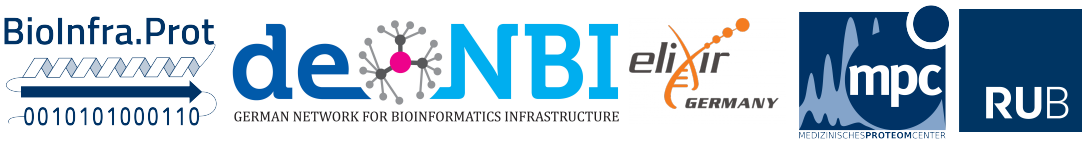
Advanced analysis of quantitative proteomics data using R

Solution

Michael Turewicz, Karin Schork

November 9, 2020



Contents

1 Hands-on session I: R packages and data preprocessing

General hints:

- look at the slides to solve the exercises
- write down your code in an editor (e.g. the upper left window of RStudio)
- also look at the help pages of the mentioned R functions

Exercise 1.1: Installing and using R packages

Please start your R console and set your current R repositories to "CRAN" and "BioC software". Then, choose a location near your current whereabouts as your CRAN as well as your Bioconductor mirror. Install the following packages, that you will need in the following exercises:

- openxlsx (for reading and writing xlsx files)
- limma (for normalization)
- affy (for MA-Plots)
- gplots (for heatmap)
- pROC (for ROC curves)

Please check whether you have correctly installed the packages by loading them and calling their help pages.

```
> # use "CRAN" and "BioC software" as repositories
> # first, use setRepositories() to find the repository indices for 'ind'
> setRepositories(ind = c(1,2))

> # example: use Muenster as CRAN mirror
> # first, use chooseCRANmirror() to find the mirror index for 'ind'
> chooseCRANmirror(ind = 37)

> # example: use Tokyo as BioC mirror
> # first, use chooseBioCmirror() to find the mirror index for 'ind'
> chooseBioCmirror(ind = 47)

> #install packages including dependencies
> install.packages(c("openxlsx", "limma", "affy", "gplots", "pROC"),
+                  dependencies = TRUE)
```

```
> #load packages and check help pages
> library(openxlsx)
> library(limma)
> library(affy)
> library(gplots)
> library(pROC)

> help(package="openxlsx")
> help(package="limma")
> help(package="affy")
> help(package="gplots")
> help(package="pROC")
```

Exercise 1.2: Missing values

In the following hands-on sessions we will use a dataset containing 19 samples of Hepatocellular Carcinoma (HCC) and 19 samples of corresponding nontumorous tissue. For more information on the study see

Wael Naboulsi et al. Quantitative Tissue Proteomics Analysis Reveals Versican as Potential Biomarker for Early-Stage Hepatocellular Carcinoma Journal of Proteome Research 2016 15 (1), 38-47 DOI: 10.1021/acs.jproteome.5b00420

Import the dataset `HCC_19vs19_raw_abundances.xlsx` (unnormalized data) into R using the function `read.xlsx` from the R package `openxlsx`. Alternatively you can import the csv file.

Write the first 9 columns from the data set (containing protein accessions and other information) in a different dataframe for later use and delete them from the original dataset. Overwrite all zero values in the remaining dataframe with `NA`.

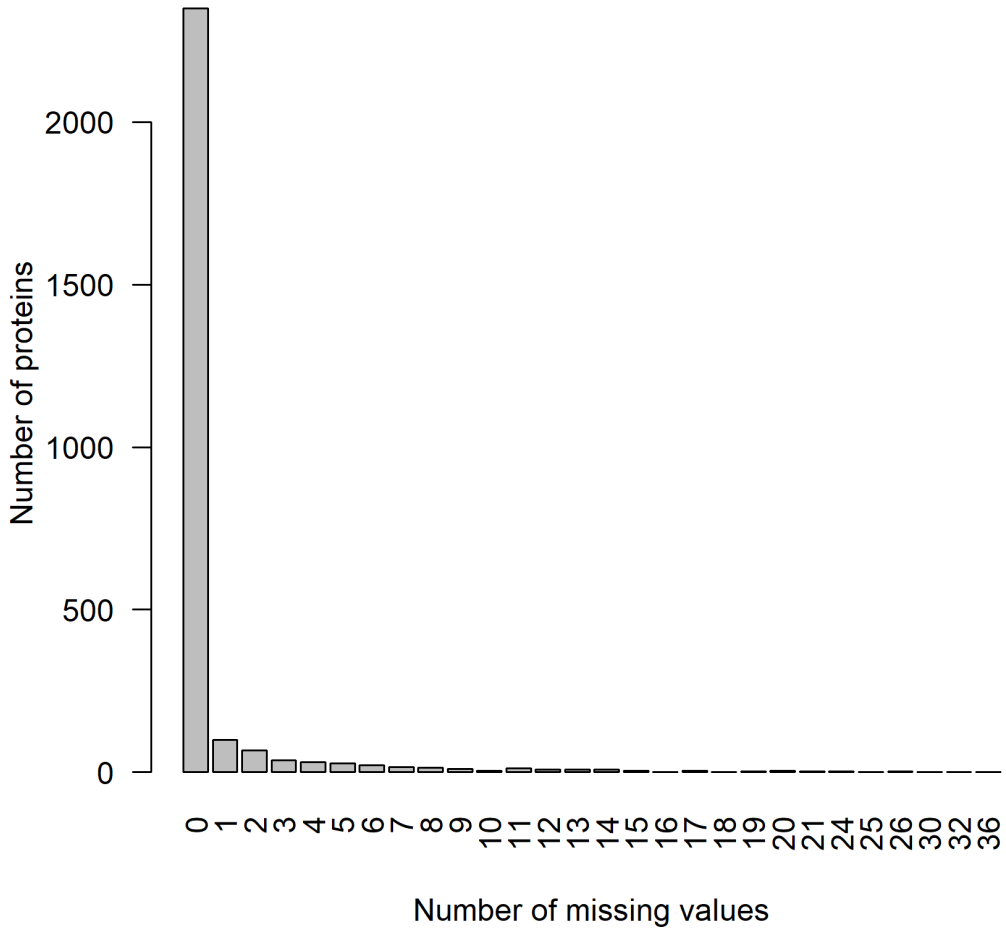
Answer the following questions:

- How many missing values are there in the dataset in total?
- How many proteins have no missing values? Create a barplot showing how many proteins exists with different numbers of missing values (hint: use the function `table()`).
- Which sample has the most missing values and how many? Create a barplot showing the number of missing values for each sample.

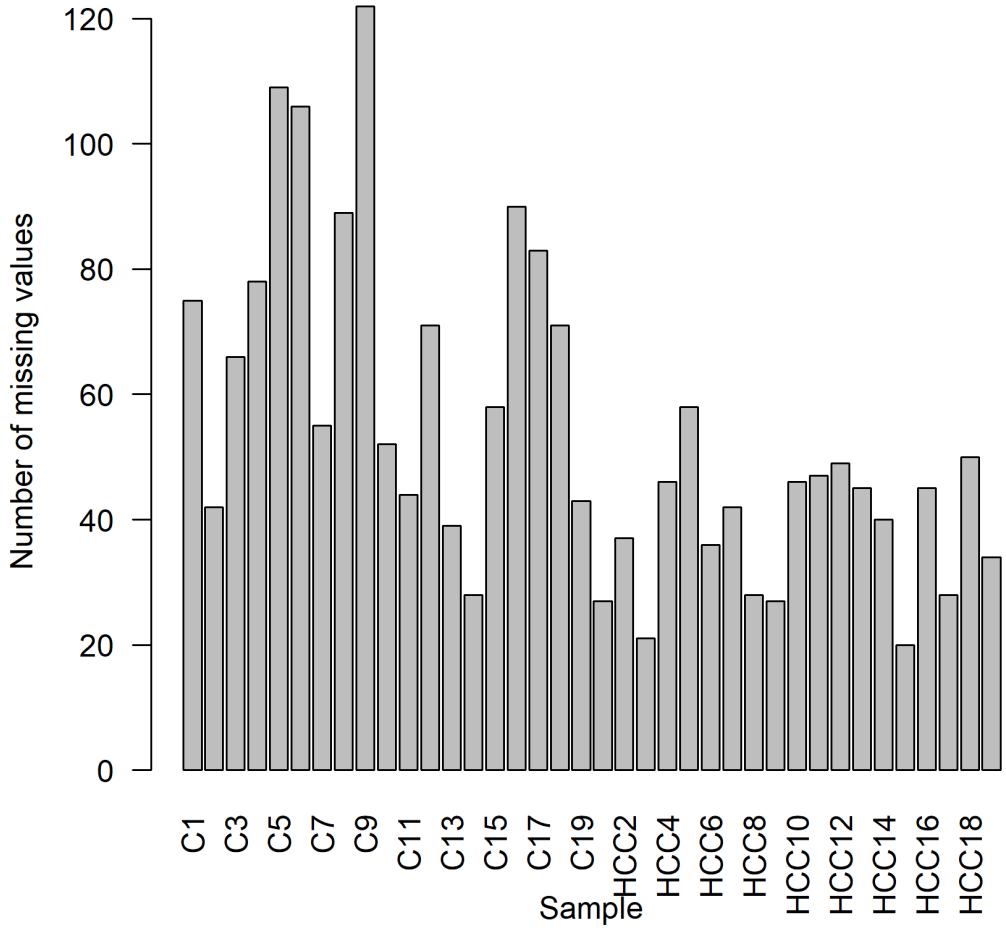
Hint: the funtions `rowSums()` and `colSums()` may be useful.

```
> #import data (as  csv file)
> wd <- "C:/USB_drive/data/"
> setwd(wd)
> dat <- read.csv(file="HCC_19vs19_raw_abundances.csv")
```

```
> # alternative import as xlsx file using the openxlsx package
> library(openxlsx)
> dat <- openxlsx::read.xlsx(file="HCC_19vs19_raw_abundances.xlsx")

> # save first 9 columns in seperate dataframe (accessions) and delete them
> # from dat
> accessions <- dat[, c(1:9)]
> dat <- dat[, -c(1:9)]
> # replace all 0s with NA
> dat[dat == 0] <- NA
> # total number of missing values in the dataset:
> sum(is.na(dat))

[1] 2047

> # number of missing values per protein (row wise):
> nr_na_row <- rowSums(is.na(dat))
> # number of proteins with 0 missing values:
> sum(nr_na_row == 0)

[1] 2351

> # table and barplot of numbers of missing values
> tab <- table(nr_na_row)
> barplot(tab, xlab = "Number of missing values", ylab = "Number of proteins",
+         las = 2)
```




```
> # number of missing values per sample (column-wise)
> nr_na_col <- colSums(is.na(dat))
> # which sample has the most missing values?
> which.max(nr_na_col)

C9
 9

> # show number of missing values per sample in a barplot
> barplot(nr_na_col, xlab = "Sample", ylab = "Number of missing values",
+         names.arg = names(nr_na_col), las = 2)
```



Exercise 1.3: Normalization

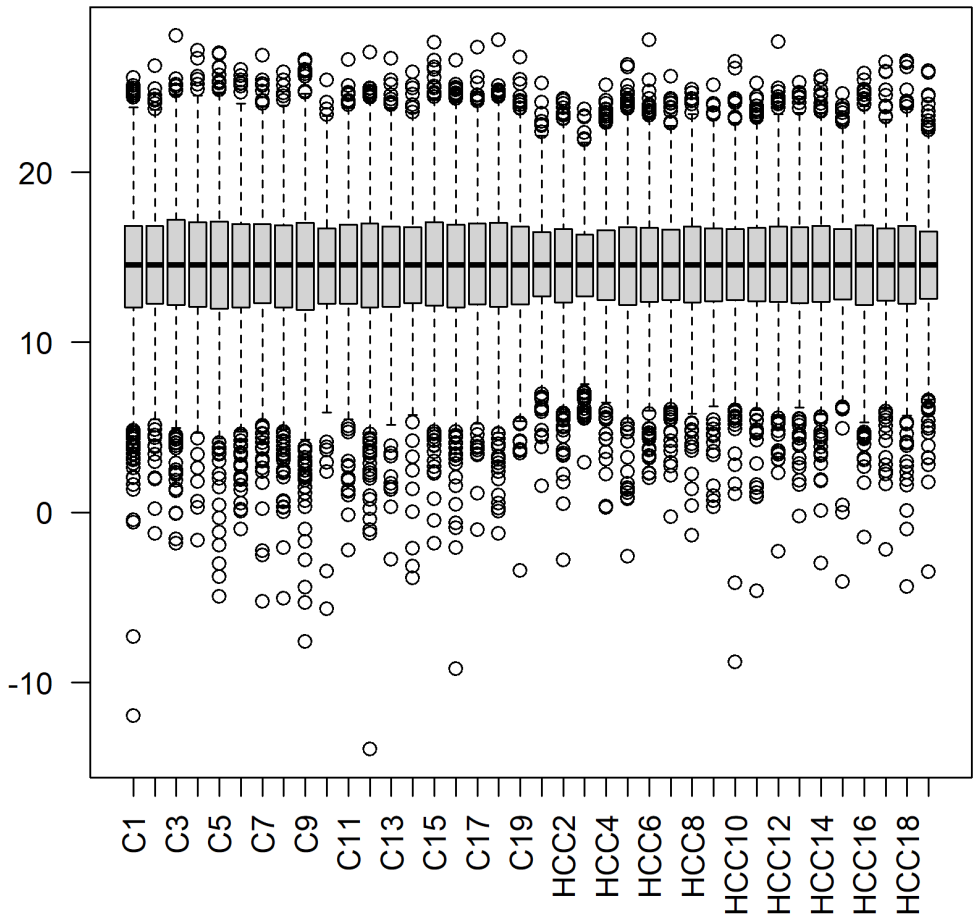
Use the function `normalizeBetweenArrays` from the `limma` package to normalize the dataset from Exercise 1.2. Apply a log2-transformation before normalizing. Create three new datasets, containing the median, quantile and LOESS normalized data.

Compare the three normalization methods with the non-normalized data by creating boxplots and MA-plots. For the MA-plots, use the function `MAPlots` given in the file `MA_Plots.R` (the R package `affy` needs to be installed). The file for this can be found in the exercise folder of the material. Look at the script for information the arguments.

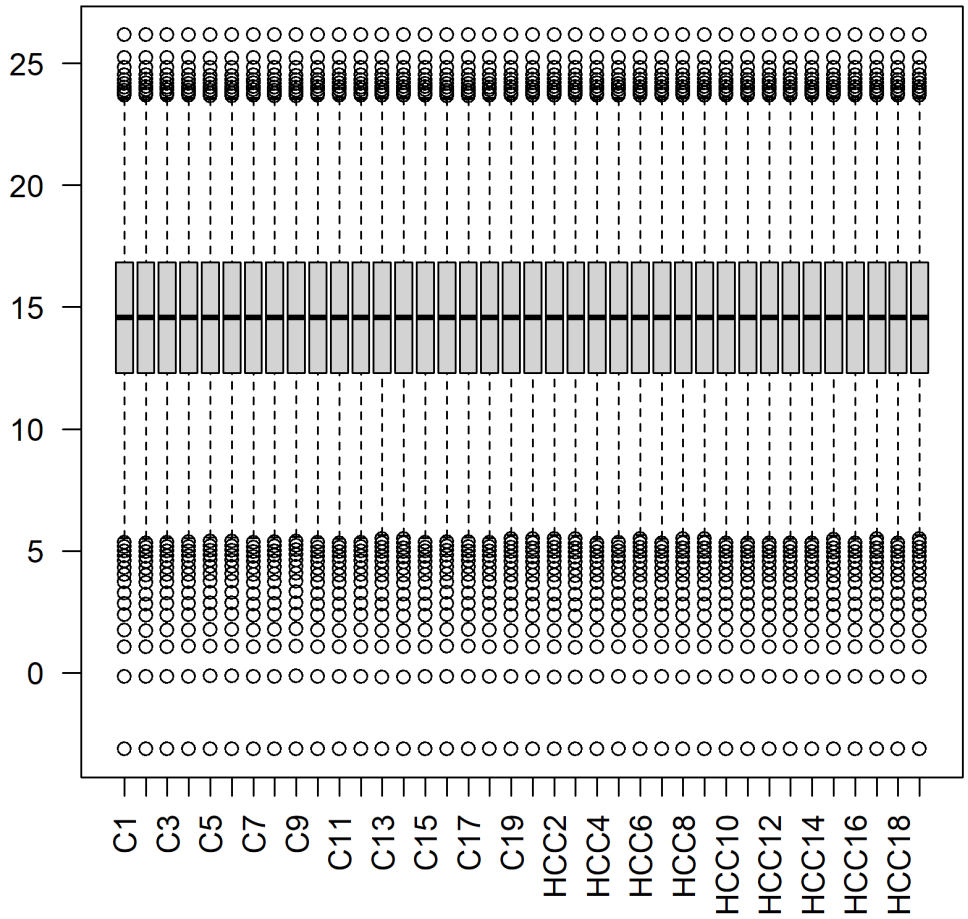
As the normalization was conducted on log2-transformed data, the argument `log` of the function must be set to `FALSE`.

Which is the most suitable normalization method for this dataset?

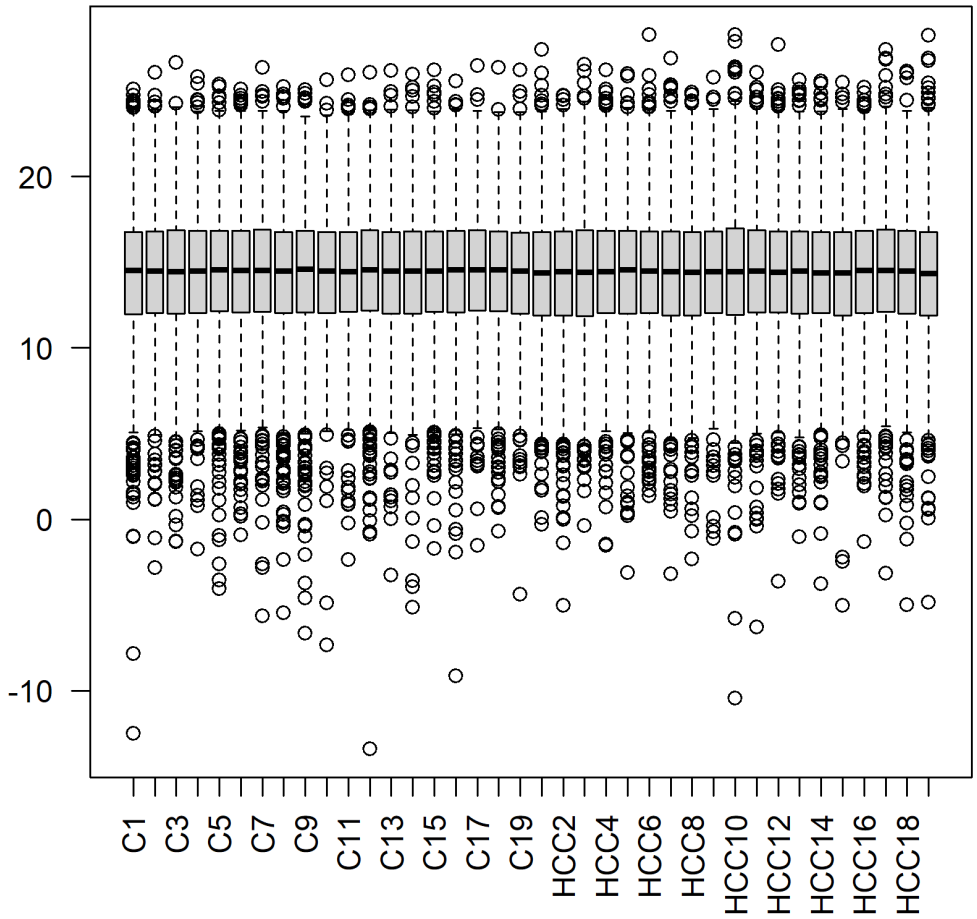
```
> library(limma)
> ## apply three different normalization methods to the data
> dat_median <- limma::normalizeBetweenArrays(log2(dat), method = "scale")
> dat_quantile <- limma::normalizeBetweenArrays(log2(dat), method = "quantile")
> dat_loess <- limma::normalizeBetweenArrays(log2(dat), method = "cyclicloess")
> # boxplot of the non-normalized data
> # (las = 2 means to rotate the x-axis labels for better readability)
> boxplot(log2(dat), las = 2)
> # show the resulting normalized data in boxplots (no log-transformation needed)
> boxplot(dat_median, las = 2)
```




```
> boxplot(dat_quantile, las = 2)
```




```
> boxplot(dat_loess, las = 2)
```




```
> # source() runs the R-script MA_Plots.R which contains a self-written function
> # for MA-Plots.
> source("MA_Plots.R")
> ### MA-Plots of non-normalized data (log-transformation needed)
> MAPlots(dat, log = TRUE, file="MAPlots_nonorm.pdf")
> ## MA-Plots of normalized datasets (no log-transformation needed
> ## as the data is already on log-scale)
> MAPlots(dat_median, log = FALSE, file="MAPlots_median.pdf")
> MAPlots(dat_quantile, log = FALSE, file="MAPlots_quantile.pdf")
> MAPlots(dat_loess, log = FALSE, file="MAPlots_loess.pdf")
>
> # For the non-normalized data you can clearly see the shifts in
```

```
> # the MA-Plots for many of the sample pairs.
> # With Quantile normalization, the line in the MA-Plots shows the least
> # deviation from M = 0. This seems to be the best choice for this
> # dataset.
> # The LOESS-Normalization shows sometimes small deviations from M=0,
> # especially for the low abundant proteins.
> # The median normalization does not seem to be suitable here as many
> # MA-Plots show a increasing or decreasing regression line, i.e. the
> # MA-Plots are not symmetric.
>
> # In summary, the quantile normalization seems to be best suited for
> # this dataset.
```

2 Hands-on session II: Clustering

Exercise 2.1: Clustering

Perform hierarchical clustering on the quantile normalized dataset using the function `hclust`. Compare different combinations of the distances and linkage methods by looking at the corresponding dendrograms. How does only using the 20 most differential proteins (by means of p-value) effect the clustering results?

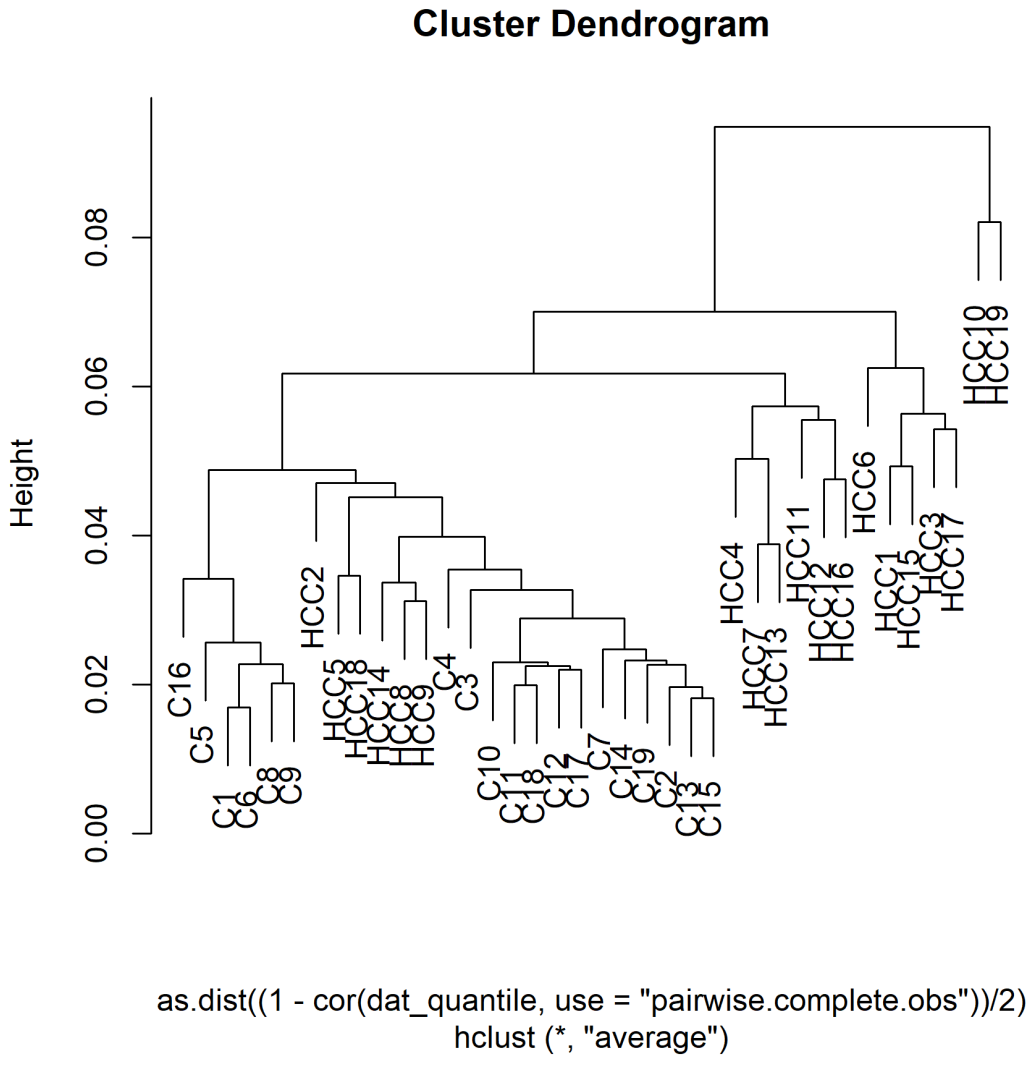
As linkage methods test `"average"`, `"complete"` and `"single"`. As distances compare `"euclidean"`, `"manhattan"` and your self-defined correlation-based distance. For an arbitrary dataset x latter can be defined as follows:

```
> as.dist((1-cor(x, use = "pairwise.complete.obs"))/2)
```

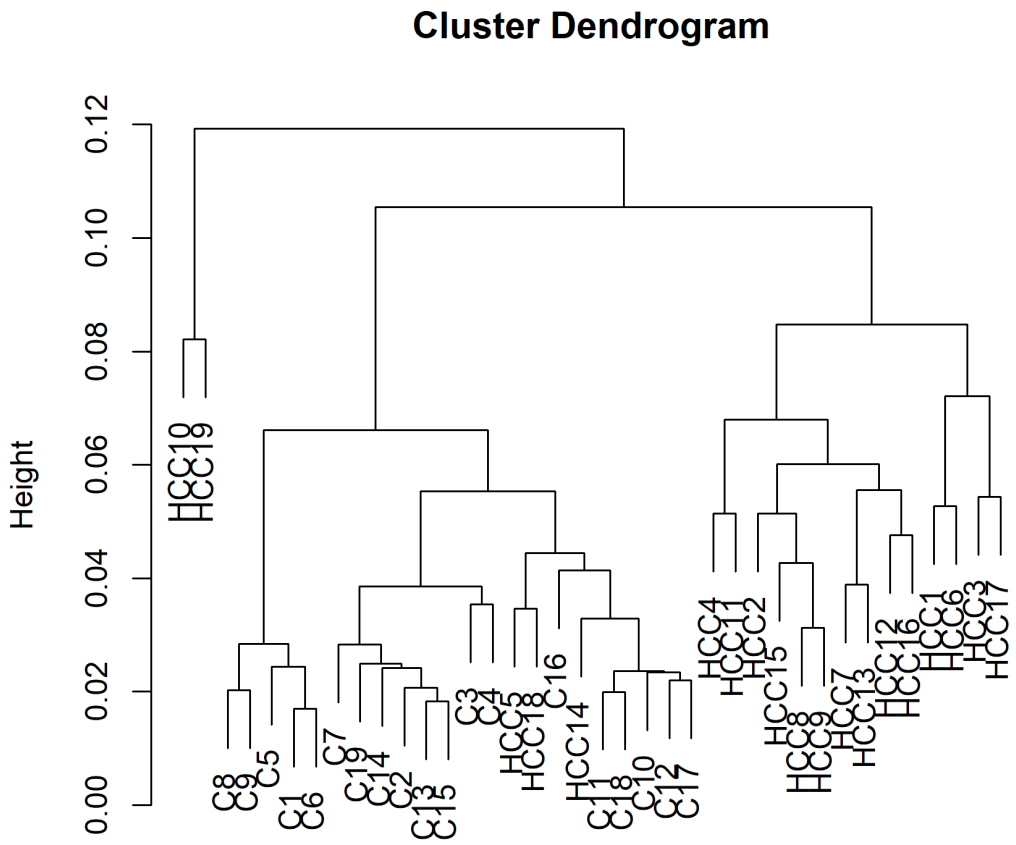
Hint: Read the help pages of the functions `dist`, `as.dist` and `hclust`! Don't forget to check whether you have to transpose your data matrix first (using the function `t`)!

```
> #euclidean and average
> ### data has to be transposed before applying the dist() function!
> hc <- hclust(d=dist(x=t(dat_quantile), method="euclidean"), method="average")
> plot(hc)

> #euclidean and complete
> hc <- hclust(d=dist(x=t(dat_quantile), method="euclidean"), method="complete")
> plot(hc)

> #euclidean and single
> hc <- hclust(d=dist(x=t(dat_quantile), method="euclidean"), method="single")
> plot(hc)

> #manhattan and average
> hc <- hclust(d=dist(x=t(dat_quantile), method="manhattan"), method="average")
> plot(hc)
```

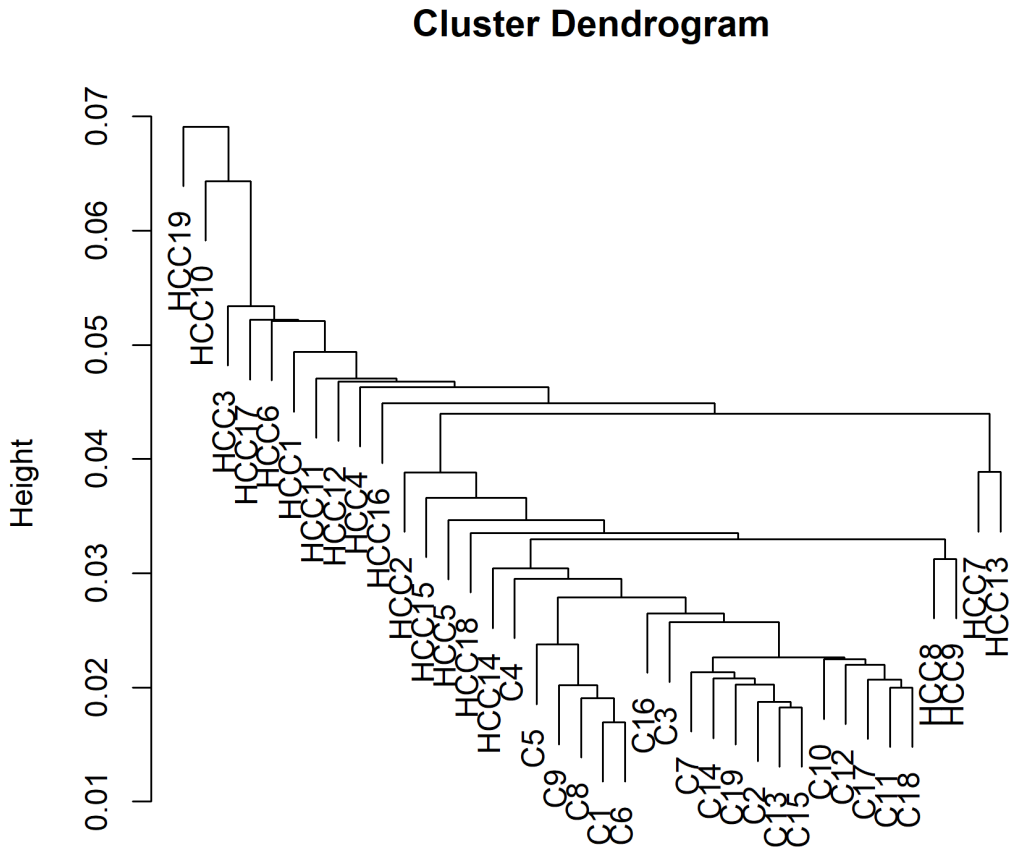
```
> #manhattan and complete
> hc <- hclust(d=dist(x=t(dat_quantile), method="manhattan"), method="complete")
> plot(hc)

> #manhattan and single
> hc <- hclust(d=dist(x=t(dat_quantile), method="manhattan"), method="single")
> plot(hc)

> #correlation-based and average
> # use as.dist() to use a self-defined distance function
> hc <- hclust(d=as.dist((1-cor(dat_quantile, use = "pairwise.complete.obs"))/2),
+              method="average")
> plot(hc)
```



```
> #correlation-based and complete
> hc <- hclust(d=as.dist((1-cor(dat_quantile, use = "pairwise.complete.obs"))/2),
+              method="complete")
> plot(hc)
```




```
> #correlation-based and single
> hc <- hclust(d=as.dist((1-cor(dat_quantile, use = "pairwise.complete.obs"))/2),
+              method="single")
> plot(hc)
```



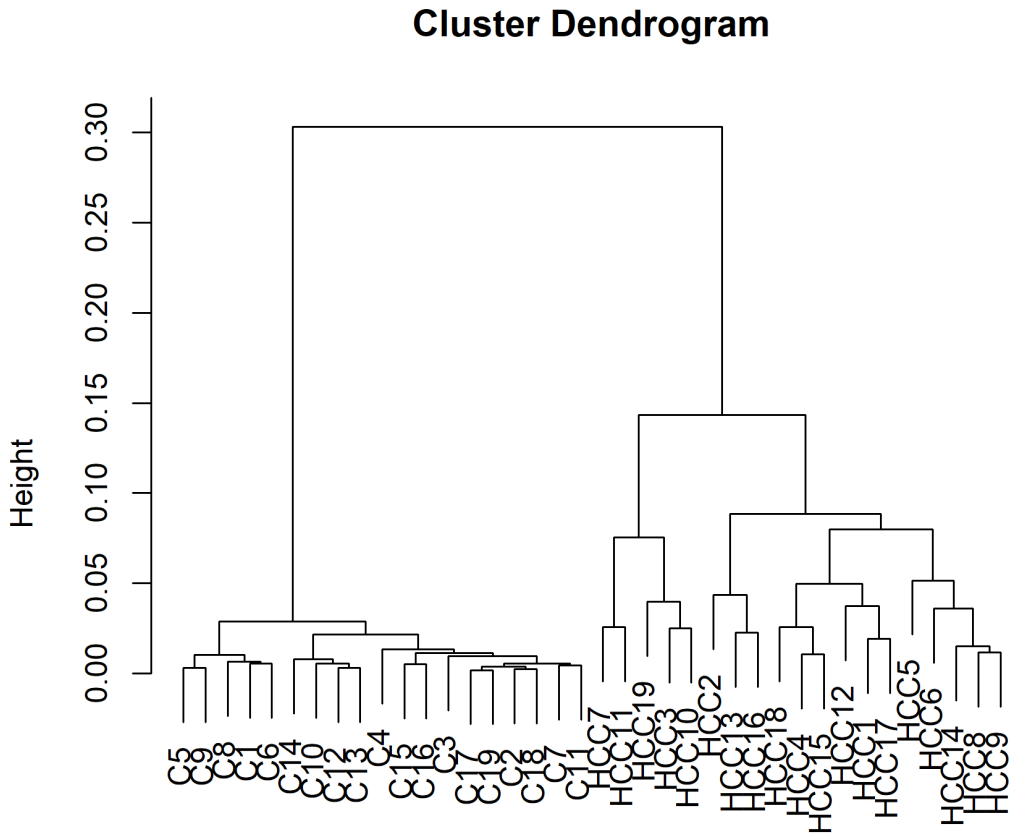
general observation while comparing linkage methods:

- complete: at the bottom, pairs of similar samples are build that are later merged together to bigger clusters
- single: the next sample with the closest distance is added step by step to the cluster
- average: balance between the two other methods

The differences between different distance method is less pronounced compared to the differences in linkage methods.

```
> #top20, euclidean and complete
> hc <- hclust(d=dist(x=t(dat_quantile[1:20, ]), method="euclidean"),
```

```
+              method="complete")
> plot(hc)

> #top20, manhattan and complete
> hc <- hclust(d=dist(x=t(dat_quantile[1:20, ]), method="manhattan"),
+              method="complete")
> plot(hc)

> #example: top 20, correlation-based and complete
> hc <- hclust(d=as.dist((1-cor(dat_quantile[1:20, ],
+                               use = "pairwise.complete.obs"))/2),
+              method="complete")
> plot(hc)
```



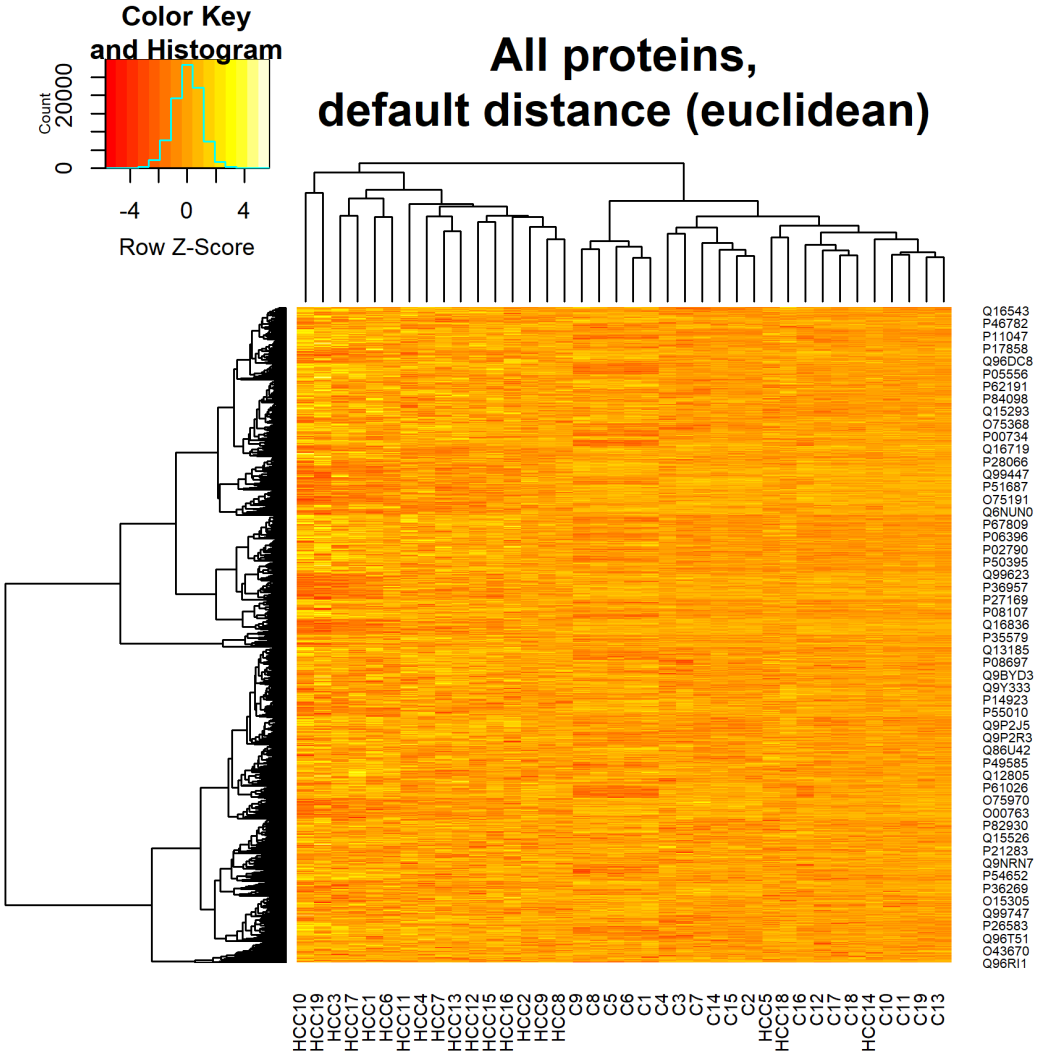
as.dist((1 - cor(dat_quantile[1:20,], use = "pairwise.complete.obs"))/2)
hclust (*, "complete")

Using only the top 20 proteins, both groups (HCC vs. C) can be seperated perfectly. Be careful with this type of clustering as it could give a too optimistic view of the data and the ability to separate both groups.

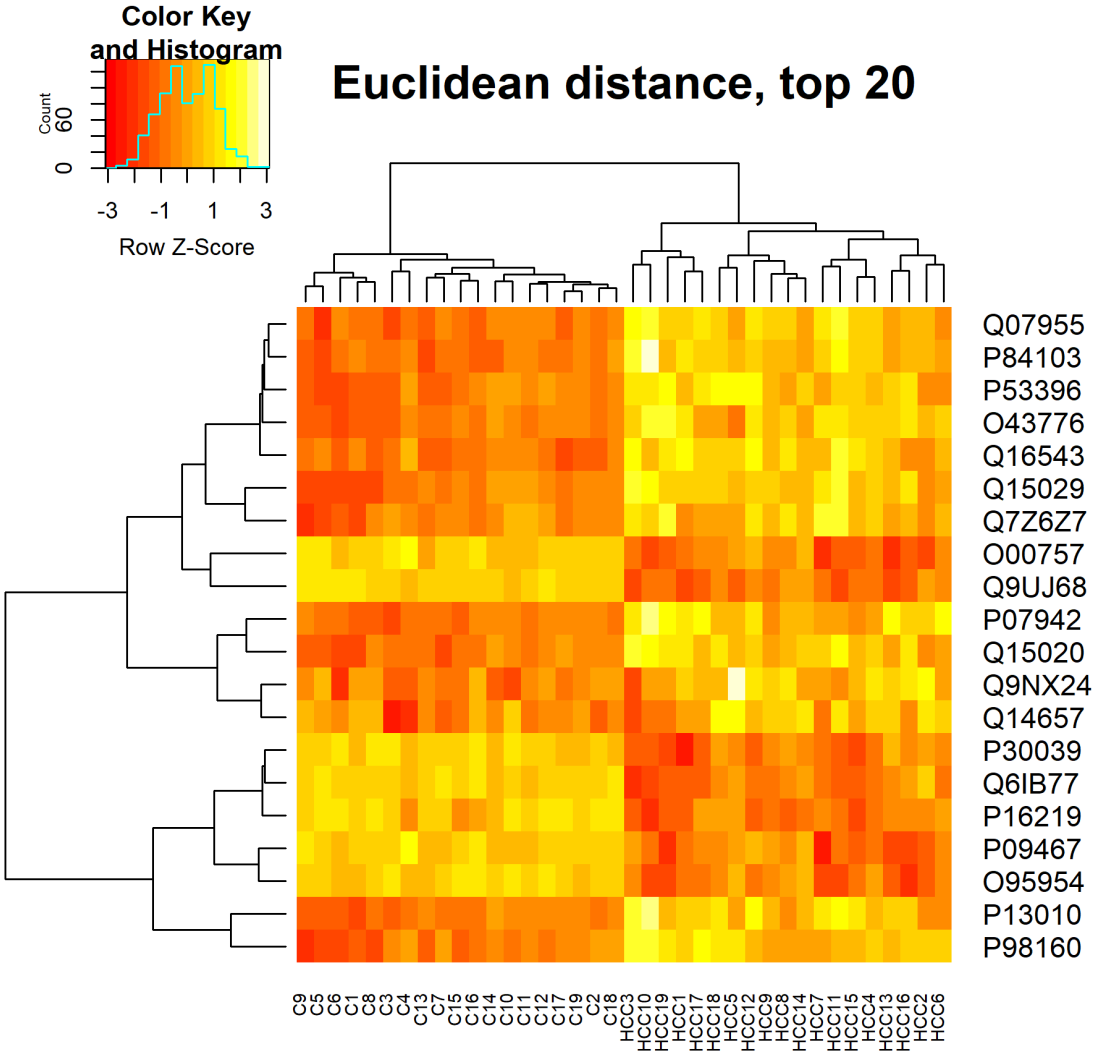
Exercise 3.2: Heatmaps

Draw a heatmap for the given dataset that has been quantile normalized using the function `heatmap.2` from the package `gplots`. Remove proteins with missing values from the dataset. Your heatmap shall include both row and column dendrograms as well as a color key. Optimize the arguments trace, scale, cexRow and cexCol. Is there any effect when instead of all proteins only the 20 most differential hits (by means of p-value) are used?

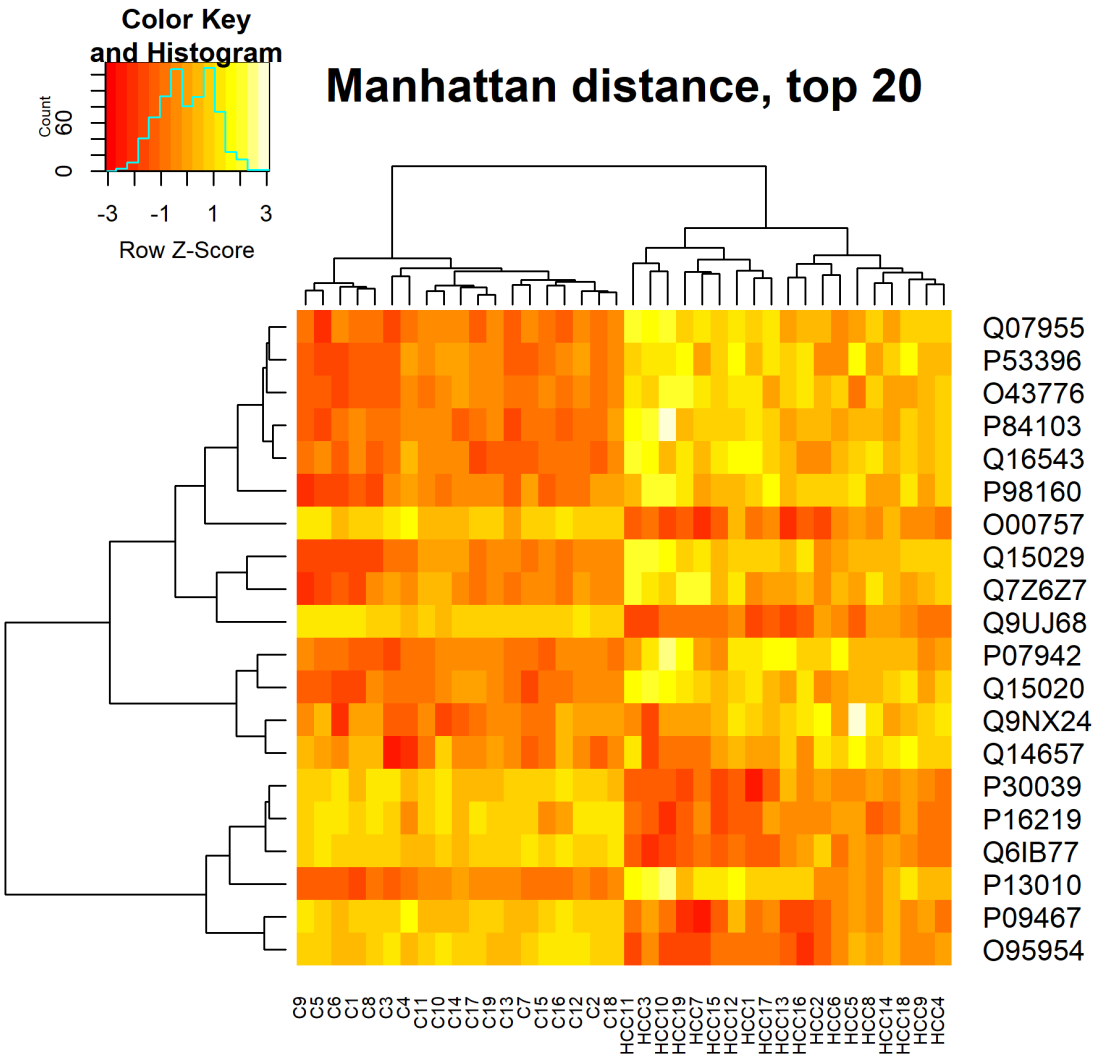
```
> library(gplots)
> # change row names to Accessions numbers for better labelling in the heatmaps
> rownames(dat_quantile) <- accessions$Accession
> # remove rows with missing values
> dat_quantile2 <- na.omit(dat_quantile)

> #all proteins
> heatmap.2(x=dat_quantile2, Rowv=TRUE, Colv=TRUE, trace="none", scale="row",
+           cexCol=0.7, main="All proteins,\ndefault distance (euclidean)")
```



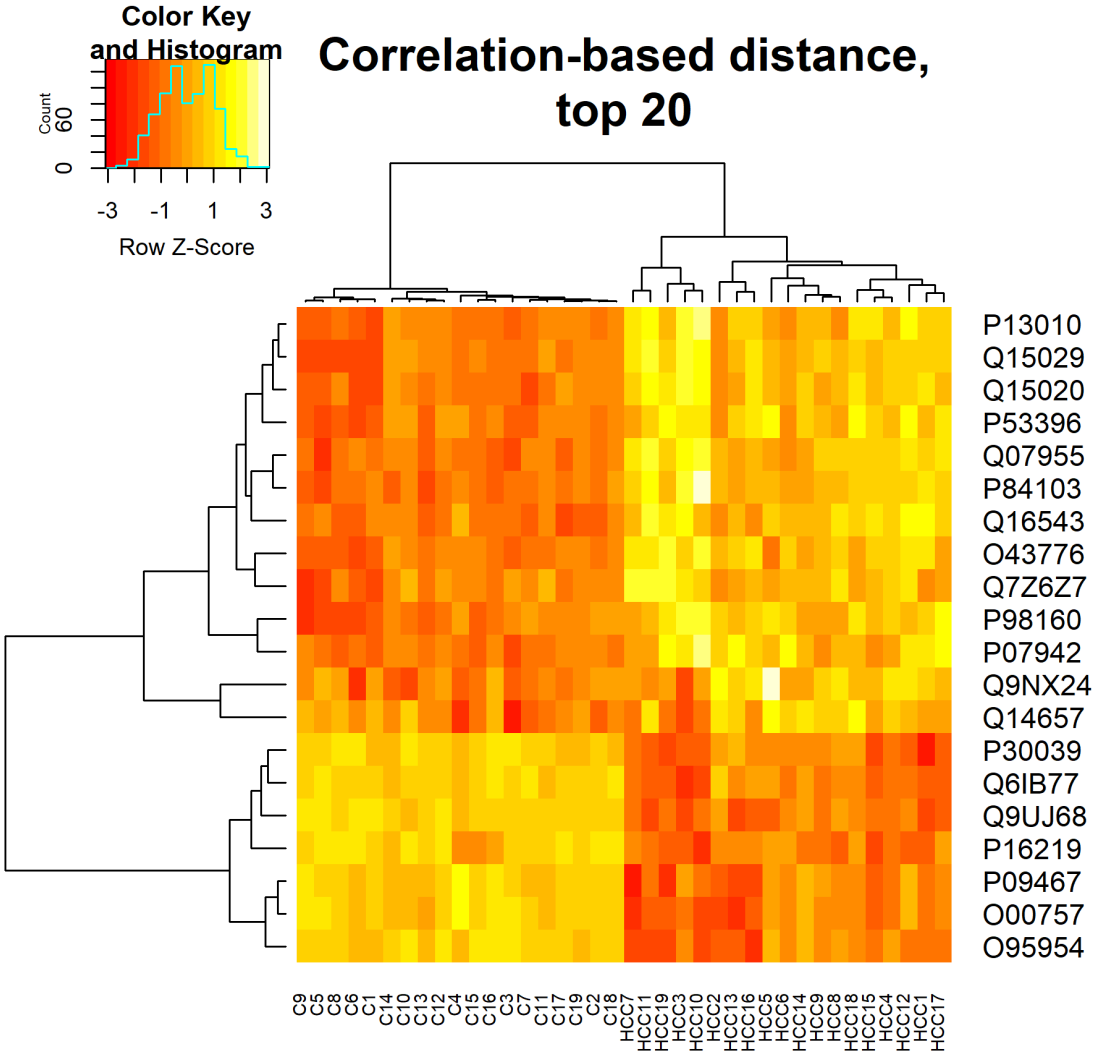

```
> #euclidean, top 20
> heatmap.2(x=dat_quantile2[1:20,],
+           distfun=function(x) {dist(x,method="euclidean")},
+           Rowv=TRUE, Colv=TRUE, trace="none", scale="row", cexRow=1.1,
+           cexCol=0.7, main="Euclidean distance, top 20")
```




```
> #manhattan, top 20
> heatmap.2(x=dat_quantile2[1:20,],
+           distfun=function(x) {dist(x,method="manhattan")},
+           Rowv=TRUE, Colv=TRUE, trace="none", scale="row", cexRow=1.1,
+           cexCol=0.7, main="Manhattan distance, top 20")
```




```
> #correlation-based, top 20
> heatmap.2(x=dat_quantile2[1:20,], distfun=function(x) as.dist((1-cor(t(x),
+                 use = "pairwise.complete.obs"))/2),
+           Rowv=TRUE, Colv=TRUE, trace="none", scale="row", cexRow=1.1,
+           cexCol=0.7, main="Correlation-based distance,\ntop 20")
```



Again using only the top 20 proteins a perfect separation of groups can be achieved. Using a correlation based distance function, proteins with a similar pattern cluster together, which is usually the intended behaviour.

3 Hands-on session III: PCA and ROC

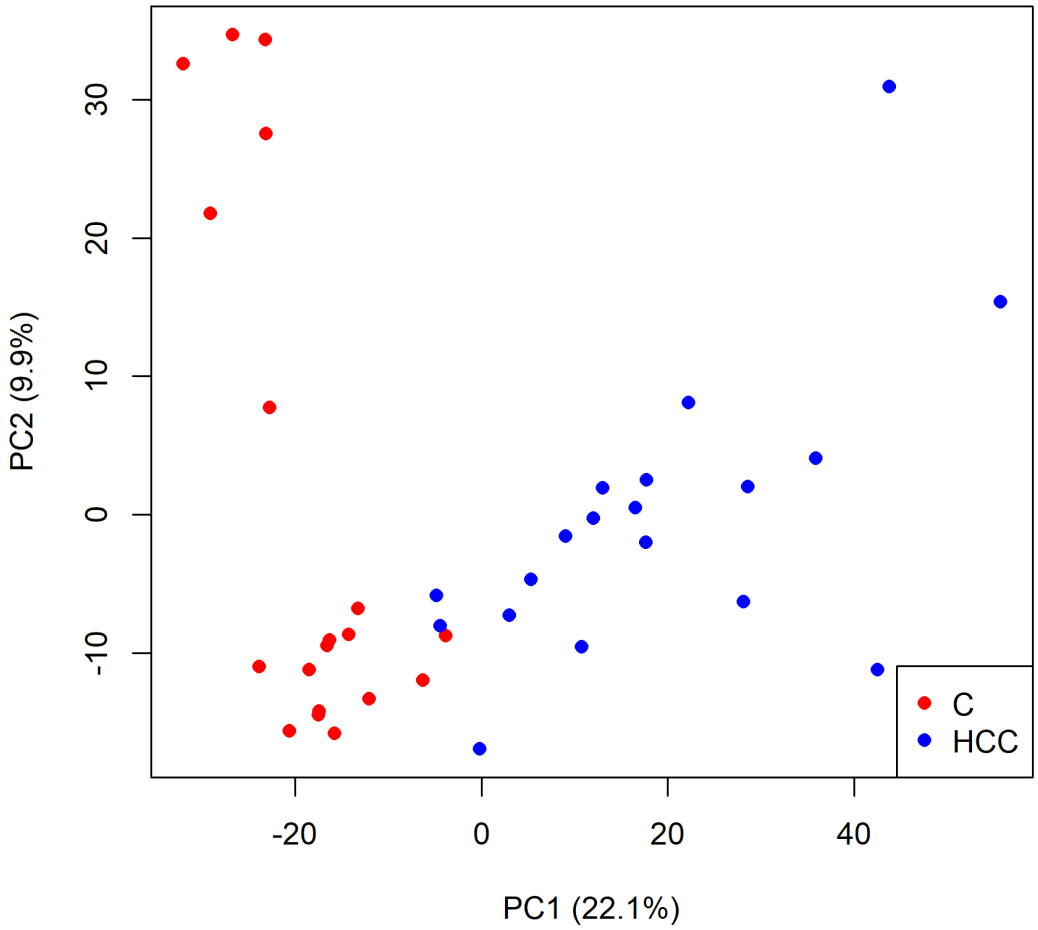
Exercise 3.1: PCA

Compute a principal component analysis on the quantile normalized data set from Exercise 1.2.

To do this, first delete all rows (proteins) containing missing values using `na.omit()`. Second, transpose the dataset using `t()`, as the variables (proteins) need to be in columns and observations (samples) in rows. Then, apply `prcomp` to the dataset.

Create a plot of the first two principal components.

```
> ## remove rows with missing values
> dat_quantile2 <- na.omit(dat_quantile)
> ## transpose data
> dat_quantile2 <- t(dat_quantile2)
> # calculate PCA
> pca <- prcomp(dat_quantile2, scale. = TRUE)
> # calculate summary of PCA (for explained variance)
> summ <- summary(pca)
> # plot the PCA data and add a legend
> plot(pca$x[,1], pca$x[,2], pch = 16, col = rep(c("red", "blue"), each = 19),
+      xlab = paste0("PC1 (", round(100*summ$importance[2,1],1), "%)"),
+      ylab = paste0("PC2 (", round(100*summ$importance[2,2],1), "%)"),
+      )
> legend("bottomright", col = c("red", "blue"), legend = c("C", "HCC"), pch = 16)
```



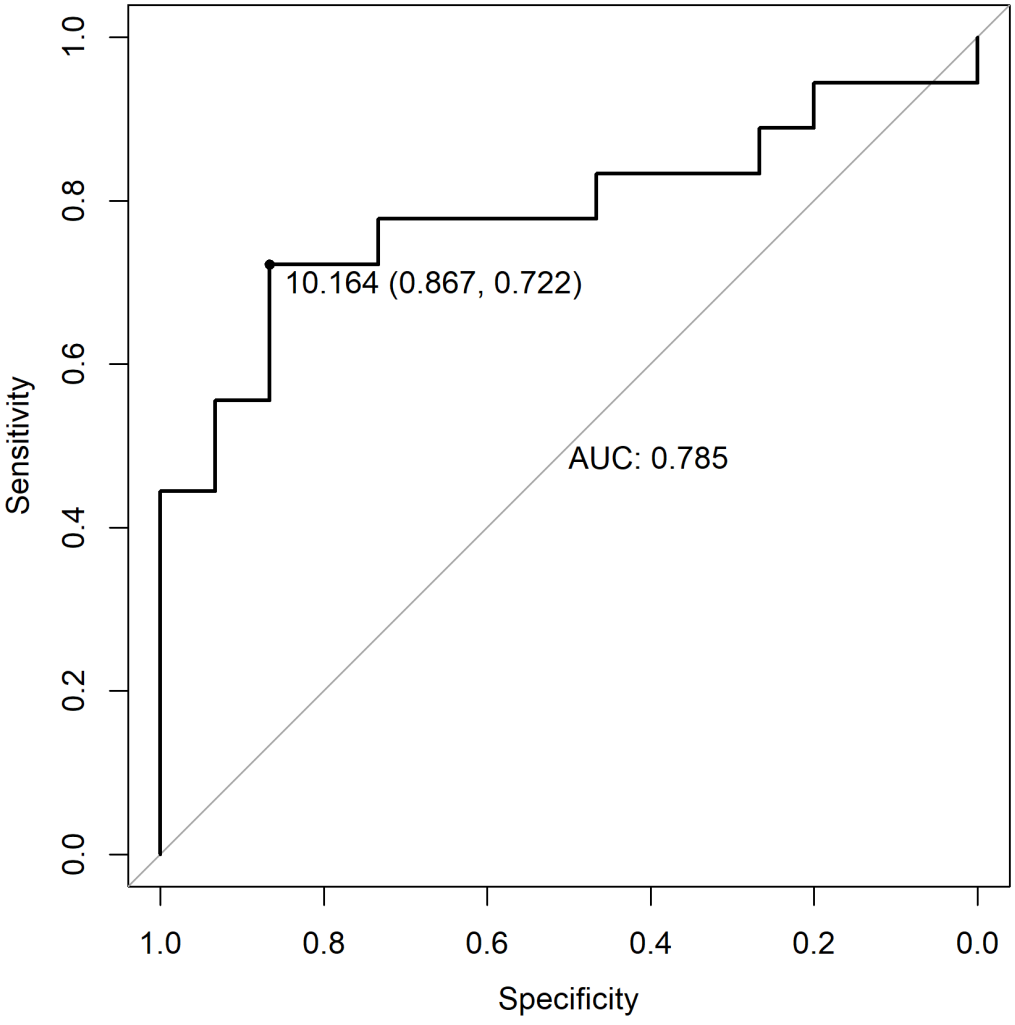
Exercise 3.2: ROC

In the paper corresponding to the dataset amongst others the following proteins were found significantly regulated between Hepatocellular carcinoma and healthy tissue.

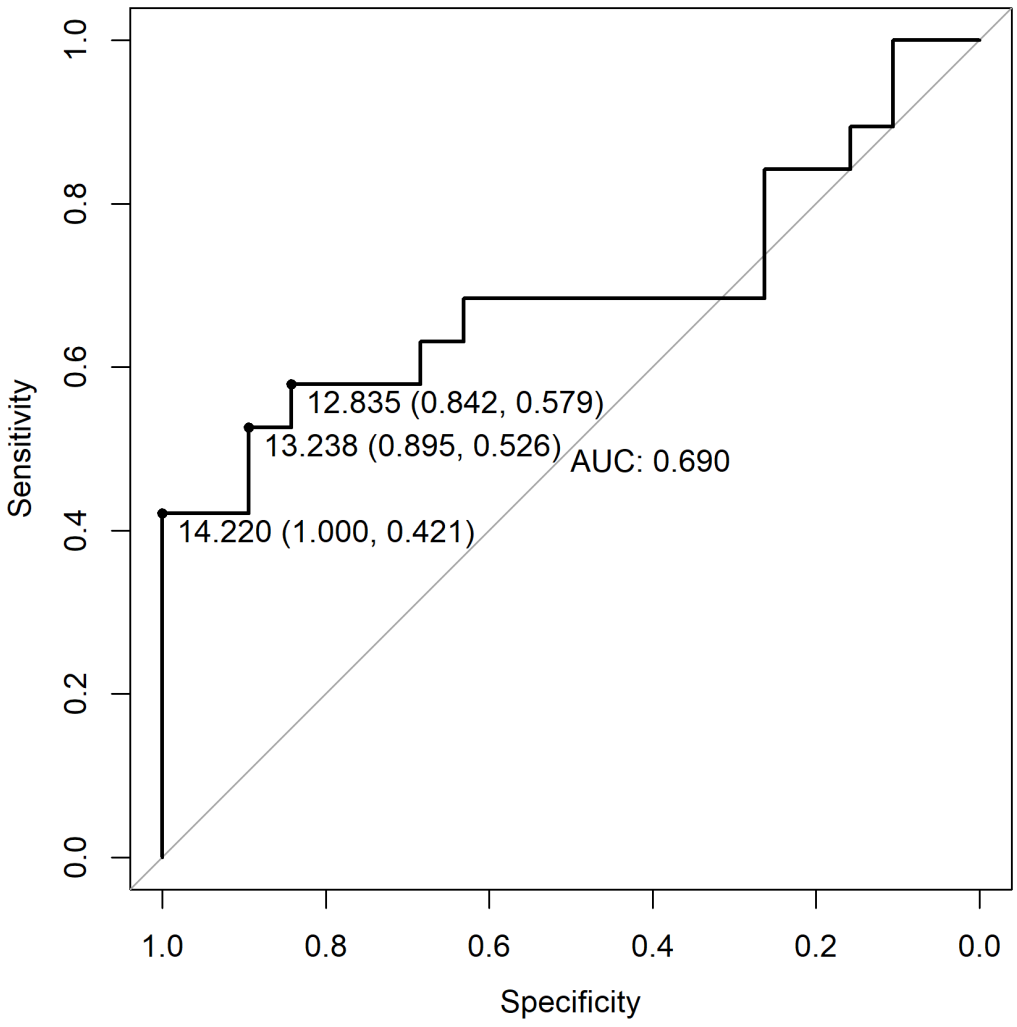
- ATP-dependent RNA helicase (DDX39) (O00148)
- Fibulin-5 (FBLN5) (Q9UBX5)
- Myristoylated alanine-rich C-kinase substrate (MARCKS) (P29966)
- Serpin H1 (SERPINH1) (P50454)

Create ROC-curves for these proteins, showing the best threshold and the AUC on the normalized dataset.

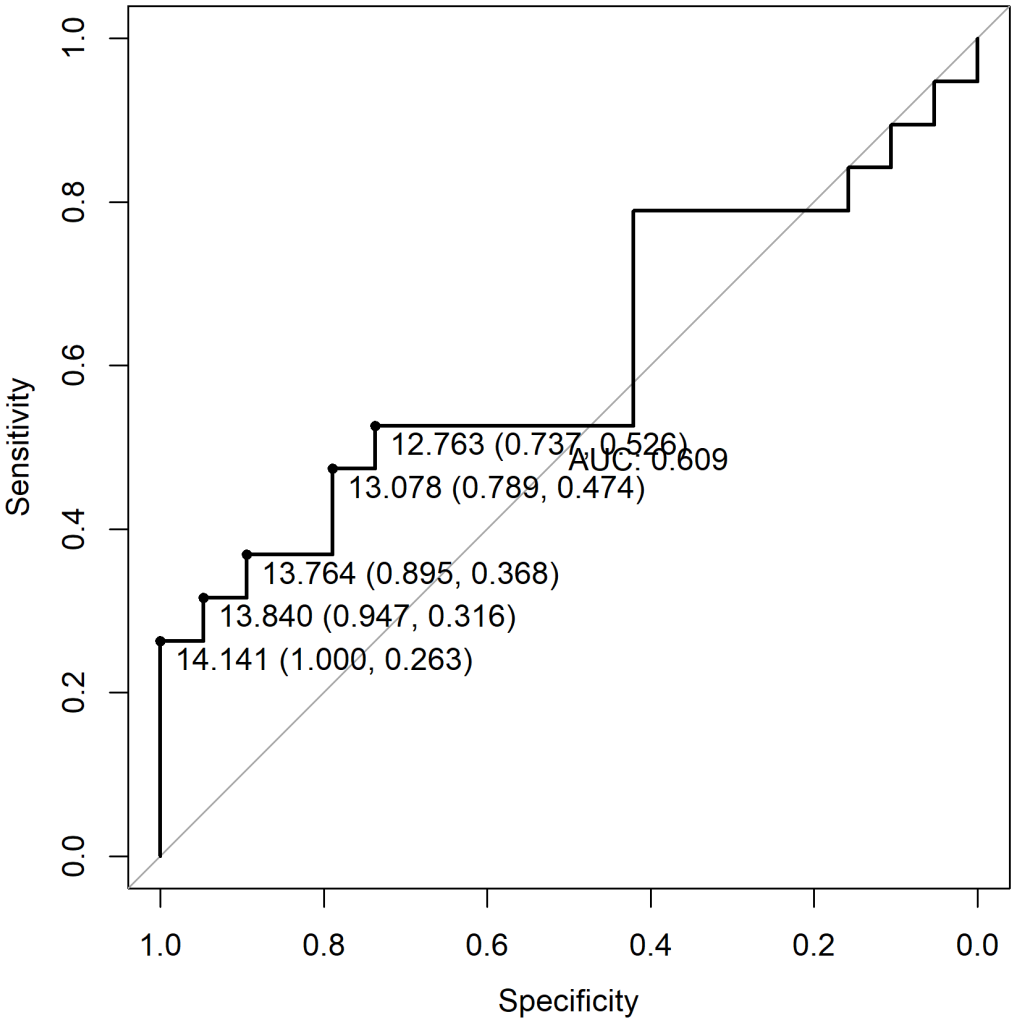
```
> library(pROC)
> # find out in which row the specific protein is
> ind <- which(accessions$Accession == "O00148")
> # choose only the one row with this protein:
> D <- dat_quantile[ind,]
> # do ROC analysis (19 controls vs 19 cases)
> ROC <- pROC::roc(controls = D[1:19], cases = D[20:38])
> # plot ROC curve
> plot(ROC, print.thres = "best", print.auc = TRUE)
```



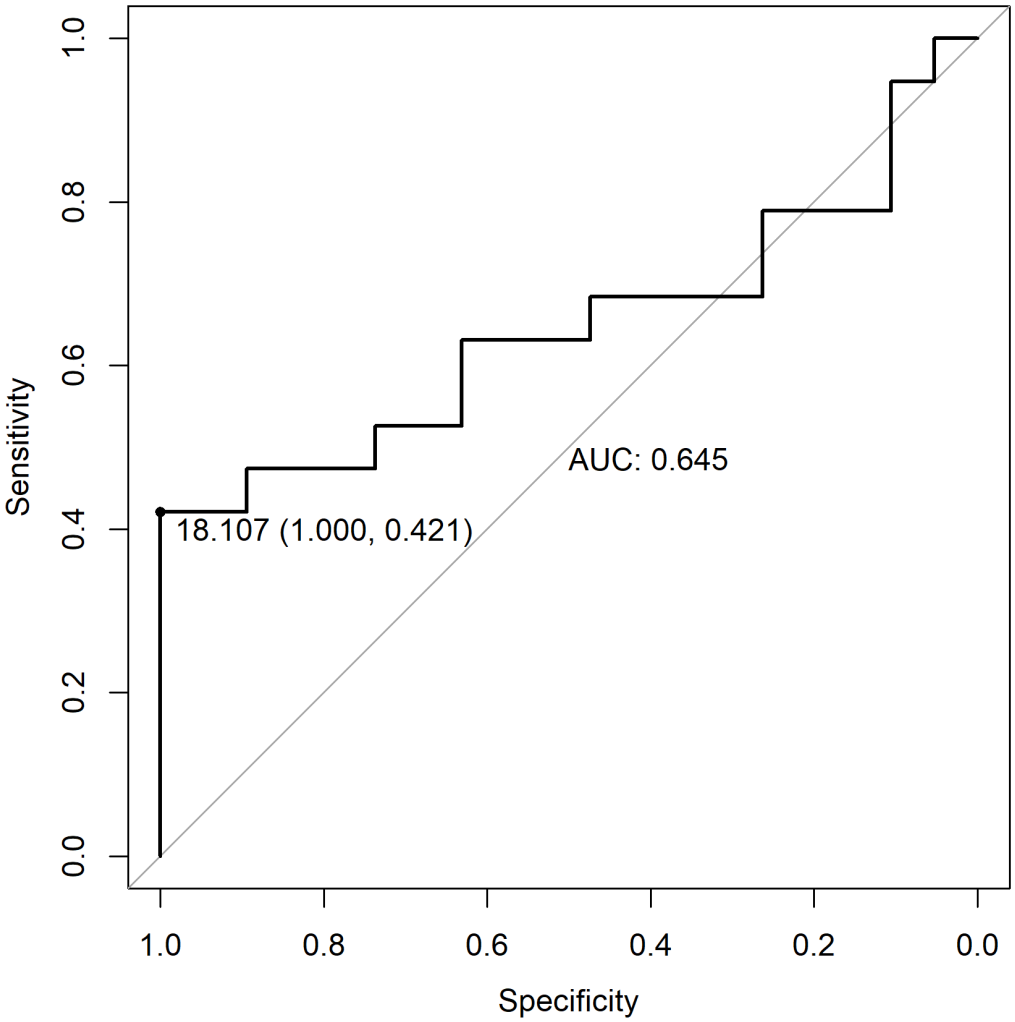
```
> ind <- which(accessions$Accession == "Q9UBX5")
> D <- dat_quantile[ind,]
> ROC <- pROC::roc(controls = D[1:19], cases = D[20:38])
> plot(ROC, print.thres = "best", print.auc = TRUE)
```




```
> ind <- which(accessions$Accession == "P29966")
> D <- dat_quantile[ind,]
> ROC <- pROC::roc(controls = D[1:19], cases = D[20:38])
> plot(ROC, print.thres = "best", print.auc = TRUE)
```




```
> ind <- which(accessions$Accession == "P50454")
> D <- dat_quantile[ind,]
> ROC <- pROC::roc(controls = D[1:19], cases = D[20:38])
> plot(ROC, print.thres = "best", print.auc = TRUE)
```



4 Hands-on session IV: Writing own R functions

Exercise 4.1: Simple function

Write a short R function that converts a given temperature value in Fahrenheit to Celsius (substract 32, multiply by 5, then divide by 9) and test it.

```
> ## function is called temperature_F_to_C, which has one single argument
> ## temp_F without a default value.
> temperature_F_to_C <- function(temp_F) {
+         temp_step1 <- temp_F - 32           # 1st step: substract 32
+         temp_step2 <- temp_step1 *5         # 2nd step: multiply by 5
```

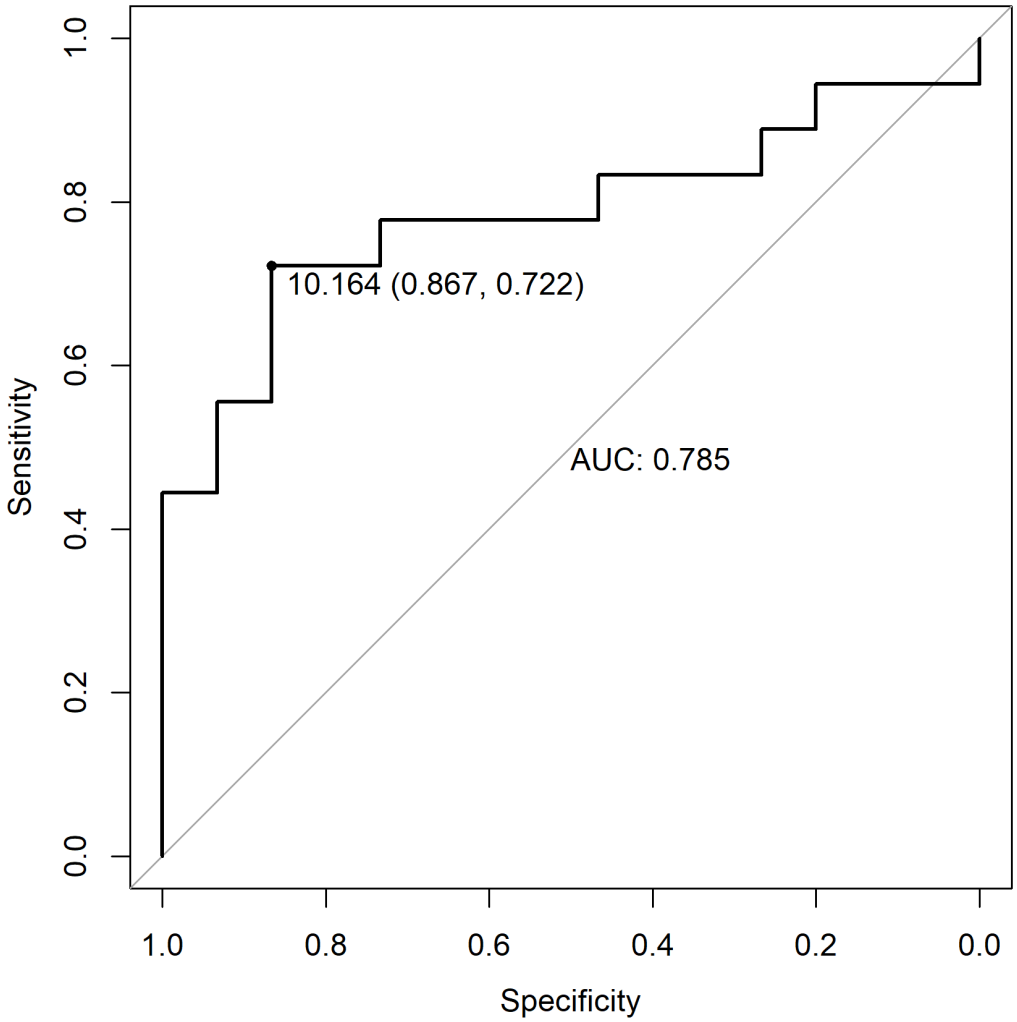
```
+         temp_C <- temp_step2 /9              # 3rd step: divide by 9
+         return(temp_C)                       # return result (temp in C)
+ }
> ## you can now use this function like any predefined function in R:
> temperature_F_to_C(0)

[1] -17.77778

> temperature_F_to_C(32)

[1] 0

> temperature_F_to_C(212)

[1] 100
```

Exercise 4.2: Function for ROC curves

Re-use the code from exercise 3.2 and write a function that automatically plots the ROC curve for a given protein accession. Use `stop()` to give an error if the defined accession is not present in the dataset. Return the ROC-object obtained by ROC() in the end.

Add arguments to the function (with appropriate default values) that allows costumization of the plot, e.g. turning on/off the printing of the AUC or best cutoffs.

```
> ## begin of function: argument accession without default values,
> ## the arguments print.thres, print.auc and col have default values
> ROC_curve <- function(accession, print.thres = "best", print.auc = TRUE,
+                       col = "black") {
+
+         ### error message: if the given accession is not present in the
+         ### Accession-column, then give an error message
+         if (!is.element(accession, accessions$Accession)) {
+                 stop("Chosen accession is not present in dataset!")
+         }
+
+         #### copy of the code from 3.2 but replace arguments with the
+         #### argument names of the ROC_curve function
+         ind <- which(accessions$Accession == accession)
+         D <- dat_quantile[ind,]
+         ROC <- pROC::roc(controls = D[1:19], cases = D[20:38])
+         plot(ROC, print.thres = print.thres, print.auc = print.auc, col = col)
+         return(ROC)
+ }
> # same plot as in 3.2:
> ROC_curve(accession = "O00148")

Call:
roc.default(controls = D[1:19], cases = D[20:38])

Data: 15 controls < 18 cases.
Area under the curve: 0.7852
```




```
> # error as acession is not present in dataset:
> ROC_curve(accession = "O00149")

Error in ROC_curve(accession = "O00149") :
  Chosen accession is not present in dataset!

> # customize plotting
> ROC_curve(accession = "O00148", print.thres = FALSE, print.auc = FALSE,
+           col = "red")

Call:
roc.default(controls = D[1:19], cases = D[20:38])
```

```
Data: 15 controls < 18 cases.
Area under the curve: 0.7852
```