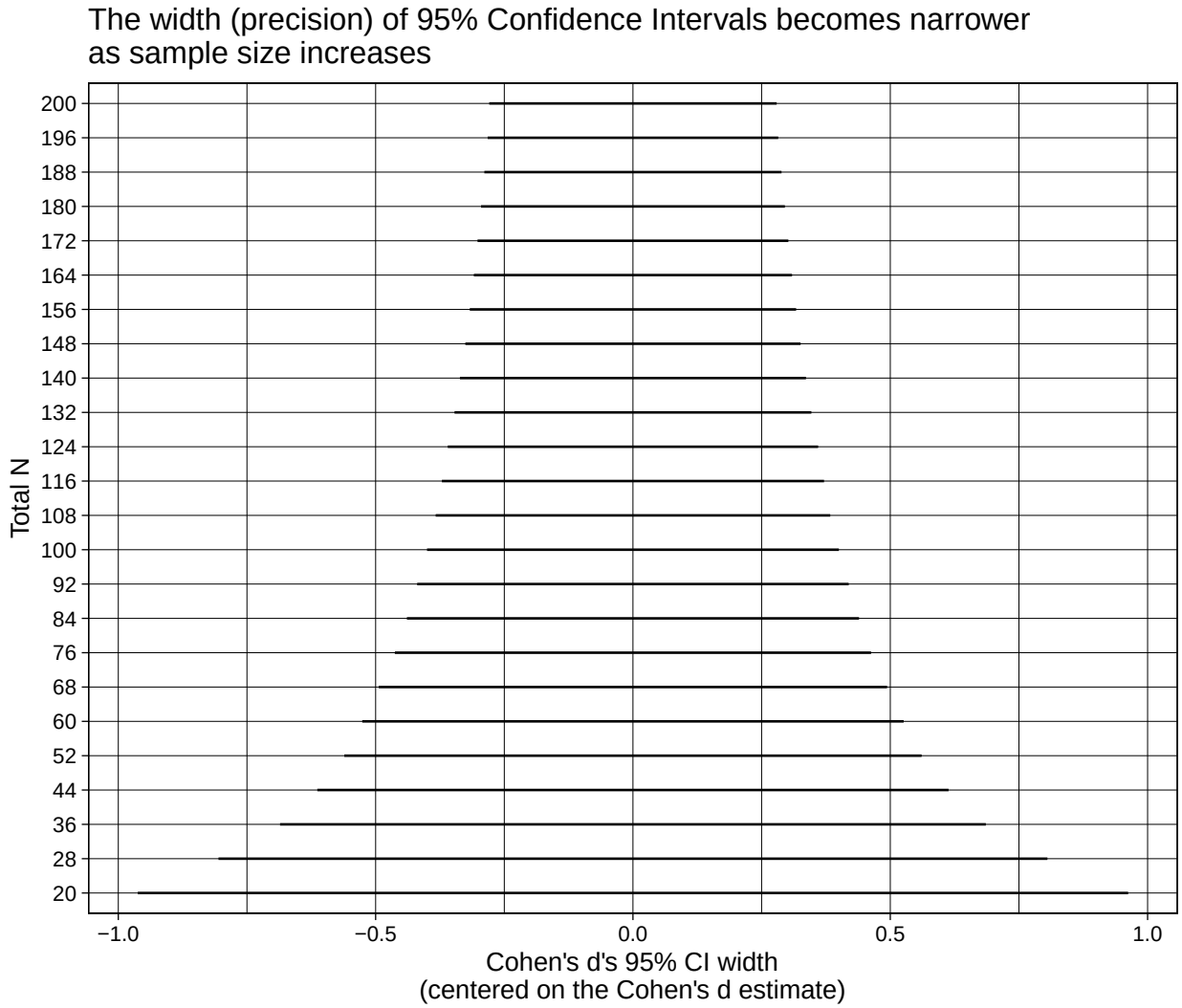
extreme results here).

- This is why "optional stopping", or selecting determining when to stop collecting data, usually based on obtaining significant results, is problematic and a form of p-hacking. If this was a real study and the authors stopped collecting the moment they hit statistical significance (at n = 60) they might have a more publishable result, but it would be the incorrect conclusion: the population effect is in fact null, and these significant results are just random noise.
- This is why determining sample size ahead of time (e.g., via power analysis using the Smallest Effect Size of Interest) and holding ourselves to this sample size via preregistration are important in order to control false-positive rates.

Effect size estimation precision increases with sample size

Which is another way of saying that statistical power increases with sample size. This is more obvious when the population effect size is non-zero (see furhter below).

```
p_effect_size_interval_width_null <-
  ggplot(simulation_summary_null, aes(cohens_d_centered, as.factor(sequential_analysis_n_per_group*2)))
  geom_linerangeh(aes(xmin = cohens_d_ci_lower_centered, xmax = cohens_d_ci_upper_centered), position =
  #geom_point(position = position_dodge(width = 0.3)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = pretty_breaks()) +
  theme_linedraw() +
  xlab("Cohen's d's 95% CI width\n(centered on the Cohen's d estimate)") +
  ylab("Total N") +
  ggtitle("The width (precision) of 95% Confidence Intervals becomes narrower\nas sample size increases

p_effect_size_interval_width_null
```




```
# ggsave(filename = "p_effect_size_interval_width_null.pdf",
#        plot = p_effect_size_interval_width_null,
#        path = "plots/",
#        device = "pdf",
#        width = 6,
#        height = 6)
```

Population non-null effect

I.e., population Cohen's d = 0.1 (very small, but real)

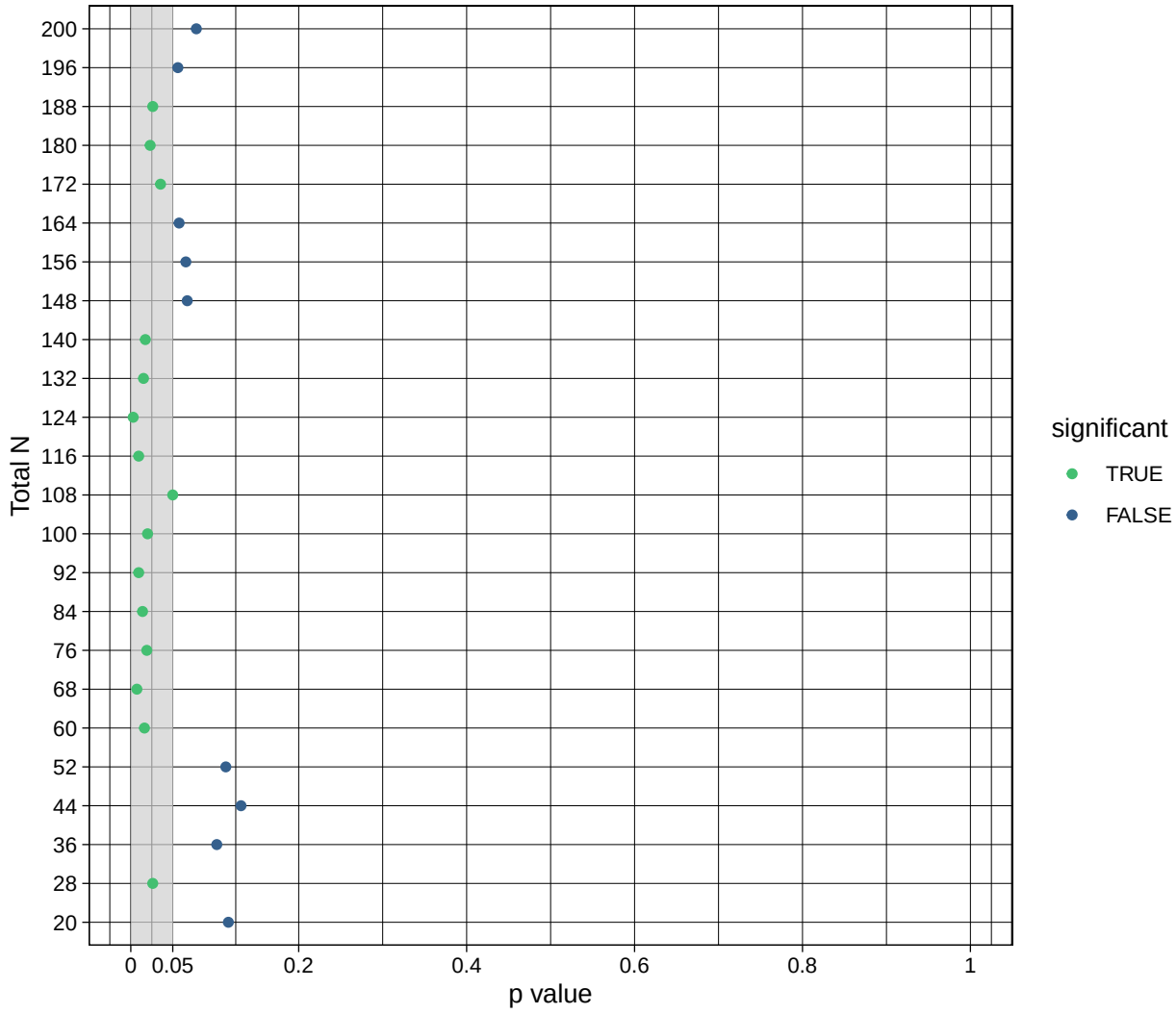
```
# set the seed ----
# for the pseudo random number generator to make results reproducible
set.seed(47)

# define experiment parameters ----
population_es <- 0.1 # used later for plotting

experiment_parameters_grid_true <- expand_grid(
  n_control = 100,
```

```
  n_intervention = 100, # functions above assume n_control == n_intervention
  mean_control = 0,
  mean_intervention = population_es,
  sd_control = 1,
  sd_intervention = 1,
  minimum_n_per_group = 10,
  additional_n_per_group_per_step = 4,
  iteration = 1:1 # note only one iteration as the "do it lots of time" part of this simulation is, aty
)


# run simulation ----
simulation_true <-
  # using the experiment parameters
  experiment_parameters_grid_true |>

  # generate data using the data generating function and the parameters relevant to data generation
  mutate(generated_data = pmap(list(n_control,
                                    n_intervention,
                                    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,
                                      minimum_n_per_group,
                                      additional_n_per_group_per_step),
                                 analyse_data_sequential))


# summarise simulation results over the iterations ----
## estimate power
## ie what proportion of p values are significant (< .05)
simulation_summary_true <- simulation_true |>
  unnest(analysis_results) |>
  mutate(cohens_d_centered = cohens_d - cohens_d,
         cohens_d_ci_lower_centered = cohens_d_ci_lower - cohens_d,
         cohens_d_ci_upper_centered = cohens_d_ci_upper - cohens_d,
         significant = p < .05)
```

Dance of the p values

```
p_dance_of_p_values_true <-
  ggplot(simulation_summary_true, aes(p, as.factor(sequential_analysis_n_per_group*2), color = significa
  geom_rect(aes(xmin = 0, xmax = 0.05, ymin = -Inf, ymax = Inf), fill = "grey85", alpha = 0.05, color =
  geom_point(position = position_dodge(width = 0.3)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = c(0, 0.05, 0.2, 0.4, 0.6, 0.8, 1),
                     labels = c(0, 0.05, 0.2, 0.4, 0.6, 0.8, 1)) +
  coord_cartesian(xlim = c(0, 1)) +
  theme_linedraw() +
```

```
  xlab("p value") +
  ylab("Total N")

p_dance_of_p_values_true
```

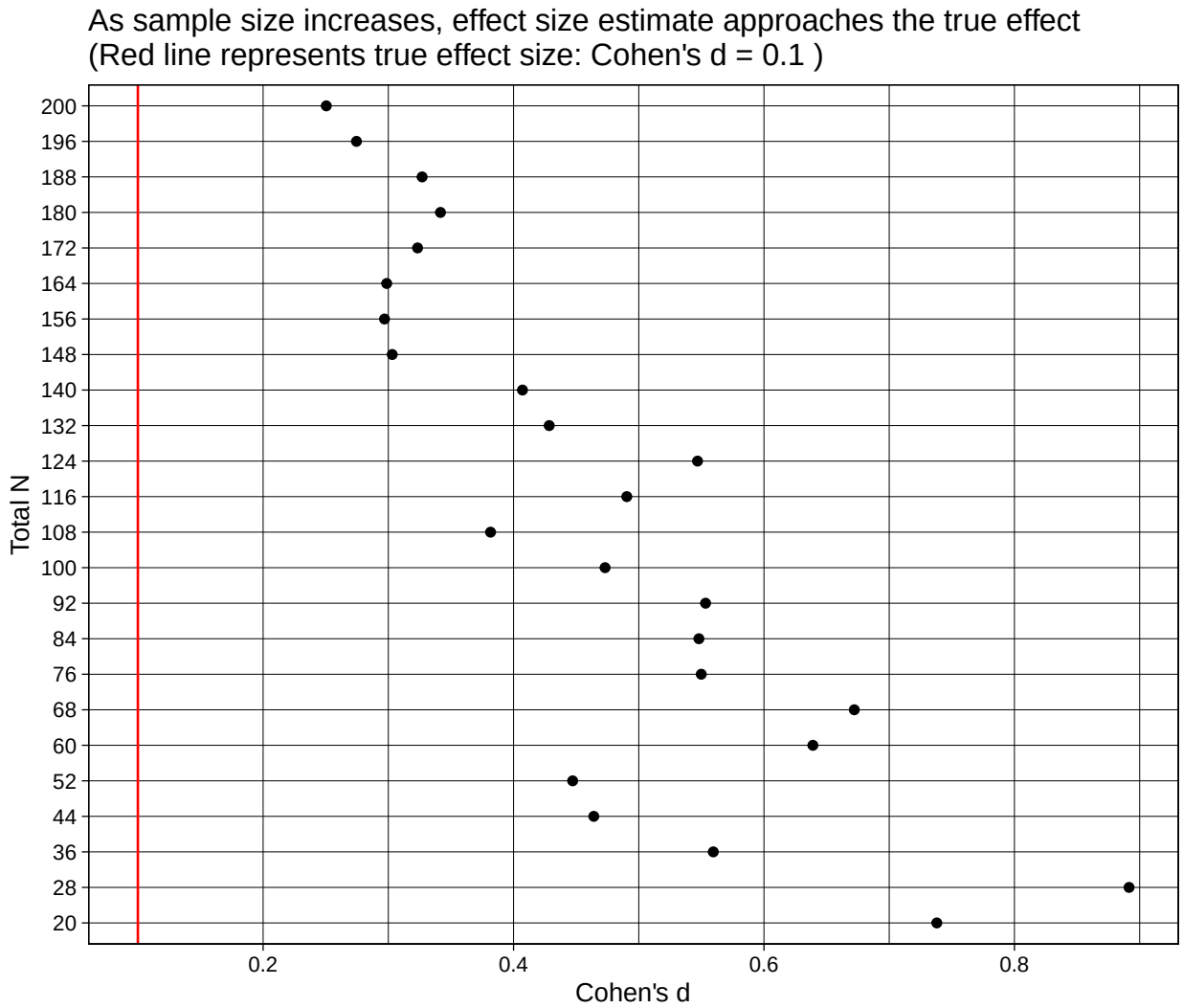



```
# ggsave(filename = "p_dance_of_p_values_true.pdf",
#        plot = p_dance_of_p_values_true,
#        path = "plots/",
#        device = "pdf",
#        width = 7,
#        height = 6)
```

Dance of the effect size estimates

```
p_dance_of_effect_sizes_true <-
  ggplot(simulation_summary_true, aes(cohens_d, as.factor(sequential_analysis_n_per_group*2))) +
  geom_vline(xintercept = population_es, color = "red") +
  #geom_linerangeh(aes(xmin = cohens_d_ci_lower, xmax = cohens_d_ci_upper), position = position_dodge(w
  geom_point(position = position_dodge(width = 0.3)) +
```

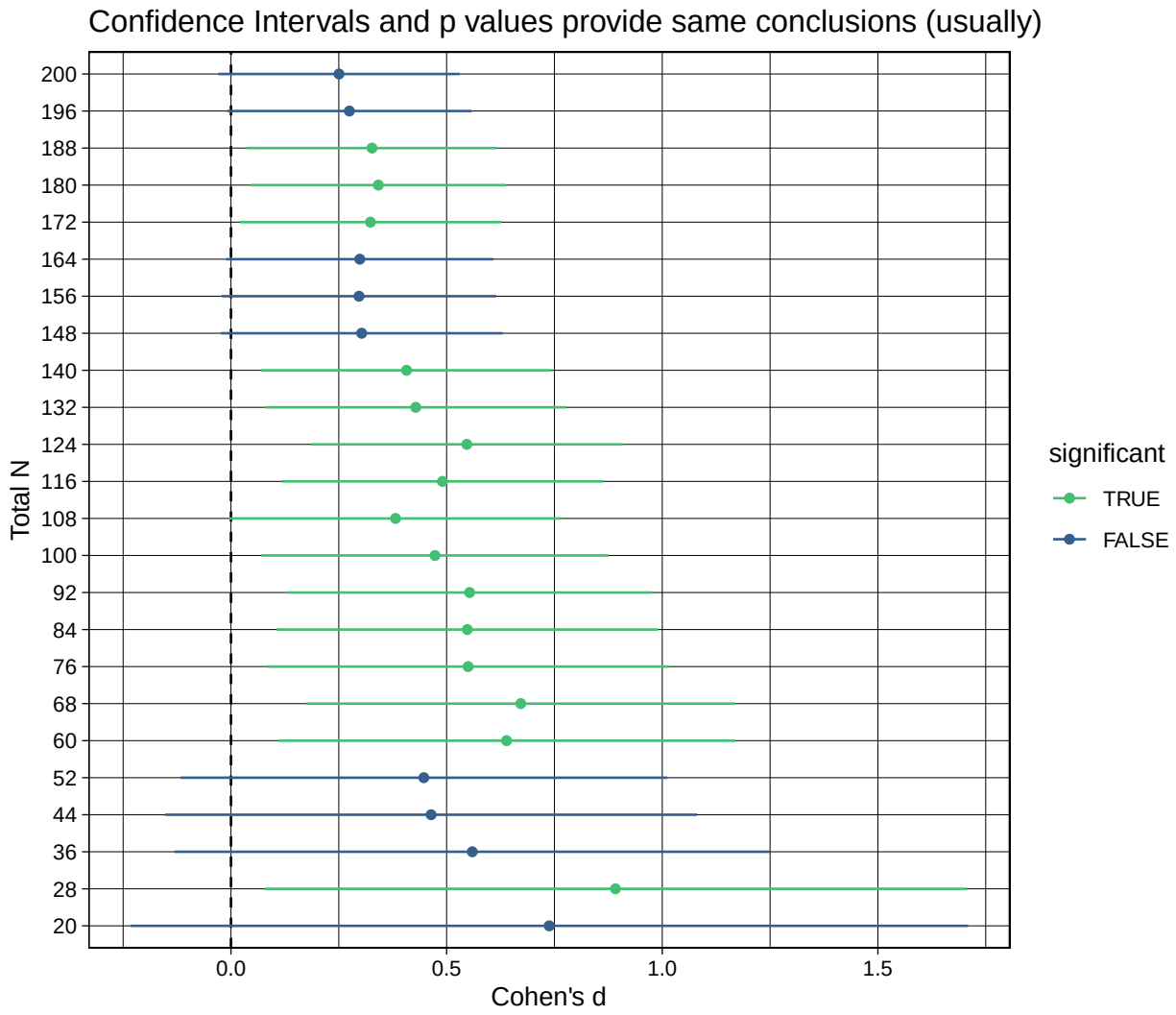
```
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = pretty_breaks()) +
  theme_linedraw() +
  xlab("Cohen's d") +
  ylab("Total N") +
  ggtitle(paste("As sample size increases, effect size estimate approaches the true effect\n(Red line re

p_dance_of_effect_sizes_true
```




```
# ggsave(filename = "p_dance_of_effect_sizes_true.pdf",
#        plot = p_dance_of_effect_sizes_true,
#        path = "plots/",
#        device = "pdf",
#        width = 7,
#        height = 6)
```

Confidence intervals and p values

```
p_confidence_intervals_p_values_true <-
  ggplot(simulation_summary_true, aes(cohens_d, as.factor(sequential_analysis_n_per_group*2), color = si
  geom_vline(xintercept = 0, linetype = "dashed") +
  geom_linerangeh(aes(xmin = cohens_d_ci_lower, xmax = cohens_d_ci_upper), position = position_dodge(wi
  geom_point(position = position_dodge(width = 0.3)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = pretty_breaks()) +
  theme_linedraw() +
  xlab("Cohen's d") +
  ylab("Total N") +
  ggtitle("Confidence Intervals and p values provide same conclusions (usually)")

p_confidence_intervals_p_values_true
```



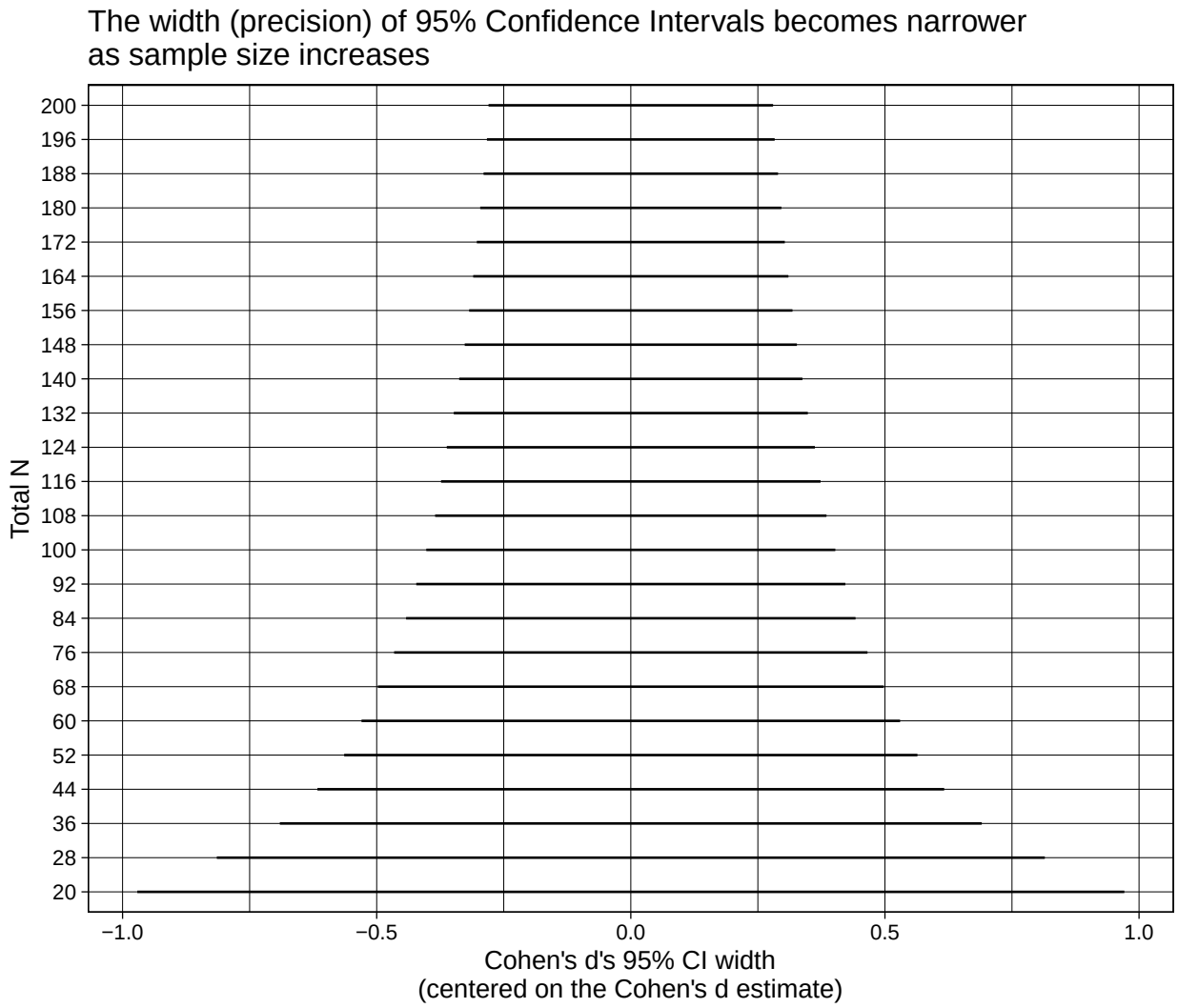

```
# ggsave(filename = "p_confidence_intervals_p_values_true.pdf",
#        plot = p_confidence_intervals_p_values_true,
#        path = "plots/",
#        device = "pdf",
```

```
#         width = 7,
#         height = 6)
```

Effect size estimation precision increases with sample size

Which is another way of saying that statistical power increases with sample size.

```
p_effect_size_interval_width_true <-
  ggplot(simulation_summary_true, aes(cohens_d_centered, as.factor(sequential_analysis_n_per_group*2)))
  geom_linerangeh(aes(xmin = cohens_d_ci_lower_centered, xmax = cohens_d_ci_upper_centered), position =
  #geom_point(position = position_dodge(width = 0.3)) +
  scale_color_viridis_d(begin = 0.3, end = 0.7, guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = pretty_breaks()) +
  theme_linedraw() +
  xlab("Cohen's d's 95% CI width\n(centered on the Cohen's d estimate)") +
  ylab("Total N") +
  ggtitle("The width (precision) of 95% Confidence Intervals becomes narrower\nas sample size increases

p_effect_size_interval_width_true
```



```
# ggsave(filename = "p_effect_size_interval_width_true.pdf",
#        plot = p_effect_size_interval_width_true,
#        path = "plots/",
#        device = "pdf",
#        width = 6,
#        height = 6)
```

This plot purposefully ignores what the actual Cohen's d estimate is in each analysis, to focus instead on what the width of its confidence intervals are. Confidence intervals and p values are - putting aside some issues of how they're implemented - intended to be a restatement of the same information about long run probabilities, and should (usually) produce the same conclusions.

Notice how the 95% CI widths narrow as sample size increases. Consider this in terms of statistical power: if the population effect size was Cohen's d = .50, the plot suggests that sample sizes less than 68 are unlikely to detect it, because even if the Cohen's d estimate was perfectly correct the 95% CIs would still overlap zero (i.e., be non-significant). Conversely, at larger sample sizes, it becomes easier and easier for an effect size that is truly non-zero to be detectably non-zero, as the 95% CIs are narrower.

Separately, use this plot to think back to the interpretation of non-significant p values. Remember the phrase "absence of evidence does not equal evidence of absence"? This plot helps explain why we cannot interpret non-significant p values as evidence of equivalence: when the 95% CI is very wide, a very large range of Cohen's ds are all compatible with the results. E.g., when N = 20, values of Cohen's d that are the estimate ± 1.0 are all equally compatible. If the estimate was 0.10, the population value is estimated to be in the range [-.90, +1.10], or anywhere between a "large negative effect" and a "large positive effect". The non-significance of this result tells us little, because non-results were very likely due to low statistical power, aka poor precision. Equally, a non-significant p values when the 95% CI is ± 0.1 (when the sample size is much higher) tells us a lot more than when it is ± 1.0. So, non-significant p values by themselves cannot tell us about equivalences.

Try other simulation parameters and seeds

Note that I have chosen values for `set.seed` that produce particularly extreme results that dance back and forth. Try changing the seed values and the parameter estimates (e.g., inrease the sample size and/or population effect sizes) and observe the results.

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] scales_1.3.0   ggstance_0.3.7 effsize_0.8.1  ggplot2_3.5.1  purrr_1.0.2
## [6] readr_2.1.5    forcats_1.0.0  dplyr_1.1.4    tidyr_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.5      crayon_1.5.2      compiler_4.3.2    highr_0.10
##  [5] tidyselect_1.2.1  yaml_2.3.8        fastmap_1.1.1     R6_2.5.1
##  [9] generics_0.1.3    knitr_1.46        tibble_3.2.1      munsell_0.5.1
## [13] pillar_1.9.0      tzdb_0.4.0        rlang_1.1.3       utf8_1.2.4
## [17] xfun_0.43         viridisLite_0.4.2 cli_3.6.2         withr_3.0.0
## [21] magrittr_2.0.3    digest_0.6.35     grid_4.3.2        rstudioapi_0.16.0
## [25] hms_1.1.3         lifecycle_1.0.4   vctrs_0.6.5       evaluate_0.23
## [29] glue_1.7.0        farver_2.1.1      fansi_1.0.6       colorspace_2.1-0
## [33] rmarkdown_2.26    tools_4.3.2       pkgconfig_2.0.3   htmltools_0.5.8.1
```