SCR Week 08 Exercises

R-team

07 November, 2019

Exercises part 1

1.1 Tutorial of Combinations and Permutations

In this section, we reflect upon some typical combination and permutation problems, the corresponding mathematical notation, and the implementation in R. In the first half, we will show two examples of combination and permutation of a vector, without using replacement. This means that each element of a vector may occur only once (it is not repeated). In the second half, we will show examples with replacement.

For more information, see combinations and permutations.

a: Combination 1

How many different committees of 4 ($r = 4$) students can be chosen from a group of 5 ($n = 5$) students? Make sure your answer is verified with `R` code.

Hint: when writing your LaTeX, you may like the code `\binom{}{}` *or* `{\choose}`.

Answer: The total number of different committees is $\binom{5}{4}$.

Mathematical notation in general:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

R-code that gives the total number of combinations of 4 out of 5:

```
choose(5, 4)
```

```
## [1] 5
```

b: Combination 2

Take a look at examples of the helpfile of the function `utils::combn`. Can you create the whole "sample space" for all combinations of 4 students out of the 5 available students? Store it inside a variable.

```
all_students <- c("Xinru", "Ionica", "Elise", "Maryam", "Gina")
```

Answer:

```
r <- 4
all_combs <- t(combn(all_students, r)) #the function combn() gives the combinations
```

IMPORTANT: **the order** of the persons in the row does not matter here. Also, it makes no sense to repeat a student in a comittee. Each student occurs only once (no `replacement = TRUE`).

If the order matters, we speak of: permutations.

c: Permutation 1

In how many ways can we permute a vector with 5 elements?

Answer:

In 5! ways.

`R` code:

```
factorial(5) # gives you the total number of ways
```

```
## [1] 120
```

c: Permutation 2

Could you use the function `sample()` to draw a random permutation out of all possible permutations of the variable `all_students`?

Answer:

```
set.seed(13)
sample(all_students, 5, replace = FALSE) #gives you one possible permutation
```

```
## [1] "Elise"  "Maryam" "Xinru"  "Ionica" "Gina"
```

d: Permutation 3

Store 4 random draws (with replacement) from all possible permutations of `all_students` in a `list` or `matrix`.

Answer:

A good, readable, but perhaps cumbersome answer would be:

```
B <- 4
my_perms_mat <- matrix(NA, nrow = length(all_students), ncol = B)
for (b in 1:B) {
  my_perms_mat[, b] <- sample(all_students)
}
```

An shorter, and still readable answer would be:

```
replicate(B, sample(all_students))
```

```
##      [,1]     [,2]     [,3]     [,4]
## [1,] "Elise"  "Gina"   "Maryam" "Elise"
## [2,] "Maryam" "Elise"  "Ionica" "Maryam"
## [3,] "Gina"   "Maryam" "Gina"   "Gina"
## [4,] "Ionica" "Xinru"  "Xinru"  "Xinru"
## [5,] "Xinru"  "Ionica" "Elise"  "Ionica"
```

e: Permutation and combination with replacement

How many permutations are possible of a vector of 5 elements, when we allow for repetition, i.e. the same student can occur more often than once.

NB. For your answer no R code is needed.

Answer:

5^5

f: Permutation and combination with replacement

In this R-code, we use a the function `expand.grid()`. This function creates a data frame from all combinations of the categories of the supplied vectors or factors. We will first give a small illustration of this function:

```
sunsetcolor <- c("red", "orange", "yellow")
expand.grid(sunsetcolor, sunsetcolor)
```

```
##     Var1   Var2
## 1    red    red
## 2 orange    red
## 3 yellow    red
## 4    red orange
## 5 orange orange
## 6 yellow orange
## 7    red yellow
## 8 orange yellow
## 9 yellow yellow
```

Here, all possibilities of 2 colors are shown, and repetition of the same color is allowed.

Use `expand.grid()` to show all possibilities there are for 5 student names using `all_students`.

Note that the first argument(s) in `expand.grid()` can be vectors, factors or a list containing these. Thus,

```
list_args <- rep(list(all_students), 5)
```

could be convenient to use in your answer.

Answer:

```
all_possibs <- expand.grid(list_args)
```

In some R functions it is possible and very convenient to give the arguments in a list structure.

1.2 All Pincodes

Test whether you are fluent with R already, this exercise should not take you more than 10 minutes at an exam.

Create a vector that contains all possible unique PIN codes of bank cards that consist of 4 numbers. The first element of the vector is `"0000"` and the last element of the vector is `"9999"`.

Hint: You may want to use the `paste0()` function with the `collapse = ""` argument in combination with an implicit loop (`[*]pply`), or an explicit loop (`for (elem in vector){}`.

Answer:

```
all_pincodes <- expand.grid(rep(list(0:9), 4))
all_pincodes <- apply(all_pincodes, 1, paste0, collapse = "")
```

1.3 Lady tasting tea to the max.

Perform a permutation test for Dr. Muriel Bristol's exhaustive tea tasting experiment where she had 800 cups of the "Milk first" cups correct.

We generated the data with the following *R* code:

```
set.seed(171123)
Truth <- rep(c("Milk", "Tea"), each = 1000)
choice_Lady <- Truth
MilkyChange_idx <- sample(1:1000, 200)
TeaChange_idx <- sample(1001:2000, 200)
choice_Lady[MilkyChange_idx] <- "Tea"
choice_Lady[TeaChange_idx] <- "Milk"
```

and we use the function:

```
GetNrSuccesses <- function(choice, truth) {
  sum("Milk" == choice & "Milk" == truth) # R recycling behavior trick
}
t_stat <- GetNrSuccesses(choice_Lady, Truth)
t_stat
```

```
## [1] 800
```

Code your own permutation test with the above ingredients. What do you think Sir Ronald Fisher would have concluded from this experiment?

Answer

```
B <- 1000; N <- length(choice_Lady)
T_bs <- numeric(B)
set.seed(171123)
for (b in 1:B) {
  T_bs[b] <- GetNrSuccesses(sample(choice_Lady), Truth)
}
mean(T_bs >= t_stat)
```

```
## [1] 0
```

The probability is practically zero that Dr. Bristol's results (or better) are a realization from the hypothesis that Dr. Bristol's results are based on pure chance allone.

Exercises part 2

2.1 A Permutation test on Student's sleep data

Create a function with which you can perform a permutation test instead of the following two-samples t.test:

```
t_stat <- t.test(extra ~ group, data = sleep)
```

Note: stick to the example of the helpfile of the `t.test()` function in `R` (`example(t.test)`). Thus, assume we have two independent groups being measured on their `extra` hours of sleep. We supposedly have 2 groups of each 10 persons in the data.

Answer:

We have `choose(20, 10)` possible combinations from which we can create our estimated sampling distribution of the default `t.test` statistic under the nullthypothesis.

The simplest form of performing a permutation test would be:

```
B <- 1000 # number of samples
tobs <- t.test(extra ~ group, data = sleep)$statistic
Tprms <- numeric(B)
for (b in 1:B) {
  Tprms[b] <- t.test(extra ~ sample(group), data = sleep)$statistic
}; rm(b)
p_estim1 <- mean(abs(tobs) < abs(Tprms)) # estimated p-value
```

Hence, the estimated p-value is 0.072 for the permutation test we performed above.

2.2 Two samples permutation tests of the trimmed mean

In this exercise we create a test statistic T to test whether the "20% trimmed mean" of the samples $X_1, ..., X_m$ and those of $Y_1, ..., Y_n$ come from populations where the mean is the same.

The data example:

```
set.seed(160929)
mu <- 0.5
x <- rnorm(8) # sampling distr F
y <- rnorm(12, mu, 2) # sampling distr G
```

The test statistic:

```
Get20TrimmedMean <- function(x,y) {
  out <- mean(x, trim = 0.2) - mean(y, trim = 0.2)
  return(out)
}
t_obs <- Get20TrimmedMean(x,y)
t_obs
```

```
## [1] -1.089706
```

a

How many times does each **combination** of the $m = 8$ samples out $N = n + m = 20$ occur in the `factorial(N)` possible **permutations**?

Answer:

Every combination out of the `choose(m + n, m)` combinations can have `factorial(N)/choose(N, m)` doubles.

However, to draw samples from the `choose(n + m, m)` combinations directly you'll need a lot of tricks when the sample space is too large, that is why, in practice, we take A LARGE number of random partitions instead.

b

Can you create the whole "sample space" for all combinations of $m = 8$ combinations out of $N = 20$? Store it into an object. Use `utils::combn()`.

Answer

```
N <- 20; m <- 8
all_combs <- combn(20, 8) # the whole sample space for m = 8 and N = 20
```

c

Perform a two-sided permutation test for the difference between the 20% trimmed of $X_1, ..., X_m$ and $Y_1, ..., Y_n$.

Use all samples from the complete sample space (that does not contain any doubles). You have computed this sample space in **b**. Thus, `B <- choose(m + n, m)`. Using the same data everytime, and without setting a seed, would your p-value be the same every time you run this test?

Note: the test may run a while on your computer!

Answer

```
N <- 20; m <- 8
all_combs <- combn(N, m) # the whole sample space for m = 8 and N = 20
```

```
RunRefinedPermTest <- function(x, y, combs, Get20TrimmedMean) {
  B <- ncol(combs)
  tperm <- numeric(B)
  t_obs <- Get20TrimmedMean(x,y)
  z <- c(x,y); N <- length(z); m <- length(x)
  for (b in 1:B) {
    xco <- combs[, b]
    tperm[b] <- Get20TrimmedMean(z[xco], z[-xco])
  }
  pval <- mean(abs(tperm) >= abs(t_obs))

  out <- c(pval = pval, theta = mean(tperm), se = sd(tperm))
  return(out)
}
RunRefinedPermTest(x = x, y = y, combs = all_combs, Get20TrimmedMean = Get20TrimmedMean)
```

```
##       pval      theta         se
## 0.17485909 0.04602594 0.78413016
```

Since we use the complete permutation distribution of the statistic, the p-value will always be the same (given tat it is conditional on this data set, the data does not change)

d

Perform the two-sided permutation test again, but this time you do not need to use the whole sample space variable `all_combs` from subtask **b**. Instead, use $B = 1000$ random permutations, e.g. `sample(1:N, m)`. Why is your p-value different from **c**?

Answer

A two-sided permutation test with be the following:

```
RunDefaultPermTestt <- function(x, y, B, Get20TrimmedMean) {
  tperm <- numeric(B)
  t_obs <- Get20TrimmedMean(x,y)
  z <- c(x,y); N <- length(z); m <- length(x)
  for (b in 1:B) {
    xco <- sample(1:N, m)
    tperm[b] <- Get20TrimmedMean(z[xco], z[-xco])
  }
  pval <- mean(abs(tperm) >= abs(t_obs))

  out <- c(pval = pval, theta = mean(tperm), se = sd(tperm))
  return(out)
}
RunDefaultPermTestt(x, y, B = 1e3, Get20TrimmedMean)
```

```
##       pval      theta         se
## 0.16400000 0.02226891 0.77231619
```

We are now working with a (random) sample of the complete permutation distribution. Each time with a different sample of permuted data sets, the p-value can be different.

e

Would you know of a study / experiment to check whether on average the previous two test are the same? You don't need to programme this in `R`, though for the assignment this could perhaps be a task. . . . . .

Answer:

You could use a Monte Carlo study!

2.3 Paired two-sample permutation test

The data:

```
set.seed(190933)
mu <- 1.75
x <- rnorm(30)
y <- rnorm(30, mu, 2)
```

The test statistic (but now for paired testing...):

```
Get20TrimmedMean <- function(x,y) mean(x, trim = 0.2) - mean(y, trim = 0.2)
t_obs <- Get20TrimmedMean(x,y)
t_obs
```

```
## [1] -2.040649
```

To perform a permutation test we need to swap the elements in X and Y. We have `2^length(x)` possibilities for swapping the elements in X and Y.

a

Create a function that can perform a permutation test (set $B = 1000$ as a default). Your function shoul output the p-value, the estimated mean and the estimated standard error of your test statistic.

Start with the default permutation test where we just use

`swaps <- sample(c(0, 1), N/2, replace = TRUE)`

directly for each permutation. When `swaps[1] == 1`, it means that the first value for X and Y should be swapped, when it `swaps[1] == 0`, then we do not swap these values.

Answer:

```
RunDefaultPairedPermTest <- function(x, y, B, Get20TrimmedMean) {
  if(length(x) != length(y)) stop("length(x) != length(y): cannot do a paired sample test")
  tperm <- numeric(B)
  t_obs <- Get20TrimmedMean(x,y)
  N <- 2 * length(x)
  z <- c(rbind(x, y)) # x and y in pairs
  idx <- seq(1, N, by = 2)
  for (b in 1:B) {
    xsh <- idx + sample(c(0, 1), length(x), replace = TRUE)
    tperm[b] <- Get20TrimmedMean(z[xsh], z[-xsh])
  }

  pval <- mean(abs(tperm) >= abs(t_obs))

  out <- c(pval = pval, mean = mean(tperm), se = sd(tperm))
  return(out)

}

RunDefaultPairedPermTest(x, y, B = 1000, Get20TrimmedMean)
```

```
##       pval       mean         se
## 0.00000000 0.01563536 0.59197427
```

```
# MC_result <- rowMeans(replicate(100, RunDefaultPairedPermTest(x, y, B = 1000, Get20TrimmedMean)))
```

b

Note that the p-values in **a** can vary a lot (if you don't use `set.seed`). Since there are "only" `2^length(x)` possibilties, it is highly likely that with `B = 1000` draws, and with `sample(c(0, 1), length(x), replace = TRUE)` we will get multiple swap vectors that may be the same.

The probability of one double swap vector is related birthday problem formula:

```
B <- 1000
1 - prod(1 - 1:(B - 1)/(2^length(x)))
```

```
## [1] 0.0004650876
```

Hence, the probability is practically 1!

Perhaps a practical solution to this variability problem of the p-values in **a** would be to create a random swap matrix of B rows (and N/2 columns) beforehand. Each row b (out of the B rows) is a realization of `sample(c(0, 1), length(x), replace = TRUE)`, but all rows are unique.

To do so, first sample e.g. $3B$ rows, and then check whether you can reduce this "swap" matrix into B unique rows. If not, sample $4B$ rows, etc. The code we used:

```
# The swap matrix:
swaps <- sample(c(0, 1), length(x)*(B*3), replace = TRUE)
dim(swaps) <- c(B*3, length(x))
swaps <- unique(swaps)[1:B, ] # select only the unique rows
```

Code again the permutation test, but this time use a beforehand specified swap matrix. Did it help to get a more robust (= stable) p-value?

Answer

```
# The permutation test:
RunRefinedPairedPermTest <- function(x, y, B, Get20TrimmedMean) {

  # The swap matrix:
  swaps <- sample(c(0, 1), length(x)*(B*3), replace = TRUE)
  dim(swaps) <- c(B*3, length(x))
  swaps <- unique(swaps)[1:B, ] # select only the unique rows

  B <- min(B, nrow(swaps))
  tperm <- numeric(B)
  t_obs <- Get20TrimmedMean(x, y)
  z <- c(rbind(x, y)) # x and y in pairs
  idx <- seq(1, length(x), by = 2)
  for (b in 1:B) {
    xsh <- idx + swaps[b, ]
    tperm[b] <- Get20TrimmedMean(z[xsh], z[-xsh])
  }
  pval <- mean(abs(tperm) > abs(t_obs))

  pval <- mean(abs(tperm) >= abs(t_obs))
  out <- c(pval = pval, mean = mean(tperm), se = sd(tperm))
  return(out)
```

```
}
RunRefinedPairedPermTest(x,y, B, Get20TrimmedMean)
```

```
##        pval        mean          se
##  0.00000000 -0.09231921  0.35514384
```

```
# MC_result_rfnd <- rowMeans(replicate(100, RunRefinedPairedPermTest(x, y, B = 1000, Get20TrimmedMean))
# MC_result_dflt <- rowMeans(replicate(100, RunDefaultPairedPermTest(x, y, B = 1000, Get20TrimmedMean))
```

Nope, the p-value seems to be still quite unstable, but they also seem to be lower!!

c

The p-value remains quite unstable in **b**. Code a permutation test where we use all unique swap possibilities (the whole sample space, without duplications). Thus, $B = 4096$. What is the permutation test p-value that we were trying to estimate in **a** and **b**?

Hint: You may want to use the function ***expand.grid()***

Answer:

All possibilities on whether we would swap (= 1), or not (= 0):

```
all_combs <- expand.grid(rep(list(0:1), 12))
dim(all_combs)
```

```
## [1] 4096   12
```

```
B <- nrow(all_combs); tperm <- numeric(B)
z <- c(rbind(x, y)) # x and y in pairs
idx <- seq(1, 24, by = 2)
for (b in 1:B) {
    xsh <- idx + as.matrix(all_combs[b, ])
    tperm[b] <- Get20TrimmedMean(z[xsh], z[-xsh])
    }
pval <- mean(abs(tperm) > abs(t_obs))

pval <- mean(abs(tperm) >= abs(t_obs))
out <- c(pval = pval, mean = mean(tperm), se = sd(tperm))
return(out)
```

```
##       pval       mean         se
## 0.00000000 0.01975471 0.46187202
```

If there are not soo many possibilities from which we can sample, better use the whole sample space, or find a way to avoid having too many identical permutations!

c

What could be a reason that the variability in results of the permutation test in **a** and **b** are impossible to test for this data set?

Answer:

Sample size could be too low. Should / could be tested with a Monte-Carlo studie on the test-statistic. . . .

self-study: Permutations in Genetics

by courtesy of Jelle Goeman

Install the package `penalized`, and have a look at the nki70 data contained in that package. You should be able to run the following code:

```
library(penalized)
```

```
## Loading required package: survival
```

```
## Welcome to penalized. For extended examples, see vignette("penalized").
```

```
data(nki70)
```

The nki70 data consists of gene expression data of 70 genes that are prognostic for disease-free survival in breast cancer patients, the so-called "mammaprint" signature, together with survival status and some clinical covariates. These genes and covariates are measured for a sample of 144 young breast cancer patients.

We are interested in association between gene expression and estrogen receptor status (ER), which is itself an important predictor of breast cancer prognosis.

Install the package `penalized`, and have a look at the nki70 data contained in that package. You should be able to run the following code:

```
library(penalized)
data(nki70)
```

2.4 Permutations in Genetics

a

Test whether the expression of the WISP1 gene is associated with ER status. You can choose any test of your preference. In the model answers we used a t-test.

Answer:

```
t.test(WISP1~ER, data=nki70)
```

```
##
##  Welch Two Sample t-test
##
## data:  WISP1 by ER
## t = -2.0826, df = 36.652, p-value = 0.04432
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.163082749 -0.002214314
## sample estimates:
## mean in group Negative mean in group Positive
##            -0.06293388             0.01971465
```

b

Make a randomly permuted version of the ER status variable, and test for association of the WISP1 gene with the permuted ER.

Answer:

```
t.test(WISP1~ER[sample(144)], data=nki70)
```

```
##
##  Welch Two Sample t-test
##
## data:  WISP1 by ER[sample(144)]
## t = 1.5492, df = 40.49, p-value = 0.1291
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.0172157  0.1304426
## sample estimates:
## mean in group Negative mean in group Positive
##            0.050216486           -0.006396973
```

c

Repeat exercise (b) 100 times and make a histogram of the 100 resulting test statistics.

Answer:

```
permT <- replicate(100, t.test(WISP1~ER[sample(144)], data=nki70)$statistic)
hist(permT)
```

d

Calculate a permutation p-value. Hint: two-sided test: remember to take absolute values

Answer:

```
trueT <- t.test(WISP1~ER, data=nki70)$statistic
mean(abs(permT) >= abs(trueT))
```

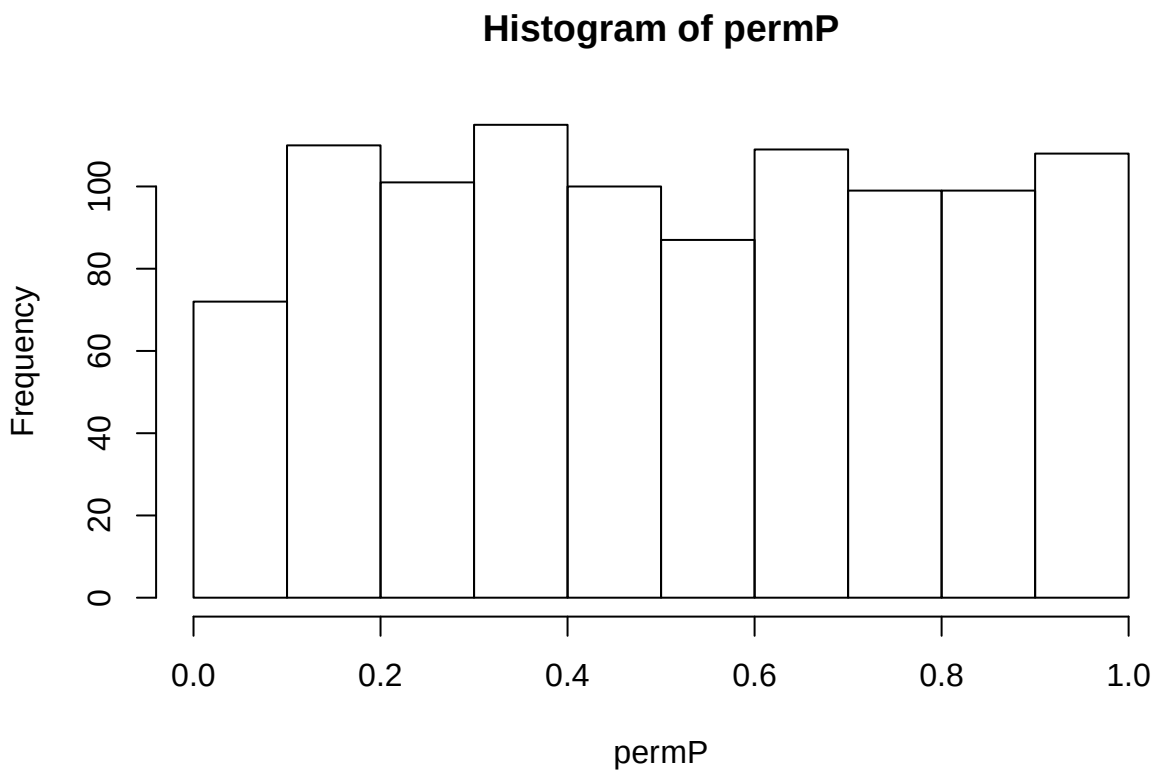
```
## [1] 0.04
```

e

Make a histogram of the p-values for the 100 permutations.

Answer:

```
permP <- replicate(1000, t.test(WISP1~ER[sample(144)], data=nki70)$p.value)
hist(permP)
```



f

Test association with ER status for all 70 genes (columns 8 to 77).

Answer:

```
allP <- sapply(8:77, function(i) t.test(nki70[,i]~ER, data=nki70)$p.value)
names(allP) <- colnames(nki70)[8:77]
```

g

Use Bonferroni to correct the results for multiple testing. How many null hypotheses can be rejected at a type-I error rate of 0.05?

Answer:

```
sum(allP<=0.05/length(allP))
```

```
## [1] 30
```

2.5 Multipe testing by permutation

a

Calculate the minimum p-value over all 70 genes.

Answer:

```
min(allP)
```

```
## [1] 6.40646e-30
```

b

Permute ER status, test association of all genes with permuted ER status and calculate the minimum p-value.

Answer:

```
permPs <- sapply(8:77, function(i) {
  ERperm <- nki70$ER[sample(144)]
  t.test(nki70[,i]~ERperm)$p.value
})
min(permPs)
```

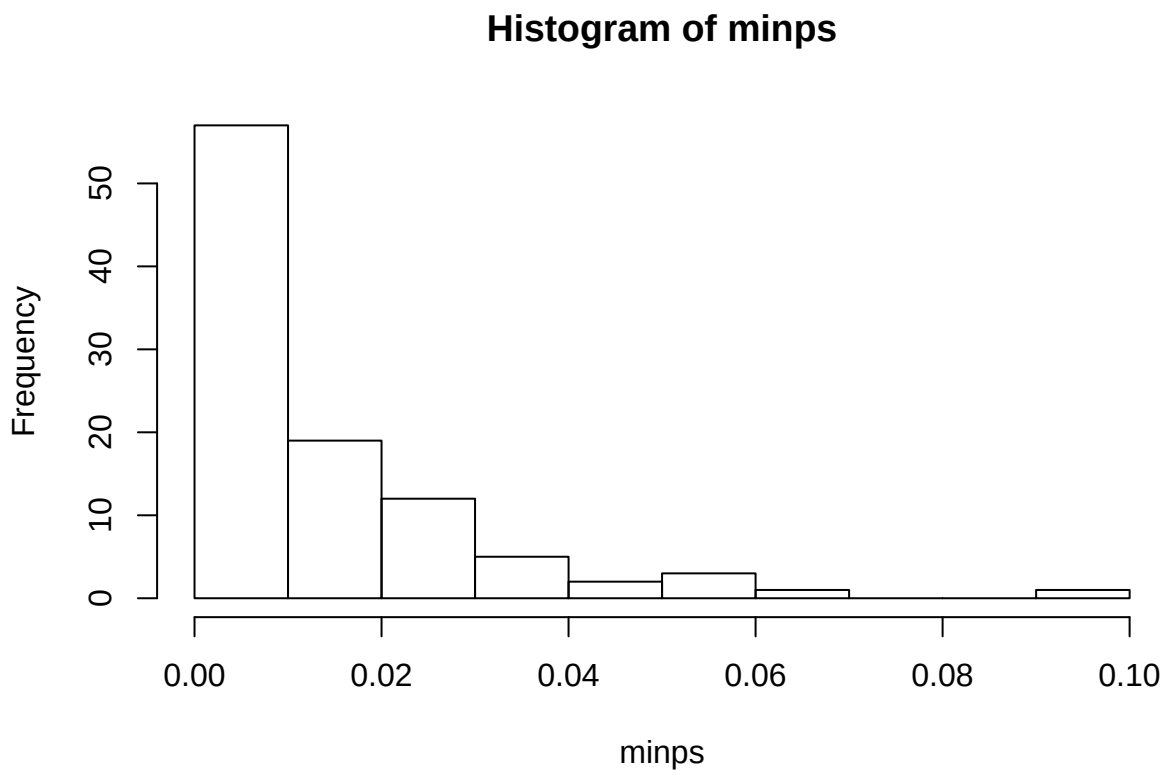
```
## [1] 0.0176327
```

c

Repeat the permutation 100 times and make a histogram of the minimum p-values.

Answer:

```
permPs <- replicate(100, sapply(8:77, function(i) {
  ERperm <- nki70$ER[sample(144)]
  t.test(nki70[,i]~ERperm)$p.value
}))
minps <- apply(permPs, 2, min)
hist(minps)
```



d

Find the 0.05 quantile of the permutation distribution of minimum p-values, and compare with the Bonferroni threshold.

Answer:

```
quantile(minps, 0.05)
```

```
##           5%
## 0.0004821256
```

```
0.05/length(8:77)
```

```
## [1] 0.0007142857
```

e

How many hypotheses can be rejected based on the critical value found in exercise (d)?

Answer:

```
sum(allP <= quantile(minps, 0.05))
```

```
## [1] 28
```

f

Repeat sub exercises 2.5c - 2.5e using only the hypotheses that could not (yet) be rejected in exercise (e). How many new rejections are made? And how many hypotheses in total?

Hint: The answers you should find with `R` are 3 and 34

Answer:

```
rejected <- (allP <= quantile(minps, 0.05))
permPs2 <- permPs[!rejected,]
minps <- apply(permPs2, 2, min)
sum(allP <= quantile(minps, 0.05))
```

```
## [1] 34
```

g

Optional: repeat exercise (f) until no further rejections occur. In this way you've created a permuted version of the [holm's p-value method]. See the `stats::p.adjust` function or check:

Holm, S. (1979). A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics 6, 65-70.

Answer:

```
ready <- FALSE
reject <- rep(F,length(allP))
steps <- 0
while (!ready) {
  steps <- steps + 1
  minps <- apply(permPs[!rejected,], 2, min)
  critical <- quantile(minps, 0.05)
  newreject <- allP <= critical & (!reject)
  reject <- reject | newreject
  print(sum(reject))
  ready <- !any(newreject)
}
```

```
## [1] 34
## [1] 34
```