MedBioinfo 2025

23/11 2025

Andreas Tilevik

Institutionen för Biovetenskap

Högskolan i Skövde

Contents

About this tutorial

The aim of this tutorial is to show how the theories in the video lectures can be applied on gene expression data from RNA-seq. Thus, it is therefore crucial that you have watched these videos since almost no theory will be explained in this tutorial. Also, we have tried to use only basic R functions as much as possible to help beginners to understand the code. Only five packages are needed for this tutorial (reshape, MASS, pvclust, pROC, ggplot2 and ncvreg). If you have **not** installed these packages, run the following code:

```
install.packages(c("reshape","pROC","MASS", "pvclust","tibble","ggplot2","dplyr","tidyr"))
```

Note that there is a package called tidyverse, which is a collection of R packages designed for data science. Tidyverse includes packages such as ggplot2, tidyr, dplyr and tibble, and much more. To save installation time, we here install just the packages ggplot2, dplyr, tidyr and tibble.

The dataset

In this computer lab, we will analyze just a subset of the original data (Friman et al. Cell Reports, 2023). The data that we will use here includes the gene expression of 25 genes from an RNA-seq experiment. In most cases, one should normalize the count data across the different individuals so that all individuals have about the same total amount of counts. One simple way to do this is to divide all gene counts by the total counts for all genes for each individual and then log those counts. Anyway, we will work with the raw counts here. The study involves nine patients with the disease called "Common Variable ImmunoDeficiency (CVID)", which is associated with a deficiency in the production of antibodies. The patients therefore have problems defending themselves against pathogens. To investigate the cause of this disease, one has isolated naive B-cells from nine CVID patients and nine healthy donors (HD). Let's read in the data file (MedBioinfo_CVID.csv) in R by:

```
df <-read.table(file.choose(),header=TRUE,sep=";",row.names=1)
```

After you have run the above command, the data will be stored in a data frame called "df", which is a type of list in R. We can find the type of storage mode of the object "df" by:

```
mode(df)
"list"
```

We see that our object is a list, and the type of list in our case is a so-called data frame, which can, in comparison to a matrix, include both numbers and characters.

Next, we print the data frame by:

```
print(df)
```

or simply by running the name of the data frame:

```
df
```

```
> print(df)
          CVID1 CVID2 CVID3 CVID4 CVID5 CVID6 CVID7 CVID8 CVID9  HD1  HD2  HD3  HD4  HD5  HD6  HD7  HD8  HD9
TSPAN6        5     1     2     7     5     3     2     6     1    0    2    3    9    2    0    0    3    7
DPM1        487   377   438   494   719   408   622   505   365 2548 2472 2340 3386 1958 1864 6127 3728 3226
SCYL3       583   671   818   745   456   750   816   430   775  441  359  444  699  312  497  561  638  584
C1orf112    300   220   193   318   134   152   315   131   226  232  317  334   45  190  227  201   99  234
FGR       13984 13241 10448 10438 19922 11601 11018 19839  8753 2124 3142 3496 2809 4179 4352 3648 3137 3704
CFH           8     6     0     0     9     6     0     0     1    0    0    3    0    2    2    0    0    0
FUCA2       177   250   217   191   458   183   186   439   172  108   76  125  188   96  102  137  112   84
GCLC        608   442   566   558   572   502   732   637   724  629  510  650  647  545  340  466  519  567
NFYA       1194  1597  1774  1919  1313  1055  1678   971  1478 2470 3238 2624 1740 2111 2484 3130 2705 2245
STPG1       122   142   136   140   160   126   124   134   129  125  104   98   95   96   71   77  176  166
NIPAL3      971   894  1307  1219   890   720   900   839  1163  446  740  762  348  454  598  571  473  529
LAS1L      1691  2183  2515  1702  1591  1623  1545  1700  1894 3472 3493 3783 6128 3819 2754 6012 5262 4507
ENPP4       283   123   156   293   625   382   409   332   373 1618  622 2383 1586 1079  648 2670 1045  993
SEMA3F        0     1     0     0     2     0     2     4     1   16   32   44    9   31   51  104  156  152
CFTR          0     0     0     0     0     0     0     0     0    0    0    0    9    0    0    0    0    0
ANKIB1     1233  1083  1447  1448  1077  1135  1634  1235  1433 2119 2412 2648 1772 1800 2025 2890 2123 2046
CYP51A1       4    10    11    14    21     2     8    11    13   47   56   52    0   30   62   97   72   53
KRIT1       389   145   158   271   316   316   290   375   300  288  334  284  292  369  306  412  222  269
RAD52       746   697  1040   780   608   734   704   632   675  466  764  824  517  604  897  493  575  635
MYH16         3     6     2     1     1     0     2     1     2    1    1    0    0    0    1    0    3    0
BAD         245   271   276   245   454   312   211   477   258  125  160  164  275  338  248  619  321  192
LAP3        623   756   683   575   673   640   529   740   712 1204 1292  877  952 1228 1150 2347 1960 1637
CD99       1550  2634  3163  2205  4289  4006  1064  3817  1970 1224 2022 1568 2248 2517 2266 2201 1670 1462
HS3ST1      894   519   795   608   268   478   626   469   502   65  154  140  143  130  149   56   78  176
AOC1          0     0     0     0     2     2     0     4     0    0    0    0    2    0    0    0    0    0
>
```

Each row in this data frame contains the expression (the read count) of the 25 genes for all the individuals, whereas the columns show the expression for each individual for all the 25 genes. Note that the first nine columns show the gene expression in the CVID patients, whereas the last nine columns show the gene expression of the healthy donors (HD). The numbers that you see in this table are called read counts because they represent the number of reads that have mapped to the specific gene in the human genome.

Exercise 1: What is the read count of the gene "STPG1" for the CVID patient number 8?

Tibble

There are a range of different ways to read in a data file in R. If you use functions in RStudio, your data frame will instead be a tibble. A tibble is a modern type of data frame. In comparison to data frames, a tibble makes fewer assumptions about the data. For example, a tibble never changes the names of the variables or sets row names when we read in data from a file. To see what a tibble looks like, we will here convert our data frame to a tibble by:

```
library(tibble)
tib=as_tibble(df)
tib
```

Note that when we print the tibble, it shows also the data type (<int>) for each column.

Transpose the data frame

Usually, in basic statistics, it is common that the subjects are on the rows and the variables on the columns. As you see in the output above, the genes are on the rows and the subjects on the columns. The reason for this is that it is a lot easier to view the data, including thousands of genes on the rows instead of thousands of genes as columns. To transpose the data frame, we use the function "t":

```
tdf=t(df)
```

The output of this function gives a transposed matrix. However, since we want a data frame instead of a matrix, we will convert the matrix to a data frame by:

```
tdf=data.frame(tdf)
tdf
```

```
> tdf
      TSPAN6 DPM1 SCYL3 C1orf112   FGR CFH FUCA2 GCLC NFYA STPG1 NIPAL3 LAS1L ENPP4 SEMA3F CFTR ANKIB1 CYP51A1 KRIT1 RAD52 MYH16 BAD LAP3 CD99 HS3ST1 AOC1
CVID1      5  487   583       300 13984   8   177  608 1194   122    971  1691   283      0    0   1233       4   389   746      3 245  623 1550    894    0
CVID2      1  377   671       220 13241   6   250  442 1597   142    894  2183   123      1    0   1083      10   145   697      6 271  756 2634    519    0
CVID3      2  438   818       193 10448   0   217  566 1774   136   1307  2515   156      0    0   1447      11   158  1040      2 276  683 3163    795    0
CVID4      7  494   745       318 10438   0   191  558 1919   140   1219  1702   293      0    0   1448      14   271   780      1 245  575 2205    608    0
CVID5      5  719   456       134 19922   9   458  572 1313   160    890  1591   625      2    0   1077      21   316   608      1 454  673 4289    268    2
CVID6      3  408   750       152 11601   6   183  502 1055   126    720  1623   382      0    0   1135       2   316   734      0 312  640 4006    478    2
CVID7      2  622   816       315 11018   0   186  732 1678   124    900  1545   409      2    0   1634       8   290   704      2 211  529 1064    626    0
CVID8      6  505   430       131 19839   0   439  637  971   134    839  1700   332      4    0   1235      11   375   632      1 477  740 3817    469    4
CVID9      1  365   775       226  8753   1   172  724 1478   129   1163  1894   373      1    0   1433      13   300   675      2 258  712 1970    502    0
HD1        0 2548   441       232  2124   0   108  629 2470   125    446  3472  1618     16    0   2119      47   288   466      1 125 1204 1224     65    0
HD2        2 2472   359       317  3142   0    76  510 3238   104    740  3493   622     32    0   2412      56   334   764      1 160 1292 2022    154    0
HD3        3 2340   444       334  3496   3   125  650 2624    98    762  3783  2383     44    0   2648      52   284   824      0 164  877 1568    140    0
HD4        9 3386   699        45  2809   0   188  647 1740    95    348  6128  1586      9    9   1772       0   292   517      0 275  952 2248    143    2
HD5        2 1958   312       190  4179   2    96  545 2111    96    454  3819  1079     31    0   1800      30   369   604      0 338 1228 2517    130    0
HD6        0 1864   497       227  4352   2   102  340 2484    71    598  2754   648     51    0   2025      62   306   897      1 248 1150 2266    149    0
HD7        0 6127   561       201  3648   0   137  466 3130    77    571  6012  2670    104    0   2890      97   412   493      0 619 2347 2201     56    0
HD8        3 3728   638        99  3137   0   112  519 2705   176    473  5262  1045    156    0   2123      72   222   575      3 321 1960 1670     78    0
HD9        7 3226   584       234  3704   0    84  567 2245   166    529  4507   993    152    0   2046      53   269   635      0 192 1637 1462    176    0
>
```

Note that this data frame "tdf" now has the subjects on the rows and the genes on the columns.

Before we move on, we should create a vector that can be used to help us find on which rows (or columns) the CVID patients and the HD are located. We see, in the figure above, that the CVID patients are located on the first nine rows, whereas the HDs are located on the last nine rows. We therefore create a vector with a length of 18, where the character string "CVID" is stored in the first nine elements and "HD" in the last nine elements. To avoid typing the words "CVID" and "HD" nine times manually in R, we use the "rep" function to repeat these words nine times each.

```
Group=rep(c("CVID","HD"),c(9,9))
Group
[1] "CVID" "CVID" "CVID" "CVID" "CVID" "CVID" "CVID" "CVID" "CVID" "HD"
[11] "HD"  "HD"  "HD"  "HD"  "HD"  "HD"  "HD"  "HD"
```

We then define the "Group" variable as a factor, which tells R that this is a categorical variable with two levels:

```
Group=factor(Group)
```

Descriptive statistics

We are now ready to generate some descriptive statistics of our data.

Let's say that we like to calculate the mean read count of the gene STPG1. To extract only the column for the gene STPG1, we use the name of the data frame "tdf" and a dollar sign "$", followed by the name of the column, such as:

```
tdf$STPG1
122 142 136 140 160 126 124 134 129 125 104  98  95  96  71  77 176 166
```

The output shows the read counts for the gene STPG1 for all 18 individuals. If we like to extract only the counts for the CVID patients, we can use the following code to check which elements in the group variable that contain the string "CVID"

```
Group=="CVID"
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
FALSE FALSE FALSE FALSE FALSE FALSE
```

By using this condition, we can extract the read counts for the gene STPG1, only for the CVID patients by:

```
tdf$STPG1[Group=="CVID"]
122 142 136 140 160 126 124 134 129
```

Another common function to extract a subset of a data frame is the functions "select" and "filter" in the **dplyr** package:

```
library(dplyr)
select(tdf, STPG1) %>% filter(Group=="CVID")
```

With the function select, we can select one or several variables in the data frame. The pipe operator (%>%) feeds the results of the code from the left-hand side of the pipe into the operation to the right-hand side of the pipe. Thus, the selected variable STPG1 goes into the function filter, which extracts only the CVID patients. The select function keeps columns in a data frame.

Let's calculate the mean, median, SD and IQR for the gene STPG1 for all 18 individuals:

```
mean(tdf$STPG1) # tdf %>% pull(STPG1) %>% mean()
123.3889
median(tdf$STPG1)
125.5
sd(tdf$STPG1)
29.03778
IQR(tdf$STPG1)
39.5
```

Exercise 2: what is the median value of the gene "FUCA2"?

If we like to compute the above statistics on only the CVID patients, we extract only the read counts in the elements of the vector "tdf$STPG1" where "Group" is equal to "CVID":

```
mean(tdf$STPG1[Group=="CVID"])
134.7778
```

or we can use the functions in the dplyr package, where we first filter the CVID patients, pull out STPG1 from the filtered data frame, and then compute the mean.

```
tdf %>% filter(Group == "CVID") %>% pull(STPG1) %>% mean()
```

Exercise 3: compute the IQR for the gene ANKIB1 only for the HD

Apply

Let's say that we like to calculate the median value for all the genes. To apply, for example, the function "median" on all the columns, we can use the function apply:

```
apply(tdf,2,median)
```

Note that the output gives us the median value for all genes. The argument "2" specifies that the median should be calculated based on the columns. Try and change the argument to "1" and interpret the output. What do the medians represent?

Note that we will get the exact same result if we use our original data frame "df" when we calculate the median of the rows as when we calculate the median of the columns of the transposed data frame:

```
apply(tdf,2,median) # Calculate the median for the columns in the transposed data frame
apply(df,1,median) # Calculate the median for the rows in the original data frame
```

To calculate the mean gene expression separately for the nine CVID patients and the nine healthy donors, we can use the following code where we extract only the rows for CVID or HD:

```
apply(tdf[Group=="CVID",],2,mean)
apply(tdf[Group=="HD",],2,mean)
```

In the dplyr package, there is a similar function "summarise" that can compute the same things:

```
tdf %>% filter(Group == "CVID") %>% summarise(across(where(is.numeric), mean))
```

This avoids errors if non-numeric columns exist because “where(is.numeric)” makes sure that we only calculate the mean of numeric columns.

Log2FC

Once we have calculated the means of the two groups, CVID and HD, we can calculate the log2 fold change (log2FC) by dividing the disease state (CVID patient) by the baseline (HD) for each gene. Let’s first save the mean values in two vectors:

```
Mean_CVID=apply(tdf[Group=="CVID",],2,mean)
Mean_HD=apply(tdf[Group=="HD",],2,mean)
```

Next, we divide the mean values for the CVID patients by the mean values for the HD:

```
FC=Mean_CVID/Mean_HD
```

We have now calculated the fold change (FC). By computing the log2 of these FC values, the FC values are transformed, so that upregulated genes get positive FC values, whereas downregulated genes get negative FC values:

```
log2FC=log2(FC)
```

Let’s summarize the means and the log2 FC values in a data frame:

```
sumdf=data.frame(Mean_CVID,Mean_HD,log2FC)
round(sumdf,2) # Print data frame with two decimals
```

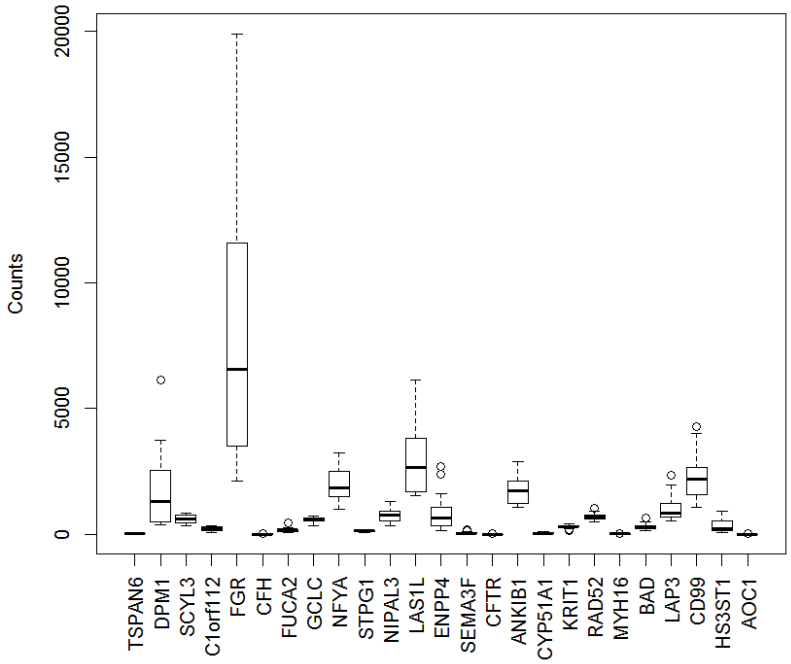
	Mean_CVID	Mean_HD	log2FC
TSPAN6	3.56	2.89	0.30
DPM1	490.56	3072.11	-2.65
SCYL3	671.56	503.89	0.41
C1orf112	221.00	208.78	0.08
FGR	13249.33	3399.00	1.96
CFH	3.33	0.78	2.10
FUCA2	252.56	114.22	1.14
GCLC	593.44	541.44	0.13
NFYA	1442.11	2527.44	-0.81
STPG1	134.78	112.00	0.27
NIPAL3	989.22	546.78	0.86
LAS1L	1827.11	4358.89	-1.25
ENPP4	330.67	1404.89	-2.09
SEMA3F	1.11	66.11	-5.89
CFTR	0.00	1.00	-Inf
ANKIB1	1302.78	2203.89	-0.76
CYP51A1	10.44	52.11	-2.32
KRIT1	284.44	308.44	-0.12
RAD52	735.11	641.67	0.20
MYH16	2.00	0.67	1.58
BAD	305.44	271.33	0.17
LAP3	659.00	1405.22	-1.09
CD99	2744.22	1908.67	0.52
HS3ST1	573.22	121.22	2.24
AOC1	0.89	0.22	2.00

Exercise 4: identify the top two genes that are upregulated and the top two genes that are downregulated in CVID compared to the HD based on the log2FC.

Box plot

By using the function “boxplot”, we can plot the distribution of counts for each gene by:

```
boxplot(tdf,las=3,ylab="Counts")
```

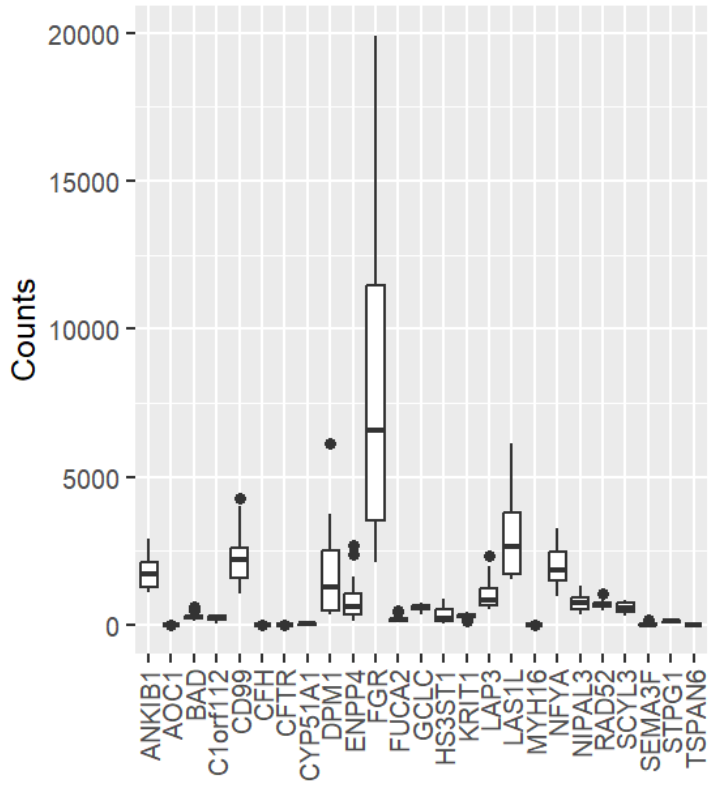


Which gene has the highest counts?

We can make a similar plot with the package ggplot2:

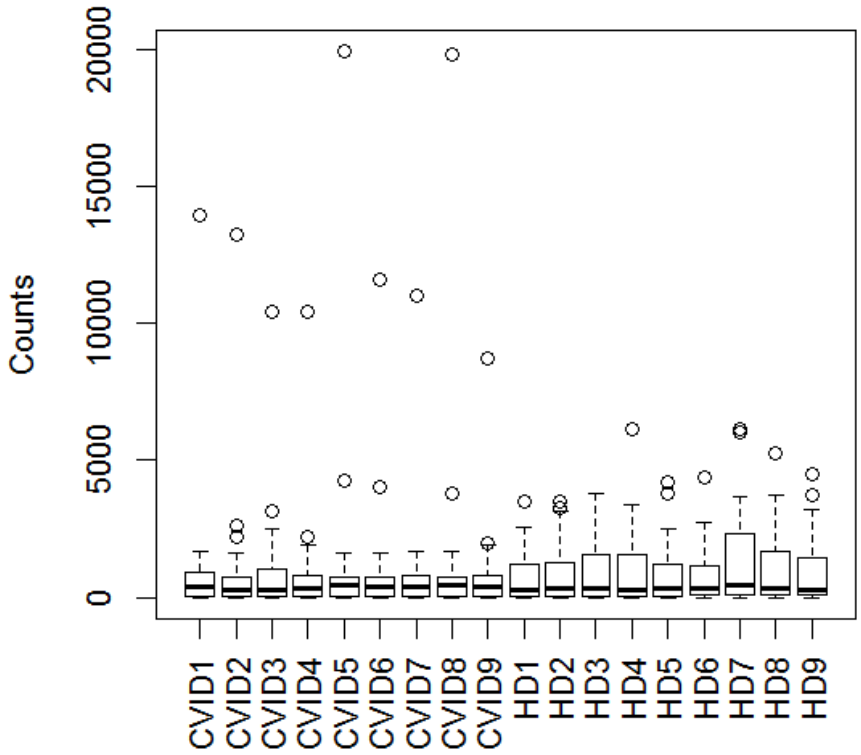
```
library(tidyr)
library(ggplot2)

tdf_long <- tdf %>%
  pivot_longer(cols = where(is.numeric),
               names_to = "variable",
               values_to = "value")
ggplot(tdf_long, aes(x = variable, y = value)) +
  geom_boxplot() +
  labs(y = "Counts", x = "") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```



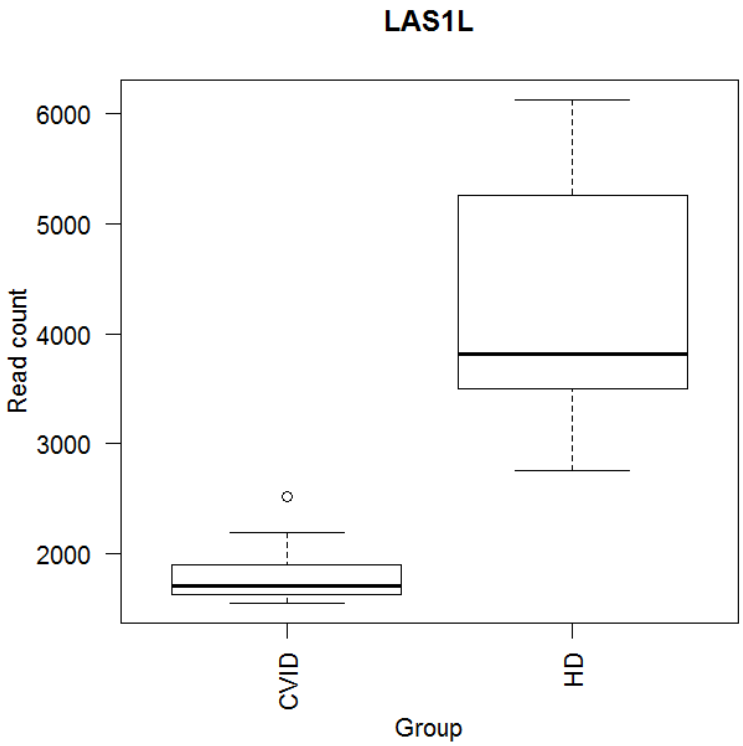
We can also plot the total counts, for all genes, for each individual by using our original data frame “df”

```
boxplot(df,las=3,ylab="Counts")
```



To compare, for example, the expression of the gene LAS1L between CVID and HD, we can generate a boxplot that separates the counts of the two groups:

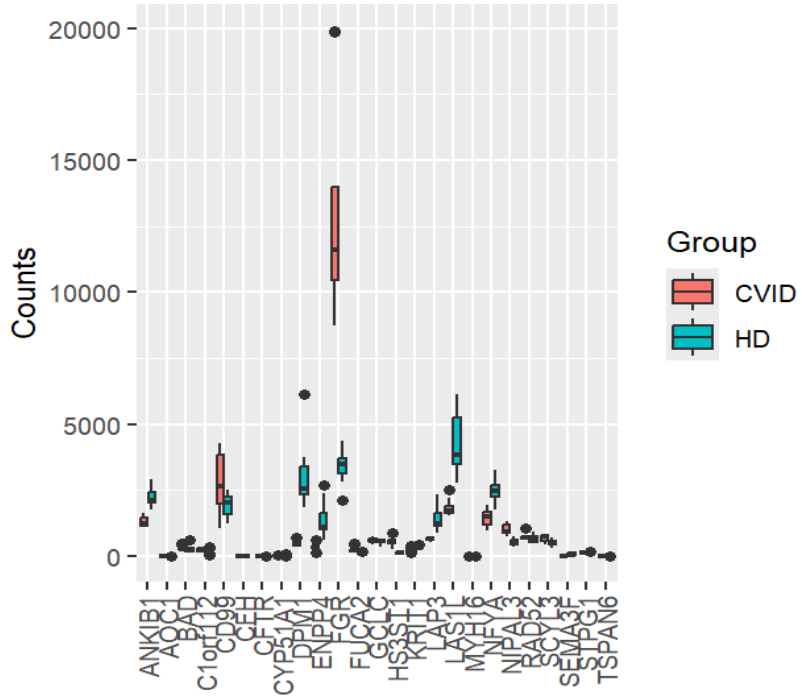
```
boxplot(tdf$LAS1L~Group ,las=2,ylab="Read count", names=c("CVID","HD"), main="LAS1L")
```



We can make a similar boxplot for all genes in ggplot2

```
tdf_long <- tdf %>%
  mutate(Group = Group) %>%   # add group vector temporarily
  pivot_longer(
    cols = where(is.numeric),
    names_to = "variable",
    values_to = "value"
  )

ggplot(tdf_long, aes(x = variable, y = value, fill = Group)) +
  geom_boxplot(position = position_dodge(width = 0.8)) +
  labs(y = "Counts", x = "") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```

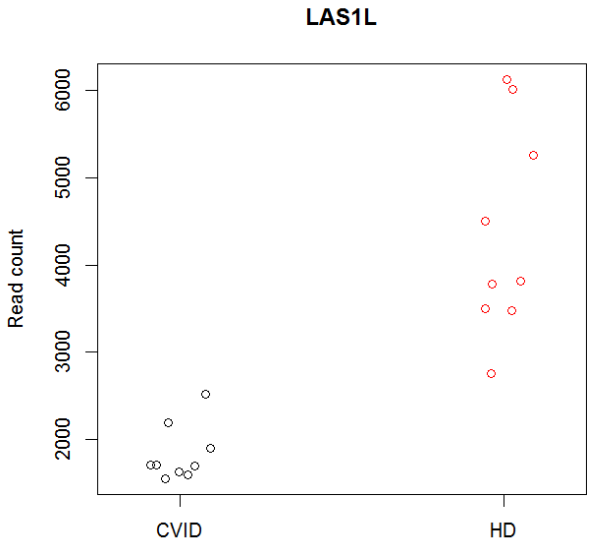


From this plot, we see that the two groups are highly different in the expression of the FGR gene.

Dot plot

We can also create a dot plot, by using the stripchart function, where the read count for each patient is shown:

```
stripchart(tdf$LAS1L~Group,vertical = T, method="jitter",pch=1,col=c(1:2),ylab="Read count",main="LAS1L")
```



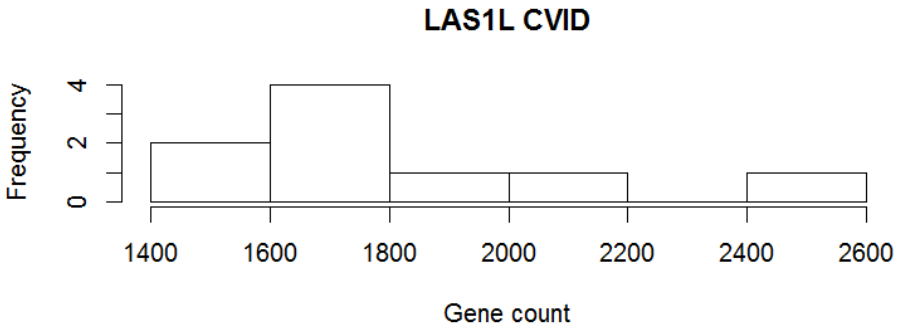
In ggplot 2:

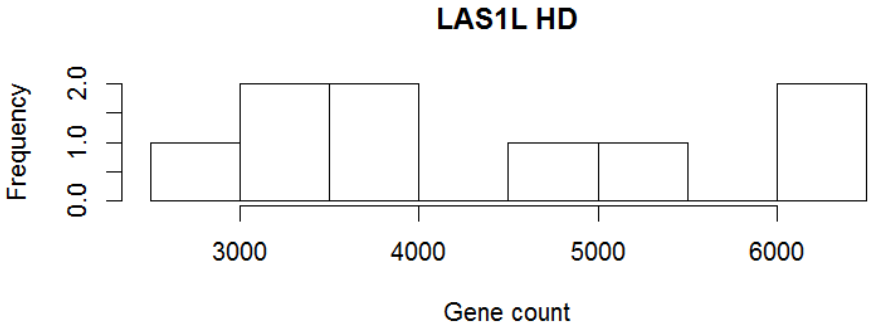
```
ggplot(tdf_long, aes(x = variable, y = value, color = Group)) +
  geom_jitter(size=2,width=0.1,height=0,alpha = 0.5) +
  labs(y = "Counts", x = "") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```

Histogram

In addition to box plots, histograms can also be used to show the distribution of the read count. For example, we can show the distribution of the gene LAS1L with the following code:

```
par(mfrow=c(2,1))
hist(tdf$LAS1L[Group=="CVID"],main="LAS1L CVID",xlab=c("Read count"))
hist(tdf$LAS1L[Group=="HD"],main="LAS1L HD",xlab=c("Read count"))
```





Since we have only nine data points in each group, it is hard to identify a clear pattern in the distribution.

Statistical inference

We will now briefly cover some common statistical tests that can be used to make predictions about the population based on our sample of the nine CVID patients and HD.

t-tests

We will here detect differentially expressed genes by using a simple t-test. However, since the gene expressions are based on count data, Poisson regression or negative binomial regression are the most appropriate tests to use, which we will discuss later. We will here use a t-test to check if there is a significant difference in the mean count of the gene "TSPAN6" between the CVID patients and the HD. In this example, we use the Welsh t-test (*var.equal = FALSE*) that does not assume an equal variance in read counts between the two groups:

```
t.test(tdf$TSPAN6~Group,var.equal = FALSE)
        Welch Two Sample t-test
data:  tdf$TSPAN6 by Group
t = 0.51402, df = 14.379, p-value = 0.615
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.108171  3.441504
sample estimates:
mean in group CVID   mean in group HD
         3.555556          2.888889
```

We see that the p-value is 0.615, which is greater than the general significance level of 0.05. We therefore conclude that there is no significant difference in the mean count of the gene TSPAN6 between the two groups. If we like to extract only the p-value from the output above, we can add "$p.value" at the end:

```
t.test(tdf$TSPAN6~Group,var.equal = FALSE)$p.value
0.6150441
```

We would now like to compute the t-test for all 25 genes. To do that, we can use the apply function:

```
pvals=apply(tdf,2,function(x) {t.test(x~Group,var.equal = FALSE)$p.value})
```

Let's add the p-values to our data frame where we have summarized the means and log2FC:

```
sumdf$pvals=pvals
```

Then we print the genes, sorted on the p-values:

```
sumdf_order_p=sumdf[order(sumdf$pvals),]   # Order based on the p-values
round(sumdf_order_p,5)                     # Print sorted data frame with five decimals
```

```
> round(sumdf_order_p,5) # Print sorted data fram
           Mean_CVID    Mean_HD   log2FC   pvals
ANKIB1    1302.77778 2203.88889 -0.75846 0.00003
NFYA      1442.11111 2527.44444 -0.80950 0.00006
HS3ST1     573.22222  121.22222  2.24144 0.00006
NIPAL3     989.22222  546.77778  0.85534 0.00007
FGR      13249.33333 3399.00000  1.96274 0.00007
LAS1L     1827.11111 4358.88889 -1.25440 0.00016
DPM1       490.55556 3072.11111 -2.64674 0.00035
CYP51A1     10.44444   52.11111 -2.31886 0.00149
LAP3       659.00000 1405.22222 -1.09245 0.00164
ENPP4      330.66667 1404.88889 -2.08701 0.00204
FUCA2      252.55556  114.22222  1.14476 0.00631
SEMA3F       1.11111   66.11111 -5.89482 0.00888
SCYL3      671.55556  503.88889  0.41440 0.02115
CD99      2744.22222 1908.66667  0.52383 0.06709
MYH16        2.00000    0.66667  1.58496 0.06718
CFH          3.33333    0.77778  2.09954 0.08727
STPG1      134.77778  112.00000  0.26708 0.10936
RAD52      735.11111  641.66667  0.19614 0.17804
AOC1         0.88889    0.22222  2.00000 0.23635
GCLC       593.44444  541.44444  0.13230 0.27314
CFTR         0.00000    1.00000     -Inf 0.34659
KRIT1      284.44444  308.44444 -0.11686 0.48968
BAD        305.44444  271.33333  0.17084 0.57372
TSPAN6       3.55556    2.88889  0.29956 0.61504
C1orf112   221.00000  208.77778  0.08208 0.76275
```

Exercise 5: how many genes are differentially expressed if you use a significance level of 0.05?

Adjusted p-values

One problem with the above analysis is that we have computed 25 independent t-tests. The risk that we commit a type 1 error, if the null hypothesis is true, is 5%, if our significance level is 0.05. When we compute 25 t-tests, and the null hypothesis is true, the risk that we commit at least one type 1 error is:

$$\text{Risk of making at least one type I error} = 1-(1-\alpha)^k$$

where alpha is our significance level and k is the number of t-tests that we compute. If we set alpha to 0.05 and k to 25, we can calculate this risk by:

```
1-(1-0.05)^25
[1] 0.7226104
```

We see that the risk of committing at least one type 1 error is 72%. A number of different post-hoc tests have been developed to deal with the increased risk of committing a type 1 error when we compute multiple tests. For example, the Bonferroni method corrects the p-values by multiplying them by the number of tests that we do. In our case, applying the Bonferroni method on our unadjusted p-values would mean that we should multiply the p-values by 25. This correction makes sure that the risk of making at least one type 1 error is still only 5%:

```
sumdf$pvals_BF=pvals*25 # Bonferroni correction
sumdf_order_p=sumdf[order(sumdf$pvals),]
round(sumdf_order_p,5)
```

Exercise 6: how many genes are differentially expressed if you use a significance level of 0.05 for the Bonferroni corrected p-values?

** Note, some p-values are greater than one after the Bonferroni adjustment. Adjusted p-values greater than one should be set to one since p-values are probabilities that can only vary between zero and one. Also, it is also possible to compare the original p-values to an adjusted significance level (dividing 0.05 by 25 in this example) instead of adjusting the p-values. However, you will end up with the same conclusion in both cases.*

The main problem with the Bonferroni method is that it is very conservative, which means that it corrects the p-values too much, which results in many type 2 errors. Thus, we will end up with many false negatives, where genes that actually are differentially expressed will not be detected. One common method to correct the p-values more gently is the Benjamini-Hochberg method, which is based on the false discovery rate (FDR). The method adjusts the p-values so that the expected proportion of false positives among the total number of positives is controlled below, for example, 5%. We can compute the adjusted p-values based on the Benjamini-Hochberg method by using the function p.adjust:

```
sumdf$pvals_BH=p.adjust(pvals,method="BH")
```

Then we print the data frame again.

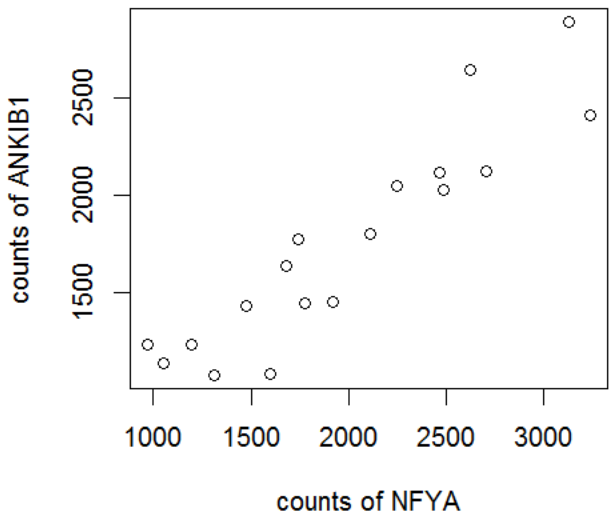
```
sumdf_order_p=sumdf[order(sumdf$pvals),]
round(sumdf_order_p,5)
```

Exercise 7: how many genes are differentially expressed if you use a significance level of 0.05 for the Benjamini-Hochberg corrected p-values?

Correlation

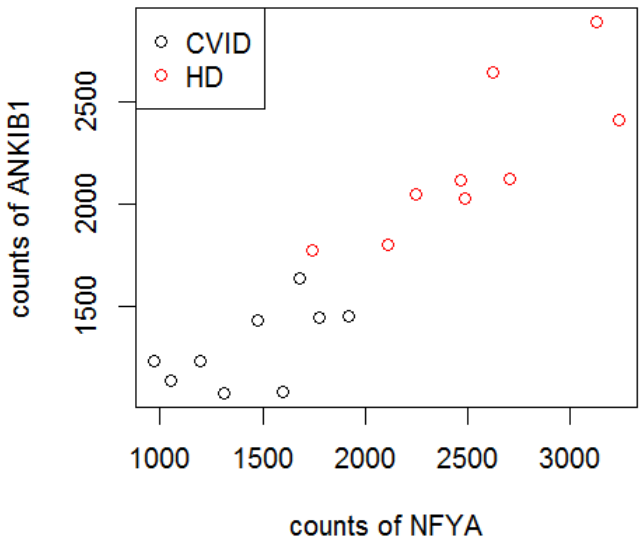
We will now have a look at how we can identify if there are two genes that correlate in their read counts. If two genes show a strong correlation (r > 0.8 or r <-0.8) among the different individuals, this is an indication that the two genes are connected. For example, they might be involved in the same biological process or pathway. Remember from the lectures that the Pearson correlation is very sensitive to outliers. It is therefore important to plot the counts of the two genes with a scatter plot to identify such outliers. We will here check the relationship between the genes "NFYA" and "ANKIB1":

```
plot(tdf$NFYA, tdf$ANKIB1,xlab="counts of NFYA",ylab="counts of ANKIB1")
```



Note that each circle in this plot represents the counts of the two genes for each of the 18 individuals. If we set the argument "col" equal to our "Group" variable, we can also set different colors for the CVID and HD:

```
plot(tdf$NFYA, tdf$ANKIB1,xlab="counts of NFYA",ylab="counts of ANKIB1",col=Group)
legend("topleft", legend=levels(Group), pch=1, col=1:2)
```



From this plot, it is clear that the CVID patients have lower counts than the HD for both genes. We see that no clear outlier is visible in the plot and that we fulfill the assumptions to compute the Pearson correlation:

```
cor.test(tdf$LAS1L,tdf$DPM1,method="pearson")
        Pearson's product-moment correlation
data:  tdf$LAS1L and tdf$DPM1
t = 9.9889, df = 16, p-value = 2.79e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
  0.8144620 0.9733455
sample estimates:
      cor
0.9283347
```

We see that the Pearson correlation coefficient is 0.928 and that the correlation coefficient is significantly different from zero since the p-value is less than 0.05. To compute the non-parametric alternative, the Spearman correlation, we just set the method to "spearman"

```
cor.test(tdf$LAS1L,tdf$DPM1,method="spearman")
```

To compute the correlation between all pairs of genes, we can generate a so-called correlation matrix by (we here use the Spearman correlation because the assumptions for the Pearson correlation will not be fulfilled for all the comparisons):

```
cor(tdf,method="spearman")
```

Exercise 8a: find which pair of genes that shows the strongest negative correlation. Make a scatter plot of these two genes.

It is very hard to read a large correlation matrix. To simplify the analysis, we can convert the matrix into a vector and sort based on the correlation coefficients:

```
cor_mat =cor(tdf,method="spearman")
library(reshape)
diag(cor_mat) <- NA # Set main diagonal to NA
cor_mat[upper.tri (cor_mat)] <- NA # Set upper triangle to NA
cor_melt=melt(cor_mat) # Convert matrix to a vector
cor_melt=na.omit(cor_melt) # Remove NA
cor_melt[order(cor_melt$value,decreasing = TRUE),] # Print the sorted correlations
```

Exercise 8b: find which pair of genes that show the strongest positive correlation. Make a scatter plot of these two genes.

Regression

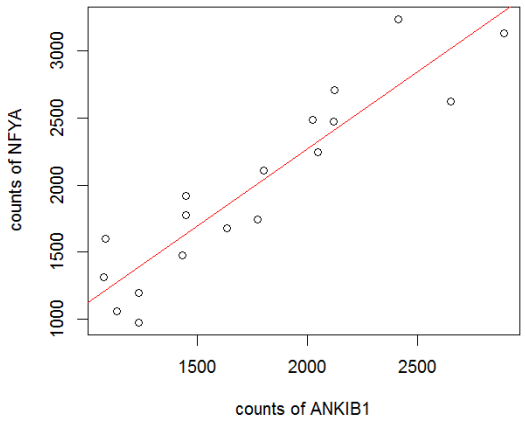
We will now use simple linear regression to predict the expression of the gene "NFYA", given the expression of the gene "ANKIB1". Thus, we will use the gene "NFYA" as the dependent variable and the gene "ANKIB1" as the independent variable:

```
fit=lm(tdf$NFYA~tdf$ANKIB1)
summary(fit)
…
Coefficients:
             Estimate   Std. Error   t value    Pr(>|t|)
(Intercept)  -35.2945    222.2552    -0.159      0.876
tdf$ANKIB1    1.1521     0.1213      9.498     5.6e-08 ***
…
Multiple R-squared:  0.8494,    Adjusted R-squared:  0.8399
…
```

The output tells us that the intercept is -35.29, which means that the gene NFYA has a count that is predicted to be negative when the count of the gene ANKIB1 is zero. This is, of course, not biologically realistic since no gene can have a count less than zero. This is one of the reasons why Poisson regression is more appropriate to use on count data than linear regression. In the output, we also see that the slope is equal to 1.15, which means that the read count of NFYA is increased by 1.15 when the count of the gene ANKIB1 is increased by one. In the last column, we find the p-value. Since the p-value is less than 0.05, we

can conclude that the slope is significantly greater than zero. At the bottom of the output, we find the R-squared value, which tells us that 84.9% of the variation in NFYA can be explained by the expression of the gene ANKIB1. We can plot the data with the regression line by:

```
plot(tdf$ANKIB1,tdf$NFYA,xlab="counts of ANKIB1",ylab="counts of NFYA")
abline(fit,col="red")
```



Multiple linear regression

In the previous example, we used only one explanatory variable to predict the read count of NFYA. Let's add the genes CD99 and BAD to our model to see if they improve the prediction of NFYA. Since we now use more than one explanatory variable, we will perform multiple linear regression:

```
fit=lm(tdf$NFYA~tdf$ANKIB1 + tdf$CD99 + tdf$BAD)
summary(fit)
..
Coefficients:
             Estimate   Std. Error   t value   Pr(>|t|)
(Intercept) -39.80024   421.42093   -0.094      0.926
tdf$ANKIB1    1.19771     0.16213    7.387    3.42e-06 ***
tdf$CD99      0.05691     0.11419    0.498      0.626
tdf$BAD      -0.72058     0.73323   -0.983      0.342
..
Multiple R-squared:  0.8594,    Adjusted R-squared:  0.8293
```

Note that only the gene ANKIB1 has a slope that is significantly different from zero since the genes CD99 and BAD have p-values that are greater than 0.05. Although the R-squared value has increased from 0.849 to 0.859, the increase is relatively small. We can therefore conclude that the genes CD99 and BAD are not very helpful in explaining the variation in the gene NFYA. We can use the "step" function in R, which removes genes that are not likely to contribute in explaining the variation of the dependent variable. This

is done by computing the so-called AIC value. If the removal of one explanatory variable reduces the AIC value, the variable is removed. The variable that generates the lowest AIC value when it is removed will be deleted first. Run the function below and try to understand the output:

```
step(fit,direction ="backward")
```

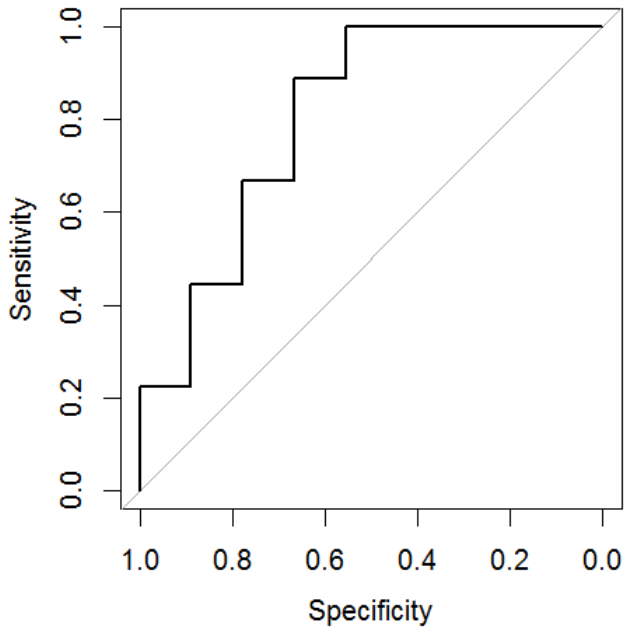
The model that is “the best model” is shown at the bottom of the output.

Exercise 9: which explanatory variable(s) are included in the “best model”?

Sensitivity, Specificity and ROC curves

Let’s say that we would like to evaluate whether a certain gene can be used as a biomarker to predict if a person has CVID. For such a test, one can isolate the naïve B-cells from a blood sample and extract the mRNA from these cells and do a qPCR. To get an estimation for how well such a test would perform by using, for example, the gene “SCYL3”, we could generate an ROC curve for this gene by:

```
library(pROC)
roc1=roc(Group, tdf$SCYL3)
plot(roc1)
```



If we print the output of the roc function, we see that

```
roc1
Data: tdf$SCYL3 in 9 controls (Group CVID) > 9 cases (Group HD).
Area under the curve: 0.8025
```

the area under the ROC curve is about 0.8. We can use a Mann-Whitney U test, to test if the AUC is significantly greater than 0.5:

```
wilcox.test(tdf$SCYL3~Group)
data:  tdf$SCYL3 by Group
W = 65, p-value = 0.03147
alternative hypothesis: true location shift is not equal to 0
```

Since we use a one-sided test, we can divide the above p-value by 2. Since this value (0.016) is less than the significance level, 0.05, we can conclude that the AUC is significantly greater than 0.5. We can also compute a 95% confidence interval:

```
ci.auc(roc1)
95% CI: 0.586-1 (DeLong)
```

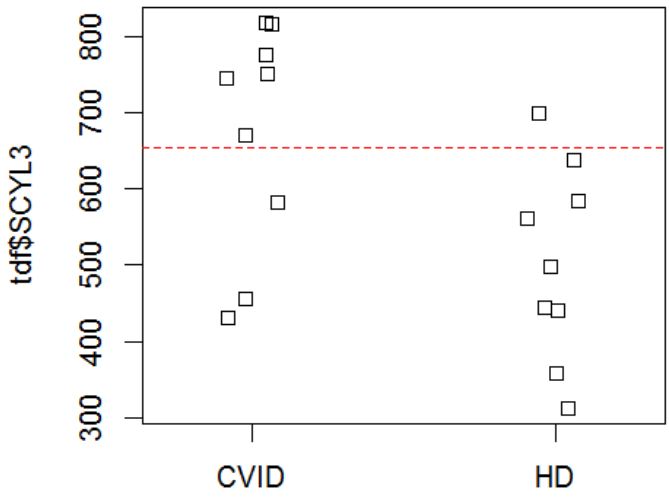
Since this interval does not include the value 0.5, we know that the AUC is significantly greater than 0.5.

We can print the output of the ROC curve to see the sensitivity and specificity for the different cutoff values:

```
data.frame(roc1$thresholds,roc1$sensitivities,roc1$specificities)
   roc1.thresholds roc1.sensitivities roc1.specificities
1              Inf          1.0000000          0.0000000
2            817.0          1.0000000          0.1111111
3            795.5          1.0000000          0.2222222
4            762.5          1.0000000          0.3333333
5            747.5          1.0000000          0.4444444
6            722.0          1.0000000          0.5555556
7            685.0          0.8888889          0.5555556
8            654.5          0.8888889          0.6666667
9            611.0          0.7777778          0.6666667
10           583.5          0.6666667          0.6666667
11           572.0          0.6666667          0.7777778
12           529.0          0.5555556          0.7777778
13           476.5          0.4444444          0.7777778
14           450.0          0.4444444          0.8888889
15           442.5          0.3333333          0.8888889
16           435.5          0.2222222          0.8888889
17           394.5          0.2222222          1.0000000
18           335.5          0.1111111          1.0000000
19            -Inf          0.0000000          1.0000000
```

A cutoff value of 654.5 seems to result in a quite high sensitivity (89%) and specificity (67%). Let's generate a dot plot of the read counts and add a horizontal line for that cutoff value:

```
stripchart(tdf$SCYL3~Group,data=df,vertical = T,xlim=c(0.7,2.3),method="jitter")
abline(h=654.5,lty=2,col="red") # Add a horizontal line to the plot
```



By using a cutoff value of 654.5, we get only one false positive and three false negatives. It seems like the gene SCYL3 could be a good candidate biomarker for predicting CVID.

Exercise 10: Select another gene from the list and create an ROC curve. What results did you get? Could your gene be a potential biomarker for CVID?

Logistic regression

In the previous example, we selected an arbitrary gene to see how well it could discriminate between the CVID patients and the HD. By using logistic regression, we can use the "step" function to identify the "best" biomarker from a list of genes. Also, logistic regression can help us to identify a set of genes that can together be used to discriminate between CVID and HD.

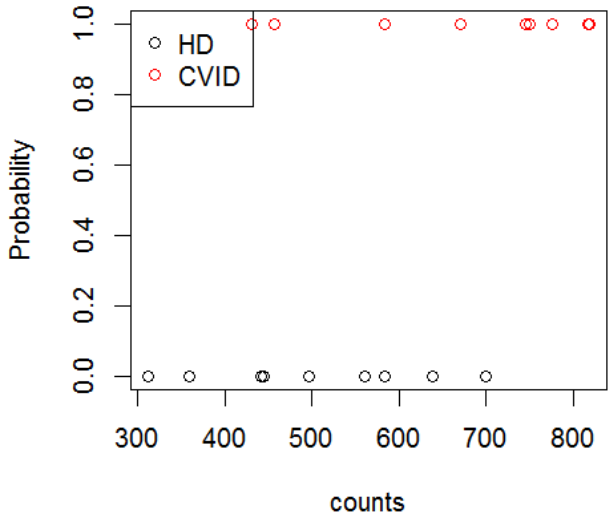
Logistic regression is a regression technique where the dependent variable is on a categorical scale, compared to linear regression, where the dependent variable should have a continuous scale. Logistic regression can be used to predict probabilities and odds ratios. We will here only focus on the binary logistic regression where the dependent variable only has two outcomes; for example, Yes/No, Disease/Healthy, etc. In addition, we will here only use logistic regression to classify based on the estimated probabilities rather than focusing on the estimated coefficients. The predicted values of the dependent variables are probabilities and may therefore only take values between zero and one. When we run logistic regression, we must first determine which of the categories, CVID or HD, that should be set

as the baseline category. Usually, one sets the "healthy/normal" category as the baseline. By setting the category "HD" as the first level in the code below, we make sure that the HD is set as our baseline category:

```
Group=factor(Group,levels=c("HD","CVID"))
```

Let's make a plot where we put the Group variable on the y-axis and the read counts on the x-axis. CVID is here coded as 1 and HD as 0:

```
state=ifelse(Group=="CVID",1,0) # Recode to CVID -> 1, HD -> 0
plot(tdf$SCYL3,state,col=Group,xlab="counts",ylab="Probability")
legend("topleft",levels(Group),col=1:2,pch=1)
```



Note that the dependent variable (the group variable on the y-axis) can only take the values zero (for HD) or one (for CVID). From the figure above, we see that the counts of the SCYL3 gene are a bit higher for the CVID patients compared to the HD. The next step is to fit an s-shaped curve that takes values between zero and one. The logistic regression is usually expressed as the logged odds ratio:

$$\log\left(\frac{p}{1-p}\right) = a + b \cdot x$$

where the right-hand side consists of only linear terms. Solving this function for p (the probability) will result in the logistic function:

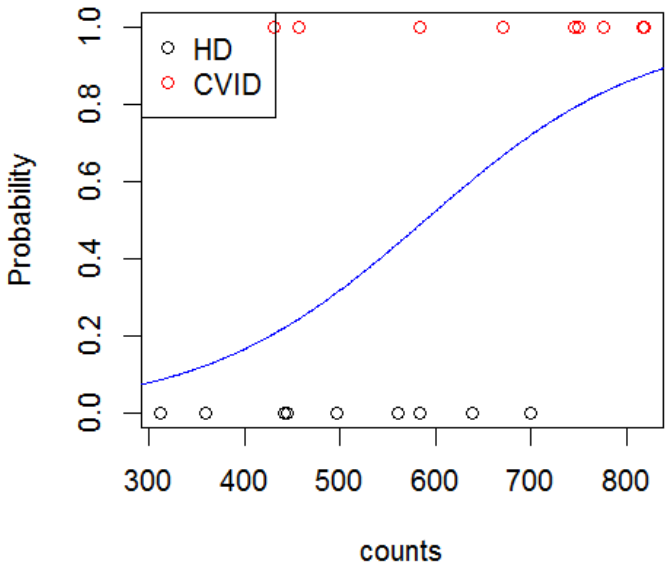
$$p = \frac{1}{1+e^{-(a+b\cdot x)}}$$

This function has an S-shaped (sigmoid) curve that takes values between zero and one. We will now fit this function to the above data, meaning that we will find values of "*a*" and "*b*" that generate a curve that fits the data as good as possible. To fit the logistic function to the data, we use the "glm" function for generalized linear models where we set the argument "family" to "binomial":

```
fit <- glm(Group~SCYL3,data=tdf,family=binomial)
fit
…
Coefficients:
(Intercept)    tdf$SCYL3
 -5.055720     0.008588…
…
```

We see that the function has fitted the coefficients "*a*" (intercept) to -5.05 and "*b*" (SCYL3) to 0.008588. Now, let's plot the predicted probabilities as a function of x (counts of SCYL3) using the estimated coefficients. To plot a smooth curve, we generate a range of possible x-values (reasonable SCYL3 count values).

```
a=coef(fit)[1] # Estimated value of a
b=coef(fit)[2] # Estimated value of b
x=0:1000 # x-values for the curve
p=1/(1 + exp(-(a+b*x)))
lines(x,p,col="blue")
```



Based on the fitted curve, we can now predict the probability of having CVID given a certain count value for the gene SCYL3. For example, a read count of 700 of this gene would result in about 70% probability that a person has CVID. This probability can be calculated by the predict function:

```
predict(fit,data.frame(SCYL3=700),type="response")
```

```
0.7222584
```

Variable selection with logistic regression

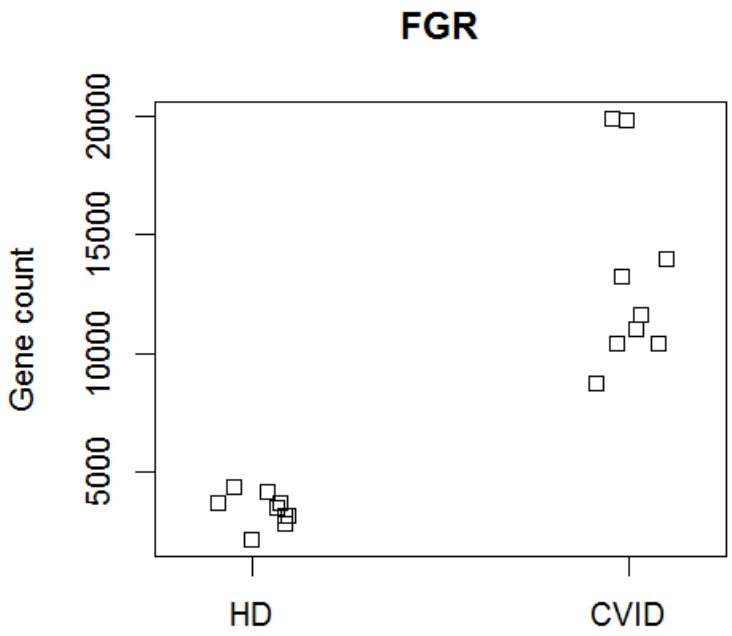
Similar to the multiple regression, we can have more than one explanatory variable in logistic regression, and by using the "step" function, we can find the "best" gene or a set of genes that can discriminate between CVID and HD. In the code below, we use the seven most significant genes in the model:

```
fit <- glm(Group~ANKIB1+NFYA+HS3ST1+NIPAL3+FGR+LAS1L+DPM1,data=tdf,family=binomial)
step(fit, direction="backward")
```

From the output, it seems like the gene FGR is the best gene in discriminating between CVID and HD. Note, you should not use the step function if your number of dependent variables exceeds the sample size. For such cases, one should use a regularization method, such as LASSO regression.

Let's create a dot plot of the expression of this gene:

```
stripchart(tdf$FGR~Group,vertical = T, method="jitter",ylab="Read count",main="FGR")
```



It seems like we have found the winner. The gene expression of FGR shows a complete separation between CVID and HD. Note that FGR had the lowest p-value out of the 25 genes when we ran the t-tests.

Exercise 11: Generate an ROC curve based on the gene FGR and propose an appropriate cutoff value to use in order to discriminate between the HD and CVID patients.

Note, once we have found a potential biomarker, we need to validate its performance by using new individuals and another experimental technique. For example, we might collect 100 new subjects and use qPCR to determine how well the expression of the gene FGR can discriminate between CVID and HD.

Poisson regression

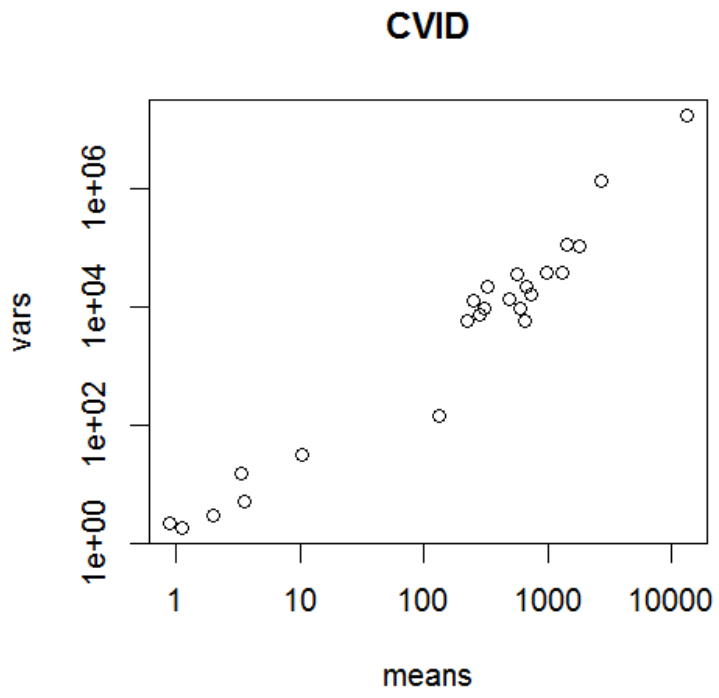
Remember that t-tests are not appropriate to identify differentially expressed genes when we have count data. For RNA-seq data, one usually uses Poisson regression. However, Poisson regression assumes that the variance is equal to the mean according to the Poisson distribution. Let's calculate the variance and the mean for all 25 genes for the CVID patients:

```
vars=apply(tdf[Group=="CVID",],2,var)
means=apply(tdf[Group=="CVID",],2,mean)
data.frame(vars,means) # Print the means and variances
```

Are the variances roughly equal to the mean values?

We can also plot the variances against the means by:

```
plot(means,vars,log="xy",main="CVID")
```



We see that the variances are generally 10-100 fold greater than the means. When the variances are greater than the means, it is called overdispersion. Overdispersion is commonly seen in RNA-seq data. The Poisson regression is therefore not appropriate for RNA-seq data. Instead, Negative Binominal (NB)

regression is used because it can handle overdispersion. We will here use the glm.nb function in the MASS package to run NB-regression on our 25 genes. Let's first run NB-regression on the gene SCYL3:

```
library(MASS)
NB_fit <- glm.nb(SCYL3~Group,data=tdf)
summary(NB_fit)
…
Coefficients:
             Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)  6.22236    0.07819      79.580    < 2e-16 ***
GroupCVID    0.28724    0.11033      2.604     0.00923 **
…
```

We see that the p-value (0.009) is less than 0.05, which means that the gene is significantly differentially expressed. We will now use the apply function for all 25 genes and save their p-values:

```
pval=apply(tdf,2,function(x) {summary(glm.nb(x~Group,data=tdf))$coefficients[2,4]})
```

When you run this line, you will get warnings, which is due to that the gene CFTR only has zero counts. One therefore usually removes such low-count genes before analyzing the data.

```
tdf$CFTR
```

We then adjust the p-values by the Benjamini-Hochberg method, and add these adjusted p-values to our previous data frame (sumdf):

```
sumdf$pvals_BH_NB=p.adjust(pval,method="BH")
```

Finally, we print the genes based on sorted p-values:

```
sumdf_order_p=sumdf[order(sumdf$pvals),]
round(sumdf_order_p,5)
```

	Mean_CVID	Mean_HD	log2FC	pvals	pvals_BF	pvals_BH	pvals_BH_NB
ANKIB1	1302.77778	2203.88889	-0.75846	0.00003	0.00087	0.00036	0.00000
NFYA	1442.11111	2527.44444	-0.80950	0.00006	0.00138	0.00036	0.00000
HS3ST1	573.22222	121.22222	2.24144	0.00006	0.00154	0.00036	0.00000
NIPAL3	989.22222	546.77778	0.85534	0.00007	0.00163	0.00036	0.00000
FGR	13249.33333	3399.00000	1.96274	0.00007	0.00180	0.00036	0.00000
LAS1L	1827.11111	4358.88889	-1.25440	0.00016	0.00400	0.00067	0.00000
DPM1	490.55556	3072.11111	-2.64674	0.00035	0.00869	0.00124	0.00000
CYP51A1	10.44444	52.11111	-2.31886	0.00149	0.03721	0.00455	0.00000
LAP3	659.00000	1405.22222	-1.09245	0.00164	0.04098	0.00455	0.00000
ENPP4	330.66667	1404.88889	-2.08701	0.00204	0.05104	0.00510	0.00000
FUCA2	252.55556	114.22222	1.14476	0.00631	0.15781	0.01435	0.00000
SEMA3F	1.11111	66.11111	-5.89482	0.00888	0.22211	0.01851	0.00000
SCYL3	671.55556	503.88889	0.41440	0.02115	0.52873	0.04067	0.01774
CD99	2744.22222	1908.66667	0.52383	0.06709	1.67729	0.11197	0.03987
MYH16	2.00000	0.66667	1.58496	0.06718	1.67951	0.11197	0.05278
CFH	3.33333	0.77778	2.09954	0.08727	2.18175	0.13636	0.11240
STPG1	134.77778	112.00000	0.26708	0.10936	2.73393	0.16082	0.10785
RAD52	735.11111	641.66667	0.19614	0.17804	4.45096	0.24728	0.17695
AOC1	0.88889	0.22222	2.00000	0.23635	5.90873	0.31099	0.30968
GCLC	593.44444	541.44444	0.13230	0.27314	6.82862	0.34143	0.30968
CFTR	0.00000	1.00000	-Inf	0.34659	8.66484	0.41261	1.00000
KRIT1	284.44444	308.44444	-0.11686	0.48968	12.24205	0.55646	0.58229
BAD	305.44444	271.33333	0.17084	0.57372	14.34308	0.62361	0.58229
TSPAN6	3.55556	2.88889	0.29956	0.61504	15.37610	0.64067	0.66163
C1orf112	221.00000	208.77778	0.08208	0.76275	19.06873	0.76275	0.80954

Exercise 12: Compare the BH-adjusted p-values (last two columns) from the t-tests and the NB-regression. What is the major difference?

A tutorial on clustering - optional

We will use our previous RNA-seq data to cluster the genes based on similar expressions across the 18 individuals. The first step in hierarchical clustering is to compute the distances between the genes. We will here calculate the Euclidean distance between the genes by the function "dist"

```
h=dist(df, method = "euclidean")
round(h)
          TSPAN6   DPM1   SCYL3   C1orf112   …
DPM1      10039
SCYL3      2566     8515
C1orf112    961     9437    1754
FGR       42647    40164   40559   41928
..
```

As you see, the function generates a distance matrix, which shows the distance between every pair of genes. For example, the Euclidean distance between the gene DPM1 and TSPAN6 is 10039. How is this value calculated? The Euclidean distance is calculated by the following formula:

$$d = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

where *q* and *p* in our example is the count of our two genes DPM1 and TSPAN6, whereas *i* represents the index of the 18 individuals. Thus, the Euclidean distance, in our example, is the square-root of the sum of the squared differences between the counts of the two genes for all the 18 individuals. We can compute this distance manually in R by:

```
p=tdf$TSPAN6  # Extract counts for the gene TSPAN6
q=tdf$DPM1    # Extract counts for the gene DPM1
sqrt(sum((p-q)^2))
10038.81
```

Exercise 1: Pick another pair of genes and compute the Euclidean distance. Make sure that the value you get corresponds to the value in the distance matrix (h).

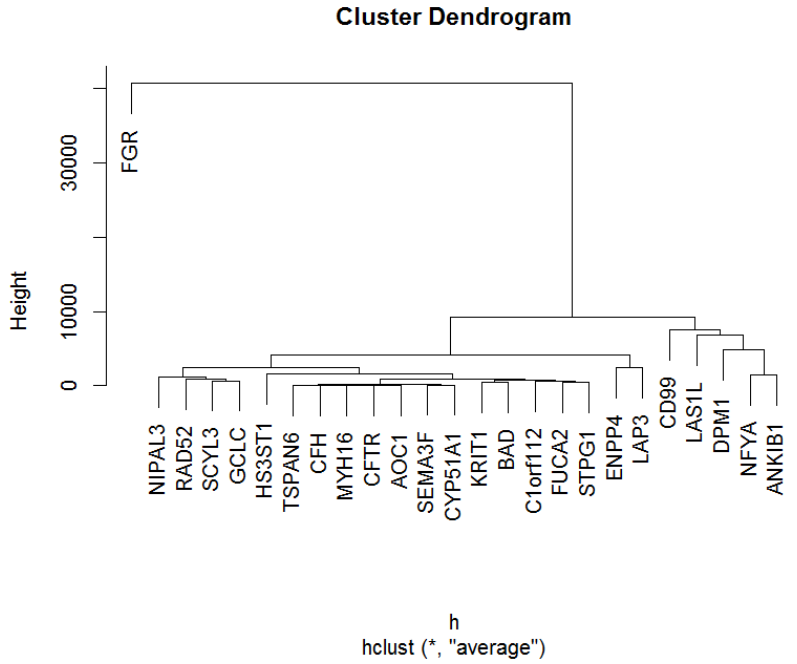
Exercise 2: Find a pair of genes that have a very short distance to each other? Remember this pair and see if you can find this pair in the dendrogram that we will generate later.

Once we have computed all the distances between the genes, we next cluster the genes that have the shortest distance to each other. However, when we combine two clusters with several genes, we need to decide if we should use the mean distances between the genes in the cluster, or the maximum distances between the clusters (complete-linkage clustering) or the minimum distance (single-linkage clustering). However, we will here use the mean/average linkage function. To cluster the genes based on the average linkage function, we can run the following code:

```
hp=hclust(h,method="average")
```

We can now generate a dendrogram for this output by:

```
plot(hp)
```



The height represents the Euclidean distance between the clusters.

Exercise 3: Can you find your pair of genes, from exercise 2, in the dendrogram above?

Exercise 4: Look at the count data (in "df") and try to understand why the gene FGR is so far away from the other genes in the dendrogram.

The problem with the above dendrogram is that it is based on the distance between the actual count data. Genes with a low expression will cluster because the distances between the small values are shorter. The gene FGR has very high count values in comparison to the other genes, which means that the distance to the other genes is very far. The previous method therefore clusters genes that have similar values (counts). We usually like to cluster genes that have a similar profile among the individuals (similar relative expression levels). To remove the impact of the actual count values, we can normalize the values so that all genes have a mean of zero and a standard deviation of one. This is called standardization. To standardize the counts of each gene (x), we subtract each read count value (x_i) by the mean value ($\bar{x}$) of the gene count based on all individuals, and divide this by the standard deviation (s) of that gene:

$$z_i = \frac{x_i - \bar{x}}{s}$$

After standardizing the variable (the gene), the standardized variable (Z), will have a mean of zero and a standard deviation of one. To standardize our data, we can use the function "scale", which standardizes each column. Remember that "tdf" has the genes in the columns, so if we use function "scale" on the "tdf" object, we will standardize the values in the data frame based on the genes:

```
stdf=data.frame(scale(tdf))
```

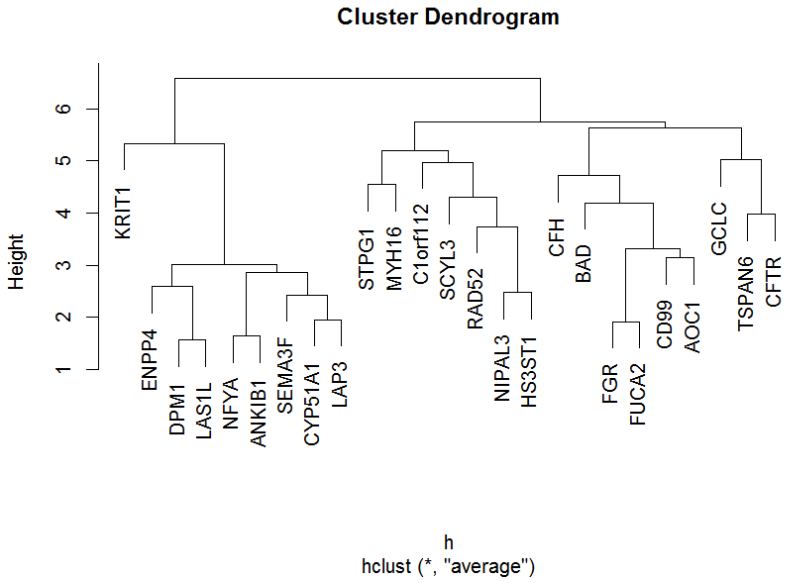
Our standardized values are now stored in the data frame (stdf). To check that we have done the correct calculations, let's check the mean and the standard deviation of an arbitrary gene:

```
mean(stdf$AOC1)
-5.551115e-17
sd(stdf$AOC1)
1
```

It seems like it is working, the mean of this gene is basically zero and the standard deviation is 1. Try another gene and make sure that you get the same results.

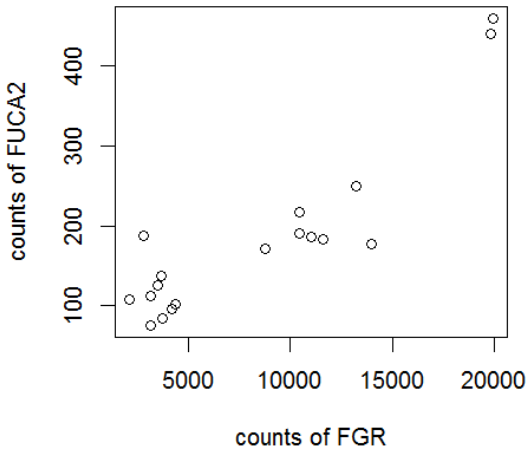
Let's cluster the genes based on the standardized data:

```
h=dist(t(stdf), method = "euclidean")
hp=hclust(h,method="average")
plot(hp)
```



The clustering is no longer based on the actual count values; instead, it is based on the relative differences in the gene expression between the individuals. For example, we see that the gene FGR now forms a cluster with FUCA2. Let's generate a scatter plot of the counts of these two genes:

```
plot(tdf$FGR, tdf$FUCA2,xlab="counts of FGR",ylab="counts of FUCA2")
```



Note that the counts of FGR are much greater than the counts of FUCA2. Thus, using the Euclidean distance on the original counts, these two genes would be considered to be far away. However, when we use the Euclidean distance on the standardized data, these two genes cluster because their relative expressions are similar in the 18 individuals. For example, if the gene is highly expressed in one individual, the other gene is also highly expressed.

Use the correlation coefficient as a distance

Another way to cluster genes is to compute the correlation coefficient between each pair of genes. This will result in quite a similar result as when we used the Euclidean distance on the standardized data because correlation also involves normalization of the values (remember that the correlation coefficient spans from minus one to one). If two genes have a correlation coefficient close to 1, they have a very strong positive correlation. If we take one minus the correlation coefficient, a strong positive correlation will result in a value close to zero. This is exactly what we want because two genes that have a very strong positive correlation should have a short distance to each other and cluster together. If we run the following code,

```
cor_dist=1-cor(tdf)
```

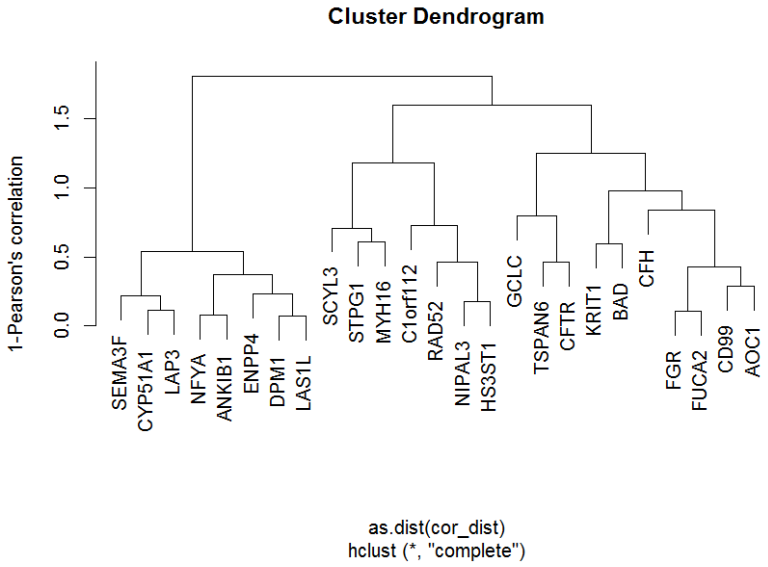
we will get a matrix with one minus the Pearson correlation coefficient (you can also use the Spearman correlation if you like). Next, we print the corresponding distance matrix by:

```
as.dist(cor_dist)
```

Note that the values vary between zero and two. A value close to zero tells us that the two genes have a very strong positive correlation, a value close to one – no correlation, and a value close to two – a very

strong negative correlation. Finally, we perform clustering based on the Pearson correlation coefficient as the distance measure:

```
clust=hclust(as.dist(cor_dist))
plot(clust,ylab="1-Pearson's correlation")
```



Note that the y-axis now represents one minus the Pearson correlation coefficient.

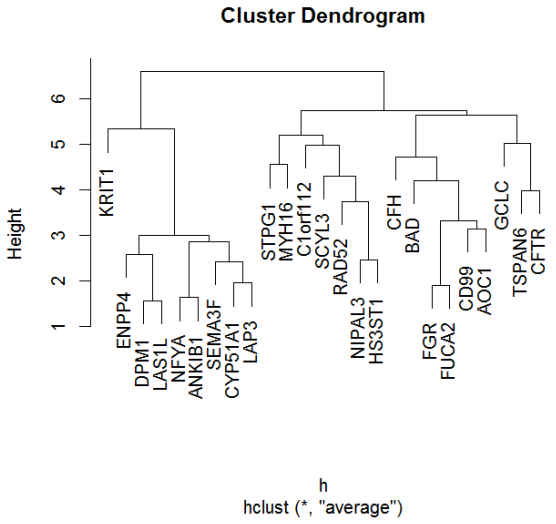
Exercise 5: Compare all three dendrograms that you have generated so far and identify differences.

Cluster the individuals

So far, we have only clustered the genes. However, we can easily cluster the individuals by simply transposing (t) our previous data frame (or not transposing as in the example below). In the following example, we will use the Euclidean distance on the standardized data.

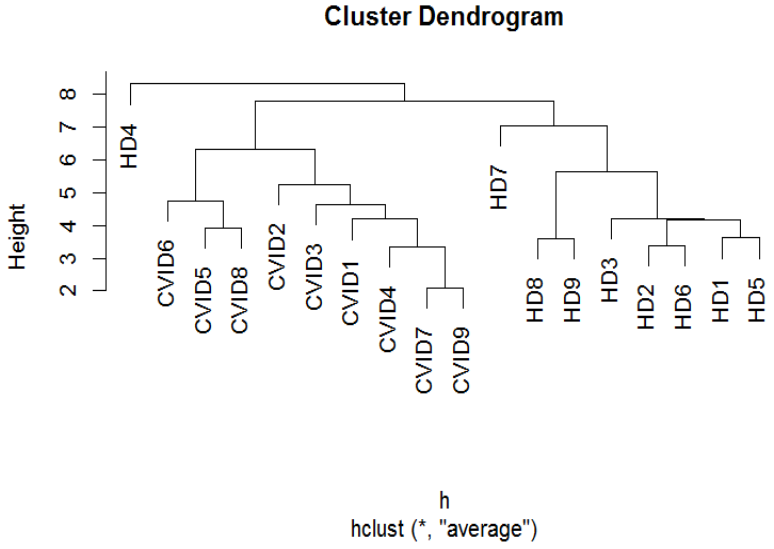
Cluster genes

```
h=dist(t(stdf), method = "euclidean")
hp=hclust(h,method="average")
plot(hp)
```



Cluster the individuals (do not transpose the data frame)

```
h=dist(stdf, method = "euclidean")
hp=hclust(h,method="average")
plot(hp)
```

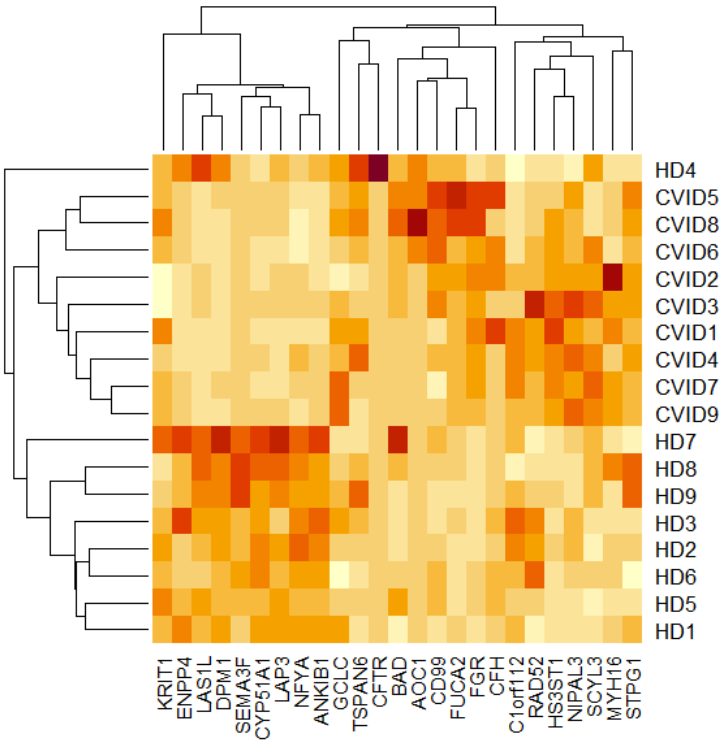


Exercise 6: Do the CVID patient and HD form two distinct clusters? Can you find an individual that has a gene profile that deviates a lot from the other?

Heatmap

We can combine the two clusters based on the individuals and the genes in one figure, where we also show the standardized expression values of the genes:

```
heatmap(as.matrix(stdf),scale="none",hclustfun=function(d) hclust(d, method="average"))
```



Note that the rows have been ordered according to how the individuals cluster (how similar their gene expression is) and that the columns are ordered based on the similarity of the relative expressions of the genes. The colors in the heatmap represent high numbers with dark red color and low numbers with white color. For example, it seems like the HD have a high expression of the genes KRIT1, ENPP4, LAS1, DPM1, SEMA3F, CYP51A1, LAP3, NFYA and ANKIB1, whereas the CVID patients have relatively low expression of these genes. The reason why HD4 is a bit off from the other HD is its very high expression of the CFTR gene.

Exercise 7: Try the code below, using the original data, and explain why the heatmap looks very different from the one above.

```
heatmap(as.matrix(df),scale="none",hclustfun=function(d) hclust(d, method="average"))
```