Some basics on R

Magnus Fontes

Copenhagen 2017

R Markdown

The documentation for the course is beeing produced using *Markdown*. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

Online Help

We assume that you have R as well as R Studio installed. You must first download R from the CRAN website https://cran.r-project.org/ and then R studio from the RStudio website https://www.rstudio.com/

Then the most important thing once you have started R through for example RStudio is to know how to get **help**

Again at https://cran.r-project.org/ you find R tutorials.

Often you can find useful information simply by googling a keyword (like the name of a command) followed by R. Fora for programmers http://stackoverflow.com/.

For statistical questions check out **Crossvalidated** at http://stats.stackexchange.com.

And for bioinformatics and biostatistics https://www.bioconductor.org/

Built In Help

In the R command line you simply type **help.start()** to start the general help.

?x or **help(x)** to get the R documentation around the function x. In particular the examples at the end of the help text often give enough information to understand how to use a function. You can also type **example(x)**.

The command **apropos(x)** lists all functions containing the string "x".

For a fuzzy help search try **??x**

R, as well as R packages come with example datasets. To list the available example datasets type **data()**.

Starting and quitting R

Double-click on the R Studio icon (or the R icon to only open R).

To quit your R session type **q()** in the command line or select quit R Studio in the dropdown file menu in R Studio.

The working directory

The *working directory* is to where R by default directs input and output when you e.g. use commands like **source** and **dump** to create R files that hold the commands neccessary to recreate R objects or **save** and **load** to directly save and load R objects.

You can find your current working directory:

```
getwd()
```

```
## [1] "C:/Users/Magnus/Documents/R/RMarkdown"
```

and you can change your working directory:

```
setwd("C:/Users/Magnus/Documents/R/RMarkdown")
getwd()
```

```
## [1] "C:/Users/Magnus/Documents/R/RMarkdown"
```

Creating vectors

Create vectors by concatenation with the function **c()**.

Creating a vector containing the natural numbers 1 to 10 in increasing order:

```
x <- c(1:10)
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Or by the quick command

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Sampling in general

To randomly permute a vector:

```
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
sample(x)
```

```
##  [1]  9  5  1  4  3  7  6  8  2 10
```

This is useful for example if you want to randomly partition a cohort into two groups that will serve as case and control.

Assume that we have 40 patients. We then number them from 1 to 40:

```
options(width=50)
patient.numbers <- 1:40
patient.numbers
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
## [16] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
## [31] 31 32 33 34 35 36 37 38 39 40
```

The command ***options(width=50)*** *is there to get a nice layout.*

If we randomly want to select half of them to undergo a treatment we do this by randomly selecting 20:

```
sample(patient.numbers,size=20)
```

```
##  [1] 27 33  1 24  5 17 29 37 25  6 26 36 34 20 14
## [16] 16 32 23 15 18
```

Sampling from a predefined distribution

Creating a random vector of size 10 drawn from a normal distribution (mean=0 and standard deviation sd=1):

```
y <- rnorm(10, mean = 0, sd = 1)
y
```

```
##  [1]  0.5461259  0.4906653  0.7251091 -0.5567929
##  [5] -0.5719644  1.7191916  1.1771126 -0.5030103
##  [9] -1.3031281  1.0462450
```

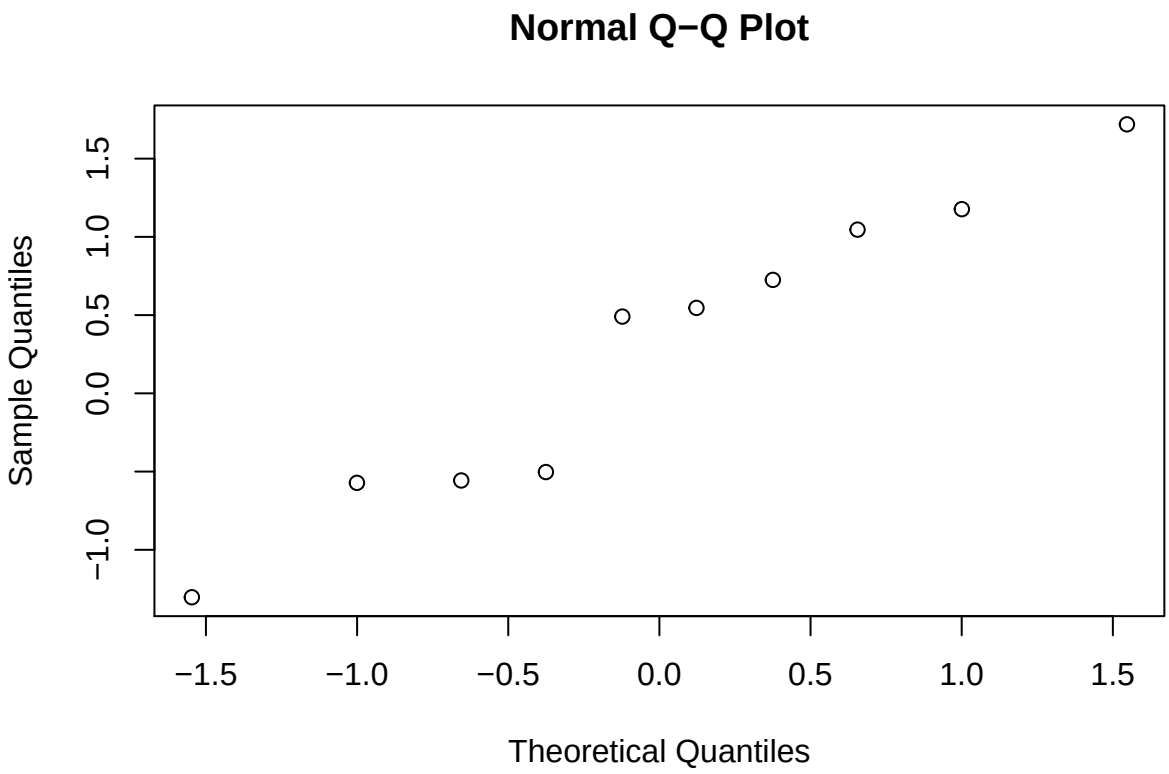
summary is used to get some basic statistics on a given sample:

```
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.3030 -0.5433  0.5184  0.2770  0.9660  1.7190
```

But a quantile-quantile (qq) plot might give more:

```
qqnorm(y)
```



Sampling with replacement

Given a vector:

```
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

We sample with replacement:

```
sample(x, replace=TRUE)
```

```
##  [1]  6  2  9  3  9 10  6  1  3  6
```

Bootstrapping is an important technique for understanding e.g. the *robustness* of inferences. It is in general used to learn as much as possible about an unknown probability distribution that we have sampled from. It is based on resampling many times. To perform it we need flow control constructs.

Control Flow

These are the basic control-flow constructs of the R language. They function in much the same way as control statements in any Algol-like language (like **Python**, **C++**, **Julia** etc). They are all reserved words.

if(cond) expr

if(cond) cons.expr else alt.expr

for(var in seq) expr

while(cond) expr

repeat expr

break

next

Bootstrapping

We create a sample vector drawing from a normal distribution:

```
x <- rnorm(1000)
```

We set up a numerical vector, named **m.d** of length 10000:

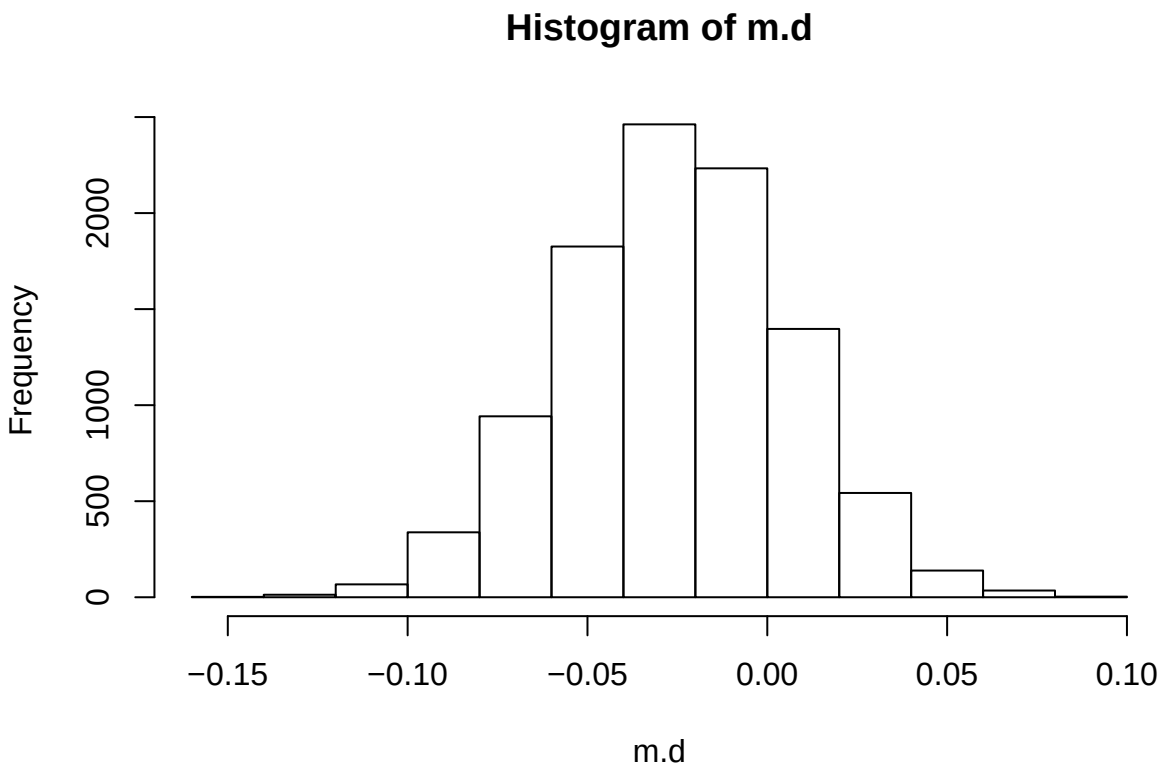
```
m.d <- numeric(10000)
```

And then we store the mean values of bootstrapped samples in **m.d**

```
for (i in 1:10000) m.d[i] <- mean(sample(x, replace=TRUE))
```

We can now plot a histogram for the meanvalues of the bootstrapped samples

```
hist(m.d)
```

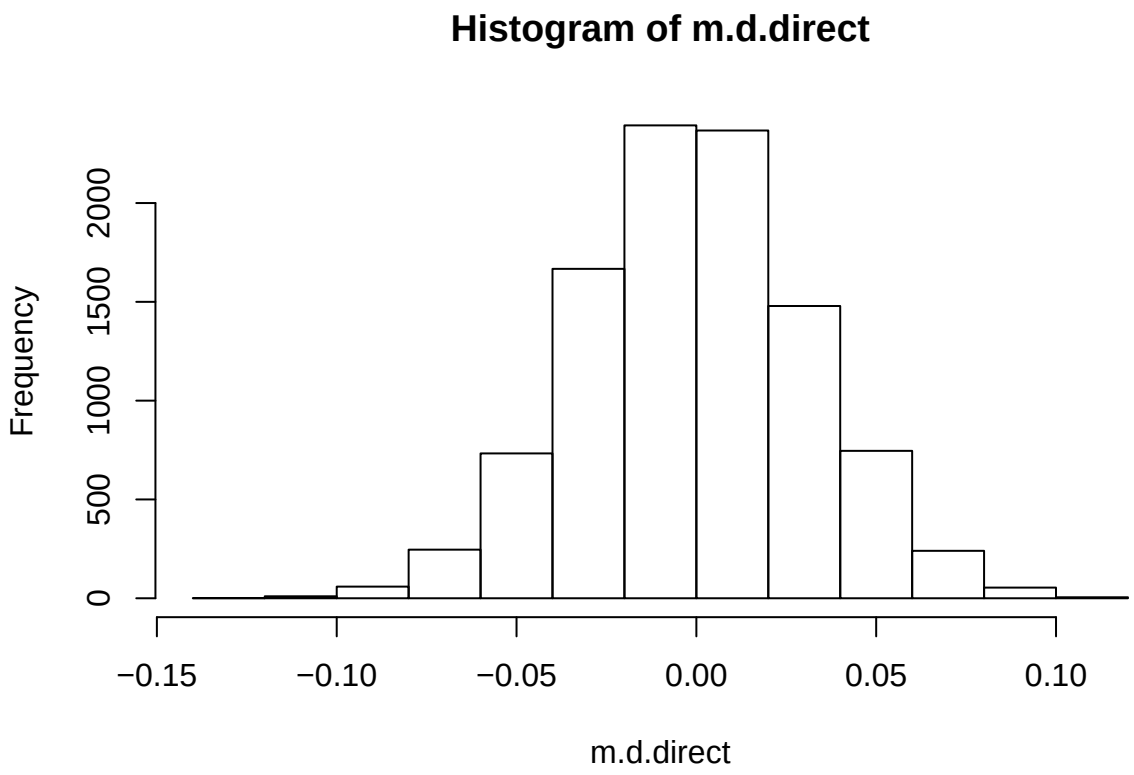


This is an approximation of a sample distribution based on bootstrapping one particular sample that can be used in particular to asses the variation, or standard deviation, in our data. This is important when we for example construct **confidence intervals**.

We can compare our bootstrap approximation with a direct approximation of the mean value statistic for normally distributed data:

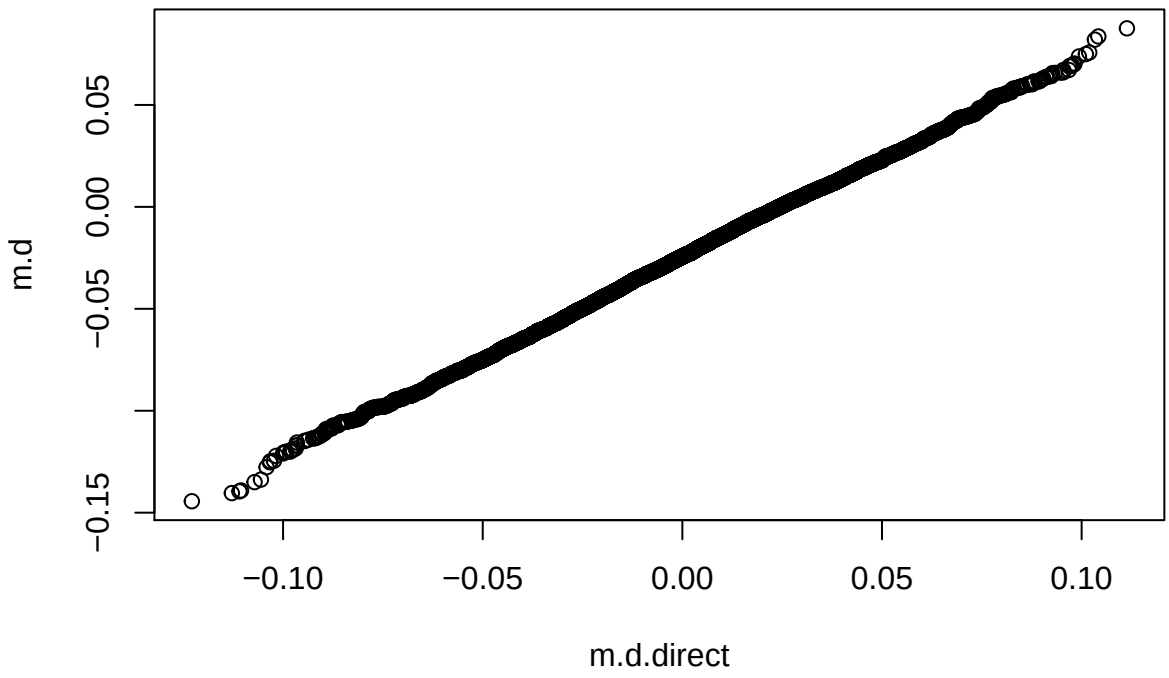
```
m.d.direct <- numeric(10000)
for (i in 1:10000) m.d.direct[i] <- mean(rnorm(1000))
```

```
hist(m.d.direct)
```



And we compare the two distributions with a qqplot

```
qqplot(m.d.direct, m.d)
```



We note that the bootstrapped and the directly sampled distributions are very similar and that the standard deviation estimated by bootstrapping provides a good estimate of the empirical standard deviation:

```
mean(m.d.direct)
```

```
## [1] -0.0006580515
```

```
mean(m.d)
```

```
## [1] -0.02536859
```

```
sd(m.d.direct)
```

```
## [1] 0.03179799
```

```
sd(m.d)
```

```
## [1] 0.03137605
```

The standard deviation is well estimated by bootstrapping, but the mean is highly influenced by outliers. This can be dealt with by trimming the means. Using the same underlying sample **x**:

```
m.d.t <- numeric(10000)

for (i in 1:10000) m.d.t[i] <-
  mean(sample(x, replace=TRUE), trim=0.003)

m.d.t.direct <- numeric(10000)
for (i in 1:10000) m.d.t.direct[i] <-
  mean(rnorm(1000), trim=0.003)
```

```
mean(m.d.t.direct)
```

```
## [1] 0.0005068138
```

```
mean(m.d.t)
```

```
## [1] -0.02407681
```

```
sd(m.d.t.direct)
```

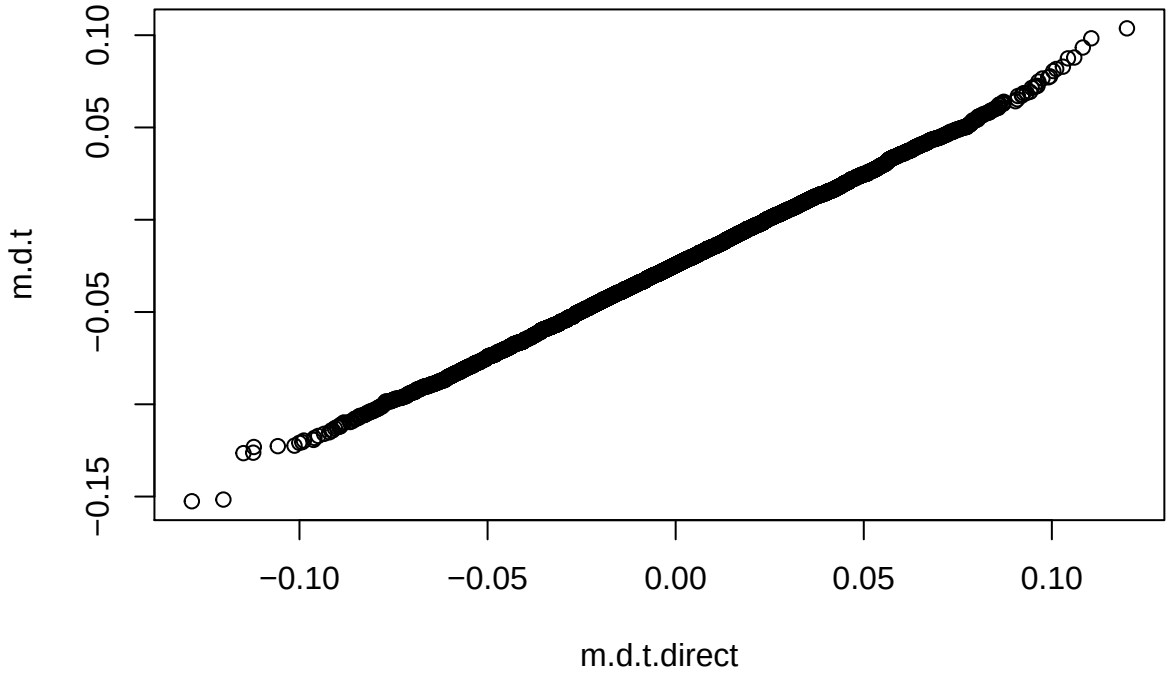
```
## [1] 0.03173242
```

```
sd(m.d.t)
```

```
## [1] 0.03161846
```

And we compare them in a qqplot

```
qqplot(m.d.t.direct, m.d.t)
```



Robust non parametric methods

Concerning estimating a central tendency in a dataset, the **median** is a more robust statistic than the **mean**. Look at standard available distributions by typing **help(distributions)** on the R command line.

Exercise: Redo the analyses and comparisons that we have made above using other underlying distributions to create the samples.

For the normal distribution the mean is a very good choice, but below we use the median on the same underlying sample x:

```
med.d <- numeric(10000)

for (i in 1:10000) med.d[i] <-
  median(sample(x, replace=TRUE))

med.d.direct <- numeric(10000)
for (i in 1:10000) med.d.direct[i] <-
  median(rnorm(1000))
```

```r
mean(med.d.direct)
```

```
## [1] -0.0001121217
```

```r
mean(med.d)
```

```
## [1] 0.0009193478
```

```r
sd(med.d.direct)
```

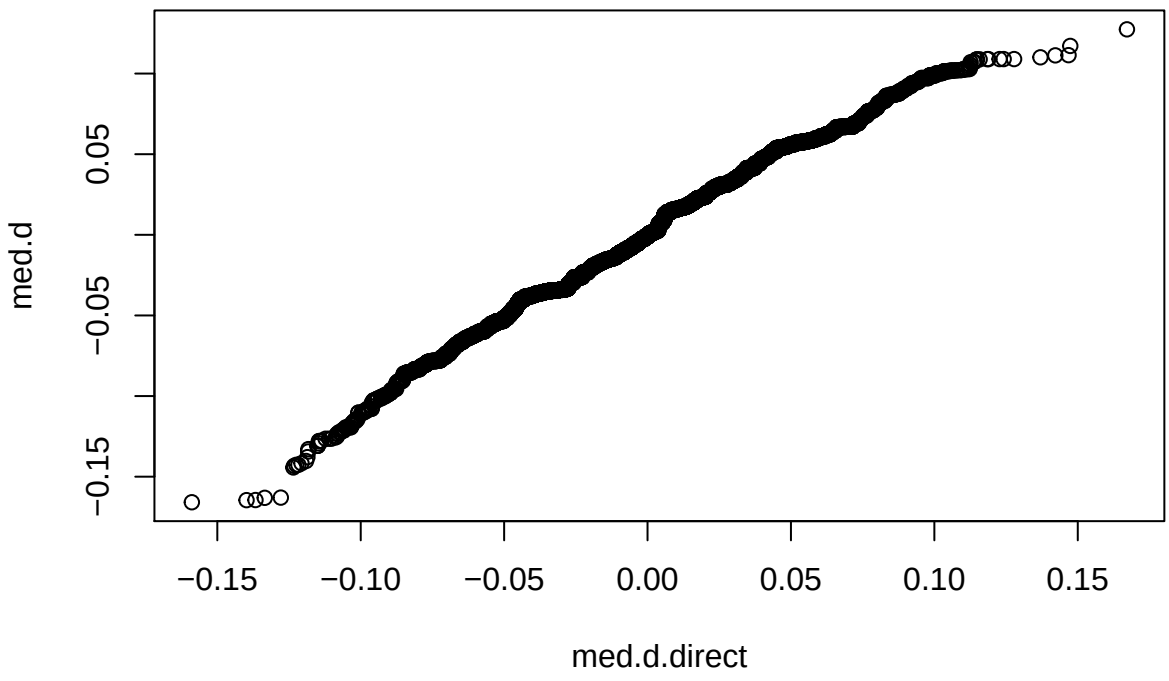
```
## [1] 0.03922468
```

```r
sd(med.d)
```

```
## [1] 0.04140167
```

And we compare them in a qqplot

```r
qqplot(med.d.direct, med.d)
```

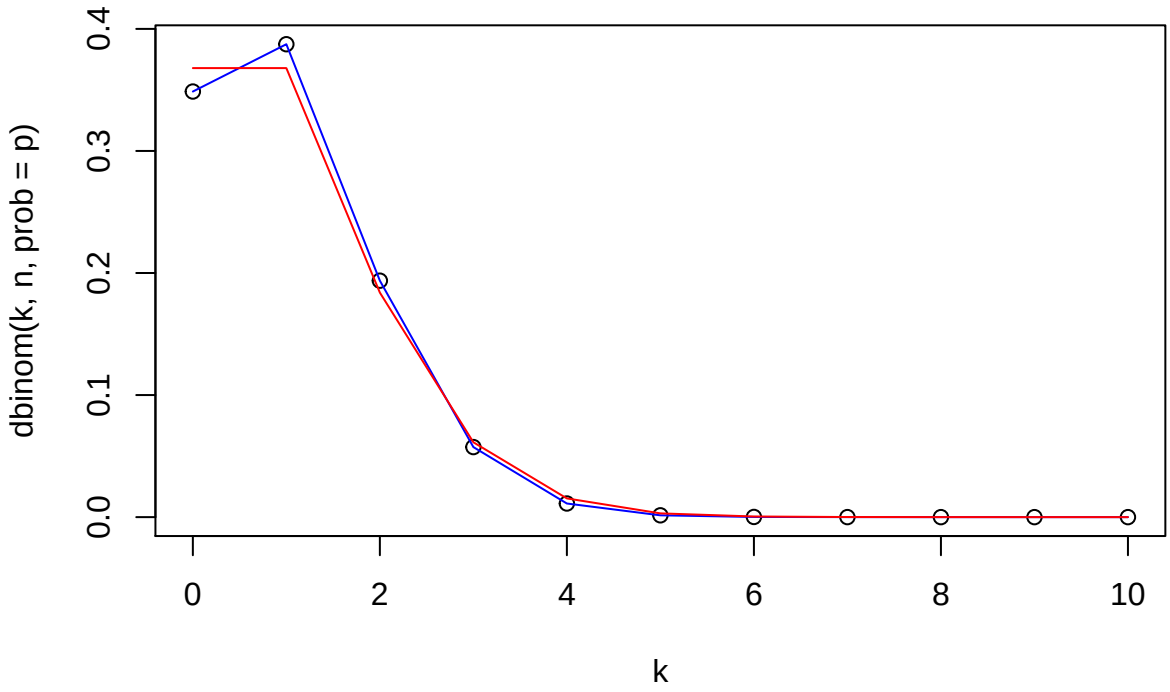


Plotting a binomial distribution and compare with Poisson

```
p <- 0.1
n <- 10
k <- seq(from=0,to=n,by=1)
```

And the plot

```
plot(k,dbinom(k,n,prob=p))
lines(lowess(k,dbinom(k,n,prob=p),f=0.01), col="blue")
lines(lowess(k,dpois(k,lambda=n*p),f=0.01), col="red")
```



Ladislaus Bortkiewicz data set on preussian officers killed by horsekicks

```
horsekicks <- read.csv("~/R/RMarkdown/horsekicks.csv")
typeof(horsekicks)
```

```
## [1] "list"
```

```
class(horsekicks)
```

```
## [1] "data.frame"
```

Writing datafiles from R back to the working directory:

```
kicks.matrix <- as.matrix(horsekicks)
typeof(kicks.matrix)
```

```
## [1] "integer"
```

```
class(kicks.matrix)
```

```
## [1] "matrix"
```

```
kicks.vector <- as.vector(kicks.matrix)
typeof(kicks.vector)
```

```
## [1] "integer"
```

```
class(kicks.vector)
```

```
## [1] "integer"
```

```
write(kicks.vector, file="kicks.vector")
```

Ladislaus Bortkiewicz data set on preussian officers

```
table.kicks <- table(kicks.vector)
table.kicks
```

```
## kicks.vector
##   0   1   2   3   4
## 133  88  32  11   2
```

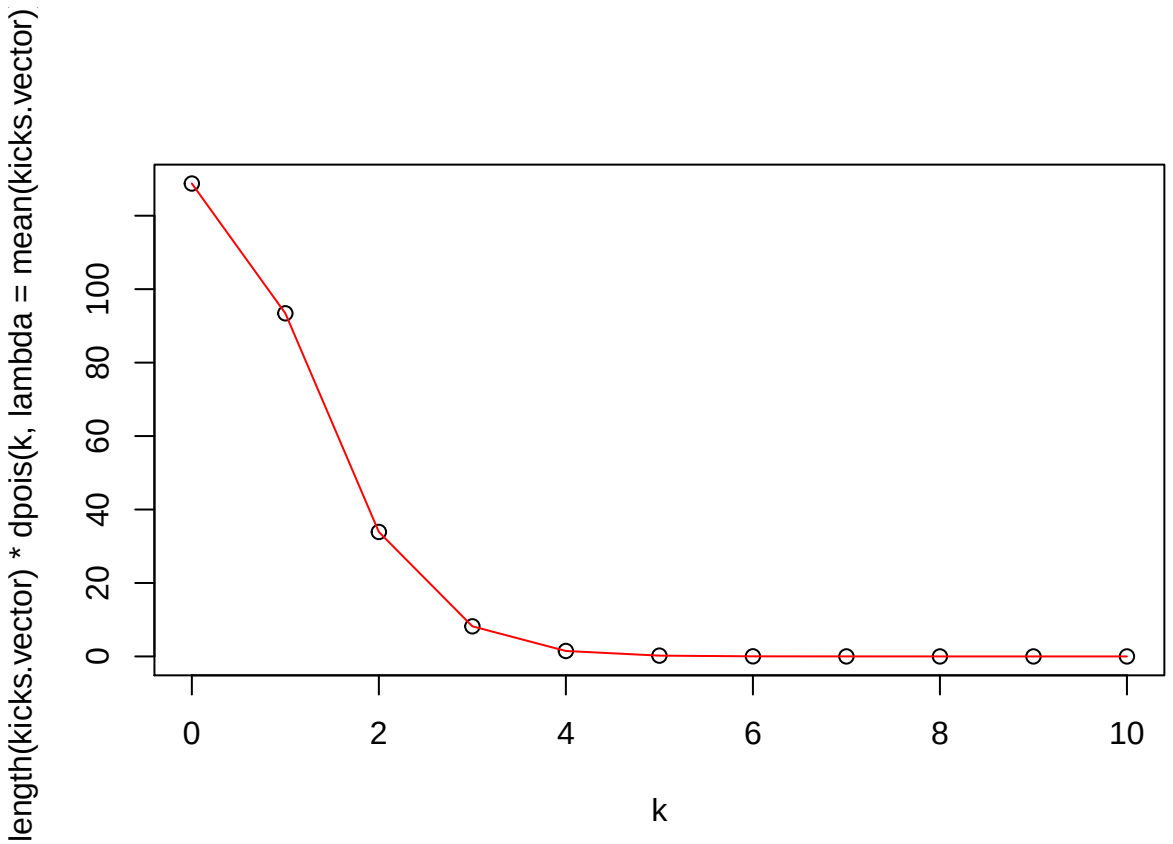
Comparison with Poisson

```
options(width=50)
table(length(kicks.vector)*
        dpois(k,lambda=mean(kicks.vector)))
```

```
##
## 1.43471924950718e-06 1.97738507963166e-05
##                    1                    1
## 0.000245277714022705  0.00270440919917262
##                    1                    1
##    0.026091243154712    0.215759710232748
##                    1                    1
##     1.48684152647438     8.19688800087428
##                    1                    1
##     33.8917959828895     93.4219454036126
##                    1                    1
##     128.757707136585
##                    1
```

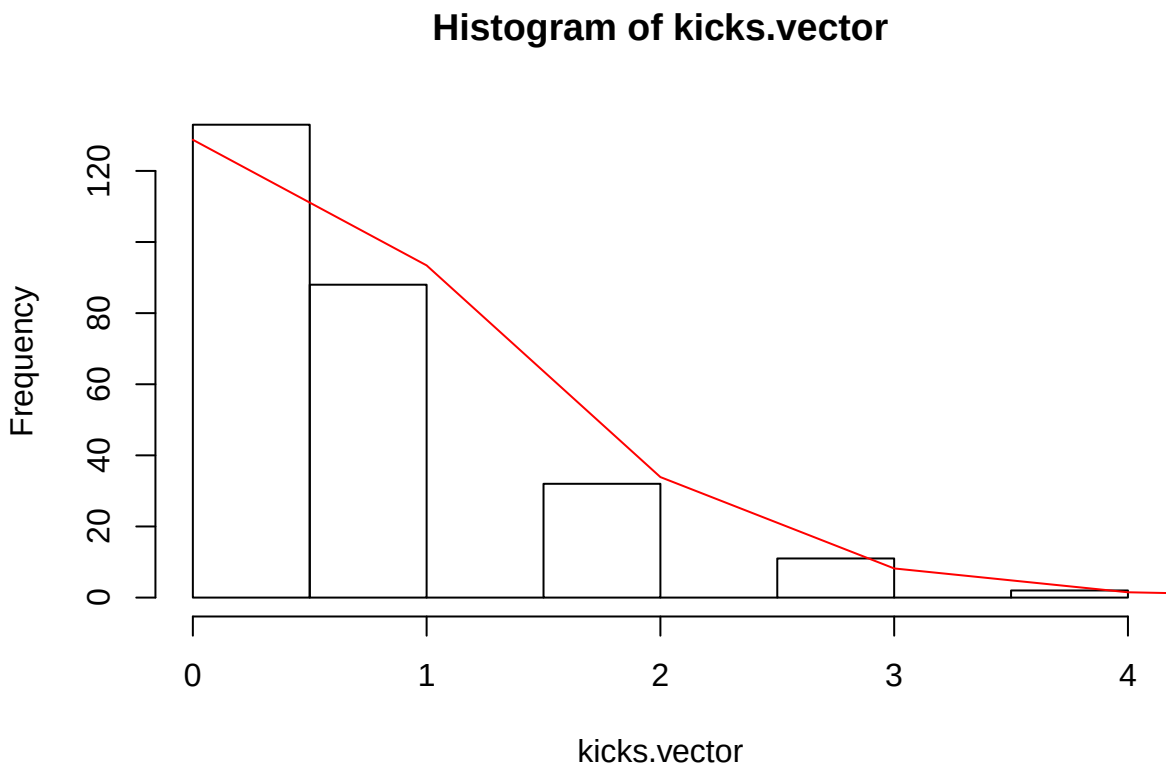
Plotting using the lowess() function

```
plot(k,length(kicks.vector)*
       dpois(k,lambda=mean(kicks.vector)))
lines(lowess(k,length(kicks.vector)*
               dpois(k,lambda=mean(kicks.vector)),
             f=0.01), col="red")
```



Histogram of Bortkiewicz data with Poisson approximation

```
hist(kicks.vector)
lines(lowess(k,length(kicks.vector)*
               dpois(k,lambda=mean(kicks.vector)),
             f=0.01), col="red")
```



We compute the mean and the variance of the sample set

```
mean(kicks.vector)
```

```
## [1] 0.7255639
```

```
var(kicks.vector)
```

```
## [1] 0.7810044
```

We will assume that you have the files **leukemia.RData** and **leukemiaSA.RData** in your working directory. We load and check the class of the loaded objects:

```
load("leukemia.RData")
class(leukemia)
```

```
## [1] "matrix"
```

```
load("leukemiaSA.RData")
class(leukemiaSA)
```

```
## [1] "data.frame"
```

Matrices

A matrix is a two dimensional array.

We create a random matrix of dimension 132 times 22282, where all the entries are drawn independently from the normal distribution with mean value equal to 0 and standard deviation equal to 1:

```
A = matrix(rnorm(132*22282),nrow=132,ncol=22282)
```

Data Frames

For a vector or a matrix in R all elements have to be of the same type (numeric, integer, character, logical, etc).

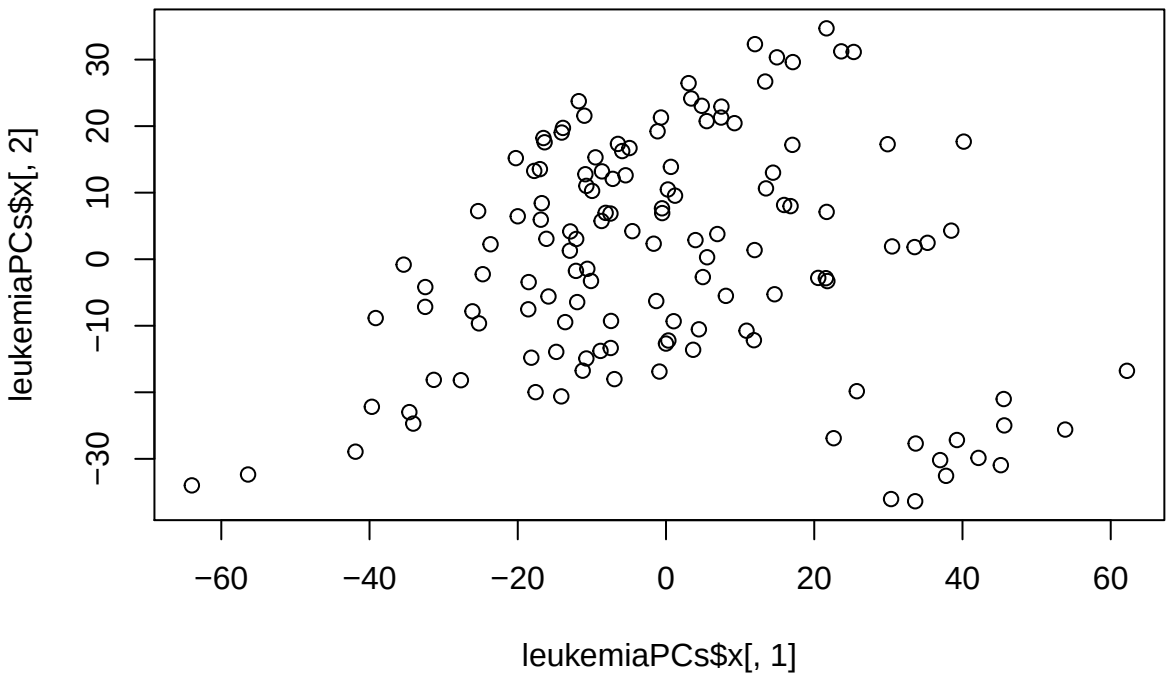
A **list** in R is a vector where the elements are R objects that are allowed to be of different types.

A dataframe is a two dimensional list where columns have there own names and are allowed to be of different types and we use the function **data.frame** to construct them.

Principal Components Analysis

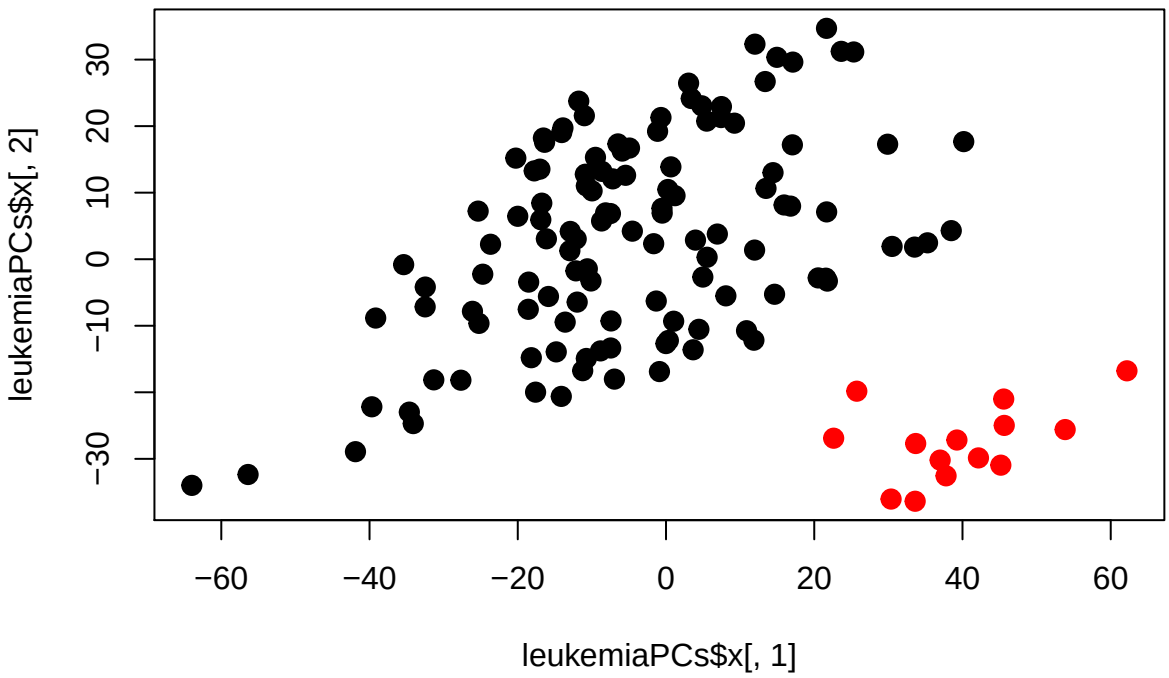
PCA is performed using the function **prcomp**

```
leukemiaPCs = prcomp(leukemia, center=TRUE,scale=FALSE)
plot(leukemiaPCs$x[,1], leukemiaPCs$x[,2])
```



We now color according to **leukemia type**. An annotation that is one of the columns (variables) in the data frame **leukemiaSA**. **leukemiaSA** has the same number of rows (cases) as the **leukemia** data frame.

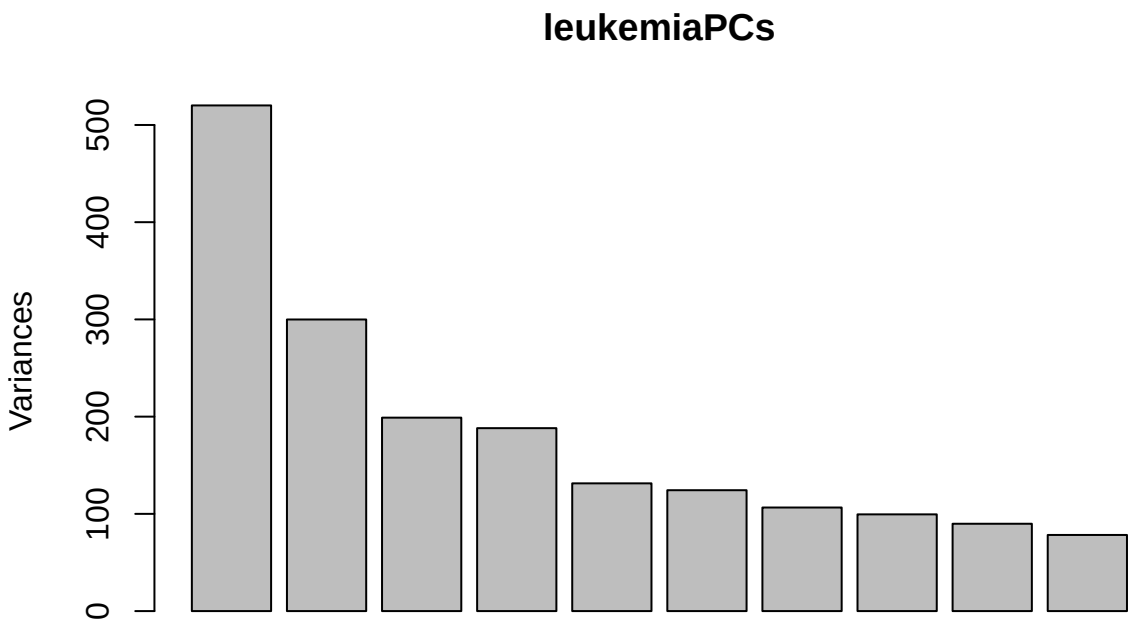
```
type = leukemiaSA[,"Leukemia.Type"]
plot(leukemiaPCs$x[,1],leukemiaPCs$x[,2],col
     =type,pch=20,cex=2)
```



PCA separates the samples in two distinct groups, B-ALL and T-ALL.

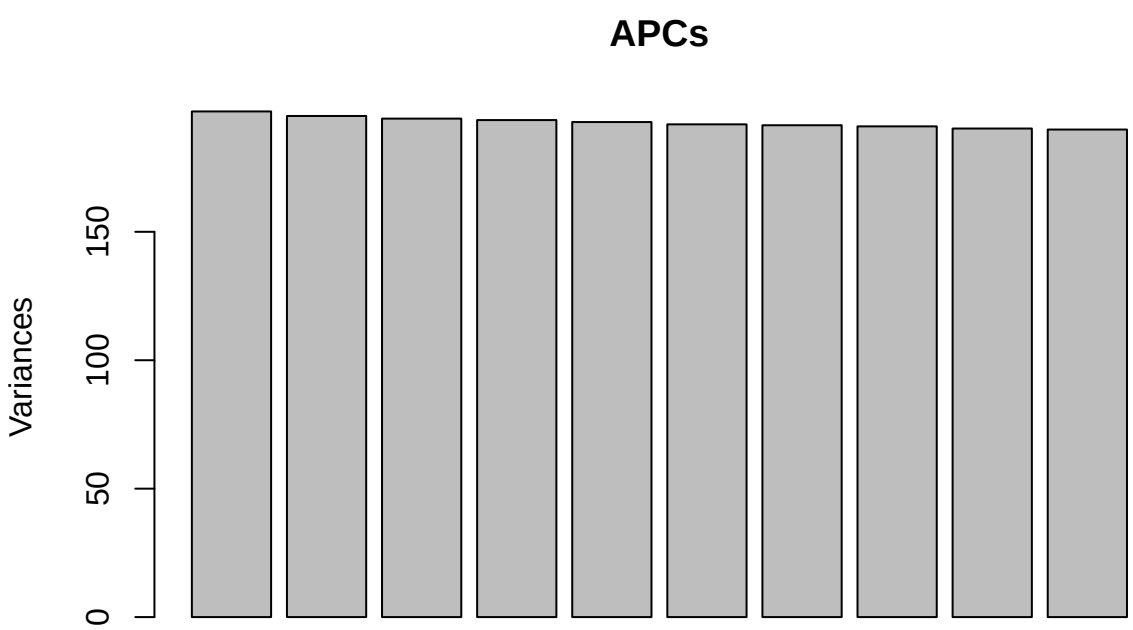
We now make a *screeplot* of the loadings for the PCs:

```
screeplot(leukemiaPCs)
```



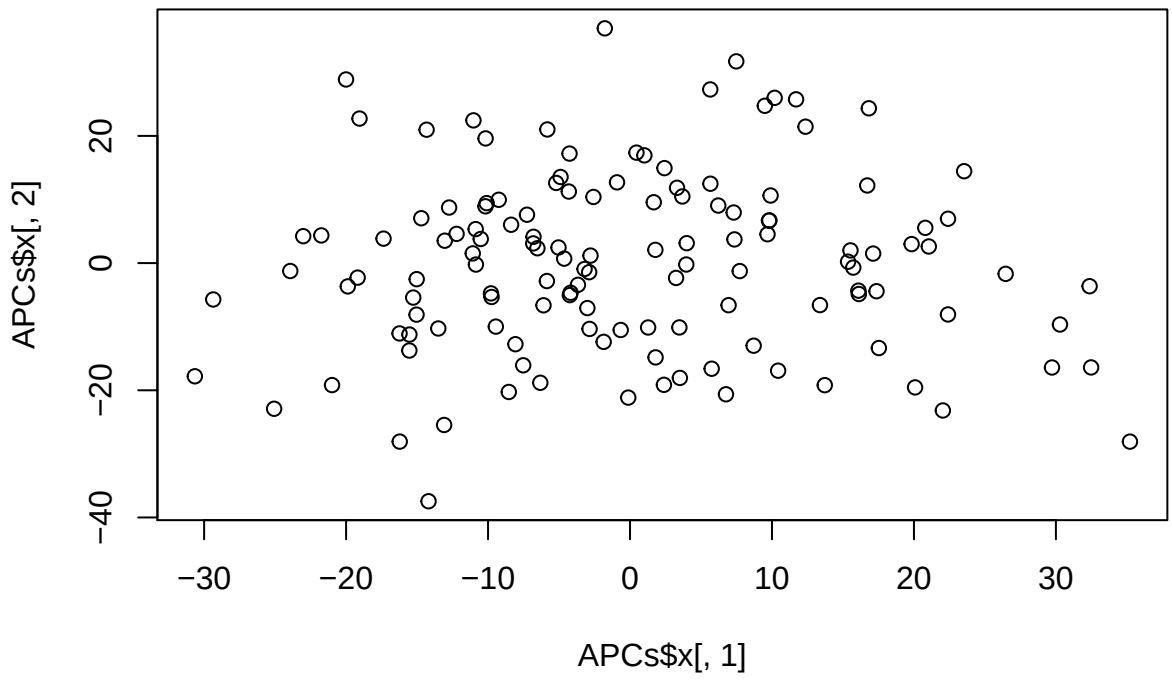
And compare with a screeplot of the loadings for our random matrix A:

```
APCs = prcomp(A, center=TRUE,scale=FALSE)
screeplot(APCs)
```



And a PCA plot of the random matrix **A**:

```
plot(APCs$x[,1], APCs$x[,2])
```



We now create a distance matrix using a Minkowski l–p distance:

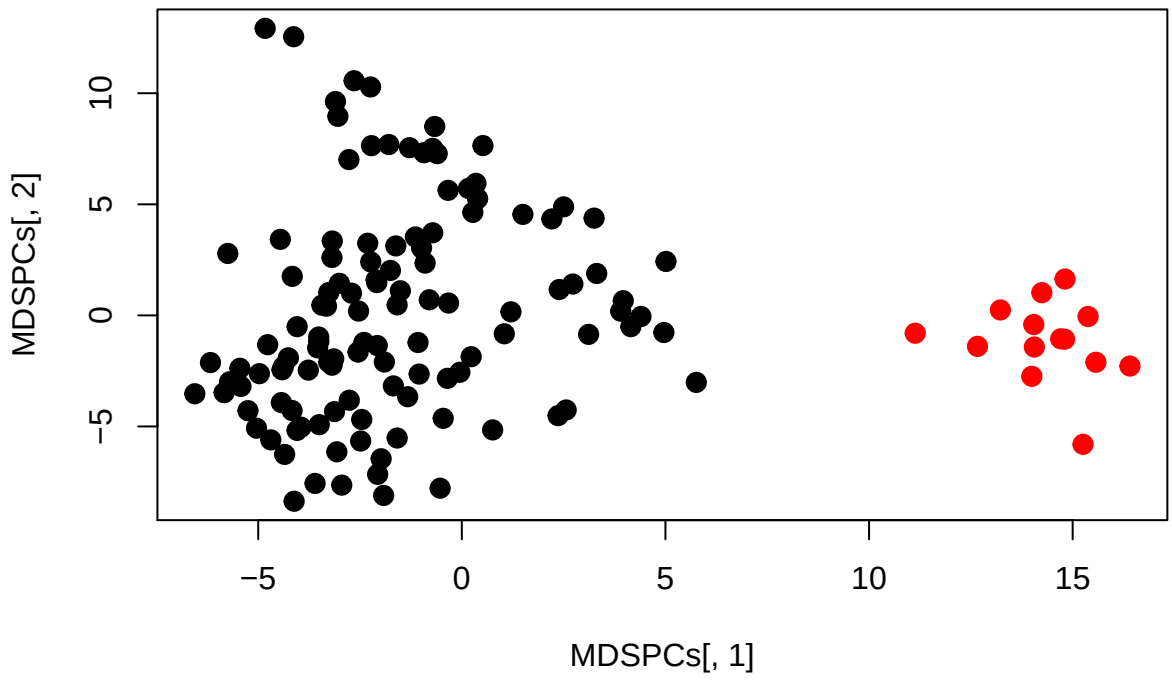
```
D <-
  dist(leukemia,method=
         "minkowski",diag=FALSE,upper=FALSE,p=3)
```

And apply the function **cmdscale** performing classical multidimensional scaling

```
MDSPCs <-
  cmdscale(D, k = 2, eig = FALSE, add =
             FALSE, x.ret = FALSE)
```

And make a scatterplot of the result:

```
plot(MDSPCs[,1], MDSPCs[,2], col=type, pch=20,cex=2)
```



Loading a package

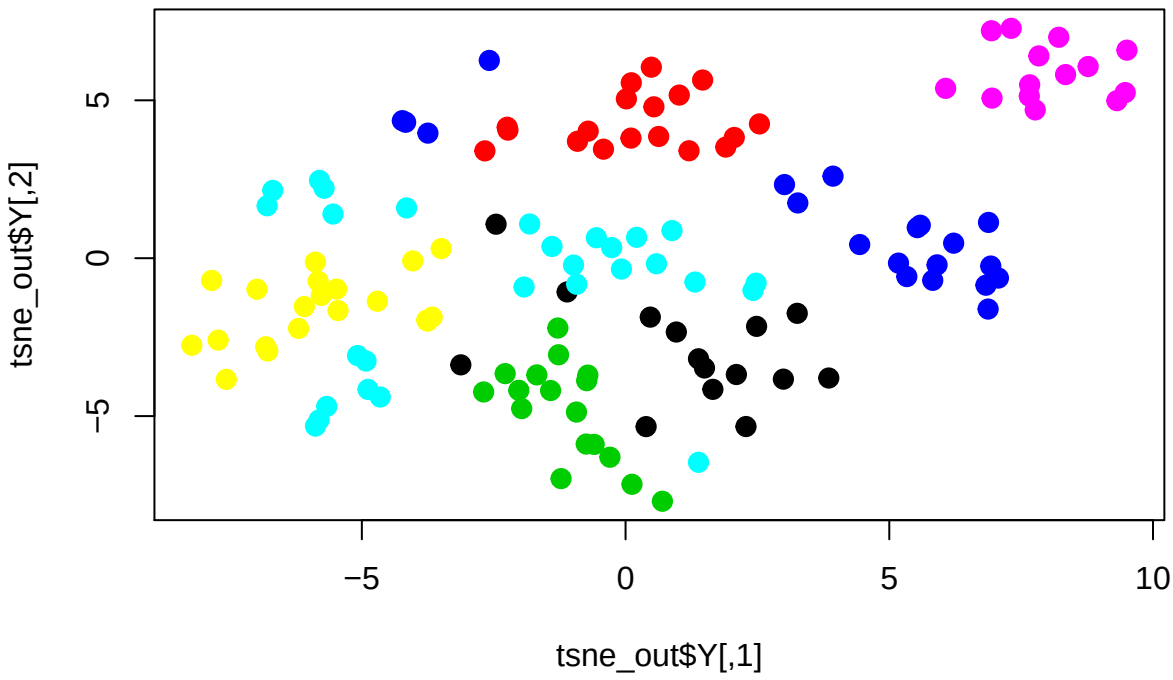
We assume that we have downloaded and installed the package **tSNE**.

We can then load the package **Rtsne** into the active memory using the library function:

```
library("Rtsne", lib.loc="~/R/win-library/3.1")
```

We perform tSNE

```
tsne_out <- Rtsne(leukemia)
type=leukemiaSA[,"Leukemia.Subtype"]
plot(tsne_out$Y, col=type, pch=20,cex=2)
```

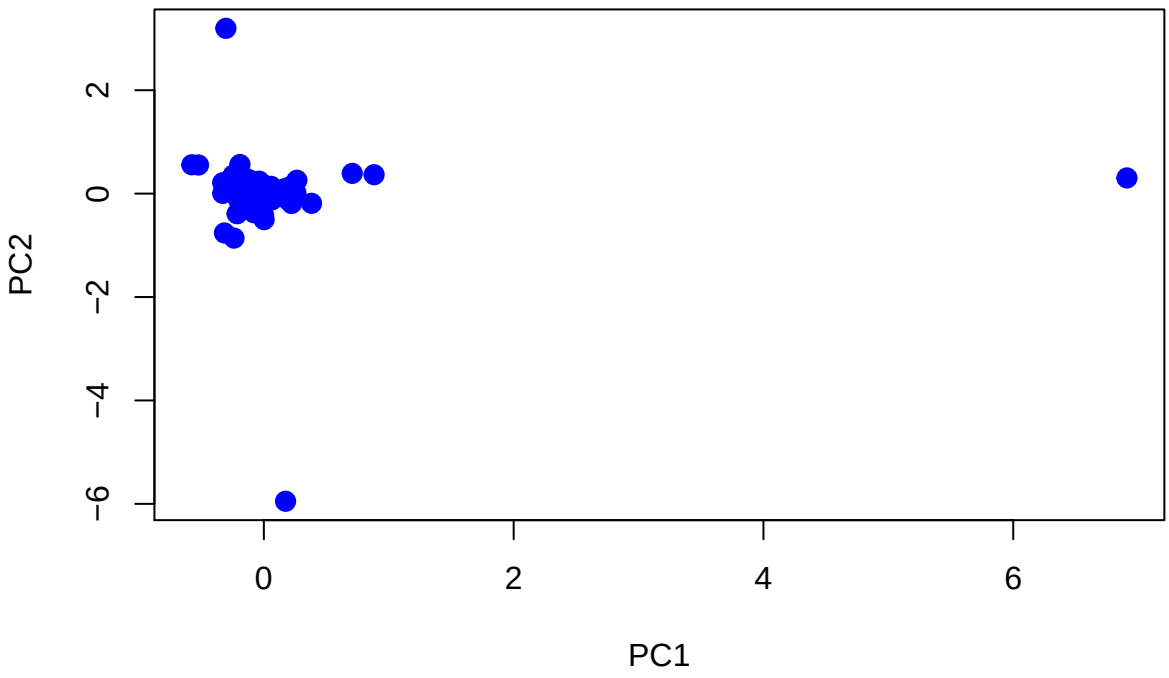


We create a random matrix with N samples and M normally distributed random variables

```
N=132
M=22282
random.matrix <- matrix(rnorm(N*M,mean=0,sd=1), N, M)
```

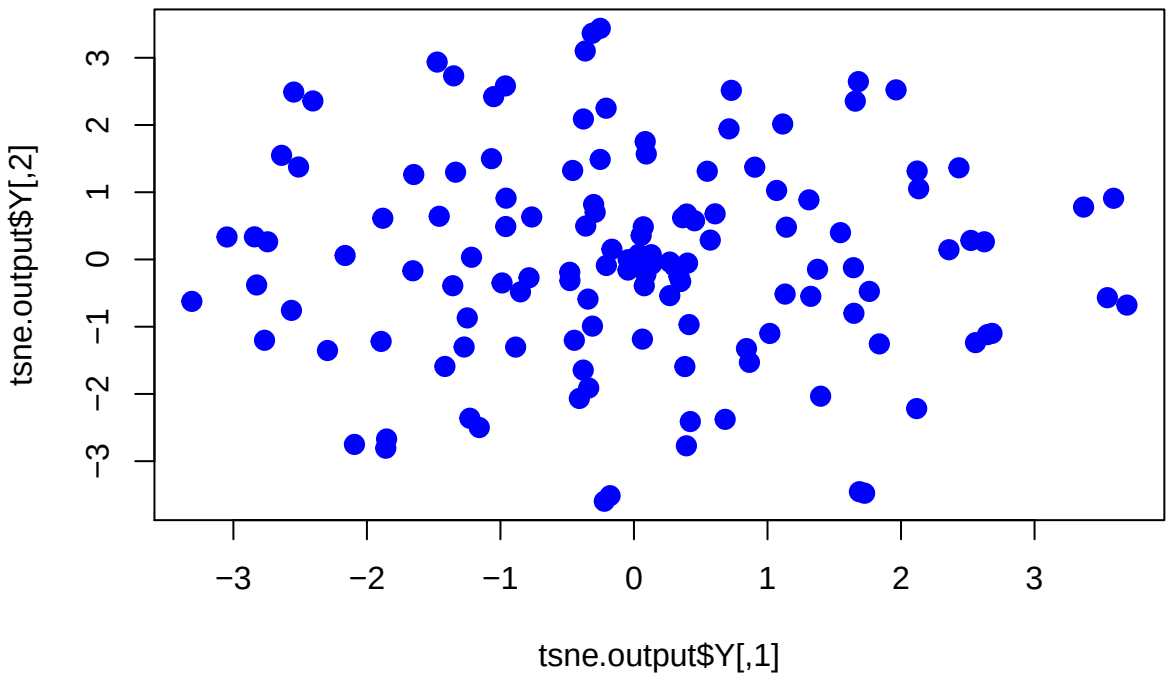
We threshold the matrix

```
random.matrix[ random.matrix < 2.4 ] = 2.4
prcomp.output <- prcomp(random.matrix)
plot(prcomp.output$x, col='blue', pch=20,cex=2)
```



And we compare with tSNE

```
tsne.output <- Rtsne(random.matrix)
plot(tsne.output$Y, col='blue', pch=20,cex=2)
```


tsne.output$Y[,2]
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
tsne.output$Y[,1]