Assignment 05

Statistical Computing and Empirical Methods

A word of advice

Think of the SCEM labs like going to the gym: if you pay for gym membership, but instead of working out you use a machine to lift the weights for you, you won't grow any muscle.

ChatGPT, DeepSeek, Claude and other GenAI tools can provide answers to most of the questions below. Before you try that, please consider the following: answering the specific questions below is not the point of this assignment. Instead, the questions are designed to give you the chance to develop a better understanding of estimation concepts and a certain level of **statistical thinking**. These are essential skills for any data scientist, even if they end up using generative AI - to write an effective prompt and to catch the common (often subtle) errors that AI produces when trying to solve anything non-trivial.

A very important part of this learning involves not having the answers ready-made for you but instead taking the time to actually search for the answer, trying something, getting it wrong, and trying again.

So, make the best use of this session. The assignments are not marked, so it is much better to try them yourself even if you get incorrect answers (you'll be able to correct yourself later when you receive feedback) than to submit a perfect, but GPT'd solution.

Introduction

Before starting, make sure you have reviewed the Week 5 lecture slides. Additionally, download the following data files from Blackboard and save them in the same folder as your solutions Rmarkdown. We will use these files for some of the questions in this assignment.

- *fc_weight_perception.csv*
- *EBV-200-HybridA-performance.rds*
- *UPMSP_SA_full_results.rds*

This assignment will be done in **RStudio**. To enter your solutions, please use the pre-structured R markdown template provided for this week. This is a similar format to what you will use in the final coursework.

You can **optionally** submit your .Rmd solutions file by 15:00h Friday 24 October. This will help us understand your work but will not count towards your final grade. If you want to upload your .Rmd file, use the "Assignment 05" link under the *Assessments, Submission and Feedback* tab in the course Blackboard page.

This assignment relates to this week's lecture on basic statistical inference, and extends the concepts covered in the lecture to also cover one-sided alternative hypotheses, power and sample size calculations, the verification of the assumptions of the t-test, and non-parametric alternatives to this test.

In this first part, we'll review the t-test, which allows us to test hypotheses about a population mean when the population variance is unknown. t-tests are implemented in base R with the aptly-named function `t.test()`. To illustrate its use, we will use the data collected in lecture 04 about students' ability to estimate the unit director (UD)'s body weight.

Q1: To warm up:

a. Use the function `read.csv()` to read the file `fc_weight_perception.csv` into a data frame called `df`.

b. Use `ggplot2` to plot a histogram of the data contained in variable `Estimate` of the data frame `df`. A combination of functions `ggplot()` and `geom_histogram()` is sufficient.
(*tip: function `geom_histogram()` has an optional argument `bins`, which regulates the coarseness of the division of data into bins. It is set to 30 by default, but try to test some different values - e.g., 10 or 50 - to see how that affects the plot.*)

c. You will notice a couple of few values that seem clearly to be either "joke" values ($\approx 120kg$) or incorrect recording ($0kg$). Use `dplyr`'s function `filter()` to remove these *outliers*: remove any rows with values of `Estimate` lower than 30kg or higher than 110kg. Store this into variable `df2`

You now have the (slightly cleaned-up) data required to test the hypothesis that the mean (i.e., the expected value) of the estimated body weight of the fully-clothed UD by data science students corresponds to the actual weight measured by a well-calibrated scale. If you get the summary of this data (using `summary(df2)`) you should see this:

```
##        Id            Estimate
##  Min.   :  1.00   Min.   : 57.00
##  1st Qu.: 39.50   1st Qu.: 70.00
##  Median : 76.00   Median : 74.00
##  Mean   : 76.76   Mean   : 74.19
##  3rd Qu.:113.50   3rd Qu.: 78.00
##  Max.   :157.00   Max.   :100.00
```

If you bring up the documentation of the R function using `?t.test`, you'll see that the function has one required input, `x`, and a number of optional inputs (which have default values):

```
> ?t.test

## This will then show in the Help tab:

## (...)
##
## Default S3 method:
# t.test(x, y = NULL,
#        alternative = c("two.sided", "less", "greater"),
#        mu = 0, paired = FALSE, var.equal = FALSE,
#        conf.level = 0.95, ...)
#
# (...)
```

For the basic (one-sample) t-test, the relevant ones are:

- `x`: *a numeric vector of data values.*
- `alternative`: *the type of alternative hypothesis. It can be either "two.sided" (default), "greater" or "less".*
- `mu`: *the value of the mean under the null hypothesis,* μ_0.
- `conf.level`: *the confidence level for computing a confidence interval. Usually set as the desired confidence of the test.*

Q2: Formalising your hypotheses

a.: Formalise the null and alternative hypotheses being tested in the standard H_0, H_1 format. Assume that the value of the mean under the null hypothesis is $\mu_0 = 81kg$, and that you want to test that against a two-sided alternative.

We are now ready to perform our t-test at the 95% confidence level.

```
mytest <- t.test(df2$Estimate,              # the sample
                 mu = 81,                   # mu_0
                 alternative = "two.sided", # type of H_1
                 conf.level = 0.95)         # confidence level
mytest

##
##  One Sample t-test
##
## data:  df2$Estimate
## t = -11.222, df = 146, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 81
## 95 percent confidence interval:
##  72.98559 75.38584
## sample estimates:
## mean of x
##  74.18571
```

Here's how to read the output of the t-test function:

- `data:  df2$Estimate`: the data used in the test
- `t = -11.222`: the value of the t_0 statistic, calculated as $(\bar{X} - \mu)/(s/\sqrt{N})$
- `df = 146`: the number of degrees-of-freedom of the test
- `p-value < 2.2e-16`: the p-value (check the lecture slides for the interpretation of the p-value)
- `alternative hypothesis`: the statement of the alternative hypothesis used (in this case, `true mean is not equal to 81`)
- `95 percent confidence interval`: the CI at the stated confidence level (in this case, `72.98559 75.38584`)
- `sample estimates`: the relevant sample statistic(s) computed from the data (in this case, the `mean of x: 74.18571`)

Some tips for interpreting these results:

Note that the result of the test does not tell you directly if H_0 was rejected or not - as is common in R, it assumes that you know what you are doing. There are two pieces of information you can use to check whether to reject (or not) H_0:

- Recall from the lecture that we reject the null hypothesis at a confidence level $1-\alpha$ if $p \leq \alpha$. In this case, we see `p-value < 2.2e-16`: much lower than the $\alpha = 0.05$ of a 95% confidence test, so we have rejected H_0.
- There is a certain equivalence between hypothesis testing and confidence intervals that can also be used: If a $(1-\alpha)$ confidence interval does not contain the value the parameter under the null hypothesis, then the corresponding hypothesis test is rejecting H_0 at that confidence level. In this example, the 95% CI was returned as $[72.986, 75.386]$, which does not contain $\mu_0 = 81$, so the null hypothesis is rejected at that confidence level.

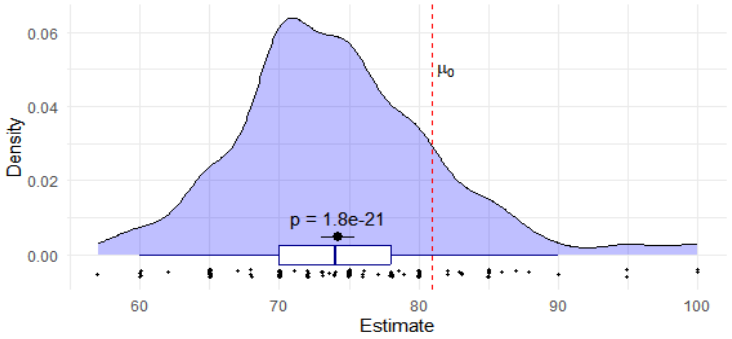
If you need to extract the CI, p-value, estimate etc. programmatically, you can access those values from the returned list object `mytest`. If you start typing `mytest$` in the console and then press tab, it will show you a list of all fields you can access. For instance, for the confidence interval:

```
mytest$conf.int

## [1] 72.98559 75.38584
## attr(,"conf.level")
## [1] 0.95
```

Besides having this information numerically, you can also use this to enrich your visual analysis. For instance, it is not uncommon to plot confidence intervals and p-values together with data summaries, or sometimes even with the raw data. A visual analysis strategy known as RDI ("Raw data, Descriptive statistics, and Inference) is often considered good practice when reporting data analyses. Here's an example (try running it in your own console and changing some of the parameters):

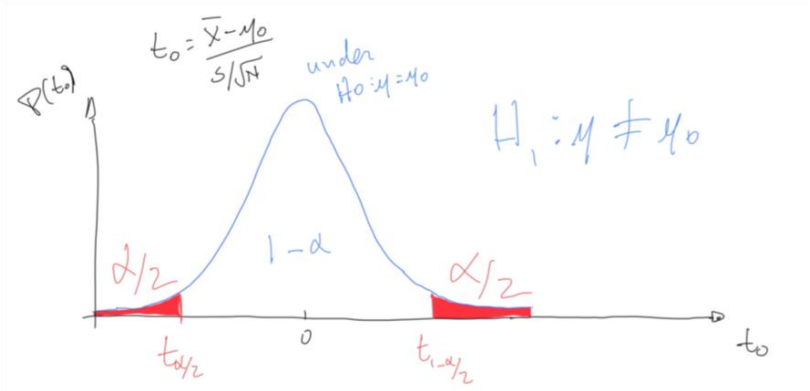
```
library(ggplot2)

ggplot(df2, aes(x = Estimate)) +
  # Density plot
  geom_density(fill = "blue", alpha = 0.25) +
  # Boxplot
  geom_boxplot(width = 0.005, outliers = FALSE,
               col = "darkblue") +
  # Plot points with vertical jittering
  geom_jitter(aes(y = -0.005), height = 0.001, size = .75) +
  # H0 value line
  geom_vline(xintercept = mytest$null.value, linetype = 2, col = "red") +
  # Add confidence interval
  annotate("pointrange", y = .005, x = mytest$estimate,
           xmin = mytest$conf.int[1], xmax = mytest$conf.int[2]) +
  # Add p-value
  annotate("text", x = mytest$estimate, y = 0.01,
           label = paste0("p = ", signif(mytest$p.value, 2))) +
  # Add text annotation
  annotate("text", x = mytest$null.value + 1, y = 0.05,
```

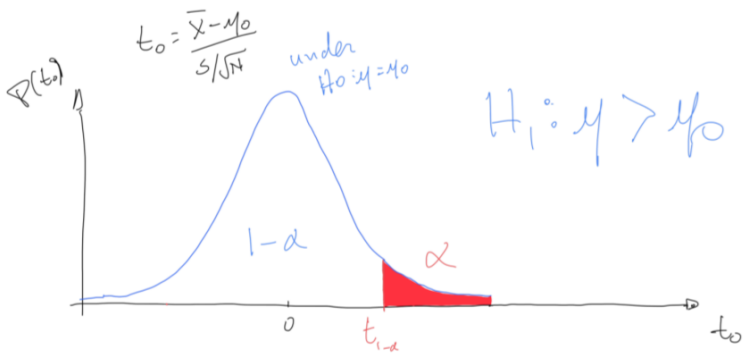
```
          label = "mu[0]",
          parse = TRUE) +
  # Minimal theme
  theme_minimal() +
  # y axis name.
  ylab("Density")
```



In some contexts, the specific direction of the effect is of interest. For example, we might be interested in finding out whether the mean execution time of a new algorithm is faster (not just "different") than an old one.

A one-sided alternative changes the rejection region, as the whole type-I error region becomes concentrated only on the side of interest. You can check the difference below:





To run t-tests with one-sided alternatives in R, just set the option `alternative` to the appropriate value ("less" if you want to use $H_1: \mu < \mu_0$ or "greater" if you want $H_1: \mu > \mu_0$).

Q3: One-sided t-tests

In the paper *Estimating the Limits of Organism-Specific Training for Epitope Prediction* (LINK), the authors explore the effect of specific data filtering and data consolidation methods on the performance of a predictive model applied to the problem of epitope prediction. Part of the data reported in that paper, related to the performance of one of the baseline methods tested, is provided in file "EBV-200-performance.rds".

For practical purposes, a method is considered useful for that application if it has an average *positive predictive value* (PPV) ***greater than 0.5***, and an average *negative predictive value* (NPV) also ***greater than 0.5***.

You have two different hypotheses to test, one related to PPV and one to NPV.

a. Use the function `readRDS()` to read that file into a variable called `df.comp`.

b. Formalise the null and alternative hypotheses being tested in each case.

c. Use the `t.test` function to test the hypotheses on the mean PPV at the 99% confidence level. Store the result of your test in a variable called `mytest.ppv`, and inspect the result of this test. Is the result *statistically significant* (i.e., was H_0 rejected)? Make sure you also check the sample mean $\bar{X}$ and the 99% confidence interval.

d. Repeat the same test (with the same confidence level) for the mean NPV, and store the result of your test in a variable called `mytest.npv`. Inspect the result of this test too - is it statistically significant, what are the sample mean $\bar{X}$ and the 99% confidence interval?

e. Based on the result of your tests and on the definition of what makes a useful test provided at the start of this question, can we consider the baseline model tested as a useful method?

The power of a test is the probability of correctly rejecting H_0 when it is false. For the t-test, power depends on a number of factors:

- the true effect size (i.e., the magnitude of the difference between μ and μ_0)
- the variability of the data, expressed as the standard deviation σ.
- the sample size N
- the significance level α
- the directionality of the alternative hypothesis H_1.

R provides simple power analysis tools for the t-test, in the form of function `power.t.test`. We can use this function, e.g., to calculate the required N to achieve a desired power, to examine the sensitivity of the test to detect a certain effect size, etc. If you look at the help available for this function you'll see that it also has a number of parameters:

```
> ?power.t.test

## (...)
# power.t.test(n = NULL, delta = NULL, sd = 1, sig.level = 0.05,
#              power = NULL,
#              type = c("two.sample", "one.sample", "paired"),
#              alternative = c("two.sided", "one.sided"),
#              strict = FALSE, tol = .Machine$double.eps^0.25)
```

The simple t-tests we are running this week are of type "one-sample". The other relevant parameters are:

- `n`: the sample size.
- `delta`: the difference between the true mean and the mean under H_0, $\delta = \mu - \mu_0$.
- `sd`: the populational standard deviation, σ.
- `sig.level`: the significance level, α.
- `power`: the power, $\pi = 1 - \beta$.

Exactly one of the parameters `n`, `delta`, `power`, `sd`, and `sig.level` must be passed as `NULL`, and that parameter is determined from the others. For instance, to estimate the power of a one-sample, two-sided t-test with $N = 20$, $\delta = 0.5$, $sd = 1$, and $\alpha = 0.05$, we'd do:

```
dummy_power <- power.t.test(n = 20, delta = 0.5, sd = 1, sig.level = 0.05,
                            type = "one.sample", alternative = "two.sided")
dummy_power

##
##      One-sample t test power calculation
##
##               n = 20
##           delta = 0.5
##              sd = 1
##       sig.level = 0.05
##           power = 0.5644829
##     alternative = two.sided
```

That is, a power of approximately 0.564 to detect a difference of 0.5 in the means with 95% confidence and a sample size of 20. To obtain the required sample size to obtain a power of 80%, we'd do:

```
tmp <- power.t.test(delta = 0.5, sd = 1, sig.level = 0.05, power = 0.8,
                    type = "one.sample", alternative = "two.sided")
tmp

##
##      One-sample t test power calculation
##
##               n = 33.3672
##           delta = 0.5
##              sd = 1
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

```r
# Since you can't have fractional sample sizes, you'd need:
cat("\n Required N:", ceiling(tmp$n))

##
##  Required N: 34
```

Some important points:

- It is usually not possible to know the populational standard deviation. However, what is really important for power and sample size calculations are not the absolute values of $\delta = \mu - \mu_0$ or of σ, but instead their proportion,

$$d = \frac{|\delta|}{\sigma} = \frac{|\mu - \mu_0|}{\sigma}$$

This "normalised difference" is known as Cohen's d coefficient, and it is commonly used as an effect size indicator. When performing sample size and power calculations, it is almost always easier to set the standard deviation to `sd=1` and interpret the value of `delta` as this normalised effect size.

- All other things being equal, the power of a test is greater when H_1 is one-sided then when it is two-sided. If you check the figures just above Q3, you will notice that absolute value of the critical point for a one-sided t-test at significance level α is smaller than that of its equivalent two-sided test. If you compare the upper rejection threshold for the two-sided test,

$$t_0 = \frac{\bar{X} - \mu_0}{s/\sqrt{N}} \geq t_{1-\alpha/2}^{(N-1)}$$

with the corresponding threshold for the one-sided upper case,

$$t_0 = \frac{\bar{X} - \mu_0}{s/\sqrt{N}} \geq t_{1-\alpha}^{(N-1)}$$

You'll notice that the threshold changes from $t_{1-\alpha/2}^{(N-1)}$ to $t_{1-\alpha}^{(N-1)}$. Since the latter is always smaller than the former, it means that it is easier to reject H_0 for the directional H_1 than for the two-sided H_1 (i.e., you need a smaller effect size $\mu - \mu_0$). For the 95% confidence level (using a sample size of 20 - feel free to test others):

```r
alpha <- 0.05
N     <- 20
cat("\n Threshold for the 2-sided H1:", qt(1-alpha/2, df = N-1))

##
##  Threshold for the 2-sided H1: 2.093024

cat("\n Threshold for the upper one-sided H1:", qt(1-alpha, df = N-1))

##
##  Threshold for the upper one-sided H1: 1.729133
```

You can also check how the statistical power increases:

```
power.t.test(n=20, delta=.5, sd = 1, sig.level = .05, type = "one.sample",
             alternative = "two.sided")

##
##      One-sample t test power calculation
##
##               n = 20
##           delta = 0.5
##              sd = 1
##       sig.level = 0.05
##           power = 0.5644829
##     alternative = two.sided

power.t.test(n=20, delta=.5, sd = 1, sig.level = .05, type = "one.sample",
             alternative = "one.sided")

##
##      One-sample t test power calculation
##
##               n = 20
##           delta = 0.5
##              sd = 1
##       sig.level = 0.05
##           power = 0.6951493
##     alternative = one.sided
```

Q4: Power and sample size

You are testing whether a new ad layout **improves** (i.e., increases) the average click-through rate (CTR) by at least 0.01 (i.e., 1%) compared to the current layout. Assume the standard deviation is known to be $\sigma = 0.05$.

a. Compute the sample size required for 90% power at $\alpha = 0.05$. Store it (just the sample size, not the entire output of `power.t.test()`) as variable `myN`.

b. Build a *power curve* for this experiment: keeping all other things equal, iterate the power calculation between $n = 10$ and $n = 300$ and store the corresponding powers for each case. Build a data frame `power.df` with columns `n` and `power`, then use ggplot2 to plot the `n` versus `power` curve. Functions `ggplot()` and `geom_line()` should be sufficient for this plot, but you can try your hand at adding other plot elements such as title/subtitles (`ggtitle()`), axis labels (using `xlab()` and `ylab()`), changing the overall plot theme (using any of the `theme_*()` functions), etc.

The one-sample t-tests assume:

- Independent observations
- Approximately normal distribution of the population, or a large enough sample size such that the sampling distribution of the means becomes approximately normal.

The first assumption (independence) usually needs to be ensured via proper experimental design, or careful analysis of the data to ensure that what counts as an *observation* is an independent realisation of the data-generating phenomenon (we'll discuss this a bit more in the next lecture and lab).

For the assumption of normality, this is usually tested in a variety of ways:

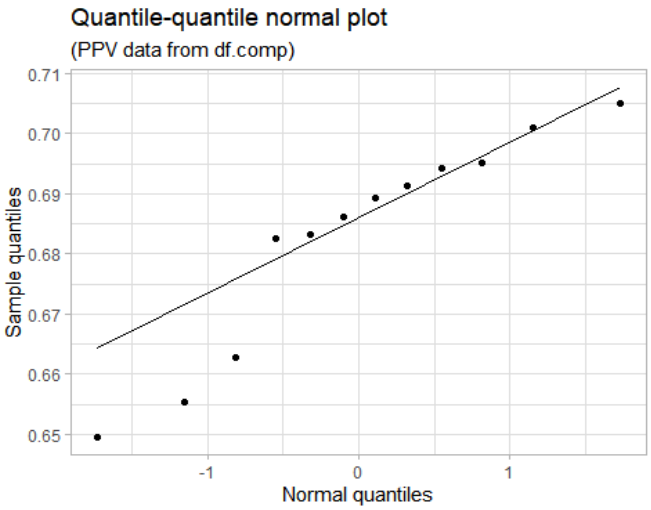
- Q–Q plots
- Inferential tests (e.g., Shapiro–Wilk normality test)
- Simulations of the sampling distribution

Before we proceed, it is important to reinforce that the assumption of normality "*does not assert that the observations within a given sample are normally distributed, nor does it assert that the values within the population (from which the sample was taken) are Normal.* ***This core element of the Assumption of Normality asserts that the distribution of sample means (across independent samples) is Normal***" Mordkoff, 2011.

Of course, if the population or the sample are approximately normal, then by definition the sampling distribution of the means will be normal for any sample size N. Therefore, a common first check is to look at the normality of the sample. If the sample is consistent with a normal variable, there's no need to check the sampling distribution of the means.

We'll use the `df.comp` data from **Q2**, and focus on variable `ppv`.

```
df.comp <- readRDS("./EBV-200-HybridA-performance.rds")
ggplot(df.comp, aes(sample = ppv)) +
  geom_qq() + geom_qq_line() +
  ggtitle("Quantile-quantile normal plot", subtitle = "(PPV data from df.comp)") +
  xlab("Normal quantiles") + ylab("Sample quantiles") +
  theme_light()
```



The closer the points fall to a straight line, the better the fit to a normal variable. In the case above, most points seem OK, but there are three points with values that are lower than the expected quantiles if the data were normal.

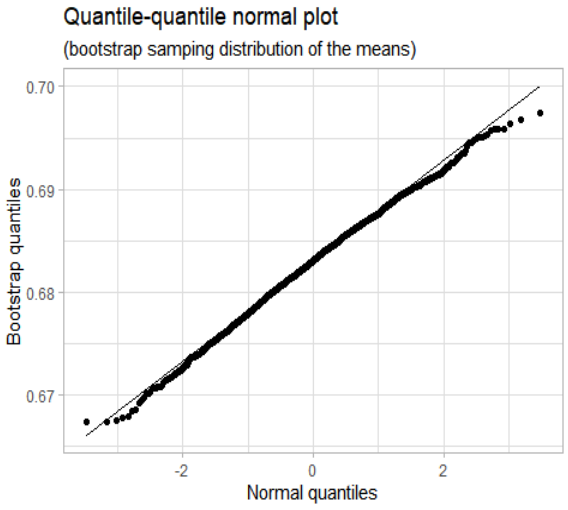
You can also run a hypothesis test of normality, using the Shapiro-Wilk test. This is an inferential test with a null hypothesis that the sample comes from a normal distribution, and an alternative hypothesis that it does not.

```
shapiro.test(df.comp$ppv)

##
##  Shapiro-Wilk normality test
##
## data:  df.comp$ppv
## W = 0.8926, p-value = 0.1273
```

(***Note**: I usually do not recommend using the Shapiro test - or other inferential tests of normality - for testing the assumptions of the t-test, especially if the sample size is large. As all inferential tests, the Shapiro-Wilk test becomes more sensitive to small deviations from normality as the sample size increases - but, at the same time, deviations from normality become less and less important for larger sample sizes, because of the CLT.*)

Finally, since the assumption of normality actually relates to the sampling distribution of the means, we can use bootstrap to estimate this directly:

```
set.seed(1234)
# Bootstrap the sampling distribution of the means, 2000 bootstrap resamples
mySDM <- replicate(n = 2000,
                   expr = {mean(sample(df.comp$ppv, replace = TRUE))},
                   simplify = TRUE)

ggplot(data.frame(x = mySDM), aes(sample = x)) +
  geom_qq() + geom_qq_line() +
  ggtitle("Quantile-quantile normal plot", subtitle = "(bootstrap samping
distribution of the means)") +
  xlab("Normal quantiles") + ylab("Bootstrap quantiles") +
  theme_light()
```



This suggests a reasonably good adherence between the sampling distribution of the means and a normal distribution, so the t-test would probably be appropriate for the PPV data in `df.comp`.

When the assumption of normality is shown not to hold, different methods (sometimes called "nonparametric methods") can be used to test hypotheses about the mean (or some other centrality parameter) of the population. One of the most commonly used nonparametric tests about the mean/median is called the *Wilcoxon Signed-Rank test*. This test (which is implemented in R as function `wilcox.test()`) is commonly assumed not to have any distributional assumptions, but this is not correct - it requires the population to be symmetrically distributed.

Another common alternative to the t-test is called the *binomial sign test*. This one (implemented in R as function really has no assumptions on the shape of the populational distribution - it only assumes independence - but it suffers from comparatively low statistical power, and it actually tests hypotheses about the median, not the mean.

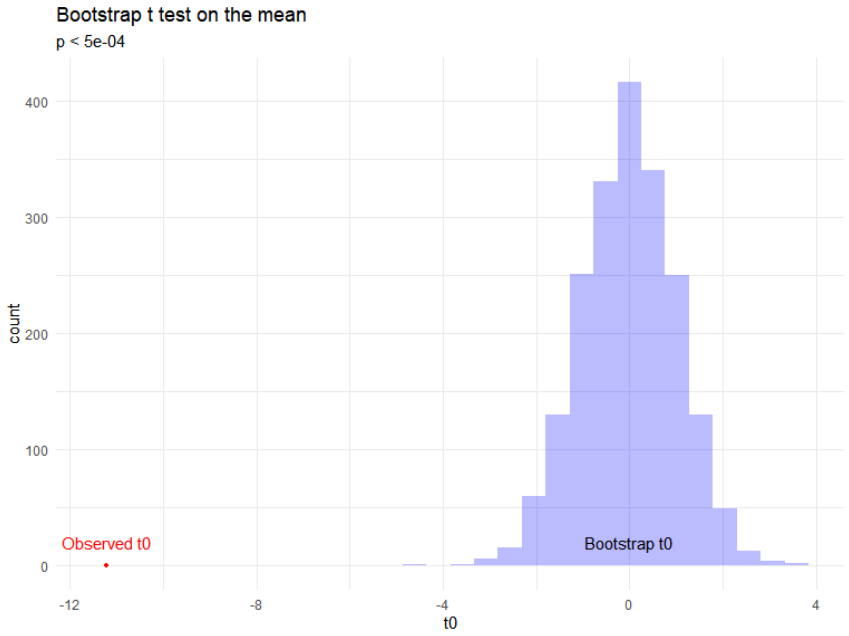
Finally, you can build a test using the *bootstrap* method, using an idea similar to the one we used to derive confidence intervals at the end of Lab 4. Similarly to the binomial sign test, bootstrap tests assume only independence of the sample, but these tests can be about the mean (or the median, or any other statistic of interest) and often have a statistical power close to the reference parametric test.

There are several different ways to perform inference using the bootstrap. One of the simplest ones is to use resampling to build an estimated distribution of the test statistic under the null hypothesis, then compare the actual observed value of the test statistic against that distribution. In summary, for a bootstrap t test:

- Compute the observed t_0 from the sample X.
- *Centre* the data on the null hypothesis ($X_c = X - \bar{X} + \mu_0$).
- Generate n_{boot} bootstrap resamples of X_c: X_c^*
- For each resample, compute a bootstrap t-statistic, $t_0^* = \frac{\bar{X}_c^* - \mu_0}{s^*/\sqrt{N}}$
- Compare the observed t_0 with the bootstrap distribution of t_0^* to estimate the p-value.

An example is provided below, using the cleaned-up weight estimation dataset from Q1, `df2`:

```r
# Value of the mean under H0
mu_0  <- 81
nboot <- 2000
X <- df2$Estimate

# Compute the observed t0 from the sample X.
t0 <- t.test(X, mu = mu_0)$statistic

# Centre the data on the null hypothesis
Xc <- X - mean(X) + mu_0

# Generate nboot bootstrap resamples of X_c
# For each resample, compute a bootstrap t-statistic
t0_star <- replicate(n = nboot,
                     expr = {
                       tmp   <- sample(Xc, replace = TRUE) # bootstrap resample
t.test(tmp, mu = mu_0)$statistic},  # bootstrap t0*
                     simplify = TRUE)

# Calculate p value from the definition:
# probability under H0 of getting a value of the test statistic at least as
# extreme as the one that was observed.
#
# For a 2-sided alternative, we need to check this probability on both sides
p <-  2 * mean(abs(t0_star) > abs(t0))

# The smallest nonzero value of p that can be calculated in a bootstrap is
# 1/nboot, so if we get zero we want to be precise with that:
p.txt <- ifelse(p == 0,
                paste0("p < ", 1/nboot),
                paste0("p = ", p))

# Histogram of the bootstrap distribution of t0 under the null hypothesis,
# vs. the actual observed value of t0
ggplot(data.frame(x = t0_star), aes(x = x)) +
  geom_histogram(fill = "#0000ff44", bins = 30) +
  annotate("point", x = t0, y = 0, col = "red", cex = 2, pch = 20) +
  # After this point it's all decoration
  annotate("text", x = t0, y = 20, label = "Observed t0", col = "red") +
  annotate("text", x = 0, y = 20, label = "Bootstrap t0") +
  xlab("t0") + ylab("count") +
  ggtitle("Bootstrap t test on the mean", subtitle = p.txt) +
  theme_minimal()
```




```
cat("\n", p.txt, "\n")

##
##  p < 5e-04
```

Q5: Challenge: putting it all together

In the paper *Sample size calculations for the experimental comparison of multiple algorithms on multiple problem instances* (LINK), the authors present an advanced method to estimate the required sample sizes when comparing multiple stochastic optimisation algorithms. One of their case studies involves the estimation of the performance of scheduling algorithms on a set of challenging problems, for which the objective was the minimisation of a time quantity called the *Makespan*. For this question, assume that an algorithm is considered useful for these problems if the expected returned *Makespan* for a new problem is **less than 240 seconds**, and that we want to check if the algorithm identified as "Full" can be useful.

a. Data loading and preparation:

- Load the data from file *UPMSP_SA_full_results.rds* into a data frame, and use `head()` to inspect its first rows.
- Use `filter()` (from package `dplyr`) to keep only the rows for which `Algorithm == "Full"`.
- In the experiment that generated this data, the algorithm was tested on several Instances, with many repeats for each instance. This is therefore a case of an experiment with multiple *repeated measurements*. If we want our inference to generalise to new problem instances, we need to make sure that the performance on each tested instance is represented only once (so that the test is run with the correct degrees-of-freedom). The most common way of doing this is aggregation by the mean: use a combination of `group_by(Instance)` and `summarise(Mean.MS = mean(Makespan))` to create a data frame `q5.df.summ` in which each instance has only one corresponding value of estimated performance, calculated as the

mean of the repeated `Makespan` observations for that instance. This resulting dataframe should have only $N = 57$ observations (one for each unique `Instance`).

b. Write down the test hypotheses H_0 and H_1. Recall from the question description that we are interested in checking if method "Full" has a mean *Makespan* lower than 240s.

c. Estimate the statistical power of the one-sample t-test to detect a (normalised) effect size of $d = 0.5$, with the test is run with significance level $\alpha = 0.05$ and a one-sided H_1. Store the power as variable `q5.t.power`.

d. Use `ggplot2` to visually check the normality of the values of `Mean.MS` in `q5.df.summ`. (`geom_qq` and `geom_qq_line()` are probably sufficient for this. Check the example earlier in this lab code for details). Is the sample consistent with a normal distribution?

e. As a baseline, run a `t.test()` on the values of `q5.df.summ$Mean.MS` (with $\alpha = 0.05$, a one-sided lower H_1, and the appropriate value of μ_0). Observe the p-value and estimated (one-sided) confidence interval (note that one-sided CIs are open intervals).

f. Now adapt the example code provided earlier and run a bootstrap t-test. Calculate the p-value and compare it against the one provided by the parametric t-test in item **e**. (Tip: think about the directionality of H_1 and how it affects the calculation of p in the bootstrap case.)