BIO 754 - Lecture 07

06-04-2017

Contents

We are continuing with the liver transcriptome dataset analysis. We had discussed the benefits of using ANOVA vs. multiple pairwise 2-sample t-tests: avoiding inflating the Type I error rate. But is ANOVA always the best choice?

ANOVA vs. Kruskal-Wallis

ANOVA is a parametric test and it assumes equal variance among groups, and normality. Both assumptions might be violated, and this can be a problem especially when the sample size is not large (`<30`). Let us study the consequences using a simulation.

We will simulate two random data sets each time (for simplicity) from a normal distribution with the same mean - therefore the null hypothesis (H0: "group means are equal") is correct. Then we will perform ANOVA and study the p-values.

ANOVA p-values when H0 is valid

How do we expect the **p-values** to be distributed, when the null hypothesis is valid? Remember: the p-value, by definition, is the probability that the H0 is valid when we find a deviation from the H0 as large or larger than that observed.

Let's check one example:

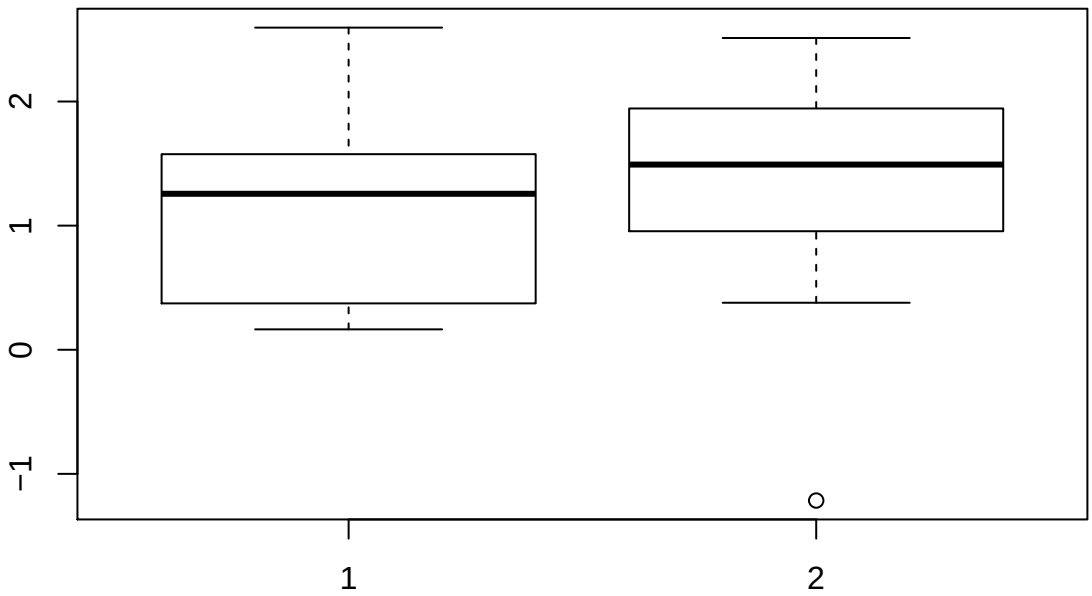
```
# simulation 1: testing mean difference when the data derives from a normal distribution
# first define the groups
groups = factor( rep(c("a", "b"), each=10) )
# create two datasets
set.seed(1)
x1 = rnorm(10, mean = 1, sd = 1)
x2 = rnorm(10, mean = 1, sd = 1)
mean(x1)
```

```
## [1] 1.132203
```

```
mean(x2)
```

```
## [1] 1.248845
```

```
boxplot(x1, x2)
```




```
# what is the probability that such a difference happens by chance?
summary( aov(c(x1, x2) ~ groups ) )
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## groups       1  0.068  0.0680   0.078  0.784
## Residuals   18 15.779  0.8766
```

```r
# we can retrieve the p-value as follows:
pval = summary( aov(c(x1, x2) ~ groups ) )[[1]][1, "Pr(>F)"]
pval
```

```
## [1] 0.7837457
```

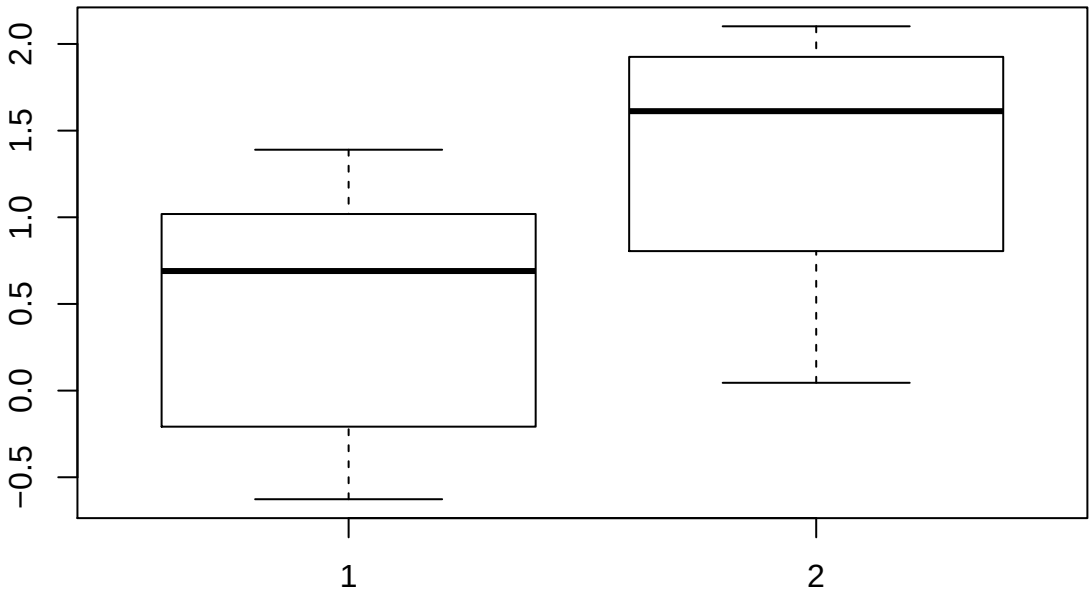
```r
# try again
set.seed(10)
x1 = rnorm(10, mean = 1, sd = 1)
x2 = rnorm(10, mean = 1, sd = 1)
mean(x1)
```

```
## [1] 0.5093432
```

```r
mean(x2)
```

```
## [1] 1.369592
```

```r
boxplot(x1, x2)
```



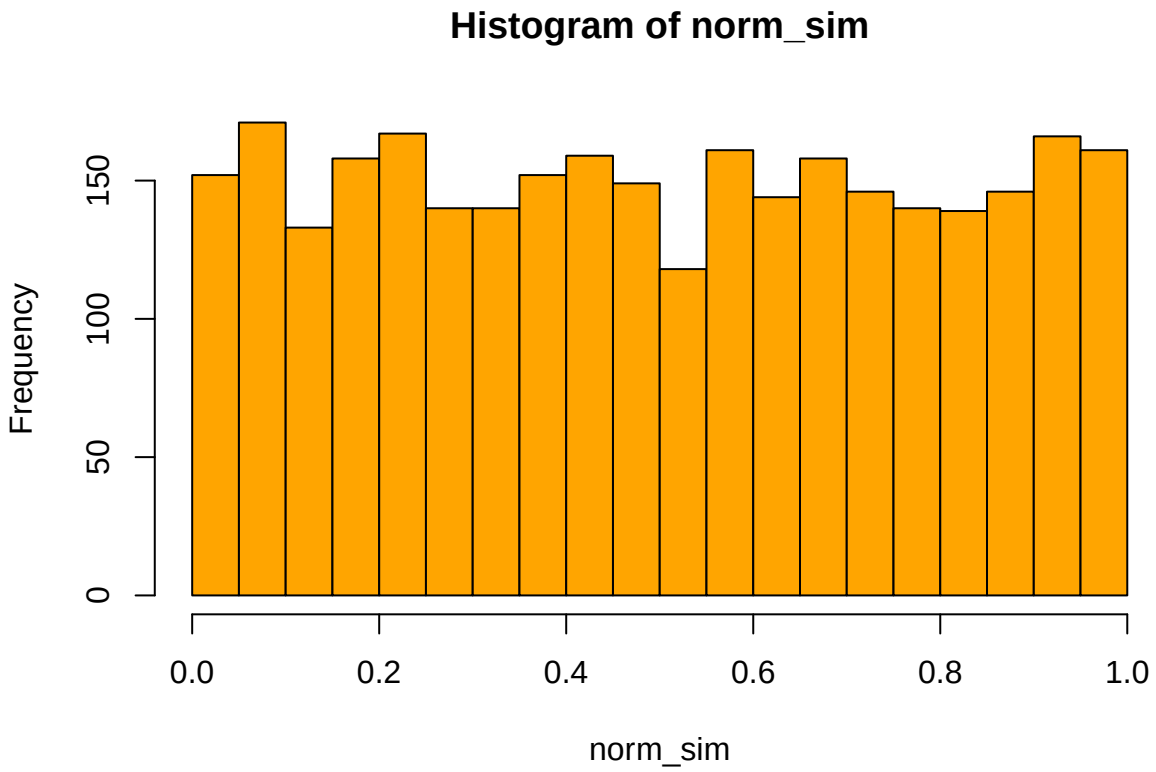

```r
# what is the probability that such a difference happens by chance?
pval = summary( aov(c(x1, x2) ~ groups ) )[[1]][1, "Pr(>F)"]
pval
```

```
## [1] 0.01167795
```

By chance, in our second trial, we obtained 2 datasets with large differences, and therefore a small ANOVA p-value `~0.01`. However, we know that the H0 is correct. Large differences (i.e. small p-values) can occur by chance, but they should happen rarely: If our test is **unbiased**, cases where `p<0.01` should occur ~1% of the time. Likewise, the probability of finding a difference that corresponds to `p<0.05` should occur ~5% of the time.

To check this expectation, let us run the ANOVA `3000` times, calculate the p-value, and and plot the histogram of the p-values:

```r
norm_sim = sapply(1:3000, function(i) {
  x1 = rnorm(10, mean = 1, sd = 1)
  x2 = rnorm(10, mean = 1, sd = 1)
  pval = summary( aov(c(x1, x2) ~ groups ) )[[1]][1, "Pr(>F)"]
})
hist(norm_sim, col="orange", br=20)
```



The distribution is **uniform**, as expected. We can also check the proportions of values less than some threshold:

```r
mean(norm_sim < 0.01)
```

```
## [1] 0.01
```

```r
mean(norm_sim < 0.05)
```

```
## [1] 0.05066667
```

```r
mean(norm_sim < 0.20)
```

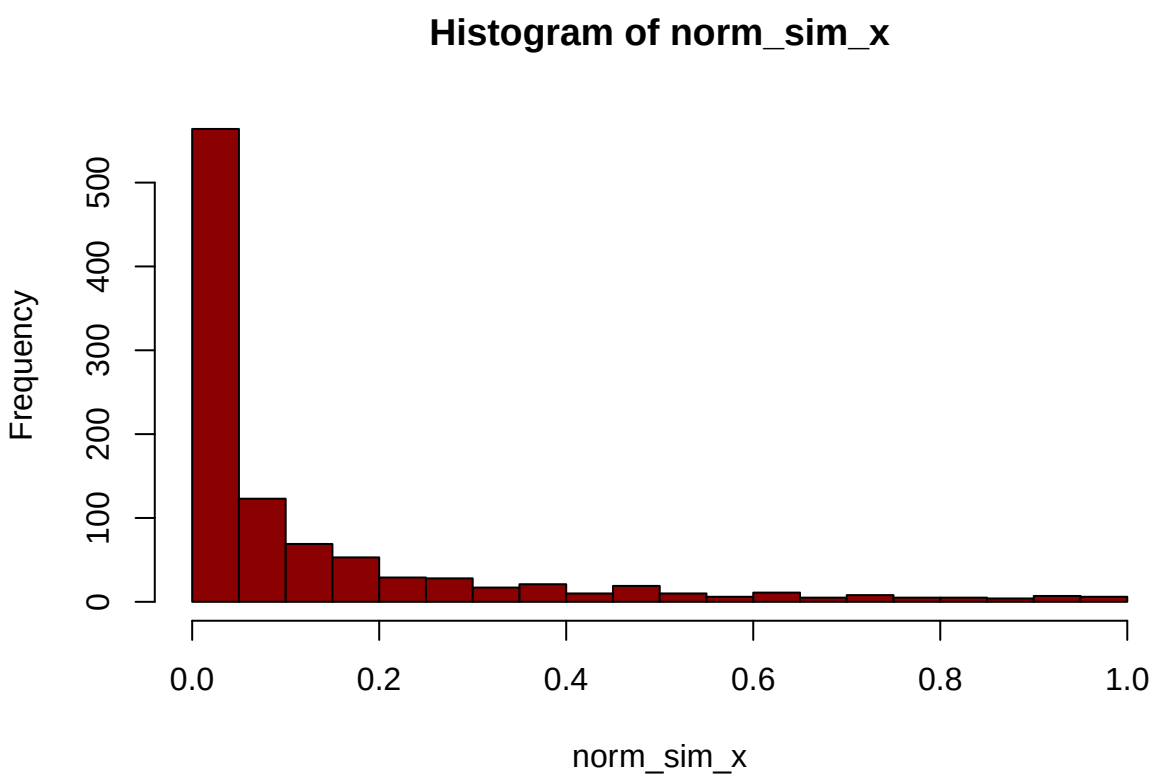
```
## [1] 0.2046667
```

This is also as expected. We have 5% false positives at `p<0.05`.

ANOVA p-values when H0 is not valid

Note that *if* the H0 was *not* true, we would also *not* expect the same result. Let's simulate data where the H0 is not true, from distributions with different means:

```
norm_sim_x = sapply(1:1000, function(i) {
  x1 = rnorm(10, mean = 1, sd = 1)
  x2 = rnorm(10, mean = 2, sd = 1)
  pval = summary(aov(c(x1, x2) ~ groups))[[1]][1,5]
  return(pval)
})
hist(norm_sim_x, col="red4", br=20)
```




```
mean(norm_sim_x < 0.05)
```

```
## [1] 0.564
```

Of course, not all cases are "significant" even though the H0 is not valid: these are our **false negatives**. The higher the power of the test (larger sample size, less variance, larger difference between groups), the less will be the false negatives.

What would happen if the data were chosen from a non-normal continuous distribution?

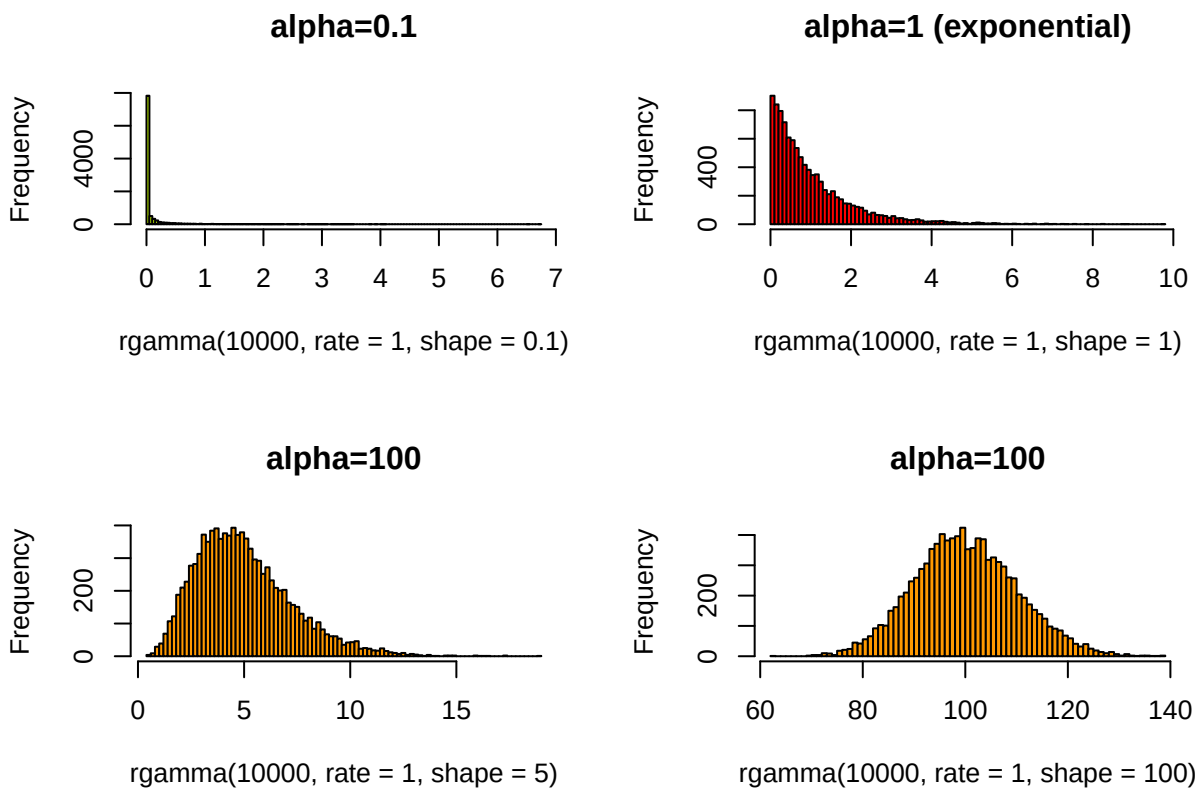
The gamma distribution

This is a family of continuous probability distributions frequently used to model waiting times (similar to the geometric), mutation effects, etc. It has two parameters, *shape* (or alpha) and *scale* (rate, or beta). A gamma distribution with shape parameter alpha = 1 and is an **exponential distribution**:

https://en.wikipedia.org/wiki/Gamma_distribution

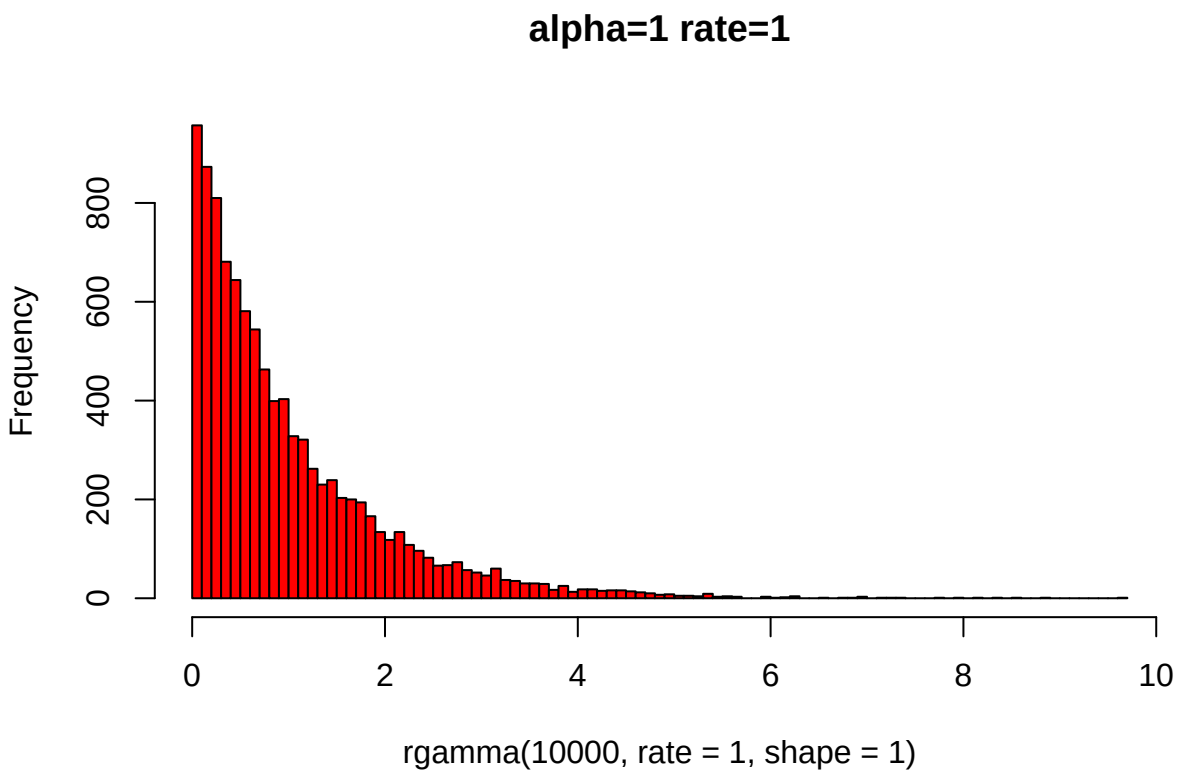
Check how the shape parameter alters the distribution:

```
par(mfrow=c(2,2))
hist(rgamma(10000, rate = 1, shape = 0.1), col=rainbow(10)[3], br=100, main="alpha=0.1")
hist(rgamma(10000, rate = 1, shape = 1), col=rainbow(10)[1], br=100, main="alpha=1 (exponential)")
hist(rgamma(10000, rate = 1, shape = 5), col=rainbow(10)[2], br=100, main="alpha=100")
hist(rgamma(10000, rate = 1, shape = 100), col=rainbow(10)[2], br=100, main="alpha=100")
```

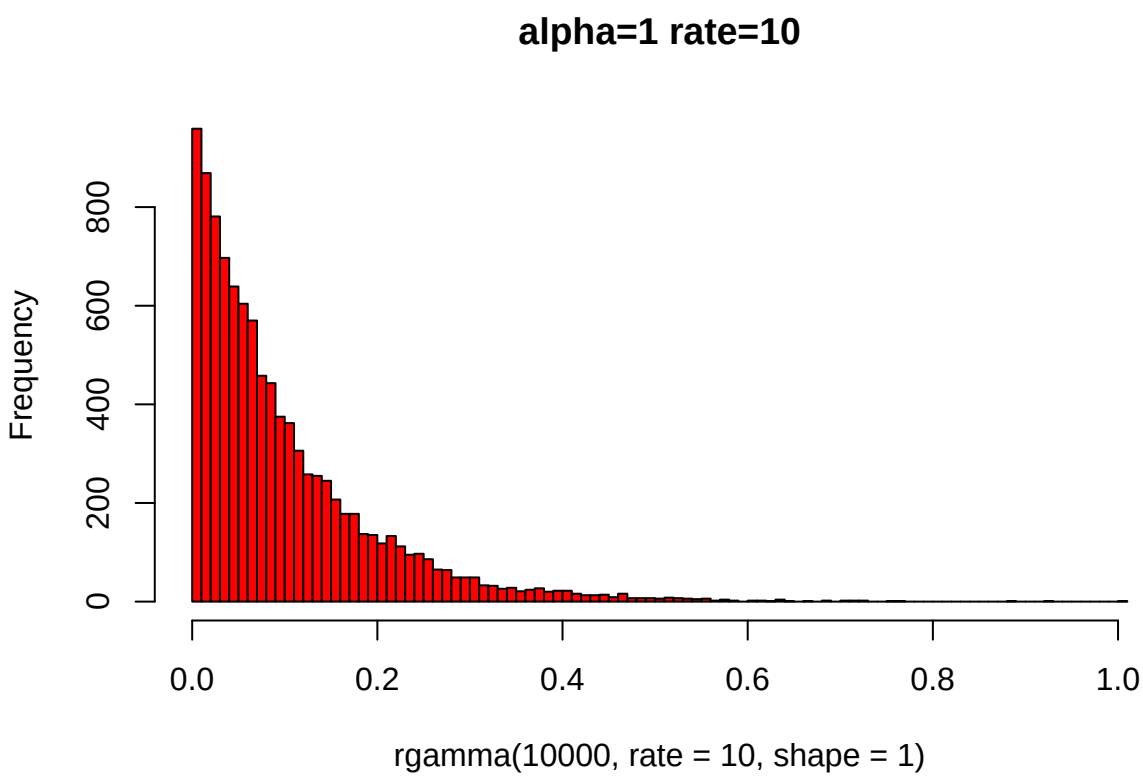


And the rate parameter alters the distribution:

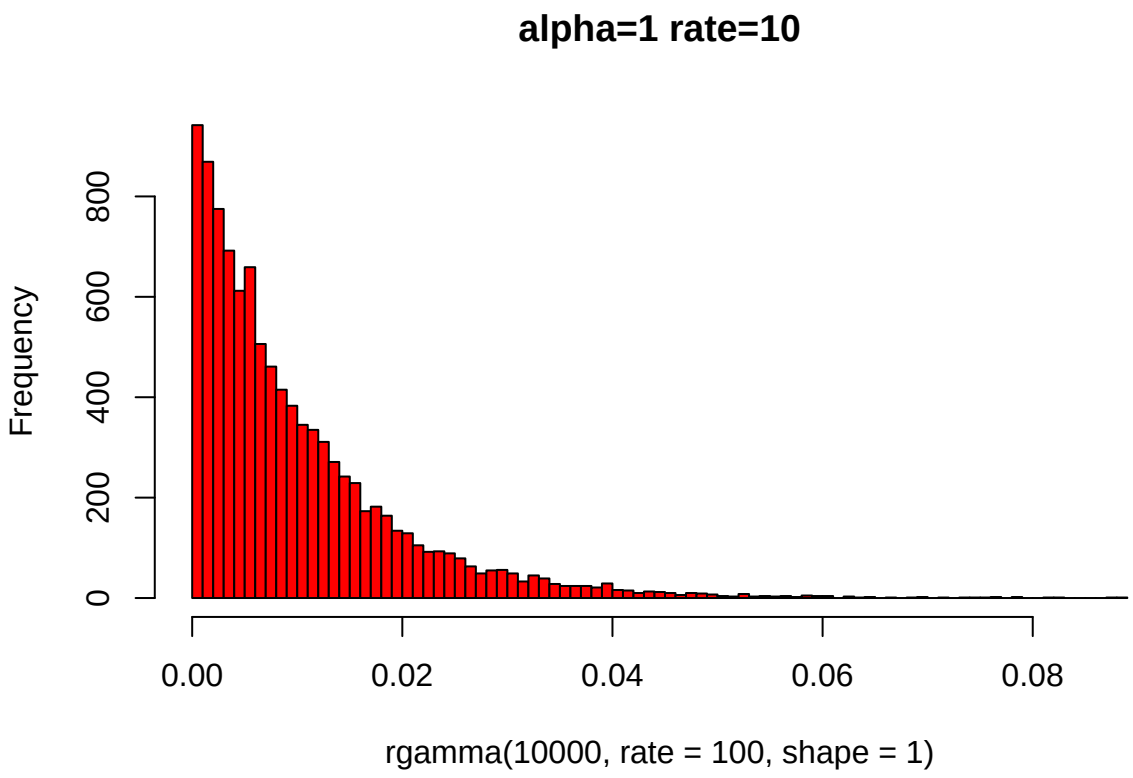
```
hist(rgamma(10000, rate = 1, shape = 1), col=rainbow(10)[1], br=100, main="alpha=1 rate=1")
```




```
hist(rgamma(10000, rate = 10, shape = 1), col=rainbow(10)[1], br=100, main="alpha=1 rate=10")
```



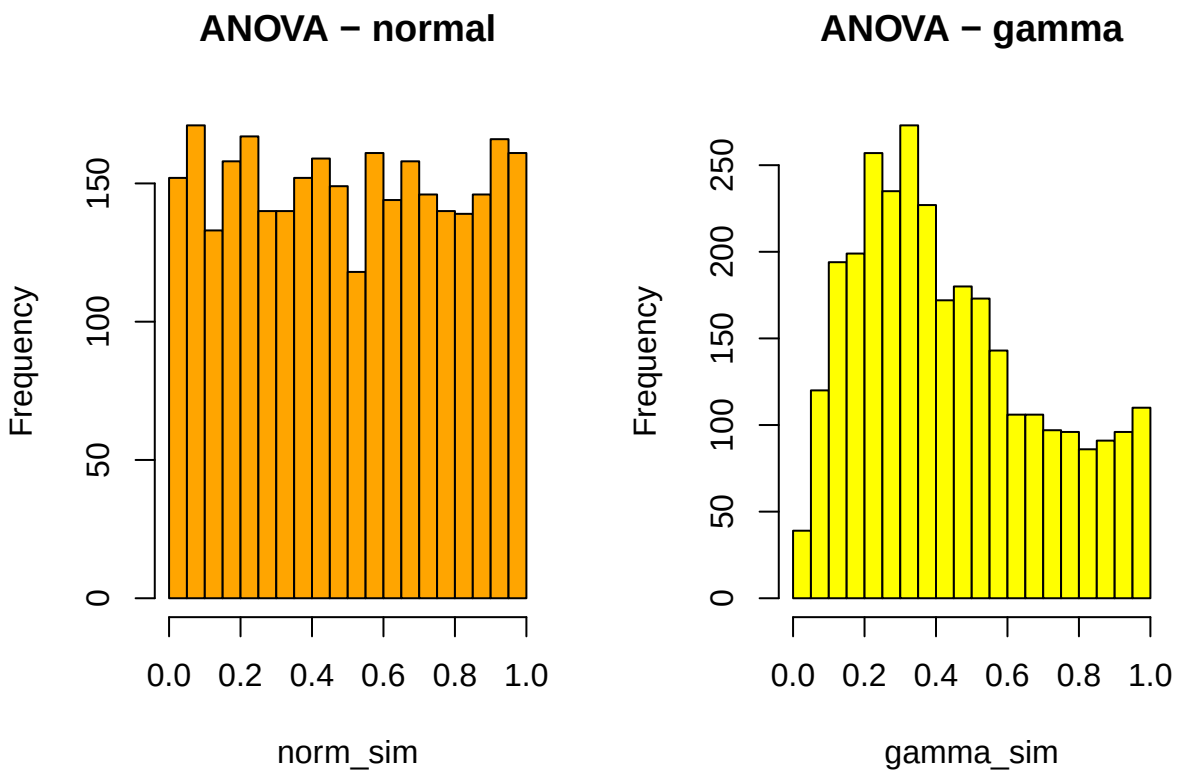
```r
hist(rgamma(10000, rate = 100, shape = 1), col=rainbow(10)[1], br=100, main="alpha=1 rate=10")
```



ANOVA p-values when H0 is valid but normality assumption does not hold

Now we can run the same ANOVA test drawing data from the gamma:

```r
# simulation 2: ANOVA when data derive from gamma
gamma_sim = sapply(1:3000, function(i) {
  x1 = rgamma(10, rate = 1, shape = 0.1)
  x2 = rgamma(10, rate = 1, shape = 0.1)
  pval = summary( aov(c(x1, x2) ~ groups ) )[[1]][1, "Pr(>F)"]
})
par(mfrow=c(1,2))
hist(norm_sim, col="orange", br=20, main="ANOVA - normal")
hist(gamma_sim, col="yellow", br=20, main="ANOVA - gamma")
```




```r
mean(norm_sim < 0.05)
```

```
## [1] 0.05066667
```

```r
mean(gamma_sim < 0.05)
```

```
## [1] 0.013
```

There is a deficiency of low p-values, which suggests the test becomes over-conservative (or not powerful enough), in this case. This is probably because the variance is overestimated due to outliers (which should not happen when the data is normally distributed).

Kruskal-Wallis

Non-parametric tests work by **ranking** the data, and therefore may be less sensitive to deviations from assumptions of normality and to outliers. One commonly used non-parametric equivalent of ANOVA is the Kruskal-Wallis test, which tests the equality of **medians** among groups. Note, however, that many non-parametric tests still assume similarity in shape between distributions (thus, they will also be sensitive to differences in variance).

We can apply the same simulation with KW:

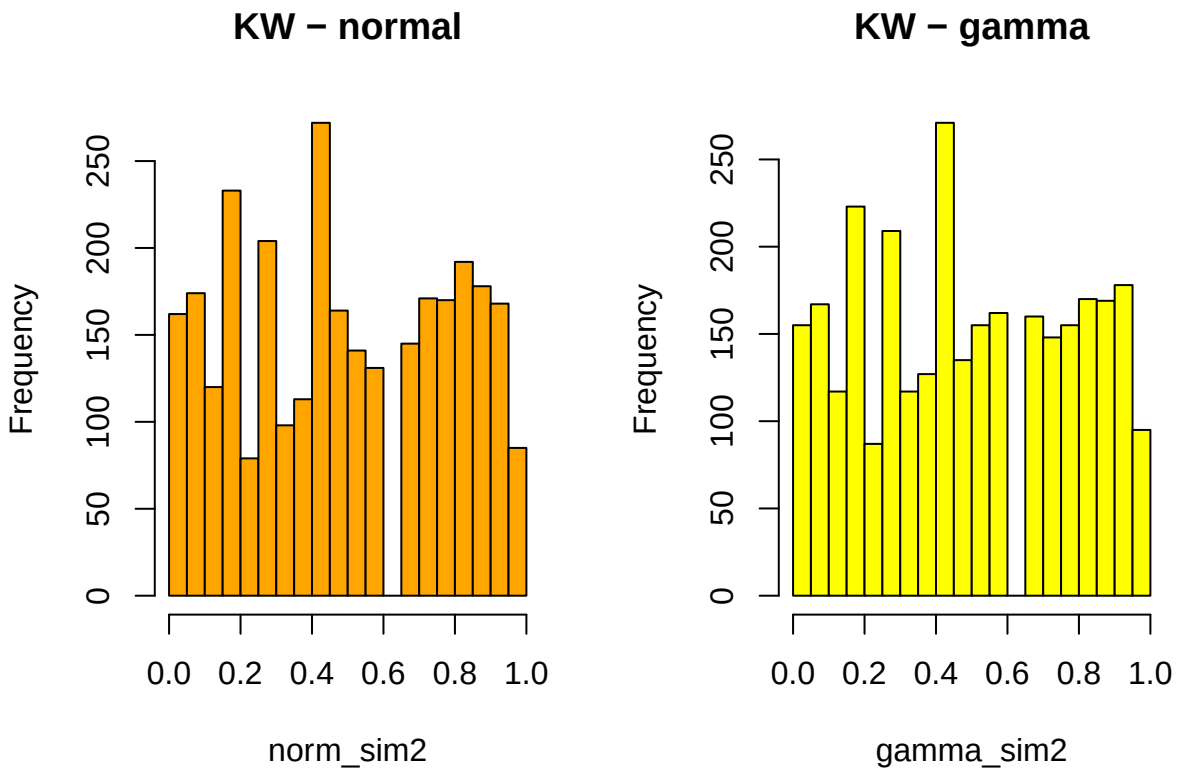
```r
# data from normal
norm_sim2 = sapply(1:3000, function(i) {
  x1 = rnorm(10, mean = 1, sd = 1)
  x2 = rnorm(10, mean = 1, sd = 1)
  pval = kruskal.test(c(x1, x2) ~ groups )$p.val
```

```
})

# data from gamma
gamma_sim2 = sapply(1:3000, function(i) {
  x1 = rgamma(10, rate = 1, shape = 0.1)
  x2 = rgamma(10, rate = 1, shape = 0.1)
  pval = kruskal.test(c(x1, x2) ~ groups )$p.val
})
```

Now plot the p-value distributions:

```
par(mfrow=c(1,2))
hist(norm_sim2, col="orange", br=20, main="KW - normal")
hist(gamma_sim2, col="yellow", br=20, main="KW - gamma")
```




```
mean(norm_sim2 < 0.05)
```

```
## [1] 0.054
```

```
mean(gamma_sim2 < 0.05)
```

```
## [1] 0.05166667
```

We see that the KW test p-values are not perfectly uniform (due to ranking and small sample size - not all p-values are possible). Also, KW can be less powerful than ANOVA (depending on the conditions) when, as it uses less information (due to ranking).

Importantly, however, the KW is **not sensitive** to the shape of the distributions - it performs equally on data from normal or gamma distributions.

A tip on method choice:

When you are *not sure* whether one or another method is theoretically superior, you may apply both, *and* report both results.

If both results are consistent, you will be on the safe side. When there is inconsistency, you should still report both results, and be more cautious about your interpretation - as there will be considerable chance that your result is non-reproducible.

Now we can apply the KW test to the sum of columns in the liver transcriptome dataset, for all the variables:

```
load("liver_transcriptome_v1.Rdata")
pvals = sapply( list(species, sex, run), function(varx) {
  kruskal.test( totalcounts ~ varx )$p.val
} )
round( pvals, 3)
```

```
## [1] 0.003 0.849 0.192
```

Summary: Using different methods (2-sample t-tests, ANOVA and Tukey HSD, Kruskal-Wallist), we repeatedly found the same results: There is a species effect on total read counts, but sex and run have limited influence. Among the 3 species, macaques appear to have the most different total read counts.

What could be the reason? biological, or technical? Given that same amount of total RNA was used, the authors (like many others) have assumed that the differences are technical and have **normalized** by dividing by total reads per sample. We will also normalize columns in a slightly different way, but later.

Summarizing data across replicates

We will continue working on the liver transcriptome dataset. Before starting transformation and normalization, we will exercise on summarising the data across replicates - as you remember each biological sample was processed twice, which we call "technical replicates".

```
load("/Users/msomel/Documents/misc/metu/ders/2380754_comp_2017/liver_transcriptome_v1.Rdata")
ls()
```

```
##  [1] "gamma_sim"   "gamma_sim2"  "groups"      "indv"        "mat"
##  [6] "norm_sim"    "norm_sim2"   "norm_sim_x"  "pval"        "pvals"
## [11] "run"         "sex"         "species"     "totalcounts" "x1"
## [16] "x2"
```

The motivation here is that we cannot treat replicates as independent samples in downstream analyses. This is necessary to avoid **pseudoreplication**, data that are treated as independent observations but are not independent. Can you think of what kind of problems pseudoreplication can bring about?

Demo of pseudoreplication

Compare two sets of integers that differ w.r.t. their means. Imagine that they represent brain gene expression measurements you made for a gene, from 5 humans and 4 chimps:

```
x = 1:5
y = 3:6
t.test(x, y)
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -1.5667, df = 6.9808, p-value = 0.1613
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.7652203  0.7652203
## sample estimates:
## mean of x mean of y
##       3.0       4.5
```

This is *not* significant. Now, let's say you are interested in pursuing this question, but instead of collecting more data from independent samples, you just conduct technical replicates, and obtain the same measurements.

Would you obtain the same result if you plugged in the new data (2 replicates each) into the t-test?

```
t.test(rep(x, 2), rep(y, 2))  # significant
```

```
##
##  Welch Two Sample t-test
##
## data:  rep(x, 2) and rep(y, 2)
## t = -2.3694, df = 15.996, p-value = 0.03074
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.842104 -0.157896
## sample estimates:
## mean of x mean of y
##       3.0       4.5
```

Why the difference? We have twice more information in the second case and thus same difference between the two groups becomes more reliable - less probable to occur by chance. But *if* we do not really have independent extra information, treating our data as such will lead to non-reproducible results.

How to deal with the replicate effect?

a. We can randomly choose one replicate. But this is imperfect, as we're loosing data.

b. We can take the average per gene per individual, and use this data downstream. This is a suboptimal solution, as we are also loosing data (i.e. the variation due to technical noise). We thus cannot estimate how much noise there occurred due to technical variance. However, this approach is also simpler.

c. We can include in individual effect in the statistical models we will use (e.g. models for differential expression) as a **random effect**, as opposed to a **fixed effect**:

https://onlinecourses.science.psu.edu/stat502/node/162

The latter is the most sophisticated approach, and would allow distinguishing technical variance and biological variance within species-sex groups.

Exercise: summarizing per replicate

We will use both approaches (b) and (c) in our analyses. Now we will conduct (b): summarise the dataset by calculating the **mean** value per replicate pair per gene. Let's first start with individual one, and then run the loop over all individuals:

```
z = indv[1]
z
```

```
## [1] HSM1
## 18 Levels: HSF1 HSF2 HSF3 HSM1 HSM2 HSM3 PTF1 PTF2 PTF3 PTM1 PTM2 ... RMM3
```

```
# we can use this for subsetting
head(mat[, indv == z] )
```

```
##                 R1L1.HSM1 R5L2.HSM1
## ENSG00000000003        60        61
## ENSG00000000005         0         0
## ENSG00000000419        17        22
## ENSG00000000457        50        64
## ENSG00000000460         9         6
## ENSG00000000938        32        41
```

```
# we can further calculate the mean for each row
z_expr = rowMeans( mat[, indv == z] )

# this is the same as:
z_expr = apply( mat[, indv == z], 1, mean )
length( z_expr )
```

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

```
# we should always double check that it worked
head( cbind( mat[, indv == z], z_expr) )
```

```
##                 R1L1.HSM1 R5L2.HSM1 z_expr
## ENSG00000000003        60        61   60.5
## ENSG00000000005         0         0    0.0
## ENSG00000000419        17        22   19.5
## ENSG00000000457        50        64   57.0
## ENSG00000000460         9         6    7.5
## ENSG00000000938        32        41   36.5
```

Now try create a new matrix `mat2` where replicates are summarized:

```
mat2 = sapply( unique( indv ), function(x) {
  rowMeans( mat[, indv == x] )
})
# row means calculated and stored as columns of a new matrix
dim( mat2 )
```

```
## [1] 20689    18
```

```
head( mat2 )
```

```
##                 [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
## ENSG00000000003 60.5 287.0 212.0 164.5 204.5 255.5 232.0 210.5 214.5 674.0
## ENSG00000000005  0.0   0.5   1.0   0.0   0.0   0.0   0.5   0.5   2.0   0.0
## ENSG00000000419 19.5  50.0  24.5  40.5  29.5  33.5  36.0  39.0  33.0  41.0
## ENSG00000000457 57.0  65.5  58.5  45.5 137.0  44.5  34.5  36.5 114.5  57.0
## ENSG00000000460  7.5   4.0   3.5   3.0   6.0   2.0   2.0   3.5   4.0   1.5
## ENSG00000000938 36.5  42.0  40.0  22.0  99.0  36.5  14.0  21.5  52.0  17.0
##                 [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18]
## ENSG00000000003 150.0 342.5 352.5 240.0 179.5 193.0 214.0    84
## ENSG00000000005   0.0   0.5   0.0   1.0   0.0   0.5   0.5     0
## ENSG00000000419  30.5  29.0  45.0  28.5  31.0  32.5  46.5    28
## ENSG00000000457  31.0 119.5  81.5  59.0  47.5  43.5  74.5    38
## ENSG00000000460   8.5   3.5   2.0   5.0   8.5   2.5   5.0     6
## ENSG00000000938  32.5  15.5  36.0  27.0  32.5  38.0  29.0   105
```

We need to add the column names back. Luckily `sapply` remains true to the original order of the vector:

```
colnames( mat2 ) = unique( indv )
head( mat2 )
```

```
##                 HSM1  PTF1  RMM1  HSF1  PTM1  RMF1  RMF2  HSM2  PTF2  RMM2
## ENSG00000000003 60.5 287.0 212.0 164.5 204.5 255.5 232.0 210.5 214.5 674.0
## ENSG00000000005  0.0   0.5   1.0   0.0   0.0   0.0   0.5   0.5   2.0   0.0
## ENSG00000000419 19.5  50.0  24.5  40.5  29.5  33.5  36.0  39.0  33.0  41.0
## ENSG00000000457 57.0  65.5  58.5  45.5 137.0  44.5  34.5  36.5 114.5  57.0
## ENSG00000000460  7.5   4.0   3.5   3.0   6.0   2.0   2.0   3.5   4.0   1.5
## ENSG00000000938 36.5  42.0  40.0  22.0  99.0  36.5  14.0  21.5  52.0  17.0
##                  HSF2  PTM2  RMM3  RMF3  HSM3  PTF3  PTM3 HSF3
## ENSG00000000003 150.0 342.5 352.5 240.0 179.5 193.0 214.0   84
## ENSG00000000005   0.0   0.5   0.0   1.0   0.0   0.5   0.5    0
## ENSG00000000419  30.5  29.0  45.0  28.5  31.0  32.5  46.5   28
## ENSG00000000457  31.0 119.5  81.5  59.0  47.5  43.5  74.5   38
## ENSG00000000460   8.5   3.5   2.0   5.0   8.5   2.5   5.0    6
## ENSG00000000938  32.5  15.5  36.0  27.0  32.5  38.0  29.0  105
```

We can also first define unique elements in `indv` as a `character` vector (not `factor`). In this way, `sapply` adds the character elements as column names in the resulting matrix automatically:

```
mat2 = sapply( as.character( unique( indv ) ), function(x) {
  rowMeans( mat[, indv == x] )
})
# column names added automatically
head( mat2 )
```

```
##                 HSM1  PTF1  RMM1  HSF1  PTM1  RMF1  RMF2  HSM2  PTF2  RMM2
## ENSG00000000003 60.5 287.0 212.0 164.5 204.5 255.5 232.0 210.5 214.5 674.0
## ENSG00000000005  0.0   0.5   1.0   0.0   0.0   0.0   0.5   0.5   2.0   0.0
## ENSG00000000419 19.5  50.0  24.5  40.5  29.5  33.5  36.0  39.0  33.0  41.0
## ENSG00000000457 57.0  65.5  58.5  45.5 137.0  44.5  34.5  36.5 114.5  57.0
## ENSG00000000460  7.5   4.0   3.5   3.0   6.0   2.0   2.0   3.5   4.0   1.5
```

```
## ENSG00000000938 36.5  42.0  40.0  22.0  99.0  36.5  14.0  21.5  52.0  17.0
##                  HSF2  PTM2  RMM3  RMF3  HSM3  PTF3  PTM3 HSF3
## ENSG00000000003 150.0 342.5 352.5 240.0 179.5 193.0 214.0   84
## ENSG00000000005   0.0   0.5   0.0   1.0   0.0   0.5   0.5    0
## ENSG00000000419  30.5  29.0  45.0  28.5  31.0  32.5  46.5   28
## ENSG00000000457  31.0 119.5  81.5  59.0  47.5  43.5  74.5   38
## ENSG00000000460   8.5   3.5   2.0   5.0   8.5   2.5   5.0    6
## ENSG00000000938  32.5  15.5  36.0  27.0  32.5  38.0  29.0  105
```

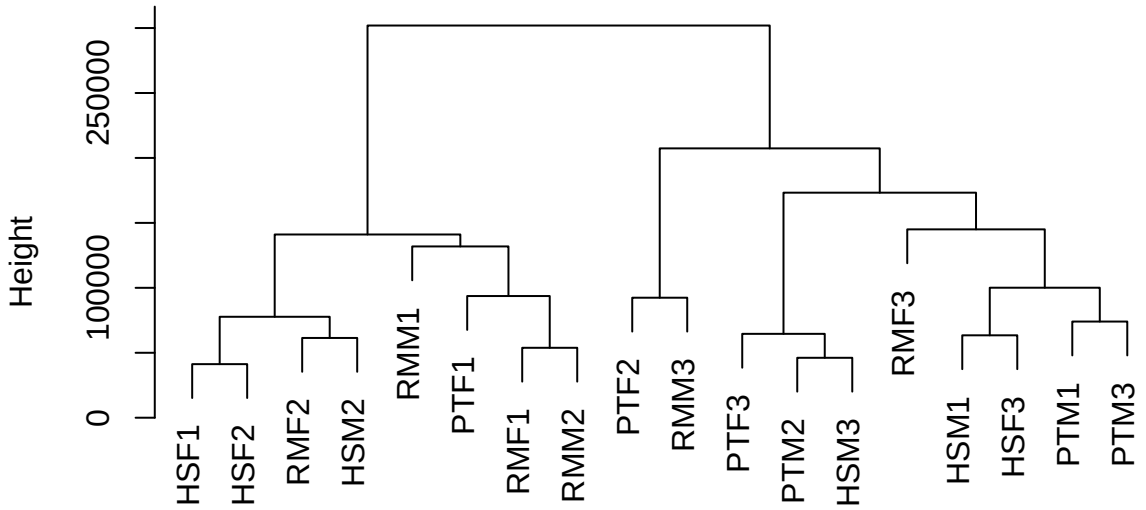
Now we need to define factors again, based on the new matrix:

```
species2 = factor( substr( colnames( mat2 ), 1, 2 ) )
sex2 = factor( substr( colnames( mat2 ), 3, 3 ) )
table(species2, sex2)
```

```
##         sex2
## species2 F M
##       HS 3 3
##       PT 3 3
##       RM 3 3
```

And draw the new tree:

```
plot( hclust(dist(t(mat2))), sub="", xlab="", main="")
```



Still no species or sex clustering visible.

Data transformation

Transformation refers to applying a simple mathematical function to the whole dataset. The function should be back-convertible. There are can be multiple rationales behind transformation:

- Avoid outliers,
- Convert the data into a normal-like (symmetric) shape,

- Improvate data visualisation (differences among low values become visible),
- Equalise variances among groups / categories where the mean may be different

These steps, if successful, may then allow using parametric methods, like ANOVA, instead of non-parametric tests.

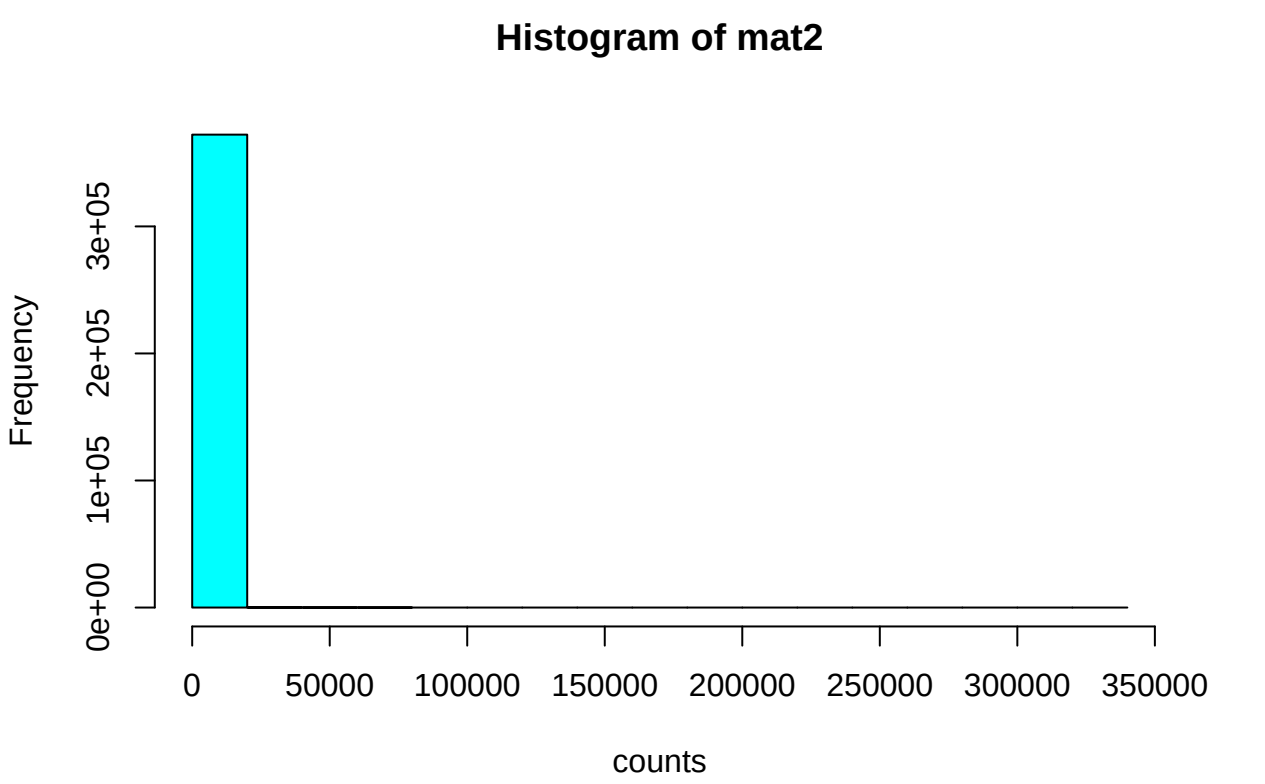
Log and square root are transformations applied to right-skewed data.

Antilog or square can be used for left-skewed data.

Arcsine can be used for proportions.

In our liver transcriptome case, to decide on whether we need to transform the data and what approach to use, let's check the histogram of the full dataset:

```
hist( mat2, col=5, xlab='counts' )
```

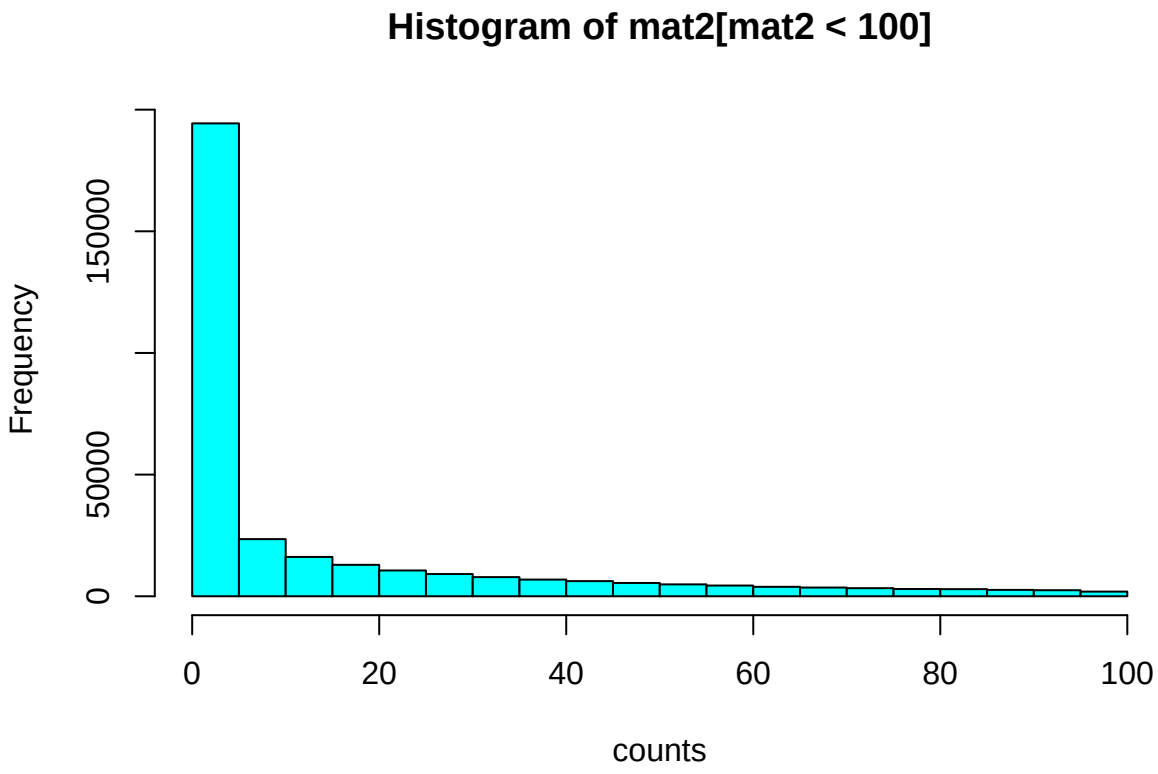



```
# how many elements are plotted?
length( mat2 )
```

```
## [1] 372402
```

The data appear highly **right-skewed**. But is it because of a few outlier genes, or only some samples? How about removing all values `> 100`?

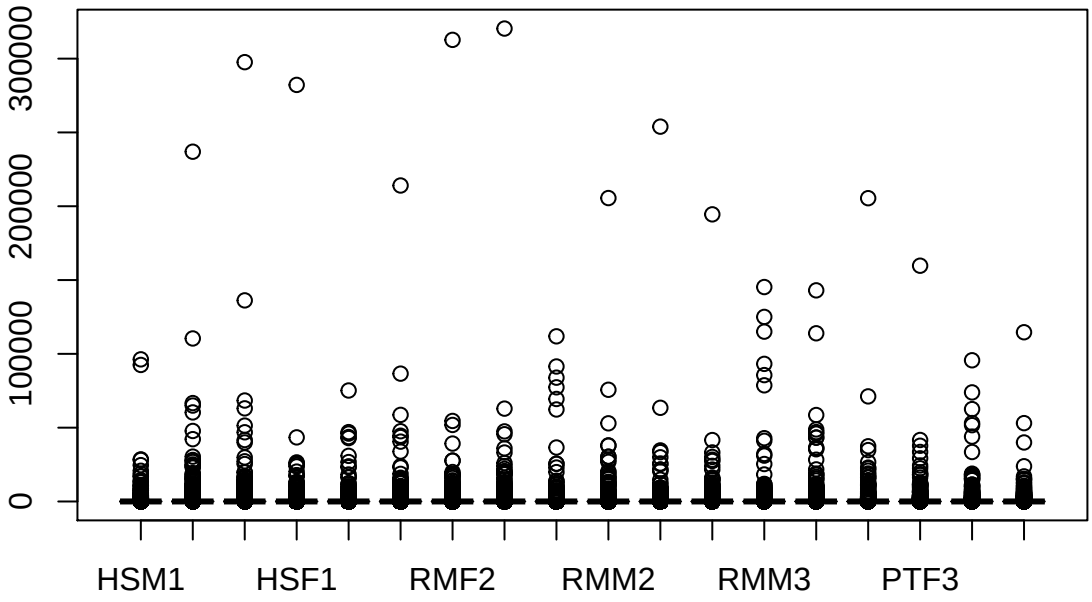
```
hist( mat2[ mat2<100 ], col=5, xlab='counts' )
```




```
# still skewed
```

We may also wonder if the skewness applies equally to all columns, for which we can draw a boxplot, as we did before:

```
boxplot( mat2, col = 6)
```



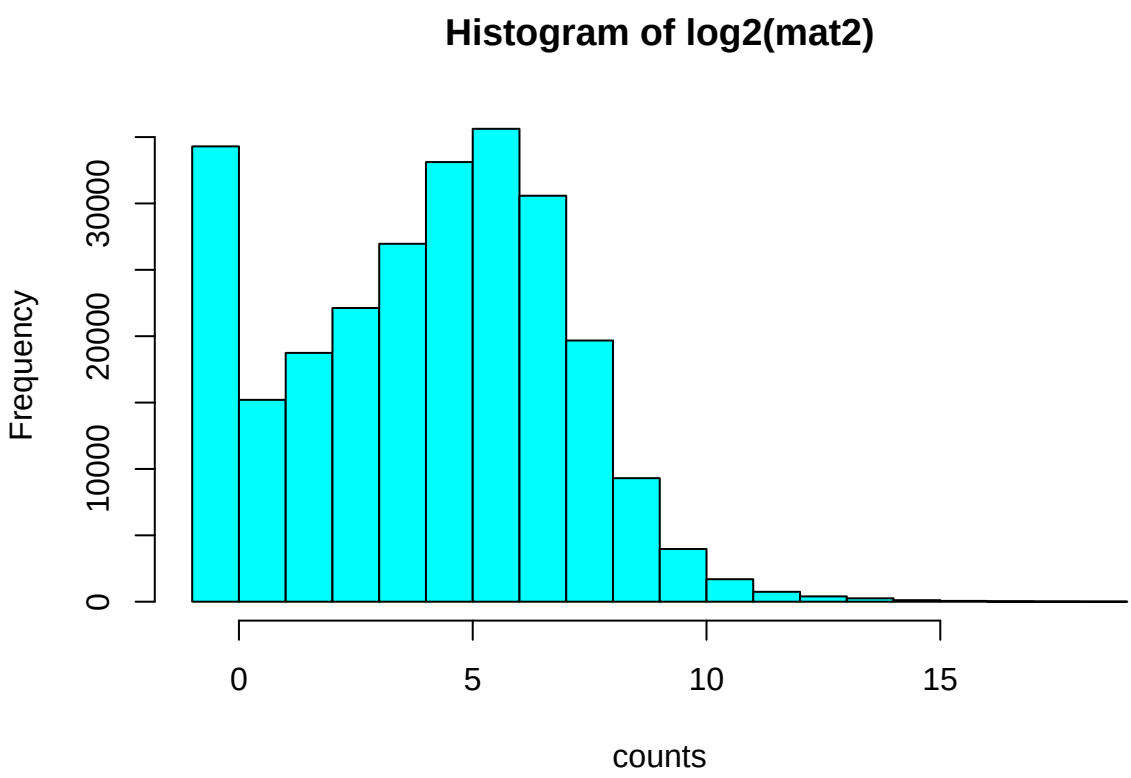

```
# indeed
```

Log transformation

An alternative approach is to take the logarithm, which can render the right-skewed distributions relatively symmetric.

We can use the `log2` function, as log of base 2 (than `e`) are easy to interpret:

```
hist( log2(mat2), col=5, xlab='counts' )
```



Do you think there may be a problem here? Check this:

```
head( log2(mat2) )
```

```
##                     HSM1      PTF1     RMM1     HSF1     PTM1     RMF1
## ENSG00000000003 5.918863  8.164907 7.727920 7.361944 7.675957 7.997179
## ENSG00000000005     -Inf -1.000000 0.000000     -Inf     -Inf     -Inf
## ENSG00000000419 4.285402  5.643856 4.614710 5.339850 4.882643 5.066089
## ENSG00000000457 5.832890  6.033423 5.870365 5.507795 7.098032 5.475733
## ENSG00000000460 2.906891  2.000000 1.807355 1.584963 2.584963 1.000000
## ENSG00000000938 5.189825  5.392317 5.321928 4.459432 6.629357 5.189825
##                      RMF2      HSM2     PTF2      RMM2     HSF2      PTM2
## ENSG00000000003  7.857981  7.717676 7.744834 9.3966048 7.228819  8.419960
## ENSG00000000005 -1.000000 -1.000000 1.000000      -Inf     -Inf -1.000000
## ENSG00000000419  5.169925  5.285402 5.044394 5.3575520 4.930737  4.857981
## ENSG00000000457  5.108524  5.189825 6.839204 5.8328900 4.954196  6.900867
## ENSG00000000460  1.000000  1.807355 2.000000 0.5849625 3.087463  1.807355
## ENSG00000000938  3.807355  4.426265 5.700440 4.0874628 5.022368  3.954196
##                     RMM3     RMF3     HSM3      PTF3      PTM3     HSF3
## ENSG00000000003 8.461479 7.906891 7.487840  7.592457  7.741467 6.392317
```

```
## ENSG00000000005     -Inf 0.000000     -Inf -1.000000 -1.000000     -Inf
## ENSG00000000419 5.491853 4.832890 4.954196  5.022368  5.539159 4.807355
## ENSG00000000457 6.348728 5.882643 5.569856  5.442943  6.219169 5.247928
## ENSG00000000460 1.000000 2.321928 3.087463  1.321928  2.321928 2.584963
## ENSG00000000938 5.169925 4.754888 5.022368  5.247928  4.857981 6.714246
```

0's are converted into `-Inf` and obviously not plotted. What to do about this?

One common approach is adding `+1` to all data. Thus :

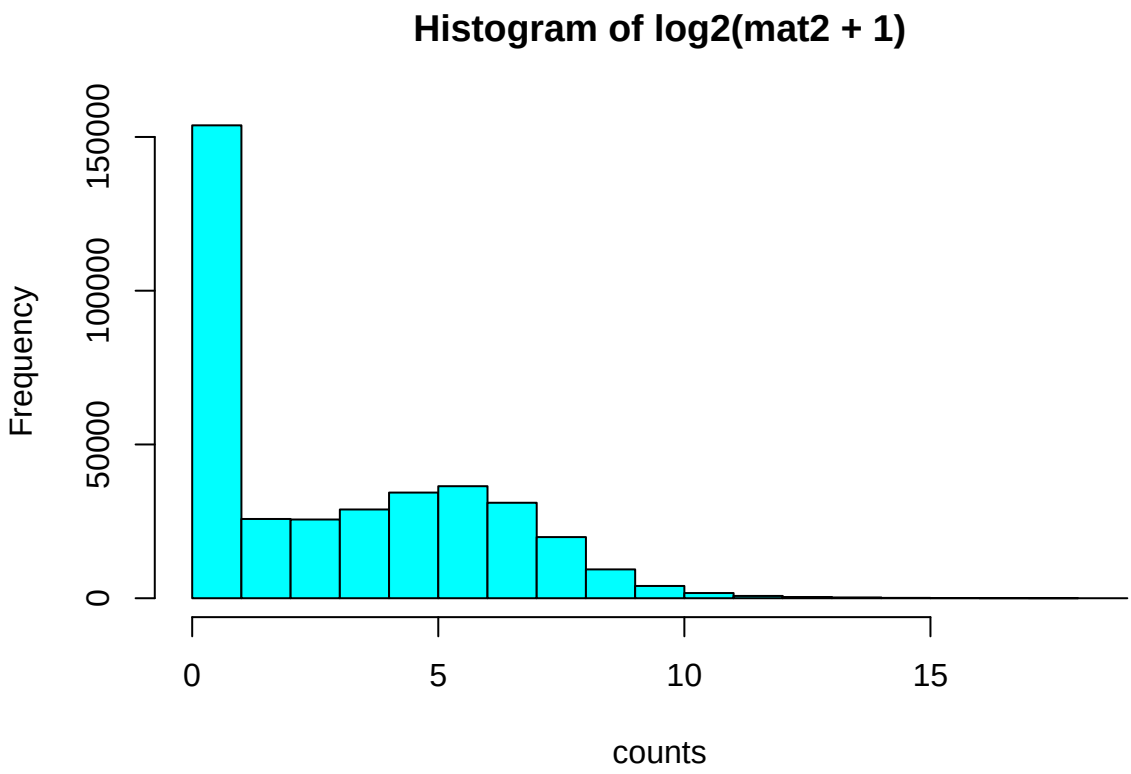
```
head( mat2 )
```

```
##                 HSM1  PTF1  RMM1  HSF1  PTM1  RMF1  RMF2  HSM2  PTF2  RMM2
## ENSG00000000003 60.5 287.0 212.0 164.5 204.5 255.5 232.0 210.5 214.5 674.0
## ENSG00000000005  0.0   0.5   1.0   0.0   0.0   0.0   0.5   0.5   2.0   0.0
## ENSG00000000419 19.5  50.0  24.5  40.5  29.5  33.5  36.0  39.0  33.0  41.0
## ENSG00000000457 57.0  65.5  58.5  45.5 137.0  44.5  34.5  36.5 114.5  57.0
## ENSG00000000460  7.5   4.0   3.5   3.0   6.0   2.0   2.0   3.5   4.0   1.5
## ENSG00000000938 36.5  42.0  40.0  22.0  99.0  36.5  14.0  21.5  52.0  17.0
##                  HSF2  PTM2  RMM3  RMF3  HSM3  PTF3  PTM3 HSF3
## ENSG00000000003 150.0 342.5 352.5 240.0 179.5 193.0 214.0   84
## ENSG00000000005   0.0   0.5   0.0   1.0   0.0   0.5   0.5    0
## ENSG00000000419  30.5  29.0  45.0  28.5  31.0  32.5  46.5   28
## ENSG00000000457  31.0 119.5  81.5  59.0  47.5  43.5  74.5   38
## ENSG00000000460   8.5   3.5   2.0   5.0   8.5   2.5   5.0    6
## ENSG00000000938  32.5  15.5  36.0  27.0  32.5  38.0  29.0  105
```

```
head( log2(mat2 + 1) )
```

```
##                      HSM1      PTF1     RMM1     HSF1     PTM1     RMF1
## ENSG00000000003 5.942515 8.1699250 7.734710 7.370687 7.682995 8.002815
## ENSG00000000005 0.000000 0.5849625 1.000000 0.000000 0.000000 0.000000
## ENSG00000000419 4.357552 5.6724253 4.672425 5.375039 4.930737 5.108524
## ENSG00000000457 5.857981 6.0552824 5.894818 5.539159 7.108524 5.507795
## ENSG00000000460 3.087463 2.3219281 2.169925 2.000000 2.807355 1.584963
## ENSG00000000938 5.228819 5.4262648 5.357552 4.523562 6.643856 5.228819
##                      RMF2      HSM2     PTF2     RMM2     HSF2      PTM2
## ENSG00000000003 7.8641861 7.7245139 7.751544 9.398744 7.238405 8.4241663
## ENSG00000000005 0.5849625 0.5849625 1.584963 0.000000 0.000000 0.5849625
## ENSG00000000419 5.2094534 5.3219281 5.087463 5.392317 4.977280 4.9068906
## ENSG00000000457 5.1497471 5.2288187 6.851749 5.857981 5.000000 6.9128893
## ENSG00000000460 1.5849625 2.1699250 2.321928 1.321928 3.247928 2.1699250
## ENSG00000000938 3.9068906 4.4918531 5.727920 4.169925 5.066089 4.0443941
##                     RMM3     RMF3     HSM3      PTF3      PTM3     HSF3
## ENSG00000000003 8.465566 7.912889 7.495855 7.5999128 7.7481928 6.409391
## ENSG00000000005 0.000000 1.000000 0.000000 0.5849625 0.5849625 0.000000
## ENSG00000000419 5.523562 4.882643 5.000000 5.0660892 5.5698556 4.857981
## ENSG00000000457 6.366322 5.906891 5.599913 5.4757334 6.2384047 5.285402
## ENSG00000000460 1.584963 2.584963 3.247928 1.8073549 2.5849625 2.807355
## ENSG00000000938 5.209453 4.807355 5.066089 5.2854022 4.9068906 6.727920
```

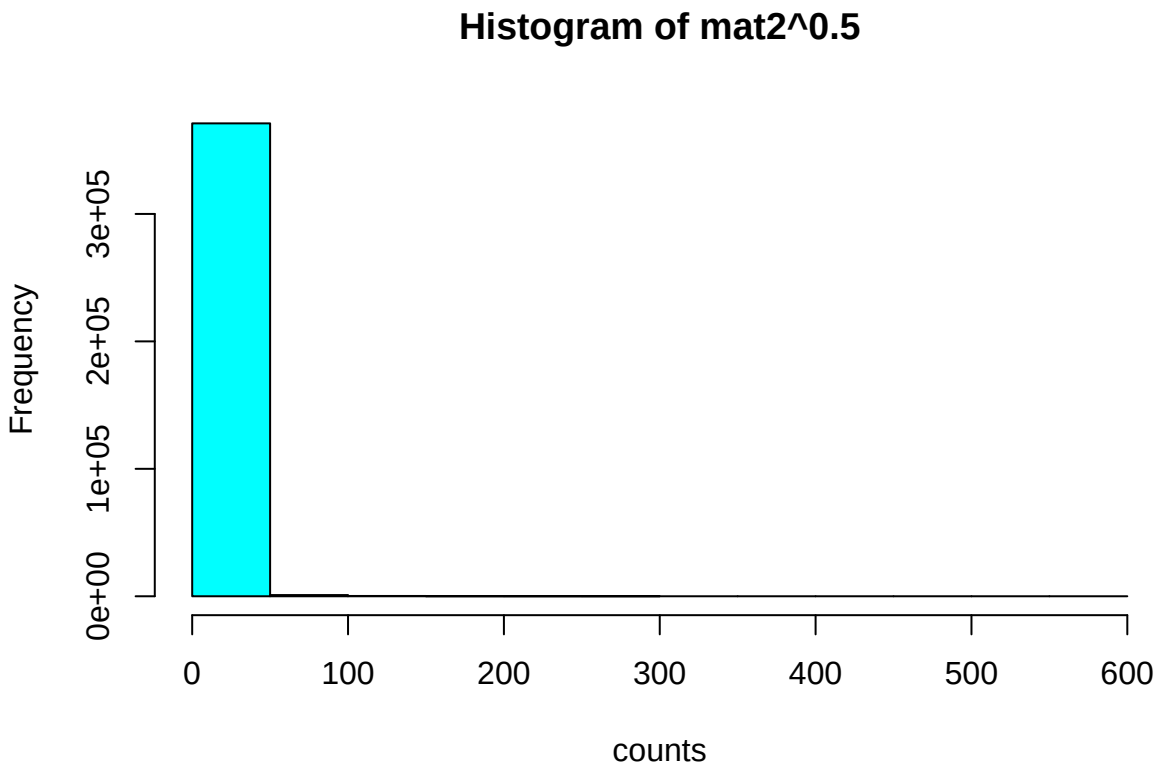
```r
# 0s remain 0
hist( log2(mat2 + 1), col=5, xlab='counts' )
```



A nice *"mosque plot"*! What do you think the minarette represents? And the dome?

Square root transformation is also commonly used for right-skewed **count data**. You thus avoid adding `+1`. This is also what the authors did in the original study (Blekhman et al. 2009 Genome Research):

```r
hist( mat2^0.5, col=5, xlab='counts' )
```



But square root appears not as efficient as log.

Stabilizing the variance

As we noted, log (and square root) transformation can be helpful for viewing / interpreting the distribution. But as importantly, these transformations help **stabilizes the variance**. That is, the transformation renders variance & mean largely independent, both across high & low expressed genes, and for each gene, among low and high expressed groups. This is important because many statistical tests assume comparable variance among groups (e.g. ANOVA).

To see how log transformation helps stabilize the variance, let's check the effect by:

- comparing variance vs mean of each gene using the raw data,
- then repeating this using log-transformed data,

We may use only the human subset of the data, which makes interpretation of variance more straightforward.

Let's create a matrix of human samples, can call it `hmat`:

```
# create the subset
hmat2 = mat2[, species2=="HS"]
# this would also have worked
hmat2 = subset(mat2, select = species2=="HS")
dim(hmat2)
```

```
## [1] 20689     6
```

```
head(hmat2)
```

```
##                 HSM1  HSF1  HSM2  HSF2  HSM3 HSF3
## ENSG00000000003 60.5 164.5 210.5 150.0 179.5   84
## ENSG00000000005  0.0   0.0   0.5   0.0   0.0    0
## ENSG00000000419 19.5  40.5  39.0  30.5  31.0   28
## ENSG00000000457 57.0  45.5  36.5  31.0  47.5   38
## ENSG00000000460  7.5   3.0   3.5   8.5   8.5    6
## ENSG00000000938 36.5  22.0  21.5  32.5  32.5  105
```

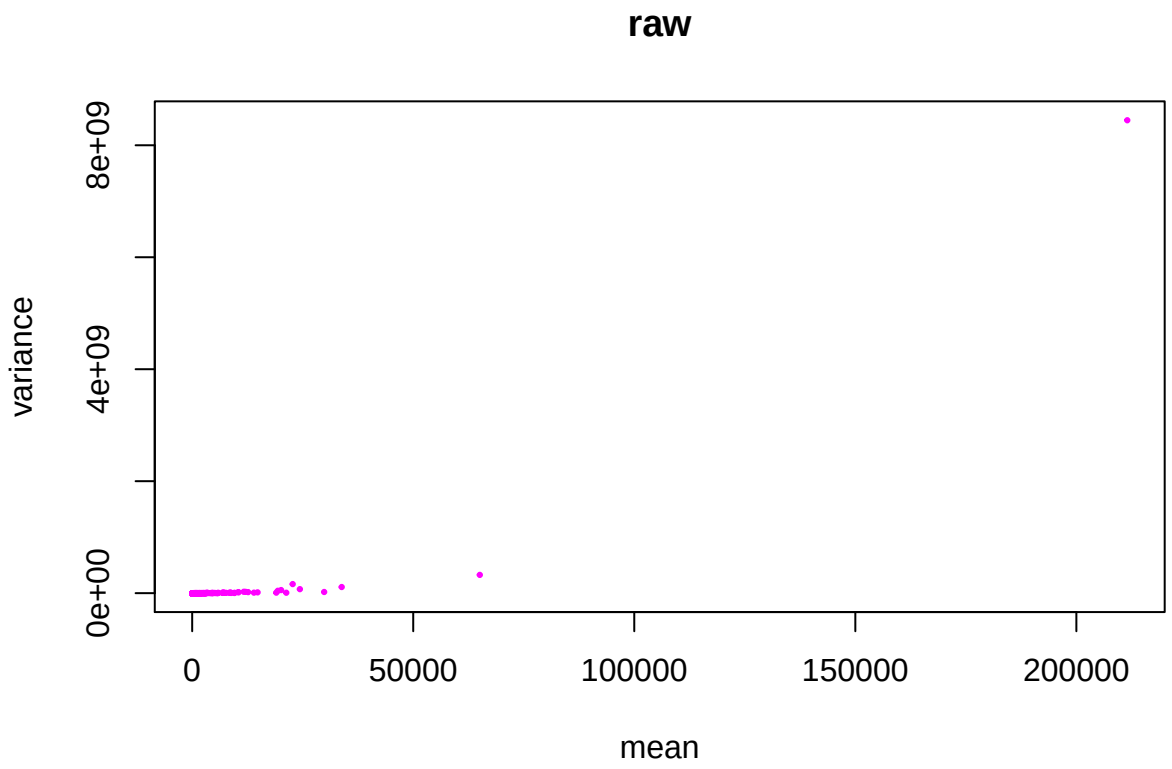
```
# the log2 version
lhmat2 = log2(hmat2+1)
```

Now calculate the row means and variances in `hmat`:

```
rm = rowMeans(hmat2)
# this would also have worked
rm = apply(hmat2, 1, mean)
rv = apply(hmat2, 1, var)
```

Plot the relationship between mean and variance, each point being a gene:

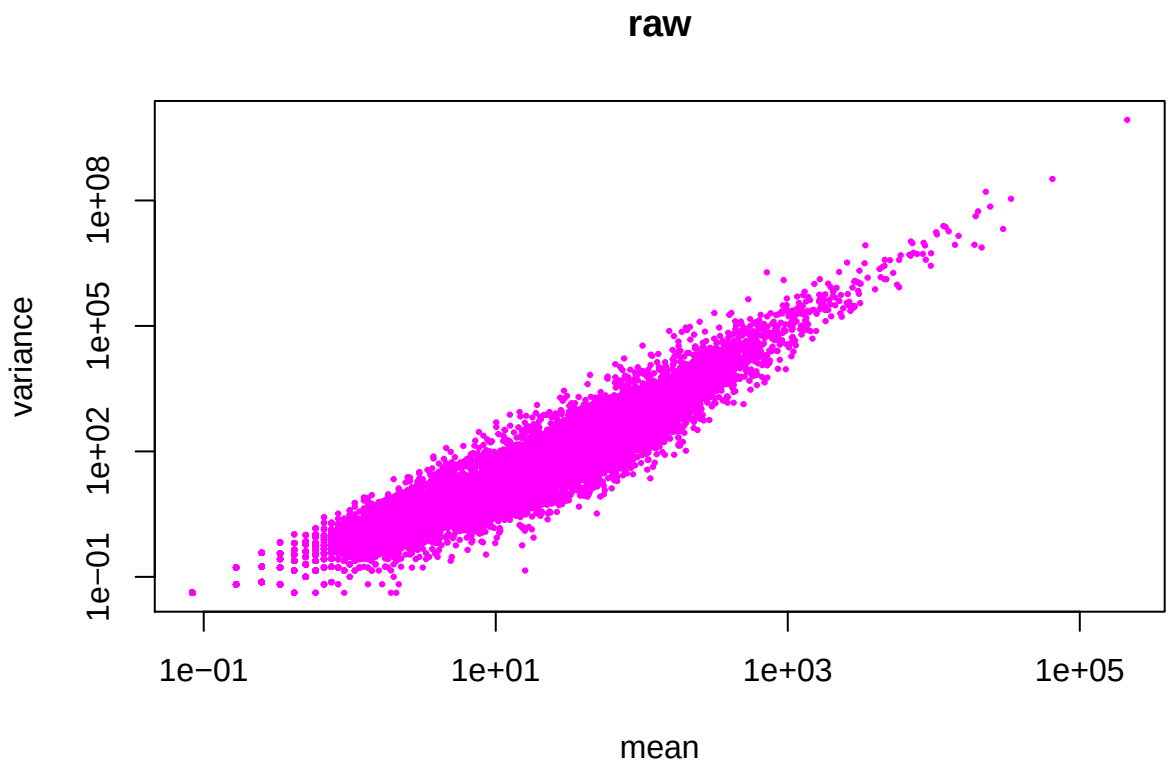
```
par(mfrow=c(1,1))
plot(rm, rv, col=6, ylab='variance', xlab='mean', main='raw', pch=19, cex=0.3)
```



```
# try plotting by converting both axes to log
plot(rm, rv, log='xy', col=6, ylab='variance', xlab='mean', main='raw', pch=19, cex=0.3)
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 4314 x values <= 0 omitted
## from logarithmic plot
```

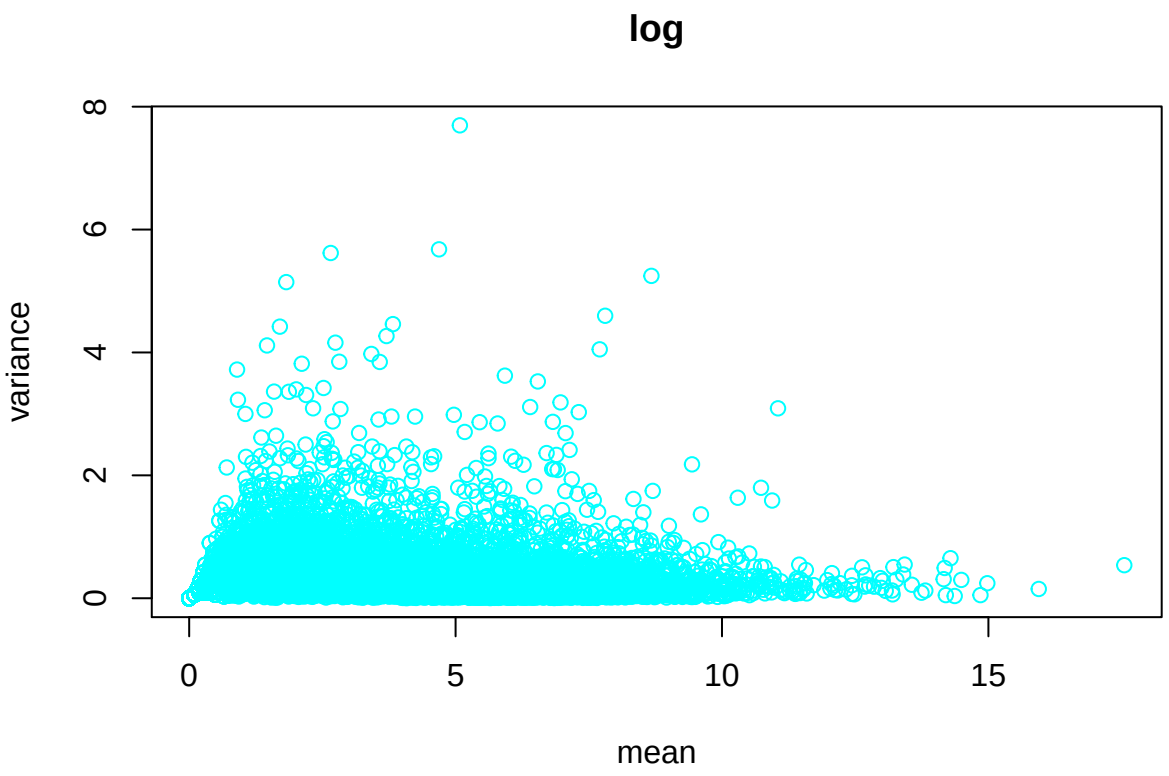
```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 4314 y values <= 0 omitted
## from logarithmic plot
```



Apparently some values are omitted (which we could avoid by `+1`, but won't do this here).

Now let's repeat the same, using the log-transformed dataset:

```
lrm = rowMeans(lhmat2)
lrv = apply(lhmat2, 1, var)
plot(lrm, lrv, col=5, ylab='variance', xlab='mean', main='log')
```



The correlation coefficient

We could also add the correlation coefficients to these plots to better comprehend the relationship and how it is influenced by the transformation.

There are two commonly used coefficients: the parametric **Pearson** (covariance between x and y / pooled variance of x and y), and the non-parametric **Spearman**. Remember that the squared correlation coefficient

```
# pearson corr coef is the default
cor(rm, rv)
```

```
## [1] 0.906999
```

```
cor(lrm, lrv)
```

```
## [1] 0.1319166
```

```
# spearman non-parametric coef
cor(rm, rv, method = "spearman")
```

```
## [1] 0.9848056
```

```
cor(lrm, lrv, method = "spearman")
```

```
## [1] 0.463686
```

To calculate the p-value one can also use:

```r
cor.test(lrm, lrv, method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  lrm and lrv
## t = 19.141, df = 20687, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1185032 0.1452819
## sample estimates:
##       cor
## 0.1319166
```

```r
cor.test(rm, rv, method = "spearman")
```

```
## Warning in cor.test.default(rm, rv, method = "spearman"): Cannot compute
## exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  rm and rv
## S = 2.2426e+10, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.9848056
```

Additional exercise: adding text on plots

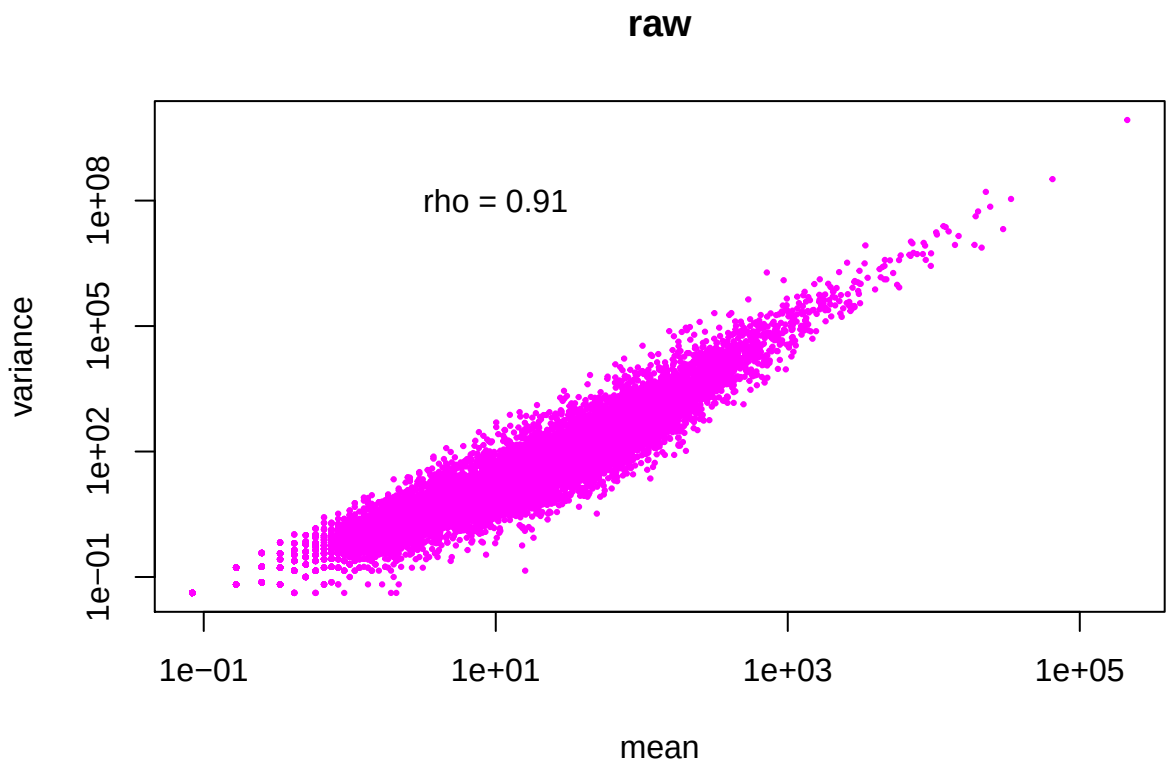
Adding the correlation coefficient to existing plots is using the function `text`, but you have to choose the location manually:

```r
plot(rm, rv, log='xy', col=6,
     ylab='variance', xlab='mean', main='raw', pch=19, cex=0.3)
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 4314 x values <= 0 omitted
## from logarithmic plot
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 4314 y values <= 0 omitted
## from logarithmic plot
```

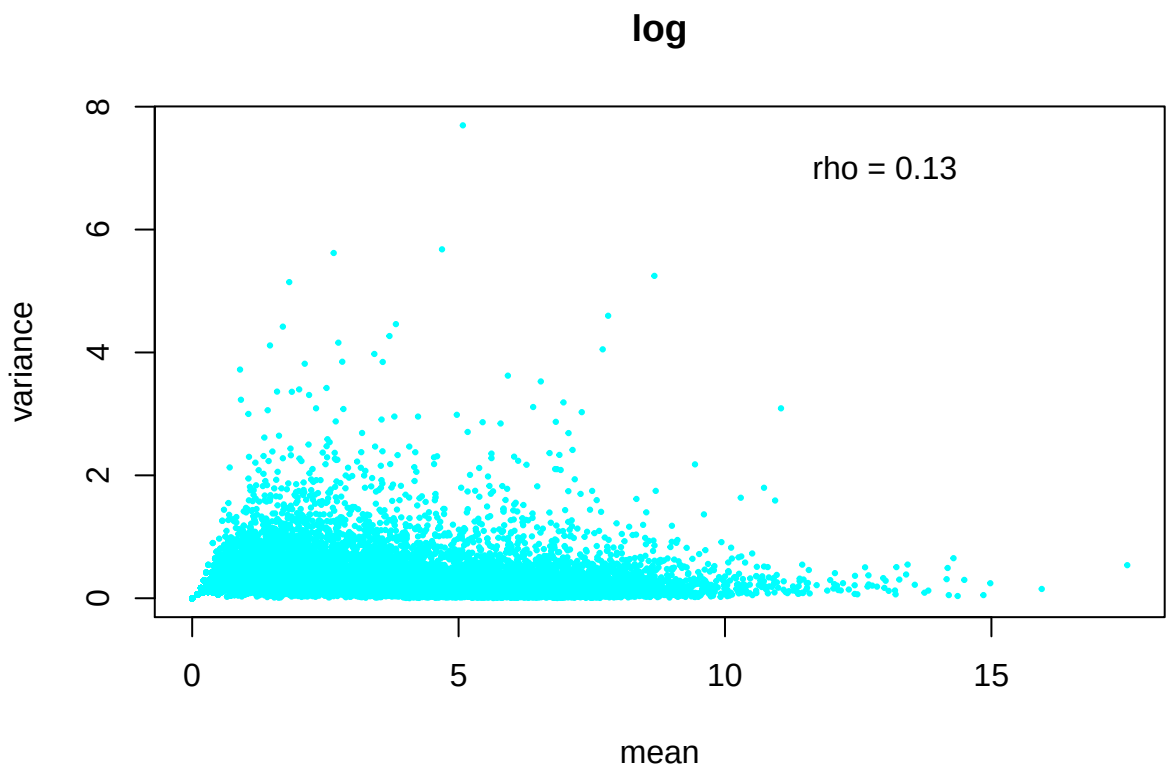
```r
text( x = 1e1, y = 1e8,
      paste("rho =", round( cor(rm, rv), 2 ) ) )
```




```
plot(lrm, lrv, col=5,
     ylab='variance', xlab='mean', main='log', pch=19, cex=0.3)
cor(lrm, lrv)
```

```
## [1] 0.1319166
```

```
text( x = 13, y = 7,
      paste("rho =", round( cor(lrm, lrv), 2 ) ) )
```



What can you conclude? Apparently, the tight positive correlation between mean and variance is partly, but not fully, removed by using log transformation. You may thus:

- use log transformed data with parametric models while keeping in mind that certain assumptions may not hold,
- use non-parametric models for estimation or hypothesis testing,
- or consider more powerful variance stabilization methods.

Here we will consider the first two options.

Exercise: Transform the data and redraw the sample tree

Now let's log transform the full summarized matrix `mat2`, and call it `mat3`:

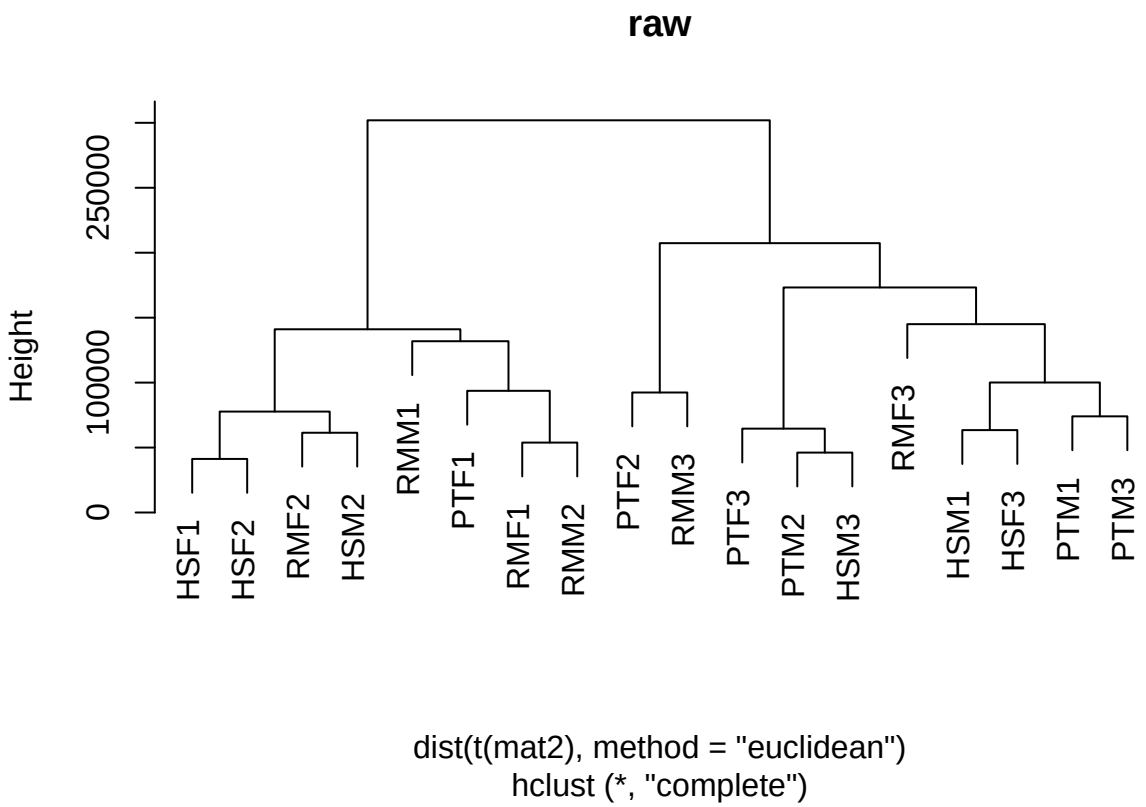
```
mat3 = log2 ( mat2 + 1 )  # log transform
head( mat3 )
```

```
##                      HSM1      PTF1      RMM1      HSF1      PTM1      RMF1
## ENSG00000000003 5.942515 8.1699250 7.734710 7.370687 7.682995 8.002815
## ENSG00000000005 0.000000 0.5849625 1.000000 0.000000 0.000000 0.000000
## ENSG00000000419 4.357552 5.6724253 4.672425 5.375039 4.930737 5.108524
## ENSG00000000457 5.857981 6.0552824 5.894818 5.539159 7.108524 5.507795
## ENSG00000000460 3.087463 2.3219281 2.169925 2.000000 2.807355 1.584963
## ENSG00000000938 5.228819 5.4262648 5.357552 4.523562 6.643856 5.228819
##                      RMF2      HSM2      PTF2      RMM2      HSF2      PTM2
## ENSG00000000003 7.8641861 7.7245139 7.751544 9.398744 7.238405 8.4241663
## ENSG00000000005 0.5849625 0.5849625 1.584963 0.000000 0.000000 0.5849625
```

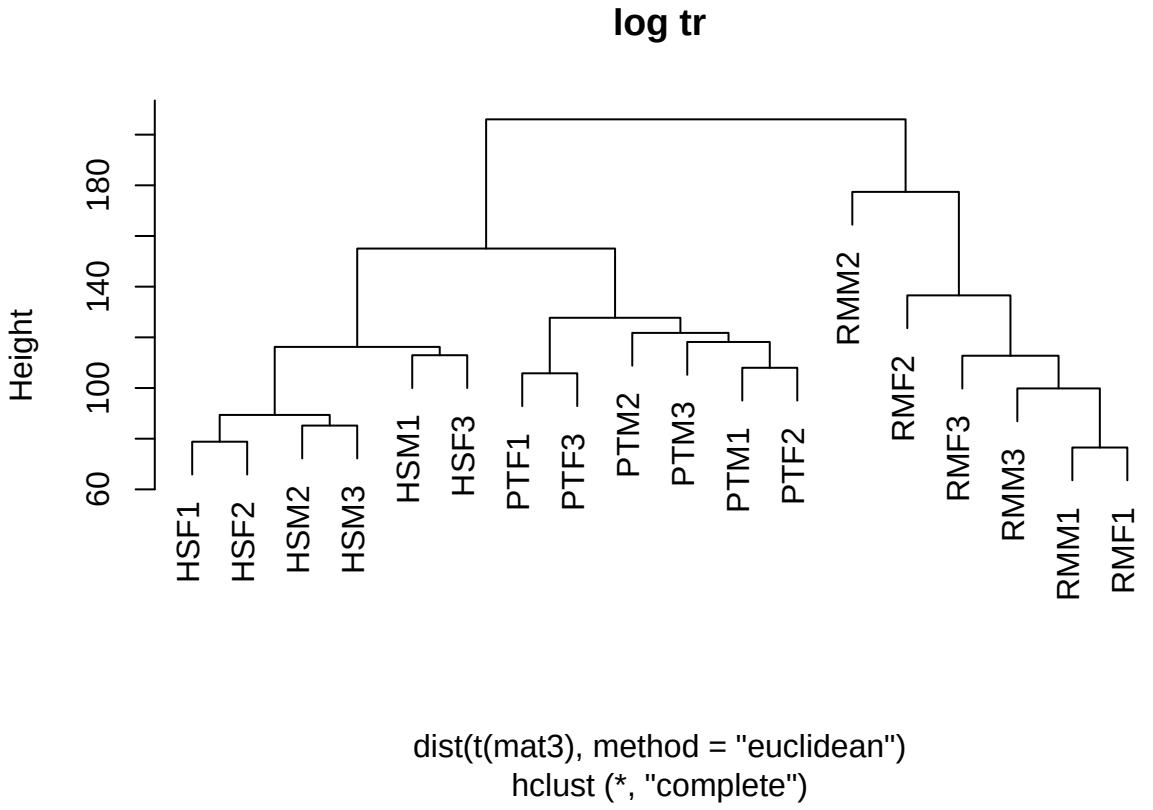
```
## ENSG00000000419 5.2094534 5.3219281 5.087463 5.392317 4.977280 4.9068906
## ENSG00000000457 5.1497471 5.2288187 6.851749 5.857981 5.000000 6.9128893
## ENSG00000000460 1.5849625 2.1699250 2.321928 1.321928 3.247928 2.1699250
## ENSG00000000938 3.9068906 4.4918531 5.727920 4.169925 5.066089 4.0443941
##                      RMM3     RMF3     HSM3      PTF3      PTM3     HSF3
## ENSG00000000003 8.465566 7.912889 7.495855 7.5999128 7.7481928 6.409391
## ENSG00000000005 0.000000 1.000000 0.000000 0.5849625 0.5849625 0.000000
## ENSG00000000419 5.523562 4.882643 5.000000 5.0660892 5.5698556 4.857981
## ENSG00000000457 6.366322 5.906891 5.599913 5.4757334 6.2384047 5.285402
## ENSG00000000460 1.584963 2.584963 3.247928 1.8073549 2.5849625 2.807355
## ENSG00000000938 5.209453 4.807355 5.066089 5.2854022 4.9068906 6.727920
```

Let's create a hierarchical clustering tree with the log-transformed data:

```
# this is the raw data tree
plot( hclust( dist( t( mat2 ), method = "euclidean")  ), main="raw")
```



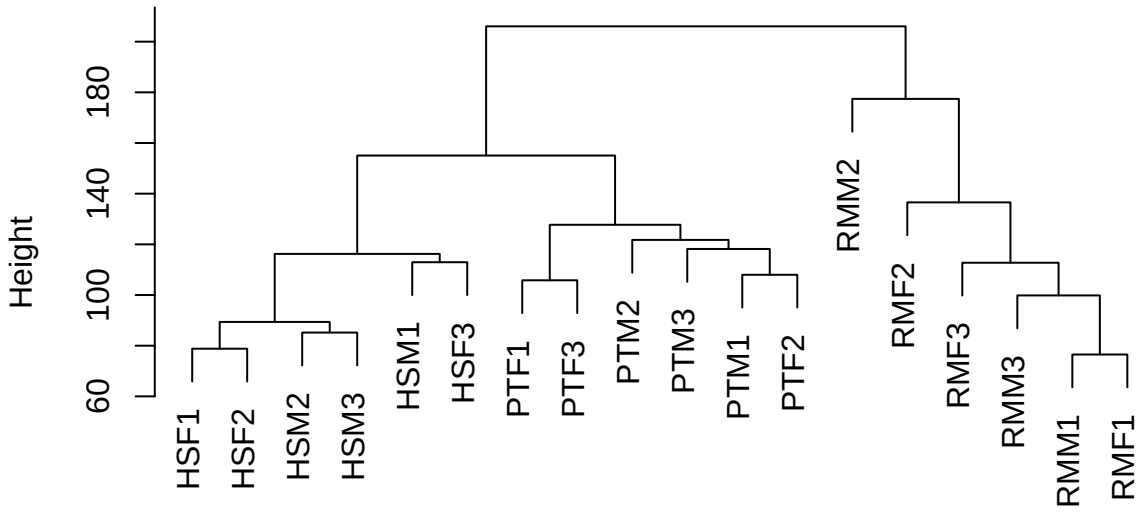

```
# and the log-tr tree
plot( hclust( dist( t( mat3 ), method = "euclidean")  ), main="log tr")
```



Cutting trees and creating clusters using `cutree`

Interestingly now we start observing the phylogenetic clustering. Can we distinguish the clusters?

```
mat3tree = hclust( dist( t( mat3 ), method = "euclidean") )
plot( mat3tree )
```




```
# divide into 2 groups
cutree( mat3tree, k = 2 )
```

```
## HSM1 PTF1 RMM1 HSF1 PTM1 RMF1 RMF2 HSM2 PTF2 RMM2 HSF2 PTM2 RMM3 RMF3 HSM3
##    1    1    2    1    1    2    2    1    1    2    1    1    2    2    1
## PTF3 PTM3 HSF3
##    1    1    1
```

```
cl1 = cutree( mat3tree, k = 2)
cl1
```

```
## HSM1 PTF1 RMM1 HSF1 PTM1 RMF1 RMF2 HSM2 PTF2 RMM2 HSF2 PTM2 RMM3 RMF3 HSM3
##    1    1    2    1    1    2    2    1    1    2    1    1    2    2    1
## PTF3 PTM3 HSF3
##    1    1    1
```

How can we check that the humans and chimpanzees are all in one cluster?

The %in% operator

Here we could use the %in% operator, which is useful for checking the match between elements of 2 vectors. It returns a logical vector indicating if there is a match or not for its left operand:

```
10:1 %in% 2:4
```

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

```r
c("a", "b", "c", "d") %in% c("b", "c")
```

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

```r
# however, these wouldn't have worked
10:1 == 2:4
```

```
## Warning in 10:1 == 2:4: longer object length is not a multiple of shorter
## object length
```

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

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

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

```r
# note that in the latter 2 cases, R recycles the shorter vectors
```

So now we can check - are all humans and chimpanzees cluster 1 member?

```r
cl1[species2 %in% c("HS", "PT")]
```

```
## HSM1 PTF1 HSF1 PTM1 HSM2 PTF2 HSF2 PTM2 HSM3 PTF3 PTM3 HSF3
##    1    1    1    1    1    1    1    1    1    1    1    1
```

Yes. Even though this case is trivial, this type of matching can be very useful when the data is large.

We could also have asked - are all cluster 1 members human and chimpanzee?

```r
table( species2[cl1 == 1] )
```

```
##
## HS PT RM
##  6  6  0
```

We can also try to:

```r
# divide into 4 groups
cl4 = cutree( mat3tree, k = 4 )
sapply(1:4, function(i) names( which( cl4 == i) ) )
```

```
## [[1]]
## [1] "HSM1" "HSF1" "HSM2" "HSF2" "HSM3" "HSF3"
##
## [[2]]
## [1] "PTF1" "PTM1" "PTF2" "PTM2" "PTF3" "PTM3"
##
## [[3]]
## [1] "RMM1" "RMF1" "RMF2" "RMM3" "RMF3"
##
## [[4]]
## [1] "RMM2"
```

One can also divide by the height of the tree:

```
sort( cutree( mat3tree, h = 140 ) )
```

```
## HSM1 HSF1 HSM2 HSF2 HSM3 HSF3 PTF1 PTM1 PTF2 PTM2 PTF3 PTM3 RMM1 RMF1 RMF2
##    1    1    1    1    1    1    2    2    2    2    2    2    3    3    3
## RMM3 RMF3 RMM2
##    3    3    4
```

```
sort( cutree( mat3tree, h = 100 ) )
```

```
## HSM1 PTF1 RMM1 RMF1 RMM3 HSF1 HSM2 HSF2 HSM3 PTM1 RMF2 PTF2 RMM2 PTM2 RMF3
##    1    2    3    3    3    4    4    4    4    5    6    7    8    9   10
## PTF3 PTM3 HSF3
##   11   12   13
```

Does one of the groups really correspond to human and chimp?

```
cutree( mat3tree, k = 3 )[ species2 %in% c("HS", "PT") ]
```

```
## HSM1 PTF1 HSF1 PTM1 HSM2 PTF2 HSF2 PTM2 HSM3 PTF3 PTM3 HSF3
##    1    1    1    1    1    1    1    1    1    1    1    1
```

```
# note that the below code wouldn't have worked the same way:
cutree( mat3tree, k = 3 )[ species2 == c("HS", "PT") ]
```

```
## HSM1 PTF1 HSF2 PTM2 HSM3 PTF3
##    1    1    1    1    1    1
```

And this code allows for showing clusters on the plot, by clicking on the plot:

```
identify ( mat3tree, MAXCLUSTER = 3 )
```

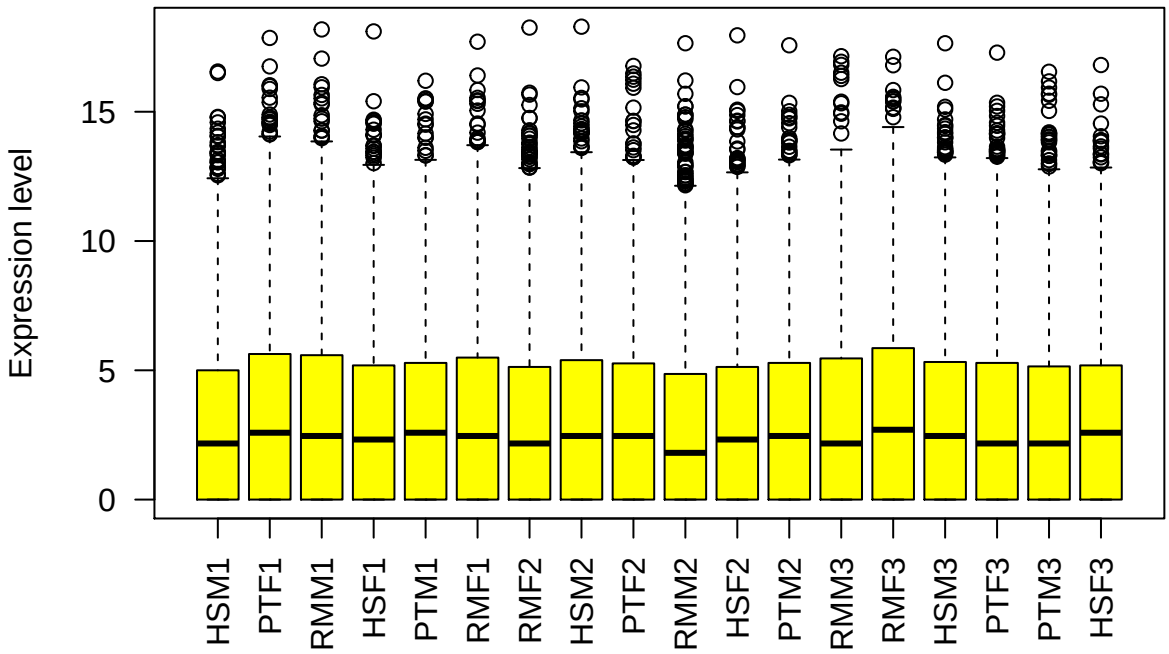
Why do we now see biological clustering, do you think?

Normalization

We are still far from starting differential gene expression analysis. One issue we have not yet addressed is differences among samples with respect to the total counts read from each.

Let's again study the distributions across columns in the log2 transformed dataset, again using `boxplot`:

```
boxplot( mat3, las=2, col=7, ylab='Expression level')
```



Compared to the boxplots we draw from `mat`, the distributions now appear more comparable. This is likely because the differences we saw in earlier boxplots were largely due to a small number of highly expressed genes (which also caused the hierarchical clustering trees to have weird topologies).

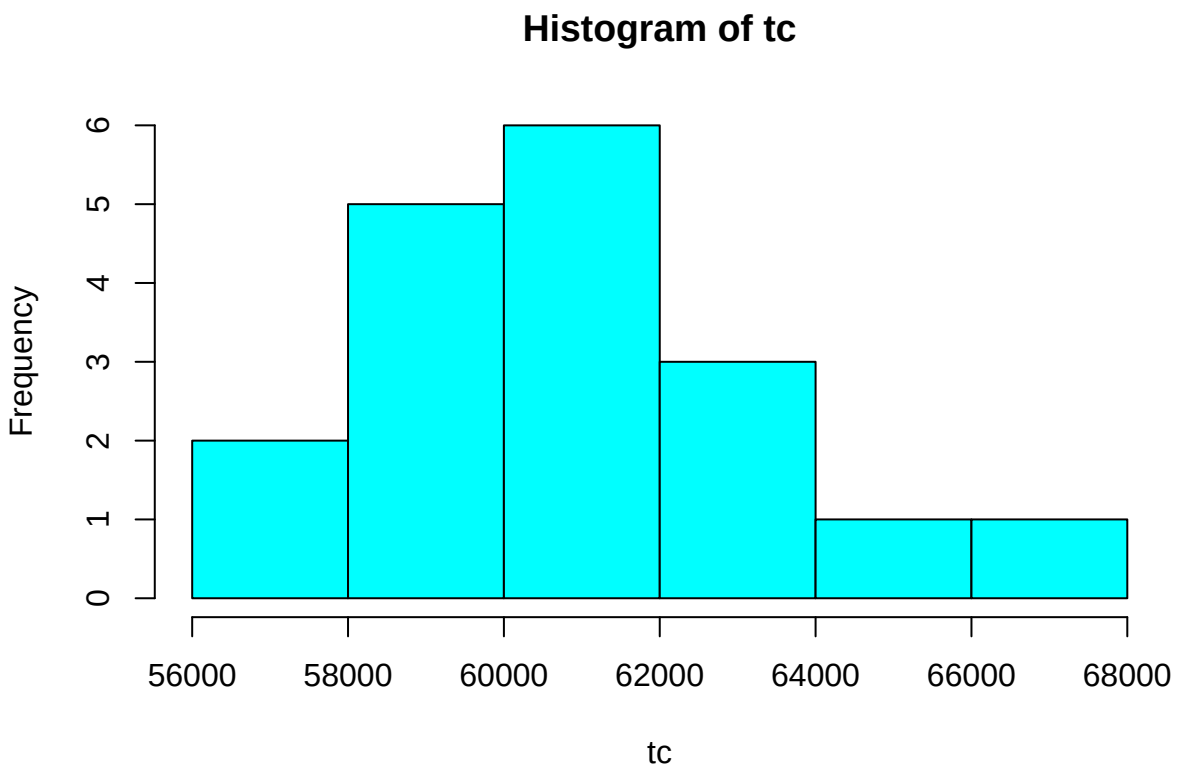
Still, the samples do present variation in their distributions. Were the different samples sequenced to similar depth? Is there any confounding with species or sex?

Additional exercise: species effect on log-tr total counts

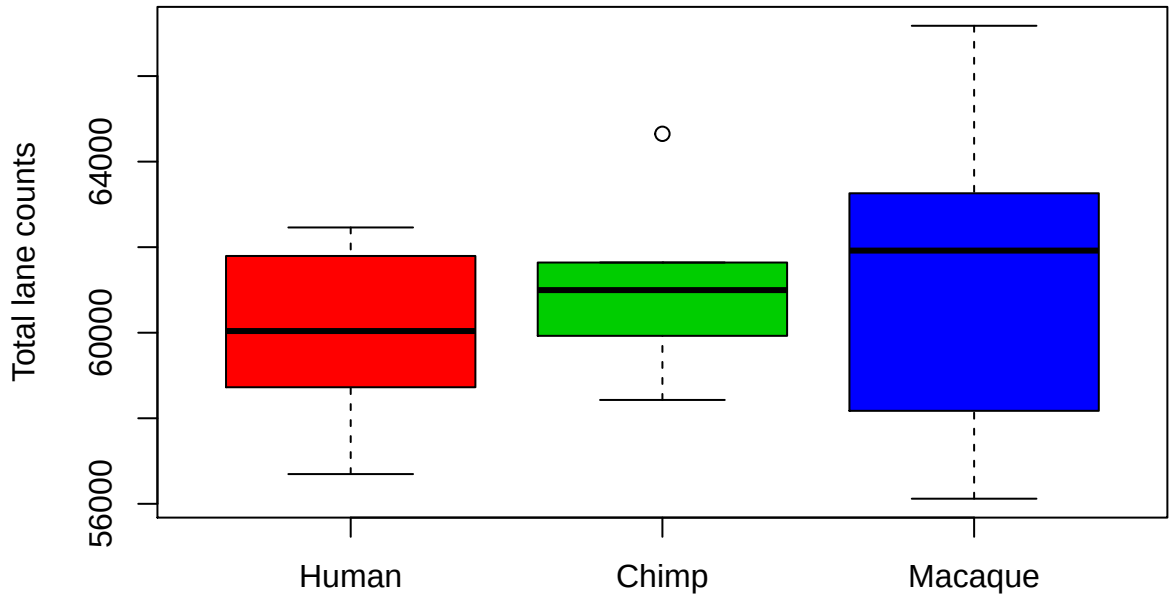
```
# new total counts
tc = colSums(mat3)
tc
```

```
##      HSM1      PTF1      RMM1      HSF1      PTM1      RMF1      RMF2      HSM2
## 56694.99 64650.15 63259.71 59346.04 61640.81 62528.58 58174.02 62460.52
##      PTF2      RMM2      HSF2      PTM2      RMM3      RMF3      HSM3      PTF3
## 61023.17 56119.81 58724.97 60969.71 61312.40 67175.69 61794.51 59926.88
##      PTM3      HSF3
## 58428.19 60732.21
```

```
# redefine the window
par(mfrow=c(1,1))
hist( tc, col=5)
```

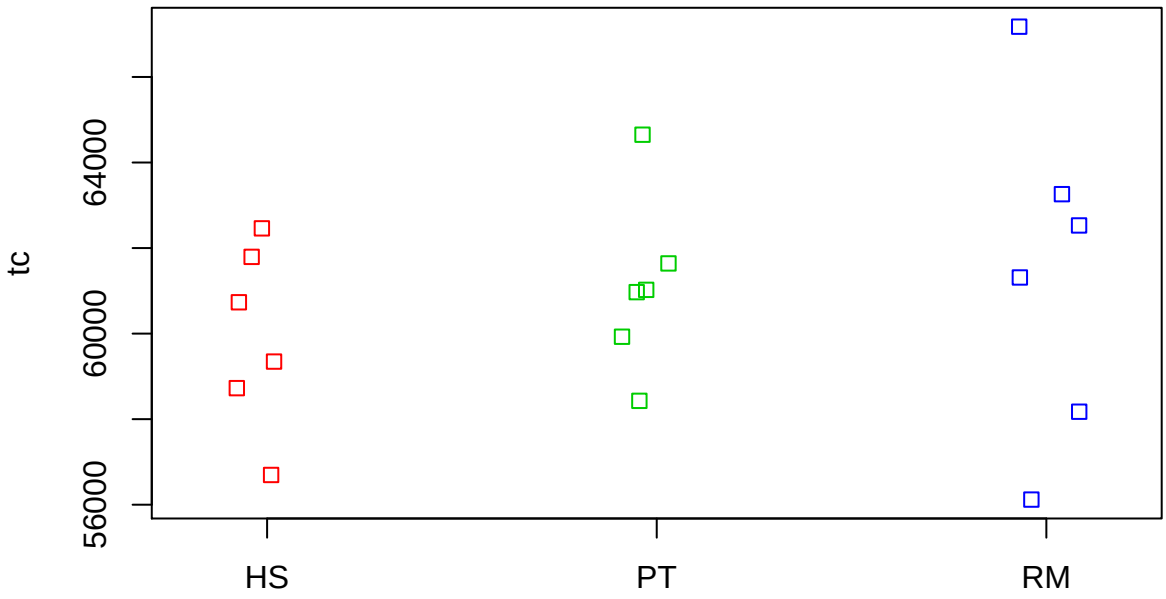



```
spnames = c('Human', 'Chimp', 'Macaque')
# plot against species
boxplot( tc ~ species2, ylab='Total lane counts', col=2:4, names=spnames)
```



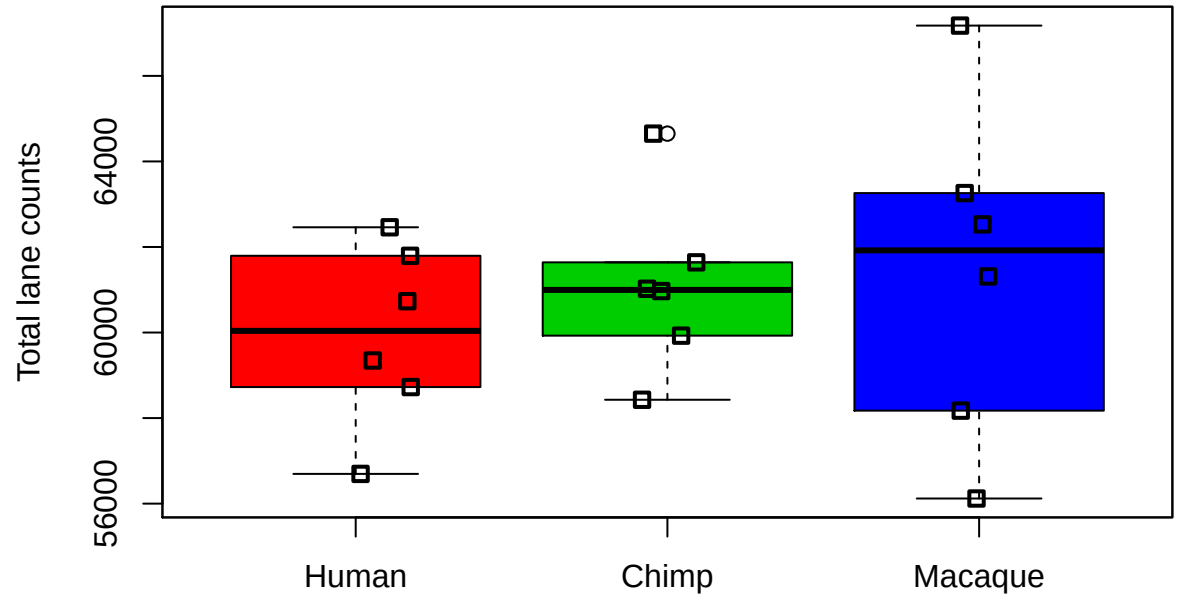
There is also this method:

```
stripchart( tc ~ species2 , method = "jitter", vertical = TRUE, col=2:4)
```



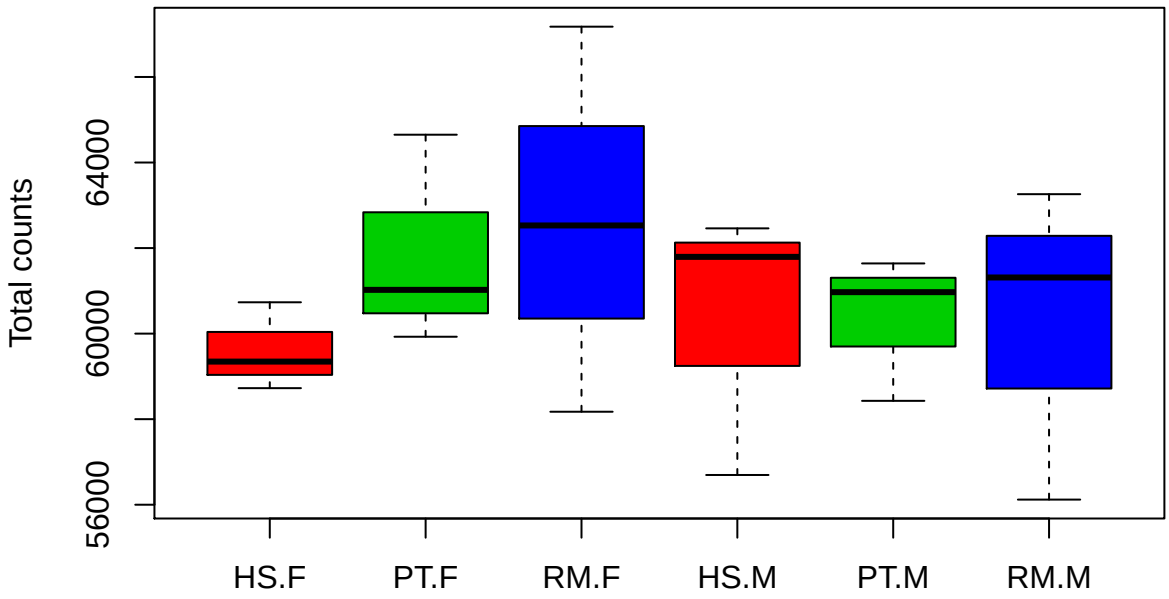
We could add this to the already existing plot using `par` and its argument `new`:

```
boxplot( tc ~ species2, ylab='Total lane counts', col=2:4, names=spnames)
par(new=TRUE)
stripchart( tc ~ species2 , method = "jitter", vertical = TRUE, col=1, lwd=2, xlab = "", ylab = "", xax
```



We could also plot the same, with interaction:

```
boxplot( tc ~ species2 + sex2,
         ylab='Total counts', col=2:4)
```



Are they significantly different? we could use the KW test again

```
kruskal.test( tc ~ species2 )$p.val
```

```
## [1] 0.6758306
```

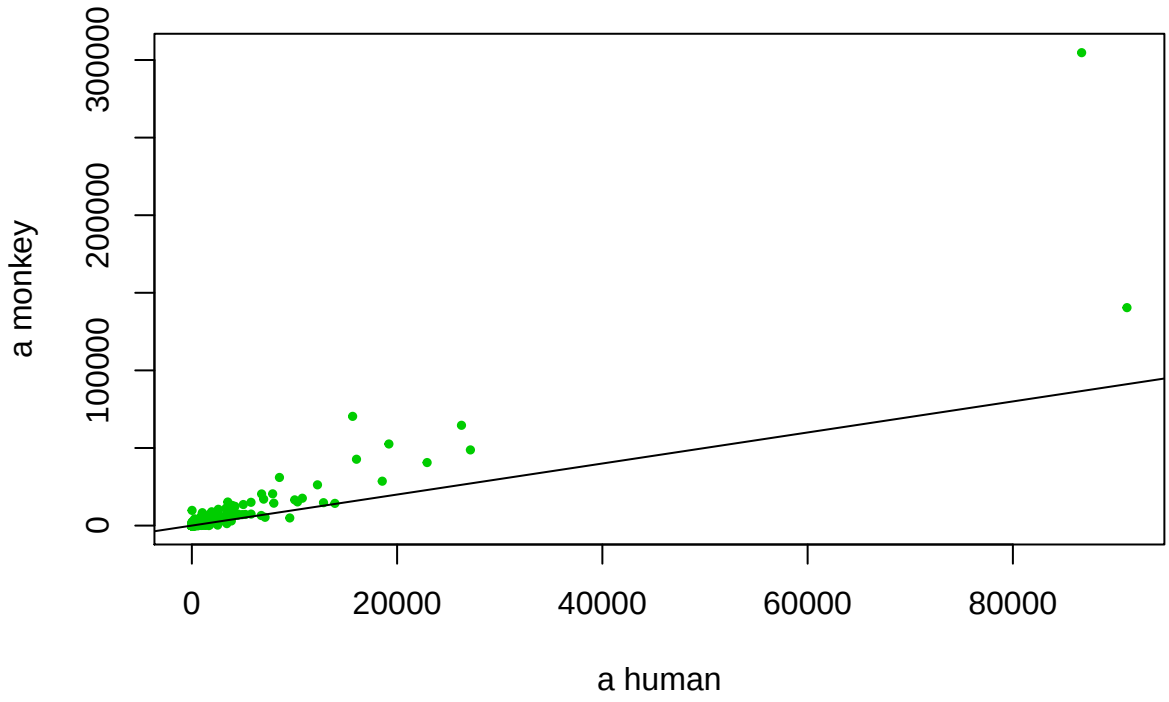
```
kruskal.test( tc ~ sex2 )$p.val
```

```
## [1] 0.7572777
```

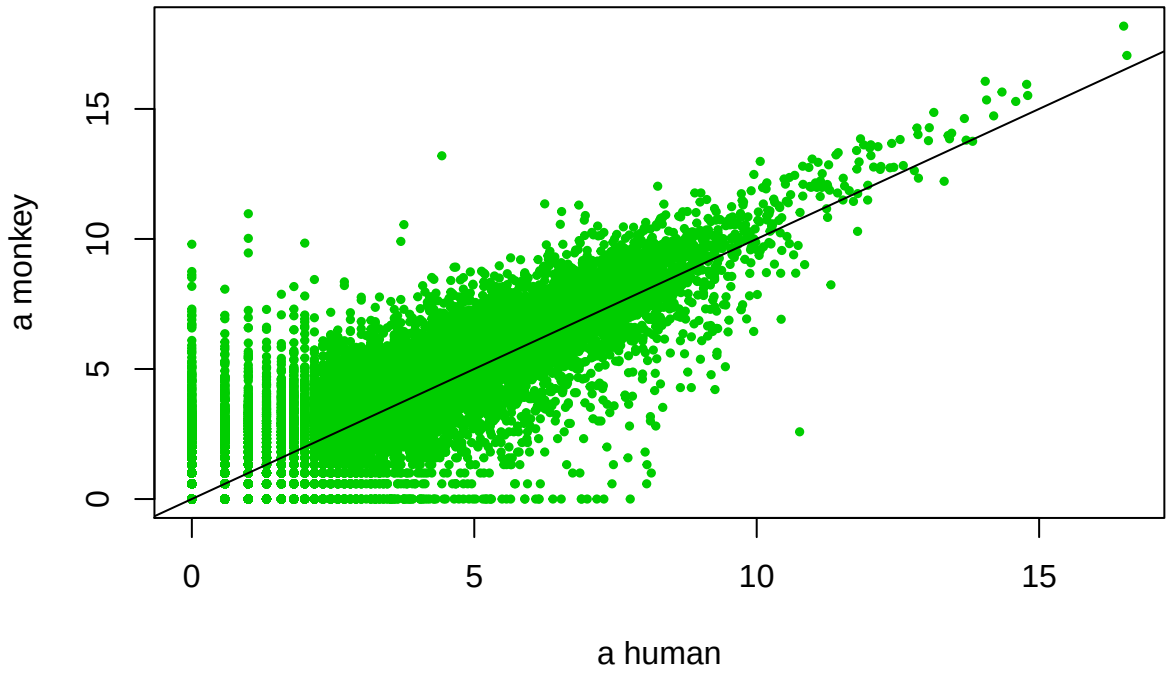
Additional exercise: comparing two individuals

The species or sex effect on column sums is not significant. How to interpret the fact that we observed a significant species difference earlier, but not now? We can guess that a few genes with very large expression levels were dominating the total signal. We can check this by comparing a human and a macaque:

```
# before log transformation
plot(mat[,"R1L1.HSM1"], mat[,"R1L3.RMM1"],
     xlab="a human", ylab="a monkey",
     col=3, pch=19, cex=0.5)
abline(0,1)
```




```
# after log transformation
plot(mat3[,"HSM1"], mat3[,"RMM1"],
     xlab="a human", ylab="a monkey",
     col=3, pch=19, cex=0.5)
abline(0,1)
```



Notice the general upward shift in the above graphs. Even after log transformation the trend remains. This is likely a technical effect, not biological, and we may consider removing it if we can.

Normalizing: dividing by column sums

How to normalize? We can imagine different ways:

- We could consider **centering** to the same mean by subtracting the mean of each column. But this is not realistic.
- Alternatively, we can **divide** by the column sums, which is more reasonable:

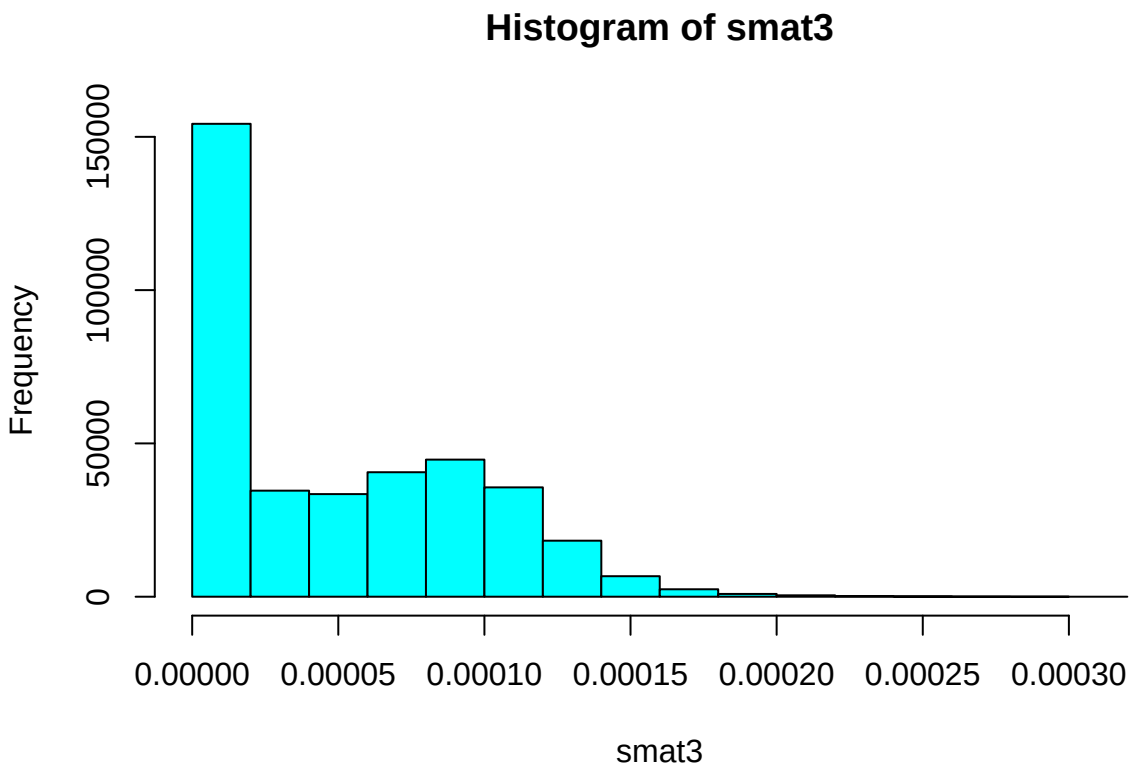
```
smat3 = apply( mat3, 2, function(x) { x / sum(x) })   # divide by the column sum
# this is the same:
smat3 = t(t( mat3 ) / colSums(mat3))
dim(smat3)
```

```
## [1] 20689    18
```

```
head(smat3)
```

```
##                         HSM1         PTF1         RMM1         HSF1
## ENSG00000000003 1.048155e-04 1.263713e-04 1.222691e-04 1.241985e-04
## ENSG00000000005 0.000000e+00 9.048123e-06 1.580785e-05 0.000000e+00
## ENSG00000000419 7.685955e-05 8.774033e-05 7.386100e-05 9.057116e-05
## ENSG00000000457 1.033245e-04 9.366231e-05 9.318440e-05 9.333662e-05
## ENSG00000000460 5.445741e-05 3.591528e-05 3.430185e-05 3.370065e-05
## ENSG00000000938 9.222717e-05 8.393275e-05 8.469138e-05 7.622348e-05
##                         PTM1         RMF1         RMF2         HSM2
## ENSG00000000003 1.246414e-04 1.279865e-04 1.351838e-04 1.236703e-04
## ENSG00000000005 0.000000e+00 0.000000e+00 1.005539e-05 9.365315e-06
## ENSG00000000419 7.999145e-05 8.169903e-05 8.954948e-05 8.520467e-05
## ENSG00000000457 1.153217e-04 8.808443e-05 8.852314e-05 8.371397e-05
## ENSG00000000460 4.554377e-05 2.534781e-05 2.724519e-05 3.474074e-05
## ENSG00000000938 1.077834e-04 8.362286e-05 6.715868e-05 7.191507e-05
##                         PTF2         RMM2         HSF2         PTM2
## ENSG00000000003 1.270262e-04 1.674764e-04 1.232594e-04 1.381697e-04
## ENSG00000000005 2.597313e-05 0.000000e+00 0.000000e+00 9.594314e-06
## ENSG00000000419 8.336936e-05 9.608581e-05 8.475576e-05 8.048080e-05
## ENSG00000000457 1.122811e-04 1.043835e-04 8.514265e-05 1.133824e-04
## ENSG00000000460 3.804994e-05 2.355546e-05 5.530743e-05 3.559022e-05
## ENSG00000000938 9.386468e-05 7.430398e-05 8.626806e-05 6.633449e-05
##                         RMM3         RMF3         HSM3         PTF3
## ENSG00000000003 1.380727e-04 1.177939e-04 1.213029e-04 1.268198e-04
## ENSG00000000005 0.000000e+00 1.488634e-05 0.000000e+00 9.761270e-06
## ENSG00000000419 9.008882e-05 7.268467e-05 8.091333e-05 8.453784e-05
## ENSG00000000457 1.038342e-04 8.793197e-05 9.062152e-05 9.137357e-05
## ENSG00000000460 2.585060e-05 3.848062e-05 5.256013e-05 3.015933e-05
## ENSG00000000938 8.496574e-05 7.156391e-05 8.198283e-05 8.819752e-05
##                         PTM3         HSF3
## ENSG00000000003 1.326105e-04 1.055353e-04
## ENSG00000000005 1.001165e-05 0.000000e+00
## ENSG00000000419 9.532822e-05 7.999018e-05
## ENSG00000000457 1.067705e-04 8.702799e-05
## ENSG00000000460 4.424170e-05 4.622514e-05
## ENSG00000000938 8.398156e-05 1.107801e-04
```

```
hist(smat3, col=5)
```

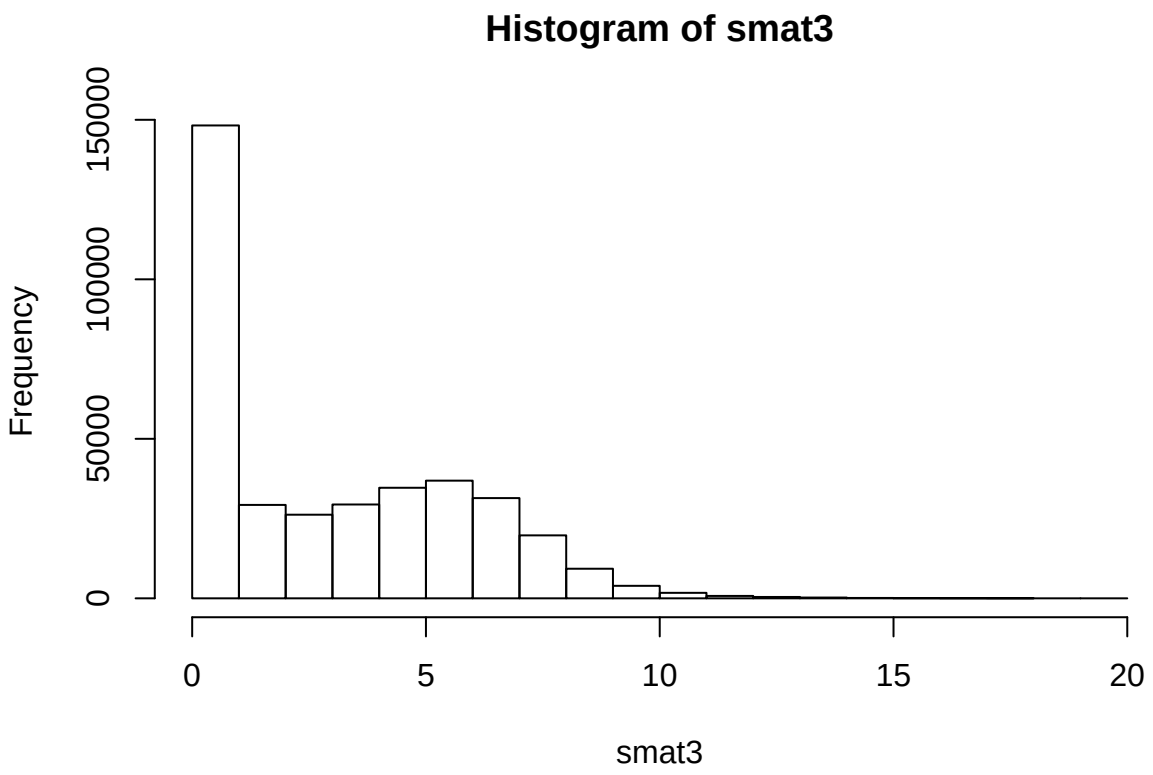


The data are now converted into very small numbers. We can multiply by a constant to shift the values to their original range:

```
smat3 = smat3 * mean(colSums(mat3))
head(smat3)
```

```
##                     HSM1      PTF1      RMM1     HSF1     PTM1     RMF1
## ENSG00000000003 6.376057 7.6873248 7.4377832 7.555147 7.582090 7.785578
## ENSG00000000005 0.000000 0.5504086 0.9616112 0.000000 0.000000 0.000000
## ENSG00000000419 4.675462 5.3373533 4.4930564 5.509556 4.865979 4.969853
## ENSG00000000457 6.285357 5.6975949 5.6685227 5.677783 7.015164 5.358285
## ENSG00000000460 3.312712 2.1847710 2.0866241 2.050052 2.770484 1.541939
## ENSG00000000938 5.610293 5.1057335 5.1518819 4.636769 6.556599 5.086882
##                      RMF2      HSM2     PTF2      RMM2     HSF2     PTM2
## ENSG00000000003 8.2233996 7.5230202 7.727164 10.187798 7.498022 8.405035
## ENSG00000000005 0.6116819 0.5697038 1.579978  0.000000 0.000000 0.583634
## ENSG00000000419 5.4474062 5.1831058 5.071462  5.845019 5.155798 4.895747
## ENSG00000000457 5.3849728 5.0924252 6.830199  6.349777 5.179333 6.897190
## ENSG00000000460 1.6573590 2.1133226 2.314625  1.432908 3.364420 2.164997
## ENSG00000000938 4.0853461 4.3746833 5.709905  4.520003 5.247793 4.035209
##                     RMM3      RMF3     HSM3      PTF3     PTM3     HSF3
## ENSG00000000003 8.399132 7.1655520 7.379008 7.7146035 8.066863 6.419842
## ENSG00000000005 0.000000 0.9055544 0.000000 0.5937902 0.609021 0.000000
## ENSG00000000419 5.480215 4.4214990 4.922059 5.1425418 5.798934 4.865902
## ENSG00000000457 6.316362 5.3490109 5.512620 5.5583680 6.494979 5.294021
## ENSG00000000460 1.572524 2.3408242 3.197298 1.8346298 2.691278 2.811933
## ENSG00000000938 5.168571 4.3533215 4.987118 5.3651645 5.108702 6.738891
```

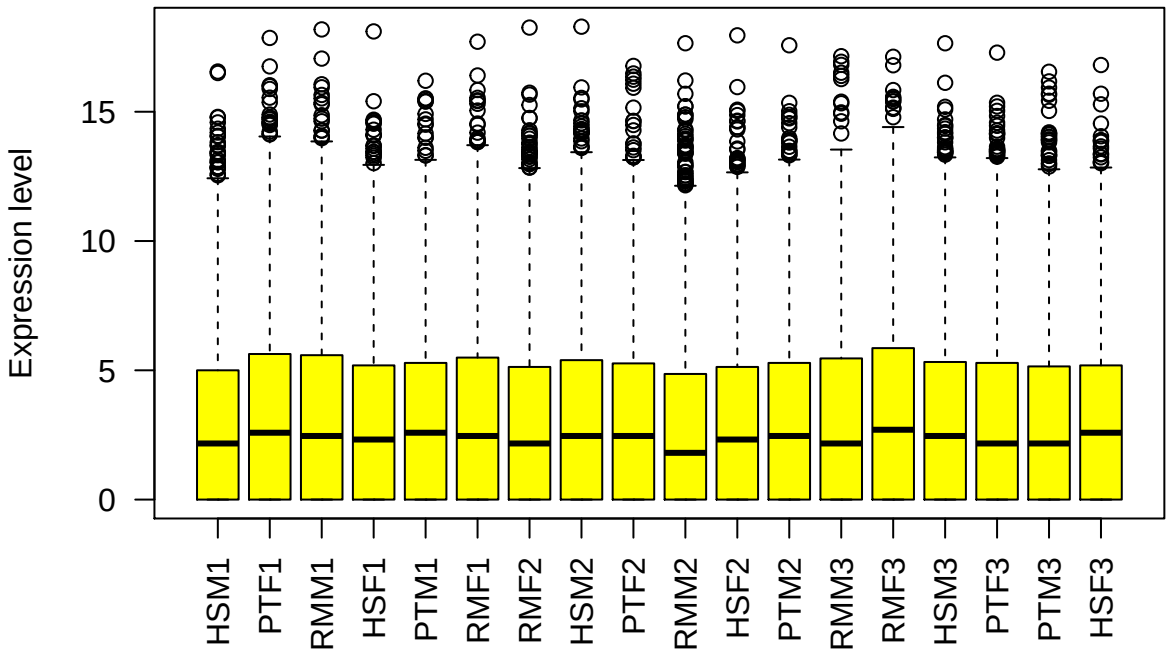
```
hist(smat3)
```



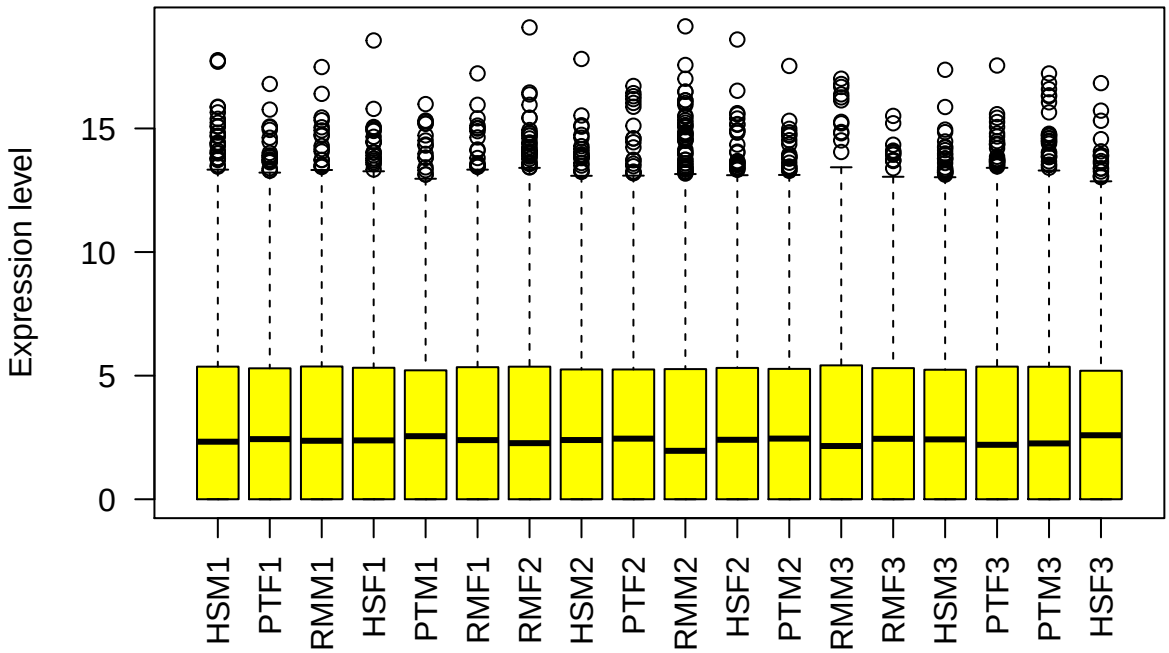

```
colSums(smat3)
```

```
##     HSM1     PTF1     RMM1     HSF1     PTM1     RMF1     RMF2     HSM2
## 60831.24 60831.24 60831.24 60831.24 60831.24 60831.24 60831.24 60831.24
##     PTF2     RMM2     HSF2     PTM2     RMM3     RMF3     HSM3     PTF3
## 60831.24 60831.24 60831.24 60831.24 60831.24 60831.24 60831.24 60831.24
##     PTM3     HSF3
## 60831.24 60831.24
```

```
par(mfrow=c(1,1))
boxplot( mat3, las=2, col=7, ylab='Expression level')
```

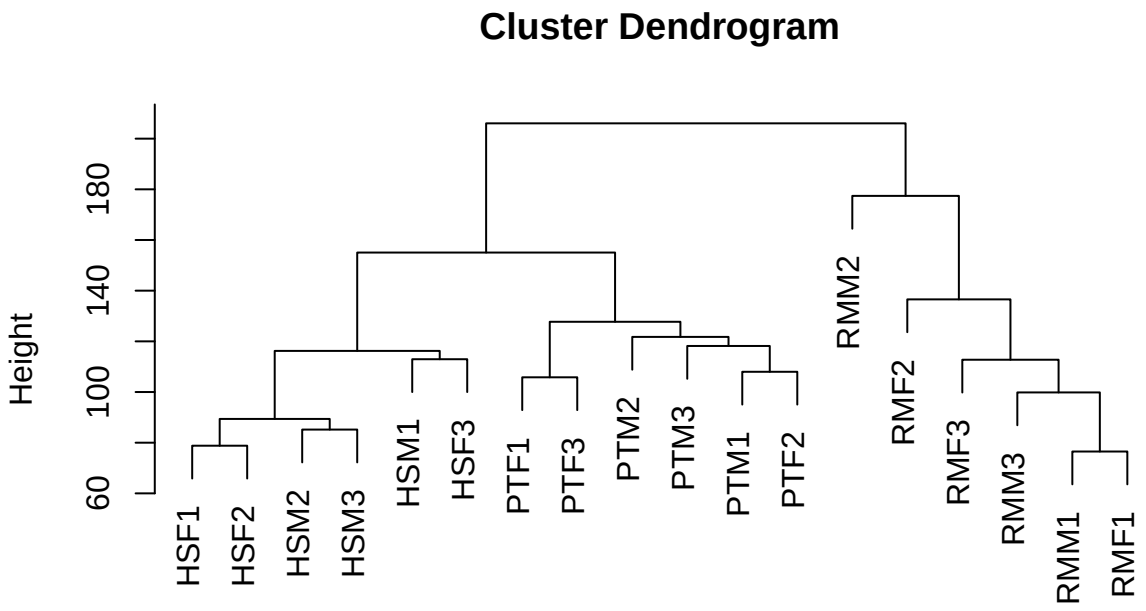



```
boxplot( smat3, las=2, col=7, ylab='Expression level')
```

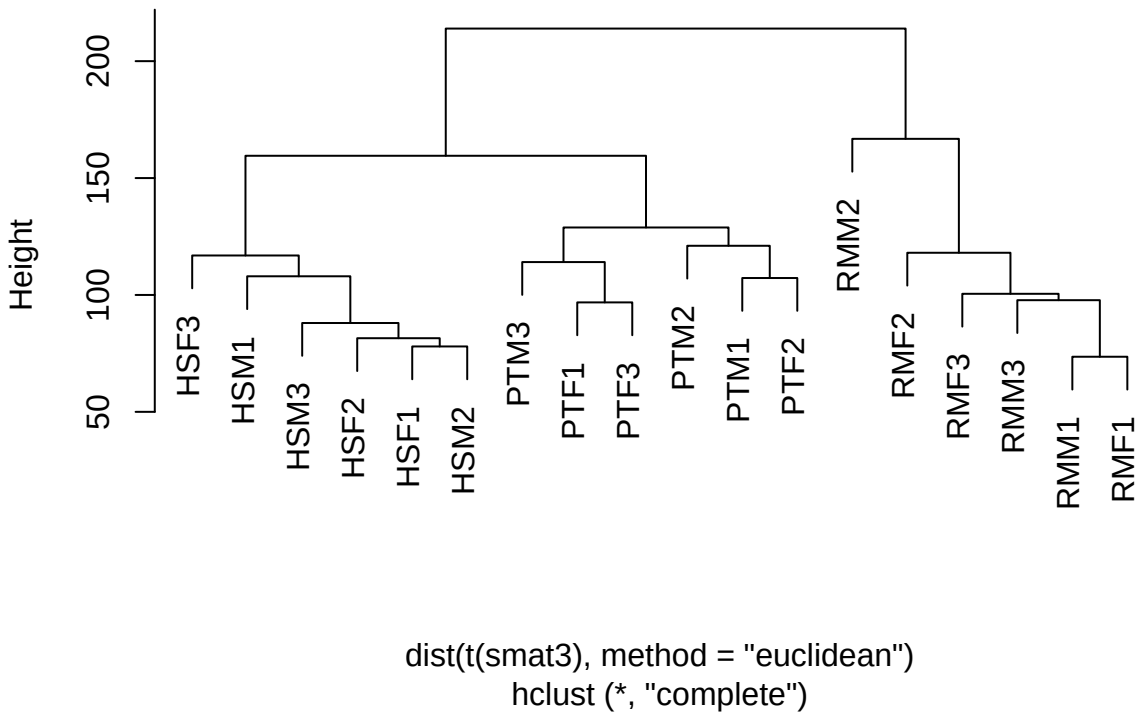


Let's also check if the trees have been affected:

```
plot( hclust( dist( t( mat3 ), method = "euclidean")  ) )
```




```
plot( hclust( dist( t( smat3 ), method = "euclidean")  ) )
```



It appears that the species clustering remains.

Next class we will learn quantile normalization.