Linguistic Data: Quantitative Analysis and Visualisation

Ilya Schurov, Olga Lyashevskaya, George Moroz, Alla Tambovtseva

19 January 2019

Seminar 2: sampling from a population

File `sizes.txt` contains a list of sizes (in bytes) of all Russian Wikipedia articles on artists (retrieved 18.01.2019). We can read it into vector with `scan()` function.

```
sizes <- scan("http://math-info.hse.ru/f/2018-19/ling-data/artists-sizes.txt")
```

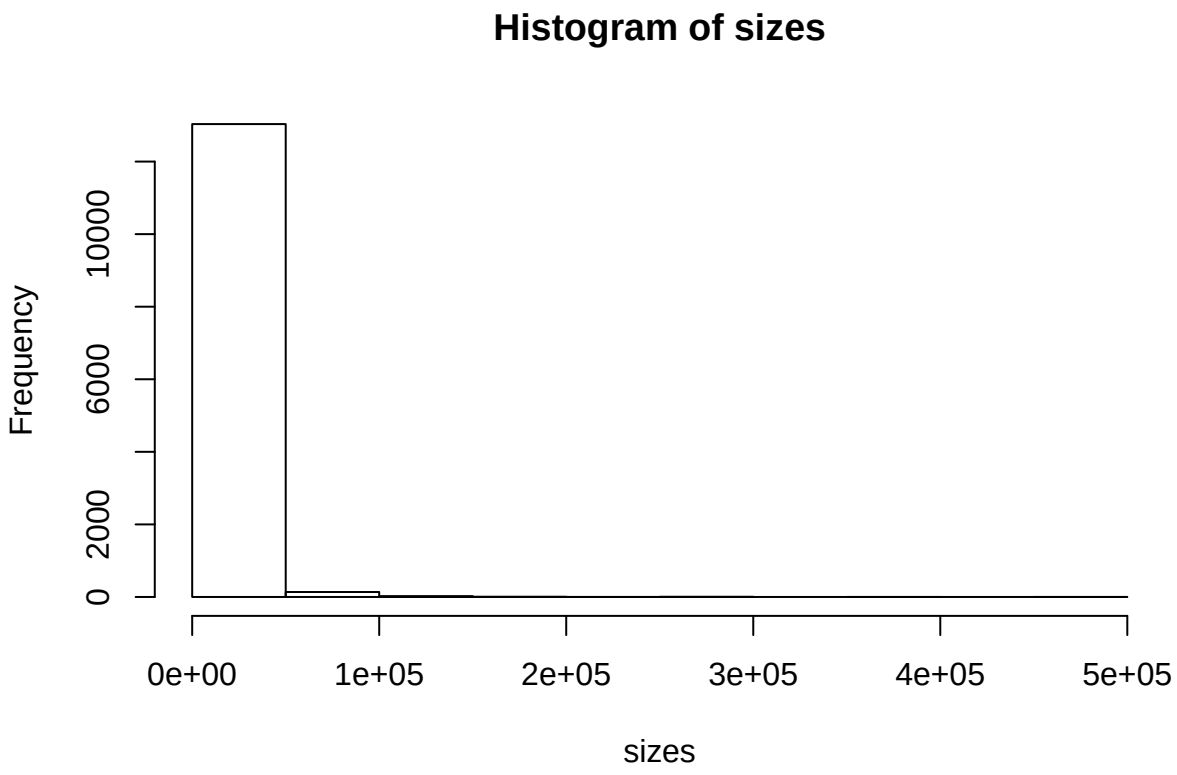
Let's look at some descriptive statistics:

```
summary(sizes)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     848    4789    7000   10427   11508  467920
```

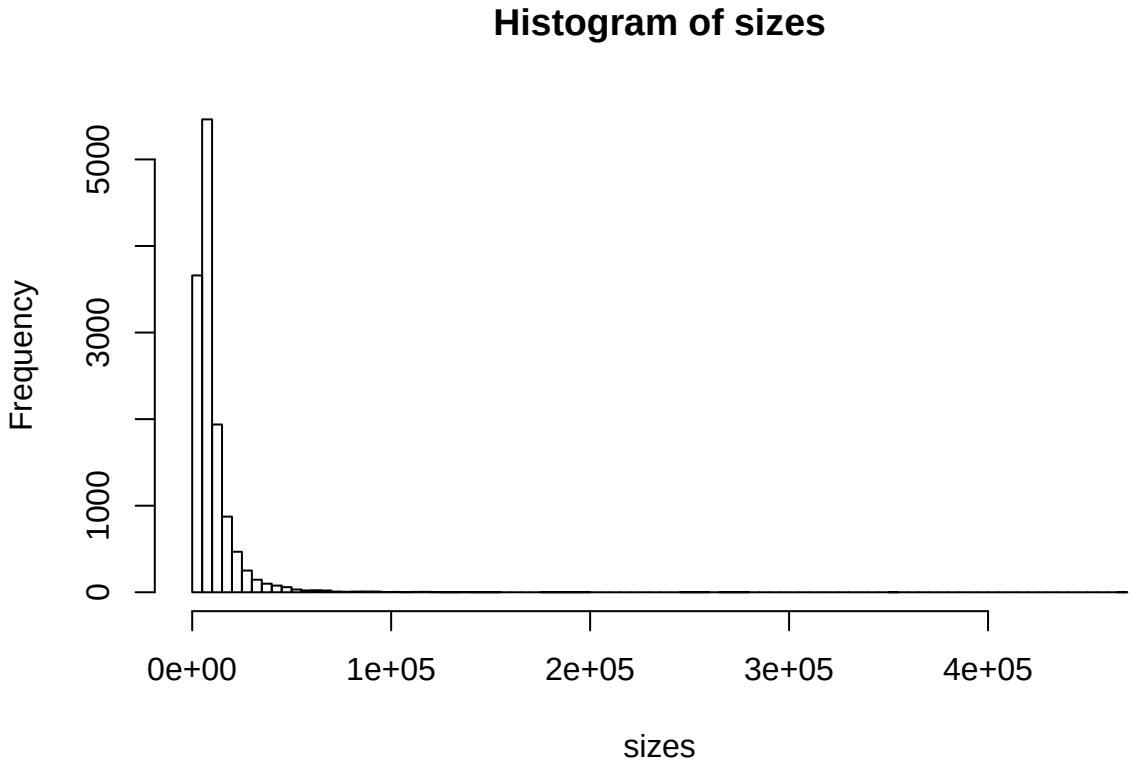
We see that there's some rather small articles (less than 1000 bytes) and also some large articles (about half a megabyte!). Let's try to visualize it using histogram:

```
hist(sizes)
```



This gives us little clues about the distribution. Let's increase number of bins.

```
hist(sizes, breaks = 100)
```



Again, the most of the picture is useless as it corresponds to very small number of very large values. What can we do?

One of the possible options is to filter our data: keep only not-so-large articles, e.g. less than 50000 bytes (50K).

```
filtered_sizes <- sizes[sizes < 50000]
summary(filtered_sizes)
```

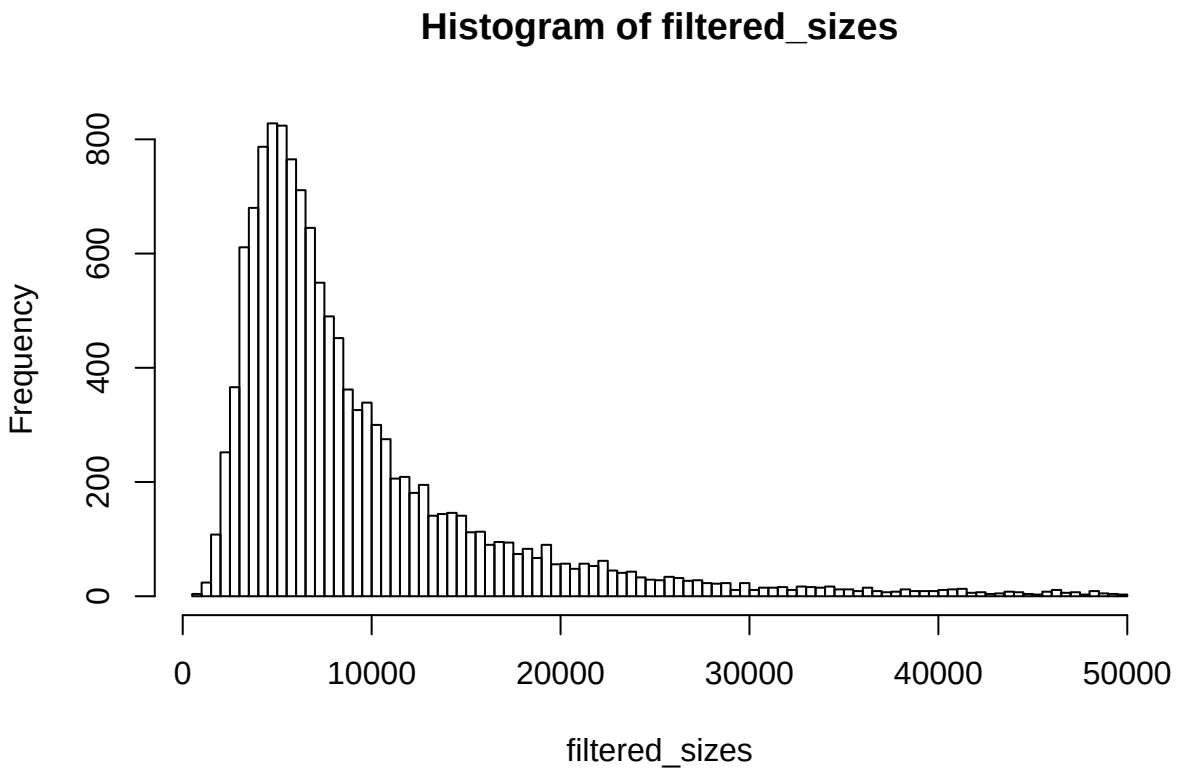
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     848    4756    6928    9390   11181   49694
```

How many elements we removed?

```
length(sizes) - length(filtered_sizes)
```

```
## [1] 174
```

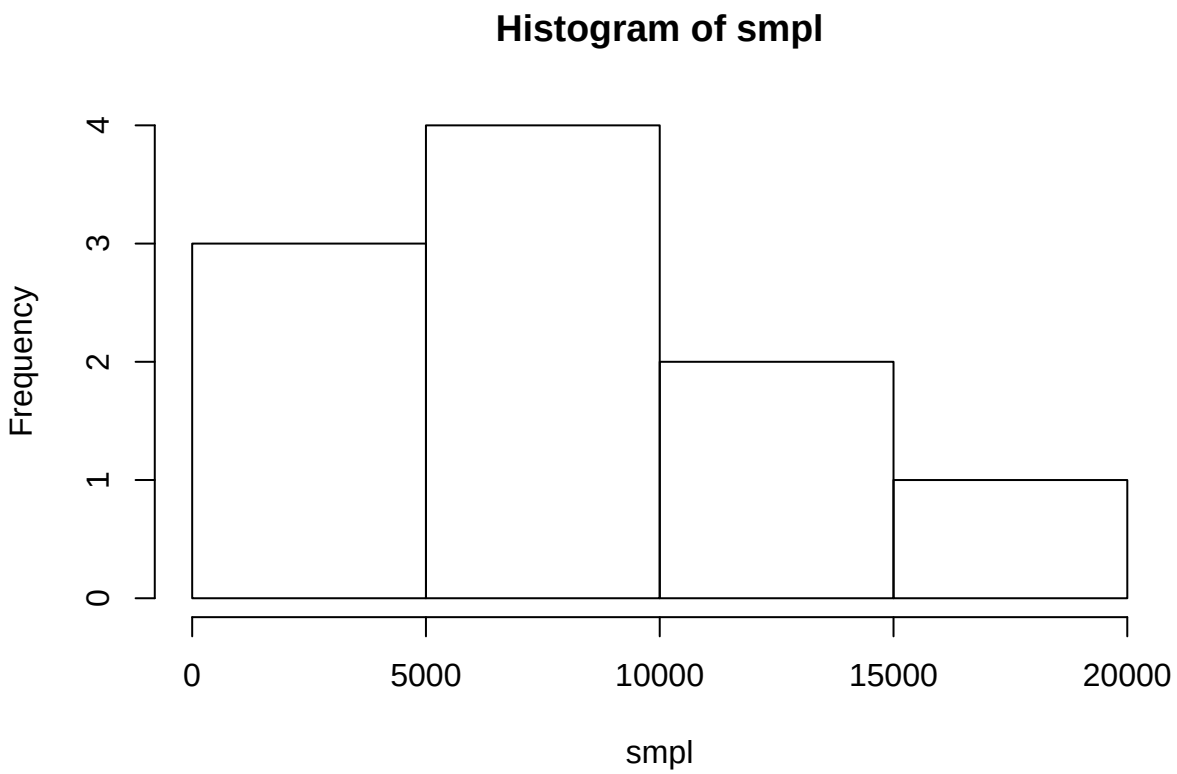
```
hist(filtered_sizes, breaks = 100)
```



This picture is nice!

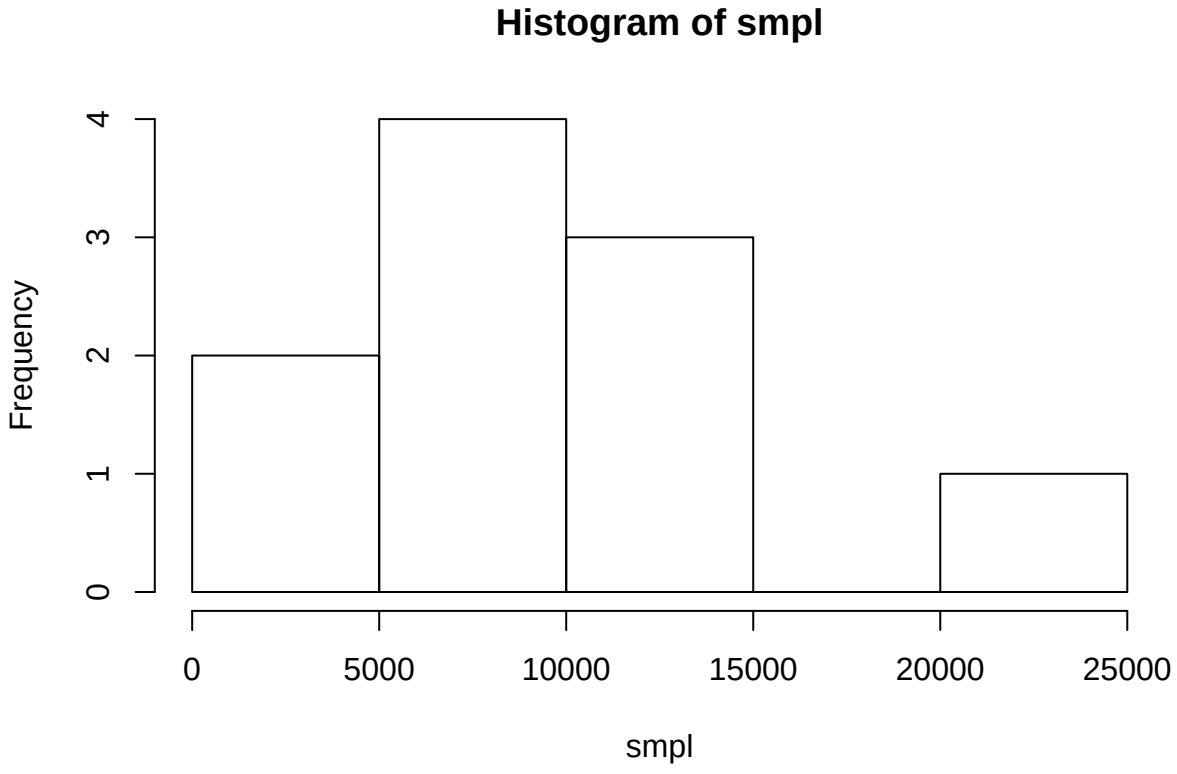
Now let us make some samples and plot their histograms. To do this we need `sample()` function. At the first place we type a name of a population, at the second place we specify a sample size. Then, if we allow repeated values in a sample, we can add the option `replace=TRUE`. It means that we put an element chosen back to the population, so it can be taken twice or even more times.

```
smpl <- sample(filtered_sizes, size = 10, replace = TRUE)
hist(smpl)
```



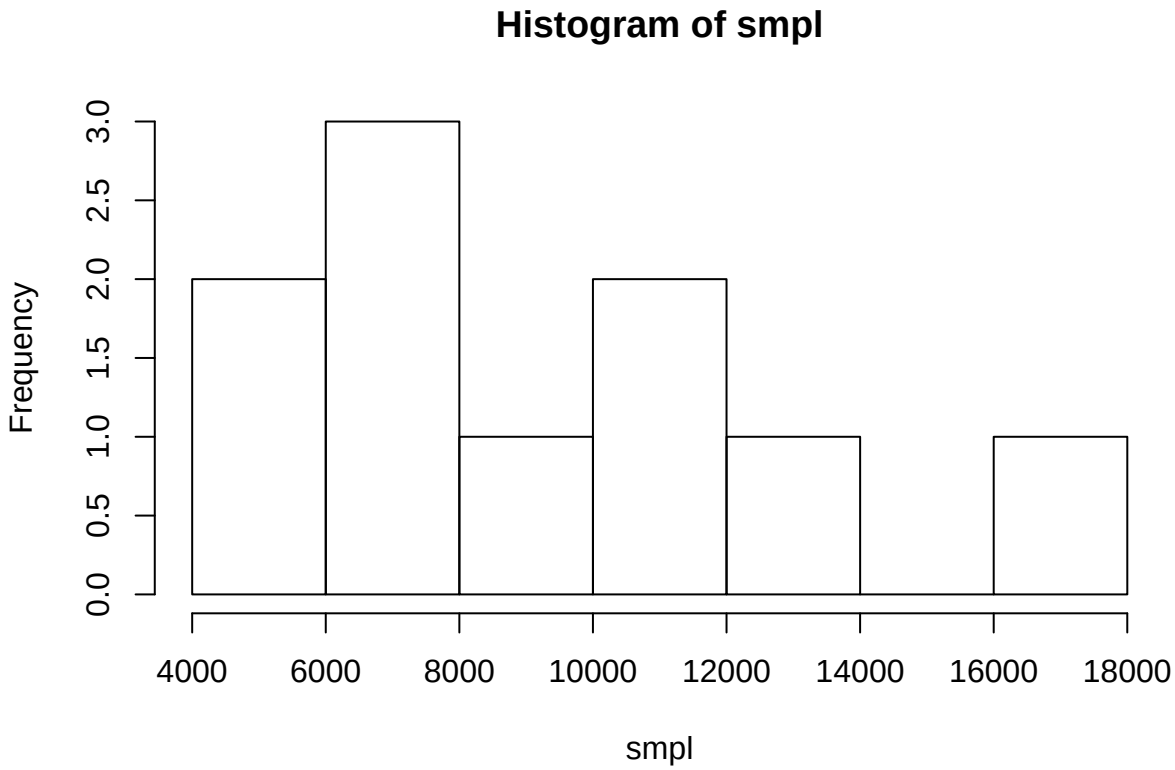
As some randomness is inherited in the process (R takes pseudo-random samples, not purely random, since algorithms of taking samples are determined), every time we will get a different sample:

```
smpl <- sample(filtered_sizes, size = 10, replace = T)
hist(smpl)
```

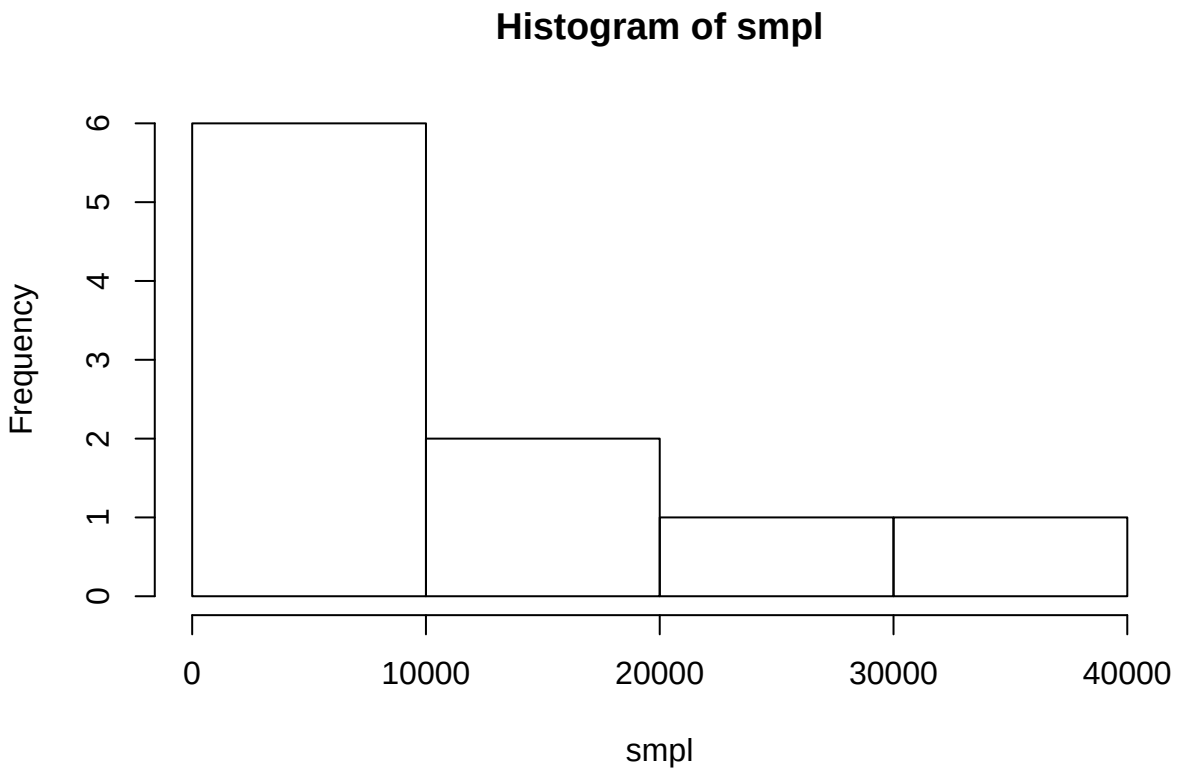


Note: value `TRUE` in R can be abbreviated as `T`, and `FALSE` as `F`, but sometimes it is better to type them in full because at some step we can create variables called `T` or `F` and forget about it.

```
smpl <- sample(filtered_sizes, size = 10, replace = TRUE)
hist(smpl)
```

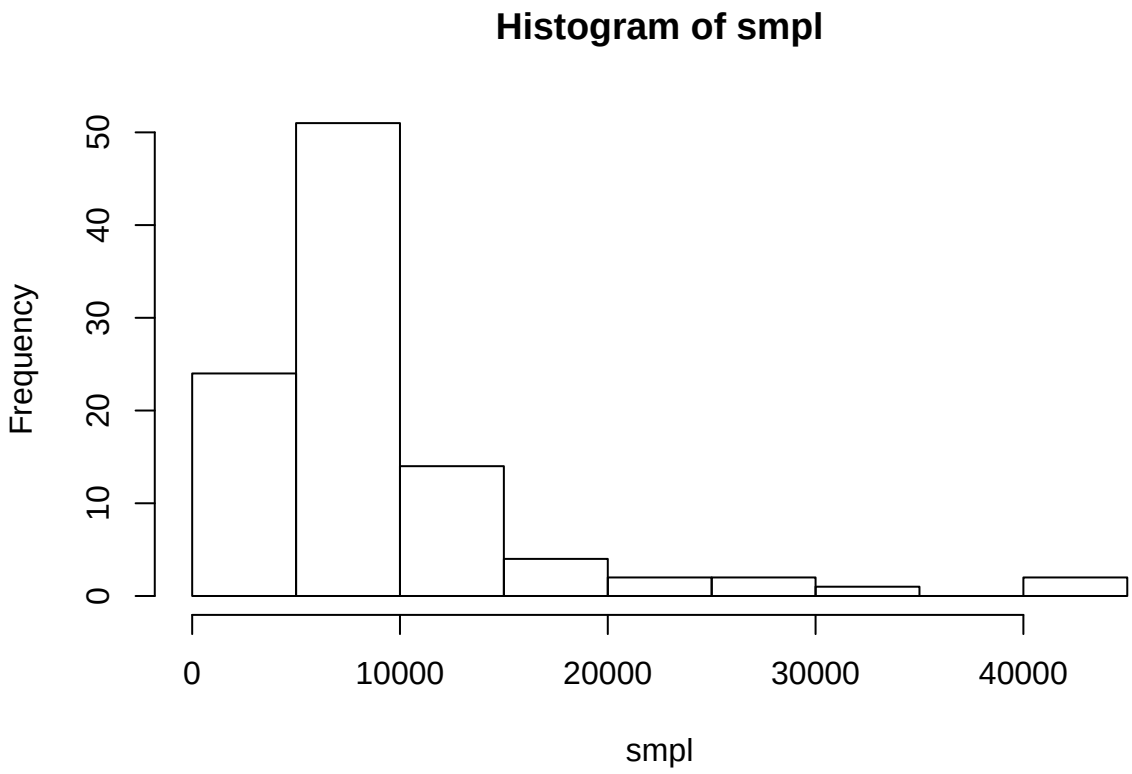



```
smpl <- sample(filtered_sizes, size = 10, replace = TRUE)
hist(smpl)
```

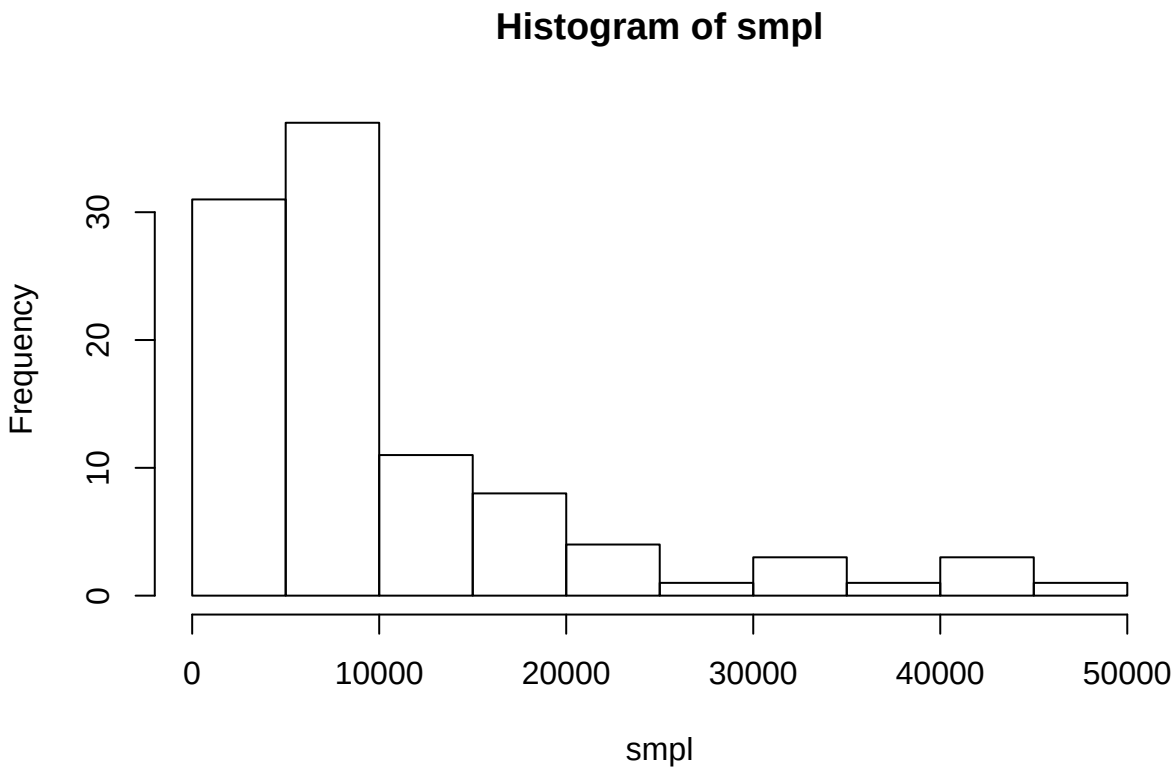


We get different histogram every time, and they are not so much close to the histogram of the initial vector. Let us increase size of a sample.

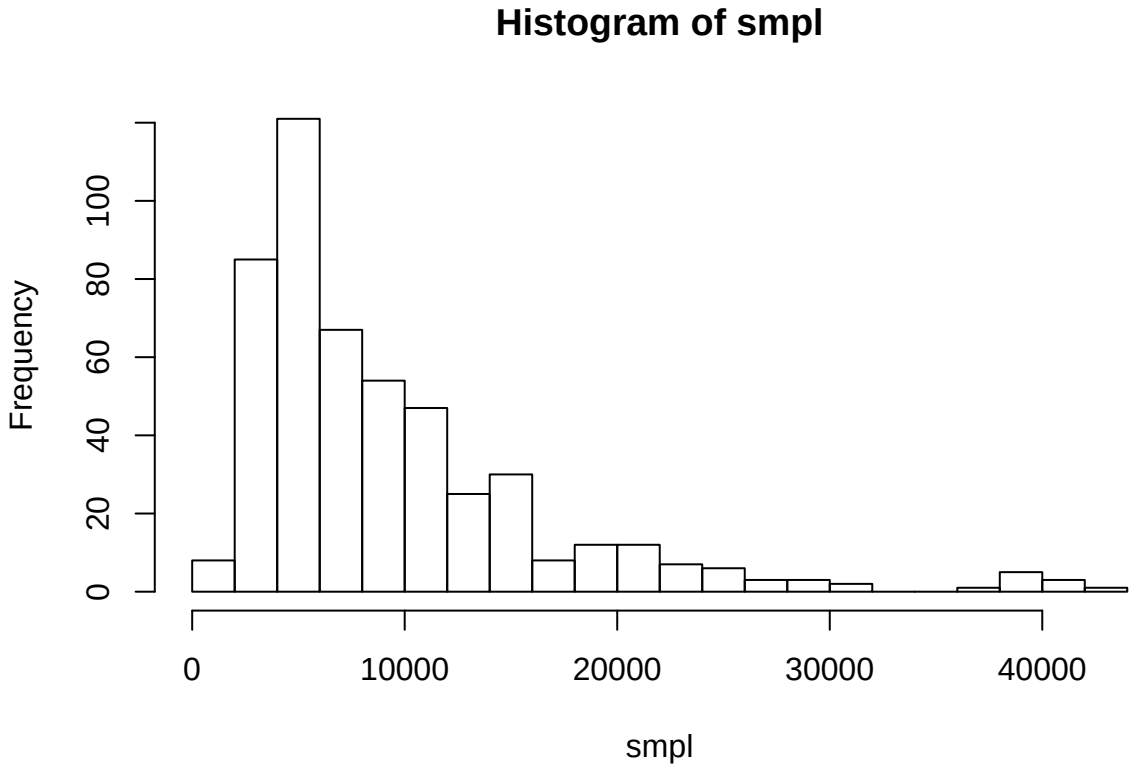
```
smpl <- sample(filtered_sizes, size = 100, replace = TRUE)
hist(smpl)
```



```r
smpl <- sample(filtered_sizes, size = 100, replace = TRUE)
hist(smpl)
```




```r
smpl <- sample(filtered_sizes, size = 500, replace = TRUE)
hist(smpl, breaks = 30)
```

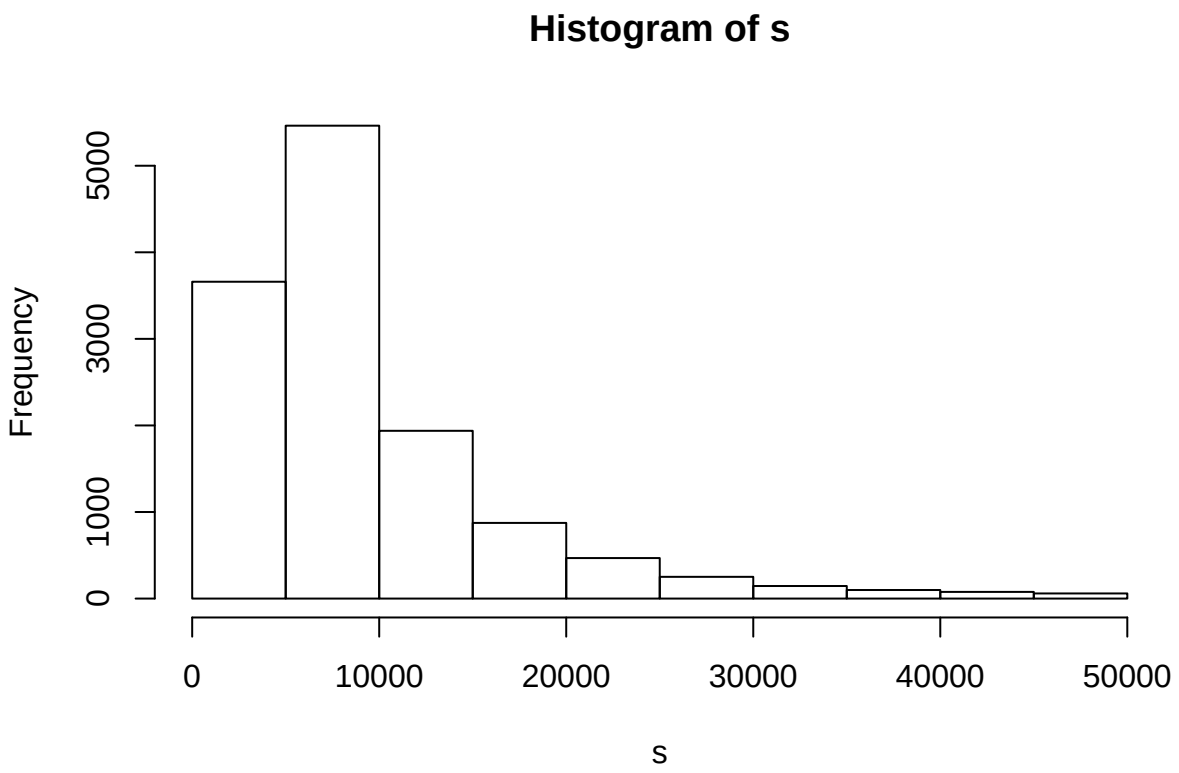


```
length(filtered_sizes)
```

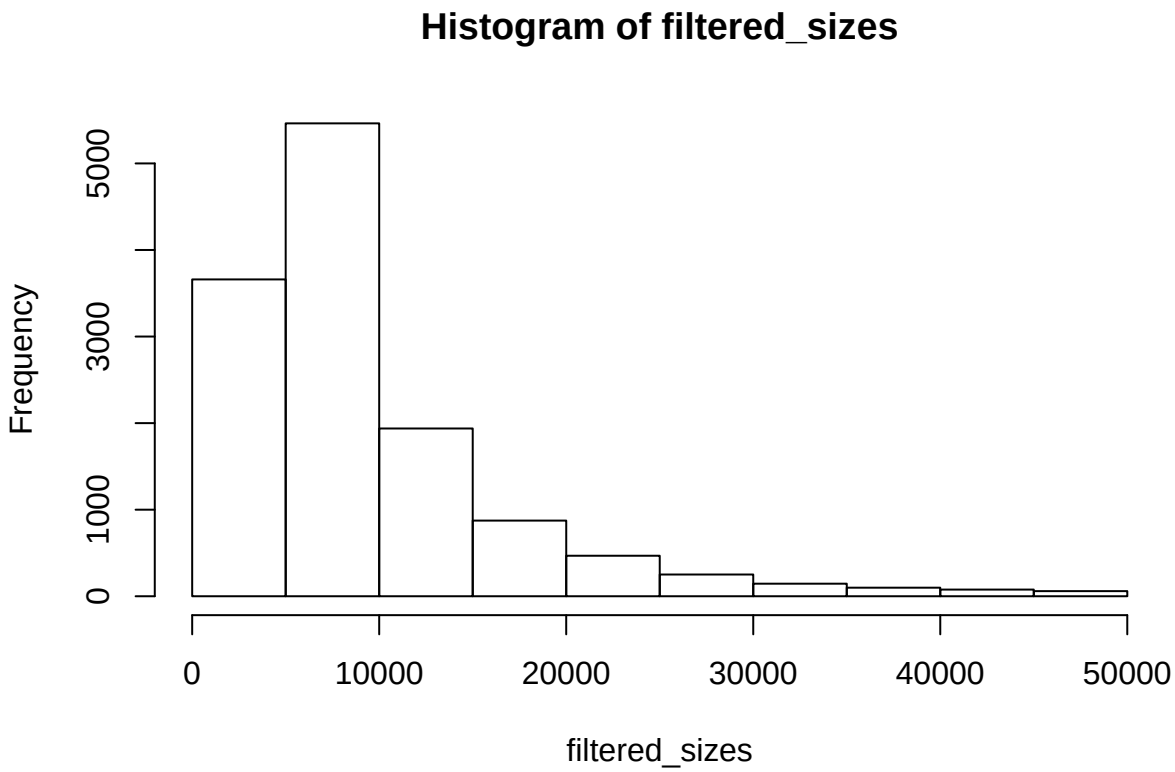
```
## [1] 13034
```

Bonus 1. At the lecture there was a question: is it true that if we take a sample of size equal to population size and add `replace=FALSE`, we will get a population itself? Yes, it is true, we can check:

```
s <- sample(filtered_sizes, size = length(filtered_sizes), replace=FALSE)
hist(s)
```

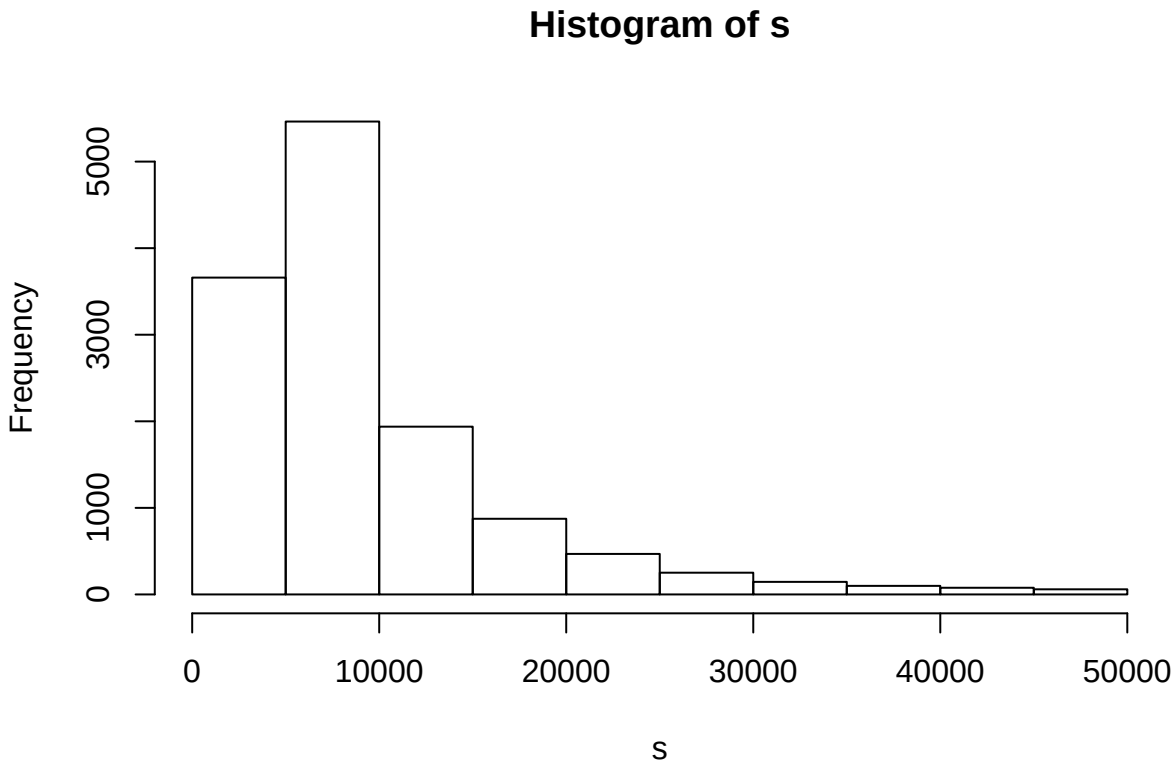



```
hist(filtered_sizes)
```



What's more, we can just skip all the options and just type:

```
s <- sample(filtered_sizes)
hist(s)
```



So, `sample()` function will return the same set of values, but in a different order, in other words, it can be

used to shuffle a vector. Look at smaller examples:

```
nums <- c(1, 2, 5, 7)
sample(nums)
```

```
## [1] 7 1 5 2
```

```
sample(nums)
```

```
## [1] 1 5 2 7
```

```
sample(nums)
```

```
## [1] 5 7 1 2
```

Bonus 2: To make our code reproducible, so all people get the same (pseudo)random samples, we can set a seed (a starting point of an algorithm):

```
set.seed(1234)
sample(filtered_sizes, size = 3)
```

```
## [1]  3273  7710 10285
```

```
# the same - due to same seed indicated
set.seed(1234)
sample(filtered_sizes, size = 3)
```

```
## [1]  3273  7710 10285
```