Crash n' Survival Course with R

Ricardo de Matos Simoes
Tanja Gesell

MFPL - CIBIV
Center for Integrative Bioinformatics in Vienna

February 6, 2008

ricardo.de.matos.simoes(at)univie.ac.at

tanja.gesell(at)univie.ac.at

Ricardo: "Your biggest friend and partner in statistics"

Today

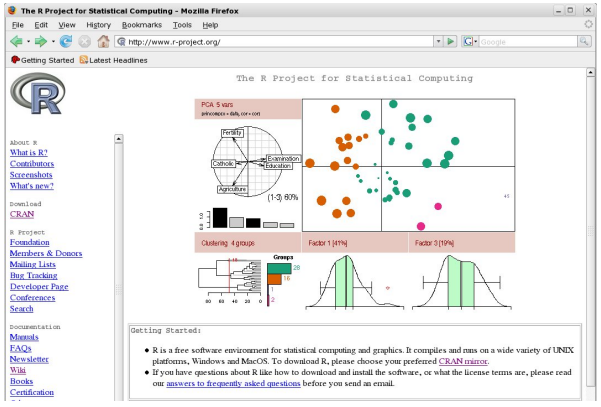
Data structure

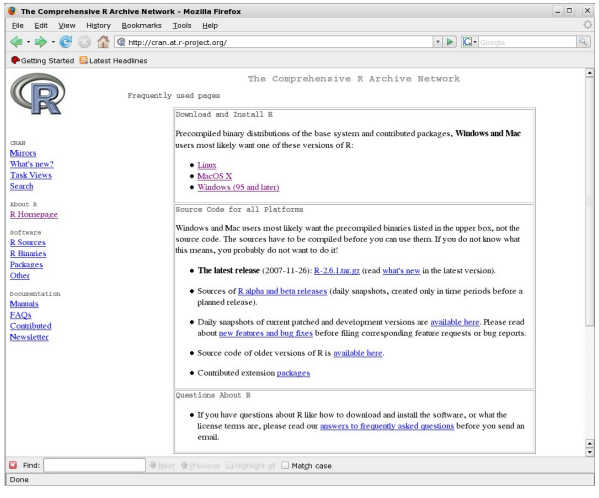
- Scalar
- Vector
- Matrices
- Arrays
- Factors
- Data frames
- Lists

Graphics

R script

The Installation of R





- Go to R's Webpage http://www.r-project.org/
- look out for precompiled binaries for your system

Using R as calculator: Basic Operations

^ or **

```
> 5**2 # power
[1] 25
```

* , /

```
> 5*2 # multiplication, division
[1] 10
```

+ , -

```
> 5-2 # addition, subtraction
[1] 3
```

%/%

```
> 5%/%2 # division
[1] 2
```

%%

```
> 5%%2 # modulo
[1] 1
```

Basic arithmetic functions and values

predefined arithmetic functions:

max(), min()
abs()
sqrt()
round, floor(), ceiling()
sum(), prod()
log(), log10(), log2()
exp()
sin(), cos(), tan(), asin(), acos(), atan()

Constant Values:

pi	π
Inf, -Inf	infinity
NaN	Not a Number
NA	Not Available
NULL	Not defined

Assigning values to variables

We do not need to declare the datatype of a variable in R.

The simplest datatype in R is a scalar.

- variables can be used to represent values:

```
> x <- 15
> x
[1] 15
```

- values can also be assigned by:

```
> a = 15
> a
[1] 15
```

- Logical values:

```
> a == 15
[1] TRUE
```

A Vector is a array of values

construct a vector with the "concatenate function" c()

Scalar vector:

```
> v <- c(15, 4, 67, 5, 9)
> v
[1] 15 4 67 5 9
```

Logical vector:

```
> v <- c(T, T, F, T)
> v
[1] TRUE TRUE FALSE TRUE
```

Character vector:

```
> v <- c("Dog", "Cat", "Mouse", "Human")
> v
[1] "Dog" "Cat" "Mouse" "Human"
```

Operations and selections on vectors

- single value from vector:

```
> x[2]
[1] 5
```

- arithmetic operations on vectors:

```
> x <- c(4, 5, 8, 23, 12)
> x*2
[1]  8 10 16 46 24
```

- Vectors need to have the same length!

```
> x + x
[1]  8 10 16 46 24
> x + x[1:4]
[1]  8 10 16 46 16
Warning message:
longer object length
        is not a multiple of shorter object length in: x + x[1:4]
```

- logical selection:

```
> x[x>7]
[1]  8 23 12
```

Other functions to create vectors

- integer sequence ":"

```
> v1 <- 5:10
> v1
[1]  5  6  7  8  9 10
```

- seq() - sequence function

```
> seq(0, 1, 0.25)
[1] 0.00 0.25 0.50 0.75 1.00
```

- rep() - repetition function

```
> rep(c(2, 7),c(1,2))
[1] 2 7 7
```

If you need information of a specific function:

```
> ?rep()
```

Matrices and Arrays are represented as vectors with dimensions

We can set or change dimensions with the function dim

```
> M <- 1:6
> dim(M) <- c(2,3)
> M
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices can be created using the matrix function

```
> matrix(1:6,nrow=2,byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

(here the Matrix will be filled in a rowwise fashion)

Useful functions that operate on matrices

Indexing

```
> MAT<-matrix(5:12,c(4,2),byrow=T)
> MAT[3]
[1] 9
> MAT[,1]
[1]  5  7  9 11
> MAT[1,]
[1] 5 6
```

```
> MAT
     [,1] [,2]
[1,]    5    6
[2,]    7    8
[3,]    9   10
[4,]   11   12
```

assign rownames and colnames

```
> rownames(M) <- LETTERS[1:2]
> colnames(M) <- LETTERS[1:3]
> M
  A B C
A 1 3 5
B 2 4 6
```

Matrix functions

Glue vectors together cbind(), rbind():

```
> v1 <- c(1,2,3,4)
> cbind(A=v1,B=1:4)
     A B
[1,] 1 1
[2,] 2 2
[3,] 3 3
[4,] 4 4
```

Using the transposition function t()

```
> M
  A B C
A 1 3 5
B 2 4 6
```

```
> t(M)
  A B
A 1 2
B 3 4
C 5 6
```

They are many more matrix functions (eigen(), solve() etc ...)

Factors: storing categorical variables

A factor consists of a vector of integers and a character vector

```
> pain <- c(0,3,2,2,1)
> fpain <- factor(pain,levels=0:3)
> levels(fpain) <- c("none","mild","medium","severe")
```

Assures the ordering of our categories
(otherwise alphabetical ordering)

```
> fpain
[1] none   severe medium medium mild
Levels: none mild medium severe
> as.numeric(fpain)
[1] 1 4 3 3 2
```

Data frames: Lists of vectors with the same length

Creating a data frame object

```
> vector1 <- c("Cat","Dog","Mouse","Fish")
> vector2 <- c("3","5","7","2")
> dframe<-data.frame(Spec=vector1,Num=as.numeric(vector2))
```

We can take a look at the structure (str()) of the data.frame:

```
> str(dframe)
'data.frame':   4 obs. of  2 variables:
 $ Spec: Factor w/ 4 levels "cat","dog",..: 1 2 4 3
 $ Num : num  3 5 7 2
```

Data frames: Selection

- get the names of a object

```
> names(dframe)
[1] "Spec" "Num"
```

- can be indexed like a matrix (# comment)

```
> dframe$Num[3] # equal dframe[[2]][3]
[1] 7
```

- mount a data.frame: attach() and detach()

```
> attach(dframe)
> Num[3]
[1] 7
```

Example datasets in R

Take a look at example datasets (please type data())

- For example choose "Edgar Anderson's Iris Data"

```
> help(iris)    # get information and details
> data(iris)    # load data
> attach(iris) # mount object (see names(iris))
```

- select a subset from our example set:

```
> iris[iris[["Species"]] == "versicolor",]
```

- now we select data from "iris versicolor" using subset()

```
> subset(iris,Species=="versicolor")
```

virginica

versicolor

setosa

Lists: composite collection of objects

- creating object using list()

```
> lobject<-list(c(1,2,2,2),age=c(98,21,56))
> lobject
[[1]]
[1] 1 2 2 2

object$age
[1] 98 21 56
```

- selection of particular values (Indexing)

```
> lobject[[2]][1]
[1] 98
> lobject$age[1]
[1] 98
```

How to read a datafile into R?

Simple example how to read a table into a data frame object:

```
> dataFrame1 <- read.table("data.txt")
```

A data frame can also be read columnwise from different files:

```
> values2 <- scan("file1.txt")
> values2 <- scan("file2.txt")
> dataFrame3 <- cbind(values1, values2)
```

You can also write a data.frame() into a file:

```
> write.table(iris,sep="\t","IRIS-DATA.txt")
```

Data Types and Structures

Scalar	:	simplest datatype
Vector	:	array of values (scalar, logical character)
Matrices	:	two-dimensional array
Arrays	:	array with dimensions
Factors	:	data structure with categorical variables
Data frames	:	list of vectors and/or factors of the same length
Lists	:	collection of objects into a larger composite object

Efficient programming: apply()

Use functions on objects without the need of loops!

```
apply(X, MARGIN, FUN, ...)
```

- X (array, matrix)
- MARGIN (1=rowwise, 2=columnwise, 3=more dimensions, ...)
- FUN (function)

Example: $arrayObject <- array(1:12, c(2,3,2))$

```
> apply(X=arrayObject,MARGIN=1,FUN=sum)
[1] 36 42
> apply(X=arrayObject,2,sum)
[1] 18 26 34
> apply(X=arrayObject,3,sum)
[1] 21 57
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Graphics: Device

"R: A Language for Data Analysis and *Graphics*" [1]

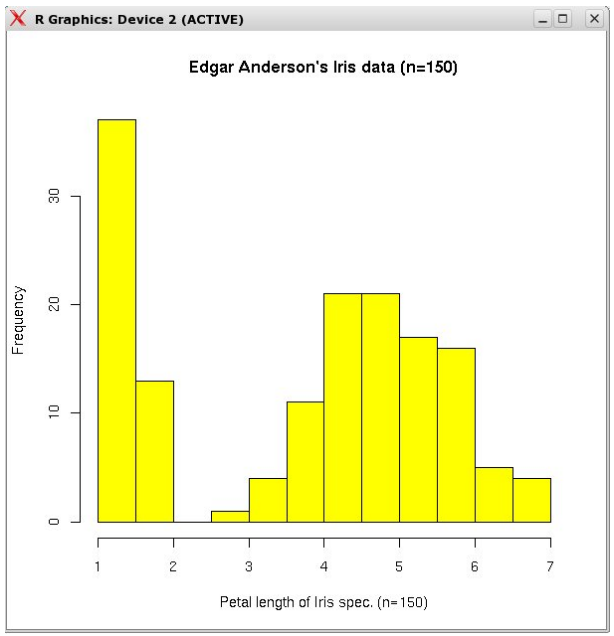
- jpeg()
- pdf()
- png()
- postscript()
- bmp()
- window()
  Don't forget to close device with dev.off()!

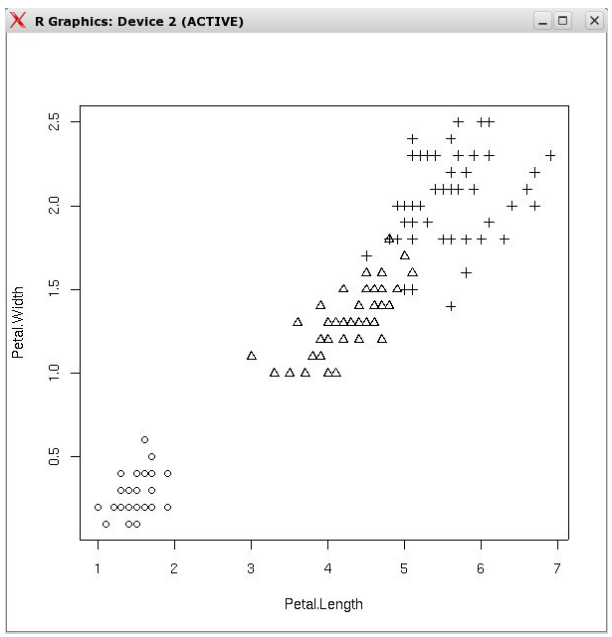
X11()
dev.print()

[1] Ihaka and Gentleman (1996)

High-level Graphics: E.g. Plots and Histograms

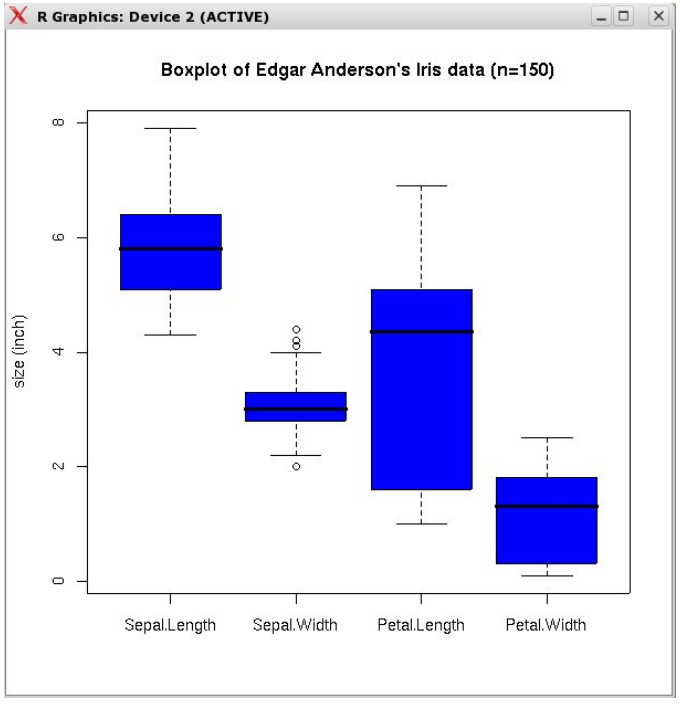
```
> data(iris)
> attach(iris)
> mtitle <- "Edgar Anderson's Iris data (n=150)"
> xaxis <- "length of petal"
> yaxis <- "Frequency"
> hist(Petal.Length,col="yellow",breaks=20,xlab=xaxis,ylab=yaxis,main=mtitle)
> plot(Petal.Length, Petal.Width, pch = as.numeric(Species))
> legend(2,2, levels(Species), pch=c(1:3))
> detach(iris)
```

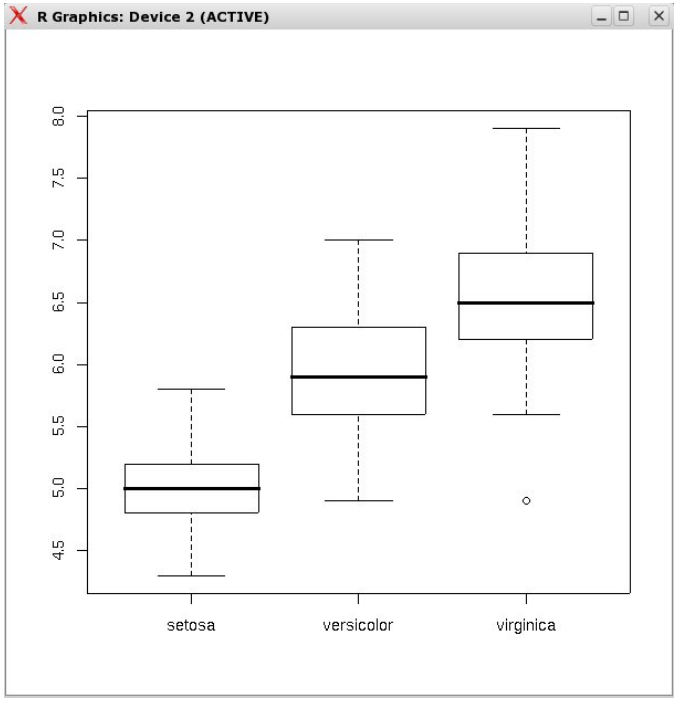




Creating a Boxplot from Edgar Anderson's Iris data

```
> boxplot(Sepal.Length ~ Species) #  "Sepal.length is described by Species"
> all<-data.frame(Sepal.Length,Sepal.Width,Petal.Length,Petal.Width)
> boxplot(all,
+col = "blue",
+main = "Boxplot of Edgar Anderson's Iris data (n=150)",
+ylab = "size (inch)")
```





Example: Microarrays

Read table into a data frame object:

```
>dataf=read.table("matrix_table")
```

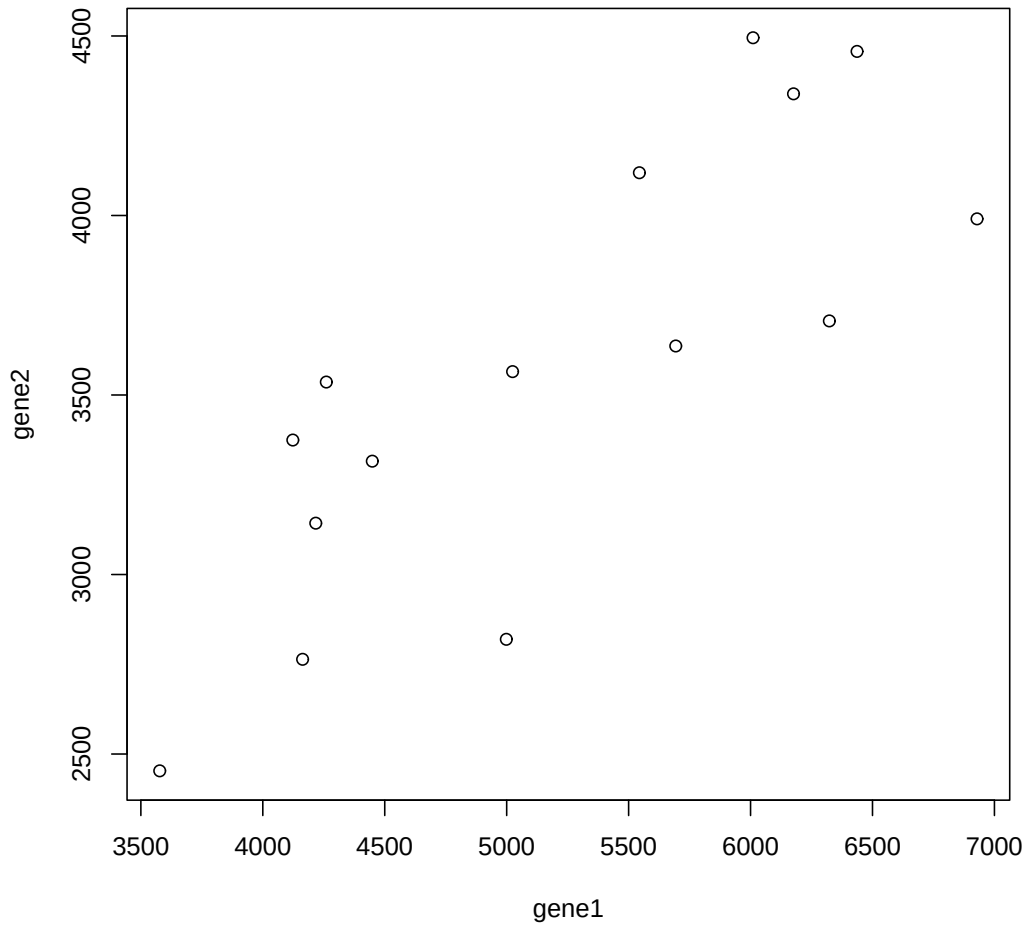
Example: Microarrays

```
>dataf=read.table("matrix_table")


>str(dataf)
> names(dataf)
 [1] "P20.H" "P21.H" "P24.H" "P33.H" "P34.H"
 "P35.L" "P37.L" "P41.H" "P43.H" "P47.H" "P48.L"
 "P50.L" "P52.L" "P55.H" "P57.L"
 >colnames(dataf)
>rownames(dataf)
> dim(dataf)
[1] 8793   15

> dataf["AFFX-HSAC07/X00351_3_at",]
                              P20.H    P21.H    P24.H    P33.H
AFFX-HSAC07/X00351_3_at 4260.364 4123.086 3577.371  .....
```

Example: Microarrays
Data Frame $->$ Matrix $->$ Plot

```
> mat=as.matrix(dataf)

> mat["AFFX-HSAC07/X00351_3_at",]
   P20.H    P21.H    P24.H    P33.H        .....
4260.364 4123.086 3577.371 4163.229 5693.291 .....

>gene1= mat["AFFX-HSAC07/X00351_3_at",]
>gene2= mat["208668_x_at",]

> names(mat)
NULL
> colnames(mat)
>rownames{mat)

> X11()
> plot(gene1, gene2)
```

Example: Microarrays - Plot

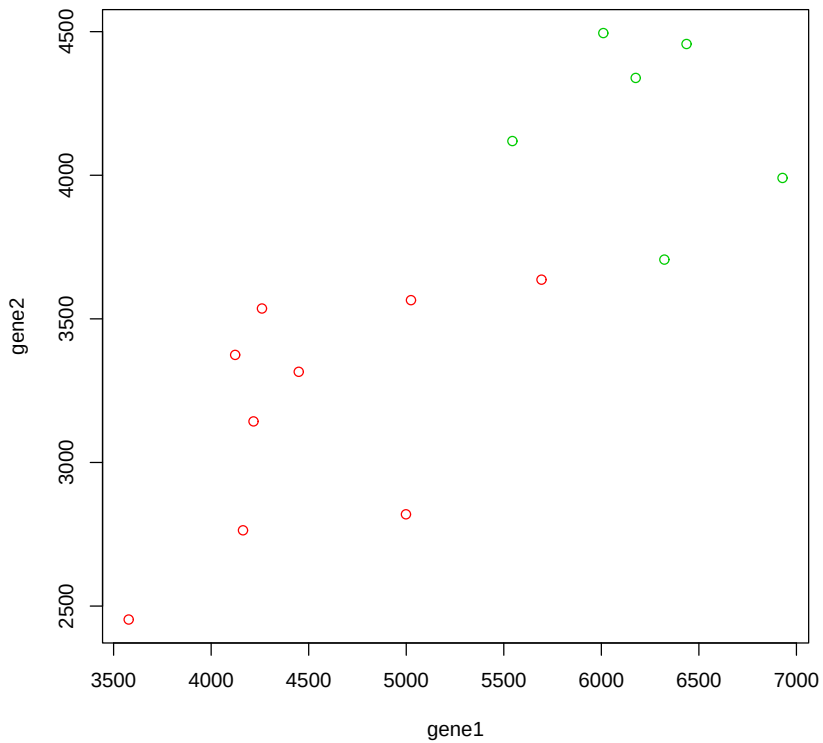


Example: Microarrays
Storing categorical variables $->$ Factors

```
> names(dataf)
 [1] "P20.H" "P21.H" "P24.H" "P33.H" "P34.H"
 "P35.L" "P37.L" "P41.H" "P43.H" "P47.H" "P48.L"
 "P50.L" "P52.L" "P55.H" "P57.L"

 > substr(names,5,5)
 [1] "H" "H" "H" "H" "H" "L" "L" "H" "H" ...

 > factors=factor(substr(names,5,5))
 [1] H H H H H L L H H H L L L H L
Levels: H L

>nfactors= as.integer(factors)
 [1] 1 1 1 1 1 2 2 1 1 1 2 2 2 1 2
```

Example: Microarrays - Plot

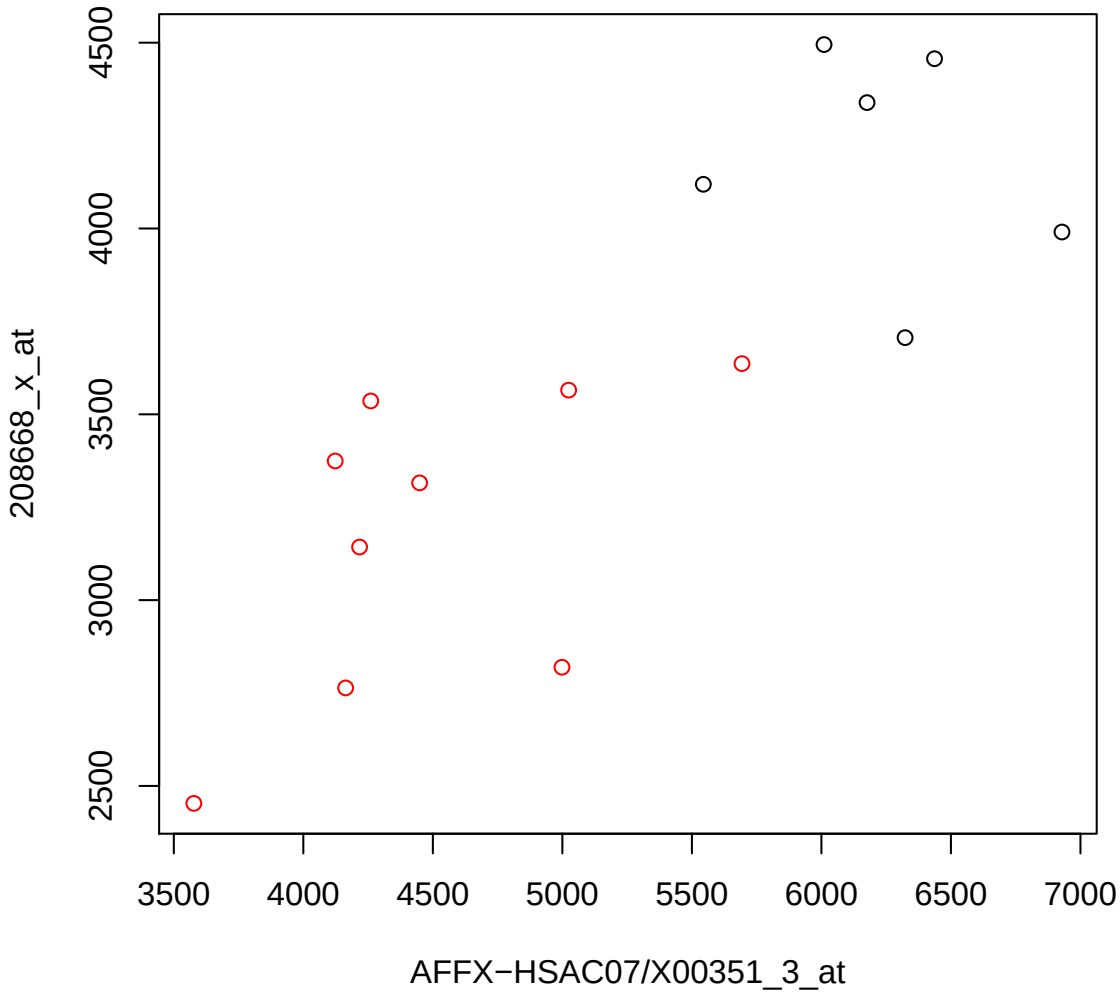
```
> plot(gene1,gene2, col=nfactors+1)
```



dev.print(file="example.eps")

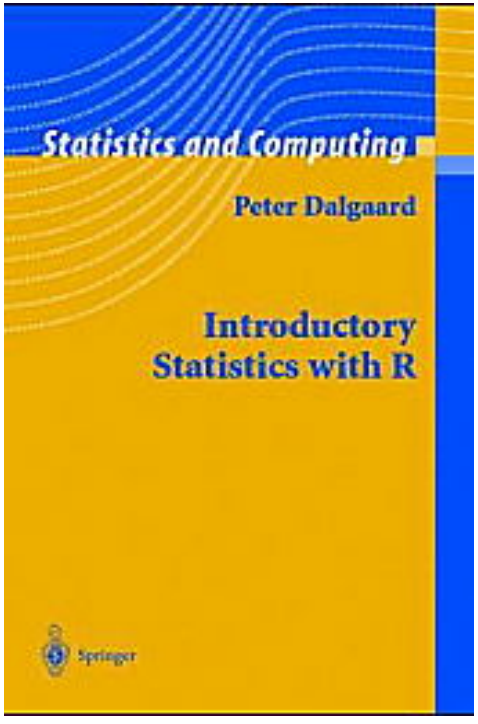
Example: Microarrays - R script

source("example.r")



Literature:

Main source of the information and for further reading:



There are also numerous free documentations of R.

You can also contact the r-boy ricardo.de.matos.simoes(at)univie.ac.at or alternative tanja.gesell(at)univie.ac.at ....

example.r

source("example.r")

```
pdf(file="example.pdf")
dataf=read.table("matrix_table")
mat=as.matrix(dataf)
names<-colnames(dataf)
factors=factor(substr(names,5,5))

genes=rownames(mat)
gene1name=genes[8756]
gene2name=genes[5072]
gene1=mat[gene1name,]
gene2=mat[gene2name,]

plot(gene1,gene2,col=as.integer(factors=="H")+1
        ,xlab=gene1name,ylab=gene2name)
dev.off()
```