Understanding p values using a simulations

Using the example of Welch's independent t-test

Ian Hussey

06 Mai, 2024

Contents

Overview of tutorial

This lesson focuses on the distributions of p values "under the null hypothesis" (i.e., when the null hypothesis is true, aka when the data generating process is a population model where the effect is zero) and "under the alternative hypothesis" (i.e., when the alternative hypothesis is true, aka when the data generating process is a population model where the effect is non-zero).

Without looking into the math, let's use simulations to visualise these distributions, in order to understand p values better.

Plotting distributions

In the first lesson, we refreshed our knowledge of distributions and used ggplot to visualise them.

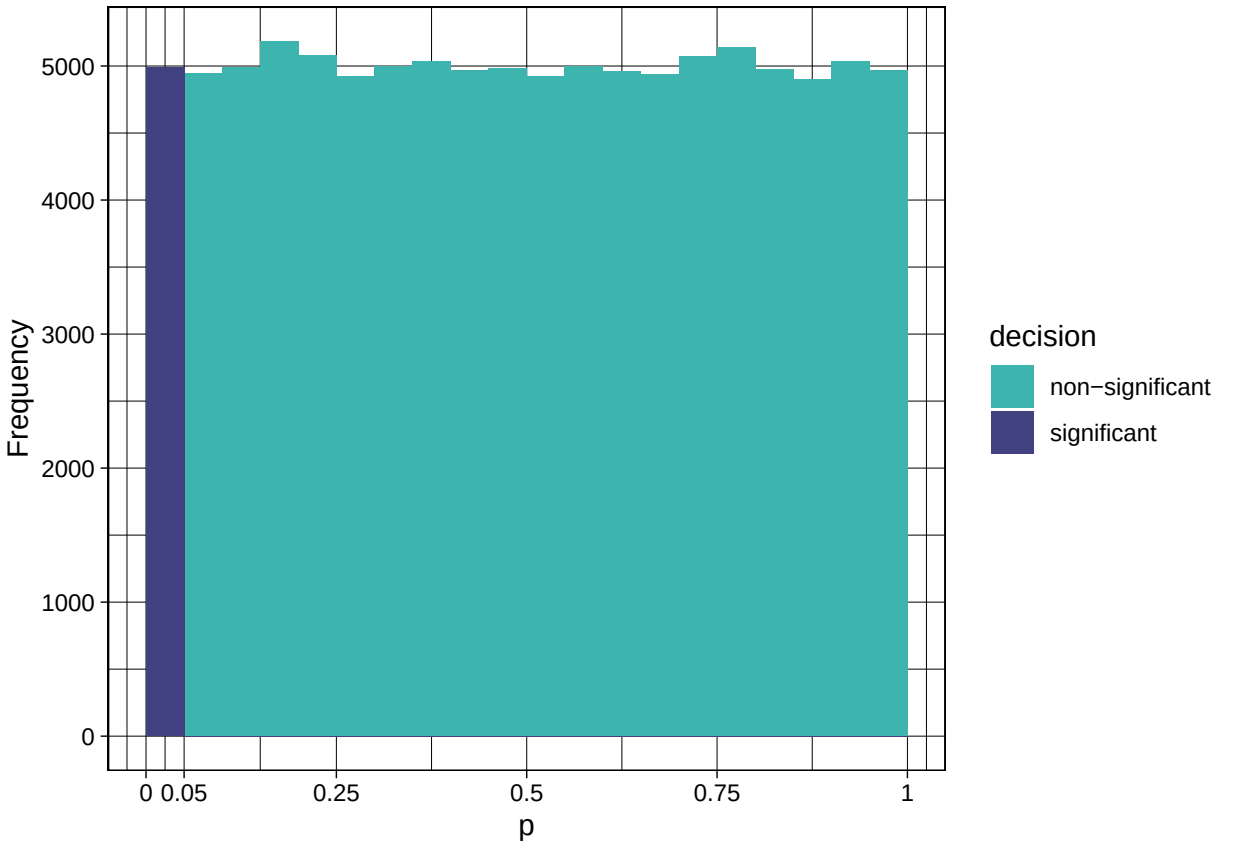
We covered the uniform distribution, where all values between a given range are equally probable, and the normal distribution, which follows a normal (gaussian) bell-shaped distribution.

Plot uniform distribution

```
library(tidyverse)
```

```
## Warning: Paket 'tibble' wurde unter R Version 4.3.3 erstellt
## Warning: Paket 'tidyr' wurde unter R Version 4.3.3 erstellt
## Warning: Paket 'purrr' wurde unter R Version 4.3.3 erstellt
## Warning: Paket 'dplyr' wurde unter R Version 4.3.3 erstellt
## Warning: Paket 'stringr' wurde unter R Version 4.3.3 erstellt
## Warning: Paket 'forcats' wurde unter R Version 4.3.3 erstellt
## Warning: Paket 'lubridate' wurde unter R Version 4.3.3 erstellt
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# sample values from a uniform distribution, from the range 0 to 1
tibble(p = runif(n = 100000, min = 0, max = 1)) |>
  # create a decision for each value, with values < .05 labelled as "significant" and those > .05 as "n
  mutate(decision = ifelse(p < .05, "significant", "non-significant")) |>
  # plot a histogram of these values, with the fill contingent on the the decision
  ggplot(aes(p, fill = decision)) +
  geom_histogram(binwidth = 0.05, boundary = 0) +
  scale_fill_viridis_d(option = "mako", begin = 0.3, end = 0.7, direction = -1) +
  scale_x_continuous(labels = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     breaks = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     limits = c(0, 1)) +
  theme_linedraw() +
  ylab("Frequency")
```



Generate analyse iterate aggregate

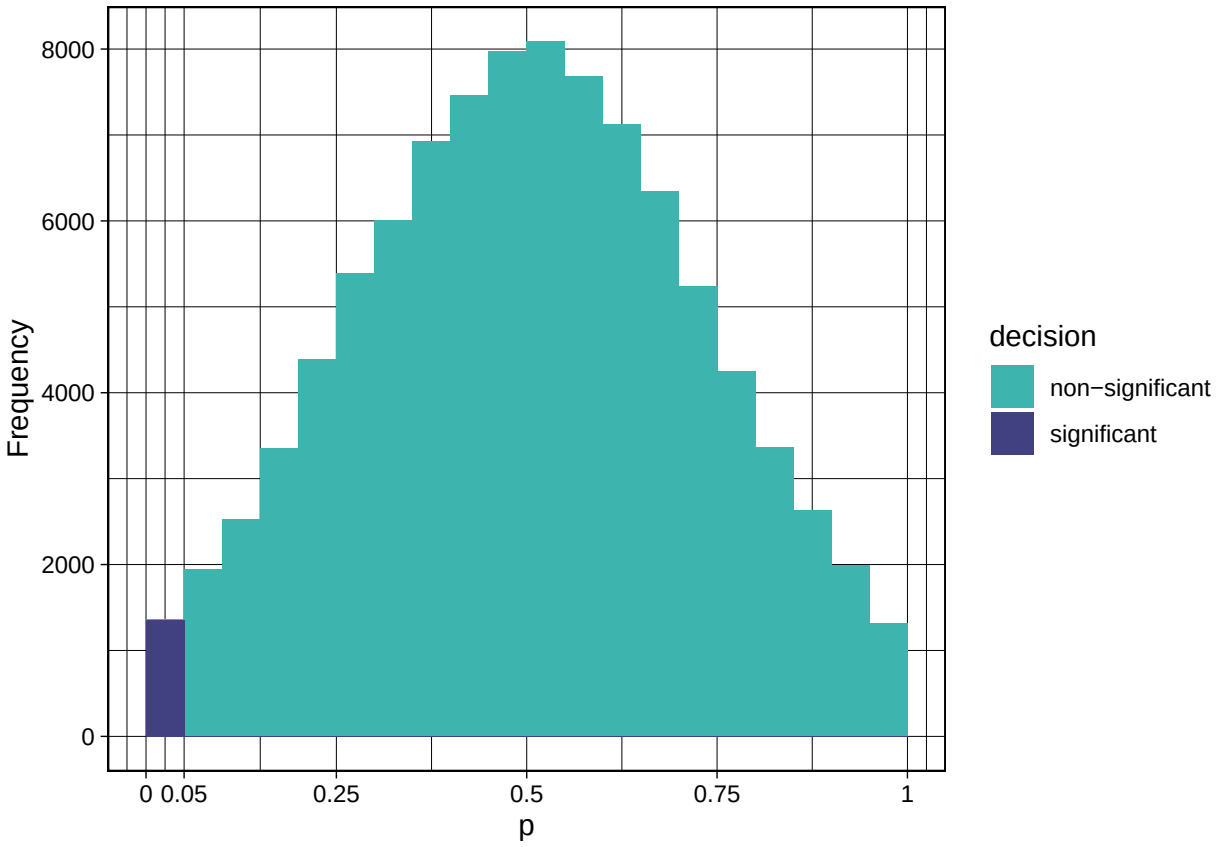
```
n     <-
mean  <-
sd    <-


my_functionggplot <- function(n, mean, sd){

  # sample values from a uniform distribution, from the range 0 to 1
plot <- tibble(p = runif(n = 100000, min = 0, max = 1)) |> # this is the input
  # create a decision for each value, with values < .05 labelled as "significant" and those > .05 as "n
  mutate(decision = ifelse(p < .05, "significant", "non-significant")) |>
  # plot a histogram of these values, with the fill contingent on the the decision
  ggplot(aes(p, fill = decision)) +
  geom_histogram(binwidth = 0.05, boundary = 0) +
  scale_fill_viridis_d(option = "mako", begin = 0.3, end = 0.7, direction = -1) +
  scale_x_continuous(labels = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     breaks = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     limits = c(0, 1)) +
  theme_linedraw() +
  ylab("Frequency")

 return(plot) # this is the output
}
```

- Note that this plot labels the x axis as p values, but the results are not actually p values yet: it's just data drawn from a uniform distribution.

Plot normal distribution

With the constraint that all values must be within the range [0, 1] to mimic p values.

```
# sample values from a uniform distribution, from the range 0 to 1
tibble(p = rnorm(n = 100000, mean = 0.5, sd = 0.25)) |>
  # drop all values that are outside the range [0, 1] to mimic p values
  filter(p >= 0 & p <= 1) |>
  # create a decision for each value, with values < .05 labelled as "significant" and those > .05 as "n
  mutate(decision = ifelse(p < .05, "significant", "non-significant")) |>
  # plot a histogram of these values, with the fill contingent on the the decision
  ggplot(aes(p, fill = decision)) +
  geom_histogram(binwidth = 0.05, boundary = 0) +
  scale_fill_viridis_d(option = "mako", begin = 0.3, end = 0.7, direction = -1) +
  scale_x_continuous(labels = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     breaks = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     limits = c(0, 1)) +
  theme_linedraw() +
  ylab("Frequency")
```



- Again, note that this plot labels the x axis as p values, but the results are not actually p values yet: it's just data drawn from a normal distribution.

Coding rule: Don't repeat yourself / practice writing functions

A key rule when writing code is, in general, *don't repeat yourself*.

If I want to change the way the above plots are made, or make new ones, I would need to change the code for

each one. Equally, if I want do make several more plots like this I would need to repeat the code many times. Every time you copy and paste code like this, it's another opportunity to make a mistake, eg. to forget to update a parameter somewhere. A good way to avoid this is to write a function to do it for you, and call this function as you need it.

Let's use this as an opportunity to practice writing functions, as you'll need to get good at this for the data generation and analysis functions you use in simulations anyway.

Let's assume that the simulated data is already generated, and that we just want to plot it. I therefore move all the code for plotting inside the new function, and pass data to it via the `data` parameter.

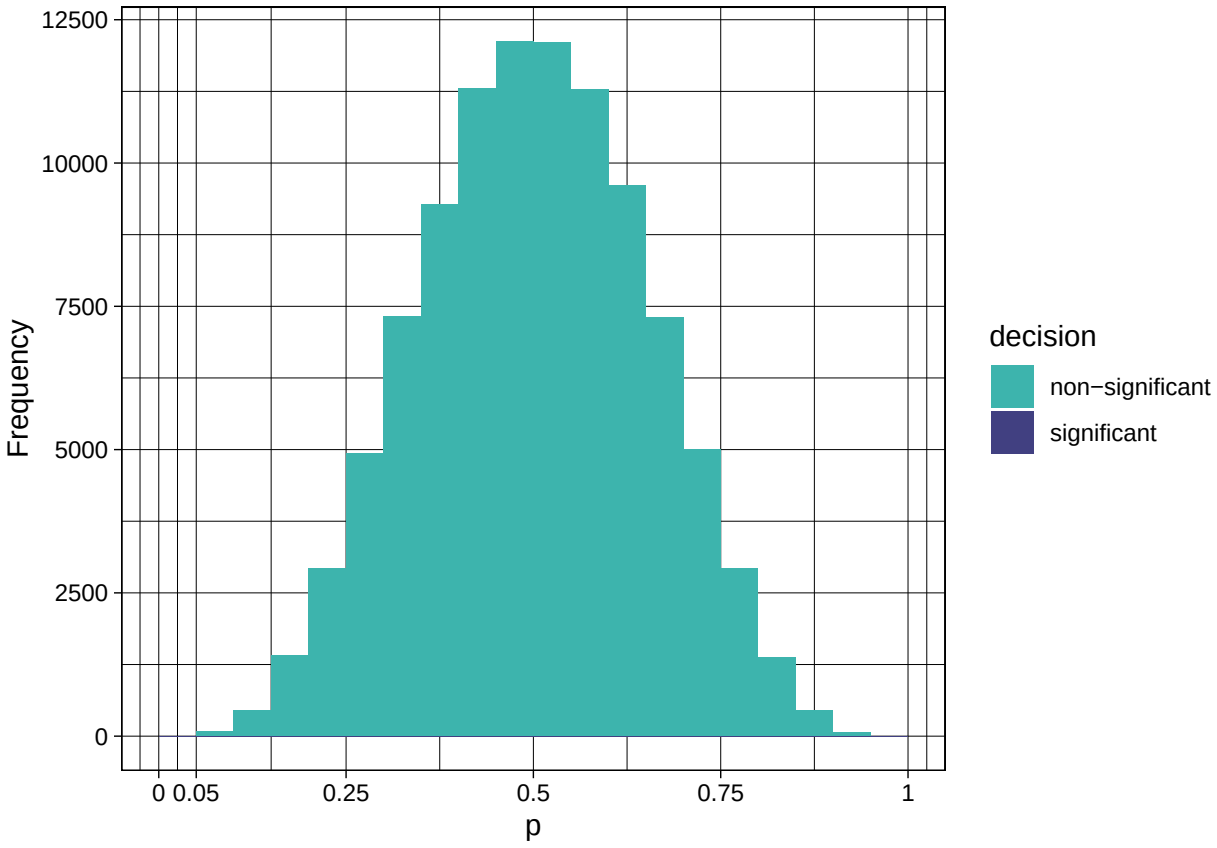
```
plot_p_values <- function(data){ # assumes that data is a data frame with a column "p"
  data |>
    mutate(decision = ifelse(p < .05, "significant", "non-significant")) |>
    ggplot(aes(p, fill = decision)) +
    geom_histogram(binwidth = 0.05, boundary = 0) +
    scale_fill_viridis_d(option = "mako", begin = 0.3, end = 0.7, direction = -1) +
    scale_x_continuous(labels = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                       breaks = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                       limits = c(0, 1)) +
    theme_linedraw() +
    ylab("Frequency")
}
```

We can then use the function to plot any suitable data, as long as it (a) has a variable named `p` and (b) has some values between 0 and 1.

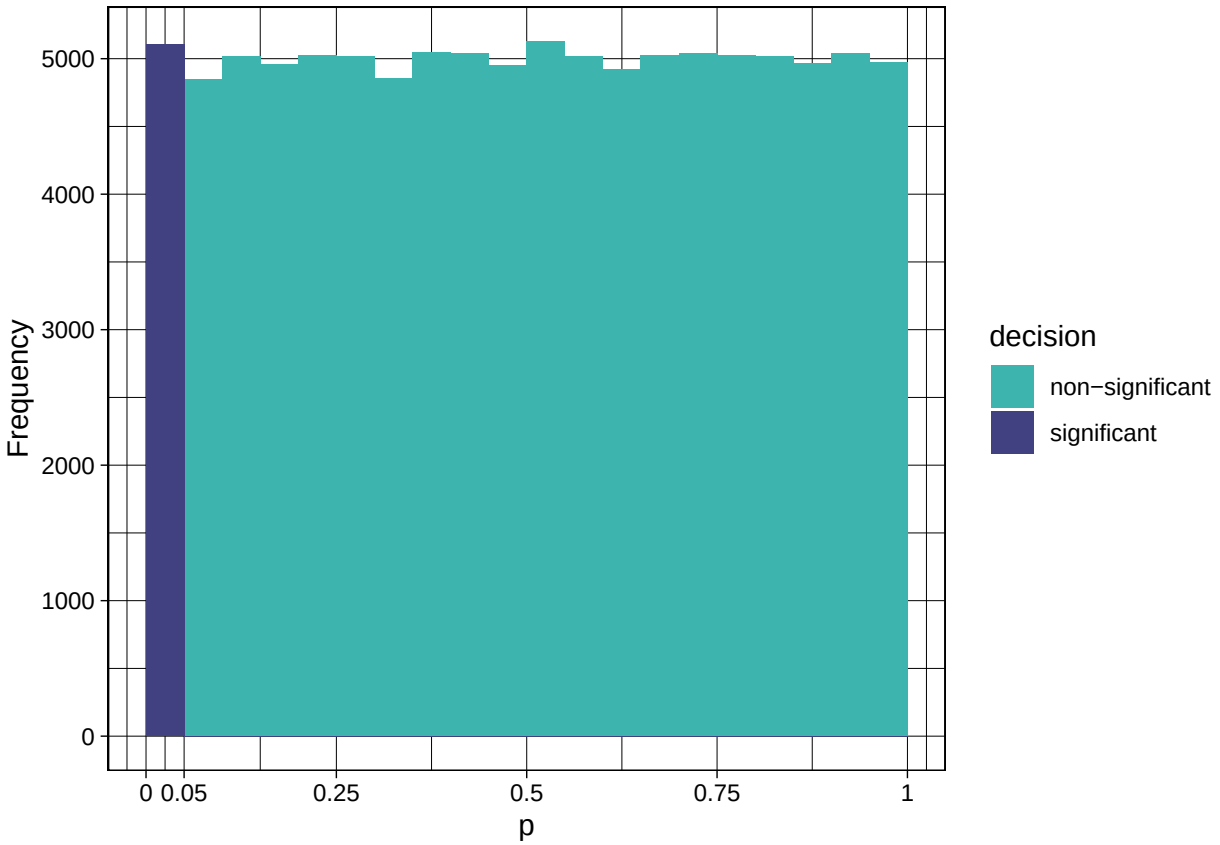
We can even plot data from other distributions, such as the Beta distribution. The beta distribution can be useful to create fake p-values, as it is bounded [0, 1] and can generate data that is anywhere between uniform, normal, or skewed depending on the values you use for its α (`shape1`) and β (`shape2`) parameters.

For example:

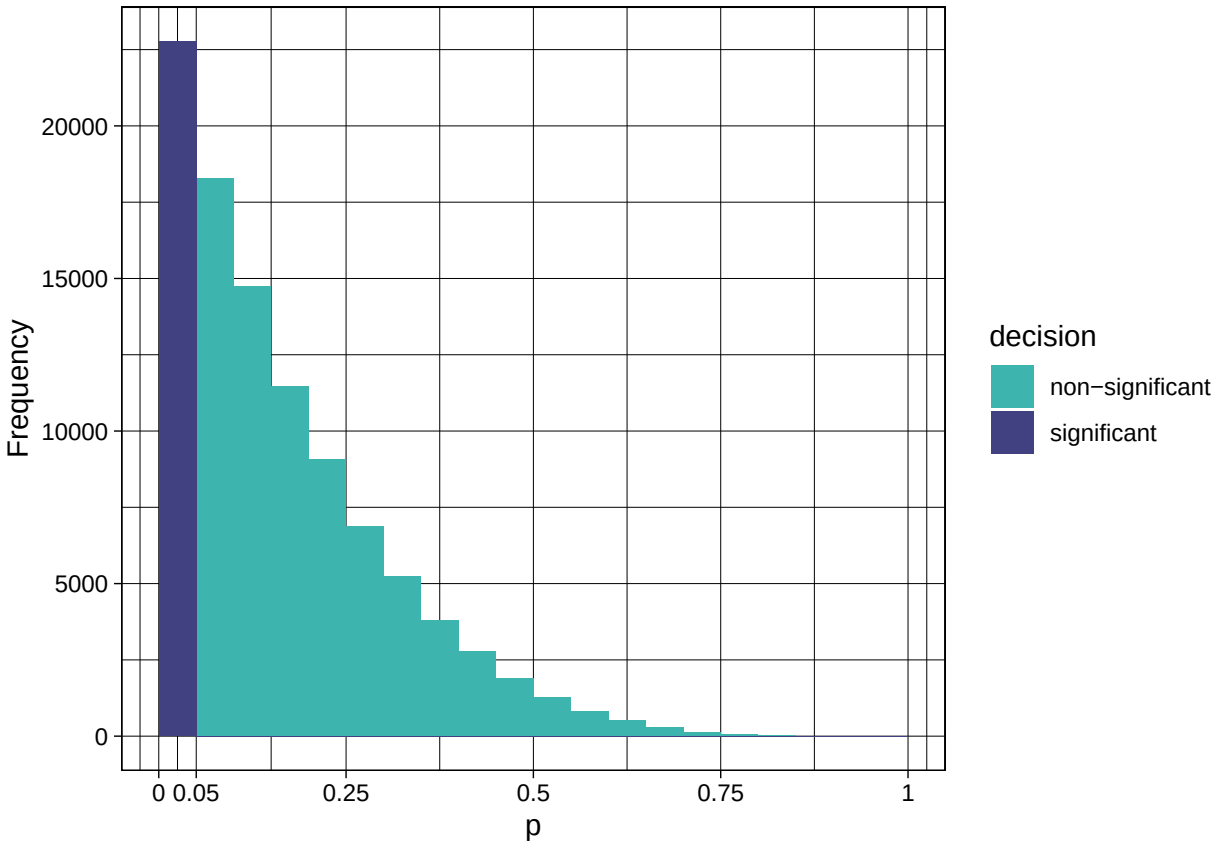
```
#beta distribution are not necessarily normal distributed
tibble(p = rbeta(n = 100000, shape1 = 5, shape2 = 5)) |>
  plot_p_values()   #the p-values distribution never looks like this instead it looks more like the fol
```



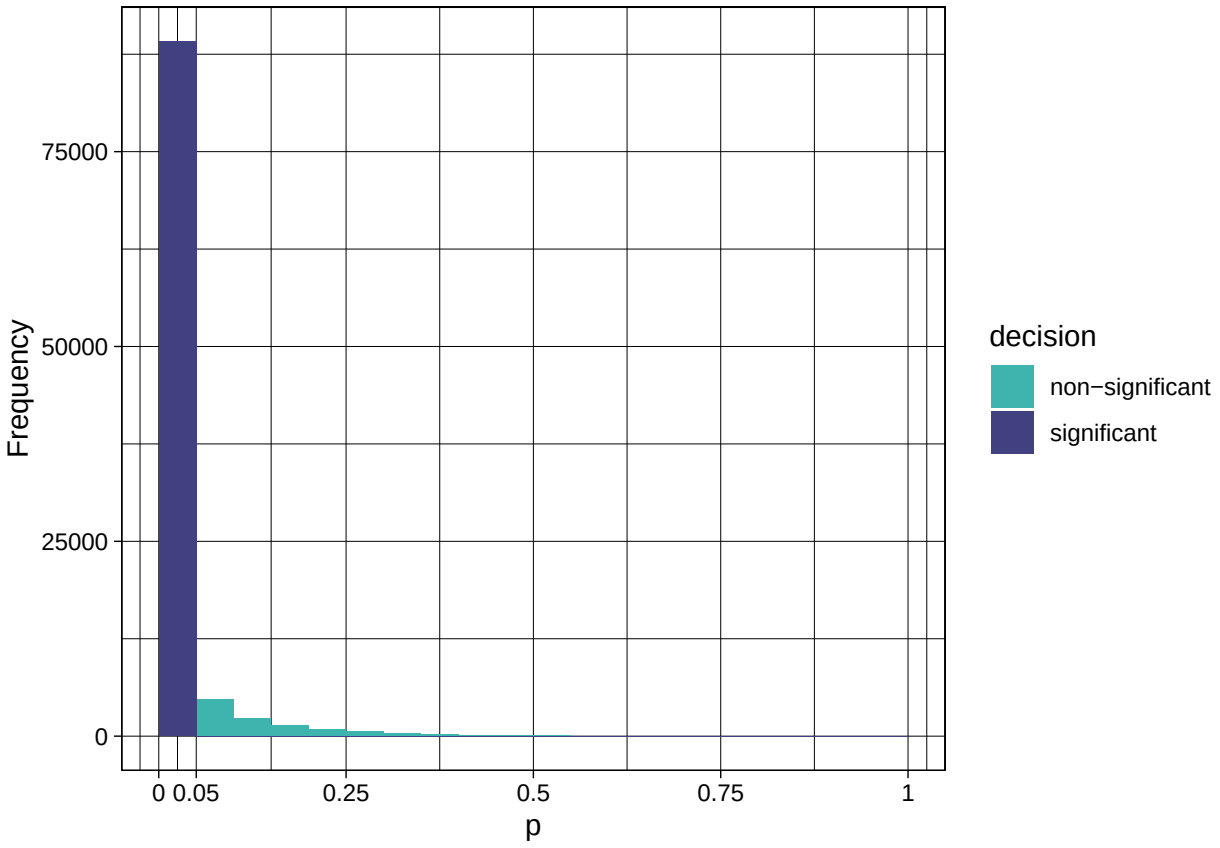

```
tibble(p = rbeta(n = 100000, shape1 = 1, shape2 = 1)) |>
  plot_p_values()
```




```
tibble(p = rbeta(n = 100000, shape1 = 1, shape2 = 5)) |>
  plot_p_values()
```




```
tibble(p = rbeta(n = 100000, shape1 = 0.1, shape2 = 5)) |>
  plot_p_values()
```



- Again, these are not real p values, just data sampled from a Beta distribution.

The distributions of p values

Let's use (a) our understanding of how to construct a simulation [from lesson 2] and (b) our function to plot data that are (or look like) p values to understand how p values are distributed.

Simulate

The code for this simulation is copied from lesson 2 with few modifications.

- No calculation of Cohen's d as its not needed here.
- Simulation parameters use a single sample size, but multiple true effect sizes including both null effects (Cohen's d = 0) and non-null effects (Cohen's d = 0.2, 0.5, 0.8, 1.0). small, medium, large, and perfect effects. (not what Cohen actually wanted but it makes it easy) it should be changed according to sample

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


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

library(tidyr)
library(dplyr)
library(forcats)
library(readr)
```

```r
library(purrr)
library(ggplot2)
library(effsize)
```

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

```r
library(kableExtra)
```

```
##
## Attache Paket: 'kableExtra'
```

```
## Das folgende Objekt ist maskiert 'package:dplyr':
##
##     group_rows
```

```r
library(janitor)
```

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

```
##
## Attache Paket: 'janitor'
```

```
## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     chisq.test, fisher.test
```

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


# define data generating function ----
generate_data <- function(n_control,                    #Input variables
                          n_intervention,
                          mean_control,
                          mean_intervention,
                          sd_control,
                          sd_intervention) {

  data <- #creating a df with two tibbles and binding them together row wise
    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"))

  return(data)  # this is so that the only the results is returned and the rest of the code is not retu
}


# define data analysis function ----
analyse_data <- function(data) {
```

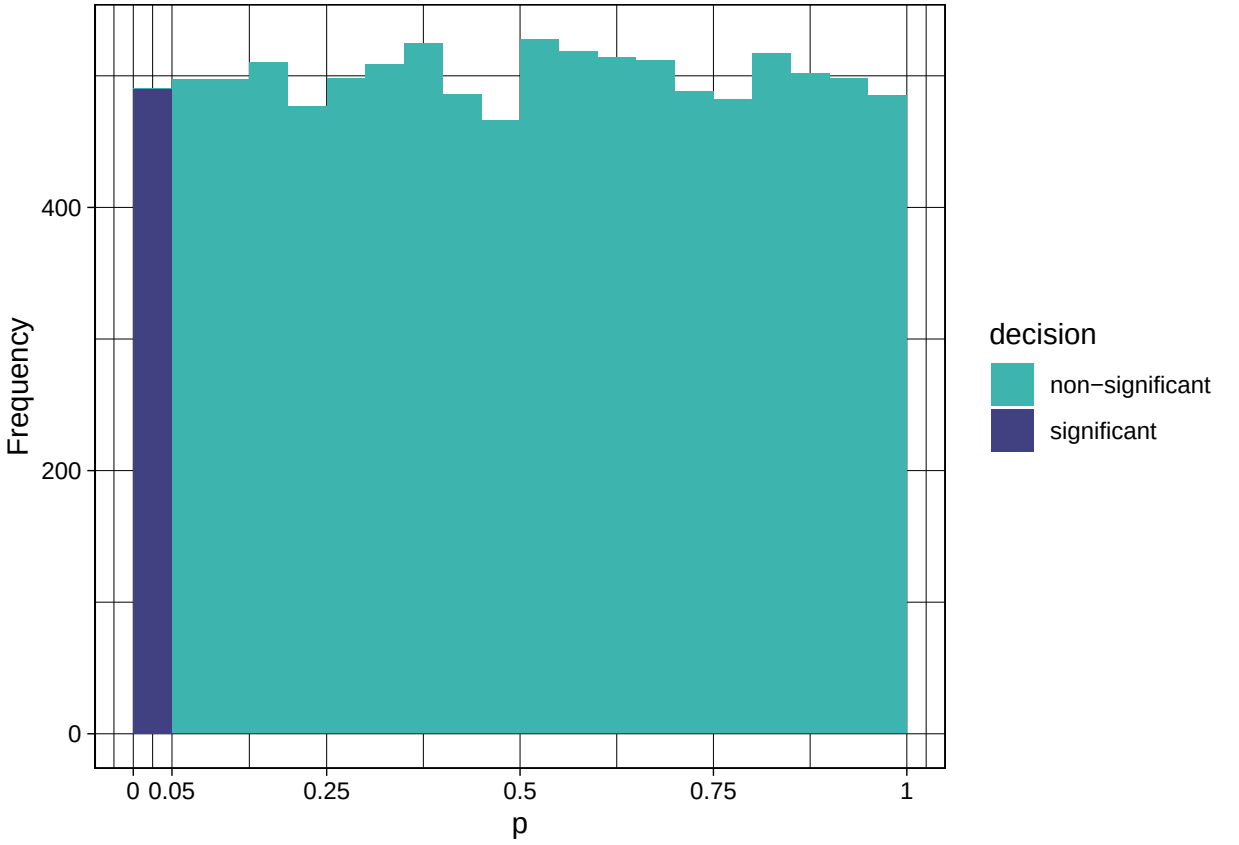
```r
  res_t_test <- t.test(formula = score ~ condition, #condition is a factor with two lvls
                       data = data,
                       var.equal = FALSE,
                       alternative = "two.sided")

  res <- tibble(p = res_t_test$p.value)

  return(res)
}


# define experiment parameters ----
experiment_parameters_grid <- expand_grid(   #creates all possible permutations* (combinations of the g
  n_control = 50,
  n_intervention = 50,
  mean_control = 0,
  mean_intervention = c(0.0, 0.1, 0.2, 0.5, 0.8, 1.0), # only this differs meaningfully from the simula
  sd_control = 1,
  sd_intervention = 1,
  iteration = 1:10000
)


# 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_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),
                                 analyse_data))


# summarise simulation results over the iterations ----
simulation_reshaped <- simulation |>
  # convert `analysis_results` nested-data-frame column to regular columns in the df. in this case, the
  unnest(analysis_results) |>
  # label the true effect value
  mutate(true_effect = paste("Cohen's d =", mean_intervention))
```

Results will be plotted in the next chunks.

Under the null hypothesis

I.e., the data generation (population) is an effect of zero.

```r
simulation_reshaped |>
  # using only the iterations where the population effect was from a true null population
  filter(true_effect == "Cohen's d = 0") |>
  # plot using our function
  plot_p_values()
```



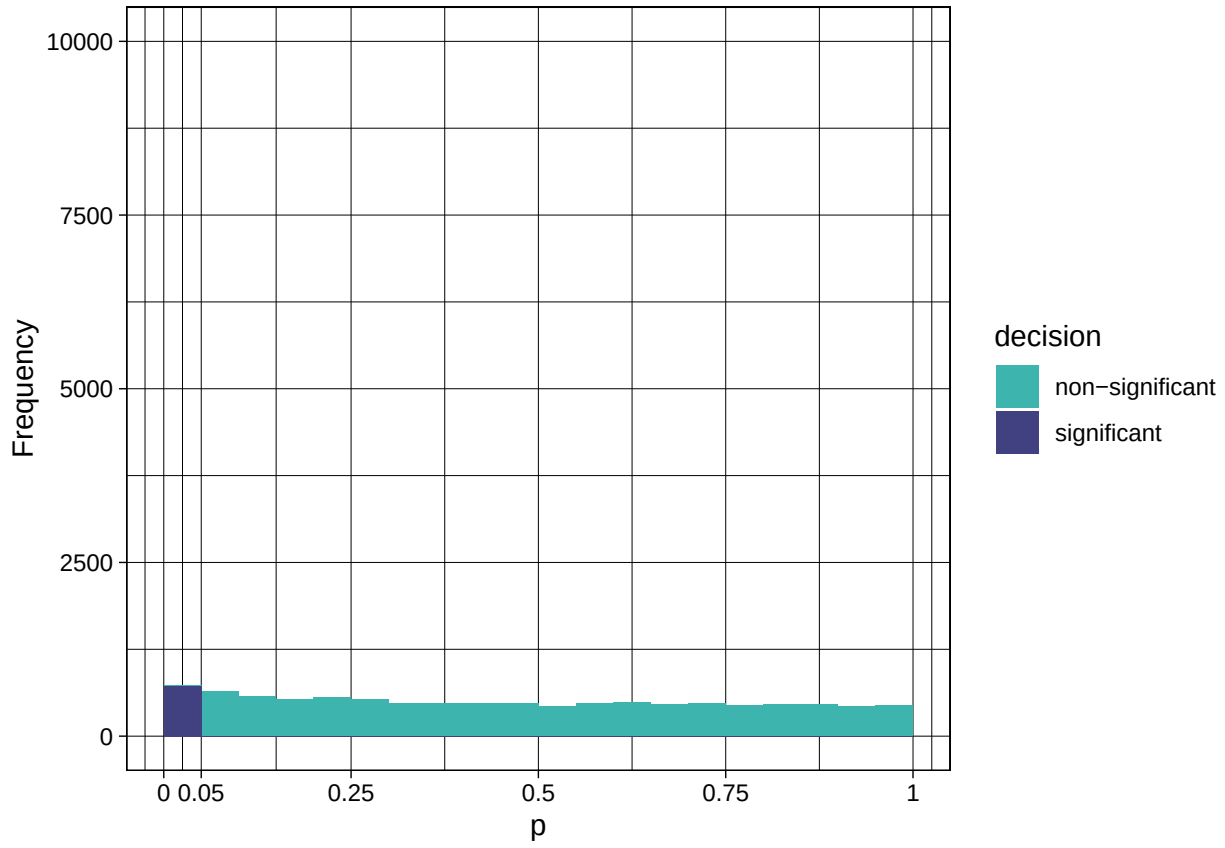

```r
# simulation_reshaped |>
#   filter(true_effect == "Cohen's d = 0") |>
#   summarize(false_positive_rate__aka_alpha = mean(p < .05)) |>
#   mutate_if(is.numeric, janitor::round_half_up, digits = 2)
```

Under the alternative hypothesis

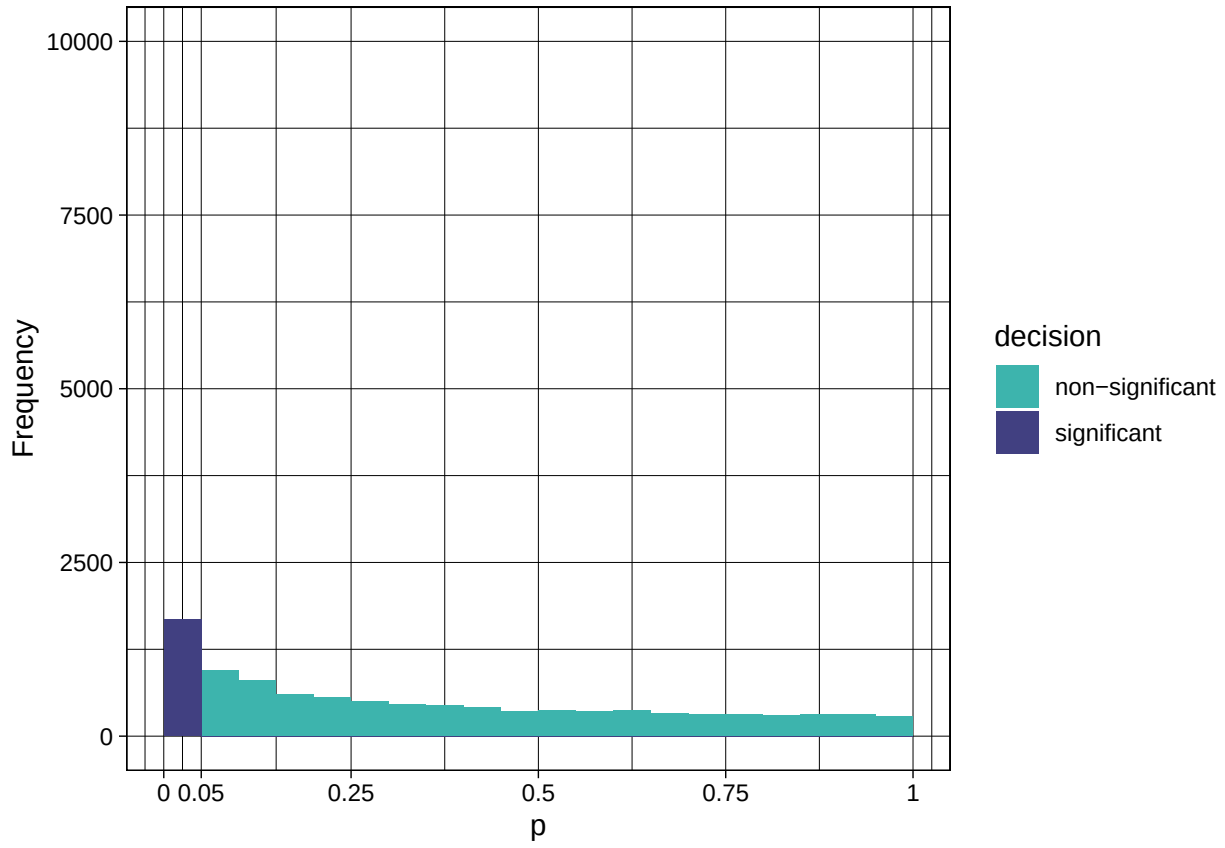
I.e., the data generation (population) is a non-zero effect.

We'll take just one non-zero effect size to start: the largest one (Cohen's d = 1.0)

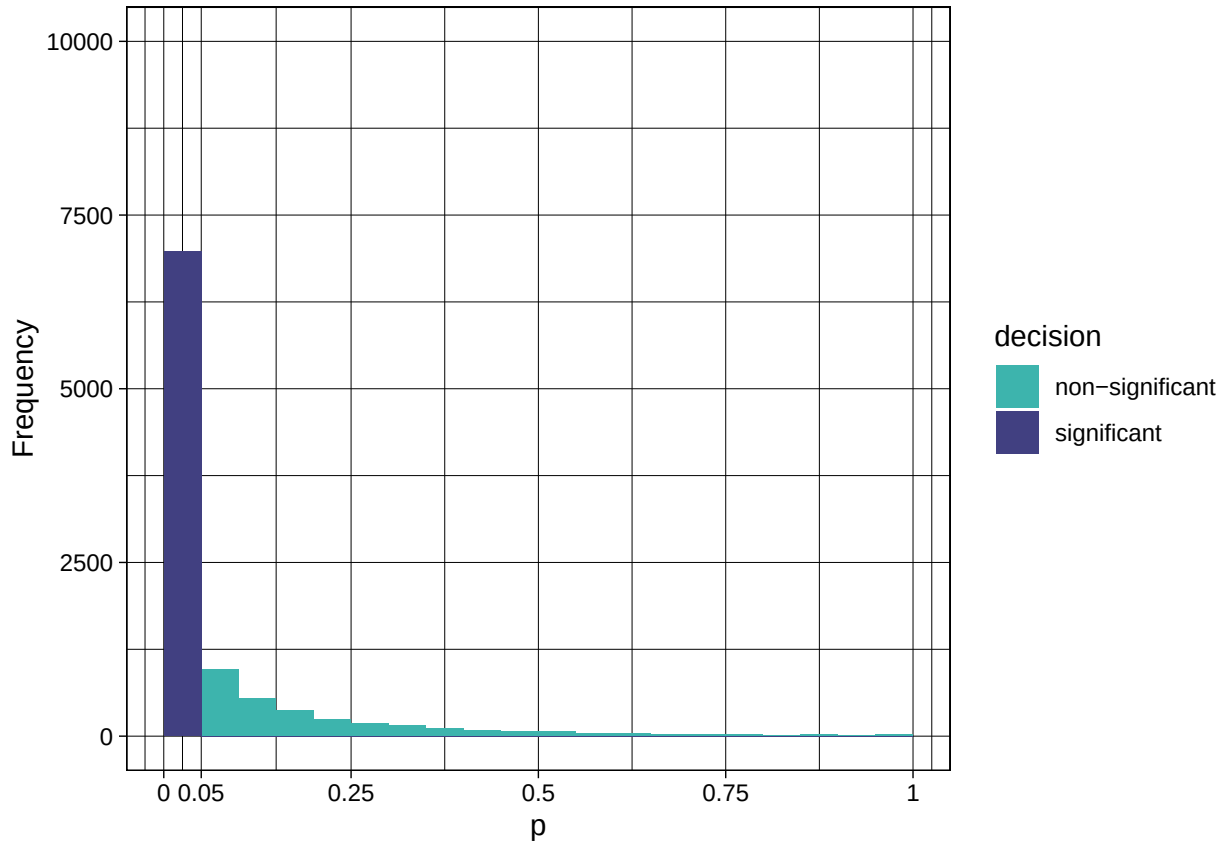
```r
simulation_reshaped |>
  filter(true_effect == "Cohen's d = 0.1") |>
  plot_p_values() +
  coord_cartesian(ylim = c(0, 10000))
```



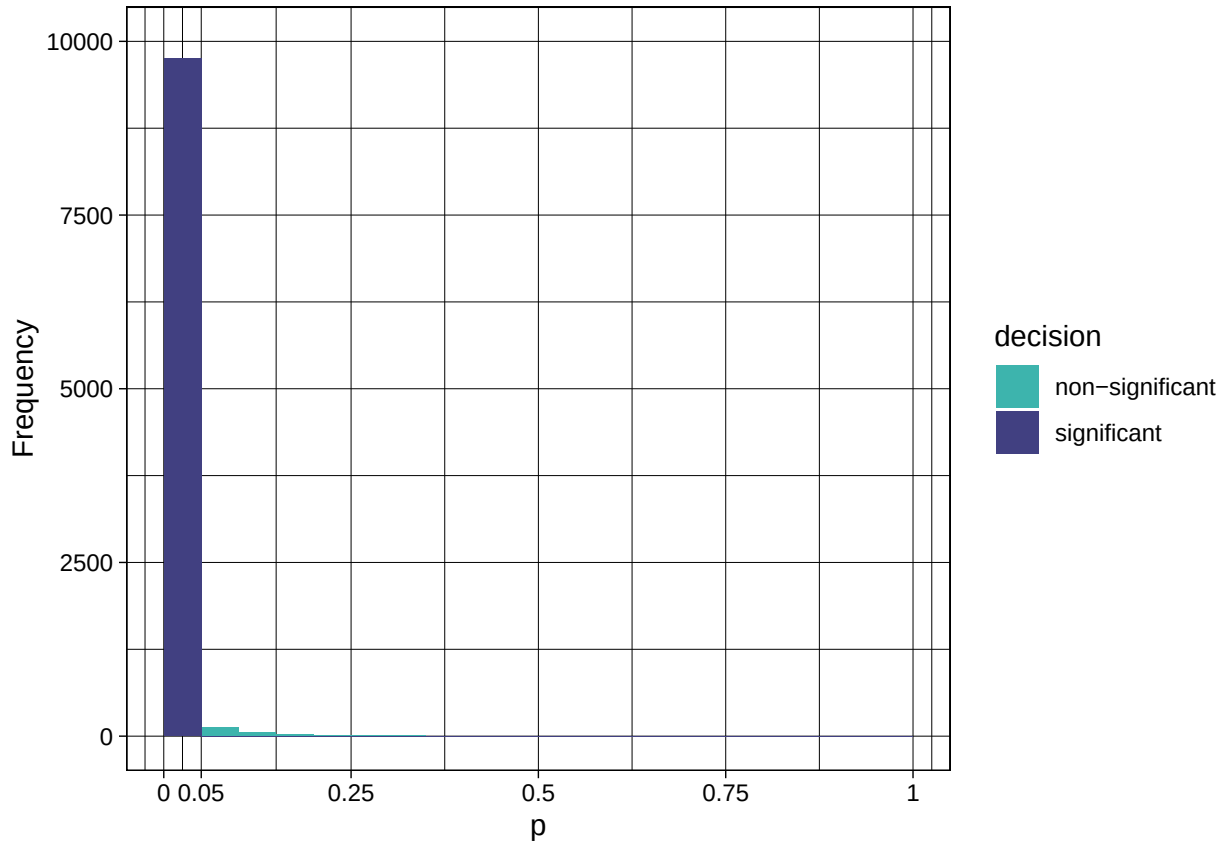

```
simulation_reshaped |>
  filter(true_effect == "Cohen's d = 0.2") |>
  plot_p_values() +
  coord_cartesian(ylim = c(0, 10000))
```



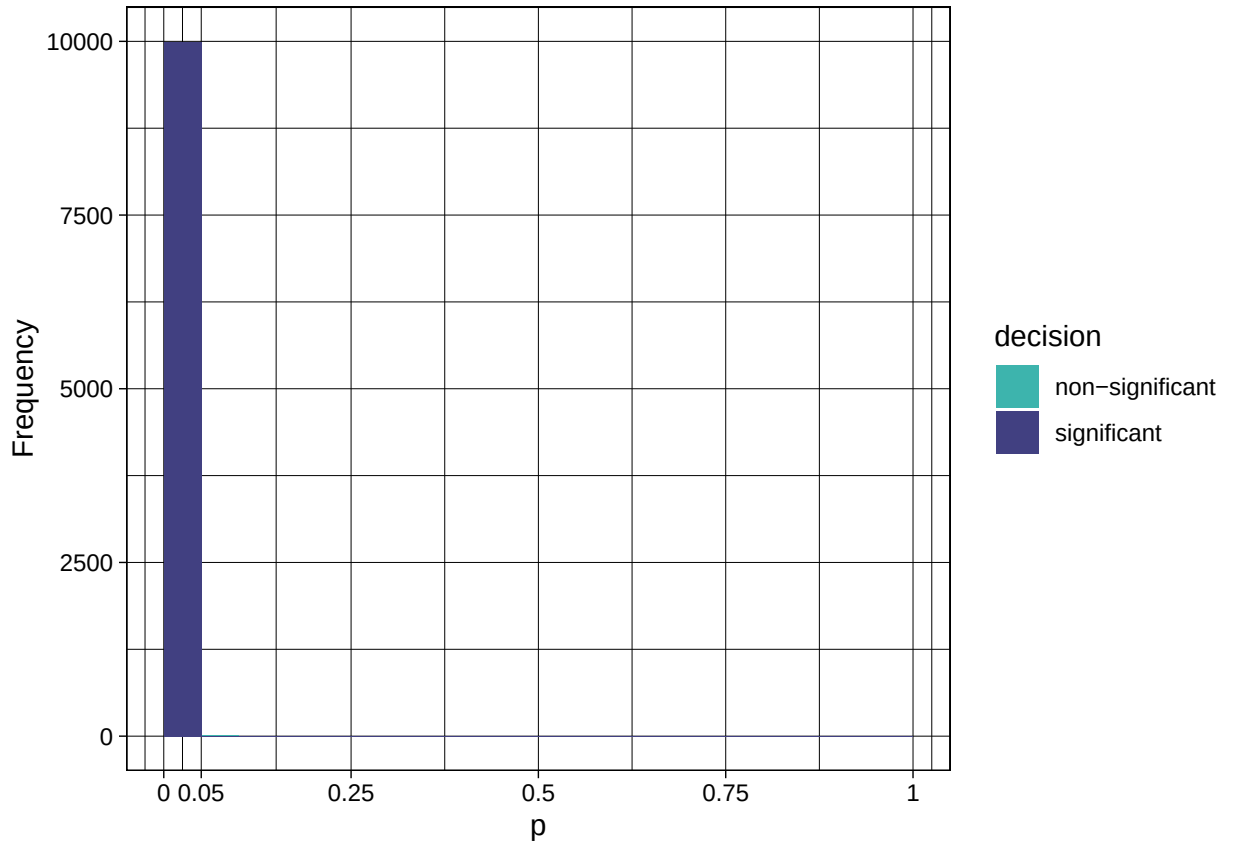

```
simulation_reshaped |>
  filter(true_effect == "Cohen's d = 0.5") |>
  plot_p_values() +
  coord_cartesian(ylim = c(0, 10000))
```




```
simulation_reshaped |>
  filter(true_effect == "Cohen's d = 0.8") |>
  plot_p_values() +
  coord_cartesian(ylim = c(0, 10000))
```




```
simulation_reshaped |>
  filter(true_effect == "Cohen's d = 1") |>
  plot_p_values() +
  coord_cartesian(ylim = c(0, 10000))
```




```
# simulation_reshaped |>
#   filter(true_effect == "Cohen's d = 0") |>
#   summarize(false_positive_rate = mean(p < .05),
#             .by = true_effect) |>
#   mutate_if(is.numeric, janitor::round_half_up, digits = 2)

# simulation_reshaped |>
#   filter(true_effect != "Cohen's d = 0") |>
#   summarize(true_positive_rate__aka_power = mean(p < .05),
#             .by = true_effect) |>
#   mutate_if(is.numeric, janitor::round_half_up, digits = 2)
```

- Why do we change the name of the variable in the table depending on the value true positive rate? Are you able to draw a 2X2 contingency table / confusion matrix that explains the difference between the different names?

Dance of the *p* values

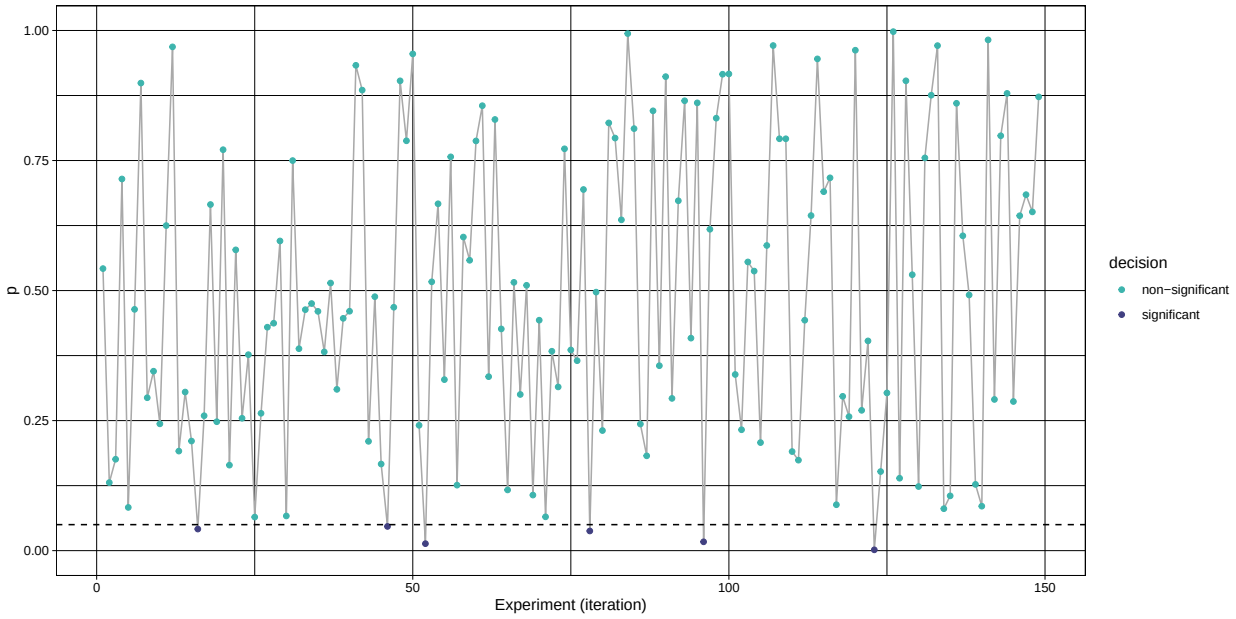
Let's look at the data a different way to understand what we mean by "p values are uniformly distributed under the null hypothesis".

Uniform distributions mean that all variables in the range are just as likely to occur.

For p values, this means that p values will dance back and forth between 0 and 1 between a studies where the null hypothesis is true (population effect is zero). I illustrate this blow by `filter()`-ing only the true effect of 0 conditions, and taking the first 150 iterations. You can see that between the iterations from left to right, which represent independent experiments here, there is no pattern among the p values. Despite the

true effect being zero, a statistically significant result is still found sometimes, because values of p = .01, .02, .03, .04 are just as probable as .48 or .97 or any other.

```
simulation_reshaped |>
  filter(true_effect == "Cohen's d = 0") |>
  filter(iteration < 150) |>
  mutate(decision = ifelse(p < .05, "significant", "non-significant")) |>
  ggplot(aes(iteration, p)) +
  geom_line(color = "darkgrey") +
  geom_point(aes(color = decision)) +
  scale_color_viridis_d(option = "mako", begin = 0.3, end = 0.7, direction = -1) +
  theme_linedraw() +
  geom_hline(yintercept = 0.05, linetype = "dashed") +
  xlab("Experiment (iteration)")
```



Diagnosticity of a given *p* value

Of course, when you read the literature you don't have the advantage of knowing whether the effect being studied is real or not. Instead, it contains a mix or null and non-null effects - even when there is absolutely no *p*-hacking or publication bias.

This simulation assumes a range of true effects ranging from null to medium, and an unknown mix of null and non-null effects. No *p*-hacking or publication bias involved - we'll come to this in a future lesson. There's also no false positives due to violation of assumptions.

Re-run these chunks yourself a few times and see if you can guess from looking at the first plot what the second plot will be made up of. Eg., are most of the significant results true positives (non-null effects) or false positives? Are most of the non-significant results true null effects, or false-negatives?

Remember that as well as assuming no *p*-hacking or publication bias, they also assume all assumptions have been met. False positive rates go up if any of these are violated.

How the simulations are structured:

- Create a literature of 100 studies, studying different effects in the same broad domain.

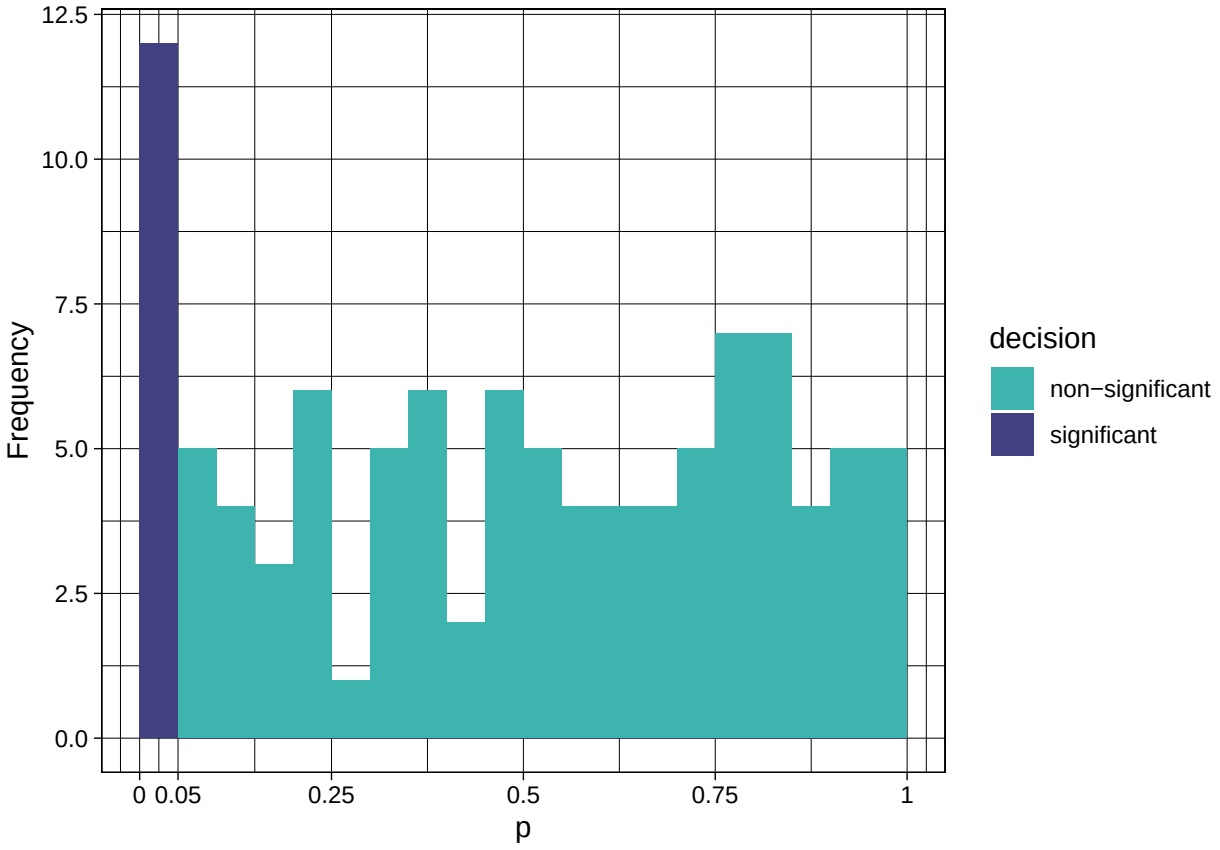
- In this literature, an unknown number of studies (k_null) represent true null effects. This is done by sampling between 0 and 100 studies from the our existing simulations with a true effect of 0. The remaining number of studies (k_nonnull) represent non-null effects of various sizes from very small to medium
- The results from all the studies are then plotted together making only a significant vs. non-significant distinction.

Here, as in the real world, our job is to judge the findings in a given literature only from the visible/published results, without ever having access to their true status. How easy do you find it to whether a given literature contains mostly real or null effects?

```
k_null <- runif(n = 1, min = 0, max = 100) |> round(0)
k_non_null <- 100 - k_null

# simulate published literature by sampling from the existing simulation iterations
unbiased_literature <-
  bind_rows(
    simulation_reshaped |>
      filter(true_effect == "Cohen's d = 0") |>
      sample_n(k_null),
    simulation_reshaped |>
      filter(true_effect %in% c("Cohen's d = 0.1", "Cohen's d = 0.2", "Cohen's d = 0.5")) |>
      sample_n(k_non_null)
  ) |>
  mutate(truth_vs_decision = case_when(true_effect == "Cohen's d = 0" & p >= .05 ~ "True negatives",
                                       true_effect == "Cohen's d = 0" & p <  .05 ~ "False positives",
                                       true_effect != "Cohen's d = 0" & p >= .05 ~ "False negatives",
                                       true_effect != "Cohen's d = 0" & p <  .05 ~ "True positives"))
```

Published literature

```
unbiased_literature |>
  plot_p_values()
```

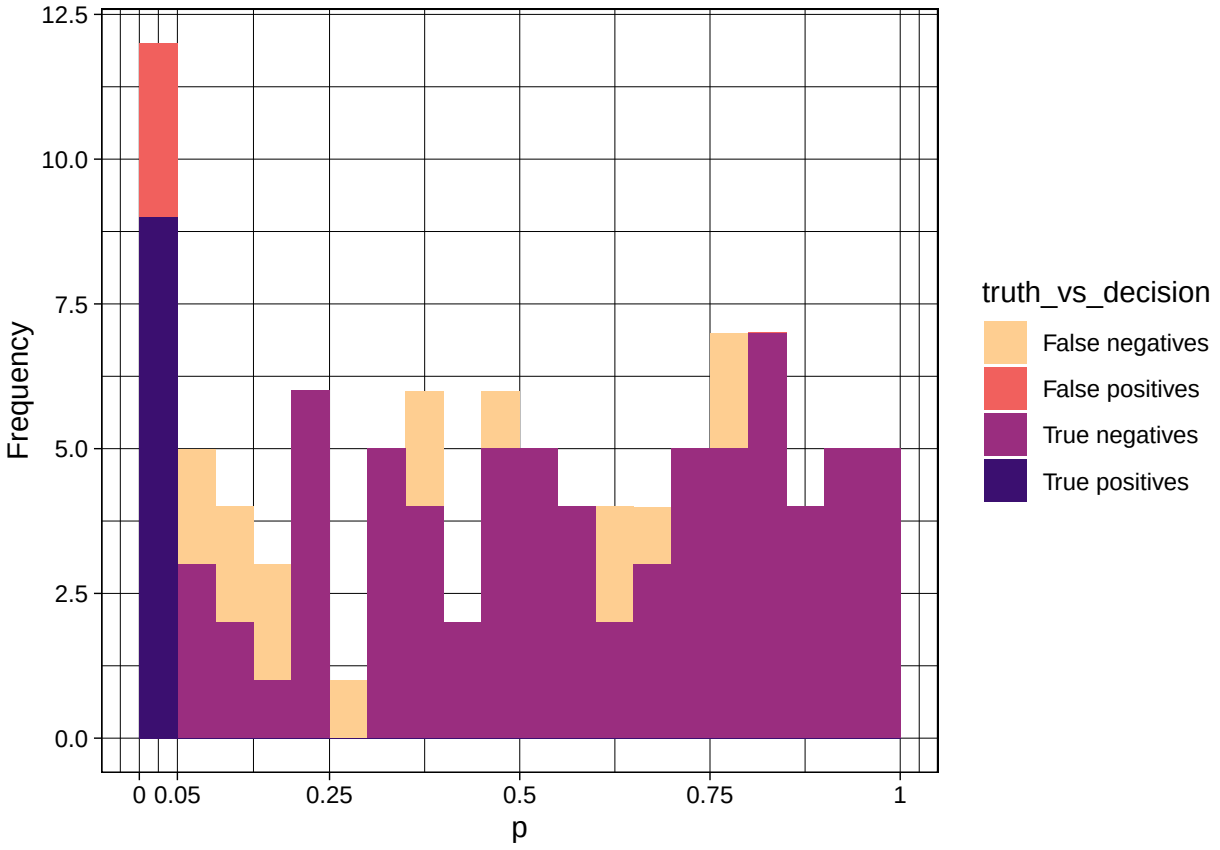


True effects underlying the published literature

(unknowable in the real world)

The table also calculates the "false discovery rate" and the "missed discovery rate" within each literature. These represent the proportion of all positives that are false positives, and the proportion of all negatives that are true negatives.

```
unbiased_literature |>
  ggplot(aes(p, fill = truth_vs_decision)) +
  geom_histogram(binwidth = 0.05, boundary = 0) +
  scale_fill_viridis_d(option = "magma", begin = 0.2, end = 0.9, direction = -1) +
  scale_x_continuous(labels = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     breaks = c(0, 0.05, 0.25, 0.50, 0.75, 1.0),
                     limits = c(0, 1)) +
  theme_linedraw() +
  ylab("Frequency")
```




```
# unbiased_literature |>
#   count(truth_vs_decision) |>
#   pivot_wider(names_from = truth_vs_decision,
#               values_from = n) |>
#   mutate(`False discovery rate` = round_half_up(`False positives` / (`False positives` + `True positi
#          `Missed discovery rate` = round_half_up(`False negatives` / (`False negatives` + `True negat
#   #pivot_longer(cols = everything()) |>
#   #mutate_if(is.numeric, janitor::round_half_up, digits = 2) |>
#   kable() |>
#   kable_classic(full_width = FALSE)
```

Single study from the published literature

The difficulty of using p values to diagnose the true population effect can also be illustrated, perhaps more easily, by using (a) single studies, where (b) the true effect is either null or small.

```
k_null <- 50
k_non_null <- 50

unbiased_literature_2 <-
  bind_rows(
    simulation_reshaped |>
      filter(true_effect == "Cohen's d = 0") |>
      sample_n(k_null),
    simulation_reshaped |>
      filter(true_effect == "Cohen's d = 0.1") |> # "Cohen's d = 0.1" for low power, "Cohen's d = 1" fo
      sample_n(k_non_null)
```

```
  ) |>
  mutate(truth_vs_decision = case_when(true_effect == "Cohen's d = 0" & p >= .05 ~ "True negatives",
                                       true_effect == "Cohen's d = 0" & p <  .05 ~ "False positives",
                                       true_effect != "Cohen's d = 0" & p >= .05 ~ "False negatives",
                                       true_effect != "Cohen's d = 0" & p <  .05 ~ "True positives"))
```

Run the following chunk yourself. Using just the p value, decide if you think the population effect is null (d = 0.0) or non null (d = 0.1).

```
random_study <- unbiased_literature_2 |>
  sample_n(size = 1)

paste("p =", round_half_up(random_study$p, 3))
```

```
## [1] "p = 0.774"
```

Then run the next chunk to see if you were right.

```
paste("Population effect:", random_study$true_effect)
```

```
## [1] "Population effect: Cohen's d = 0"
```

```
random_study$truth_vs_decision
```

```
## [1] "True negatives"
```

You can repeat the above two steps lots of times to check your intuitions.

Remember that the decision accuracy (true positive, true negative, false positive, false negative) is **unknowable** in the real world when doing substantive research - all you can know is an estimate of your test's power (i.e., 1 - β) and your false positive rate (i.e., α).

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] janitor_2.2.0    kableExtra_1.4.0 effsize_0.8.1    lubridate_1.9.3
##  [5] forcats_1.0.0    stringr_1.5.1    dplyr_1.1.4      purrr_1.0.2
```

```
##  [9] readr_2.1.5      tidyr_1.3.1      tibble_3.2.1     ggplot2_3.5.0
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.4        generics_0.1.3    xml2_1.3.6        stringi_1.8.3
##  [5] hms_1.1.3         digest_0.6.35     magrittr_2.0.3    evaluate_0.23
##  [9] grid_4.3.2        timechange_0.3.0  fastmap_1.1.1     fansi_1.0.6
## [13] viridisLite_0.4.2 scales_1.3.0      cli_3.6.2         rlang_1.1.3
## [17] munsell_0.5.1     withr_3.0.0       yaml_2.3.8        tools_4.3.2
## [21] tzdb_0.4.0        colorspace_2.1-0  vctrs_0.6.5       R6_2.5.1
## [25] lifecycle_1.0.4   snakecase_0.11.1  pkgconfig_2.0.3   pillar_1.9.0
## [29] gtable_0.3.4      glue_1.7.0        systemfonts_1.0.6 xfun_0.43
## [33] tidyselect_1.2.1  highr_0.10        rstudioapi_0.16.0 knitr_1.46
## [37] farver_2.1.1      htmltools_0.5.8.1 rmarkdown_2.26    svglite_2.1.3
## [41] labeling_0.4.3    compiler_4.3.2
```