Practice 9

Elena Tuzhilina

March 14, 2023

Simulation in R

To generate a sample from a distribution we use functions with names that start with "r".

Run the following code line to read the description of the functions.

```
?rnorm
?rbinom
?rt
?runif
```

For example, the following code will generate a random sample of size 10 from normal distribution $N(10, 5^2)$

```
rnorm(n = 10, mean = 10, sd = 5)
```

```
##  [1]  9.386761 10.091237  6.438154  5.385279 -3.026000  1.817486 14.355954
##  [8]  8.815298 17.547050 10.562980
```

and the following code will generate a random sample of size 5 from uniform distribution $Unif(-5, 5)$.

```
runif(n = 5, min = -5, max = 5)
```

```
## [1]  2.261081 -4.257286 -3.306634  2.060147  3.650199
```

Now, try to generate a normal sample several times. Note that the generation process in R is random, so every time you get a new sample!

```
rnorm(n = 10, mean = 10, sd = 5)
```

```
##  [1]  5.500731  1.556781  9.773614 14.148476  3.338902 11.535325 17.106759
##  [8]  6.752106  9.006837  4.958829
```

```
rnorm(n = 10, mean = 10, sd = 5)
```

```
##  [1]  7.665098  8.069504  4.638618 12.863541 12.363270 13.921040  5.830444
##  [8]  9.743330 13.472105  9.639360
```

```r
rnorm(n = 10, mean = 10, sd = 5)
```

```
##  [1]  2.805976 12.999724  6.210465 16.325303 11.810483  7.899196 13.336764
##  [8] 11.600243 12.331869 18.058114
```

To create a random sample that can be reproduced we use `set.seed(...)` function, which sets the "seed" of the random number generator in R. Run the following code and make sure that you get exactly the same results as in this tutorial.

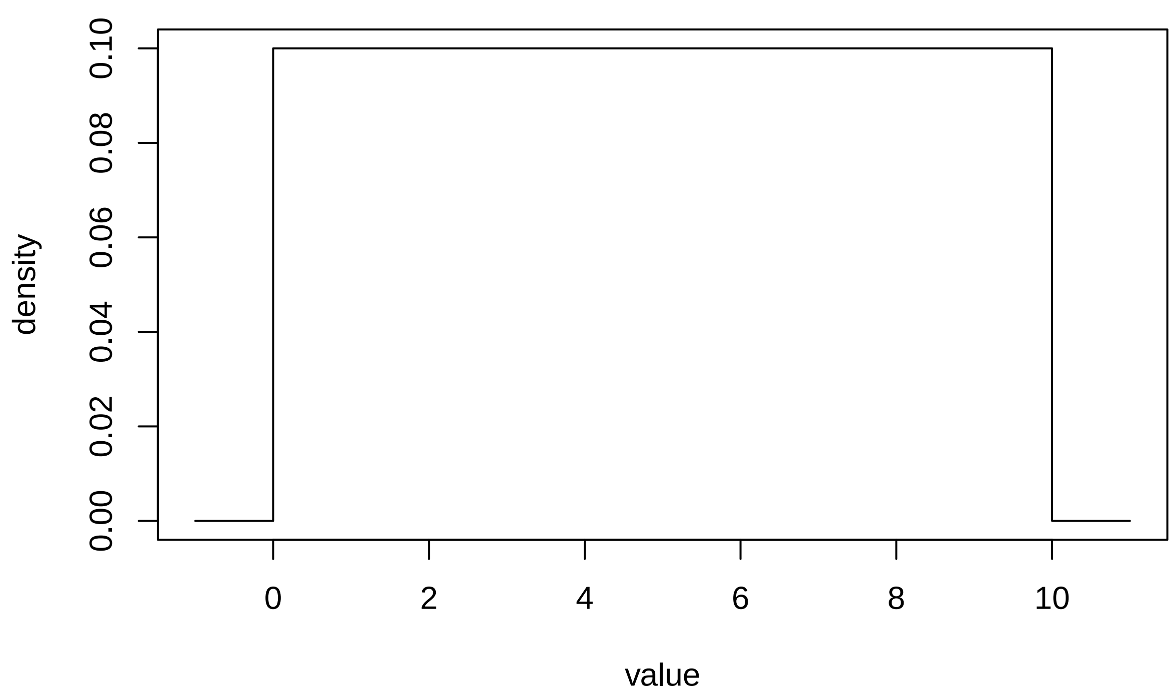
```r
set.seed(0)
rnorm(n = 10, mean = 10, sd = 5)
```

```
##  [1] 16.314771  8.368833 16.648996 16.362147 12.073207  2.300250  5.357165
##  [8]  8.526398  9.971164 22.023267
```

Central Limit Theorem

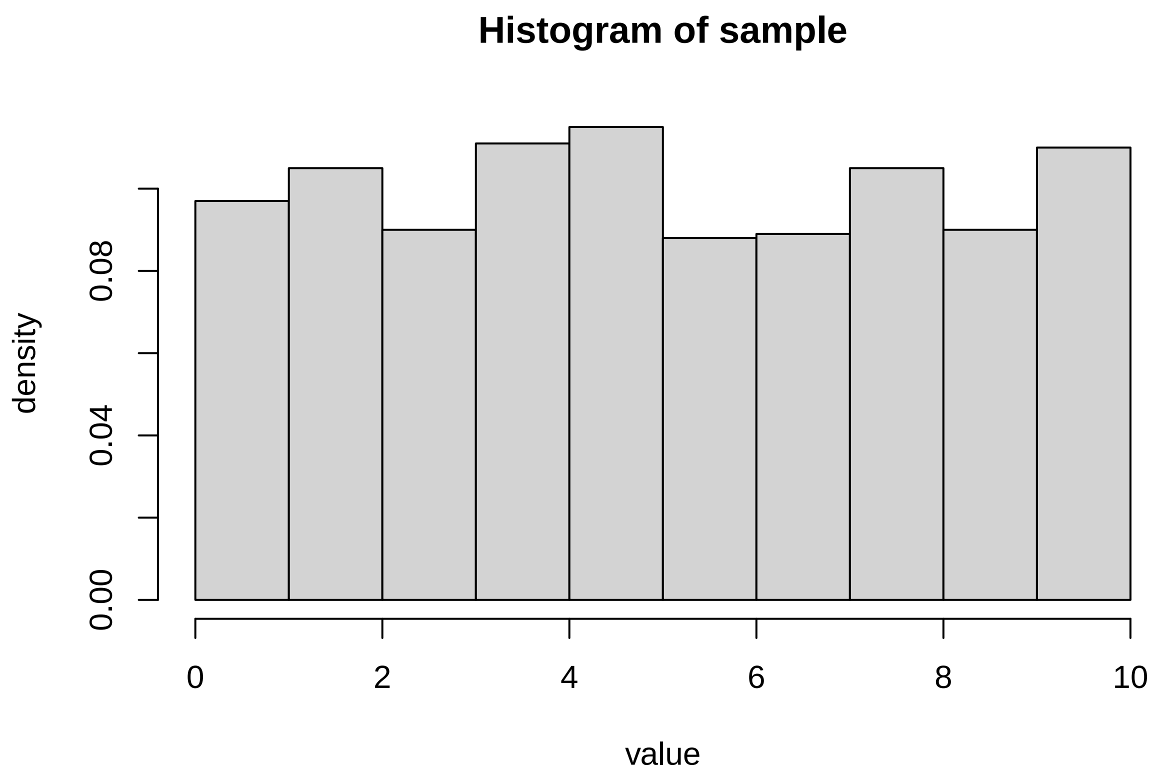
In this part we will illustrate the Central Limit Theorem for samples generated from uniform distribution $Unif(0, 10)$.

Recall that the density curve of this distribution looks like a "step" of width 10 and height 0.1.



Let's generate one large sample (of size 1000) form $Unif(0, 10)$ and plot the distribution histogram of the sample. It should look similar to the theoretical density.

```r
sample = runif(n = 1000, min = 0, max = 10)
hist(sample, freq = FALSE, breaks = 10, xlab = "value", ylab = "density")
```



Also, recall that the theoretical values of the expectation and variance for a random variable $X \sim Unif(0, 10)$ are $\mu = E(X) = \frac{10-0}{2} = 5$ and $\sigma^2 = Var(X) = \frac{(10-0)^2}{12} = 8.3333$.

```r
mu = (10 - 0)/2
mu
```

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

```r
sigma2 = (10 - 0)^2/12
sigma2
```

```
## [1] 8.333333
```

Let's check if these values are consistent with the sample mean and variance.

```r
mean(sample)
```

```
## [1] 5.000792
```

```r
var(sample)
```

```
## [1] 8.355622
```

According to CLT, for large n the distribution of the sample mean $\bar{X}$ is approximately normal $N(\mu, \frac{\sigma^2}{n})$, which, in our case, should be $N(5, \frac{8.3333}{1000})$.

In general, to find the values of a density curve, use functions that start with "d", e.g.

```r
?dnorm
?dbinom
?dt
?dunif
```

To plot the theoretical CLT distribution we first select a grid of values where the density curve will be evaluated.

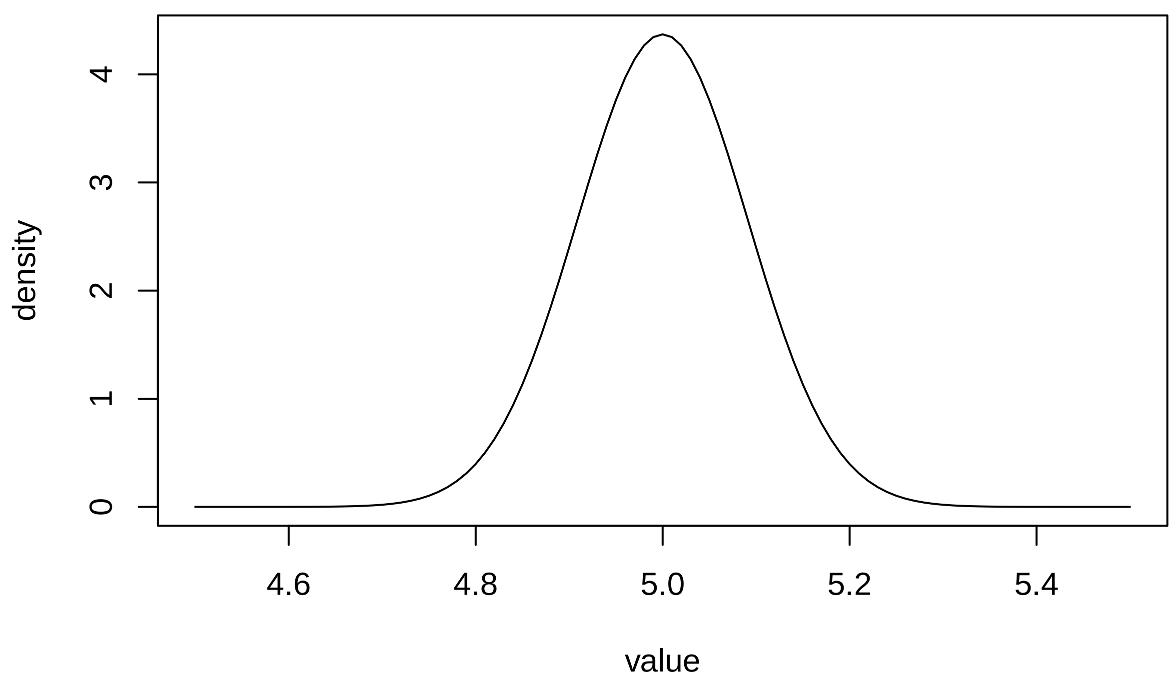
```r
grid = seq(4.5, 5.5, 0.01)
```

We use `dnorm(...)` function to find the value of the density curve of $N(5, \frac{8.3333}{1000})$ at each point of the grid.

```r
density = dnorm(x = grid, mean = mu, sd = sqrt(sigma2/1000))
```

Finally, we can plot the density curve.

```r
plot(x = grid, y = density, type = 'l', xlab = "value", ylab = "density")
```



Let's use the random sample generation to test the CLT statement for $n = 1000$. The following code generates one sample of size 1000 form $Unif(0, 10)$, and computes the mean of the sample.

```
mean(runif(n = 1000, min = 0, max = 10))
```

```
## [1] 4.877456
```

Again, if you re-run the above line multiple times you will get different values of the sample mean (because every time R generates a sample at random), but all of these values will be close to the theoretical expectation $\mu = 5$.

```
mean(runif(n = 1000, min = 0, max = 10))
```

```
## [1] 4.917563
```

```
mean(runif(n = 1000, min = 0, max = 10))
```

```
## [1] 4.958312
```

```
mean(runif(n = 1000, min = 0, max = 10))
```

```
## [1] 5.076849
```

Instead of copy-pasting the code three times, we can use `replicate(...)` function. Read the description of this function first:

```
?replicate
```

Essentially, `replicate(n = ..., expr = ...)` function re-runs the code in the `expr` argument `n` times. For instance, the following line will return 500 sample means, where each of the sample was generated from the uniform distribution.

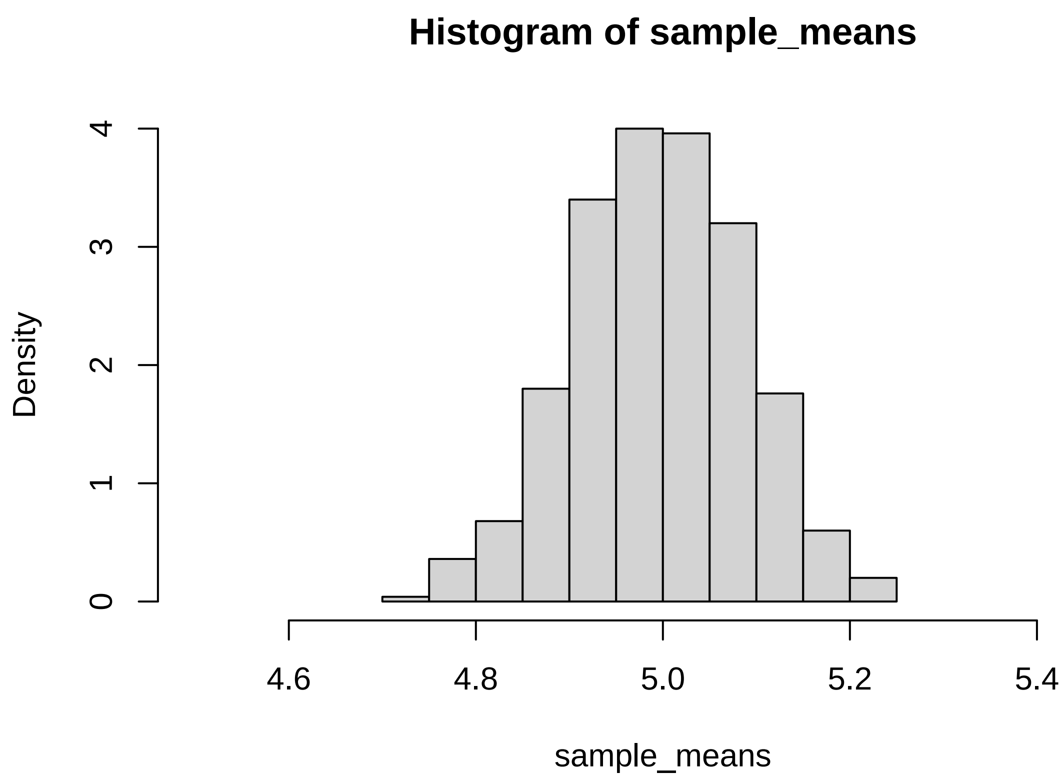
```
sample_means = replicate(n = 500, expr = mean(runif(n = 1000, min = 0, max = 10)))
sample_means
```

```
##   [1] 5.217300 5.165422 4.843427 5.018413 4.933646 5.076321 4.967200 4.927972
##   [9] 4.961912 5.005764 5.085939 4.775069 5.064276 5.088605 5.104516 4.983642
##  [17] 4.997396 4.905318 4.942314 4.872744 4.944496 4.980214 5.134307 4.843706
##  [25] 5.024972 5.011016 5.072923 5.041315 5.077371 5.019578 5.013767 4.930861
##  [33] 5.063024 4.846047 4.950929 5.192180 5.078764 5.103997 4.992412 4.921227
##  [41] 5.016025 4.968691 5.075156 5.132415 5.030373 5.094945 4.976904 4.915074
##  [49] 5.008027 4.982038 5.083267 5.101863 5.025579 4.954598 4.898961 5.084326
##  [57] 5.008723 5.047986 5.159624 5.033511 5.036293 5.059429 4.934431 4.933351
##  [65] 4.924253 5.069015 5.052752 5.057084 4.876199 4.925773 4.908488 4.998751
##  [73] 5.036396 4.947899 4.905555 4.940761 4.970123 4.938482 4.963757 5.101469
##  [81] 5.004796 4.953499 5.076905 4.921984 4.954352 4.936210 4.752722 5.027816
##  [89] 5.121969 4.921153 5.125787 4.938181 4.988424 4.905817 4.993227 4.937918
##  [97] 5.121506 4.925657 5.233869 4.945638 4.918882 4.996280 5.007090 4.941755
## [105] 4.965421 5.027092 5.152457 5.021611 4.769466 4.927260 4.914808 5.138121
## [113] 5.062017 4.921519 5.003714 5.038364 5.027586 5.065542 5.065774 4.968919
```

```
## [121] 4.935020 5.054431 5.005871 4.908535 4.924019 4.855042 5.076179 5.105352
## [129] 5.236473 5.016073 5.070181 4.864323 4.978121 5.090352 4.885587 5.000323
## [137] 5.027487 4.992319 4.832015 5.188923 5.131564 4.965701 4.908943 5.066627
## [145] 5.143136 5.026201 4.887138 5.081536 4.834420 4.992977 5.056549 4.952011
## [153] 5.007318 4.999229 5.100085 4.974104 4.928915 4.898238 4.971568 4.932503
## [161] 4.973324 4.819234 4.993121 4.975160 5.000618 5.070849 4.922183 4.961617
## [169] 5.133215 4.956828 4.909355 5.087964 4.932740 5.043739 5.195457 4.926687
## [177] 5.167399 4.994013 4.972398 5.055593 5.172230 4.932467 4.892969 5.118069
## [185] 5.043270 4.988469 5.070476 5.085567 5.012015 5.111663 4.909064 5.002360
## [193] 4.935074 5.093028 5.082830 4.892612 5.023616 5.092829 4.925345 4.854136
## [201] 5.047619 4.871046 5.076407 4.863017 5.153168 4.985827 5.125810 4.960321
## [209] 4.897078 5.001057 4.992500 5.077621 5.067592 4.966761 4.930875 4.947610
## [217] 4.947559 4.854102 5.006722 5.019524 4.893176 5.099437 5.078081 5.140764
## [225] 5.036665 4.957833 5.064297 5.091125 5.051014 5.072620 4.903325 4.861176
## [233] 5.054072 4.982208 4.954110 5.065653 5.022430 4.933200 4.974203 4.986053
## [241] 4.980907 4.845894 4.948996 4.915864 4.968373 5.041304 4.894291 4.992567
## [249] 4.918798 4.949758 5.000819 5.021279 4.994410 5.077585 5.044712 5.139652
## [257] 4.964550 4.953792 4.947632 5.020363 4.926986 4.857973 4.911075 5.070090
## [265] 5.021562 4.981289 4.971433 4.895640 4.827018 5.040485 4.889009 5.243656
## [273] 5.110604 4.923836 5.053839 5.105188 4.985393 4.966379 5.126044 4.939572
## [281] 4.995708 5.093679 4.921617 4.997481 5.005913 5.041243 4.970092 4.945560
## [289] 4.980700 5.066500 5.144286 5.094730 4.916318 4.981146 4.969164 5.051020
## [297] 5.016804 5.018206 5.036075 4.982061 4.952599 4.874060 5.139154 4.933238
## [305] 5.016511 4.946952 4.939957 5.170887 4.908628 5.004941 5.116198 4.984626
## [313] 5.118785 5.009919 4.873520 5.082115 4.949654 4.995518 5.030741 4.894851
## [321] 4.931442 4.865323 4.923871 5.013920 4.977341 5.117683 4.927689 4.994578
## [329] 5.028538 4.761615 5.034992 4.984235 5.042829 5.012016 4.990987 5.146849
## [337] 5.038041 5.007875 4.888911 4.978769 5.125958 4.863964 4.893115 5.033412
## [345] 5.016475 4.825253 5.049961 4.968168 5.026785 4.994738 5.014517 5.014009
## [353] 5.051825 5.057331 4.868613 4.944798 4.873318 4.988611 4.867318 4.940251
## [361] 4.924199 5.015969 4.943485 5.107968 4.867433 5.046453 5.057596 5.006065
## [369] 4.975251 4.983702 4.987819 4.955362 4.796693 4.896598 4.889347 5.075057
## [377] 4.973138 5.059176 5.124781 5.016600 5.105762 4.890326 5.174030 5.014188
## [385] 5.034884 5.012346 5.060907 5.082269 4.797307 4.911095 5.026603 5.111104
## [393] 5.018674 4.989036 5.093952 5.127597 5.045923 5.040408 4.988382 4.941080
## [401] 5.028816 4.955356 4.931682 4.914562 4.993173 4.839455 4.912091 5.117946
## [409] 4.879543 4.996856 4.889823 4.977463 5.033807 5.057309 5.032523 5.099290
## [417] 4.984115 4.859653 4.938532 4.890057 4.846291 4.857286 5.105378 5.040327
## [425] 5.116357 5.015412 5.126495 4.980972 4.906177 4.776651 5.024350 4.964632
## [433] 5.235392 4.957219 4.846627 4.986407 4.786100 4.904838 4.834393 5.064527
## [441] 4.986290 4.741013 5.072512 5.100946 4.973761 5.037719 4.935505 5.095819
## [449] 4.800759 4.881825 4.939409 5.046525 5.036902 4.959067 5.084667 4.887347
## [457] 4.887914 5.069749 4.812720 4.791191 5.087173 5.069333 4.993685 5.064367
## [465] 5.168320 5.067669 5.095691 4.820447 5.104252 5.004527 4.992027 5.128721
## [473] 4.846729 4.983092 4.969123 5.111080 4.945395 4.995962 5.035266 5.189729
## [481] 4.894512 5.064435 4.962645 4.908236 5.176564 5.054324 5.026688 5.058799
## [489] 5.009483 5.043745 4.856548 5.074596 5.169184 5.056804 5.006073 5.005020
## [497] 5.101635 5.025548 5.063826 4.984970
```

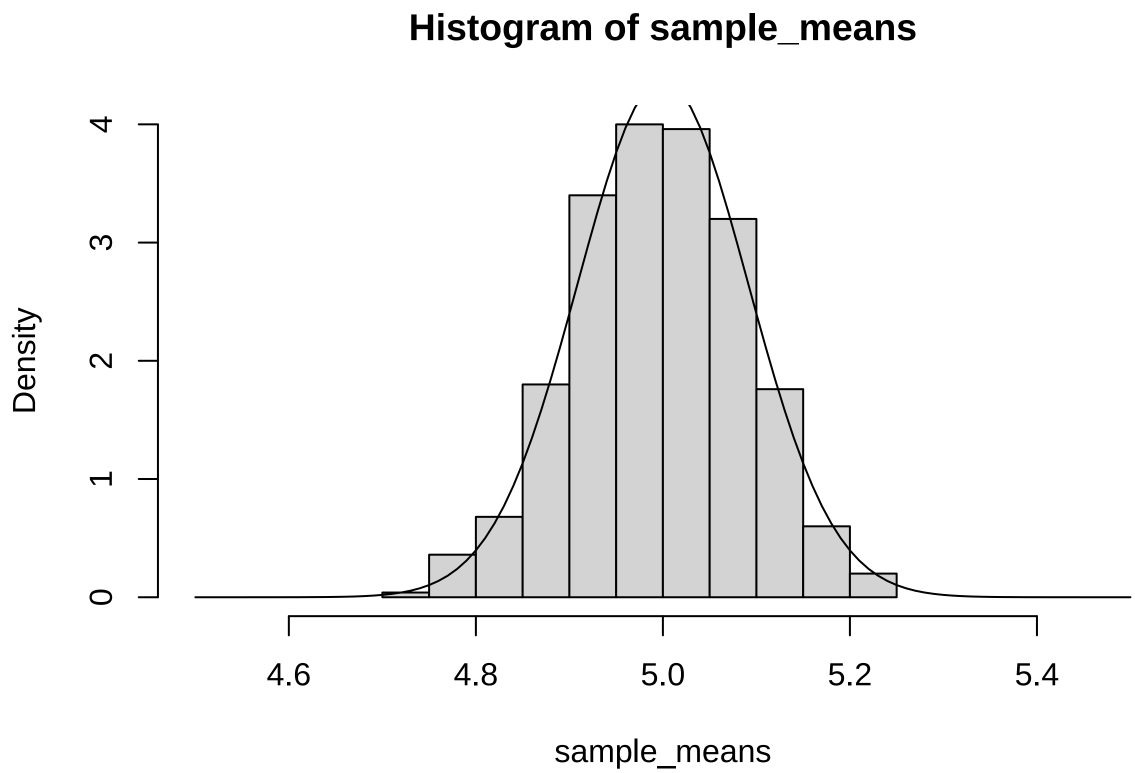
Let's plot the density histogram for the sample means (we use `xlim =` argument to set x-axis limits and make them compatible to the ones in the density curve plot).

```r
hist(sample_means, freq = FALSE, breaks = 15, xlim = c(4.5, 5.5))
```



You can also add the CLT density curve to the plot using `lines(...)` function.

```r
hist(sample_means, freq = FALSE, breaks = 15, xlim = c(4.5, 5.5))
lines(x = grid, y = density)
```



The density histogram looks like a normal distribution!

Finally, we check that the expectation and variance of the distribution histogram are consistent with $\mu = 5$ and $\frac{\sigma^2}{n} = \frac{8.3333}{1000}$.

```
mean(sample_means)
```

```
## [1] 4.996577
```

```
var(sample_means)
```

```
## [1] 0.008113006
```

Find probabilities

To find a probability $P(X \leq q)$ we use functions that start with "p".

```
?pnorm
?pbinom
?pt
?punif
```

For example, the following function will find the probability $P(X \leq -1)$ where $X \sim N(10, 5^2)$. (Use the normal distribution table form Quercus to check this result)

```r
pnorm(q = -1, mean = 10, sd = 5)
```

```
## [1] 0.01390345
```

If you want to find $P(X \geq q)$ instead, set `lower.tail = FALSE`.

```r
pnorm(q = -1, mean = 10, sd = 5, lower.tail = FALSE)
```

```
## [1] 0.9860966
```

To solve the reverse problem (find a quantile of a distribution, e.g. value a such that $P(X \leq a) = 0.25$) we use functions that start with "q".

```r
?qnorm
?qbinom
?qt
?qunif
```

For example, the following function will find the 25-th percentile of t-distribution with 10 degrees of freedom. (Use the t-distribution table form Quercus to find this quantile and compare this result to the R output)

```r
qt(0.25, df = 10)
```

```
## [1] -0.6998121
```

Statistical testing

In this part we will learn how to do statistical testing in R.

First, let's generate a sample.

```r
set.seed(0)
sample = rnorm(30, 0, 1)
sample
```

```
##  [1]  1.262954285 -0.326233361  1.329799263  1.272429321  0.414641434
##  [6] -1.539950042 -0.928567035 -0.294720447 -0.005767173  2.404653389
## [11]  0.763593461 -0.799009249 -1.147657009 -0.289461574 -0.299215118
## [16] -0.411510833  0.252223448 -0.891921127  0.435683299 -1.237538422
## [21] -0.224267885  0.377395646  0.133336361  0.804189510 -0.057106774
## [26]  0.503607972  1.085769362 -0.690953840 -1.284599354  0.046726172
```

To conduct statistical testing for the population mean μ we use `t.test(...)` function.

```r
?t.test
```

The following code will use `sample` to check $H_0 : \mu = 0$ vs. $H_a : \mu \neq 0$.

Note that you can use

- `alternative =` argument to switch between one- and two-sided alternatives;
- `conf.level =` argument to change the significance level α.

```r
t.test(x = sample, alternative = "two.sided", mu = 0, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  sample
## t = 0.13152, df = 29, p-value = 0.8963
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.3193943  0.3632959
## sample estimates:
##  mean of x
## 0.02195079
```

In the output of `t.test(...)` function you get the following information:

- the sample mean $\bar{x} = 0.02195079$
- the value of the test statistic $t_{obs} = 0.13152$
- the $p\text{-}value = 0.8963$
- BONUS: the confidence interval for μ [-0.3193943, 0.3632959]!

To run statistical testing for proportions we use `binom.test(...)` function. Note that, in this case, argument `x =` corresponds to the "number of successes" and `n =` is the "number of trials".

For example, if we got 65 successful outcomes out of 100 trials and we want to test $H_0 : p = 0.5$ vs. $H_a : p > 0.5$ at the significance level $\alpha = 0.8$ we would run the following code:

```r
binom.test(x = 65, n = 100, p = 0.5, alternative = "greater", conf.level = 0.8)
```

```
##
##  Exact binomial test
##
## data:  65 and 100
## number of successes = 65, number of trials = 100, p-value = 0.001759
## alternative hypothesis: true probability of success is greater than 0.5
## 80 percent confidence interval:
##  0.6037246 1.0000000
## sample estimates:
## probability of success
##                   0.65
```

Tn the output of `binom.test(...)` function you get:

- the sample mean $\bar{x} = 0.65$
- the $p\text{-}value = 0.001759$
- BONUS: the confidence interval for p [0.6037246 1.0000000]!