BIO 754 - Lecture 04

11-03-2017

Contents

Testing the Central Limit Theorem using `sample` and `sapply`

We already discussed the "law of large numbers" (LLN), indicating that sample estimates tend to converge toward expected (population) values as sample size increases. We will continue studying the LLN as well as another common law of statistics, the **central limit theorem**, using random simulations.

In many instances, we have a random variable and a population of interest (e.g. the range of a pigeon population, or the growth rate of a breast cancer cell line, the frequency of a SNP in East Asian humans). However, we don't have data from the full population and we have to estimate the mean and variance by taking random samples from the population.

How much can we understand about the population from the sample? How much will the sample's mean reflect the population mean? How will the sample's variance be related to the population variance? The theorem tells us that the sample mean, if it is large enough (e.g. >30) will be an **unbiased estimator** of the population mean, and the sample means' variance will be `1/n` of the population variance. In addition, the distribution of sample means will approach a normal distribution, irrespective of the shape of the original distribution.

Let's start examining this notion using our example of the population of heights with mean=170 and s.d.=20. We will first create a population of `10000` individuals, then take samples of 50, and check how the means of the samples are distributed compared to the expected value. Are they biased towards higher or lower values?

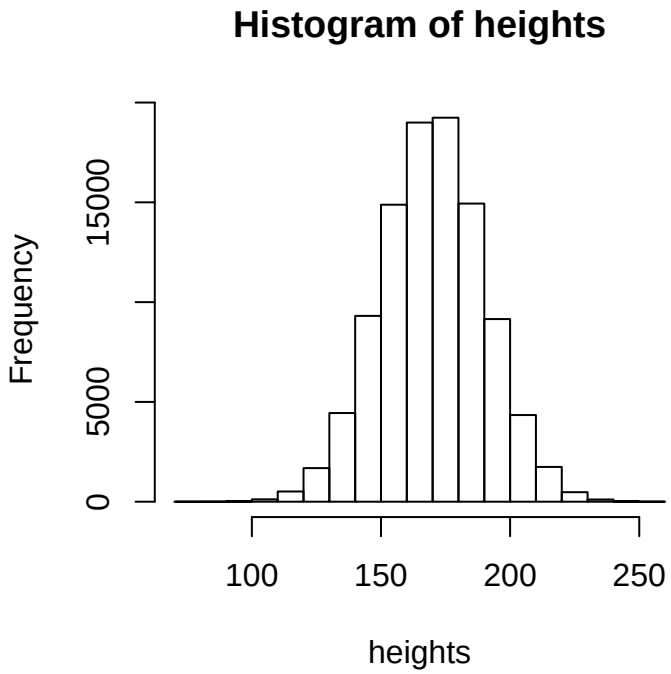
```r
set.seed(1)
heights = rnorm(100000, mean = 170, sd = 20)
mean(heights)
```

```
## [1] 169.9551
```

```r
sd(heights)
```

```
## [1] 20.07046
```

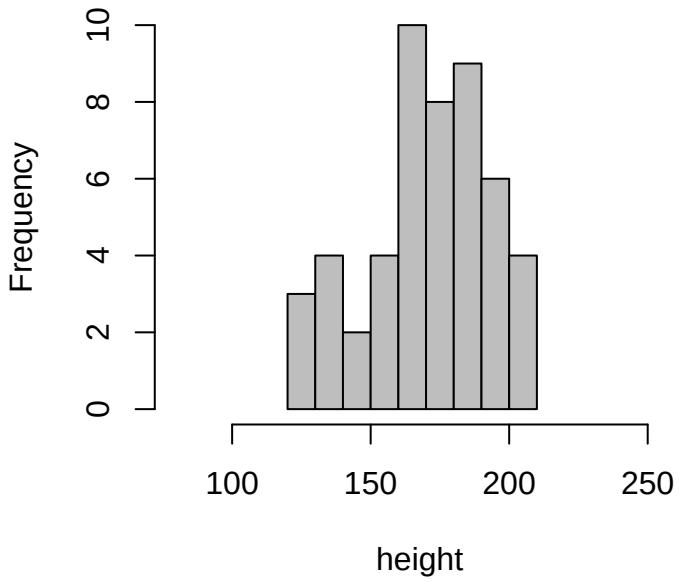
```r
hist(heights)
```




```r
# get 50 random samples, without replacement
sample(heights, 50)
```

```
##  [1] 181.5072 152.5785 170.7219 165.3029 168.8437 174.5475 145.9472
##  [8] 162.6645 125.1474 167.4261 181.8916 140.3819 156.0822 183.4041
## [15] 152.8748 136.5098 177.7254 188.2687 150.9991 189.2451 206.9393
## [22] 150.0068 155.9127 147.0260 147.5314 136.1605 196.5262 169.9828
## [29] 156.7889 184.8354 117.4149 177.2635 162.0786 146.8374 191.5555
## [36] 131.7463 162.0908 166.4008 179.5776 160.6648 136.1686 185.0787
## [43] 222.7514 161.0710 167.7023 176.9702 173.1490 217.9007 167.6799
## [50] 178.0623
```

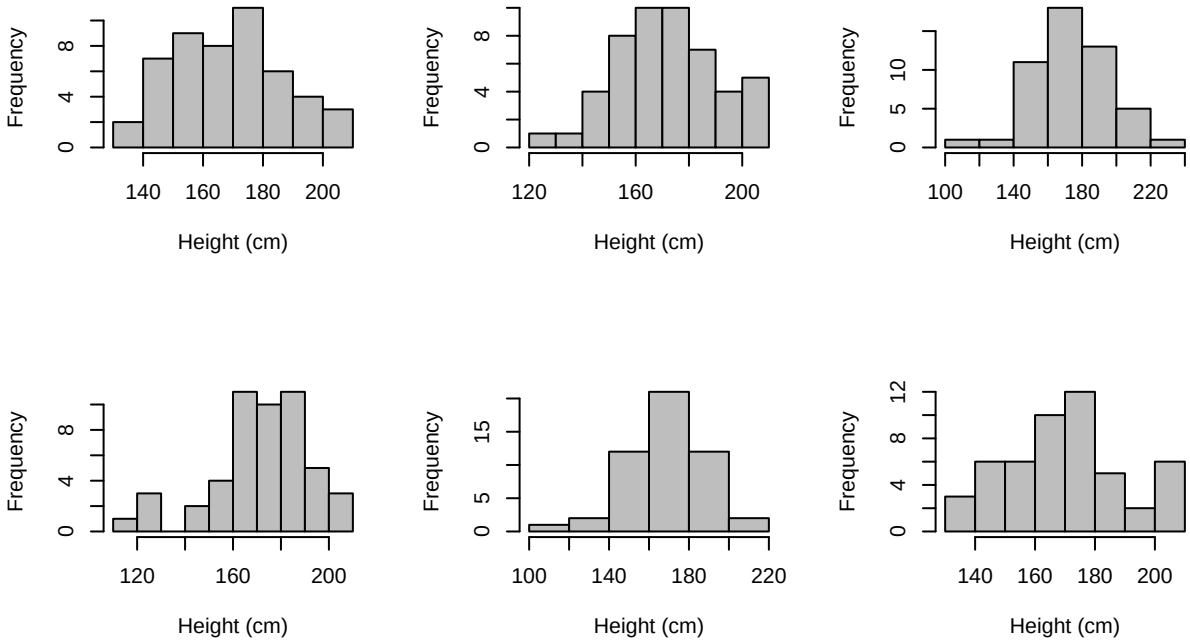
```r
sample_heights = sample(heights, 50)
hist( sample_heights, col="grey",
        xlab="height", main="",
        xlim=c(min(heights), max(heights)) )   # xlim defines the x axis limits
```



Plotting multiple random samples and improving plots using `par`, `abline`, `text`, etc

Let's repeat this 6 times, and plot all the results in a window, using `par`. `par` is used for changing different plotting parameters, and it will only show its effect once you create a plot:

```
par( mfcol = c(2,3)) # when you create plots, they will be placed in 2 rows and 3 columns,
#inside one window now the loop
sapply(1:6, function(i) {
  sample_heights = sample(heights, 50)
  hist( sample_heights, col="grey",
        xlab="Height (cm)", main="" )
})
```




```
##           [,1]             [,2]             [,3]
## breaks    Numeric,9        Numeric,11       Numeric,10
## counts    Integer,8        Integer,10       Integer,9
## density   Numeric,8        Numeric,10       Numeric,9
## mids      Numeric,8        Numeric,10       Numeric,9
## xname     "sample_heights" "sample_heights" "sample_heights"
## equidist  TRUE             TRUE             TRUE
##           [,4]             [,5]             [,6]
## breaks    Numeric,7        Numeric,8        Numeric,9
## counts    Integer,6        Integer,7        Integer,8
## density   Numeric,6        Numeric,7        Numeric,8
## mids      Numeric,6        Numeric,7        Numeric,8
## xname     "sample_heights" "sample_heights" "sample_heights"
## equidist  TRUE             TRUE             TRUE
```

We could beautify this graph in different ways. One would be adding unique titles to each graph. For this we can use the string concatenation function called `paste` (we'll learn more on this shortly):

```
# paste is for joining strings
paste("a", "b")
```

```
## [1] "a b"
```

```
paste("a", "b", sep="")
```

```
## [1] "ab"
```

```r
paste("a", "b", sep="*")
```

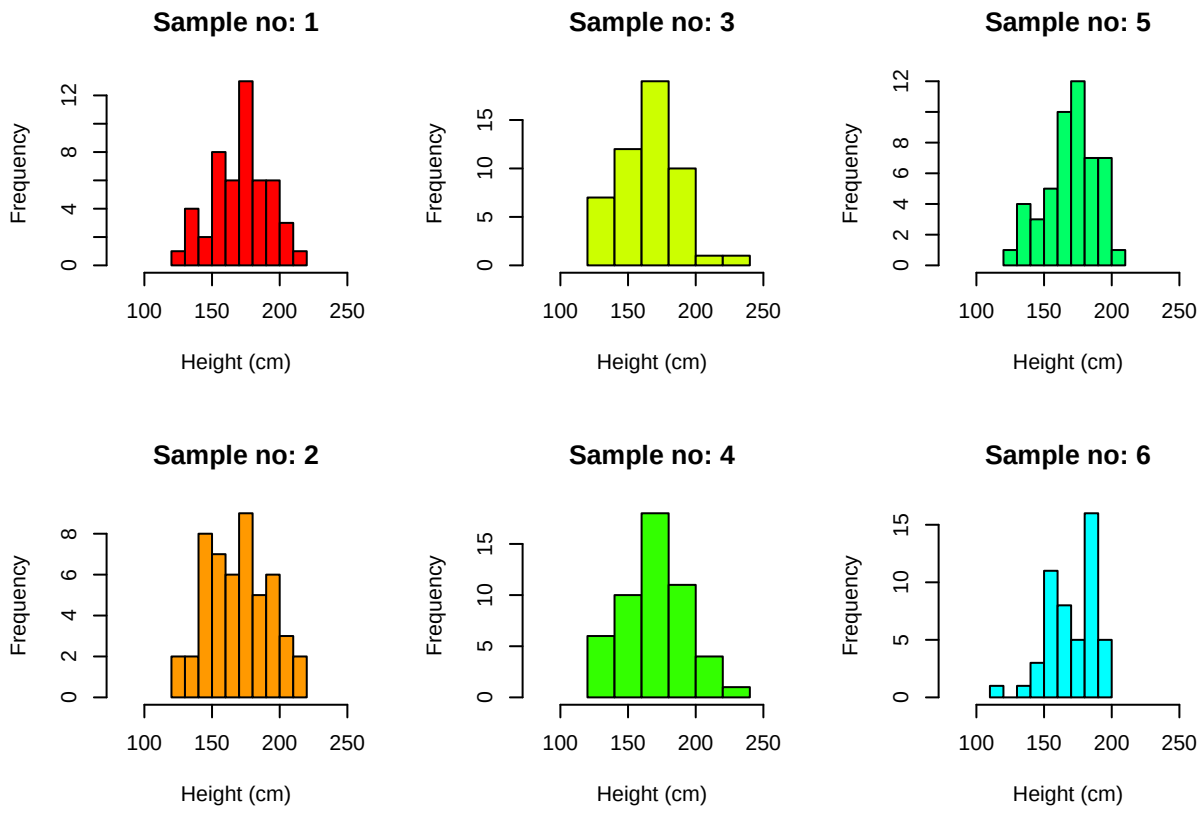
```
## [1] "a*b"
```

```r
paste("a", "b", sep=" is not ")
```

```
## [1] "a is not b"
```

Another improvement on the loop would be plotting the graphs with the same x range, so that are more readily compared. For this we would need to define the `xlim` argument of `hist` (also used in many other plotting functions), which is a 2-element vector with the minimum and maximum values of the range. We could define this vector before the loop. This is trivial in our small loop, but would save you time if you were running this 1m times (we would avoid calling the `min` and `max` functions multiple times):

```r
# define the x limit in advance, to be used inside the loop
XLIM = c(min(heights), max(heights))
par( mfcol = c(2,3))
sapply(1:6, function(i) {
  sample_heights = sample(heights, 50)
  hist( sample_heights, xlab="Height (cm)",
        col=rainbow(10)[i],             # now we use different colors
        main=paste("Sample no:", i),    # a unique title
        xlim=XLIM )
})
```



```
##          [,1]             [,2]             [,3]
## breaks   Numeric,11       Numeric,11       Numeric,7
## counts   Integer,10       Integer,10       Integer,6
## density  Numeric,10       Numeric,10       Numeric,6
## mids     Numeric,10       Numeric,10       Numeric,6
## xname    "sample_heights" "sample_heights" "sample_heights"
## equidist TRUE             TRUE             TRUE
##          [,4]             [,5]             [,6]
## breaks   Numeric,7        Numeric,10       Numeric,10
## counts   Integer,6        Integer,9        Integer,9
## density  Numeric,6        Numeric,9        Numeric,9
## mids     Numeric,6        Numeric,9        Numeric,9
## xname    "sample_heights" "sample_heights" "sample_heights"
## equidist TRUE             TRUE             TRUE
```

Now let's sample `50` elements `1000` times, store the result in a vector called `sample_heights_means`, and study the distribution of the means, compared to the expected mean, `170`:

```
# 1000 means, with n=50
set.seed(1)
sample_heights_means = sapply(1:1000, function(i) {
  mean( sample(heights, 50) )
  # even without saying return, the function will return the result
})
length(sample_heights_means)
```

```
## [1] 1000
```

```
mean(heights)
```

```
## [1] 169.9551
```

```
# an overview
summary(sample_heights_means)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   161.3   168.1   169.9   169.9   171.9   178.2
```

Plot the histogram of the population and sample means, showing the expected and the observed means on the plot using the function `abline`:

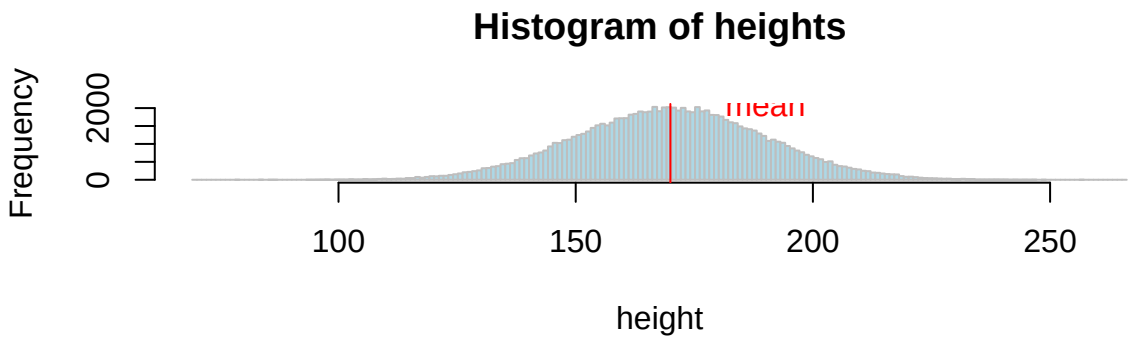
```
par( mfcol = c(2,1)) # to draw two plots per window
# to make the graph look nicer
minx = min(heights)-10
maxx = max(heights)+10
minx
```

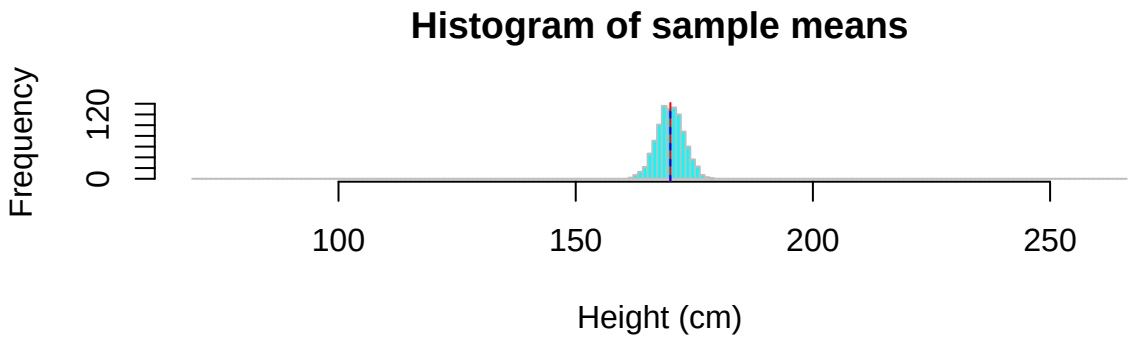
```
## [1] 69.15755
```

```
maxx
```

```
## [1] 266.2724
```

```
hist(heights, col="light blue",
        xlab="height", breaks = minx:maxx, border = "grey",
        xlim=c(minx, maxx) )
# this adds a vertical line to an existing plot, at the mean of the population
abline(v=mean(heights), col = "red")
# this adds text at specific x and y positions on an existing plot
text(x = 190, y = 2000, "mean", col="red")
hist(sample_heights_means, col="cyan", border  = "grey",
        xlab="Height (cm)", breaks = minx:maxx,
        xlim=c(minx, maxx),
        main = "Histogram of sample means" )
# add another line, at the mean of the population
abline(v=mean(heights), col = "red")
# add another line, at the mean of the sample means. lty defines line type, 3 means dashed
abline(v=mean(sample_heights_means), col = "blue", lty = 2)
```





The means are quite symmetrically distributed around the expected value, the population mean.

A note on loops

Note that in the above code, `i` is just a **counter** and does not have any role inside the loop. Therefore the following are equivalent:

```r
set.seed(1)
sample_heights_means = sapply(1:1000, function(i) { mean( sample(heights, 50) ) })
summary(sample_heights_means)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   161.3   168.1   169.9   169.9   171.9   178.2
```

```r
set.seed(1)  # we need to set the seed to the same value to ensure sample will choose the same set.
sample_heights_means = sapply(3050:4049, function(i) { mean( sample(heights, 50) ) })
summary(sample_heights_means)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   161.3   168.1   169.9   169.9   171.9   178.2
```

Creating plot files

To create pdf files of your plot output, you can use the following commands, `pdf` and `dev.off()`:

```
pdf("myplots.pdf")  # this opens an empty file with the specified name
hist(heights, col="light blue",
        xlab="height", breaks = minx:maxx, border = "grey",
        xlim=c(minx, maxx) )
hist(sample_heights_means, col="cyan", border  = "grey",
        xlab="Height (cm)", breaks = minx:maxx,
        xlim=c(minx, maxx),
        main = "Histogram of sample means" )
dev.off()  # this closes the file, only after which you can check the pdf
```

Variance of sample means

The theorem's other prediction is that the variance of sample means will be `1/n` of the population variance.

```
var(heights)
```

```
## [1] 402.8236
```

```
var(heights)/50
```

```
## [1] 8.056471
```

```
var(sample_heights_means)
```

```
## [1] 8.068148
```

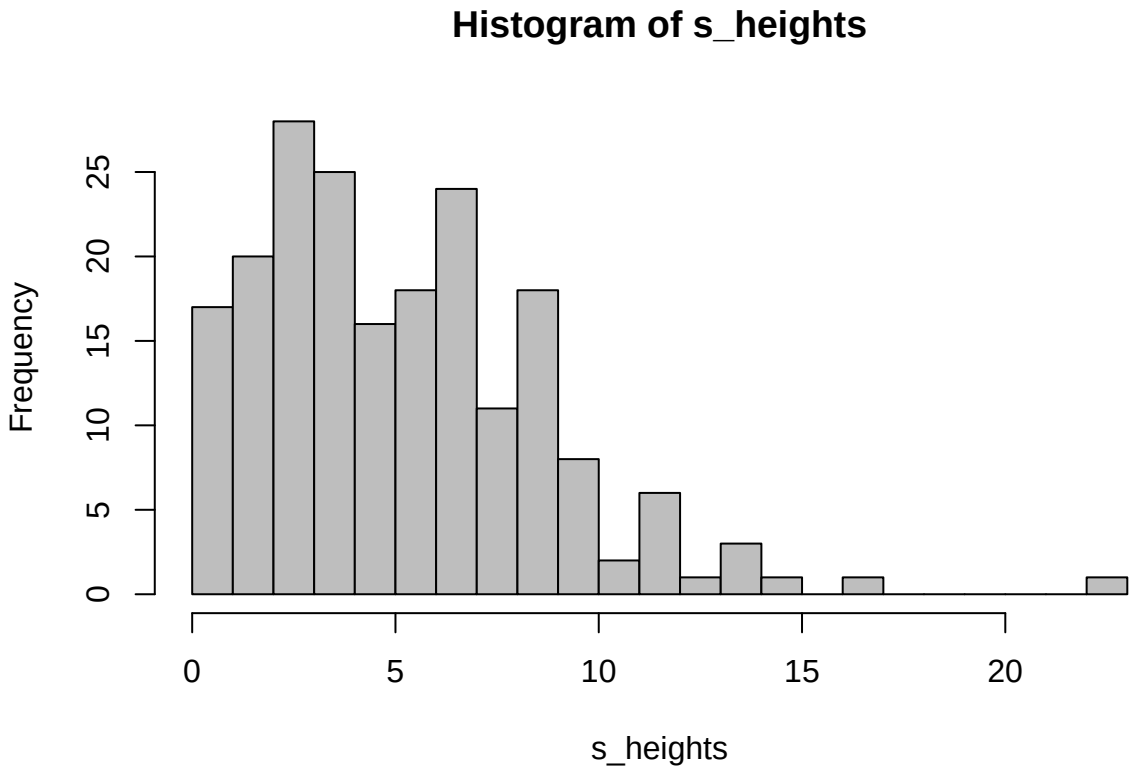
This suggests that, the larger the sample size, the closer the sample estimates will be to the population value.

Deviations from the expectation as a function of sample size

Let's now create a plot to observe how increasing sample size decreases deviation from the population mean. In other words, how means of larger samples are closer to the population mean.

For this, try sample sizes that range from `10` to `100` in steps of `10`. Each time take 200 random samples, calculate the mean, the absolute deviation from the population mean, and store these in a matrix. Let's start with `N=10`.

```
N = 10
s_heights = sapply(1:200, function(i) {
  abs( mean(sample(heights, N)) - mean(heights) )
  })
hist(s_heights, breaks=30, col="grey")
```



Now let's run the code for all the 10 sample sizes, using a `sapply` within a `sapply`:

```
sample_sizes = seq(10, 100, by=10)
s2_heights = sapply( sample_sizes, function(N) {
  sapply(1:200, function(i) {
    abs( mean(sample(heights, N)) - mean(heights) )
    })
})
head(s2_heights)
```

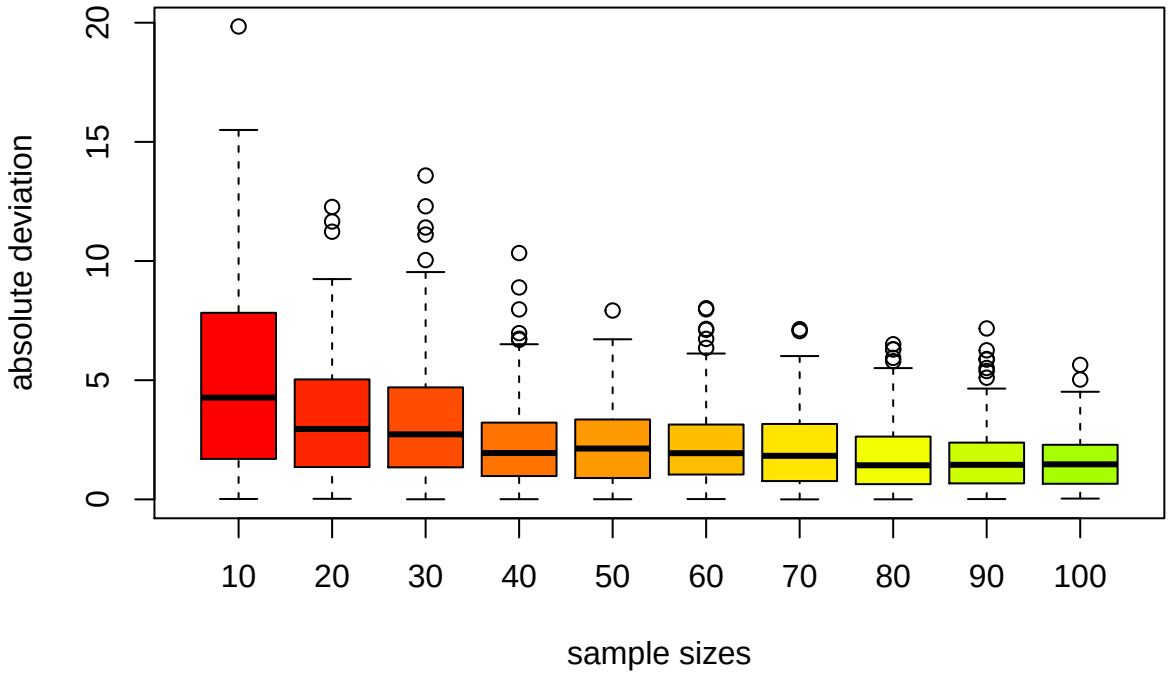
```
##            [,1]     [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 12.3196071 1.809170 8.4398167 4.4882020 3.0534937 0.2562209 0.3027082
## [2,]  0.3634716 3.637878 0.1067526 4.1345975 3.2289415 4.5217997 0.7632222
## [3,]  0.8707687 1.178541 6.0241515 1.8274736 3.7417069 2.4788556 5.0345814
## [4,]  4.9386663 2.628189 6.3449776 1.3251356 0.3474685 5.2372162 0.5993923
## [5,]  0.5628559 5.460996 2.5961281 1.4360076 1.8069962 2.7539054 1.2853214
## [6,]  0.4603929 4.677521 0.5443937 0.6945943 4.3460712 2.8508882 1.6634507
##           [,8]      [,9]     [,10]
## [1,] 0.1946474 0.6786757 4.2620640
## [2,] 2.8118147 0.7019519 4.1266898
## [3,] 2.0527629 3.2499426 1.0015931
## [4,] 1.2313925 0.7834241 0.1266977
## [5,] 2.3031657 0.1128216 0.5931169
## [6,] 0.1937582 0.8639154 1.5395941
```

```
dim(s2_heights)
```

```
## [1] 200  10
```

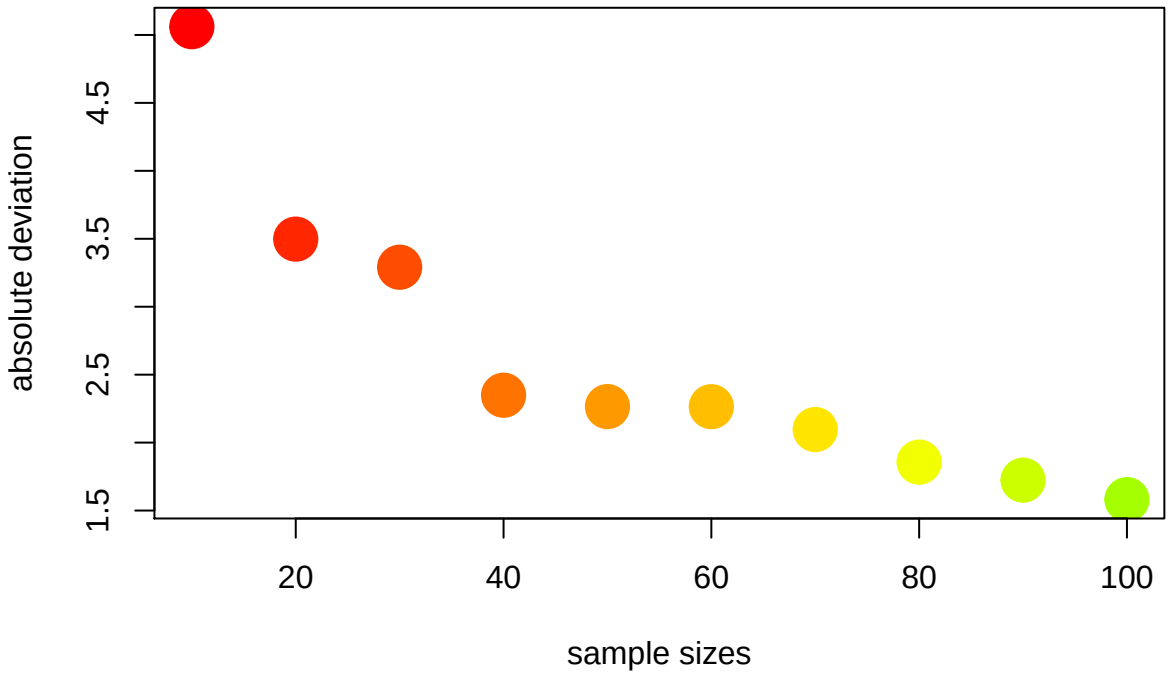
Now each column represents a sample size, and each row is a trial. We wish to study how deviations change with sample size. We could use a boxplot:

```
boxplot(s2_heights, names = sample_sizes, xlab="sample sizes", col = rainbow(40),
        ylab="absolute deviation")
```



Alternatively you can also plot the mean of the matrix columns against the sample sizes:

```
s2_heights_m = apply(s2_heights, 2, mean)
plot(sample_sizes, s2_heights_m, xlab="sample sizes", pch = 19, cex = 3, col = rainbow(40),
     ylab="absolute deviation")
```



The normal distribution of sample means: the dice throw example

The final statement we will study is that the distribution of sample means even from non-normal distribution will converge to the normal distribution with increasing sample size.

Let's test this using a simple scheme, the expectation of a dice roll. The dice roll distribution is not normal but uniform, each value has an expectation of `1/6`. Try to simulate a dice roll using the `sample` function, run it for `1000` times, and draw the histogram:

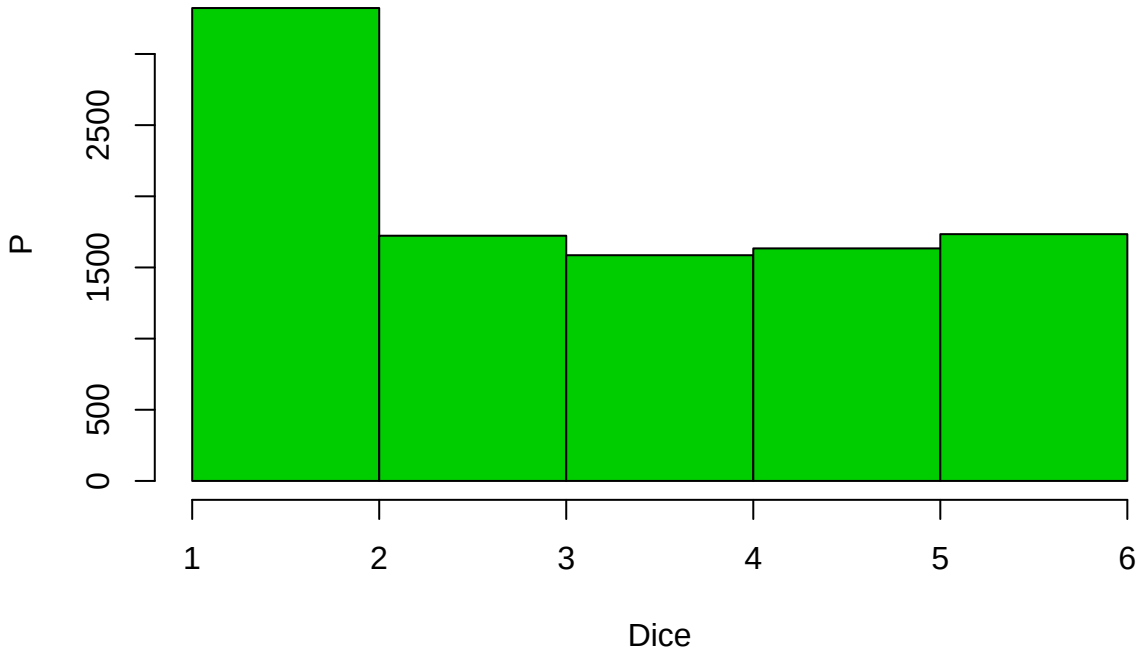
```
set.seed(1)
sample(1:6, size = 1)
```

```
## [1] 2
```

```
sample(1:6, size = 1)
```

```
## [1] 3
```

```
SIZE=10000
oneDice = sapply(1:SIZE, function(i) { sample(1:6, size = 1 ) } )
hist( oneDice, main="", xlab="Dice", breaks = 6, col = 3, ylab = "P")
```



The histogram here is not the best way of representation, as the data is not continuous but discrete. A better way would be calculating frequencies and using `barplot`:

```
barplot( table( oneDice )/length(oneDice), main="", xlab="Dice", col = 3, ylab="P")
```

If you roll two 6-sided dice you know that the most common sum would be `7`, i.e. a mean of 3.5. Try to simulate this using the `sample` function. Then run it for `1000` times, and draw the barplot:

```r
set.seed(1)
sample(1:6, size = 2, replace = TRUE)
```

```
## [1] 2 3
```

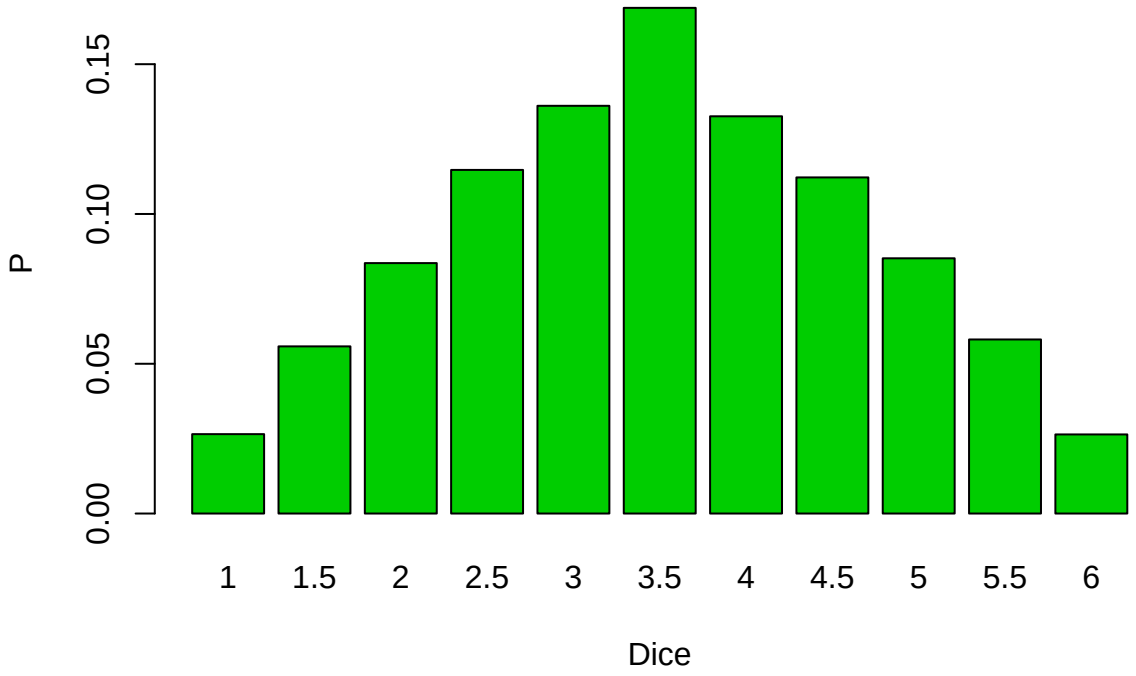
```r
sample(1:6, size = 2, replace = TRUE)
```

```
## [1] 4 6
```

```r
mean( sample(1:6, size = 2, replace = TRUE) )
```

```
## [1] 4
```

```r
twoDice = sapply(1:SIZE, function(i) { mean( sample(1:6, size = 2, replace = TRUE)) })
barplot( table( twoDice )/length(twoDice), main="", xlab="Dice", col = 3, ylab = "P")
```

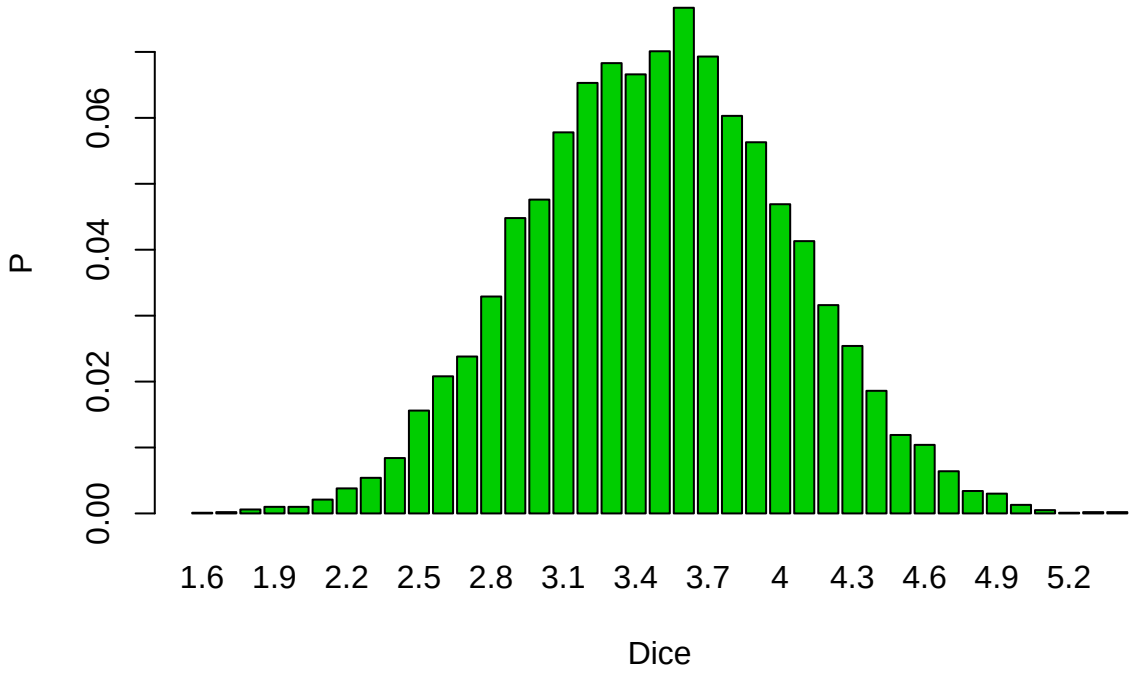


What if we increased the sample size to 10?

```r
sample(1:6, size = 10, replace = TRUE)
```

```
##  [1] 1 3 2 2 2 4 3 5 5 2
```

```r
tenDice = sapply(1:SIZE, function(i) { mean( sample(1:6, size = 10, replace = TRUE)) })
barplot( table( tenDice )/length(tenDice), main="", xlab="Dice", col = 3, ylab = "P")
```

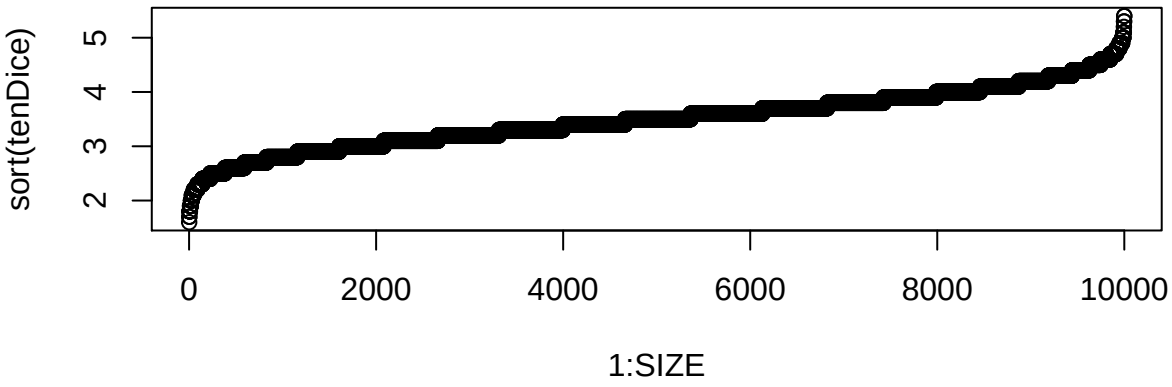


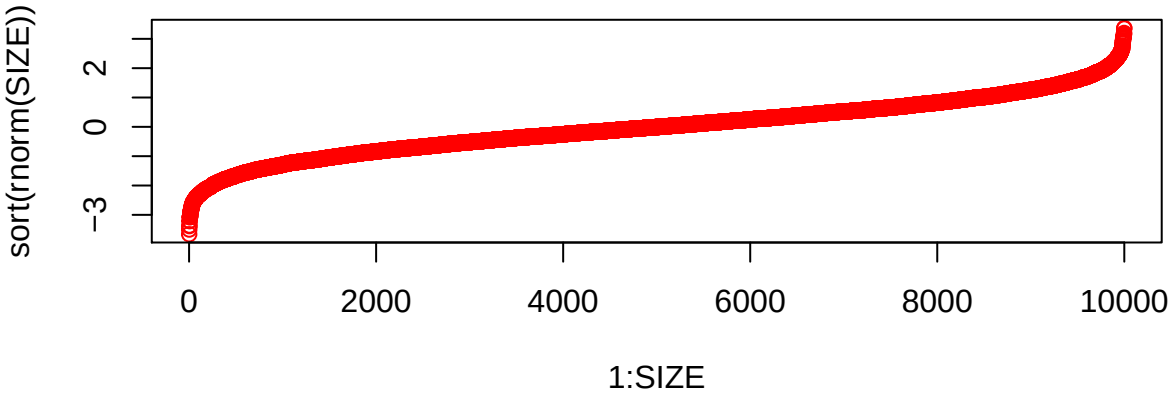
The resulting distribution has started resembling a normal distribution.

Comparison of distribution shapes: `scale`, `qqplot`

How can we compare the distribution's shape with the normal?

```
par(mfrow=c(2,1))
plot( 1:SIZE, sort(tenDice) )
# draw numbers from the normal distribution, and plot the same way
plot( 1:SIZE, sort(rnorm(SIZE) ), col="red")
```





They have similar shapes, although the mean and variances are different (the second one is the standard normal). One way to deal with this difference is to **scale** the `tenDice` data to mean=0 and s.d.=1, using the `scale` function. A small example:

```
x = 1:10
xs = scale(x)  # substracts the mean and divides by the sd
mean(xs)
```

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

```
sd(xs)
```

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

Now we apply this on the `tenDice` vector, and then plot the above, after scaling:

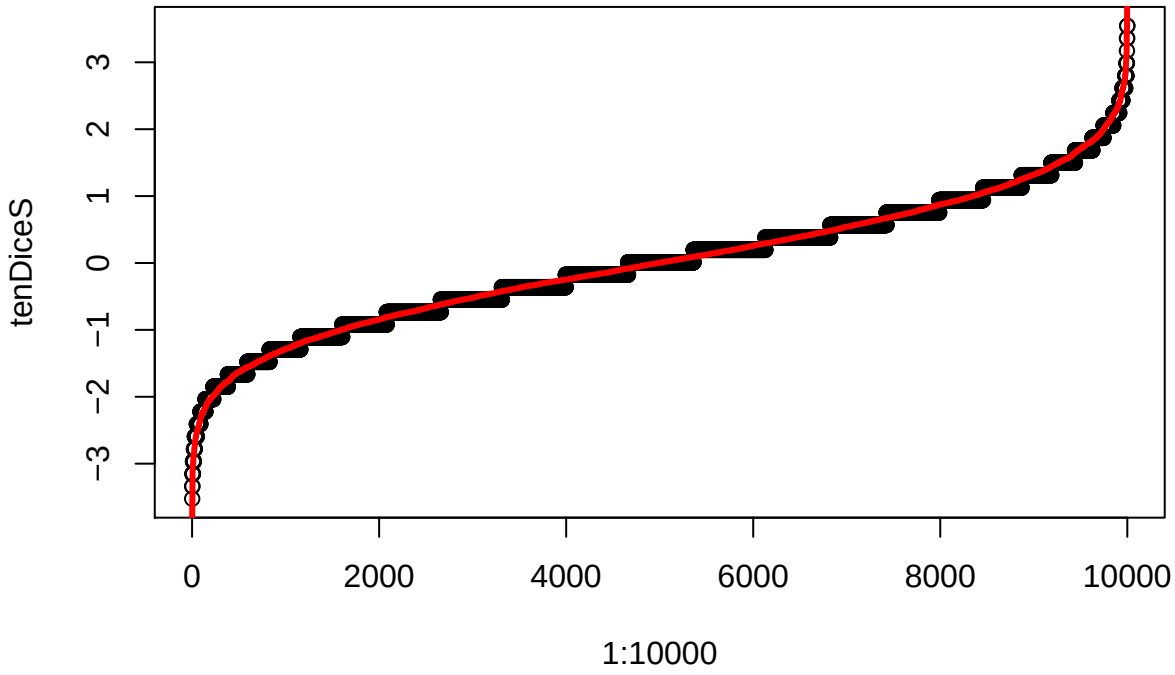
```
tenDiceS = scale(sort(tenDice))
# check it worked
mean(tenDiceS)
```

```
## [1] 3.091786e-16
```

```
sd(tenDiceS)
```

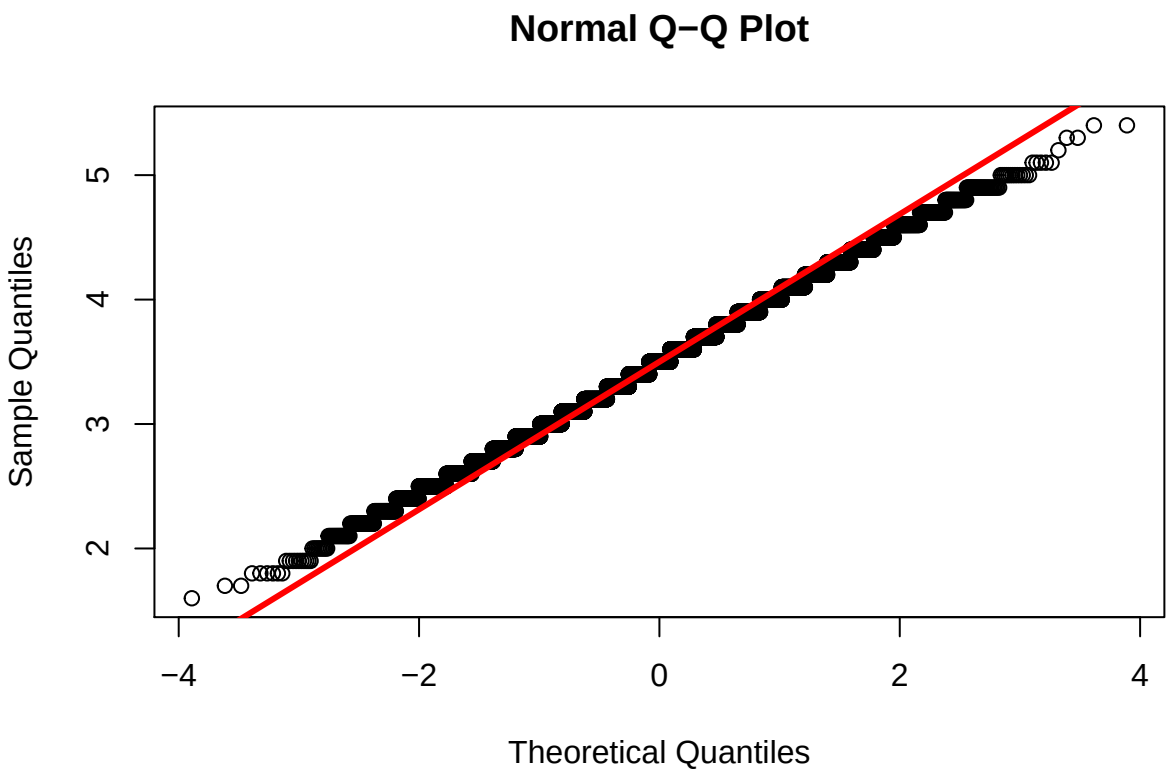
```
## [1] 1
```

```
# plot again
plot( 1:10000, tenDiceS )
# now we can add the normal data on top, as a line
lines( 1:10000, sort(rnorm(10000) ), col="red", lwd = 3)
```



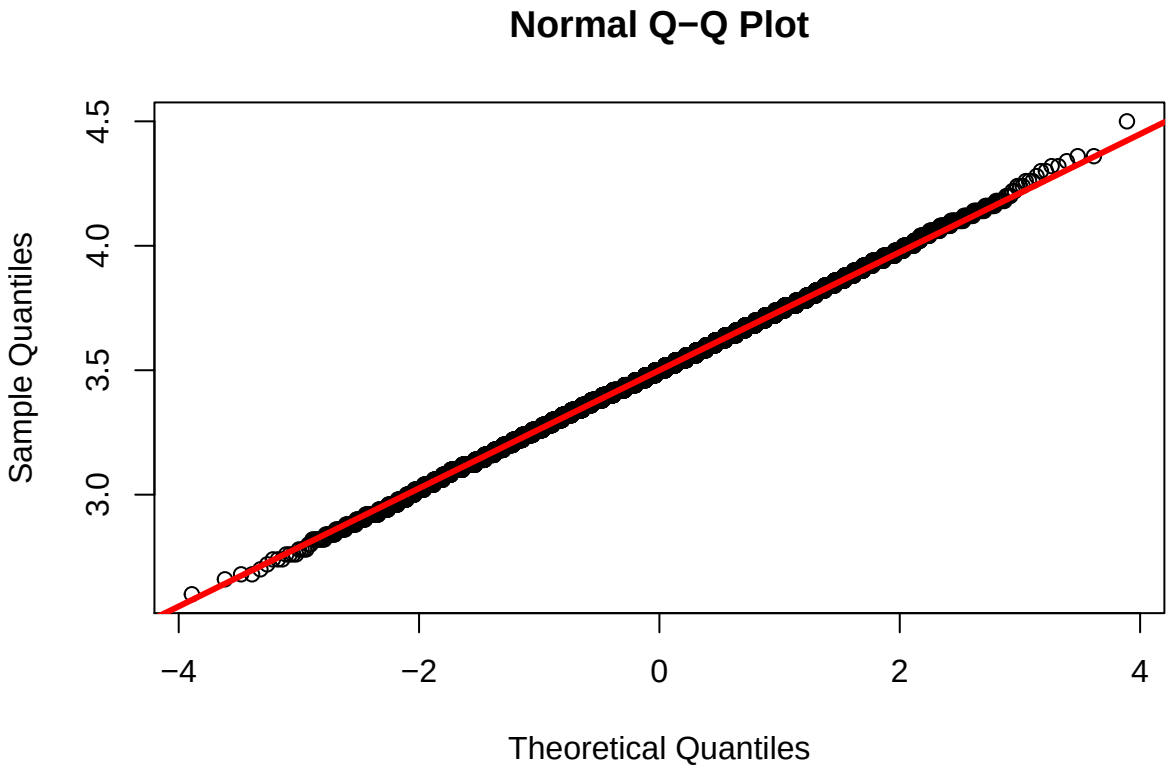
A more direct way to compare a distribution's shape with the normal distribution is the `qqnorm` function, which compares sample quantiles of a vector, to the corresponding theoretical quantiles from the normal distribution. `qqline` draws the 1:1 line. If the data lie on the line, the fit is perfect.

```
qqnorm( tenDice )
qqline( tenDice, col="red", lwd = 3)
```



Only at the tails do we see deviation. Will it disappear if we used a sample size of `50`?

```
fiftyDice = sapply(1:10000, function(i) { mean( sample(1:6, size = 50, replace = TRUE)) })
qqnorm( fiftyDice )
qqline( fiftyDice, col="red", lwd = 3)
```



So this observation supports the CLT - with larg enough sample size, **sample mean distributions** even from non-normal populations converge to the normal.

The Shapiro-Wilk Test of normality

The Shapiro-Wilk test is a formal test for the **null hypothesis** that a sample comes from a normal distribution. Thus, if the test is rejected (a low p-value), we say the sample comes from a non-normal distribution. However, the result depends not only on the amount of deviation, but on the sample size:

```
shapiro.test(tenDice[1:77])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  tenDice[1:77]
## W = 0.98809, p-value = 0.6949
```

This is not significant, because the deviations are small, and with small sample size of `n=77`, not reliable. However:

```
shapiro.test(tenDice[1:777])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  tenDice[1:777]
## W = 0.99517, p-value = 0.01503
```

This is now significant, because the sample size is larger, and the small deviations are more reliable. So we can reject the null hypothesis.

However, with `fiftyDice`, even with `n=777` the deviations are now so small that, they tend to be non-significant:

```
shapiro.test(fiftyDice[1:777])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  fiftyDice[1:777]
## W = 0.99771, p-value = 0.3661
```

If you really need to decide if your data follow a normal distribution, rather than *just* running the Shapiro-Wilk test, you should check how much deviation can be observed with the `qqplot`.

String manipulation

`paste` and `nchar`

Let's say we wish to name rows of a matrix in the form of "gene_1", "gene_2"... Here we can use the function `paste`, which concatenates string vectors (after converting them into character, if necessary). It adds the arguments in the given order by using the separator specified by the `sep` argument, the default value of which is space (' ').

```r
paste("x", "y", "z")
```

```
## [1] "x y z"
```

```r
paste("x", "y", "z", sep=".")
```

```
## [1] "x.y.z"
```

The function `nchar` counts the number of characters in a string:

```r
x = paste("x", "y", "z", sep=".")
length( x )
```

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

```r
nchar( x )
```

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

```r
nchar( paste("x", "y", "z", sep="") )
```

```
## [1] 3
```

```r
nchar( c("ali", "veli") )
```

```
## [1] 3 4
```

Also, it works more or less in the same way as `1:10 + 4` works: it iterates the through the elements, so you don't need to write a for loop if you want a smaller element to be concatenated with a longer one.

```r
paste(1:5, "x", "z", sep="*")
```

```
## [1] "1*x*z" "2*x*z" "3*x*z" "4*x*z" "5*x*z"
```

```r
paste(1:5, "x", 10:11, sep="*")  # again, the shorter vector is recycled
```

```
## [1] "1*x*10" "2*x*11" "3*x*10" "4*x*11" "5*x*10"
```

So recreate the matrix `me` with 60 rows and 5 columns, and assign the rows names "gene_1, gene_2 etc."

```r
me = matrix(1:300, 60, 5)
rownames(me) = paste("gene", 1:nrow(me), sep="_")
head(me)
```

```
##        [,1] [,2] [,3] [,4] [,5]
## gene_1    1   61  121  181  241
## gene_2    2   62  122  182  242
## gene_3    3   63  123  183  243
## gene_4    4   64  124  184  244
## gene_5    5   65  125  185  245
## gene_6    6   66  126  186  246
```

Create a character vector that combines together the states of the 3 categories in the `esoph` dataset, for each combination, and check if each combination is reported only once:

```
head(esoph)
```

```
##   agegp     alcgp    tobgp ncases ncontrols
## 1 25-34 0-39g/day 0-9g/day      0        40
## 2 25-34 0-39g/day    10-19      0        10
## 3 25-34 0-39g/day    20-29      0         6
## 4 25-34 0-39g/day      30+      0         5
## 5 25-34     40-79 0-9g/day      0        27
## 6 25-34     40-79    10-19      0         7
```

```
esoph_stat = paste("age:", esoph$agegp, "alc:", esoph$alcgp, "tob:", esoph$tobgp, sep=" ")
esoph_stat
```

```
##  [1] "age: 25-34 alc: 0-39g/day tob: 0-9g/day"
##  [2] "age: 25-34 alc: 0-39g/day tob: 10-19"
##  [3] "age: 25-34 alc: 0-39g/day tob: 20-29"
##  [4] "age: 25-34 alc: 0-39g/day tob: 30+"
##  [5] "age: 25-34 alc: 40-79 tob: 0-9g/day"
##  [6] "age: 25-34 alc: 40-79 tob: 10-19"
##  [7] "age: 25-34 alc: 40-79 tob: 20-29"
##  [8] "age: 25-34 alc: 40-79 tob: 30+"
##  [9] "age: 25-34 alc: 80-119 tob: 0-9g/day"
## [10] "age: 25-34 alc: 80-119 tob: 10-19"
## [11] "age: 25-34 alc: 80-119 tob: 30+"
## [12] "age: 25-34 alc: 120+ tob: 0-9g/day"
## [13] "age: 25-34 alc: 120+ tob: 10-19"
## [14] "age: 25-34 alc: 120+ tob: 20-29"
## [15] "age: 25-34 alc: 120+ tob: 30+"
## [16] "age: 35-44 alc: 0-39g/day tob: 0-9g/day"
## [17] "age: 35-44 alc: 0-39g/day tob: 10-19"
## [18] "age: 35-44 alc: 0-39g/day tob: 20-29"
## [19] "age: 35-44 alc: 0-39g/day tob: 30+"
## [20] "age: 35-44 alc: 40-79 tob: 0-9g/day"
## [21] "age: 35-44 alc: 40-79 tob: 10-19"
## [22] "age: 35-44 alc: 40-79 tob: 20-29"
## [23] "age: 35-44 alc: 40-79 tob: 30+"
## [24] "age: 35-44 alc: 80-119 tob: 0-9g/day"
## [25] "age: 35-44 alc: 80-119 tob: 10-19"
## [26] "age: 35-44 alc: 80-119 tob: 20-29"
## [27] "age: 35-44 alc: 80-119 tob: 30+"
## [28] "age: 35-44 alc: 120+ tob: 0-9g/day"
```

```
## [29] "age: 35-44 alc: 120+ tob: 10-19"
## [30] "age: 35-44 alc: 120+ tob: 20-29"
## [31] "age: 45-54 alc: 0-39g/day tob: 0-9g/day"
## [32] "age: 45-54 alc: 0-39g/day tob: 10-19"
## [33] "age: 45-54 alc: 0-39g/day tob: 20-29"
## [34] "age: 45-54 alc: 0-39g/day tob: 30+"
## [35] "age: 45-54 alc: 40-79 tob: 0-9g/day"
## [36] "age: 45-54 alc: 40-79 tob: 10-19"
## [37] "age: 45-54 alc: 40-79 tob: 20-29"
## [38] "age: 45-54 alc: 40-79 tob: 30+"
## [39] "age: 45-54 alc: 80-119 tob: 0-9g/day"
## [40] "age: 45-54 alc: 80-119 tob: 10-19"
## [41] "age: 45-54 alc: 80-119 tob: 20-29"
## [42] "age: 45-54 alc: 80-119 tob: 30+"
## [43] "age: 45-54 alc: 120+ tob: 0-9g/day"
## [44] "age: 45-54 alc: 120+ tob: 10-19"
## [45] "age: 45-54 alc: 120+ tob: 20-29"
## [46] "age: 45-54 alc: 120+ tob: 30+"
## [47] "age: 55-64 alc: 0-39g/day tob: 0-9g/day"
## [48] "age: 55-64 alc: 0-39g/day tob: 10-19"
## [49] "age: 55-64 alc: 0-39g/day tob: 20-29"
## [50] "age: 55-64 alc: 0-39g/day tob: 30+"
## [51] "age: 55-64 alc: 40-79 tob: 0-9g/day"
## [52] "age: 55-64 alc: 40-79 tob: 10-19"
## [53] "age: 55-64 alc: 40-79 tob: 20-29"
## [54] "age: 55-64 alc: 40-79 tob: 30+"
## [55] "age: 55-64 alc: 80-119 tob: 0-9g/day"
## [56] "age: 55-64 alc: 80-119 tob: 10-19"
## [57] "age: 55-64 alc: 80-119 tob: 20-29"
## [58] "age: 55-64 alc: 80-119 tob: 30+"
## [59] "age: 55-64 alc: 120+ tob: 0-9g/day"
## [60] "age: 55-64 alc: 120+ tob: 10-19"
## [61] "age: 55-64 alc: 120+ tob: 20-29"
## [62] "age: 55-64 alc: 120+ tob: 30+"
## [63] "age: 65-74 alc: 0-39g/day tob: 0-9g/day"
## [64] "age: 65-74 alc: 0-39g/day tob: 10-19"
## [65] "age: 65-74 alc: 0-39g/day tob: 20-29"
## [66] "age: 65-74 alc: 0-39g/day tob: 30+"
## [67] "age: 65-74 alc: 40-79 tob: 0-9g/day"
## [68] "age: 65-74 alc: 40-79 tob: 10-19"
## [69] "age: 65-74 alc: 40-79 tob: 20-29"
## [70] "age: 65-74 alc: 80-119 tob: 0-9g/day"
## [71] "age: 65-74 alc: 80-119 tob: 10-19"
## [72] "age: 65-74 alc: 80-119 tob: 20-29"
## [73] "age: 65-74 alc: 80-119 tob: 30+"
## [74] "age: 65-74 alc: 120+ tob: 0-9g/day"
## [75] "age: 65-74 alc: 120+ tob: 10-19"
## [76] "age: 65-74 alc: 120+ tob: 20-29"
## [77] "age: 65-74 alc: 120+ tob: 30+"
## [78] "age: 75+ alc: 0-39g/day tob: 0-9g/day"
## [79] "age: 75+ alc: 0-39g/day tob: 10-19"
## [80] "age: 75+ alc: 0-39g/day tob: 30+"
## [81] "age: 75+ alc: 40-79 tob: 0-9g/day"
## [82] "age: 75+ alc: 40-79 tob: 10-19"
```

```
## [83] "age: 75+ alc: 40-79 tob: 20-29"
## [84] "age: 75+ alc: 40-79 tob: 30+"
## [85] "age: 75+ alc: 80-119 tob: 0-9g/day"
## [86] "age: 75+ alc: 80-119 tob: 10-19"
## [87] "age: 75+ alc: 120+ tob: 0-9g/day"
## [88] "age: 75+ alc: 120+ tob: 10-19"
```

```
length(esoph_stat) == length(unique(esoph_stat))
```

```
## [1] TRUE
```

strsplit

`strsplit` does just the opposite of `paste`. It splits the elements of a character vector into substrings according to the `split` argument.

```
strsplit("gene_1", split="_")
```

```
## [[1]]
## [1] "gene" "1"
```

The `split` argument assumes the can be used using "metacharacters", which are symbols with special meaning in character pattern matching: `. \ | ( ) [ { ^ $ * + ?`.

E.g. `*` means "any character". Please check `?regex` for rules on pattern matching.

```
strsplit("gene1*x", split="*")
```

```
## [[1]]
## [1] "g" "e" "n" "e" "1" "*" "x"
```

Therefore if you want to split a character string by one of these, e.g. "*", you should change the command and use the additional argument `fixed=T`.

```
strsplit("gene1*x", split="*", fixed = T)
```

```
## [[1]]
## [1] "gene1" "x"
```

```
# this also works
strsplit("gene1*x", split="[*]")
```

```
## [[1]]
## [1] "gene1" "x"
```

Note that the `strsplit` output is a list, by default:

```
head( strsplit(rownames(me), "_") )
```

```
## [[1]]
## [1] "gene" "1"
##
## [[2]]
## [1] "gene" "2"
##
## [[3]]
## [1] "gene" "3"
##
## [[4]]
## [1] "gene" "4"
##
## [[5]]
## [1] "gene" "5"
##
## [[6]]
## [1] "gene" "6"
```

```
tail( strsplit(rownames(me), "_") )
```

```
## [[1]]
## [1] "gene" "55"
##
## [[2]]
## [1] "gene" "56"
##
## [[3]]
## [1] "gene" "57"
##
## [[4]]
## [1] "gene" "58"
##
## [[5]]
## [1] "gene" "59"
##
## [[6]]
## [1] "gene" "60"
```

Let's run a function and retrieve the numbers back:

```
rownam = strsplit(rownames(me), "_")  # a list
sapply( rownam, function(x) x[2])# sapply also works on lists. each step of the loop, x will be a vecto
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28"
## [29] "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42"
## [43] "43" "44" "45" "46" "47" "48" "49" "50" "51" "52" "53" "54" "55" "56"
## [57] "57" "58" "59" "60"
```

Retrieve back columns 1-3 of `esoph` from `esoph_stat`:

```
head( esoph_stat )
```

```
## [1] "age: 25-34 alc: 0-39g/day tob: 0-9g/day"
## [2] "age: 25-34 alc: 0-39g/day tob: 10-19"
## [3] "age: 25-34 alc: 0-39g/day tob: 20-29"
## [4] "age: 25-34 alc: 0-39g/day tob: 30+"
## [5] "age: 25-34 alc: 40-79 tob: 0-9g/day"
## [6] "age: 25-34 alc: 40-79 tob: 10-19"
```

```
# split and store the list
esoph_statx = strsplit(esoph_stat, split=" ")
head( esoph_statx )
```

```
## [[1]]
## [1] "age:"      "25-34"     "alc:"      "0-39g/day" "tob:"      "0-9g/day"
##
## [[2]]
## [1] "age:"      "25-34"     "alc:"      "0-39g/day" "tob:"      "10-19"
##
## [[3]]
## [1] "age:"      "25-34"     "alc:"      "0-39g/day" "tob:"      "20-29"
##
## [[4]]
## [1] "age:"      "25-34"     "alc:"      "0-39g/day" "tob:"      "30+"
##
## [[5]]
## [1] "age:"     "25-34"    "alc:"     "40-79"    "tob:"     "0-9g/day"
##
## [[6]]
## [1] "age:"  "25-34" "alc:"  "40-79" "tob:"  "10-19"
```

```
# collect the 2nd, 4th and 6th elements
sapply( esoph_statx, function(x) {
  x[c(2,4,6)]
})
```

```
##      [,1]        [,2]        [,3]        [,4]        [,5]       [,6]
## [1,] "25-34"     "25-34"     "25-34"     "25-34"     "25-34"    "25-34"
## [2,] "0-39g/day" "0-39g/day" "0-39g/day" "0-39g/day" "40-79"    "40-79"
## [3,] "0-9g/day"  "10-19"     "20-29"     "30+"       "0-9g/day" "10-19"
##      [,7]    [,8]    [,9]       [,10]    [,11]    [,12]      [,13]
## [1,] "25-34" "25-34" "25-34"    "25-34"  "25-34"  "25-34"    "25-34"
## [2,] "40-79" "40-79" "80-119"   "80-119" "80-119" "120+"     "120+"
## [3,] "20-29" "30+"   "0-9g/day" "10-19"  "30+"    "0-9g/day" "10-19"
##      [,14]   [,15]   [,16]       [,17]       [,18]       [,19]
## [1,] "25-34" "25-34" "35-44"     "35-44"     "35-44"     "35-44"
## [2,] "120+"  "120+"  "0-39g/day" "0-39g/day" "0-39g/day" "0-39g/day"
## [3,] "20-29" "30+"   "0-9g/day"  "10-19"     "20-29"     "30+"
##      [,20]      [,21]   [,22]   [,23]   [,24]      [,25]    [,26]
## [1,] "35-44"    "35-44" "35-44" "35-44" "35-44"    "35-44"  "35-44"
```

```
## [2,] "40-79"    "40-79" "40-79" "40-79" "80-119"   "80-119" "80-119"
## [3,] "0-9g/day" "10-19" "20-29" "30+"   "0-9g/day" "10-19"  "20-29"
##      [,27]    [,28]       [,29]   [,30]   [,31]       [,32]
## [1,] "35-44"  "35-44"    "35-44" "35-44" "45-54"     "45-54"
## [2,] "80-119" "120+"     "120+"  "120+"  "0-39g/day" "0-39g/day"
## [3,] "30+"    "0-9g/day" "10-19" "20-29" "0-9g/day"  "10-19"
##      [,33]       [,34]       [,35]      [,36]   [,37]   [,38]   [,39]
## [1,] "45-54"     "45-54"     "45-54"    "45-54" "45-54" "45-54" "45-54"
## [2,] "0-39g/day" "0-39g/day" "40-79"    "40-79" "40-79" "40-79" "80-119"
## [3,] "20-29"     "30+"       "0-9g/day" "10-19" "20-29" "30+"   "0-9g/day"
##      [,40]    [,41]    [,42]    [,43]      [,44]   [,45]   [,46]
## [1,] "45-54"  "45-54"  "45-54"  "45-54"    "45-54" "45-54" "45-54"
## [2,] "80-119" "80-119" "80-119" "120+"     "120+"  "120+"  "120+"
## [3,] "10-19"  "20-29"  "30+"    "0-9g/day" "10-19" "20-29" "30+"
##      [,47]       [,48]       [,49]       [,50]       [,51]      [,52]
## [1,] "55-64"     "55-64"     "55-64"     "55-64"     "55-64"    "55-64"
## [2,] "0-39g/day" "0-39g/day" "0-39g/day" "0-39g/day" "40-79"    "40-79"
## [3,] "0-9g/day"  "10-19"     "20-29"     "30+"       "0-9g/day" "10-19"
##      [,53]   [,54]   [,55]      [,56]    [,57]    [,58]    [,59]
## [1,] "55-64" "55-64" "55-64"    "55-64"  "55-64"  "55-64"  "55-64"
## [2,] "40-79" "40-79" "80-119"   "80-119" "80-119" "80-119" "120+"
## [3,] "20-29" "30+"   "0-9g/day" "10-19"  "20-29"  "30+"    "0-9g/day"
##      [,60]   [,61]   [,62]   [,63]       [,64]       [,65]
## [1,] "55-64" "55-64" "55-64" "65-74"     "65-74"     "65-74"
## [2,] "120+"  "120+"  "120+"  "0-39g/day" "0-39g/day" "0-39g/day"
## [3,] "10-19" "20-29" "30+"   "0-9g/day"  "10-19"     "20-29"
##      [,66]       [,67]      [,68]   [,69]   [,70]      [,71]    [,72]
## [1,] "65-74"     "65-74"    "65-74" "65-74" "65-74"    "65-74"  "65-74"
## [2,] "0-39g/day" "40-79"    "40-79" "40-79" "80-119"   "80-119" "80-119"
## [3,] "30+"       "0-9g/day" "10-19" "20-29" "0-9g/day" "10-19"  "20-29"
##      [,73]    [,74]       [,75]   [,76]   [,77]   [,78]       [,79]
## [1,] "65-74"  "65-74"    "65-74" "65-74" "65-74" "75+"       "75+"
## [2,] "80-119" "120+"     "120+"  "120+"  "120+"  "0-39g/day" "0-39g/day"
## [3,] "30+"    "0-9g/day" "10-19" "20-29" "30+"   "0-9g/day"  "10-19"
##      [,80]       [,81]       [,82]   [,83]   [,84]   [,85]      [,86]
## [1,] "75+"       "75+"       "75+"   "75+"   "75+"   "75+"      "75+"
## [2,] "0-39g/day" "40-79"    "40-79" "40-79" "40-79" "80-119"   "80-119"
## [3,] "30+"       "0-9g/day" "10-19" "20-29" "30+"   "0-9g/day" "10-19"
##      [,87]      [,88]
## [1,] "75+"      "75+"
## [2,] "120+"     "120+"
## [3,] "0-9g/day" "10-19"
```

Note that `sapply` returns a matrix whenever the loop's single steps return a vector of the same size.

We can **transpose** the matrix using `t`:

```
esoph_statx_123 = t( sapply( esoph_statx, function(x) {
  x[c(2,4,6)]
}) )
dim(esoph_statx_123)
```

```
## [1] 88  3
```

```
head(esoph_statx_123)
```

```
##      [,1]    [,2]        [,3]
## [1,] "25-34" "0-39g/day" "0-9g/day"
## [2,] "25-34" "0-39g/day" "10-19"
## [3,] "25-34" "0-39g/day" "20-29"
## [4,] "25-34" "0-39g/day" "30+"
## [5,] "25-34" "40-79"     "0-9g/day"
## [6,] "25-34" "40-79"     "10-19"
```

gsub

How about identifying and changing patterns inside a string? The `gsub` function does that:

```
gsub("t", "g", "the")
```

```
## [1] "ghe"
```

Let's use this to retrieve the numbers embedded in the `me` rownames:

```
gsub("gene_", "", rownames(me))
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28"
## [29] "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42"
## [43] "43" "44" "45" "46" "47" "48" "49" "50" "51" "52" "53" "54" "55" "56"
## [57] "57" "58" "59" "60"
```

```
as.numeric( gsub("gene_", "", rownames(me)) )
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
## [47] 47 48 49 50 51 52 53 54 55 56 57 58 59 60
```

Now let's translate the rownames into Turkish phrase (into "insangeni"):

```
gsub("gene", "insangeni", rownames(me))
```

```
##  [1] "insangeni_1"  "insangeni_2"  "insangeni_3"  "insangeni_4"
##  [5] "insangeni_5"  "insangeni_6"  "insangeni_7"  "insangeni_8"
##  [9] "insangeni_9"  "insangeni_10" "insangeni_11" "insangeni_12"
## [13] "insangeni_13" "insangeni_14" "insangeni_15" "insangeni_16"
## [17] "insangeni_17" "insangeni_18" "insangeni_19" "insangeni_20"
## [21] "insangeni_21" "insangeni_22" "insangeni_23" "insangeni_24"
## [25] "insangeni_25" "insangeni_26" "insangeni_27" "insangeni_28"
## [29] "insangeni_29" "insangeni_30" "insangeni_31" "insangeni_32"
## [33] "insangeni_33" "insangeni_34" "insangeni_35" "insangeni_36"
```

```
## [37] "insangeni_37" "insangeni_38" "insangeni_39" "insangeni_40"
## [41] "insangeni_41" "insangeni_42" "insangeni_43" "insangeni_44"
## [45] "insangeni_45" "insangeni_46" "insangeni_47" "insangeni_48"
## [49] "insangeni_49" "insangeni_50" "insangeni_51" "insangeni_52"
## [53] "insangeni_53" "insangeni_54" "insangeni_55" "insangeni_56"
## [57] "insangeni_57" "insangeni_58" "insangeni_59" "insangeni_60"
```

substr

Another useful function to retrieve only the gene numbers could be `substr`

```
substr("taylan", 1, 3)
```

```
## [1] "tay"
```

```
substr("taylan", 4, 4)
```

```
## [1] "l"
```

```
paste(substr("taylan", 4, 6), "taylan")
```

```
## [1] "lan taylan"
```

Let's try this function to retrieve the number in the rownames:

```
substr(rownames(me), 11, 12)
```

```
##  [1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [24] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
## [47] "" "" "" "" "" "" "" "" "" "" "" "" "" ""
```

grep

`grep` searches for a given pattern (its first argument) in each element of a character vector (the second argument).

```
x = c("taylan", "dilek", "duygu", "mustafa")
x
```

```
## [1] "taylan"  "dilek"   "duygu"   "mustafa"
```

```
# this check the presence of "a" (equality of any element to "a") within a vector
x == "a"
```

```
## [1] FALSE FALSE FALSE FALSE
```

```r
# this check the presence of "a" within each string of a vector
grep("a", x)
```

```
## [1] 1 4
```

The default output is indices of elements containing the pattern. But using the argument `value=T` you can make the function return the matching elements themselves:

```r
grep("a", x, value = T)
```

```
## [1] "taylan"  "mustafa"
```

```r
grep("la", x, value = T)
```

```
## [1] "taylan"
```

```r
grep("L", x, value = T)
```

```
## character(0)
```

```r
grep("L", x, value = T, ignore.case = T)
```

```
## [1] "taylan" "dilek"
```

Regular expressions can also be used to search for patterns in more specific manner:

```r
# only "t" at the start
grep("^t", x, value = T)
```

```
## [1] "taylan"
```

```r
# only "a"" at the end
grep("a$", x, value = T)
```

```
## [1] "mustafa"
```

```r
# number of elements containing "a"
length( unique( grep("a", x, value = T) ) )
```

```
## [1] 2
```

For more on string manipulation and regular expressions:

https://www.rstudio.com/wp-content/uploads/2016/09/RegExCheatsheet.pdf

http://stat545.com/block022_regular-expression.html

The category combinations containing `75+` in `esoph_stat`

```
grep("75+", esoph_stat, val=T)
```

```
##  [1] "age: 75+ alc: 0-39g/day tob: 0-9g/day"
##  [2] "age: 75+ alc: 0-39g/day tob: 10-19"
##  [3] "age: 75+ alc: 0-39g/day tob: 30+"
##  [4] "age: 75+ alc: 40-79 tob: 0-9g/day"
##  [5] "age: 75+ alc: 40-79 tob: 10-19"
##  [6] "age: 75+ alc: 40-79 tob: 20-29"
##  [7] "age: 75+ alc: 40-79 tob: 30+"
##  [8] "age: 75+ alc: 80-119 tob: 0-9g/day"
##  [9] "age: 75+ alc: 80-119 tob: 10-19"
## [10] "age: 75+ alc: 120+ tob: 0-9g/day"
## [11] "age: 75+ alc: 120+ tob: 10-19"
```

Just to remind ourselves: if we had to check for equality to a character element, same approach would work.

```
c("a","b","c") == "b"
```

```
## [1] FALSE  TRUE FALSE
```

The primate liver dataset

Now, we'll start working with real data! Data generated in many fields of biology is made publicly available through databases. Here we will use an expression dataset from NCBI's GEO (Gene Expression Omnibus) Database. GEO contains thousands of transcriptome and similar genome-wide molecular profiling experiments: http://www.ncbi.nlm.nih.gov/geo/

The dataset we're interested in has the ID GSE17274. To download the dataset, please go to the GEO database and search for GSE17274. Here you can learn about the experiment in summary. By following the link to the PMID: 20009012, you can also find the related article "Sex-specific and lineage-specific alternative splicing in primate". This is a liver RNA-sequencing dataset from humans, chimpanzees and macaques. If you are not familiar with NGS data, check: lyle.smu.edu/~mhd/8330f11/NextGenerationSequencing.pptx

The data from this experiment can be retrieved at the end of the page, under the title "supplementary files". Here we have links to both the raw sequencing data and the processed data file - where the authors aligned the sequencing "reads" to the species' genomes, and calculated the number of reads mapping to each gene. The file "GSE17274_ReadCountPerLane.txt" contains these read counts per "lane" (one sequencing experiment). This is the file we'll be working on. Please download it.

Reading text files using `read.table`

Now we need to learn how to import a dataset. How to read this dataset into R? You can use the RStudio interface. Alternatively, if you were working directly from the console, you could check our working directory using `getwd`, change it with `setwd`, and double check the presence of the file in that directory using `list.files`.

```
getwd()
```

```
## [1] "/Users/msomel/Documents/misc/metu/ders/2380754_comp_2017"
```

```
list.files()
```

```
##  [1] "BIO754 Student list.docx"      "BIOL 754 Section 1 Grades.ods"
##  [3] "Empirical_Rule.png"            "GSE17274_ReadCountPerLane.txt"
##  [5] "attendance BIOL 754.ods"       "homework_week_01.Rmd"
##  [7] "homework_week_01.html"         "homework_week_02.Rmd"
##  [9] "homework_week_02.html"         "homework_week_02_solution.Rmd"
## [11] "homework_week_02_solution.pdf" "homework_week_03.Rmd"
## [13] "homework_week_03.html"         "lecture_week_01.Rmd"
## [15] "lecture_week_01.pdf"           "lecture_week_02.Rmd"
## [17] "lecture_week_02.pdf"           "lecture_week_03.Rmd"
## [19] "lecture_week_03.pdf"           "lecture_week_04.Rmd"
## [21] "lecture_week_04.pdf"           "lecture_week_04_files"
## [23] "lecture_week_05.Rmd"           "lecture_week_05.pdf"
## [25] "lecturenotes"                  "stat test.docx"
## [27] "stat test.pdf"                 "syllabus_2380754_2017.docx"
## [29] "syllabus_2380754_2017.pdf"
```

```
mat = read.table("GSE17274_ReadCountPerLane.txt")
# tab separated, the default separator
head(mat)
```

```
##                V1        V2        V3        V4        V5        V6
## 1   EnsemblGeneID R1L1.HSM1 R1L2.PTF1 R1L3.RMM1 R1L4.HSF1 R1L6.PTM1
## 2 ENSG00000000003        60       285       207       172       176
## 3 ENSG00000000005         0         1         1         0         0
## 4 ENSG00000000419        17        54        20        36        23
## 5 ENSG00000000457        50        61        68        41       143
## 6 ENSG00000000460         9         6         2         3         3
##          V7        V8        V9       V10       V11       V12       V13
## 1 R1L7.RMF1 R2L2.RMF2 R2L3.HSM2 R2L4.PTF2 R2L6.RMM2 R2L7.HSF2 R2L8.PTM2
## 2       259       299       219       213       676       147       316
## 3         0         1         1         1         0         0         0
## 4        38        40        42        35        39        26        26
## 5        42        47        30       111        55        28       110
## 6         2         1         2         5         2         8         3
##         V14       V15       V16       V17       V18       V19       V20
## 1 R3L1.RMM3 R3L2.HSF2 R3L3.PTM1 R3L4.RMF3 R3L6.HSM3 R3L7.PTF3 R3L8.RMM1
## 2       338       153       233       242       199       180       217
## 3         0         0         0         0         0         0         1
## 4        36        35        36        22        33        35        29
## 5        76        34       131        61        53        46        49
## 6         2         9         9         5        11         2         5
##         V21       V22       V23       V24       V25       V26       V27
## 1 R4L1.HSM3 R4L2.HSF1 R4L3.RMM3 R4L4.PTF1 R4L6.PTM2 R4L7.RMF3 R4L8.HSM2
## 2       160       157       367       289       369       238       202
## 3         0         0         0         0         1         2         0
## 4        29        45        54        46        32        35        36
## 5        42        50        87        70       129        57        43
## 6         6         3         2         2         4         5         5
##         V28       V29       V30       V31       V32       V33       V34
## 1 R5L1.RMF1 R5L2.HSM1 R5L3.PTF3 R5L4.RMM2 R5L8.RMF2 R6L2.PTM3 R6L4.PTM3
## 2       252        61       206       672       165       216       212
```

```
## 3         0         0         1         0         0         1         0
## 4        29        22        30        43        32        40        53
## 5        47        64        41        59        22        71        78
## 6         2         6         3         1         3         5         5
##         V35       V36       V37
## 1 R6L6.PTF2 R8L1.HSF3 R8L2.HSF3
## 2       216        78        90
## 3         3         0         0
## 4        31        16        40
## 5       118        34        42
## 6         3         7         5
```

```
mat = read.table("GSE17274_ReadCountPerLane.txt",
                 row.names=1, header=T)
head(mat)
```

```
##                 R1L1.HSM1 R1L2.PTF1 R1L3.RMM1 R1L4.HSF1 R1L6.PTM1
## ENSG00000000003        60       285       207       172       176
## ENSG00000000005         0         1         1         0         0
## ENSG00000000419        17        54        20        36        23
## ENSG00000000457        50        61        68        41       143
## ENSG00000000460         9         6         2         3         3
## ENSG00000000938        32        50        44        23        99
##                 R1L7.RMF1 R2L2.RMF2 R2L3.HSM2 R2L4.PTF2 R2L6.RMM2
## ENSG00000000003       259       299       219       213       676
## ENSG00000000005         0         1         1         1         0
## ENSG00000000419        38        40        42        35        39
## ENSG00000000457        42        47        30       111        55
## ENSG00000000460         2         1         2         5         2
## ENSG00000000938        46        18        26        53        20
##                 R2L7.HSF2 R2L8.PTM2 R3L1.RMM3 R3L2.HSF2 R3L3.PTM1
## ENSG00000000003       147       316       338       153       233
## ENSG00000000005         0         0         0         0         0
## ENSG00000000419        26        26        36        35        36
## ENSG00000000457        28       110        76        34       131
## ENSG00000000460         8         3         2         9         9
## ENSG00000000938        30        16        29        35        99
##                 R3L4.RMF3 R3L6.HSM3 R3L7.PTF3 R3L8.RMM1 R4L1.HSM3
## ENSG00000000003       242       199       180       217       160
## ENSG00000000005         0         0         0         1         0
## ENSG00000000419        22        33        35        29        29
## ENSG00000000457        61        53        46        49        42
## ENSG00000000460         5        11         2         5         6
## ENSG00000000938        28        32        39        36        33
##                 R4L2.HSF1 R4L3.RMM3 R4L4.PTF1 R4L6.PTM2 R4L7.RMF3
## ENSG00000000003       157       367       289       369       238
## ENSG00000000005         0         0         0         1         2
## ENSG00000000419        45        54        46        32        35
## ENSG00000000457        50        87        70       129        57
## ENSG00000000460         3         2         2         4         5
## ENSG00000000938        21        43        34        15        26
##                 R4L8.HSM2 R5L1.RMF1 R5L2.HSM1 R5L3.PTF3 R5L4.RMM2
## ENSG00000000003       202       252        61       206       672
## ENSG00000000005         0         0         0         1         0
```

```
## ENSG00000000419        36        29        22        30        43
## ENSG00000000457        43        47        64        41        59
## ENSG00000000460         5         2         6         3         1
## ENSG00000000938        17        27        41        37        14
##                 R5L8.RMF2 R6L2.PTM3 R6L4.PTM3 R6L6.PTF2 R8L1.HSF3
## ENSG00000000003       165       216       212       216        78
## ENSG00000000005         0         1         0         3         0
## ENSG00000000419        32        40        53        31        16
## ENSG00000000457        22        71        78       118        34
## ENSG00000000460         3         5         5         3         7
## ENSG00000000938        10        30        28        51       112
##                 R8L2.HSF3
## ENSG00000000003        90
## ENSG00000000005         0
## ENSG00000000419        40
## ENSG00000000457        42
## ENSG00000000460         5
## ENSG00000000938        98
```

```
dim(mat)  #check that the dimensions fit
```

```
## [1] 20689    36
```

```
head(rownames(mat))
```

```
## [1] "ENSG00000000003" "ENSG00000000005" "ENSG00000000419" "ENSG00000000457"
## [5] "ENSG00000000460" "ENSG00000000938"
```

```
colnames(mat)
```

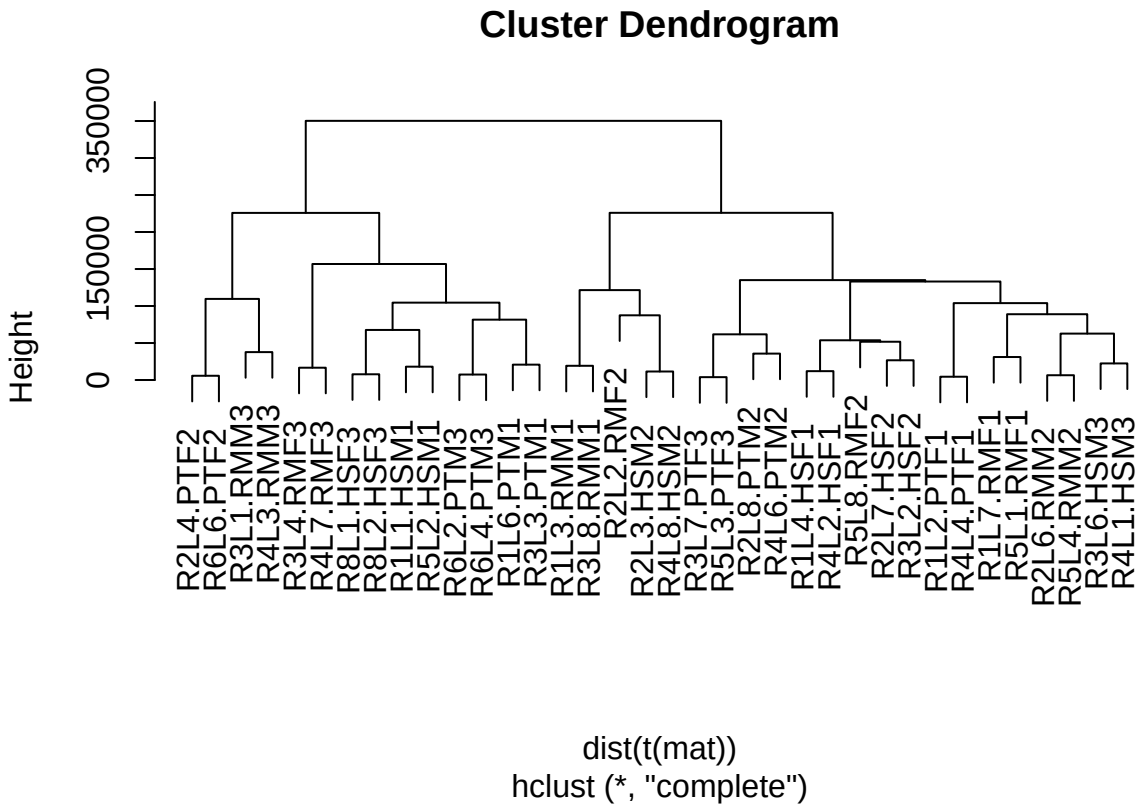
```
##  [1] "R1L1.HSM1" "R1L2.PTF1" "R1L3.RMM1" "R1L4.HSF1" "R1L6.PTM1"
##  [6] "R1L7.RMF1" "R2L2.RMF2" "R2L3.HSM2" "R2L4.PTF2" "R2L6.RMM2"
## [11] "R2L7.HSF2" "R2L8.PTM2" "R3L1.RMM3" "R3L2.HSF2" "R3L3.PTM1"
## [16] "R3L4.RMF3" "R3L6.HSM3" "R3L7.PTF3" "R3L8.RMM1" "R4L1.HSM3"
## [21] "R4L2.HSF1" "R4L3.RMM3" "R4L4.PTF1" "R4L6.PTM2" "R4L7.RMF3"
## [26] "R4L8.HSM2" "R5L1.RMF1" "R5L2.HSM1" "R5L3.PTF3" "R5L4.RMM2"
## [31] "R5L8.RMF2" "R6L2.PTM3" "R6L4.PTM3" "R6L6.PTF2" "R8L1.HSF3"
## [36] "R8L2.HSF3"
```

As you notice, the column names contain biological info on the species and the sex in the second part of the string. The first part of the string is about the "run" and "lane" numbers of the sequencing experiment (technical info).

For a start, we can make a phylogeny of gene expression differences among samples. We know that, at the DNA level, a phylogeny would cluster humans and chimps together, and macaques would be the outgroup. How about liver gene expression on average? Will it reflect the phylogeny as well? Or will it reflect e.g. diet of individuals, so that humans are the outlier with their high energy and protein diets?

To address this we can briefly draw a tree. Here we transpose the dataset using `t`, calculate gene expression distances (in this case, the euclidean distance between two vectors) among all columns to create a 36X36 **distance matrix**, using the function `dist`. Then we use `hclust` to summarise the matrix into a tree. Finally, we plot the tree.

```
par(mfrow=c(1,1))
plot(hclust(dist(t(mat))))
```



Admittedly, this looks weird, with no grouping in the tree with respect to any biological factor, either species or sex. We can only see grouping of technical replicates that belong to the same biological sample. So what is going on? Was all the experiment wasted? We will be learning soon.