BIO 754 - Lecture 08

13-04-2017

Contents

We now continue working on the liver transcriptome dataset. Let's load the earlier objects, and go through the answers of the homework questions:

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

Recalculate the summarized matrix of mat2, and create objects species2 and sex2 using colnames of mat2. Here we can use sapply and also convert unique(indv) from factor into character, which allows sapply to name the output columns automatically:

```
mat2 = sapply(as.character(unique(indv)), function(z) {
  rowMeans(mat[,indv == z])
})
species2 = factor(substr(colnames(mat2),1,2))
sex2 = factor(substr(colnames(mat2),3,3))
species2
```

```
##  [1] HS PT RM HS PT RM RM HS PT RM HS PT RM RM HS PT PT HS
## Levels: HS PT RM
```

```
sex2
```

```
##  [1] M F M F M F F M F M F M M F M F M F
## Levels: F M
```

```
mat3 = log2( mat2 + 1)
```

Testing differences using ANOVA and KW: The effect of non-expressed genes and outliers

We have not finished preprocessing yet. But as exercise, we will now run differential expression tests on the raw (unfiltered and non-log transformed) dataset, `mat2`. The purpose will be to study how both steps help in identifying differential expression (note that in the homework, we ran this on `mat3`).

We wish to test *each gene* for a species effect (just as we tested the species effect for the `totalcounts` variable), using ANOVA.

For this we can first try the first gene (first row):

```
x = mat2[1,]
x
```

```
##  HSM1  PTF1  RMM1  HSF1  PTM1  RMF1  RMF2  HSM2  PTF2  RMM2  HSF2  PTM2
##  60.5 287.0 212.0 164.5 204.5 255.5 232.0 210.5 214.5 674.0 150.0 342.5
##  RMM3  RMF3  HSM3  PTF3  PTM3  HSF3
## 352.5 240.0 179.5 193.0 214.0  84.0
```

```
summary(aov(x ~ species2))
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## species2     2 104230   52115   4.111 0.0377 *
## Residuals   15 190132   12675
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# the p-value
summary(aov(x ~ species2))[[1]][1,5]
```

```
## [1] 0.03769644
```

Now run the function on all genes. `apply` would work efficiently:

```
mat2.aov.pvals = apply(mat2, 1, function(x){
  summary( aov(x~species2) )[[1]][1,"Pr(>F)"]
})
head(mat2.aov.pvals)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##      0.03769644      0.16256982      0.53496527      0.00646531
## ENSG00000000460 ENSG00000000938
##      0.01195633      0.48057285
```

```
summary(mat2.aov.pvals)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.0000  0.0094  0.1012  0.2124  0.3895  1.0000    2803
```

What are the NAs?

```
head(mat2[is.na(mat2.aov.pvals),])
```

```
##                 HSM1 PTF1 RMM1 HSF1 PTM1 RMF1 RMF2 HSM2 PTF2 RMM2 HSF2
## ENSG00000006059    0    0    0    0    0    0    0    0    0    0    0
## ENSG00000007952    0    0    0    0    0    0    0    0    0    0    0
## ENSG00000008197    0    0    0    0    0    0    0    0    0    0    0
## ENSG00000011677    0    0    0    0    0    0    0    0    0    0    0
## ENSG00000016082    0    0    0    0    0    0    0    0    0    0    0
## ENSG00000016602    0    0    0    0    0    0    0    0    0    0    0
##                 PTM2 RMM3 RMF3 HSM3 PTF3 PTM3 HSF3
## ENSG00000006059    0    0    0    0    0    0    0
## ENSG00000007952    0    0    0    0    0    0    0
## ENSG00000008197    0    0    0    0    0    0    0
## ENSG00000011677    0    0    0    0    0    0    0
## ENSG00000016082    0    0    0    0    0    0    0
## ENSG00000016602    0    0    0    0    0    0    0
```

```
# are they all 0s?
sum(mat2[is.na(mat2.aov.pvals),])
```

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

Indeed, when there is no variance in the response variable, ANOVA and other tests cannot work (as they compare variance within and among groups):

```
summary( aov(rep(0, 18) ~ species2) )[[1]][1,"Pr(>F)"]
```

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

Reminder: `sapply` and `apply` loops create their own environments and the objects created within that environment are not available in the global environment. Thus, the object `x` still contains the first gene.

```
x
```

```
##  HSM1  PTF1  RMM1  HSF1  PTM1  RMF1  RMF2  HSM2  PTF2  RMM2  HSF2  PTM2
##  60.5 287.0 212.0 164.5 204.5 255.5 232.0 210.5 214.5 674.0 150.0 342.5
##  RMM3  RMF3  HSM3  PTF3  PTM3  HSF3
## 352.5 240.0 179.5 193.0 214.0  84.0
```

Now, we will apply the Kruskal-Wallis test (we need to have a `factor` class object in the formula used as argument inside the `kruskal.test` function):

```
mat2.krus.pvals = apply(mat2, 1, function(x){
  kruskal.test(x~as.factor(species2))$p.val
})
head(mat2.krus.pvals)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##     0.003911401     0.104345779     0.567081945     0.023702708
## ENSG00000000460 ENSG00000000938
##     0.026640977     0.410958796
```

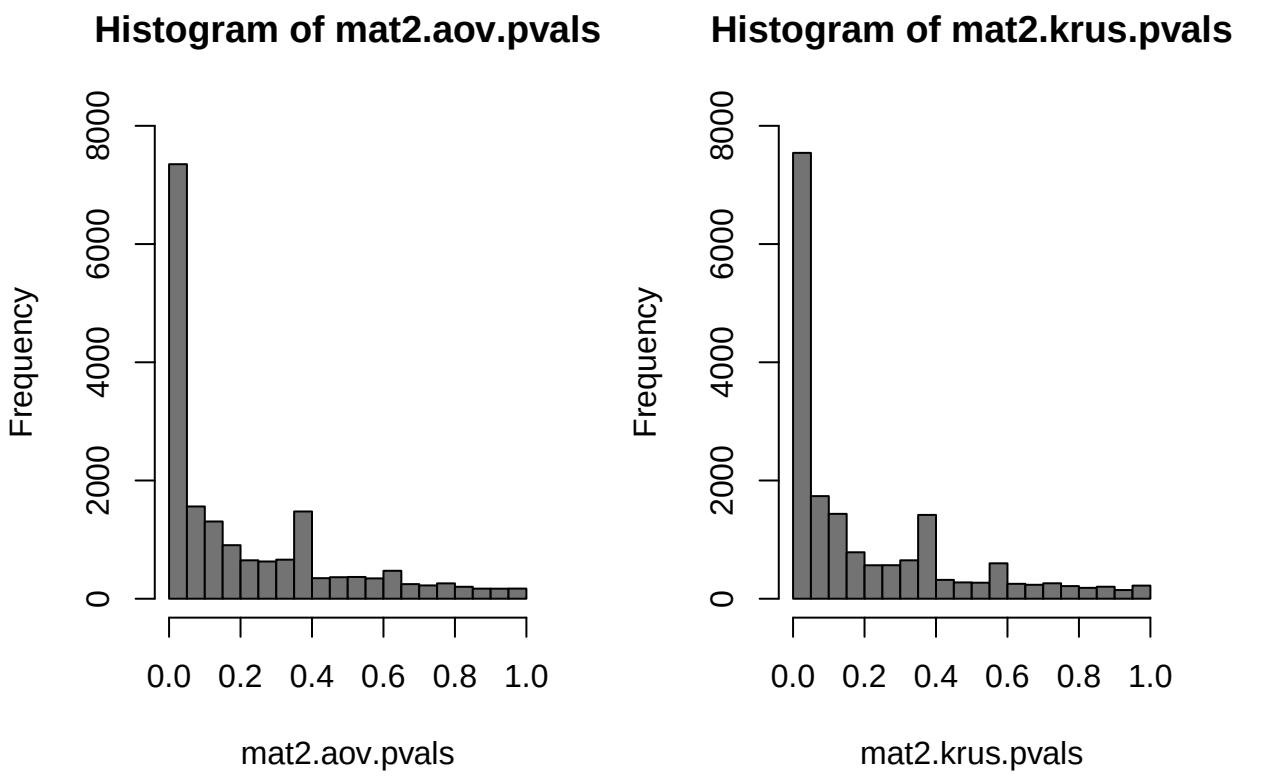
```
summary(mat2.krus.pvals)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.0003  0.0118  0.0883  0.2054  0.3655  1.0000    2803
```

```
# the same NAs
```

How do the two tests compare? We expect ANOVA to be more sensitive, and have lower p-values? Plot the histograms of `mat2.aov.pvals` and `mat2.krus.pvals` side by side and compare distribution of p-values.

```
par(mfrow=c(1,2))
# the hist() function automatically removes the NAs
hist(mat2.aov.pvals,col="gray45", ylim=c(0,8000))
hist(mat2.krus.pvals,col="gray45", ylim=c(0,8000))
```



Dealing with `NA`s: `na.omit`, `is.na`, `complete.cases`

We wish to calculate the median p-values for both `mat2.aov.pvals` and `mat2.krus.pvals`. But we have to remove the `NA`s. This is because `NA` is a nullifying element in R:

```
sum(c(1, NA, 10))
```

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

```r
5 * NA
```

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

```r
median(mat2.aov.pvals)
```

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

```r
# but we can use na.omit
na.omit(c(1, NA, 10))
```

```
## [1]  1 10
## attr(,"na.action")
## [1] 2
## attr(,"class")
## [1] "omit"
```

```r
median(na.omit( mat2.aov.pvals ) )
```

```
## [1] 0.1012183
```

```r
# or
median(mat2.aov.pvals, na.rm = T)
```

```
## [1] 0.1012183
```

Two other useful functions are `is.na` and `complete.cases`, which execute the opposite functions:

```r
is.na(c(1, NA, 10))
```

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

```r
complete.cases(c(1, NA, 10))
```

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

```r
# thus, this would also have worked
median( mat2.aov.pvals [complete.cases(mat2.aov.pvals)] )
```

```
## [1] 0.1012183
```

Coming back to our main question, which test seems to be more conservative, given the median?

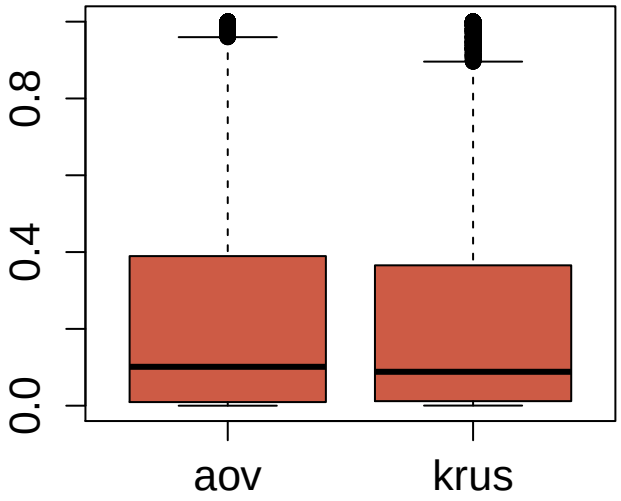
```r
median(mat2.aov.pvals, na.rm = T)
```

```
## [1] 0.1012183
```

```r
median(mat2.krus.pvals, na.rm = T)
```

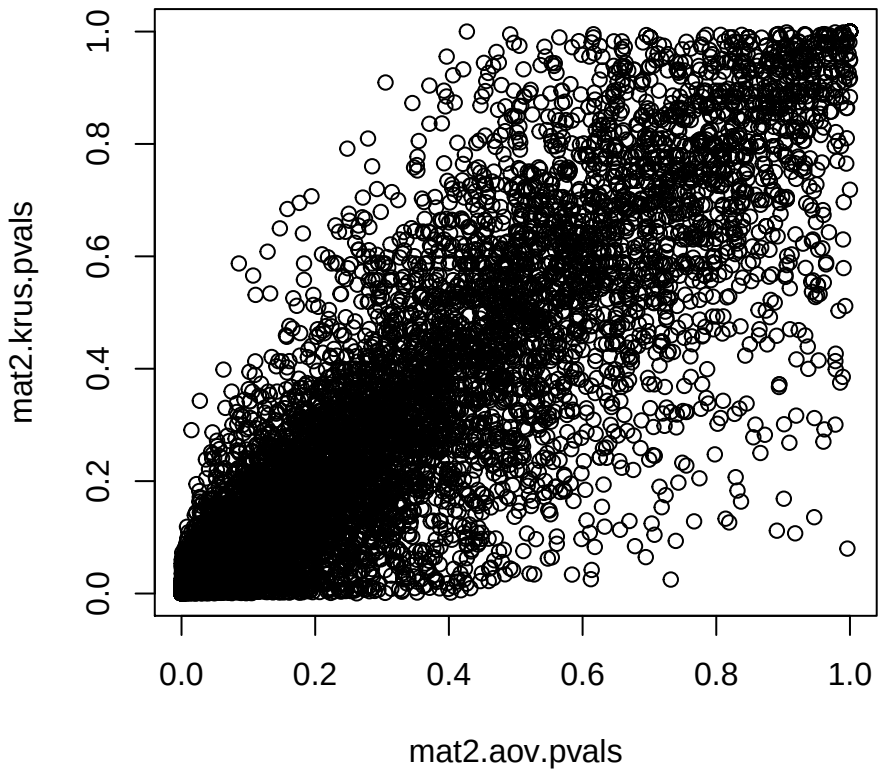
```
## [1] 0.08825783
```

```r
# a boxplot:
boxplot(mat2.aov.pvals, mat2.krus.pvals, col="coral3",
        names=c("aov","krus"), cex.axis=1.3)
```



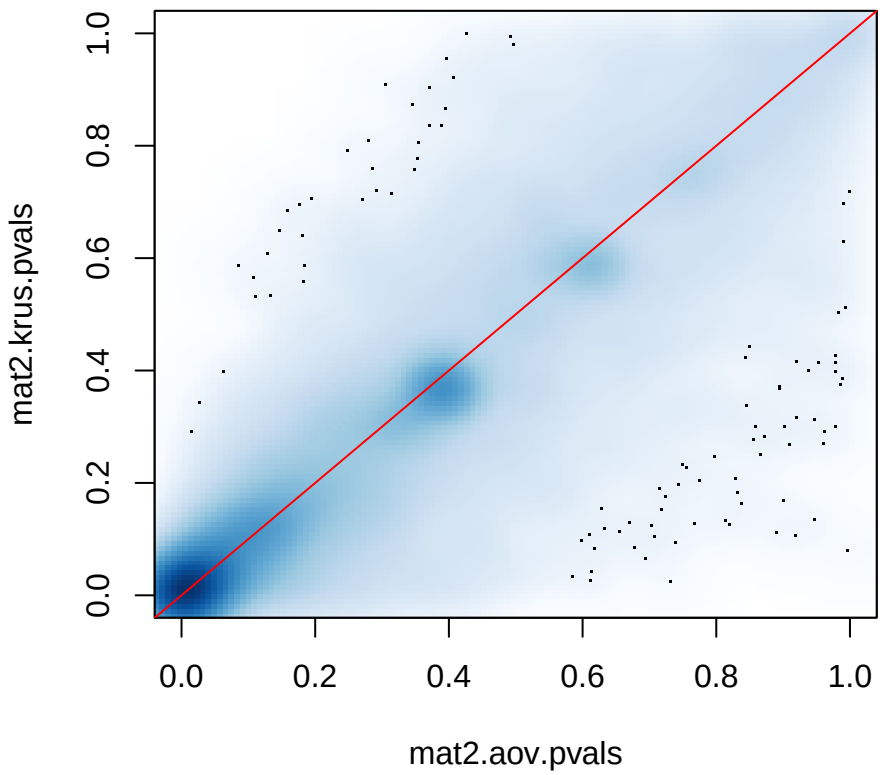
ANOVA seems to have slightly *higher* overall p-values. This suggests it is less sensitive (more conservative), opposite to what we expected. To better understand the situation, we can plot one against the other

```r
plot(mat2.aov.pvals, mat2.krus.pvals)
```



The plot reflects the close correlation, but is difficult to interpret as many dots are plotted one above the other. An alternative is using the `smoothScatter` function:

```
smoothScatter(mat2.aov.pvals, mat2.krus.pvals)
abline(0, 1, col=2)
```



Although most genes' p-values are around the diagonal, there are some exceptions. We already expect ANOVA to be more sensitive (lower p-values) than KW, as KW involves ranking the data and loses information.

Exercise: The cause of inconsistency between ANOVA and KW

What type of genes show high p-value for ANOVA, but appear significant in KW?

We could address this question by visually studying some the cases where there is inconsistency.

For instance, we can study cases where ANOVA `p>0.05`and KW `p<0.01`. Calculate how many such genes there are:

```
sum(mat2.aov.pvals > 0.05 & mat2.krus.pvals < 0.01, na.rm = T)
```

```
## [1] 189
```

Why do they behave differently? Let's check the first case:

```
i = which(mat2.aov.pvals > 0.05 & mat2.krus.pvals < 0.01)[1]
i
```

```
## ENSG00000003400
##              39
```

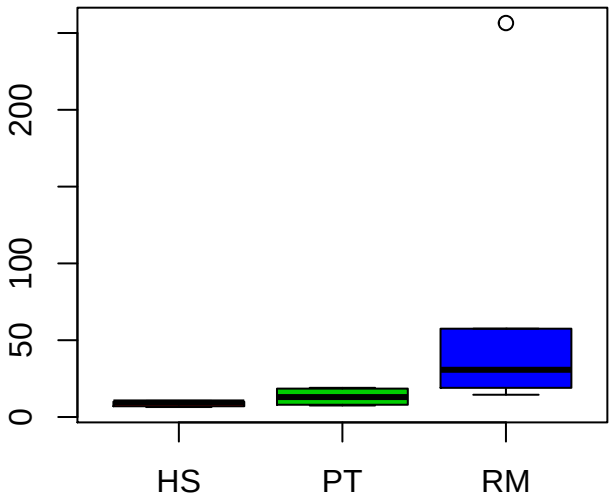
```
mat2.aov.pvals[i]
```

```
## ENSG00000003400
##       0.1401606
```

```
mat2.krus.pvals[i]
```

```
## ENSG00000003400
##      0.00343413
```

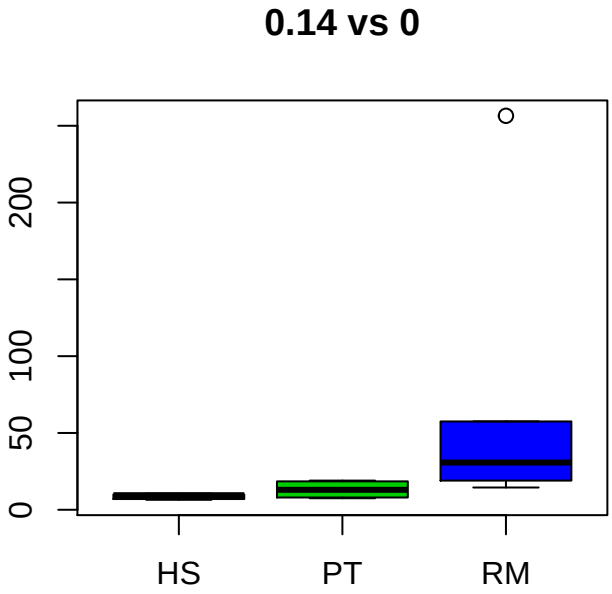
```
boxplot(mat2[i,] ~  as.factor(species2), col=2:4)
```



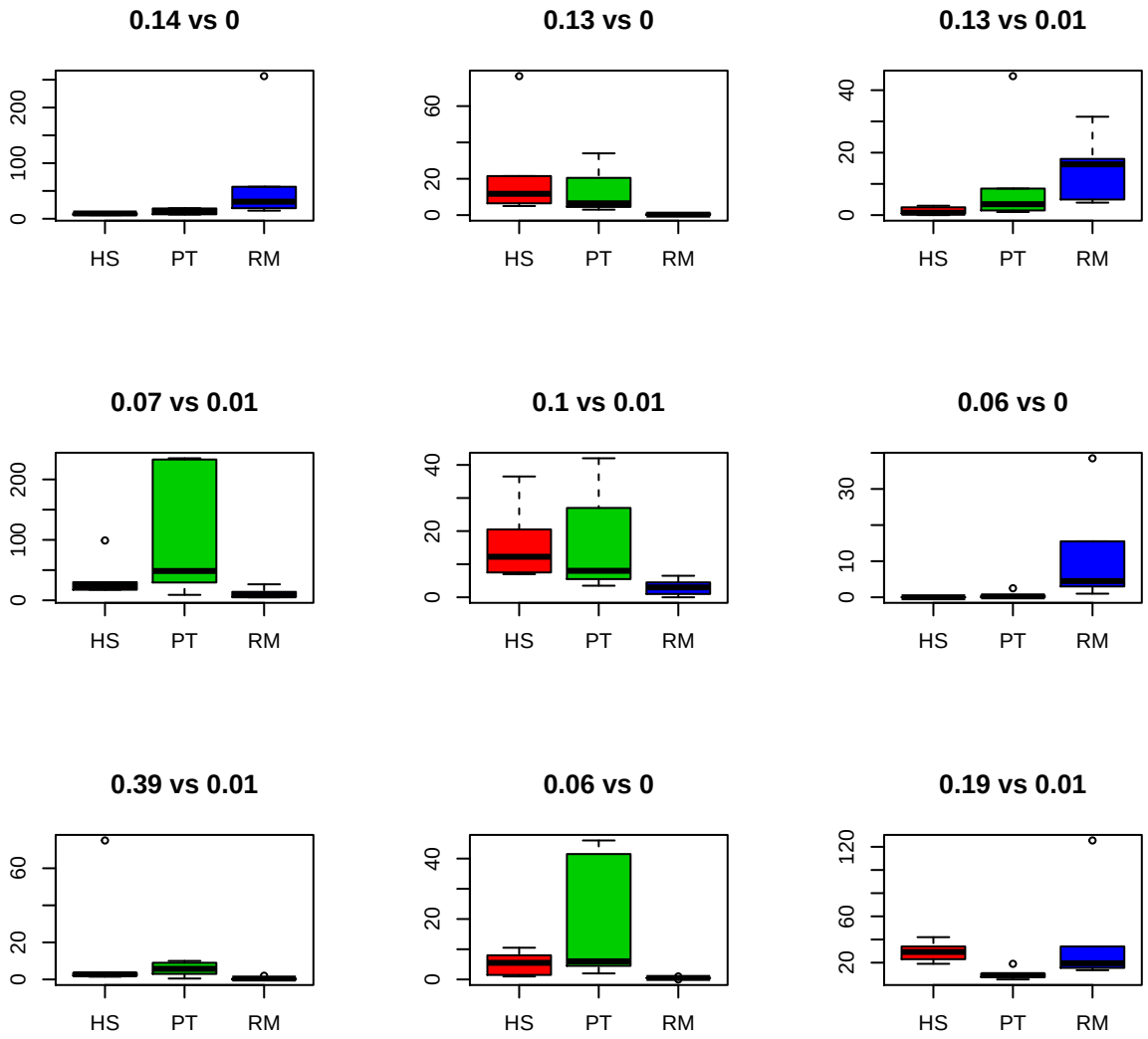

```
pvalcomp = paste(round(mat2.aov.pvals[i], 2), "vs", round(mat2.krus.pvals[i], 2))
pvalcomp
```

```
## [1] "0.14 vs 0"
```

```
boxplot(mat2[i,] ~  as.factor(species2), col=2:4,
        main=pvalcomp)
```




```
par(mfrow=c(3,3))
for (j in 1:9) {
    i = which(mat2.aov.pvals > 0.05 & mat2.krus.pvals < 0.01)[j]
  pvalcomp = paste(round(mat2.aov.pvals[i], 2), "vs", round(mat2.krus.pvals[i], 2))
  boxplot(mat2[i,] ~  as.factor(species2),
          col=2:4, main=pvalcomp)
}
```



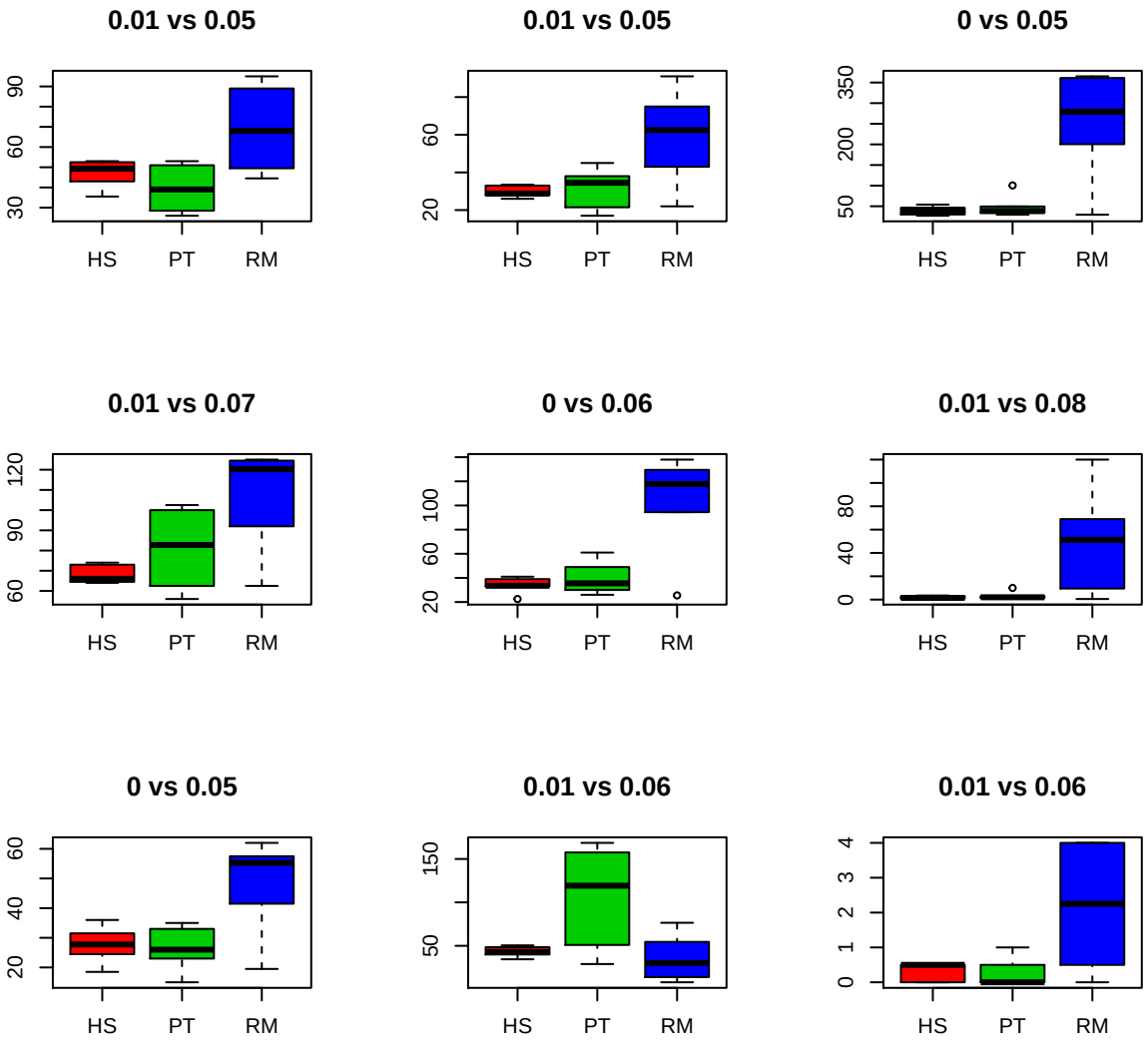
It seems as if **outliers** are common among these genes. But perhaps, this is a usual pattern among all DE genes? To test this idea, we can repeat the analysis the other way round: genes which are significant for ANOVA but *not* KW:

```
sum(mat2.aov.pvals < 0.01 & mat2.krus.pvals > 0.05, na.rm = T)
```

```
## [1] 49
```

```
par(mfrow=c(3,3))

for (j in 1:9) {
    i = which(mat2.aov.pvals < 0.01 & mat2.krus.pvals > 0.05)[j]
  pvalcomp = paste(round(mat2.aov.pvals[i], 2), "vs", round(mat2.krus.pvals[i], 2))
  boxplot(mat2[i,] ~  as.factor(species2),
          col=2:4, main=pvalcomp)
}
```



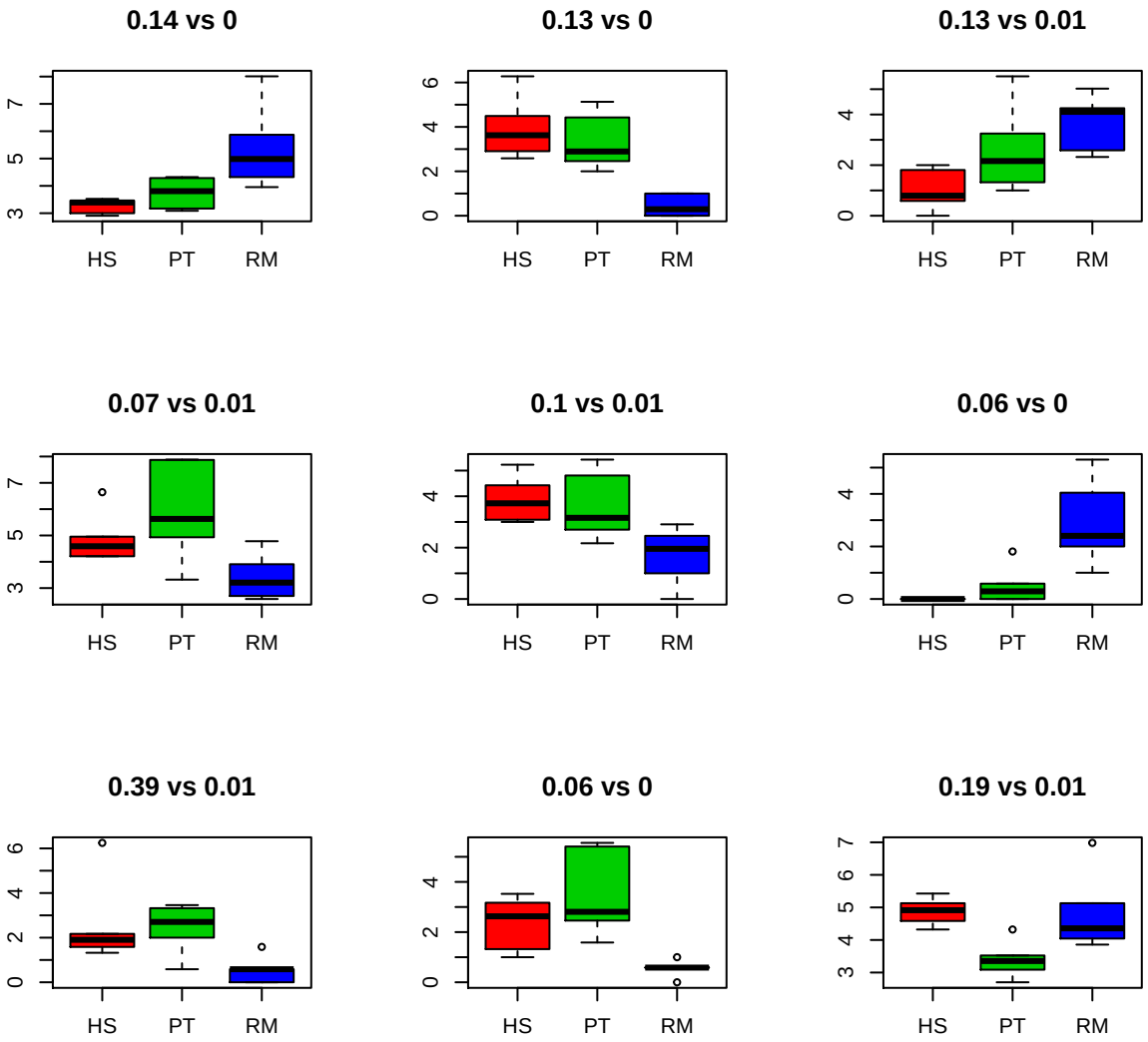
Here we do *not* notice a comparable presence of outliers when we condition upon significance of ANOVA. Of course, we only studied this superficially, but we could have calculated variance in all these cases, and could formally test the idea.

The result underscores an important point: When the data is **skewed** and has **outliers** (when the assumption of normality is strongly violated) ANOVA underperforms relative to non-parametric tests. Specifically, ANOVA becomes insensitive to possible differences among groups. This is because it calculates a large within group variance estimate (due to a single outlier), and this renders differences *among* groups insignificant.

Thus, when the data is skewed, the KW test can be more **powerful**. If we log-transform and down-weight the outliers among groups, we improve power for parametric tests. By log-transforming, the same 9 genes' expression appears as follows:

```
par(mfrow=c(3,3))

for (j in 1:9) {
    i = which(mat2.aov.pvals > 0.05 & mat2.krus.pvals < 0.01)[j]
  pvalcomp = paste(round(mat2.aov.pvals[i], 2), "vs", round(mat2.krus.pvals[i], 2))
  boxplot(mat3[i,] ~  as.factor(species2),
          col=2:4, main=pvalcomp)
}
```



We do not show it here, but running ANOVA and KW on the log-transformed data (`mat3`) produces much fewer inconsistent results.

Non-expressed genes in the transcriptome dataset

It seemed if as neither ANOVA nor KW could calculate a p-value for genes with all values `0`s. Let us study this further. First, calculate how many genes are not expressed (have value `0`) at all across the 18 samples in `mat2`:

```
# how many genes' sum is 0?
sum( rowSums(mat2) == 0)
```

```
## [1] 2803
```

```
# how many genes contain 18 0s?
sum( rowSums(mat2 == 0) == 18 )
```

```
## [1] 2803
```

We can also find how many genes are *not* expressed in any of the individuals in each species, or in each sex, separately.

First, let's do this for human:

```
x = "HS"
head(mat2[,species2==x])
```

```
##                 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
```

```
head(rowSums(mat2[,species2==x]))
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##           849.0             0.5           188.5           255.5
## ENSG00000000460 ENSG00000000938
##            37.0           250.0
```

```
head(rowSums(mat2[,species2==x])==0)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##           FALSE           FALSE           FALSE           FALSE
## ENSG00000000460 ENSG00000000938
##           FALSE           FALSE
```

```
sum( rowSums(mat2[,species2==x])==0 )
```

```
## [1] 4314
```

This works. So now we can write the loop:

```
# again, use as.character to have the vectors named automatically
sapply(as.character(unique(species2)), function(x) {
  print(x)  # to help understand what x is each time
  sum(rowSums(mat2[,species2==x])==0)
})
```

```
## [1] "HS"
## [1] "PT"
## [1] "RM"
```

```
##   HS   PT   RM
## 4314 4248 4450
```

```
sapply(as.character(unique(sex2)), function(x) {
  sum( rowSums(mat2[,sex2==x])==0)
})
```

```
##    M    F
## 3481 3527
```

Inconsistency among replicates

We may also be interested in inconsistency among replicates in `mat`, where a gene appears expressed (value `>0`) in one technical replicate but not expressed (value `0`) in the other replicate. How many such genes are there, in each replicate?

Again, first let's write to code for one case:

```
x = as.character(levels(indv))[1]
x
```

```
## [1] "HSF1"
```

```
# choose columns for individual x
submat = mat[,grep(x,colnames(mat))]
# or
submat = mat[,indv == x]
head( submat, 30)
```

```
##                 R1L4.HSF1 R4L2.HSF1
## ENSG00000000003       172       157
## ENSG00000000005         0         0
## ENSG00000000419        36        45
## ENSG00000000457        41        50
## ENSG00000000460         3         3
## ENSG00000000938        23        21
## ENSG00000000971      2262      2503
## ENSG00000001036       155       142
## ENSG00000001084       323       307
## ENSG00000001167        19        17
## ENSG00000001460         3         0
## ENSG00000001461        25        24
## ENSG00000001497        59        58
## ENSG00000001561        22        26
## ENSG00000001617        30        34
## ENSG00000001626         9         3
## ENSG00000001629        77        72
## ENSG00000001630        36        41
## ENSG00000001631        47        41
## ENSG00000002016         3         3
## ENSG00000002079         1         0
## ENSG00000002330        36        31
## ENSG00000002549        87        69
## ENSG00000002586       180       191
## ENSG00000002587         1         2
```

```
## ENSG00000002726         0         1
## ENSG00000002745         1         0
## ENSG00000002746         0         0
## ENSG00000002822        24        14
## ENSG00000002834        50        40
```

```
# which ones are 0?
head( submat == 0, 30)
```

```
##                 R1L4.HSF1 R4L2.HSF1
## ENSG00000000003     FALSE     FALSE
## ENSG00000000005      TRUE      TRUE
## ENSG00000000419     FALSE     FALSE
## ENSG00000000457     FALSE     FALSE
## ENSG00000000460     FALSE     FALSE
## ENSG00000000938     FALSE     FALSE
## ENSG00000000971     FALSE     FALSE
## ENSG00000001036     FALSE     FALSE
## ENSG00000001084     FALSE     FALSE
## ENSG00000001167     FALSE     FALSE
## ENSG00000001460     FALSE      TRUE
## ENSG00000001461     FALSE     FALSE
## ENSG00000001497     FALSE     FALSE
## ENSG00000001561     FALSE     FALSE
## ENSG00000001617     FALSE     FALSE
## ENSG00000001626     FALSE     FALSE
## ENSG00000001629     FALSE     FALSE
## ENSG00000001630     FALSE     FALSE
## ENSG00000001631     FALSE     FALSE
## ENSG00000002016     FALSE     FALSE
## ENSG00000002079     FALSE      TRUE
## ENSG00000002330     FALSE     FALSE
## ENSG00000002549     FALSE     FALSE
## ENSG00000002586     FALSE     FALSE
## ENSG00000002587     FALSE     FALSE
## ENSG00000002726      TRUE     FALSE
## ENSG00000002745     FALSE      TRUE
## ENSG00000002746      TRUE      TRUE
## ENSG00000002822     FALSE     FALSE
## ENSG00000002834     FALSE     FALSE
```

```
# how many 0s per row?
head( rowSums(submat == 0), 30)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##               0               2               0               0
## ENSG00000000460 ENSG00000000938 ENSG00000000971 ENSG00000001036
##               0               0               0               0
## ENSG00000001084 ENSG00000001167 ENSG00000001460 ENSG00000001461
##               0               0               1               0
## ENSG00000001497 ENSG00000001561 ENSG00000001617 ENSG00000001626
##               0               0               0               0
## ENSG00000001629 ENSG00000001630 ENSG00000001631 ENSG00000002016
```

```
##               0               0               0               0
## ENSG00000002079 ENSG00000002330 ENSG00000002549 ENSG00000002586
##               1               0               0               0
## ENSG00000002587 ENSG00000002726 ENSG00000002745 ENSG00000002746
##               0               1               1               2
## ENSG00000002822 ENSG00000002834
##               0               0
```

```
# the summary
table( rowSums(submat == 0) )
```

```
##
##     0     1     2
## 12177  1789  6723
```

```
# 1s indicate inconsistency: one replicate is 0, the other, non-0
sum( rowSums(submat == 0) == 1 )
```

```
## [1] 1789
```

Now we can write the full loop, calculating inconsistent cases for each individual:

```
rep_incons = sapply( as.character(unique(indv)), function(x) {
  submat = mat[,indv == x]
  sum( rowSums( submat == 0 ) == 1 )
})
summary(rep_incons)/nrow(mat)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## 0.07859249 0.08560104 0.08671275 0.08690609 0.08835613 0.09517135
```

This result suggests common (>5%) inconsistency between replicates, which could be worrying. But is the inconsistency large *in magnitude*? You can check this using two approaches. One is to calculate a correlation matrix among replicates. The other is to check the expression values of inconsistent genes - the cases where a gene is not expressed (value 0) for one replicate while it is expressed in the other replicate. If the expression level of the latter is high, this could indicate significant technical noise in the data.

For each individual, let us calculate maximum expression values for all inconsistent genes (when the other replicate has value 0). We will then print out the median and the range (minimum and maximum) for these maximum values.

Again, let's first write the code for one individual:

```
x = as.character(levels(indv))[1]
submat = mat[,indv == x]
# now we need the inconsistent genes
submat2 = submat[rowSums(submat == 0) == 1, ]
# check
head( submat2 )
```

```
##                 R1L4.HSF1 R4L2.HSF1
## ENSG00000001460         3         0
```

```
## ENSG00000002079         1         0
## ENSG00000002726         0         1
## ENSG00000002745         1         0
## ENSG00000003987         0         1
## ENSG00000004939         2         0
```

```
dim( submat2 )
```

```
## [1] 1789    2
```

```
c(median(submat2[submat2 > 0]), range(submat2[submat2 > 0]))
```

```
## [1]  1  1 11
```

```
# for all individuals
x2 = sapply(as.character(levels(indv)), function(x){
  submat = mat[,indv == x]
  submat2 = submat[rowSums(submat == 0) == 1, ]
  c(median(submat2[submat2 > 0]), range(submat2[submat2 > 0]))
})
rownames(x2) = c('median', 'min', 'max')
x2
```

```
##        HSF1 HSF2 HSF3 HSM1 HSM2 HSM3 PTF1 PTF2 PTF3 PTM1 PTM2 PTM3 RMF1
## median    1    1    1    1    1    1    1    1    1    1    1    1    1
## min       1    1    1    1    1    1    1    1    1    1    1    1    1
## max      11   10   12    9   11    8   10   10    8    9   11    9   14
##        RMF2 RMF3 RMM1 RMM2 RMM3
## median    1    1    1    1    1
## min       1    1    1    1    1
## max      11    9   10   10    9
```

So most of the time, inconsistency is caused by 0s and 1s. Thus, even when the replicates are inconsistent, the degree of inconsistency is small.

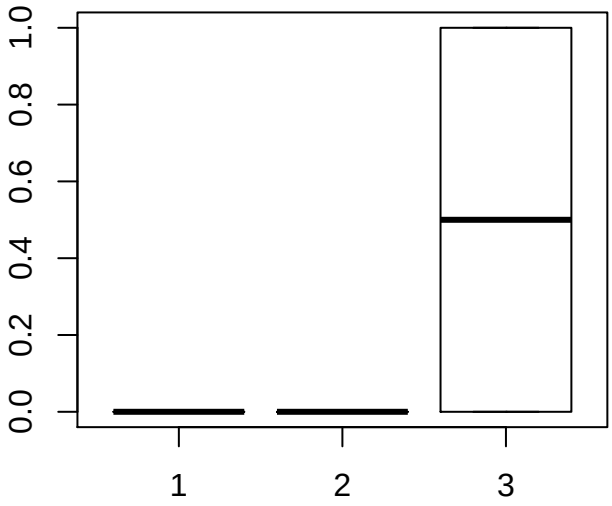
Filtering non-expressed genes

We are not interested in genes not expressed (value 0) in any sample. However, some genes might be expressed in some samples but not in other ones. It will be helpful to filter such **unreliable** genes in order to:

- avoid spurious and non-reproducible results (false positives),
- avoid penalizing our results for multiple testing, and thus, improve power.

Check this example for spurious DE - low expressed genes may appear significantly different, but may not be reproducible:

```
x = rep(0, 6)
y = rep(0, 6)
z = c(0, 0, 0, 1, 1, 1)
boxplot(x, y, z)
```




```
summary(aov(c(x, y, z) ~ c(rep("a", 6), rep("b", 6), rep("c", 6))))
```

```
##                                          Df Sum Sq Mean Sq F value Pr(>F)
## c(rep("a", 6), rep("b", 6), rep("c", 6))  2    1.0     0.5       5 0.0217
## Residuals                                15    1.5     0.1
##
## c(rep("a", 6), rep("b", 6), rep("c", 6)) *
## Residuals
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Exercise: A strict filter

First, we will remove the genes which are not expressed in *at least* one sample using `mat2` object.

Let's write a piece of code for this, on the first gene. It should return `T` or `F` depending on there being no `0`s or not.

```
g = 1
mat2[1, ]
```

```
##  HSM1  PTF1  RMM1  HSF1  PTM1  RMF1  RMF2  HSM2  PTF2  RMM2  HSF2  PTM2
##  60.5 287.0 212.0 164.5 204.5 255.5 232.0 210.5 214.5 674.0 150.0 342.5
##  RMM3  RMF3  HSM3  PTF3  PTM3  HSF3
## 352.5 240.0 179.5 193.0 214.0  84.0
```

```
# are the values 0?
mat2[g, ] == 0
```

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

```
# how many 0s?
sum(mat2[g, ] == 0)
```

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

```
# all samples are non-0 (expressed)?
sum(mat2[g, ] == 0) == 0
```

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

```
# the second gene
g = 2
mat2[g, ]
```

```
## HSM1 PTF1 RMM1 HSF1 PTM1 RMF1 RMF2 HSM2 PTF2 RMM2 HSF2 PTM2 RMM3 RMF3 HSM3
##  0.0  0.5  1.0  0.0  0.0  0.0  0.5  0.5  2.0  0.0  0.0  0.5  0.0  1.0  0.0
## PTF3 PTM3 HSF3
##  0.5  0.5  0.0
```

```
mat2[g, ] == 0
```

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

```
sum(mat2[g, ] == 0)
```

```
## [1] 9
```

```
sum(mat2[g, ] == 0) == 0
```

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

Now we can apply the filter to the whole `mat2`, save the new object to `mat4` and check its dimensions:

```
no0s = rowSums(mat2==0) == 0
head(no0s)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##            TRUE           FALSE            TRUE            TRUE
## ENSG00000000460 ENSG00000000938
##            TRUE            TRUE
```

```
mat4 = mat2[no0s, ]
head(mat4)
```

```
##                   HSM1   PTF1   RMM1   HSF1   PTM1   RMF1   RMF2   HSM2
## ENSG00000000003   60.5  287.0  212.0  164.5  204.5  255.5  232.0  210.5
## ENSG00000000419   19.5   50.0   24.5   40.5   29.5   33.5   36.0   39.0
## ENSG00000000457   57.0   65.5   58.5   45.5  137.0   44.5   34.5   36.5
## ENSG00000000460    7.5    4.0    3.5    3.0    6.0    2.0    2.0    3.5
## ENSG00000000938   36.5   42.0   40.0   22.0   99.0   36.5   14.0   21.5
## ENSG00000000971 1800.0 2691.0 7118.0 2382.5 1284.0 6783.0 4982.0 3191.5
##                   PTF2   RMM2   HSF2   PTM2   RMM3   RMF3   HSM3   PTF3
## ENSG00000000003  214.5  674.0  150.0  342.5  352.5  240.0  179.5  193.0
## ENSG00000000419   33.0   41.0   30.5   29.0   45.0   28.5   31.0   32.5
## ENSG00000000457  114.5   57.0   31.0  119.5   81.5   59.0   47.5   43.5
## ENSG00000000460    4.0    1.5    8.5    3.5    2.0    5.0    8.5    2.5
## ENSG00000000938   52.0   17.0   32.5   15.5   36.0   27.0   32.5   38.0
## ENSG00000000971 4165.5 2598.0 3612.5 2509.5 5917.0 9123.5 2773.0 2167.5
##                   PTM3   HSF3
## ENSG00000000003  214.0   84.0
## ENSG00000000419   46.5   28.0
## ENSG00000000457   74.5   38.0
## ENSG00000000460    5.0    6.0
## ENSG00000000938   29.0  105.0
## ENSG00000000971 1326.5 1702.5
```

```
dim(mat4)
```

```
## [1] 10644    18
```

So about half of the dataset is lost.

Exercise: A flexible filter

Above we removed 10045 genes which had no expression in at least one sample. This was a quite strict filtering approach. But we may wish to retain genes that are expressed in the *majority* of individuals in *at least one* species (but may or may not be expressed in the other 2 species). Such expression patterns could be species-specific and important for evolutionary divergence.

Now, perform this filter: only keep genes expressed in *at least half of the individuals* of any species. In other words (since there are 6 individuals for each species), the retained genes must be expressed in at least 3 of human OR 3 of chimp OR 3 of macaque. Use again `mat2` to calculate this and save it in object called `mat5`. Check its dimensions and calculate how many genes are excluded from `mat2`.

```
# first try human: genes with min 3 humans expressed
hsmat2 = mat2[,species2=="HS"]
head( hsmat2 > 0)
```

```
##                   HSM1  HSF1 HSM2  HSF2  HSM3  HSF3
## ENSG00000000003  TRUE  TRUE TRUE  TRUE  TRUE  TRUE
## ENSG00000000005 FALSE FALSE TRUE FALSE FALSE FALSE
## ENSG00000000419  TRUE  TRUE TRUE  TRUE  TRUE  TRUE
## ENSG00000000457  TRUE  TRUE TRUE  TRUE  TRUE  TRUE
## ENSG00000000460  TRUE  TRUE TRUE  TRUE  TRUE  TRUE
## ENSG00000000938  TRUE  TRUE TRUE  TRUE  TRUE  TRUE
```

```
head( rowSums( hsmat2 > 0) )
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##               6               1               6               6
## ENSG00000000460 ENSG00000000938
##               6               6
```

```
head( rowSums( hsmat2 > 0) >= 3 )
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##            TRUE           FALSE            TRUE            TRUE
## ENSG00000000460 ENSG00000000938
##            TRUE            TRUE
```

```
# now for all 3 species
filter2 = ( rowSums( mat2[,species2=="HS"] > 0) >= 3 ) |
          ( rowSums( mat2[,species2=="PT"] > 0) >= 3 ) |
          ( rowSums( mat2[,species2=="RM"] > 0) >= 3 )
mat5 = mat2[ filter2, ]
head(mat5)
```

```
##                 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
```

```
dim(mat5)
```

```
## [1] 15595    18
```

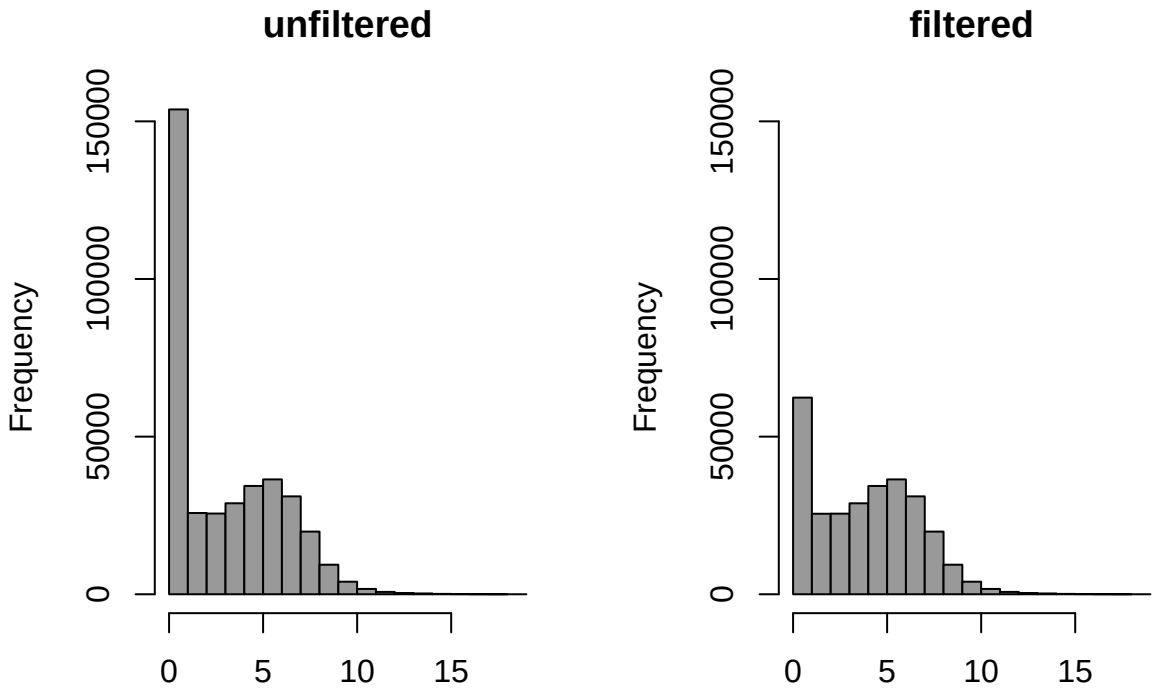
```
nrow(mat2)-nrow(mat5)
```

```
## [1] 5094
```

Now log2-transform both `mat2` and `mat5` (after adding `1`), call these `mat3` and `mat6`.

Plot their histograms side by side and properly label them. Also make x and y axes the same scale.

```
mat3 = log2 ( mat2 + 1 )  # log transform
mat6 = log2 ( mat5 + 1 )  # log transform
par(mfrow=c(1,2))
hist(mat3,col="gray60",xlab="",main="unfiltered",ylim=c(0,150000),xlim=c(0,19))
hist(mat6,col="gray60",xlab="",main="filtered",ylim=c(0,150000),xlim=c(0,19))
```



The flexible-filtered dataset is useful in that it removes genes not expressed in any species, but retains genes expressed in one (i.e., potential species-specific expression).

Now store your `mat2`, `species2`, `sex2`, `mat3`, `mat5`, `mat6`, `mat2.aov.pvals`, `mat2.krus.pvals` objects in an rdata file called `liver_transcriptome_v2.Rdata`, for use next class.

```
save(mat2, species2, sex2, mat3, mat5, mat6, mat2.aov.pvals, mat2.krus.pvals, file="liver_transcriptome_
```

Normalization among samples

We had started working on normalization last class. It generally makes sense to normalize among samples as the final step of preprocessing, after log-transformation and filtering. This is because the latter two steps alter the distributions and can differentiate samples again, rendering early normalization useless.

Scaling: dividing by column sums

So let us continue with the dataset `mat6`, which is summarized for replicates, filtered (flexible version) and log2-transformed.

Let us repeat the scaling-type normalization on this matrix:

```
smat6 = apply( mat6, 2, function(x) { x / sum(x) })   # divide by the column sum
# this is the same:
smat6 = t( mat6 ) / colSums(mat6); smat6 = t(smat6)
# then multiply by column sum means
smat6 = smat6 * mean(colSums(mat6))
dim(smat6)
```

```
## [1] 15595    18
```

```
head(smat6)
```

```
##                      HSM1     PTF1     RMM1     HSF1     PTM1     RMF1
## ENSG00000000003 6.367866 7.684454 7.432506 7.548957 7.591102 7.785490
## ENSG00000000005 0.000000 0.550203 0.960929 0.000000 0.000000 0.000000
## ENSG00000000419 4.669456 5.335360 4.489869 5.505042 4.871763 4.969797
## ENSG00000000457 6.277282 5.695467 5.664501 5.673131 7.023503 5.358225
## ENSG00000000460 3.308456 2.183955 2.085144 2.048373 2.773778 1.541921
## ENSG00000000938 5.603086 5.103827 5.148227 4.632970 6.564393 5.086824
##                      RMF2      HSM2     PTF2      RMM2     HSF2      PTM2
## ENSG00000000003 8.2179322 7.5238776 7.728107 10.201431 7.491552 8.4097050
## ENSG00000000005 0.6112752 0.5697687 1.580170  0.000000 0.000000 0.5839583
## ENSG00000000419 5.4437845 5.1836966 5.072081  5.852841 5.151349 4.8984672
## ENSG00000000457 5.3813926 5.0930056 6.831033  6.358274 5.174864 6.9010224
## ENSG00000000460 1.6562571 2.1135635 2.314908  1.434826 3.361517 2.1662000
## ENSG00000000938 4.0826300 4.3751819 5.710602  4.526052 5.243264 4.0374513
##                      RMM3      RMF3     HSM3      PTF3      PTM3     HSF3
## ENSG00000000003 8.392944 7.1759287 7.382686 7.7063333 8.0643317 6.428745
## ENSG00000000005 0.000000 0.9068658 0.000000 0.5931536 0.6088299 0.000000
## ENSG00000000419 5.476178 4.4279020 4.924512 5.1370289 5.7971148 4.872650
## ENSG00000000457 6.311708 5.3567570 5.515368 5.5524093 6.4929418 5.301362
## ENSG00000000460 1.571366 2.3442141 3.198892 1.8326630 2.6904332 2.815832
## ENSG00000000938 5.164764 4.3596257 4.989603 5.3594129 5.1070997 6.748236
```

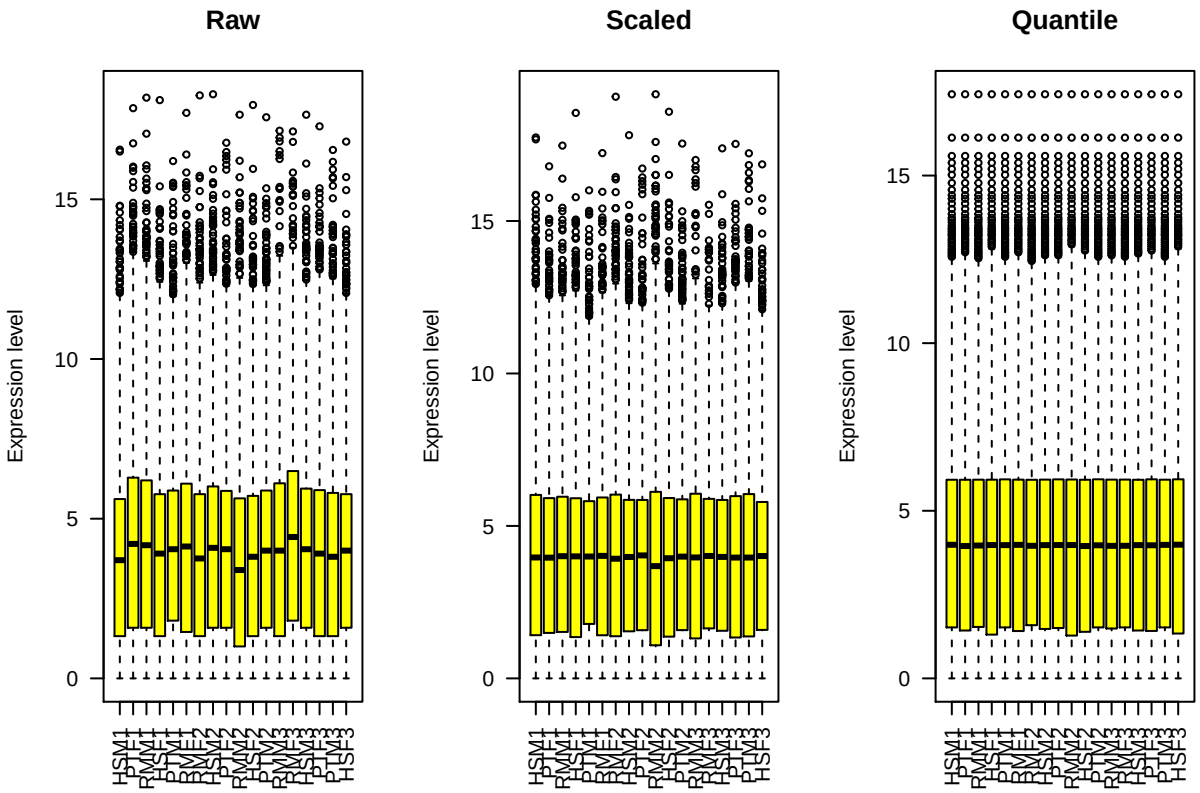
Quantile normalization

We can also use a more powerful approach, **quantile normalization**. This is an alternative method which makes distributions exactly the same. We can use the Bioconductor package `preprocessCore`s function for that. See: http://www-microarrays.u-strasbg.fr/Solo/introduction.html for more.

```
library(preprocessCore)
nmat6 = normalize.quantiles( mat6 )
# we need to redefine, because the function removes the names
colnames( nmat6 ) = colnames( mat6 )
rownames( nmat6 ) = rownames( mat6 )
```

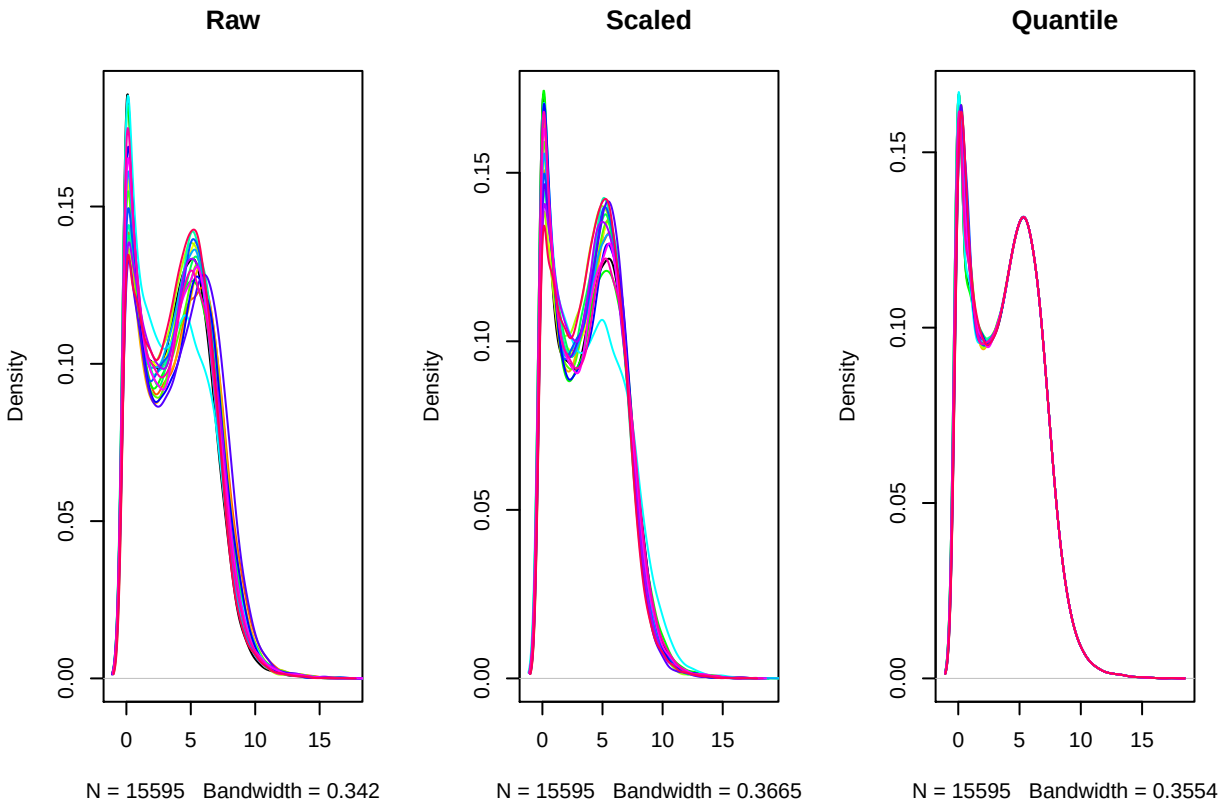
The `density` function and comparing normalization approaches

```
par(mfrow=c(1,3))
boxplot( mat6, las=2, col=7, ylab='Expression level', main='Raw' )
boxplot( smat6, las=2, col=7, ylab='Expression level', main='Scaled' )
boxplot( nmat6, las=2, col=7, ylab='Expression level', main='Quantile' )
```



We could also compare the distributions after the two normalizations in more detail, using the `density` function:

```
par(mfrow=c(1,3))
plot(density(mat6[,1]), type="l", main='Raw')
for (i in 2:18) lines(density(mat6[,i]), type="l", col=rainbow(18)[i])
plot(density(smat6[,1]), type="l", main='Scaled')
for (i in 2:18) lines(density(smat6[,i]), type="l", col=rainbow(18)[i])
plot(density(nmat6[,1]), type="l", main='Quantile')
for (i in 2:18) lines(density(nmat6[,i]), type="l", col=rainbow(18)[i])
```



Quantile normalization could improve sensitivity but only if expression differences between samples are of limited magnitude and affect *few genes*. If you are comparing very different transcriptomes, e.g. different tissues, quantile normalization can also *distort* distributions and create spurious signals of differential expression.

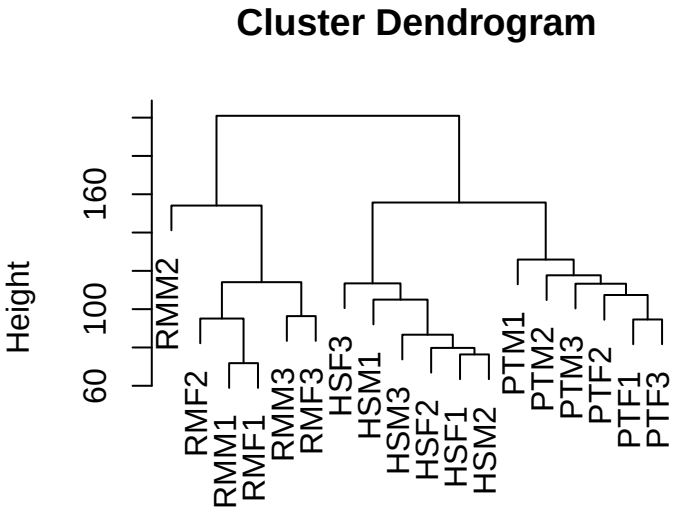
In case you cannot be sure theoretically which approach, q.n. or scaling, is superior, can run downstream tests using both methods, to ensure your choice of normalization does not have a large influence.

Principle Components Analysis

Before continuing into differential expression (DE), we still need to ensure that the data is sound. One approach is to summarise the samples (or the genes) to determine the presence of any **outlier samples** that could confound the DE analysis. These you may then consider removing before continuing the analysis.

One approach would be drawing trees again:

```
plot( hclust( dist( t( nmat6 ), method = "euclidean")  ), sub="", xlab="")
```



There are more powerful techniques for summarizing large scale data, or **dimensionality reduction**, than hierarchical clustering. PCA is one of these, which we can use to identify subtle biological/technical effects that influence many genes' expression. With PCA, we can investigate the data summarized in multiple dimensions.

PCA summarizes variance in the full matrix into few artificial dimensions, which are the 'principle components'. This you can imagine as artificial genes that represent many real genes in the dataset.

The first component always explains the highest amount of variance, the second one the second highest, etc. The components are orthogonal to each other (they don't correlate). One of the functions in R is called `prcomp`.

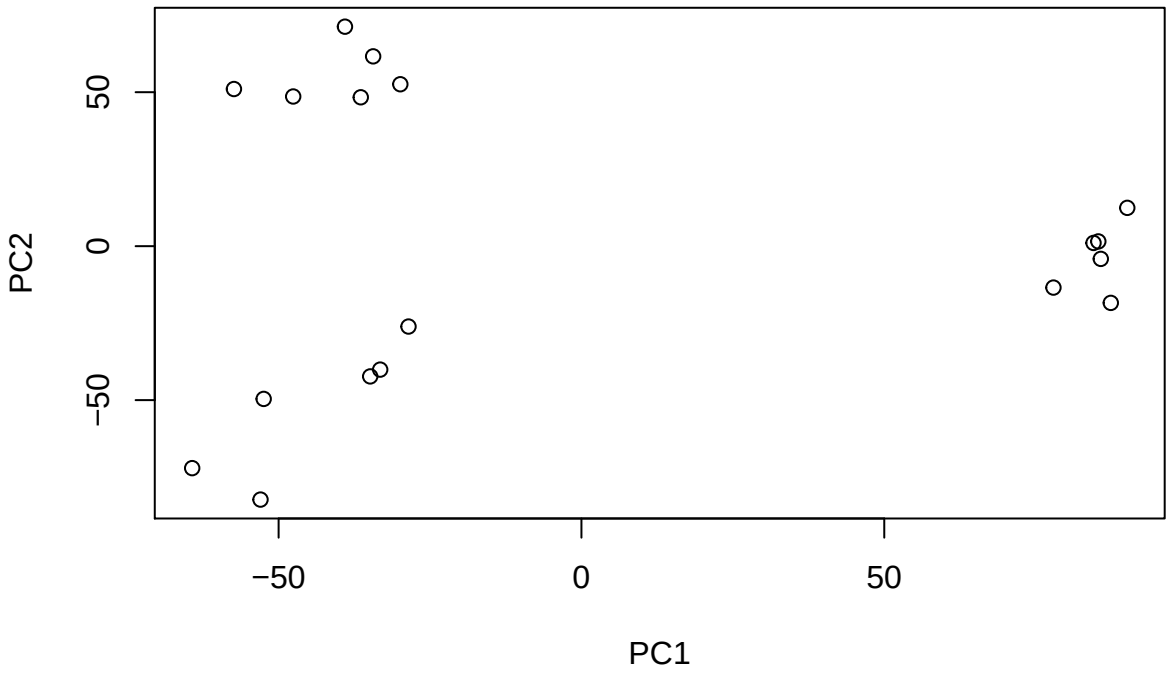
For more information on PCA you can refer to: http://setosa.io/ev/principal-component-analysis/ https://tgmstat.wordpress.com/2013/11/21/introduction-to-principal-component-analysis-pca/

```
# this summarizes variance across genes, first scaling them to the same variance
pc = prcomp(  t( nmat6 ), scale=T)

# info on the first 6 principle components, regarding total variance explained
summary( pc )$imp[, 1:6 ]
```

```
##                             PC1      PC2      PC3      PC4      PC5
## Standard deviation     62.74805 47.37336 40.56495 31.92596 29.96095
## Proportion of Variance  0.25247  0.14391  0.10552  0.06536  0.05756
## Cumulative Proportion   0.25247  0.39638  0.50190  0.56725  0.62482
##                             PC6
## Standard deviation     28.57085
## Proportion of Variance  0.05234
## Cumulative Proportion   0.67716
```

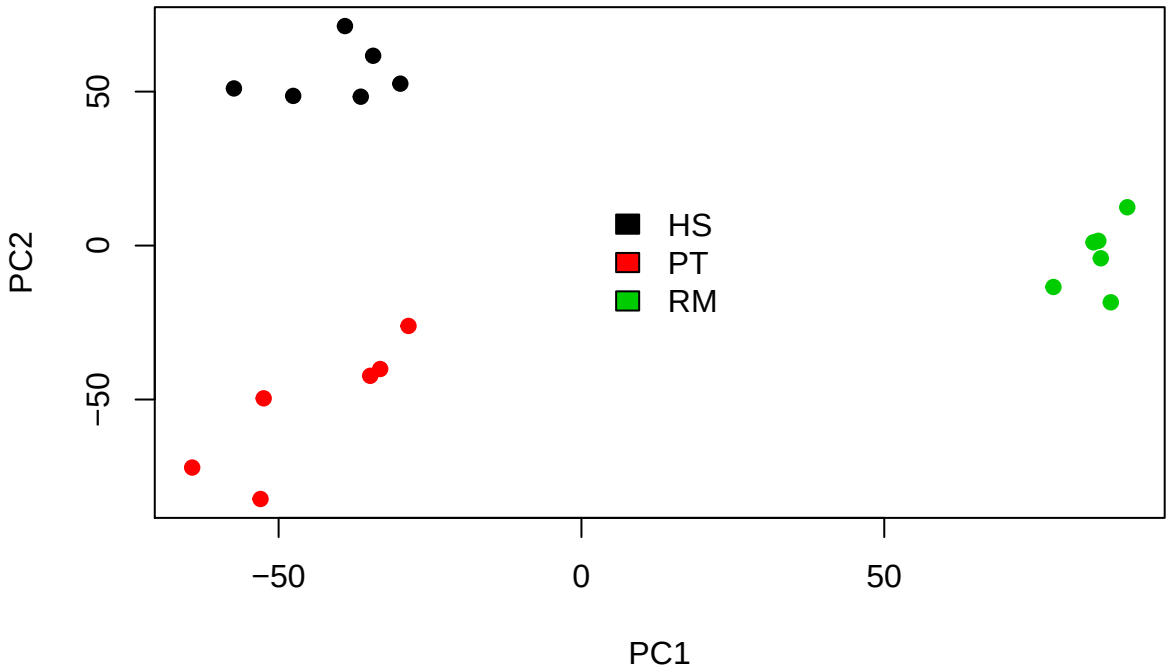
```
# plotting the first two components against each other
plot( pc$x[,1:2])
```



More on plots: `text`, `legend`, `identify`

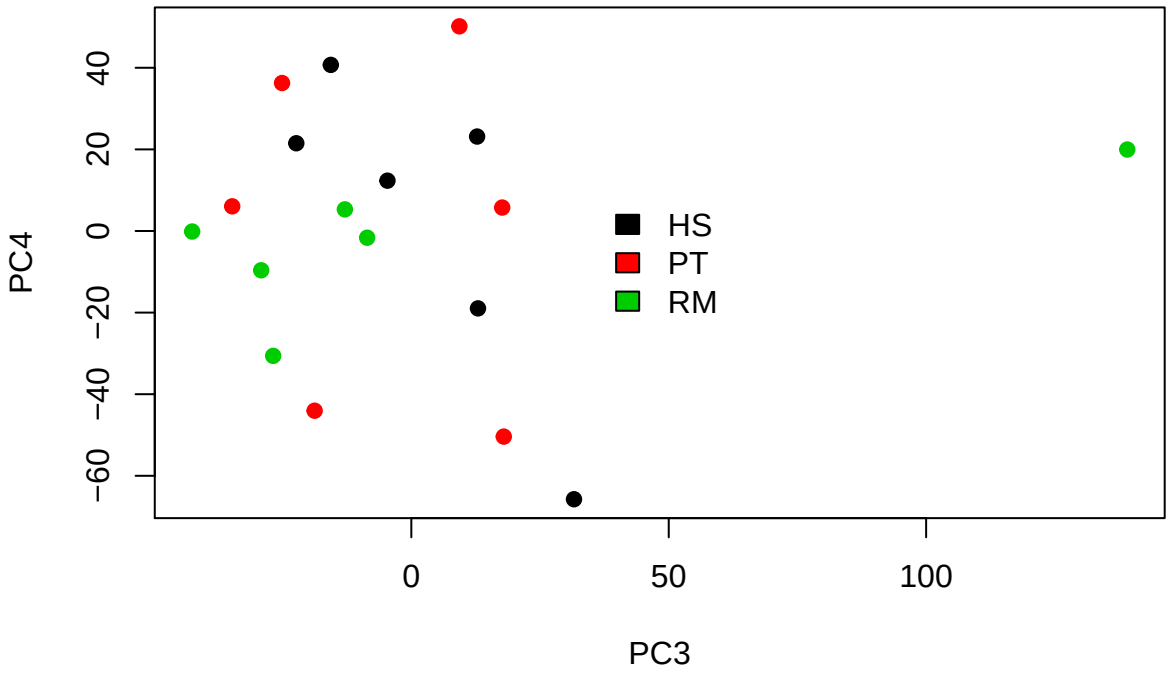
But we'd better add color to separate the species. Nicely, factor levels are automatically converted into color names by plot:

```
plot( pc$x[,1:2], col = species2, pch=19 )
legend("center", legend = levels(species2), fill = 1:3, bty = "n")
```



We can also plot the 3rd and 4th components:

```
plot( pc$x[,3:4], col=species2, pch=19)
legend("center", legend = levels(species2), fill = 1:3, bty = "n")
```

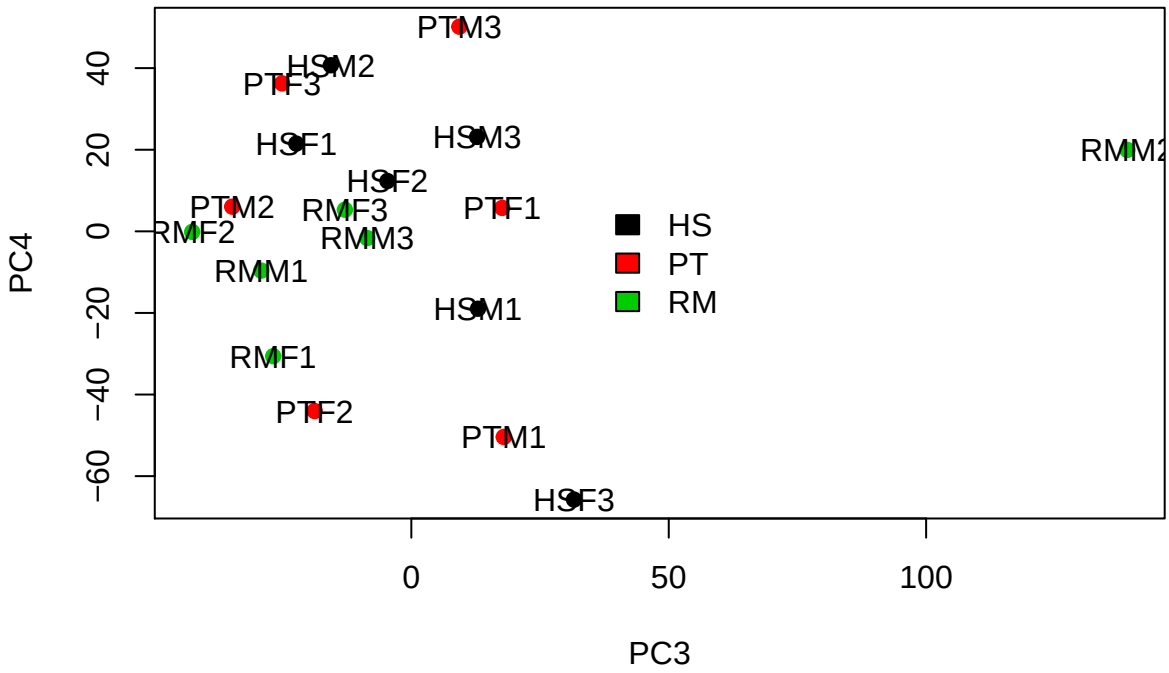


Interestingly, there is an outlier macaque, different from all others. Which macaque is that? One approach is to use the `identify` function, which allows you to click on points on a plot, and returns the indices (you stop by pressing ESC):

```
identify( pc$x[,1:2])
```

Alternatively, you can add text on the plot:

```
plot( pc$x[,3:4], col=species2, pch=19)
legend("center", legend = levels(species2), fill = 1:3, bty = "n")
text( pc$x[,3:4], rownames(pc$x) )
```



You could also have identified the individual by using the PC3 values:

```
names( which( pc$x[,3] > 100) )
```

```
## [1] "RMM2"
```

Now save your most recent objects in `liver_transcriptome_v3.Rdata`

```
save(nmat6, species2, sex2, file="liver_transcriptome_v3.Rdata")
```