

# Introduction to R and microarray analysis



Zhirui Hu  
Zack McCaw

Jan 27 & Jan 28, 2016



Welcome to RStudio - Open source  
and enterprise-ready professional  
software for R

[Download RStudio](#)[Discover Shiny](#)

### Powerful IDE for R

RStudio IDE is a powerful and productive user interface for R. It's free and open source, and works great on Windows, Mac, and Linux.

[Learn More >](#)

### R Packages

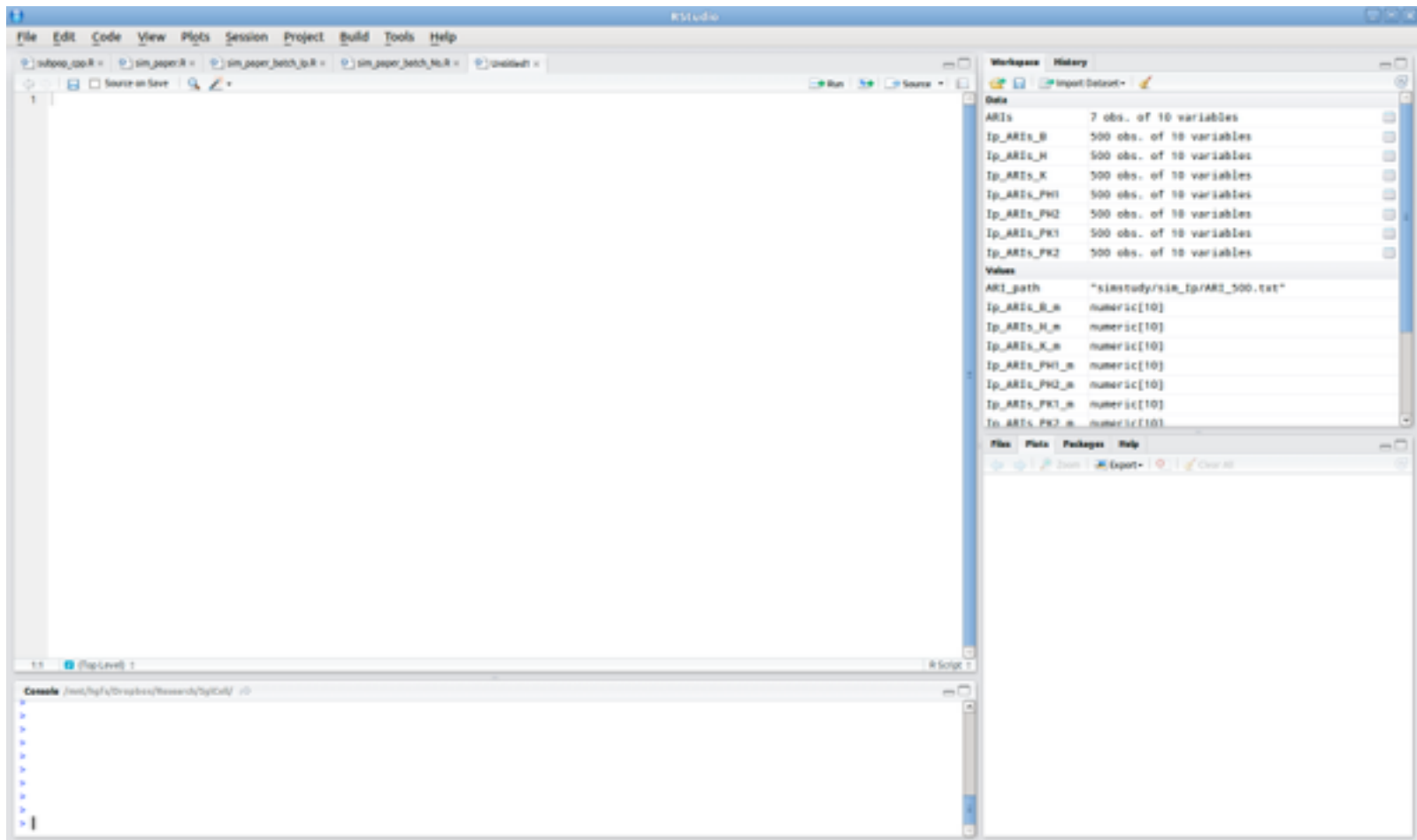
Our developers and expert trainers are the authors of several popular R packages, including ggplot2, plyr, lubridate, and others.

[Learn More >](#)

### Bring R to the web

Shiny is an elegant and powerful web framework for building interactive reports and visualizations using R — with or without web development skills.

[Learn More >](#)

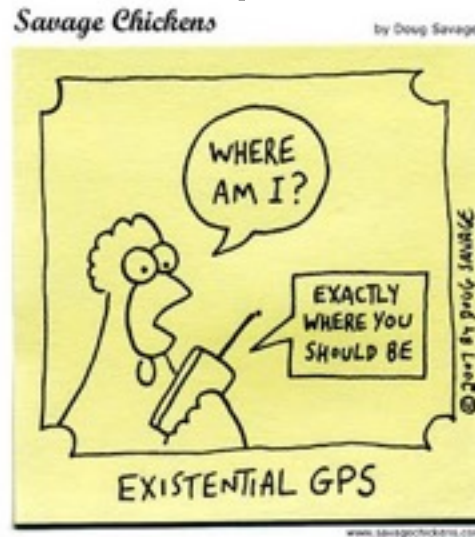


# Workspace Management

- Before jumping into R, it is important to ask ourselves

Where am I?

> getwd()



–I want to be there...

- `setwd("C://")`

–With who am I?

- `dir()` # lists all the files in the working directory

–With who I can count on?

- `ls()` #lists all the variables on the current session

# Workplace Management (2)

## Saving

- > `save(x,file="name.RData")` #Saves specific objects
- > `save.image("name.Rdata")` #Saves the whole workspace

## Loading

- > `load("name.Rdata")`

## ‘?function’ and ‘??function’

- > ? To get the documentation of the function
- > ?? Find related functions to the query

# R Objects

- Almost all things in R are OBJECTS!
  - Functions, datasets, results, etc... (graphs NO)
- OBJECTS are classified by two criteria
  - **MODE**: How objects are **stored** in R
    - Character, numeric, logical, factor, list, function...
    - To obtain the mode of an object
      - > **mode(object)**
  - **CLASS**: How objects are **treated** by functions
    - Vector, matrix, array, data.frame,...
    - To obtain the class of an object
      - > **class(object)**

# R classes

## character

```
> assembly = "hg19"  
> assembly  
> class(assembly)
```

## numeric

```
> expression = 3.456  
> expression  
> class(expression)
```

## integer

```
> nbases = "3000000000L"  
> nbases  
> class(nbases)
```

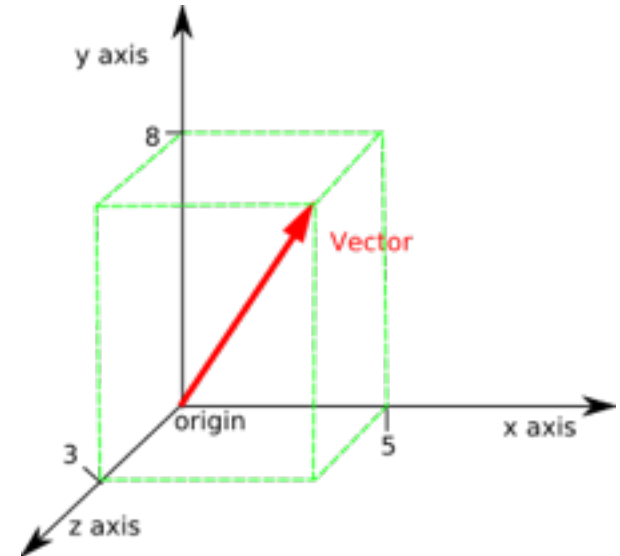
## logical

```
> completed = FALSE  
> completed  
> class(completed)
```

# R classes - Vector

## vector

- > `x=c(10,5,3,6); x[3:4]; x[1]`
- > Computations on vector are performed on each entry of the vector
- > `y=c(log(x),x,x^2)`
- > Not necessarily to have vectors of the same length in operations!
- > `w=sqrt(x)+2`
- > `z=c(pi,exp(1),sqrt(2))`
- > `x+z`
- > **Logical vectors**
- > `aux=x<7`





# R Classes - List

## list

> A vector of values of possibly different classes and different length.

### > **Creating it.**

```
> x1 = 1:5
```

```
> x2 = c(T,T,F,T,F)
```

```
> y=list(question.number = x1, question.answer = x2)
```

```
> x = vector("list", 5); x = list(); x[[1]] <- x1; x[["a"]] <- x1
```

### > **Accessing it.**

```
> y;class(y)
```

```
> y$question.answer[3]; y[[2]][3]; y[["question.answer"]][3]
```

```
> y$question.number[which(question.answer == T)]
```

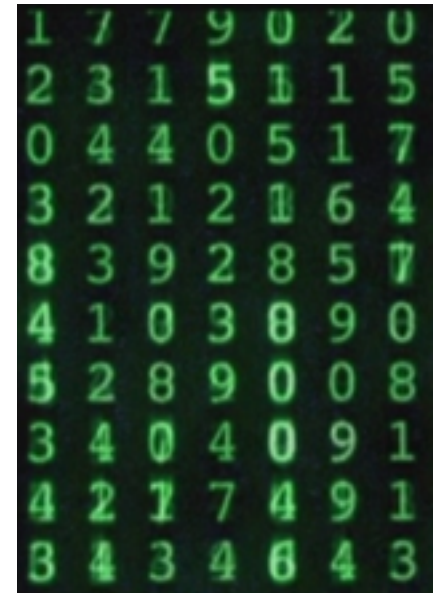
# R classes - Matrix

## matrix

```
> x=1:8  
> dim(x)=c(2,4)  
> y=matrix(1:8,2,4,byrow=F)
```

Operations are applied on each element

```
> x*x; max(x)  
> x=matrix(1:28,ncol=4); y=7:10 so then x*y  
  is...?  
> y=matrix(1:8,ncol=2)  
> y%*%t(y)
```



1	7	7	9	0	2	0
2	3	1	5	1	1	5
0	4	4	0	5	1	7
3	2	1	2	1	6	4
8	3	9	2	8	5	7
4	1	0	3	8	9	0
5	2	8	9	0	0	8
3	4	0	4	0	9	1
4	2	7	7	4	9	1
3	4	3	4	6	4	3

# R classes - Matrix

## matrix

Extracting info

```
>y[1,] or y[,1]
```

Extending matrices

```
>cbind(y,seq(101,104))
```

```
>rbind(y,c(102,109))
```

Apply is a useful function!

```
>apply(y,2,mean)
```

```
#library(matrixStats)
```

```
rowMeans(x)
```

```
>apply(y,1,log)
```

# R classes – Data Frame



## data.frame

Policy Numb	Issue Age	Sex	Smok	Face Amount
A00187	74	F	S	420,000
A00300	30	M	N	1,560,000
A00467	68	M	N	960,000
A01226	74	F	N	1,190,000
A01495	62	M	N	1,330,000

Creating it.

- > policy.number = c("A00187", "A00300", "A00467", "A01226")
- > issue.age = c(74, 30, 68, 74)
- > sex = c("F", "M", "M", "F")
- > smoke = c("S", "N", "N", "N")
- > face.amount = c(420, 1560, 960, 1190)
- > ins.df = data.frame(policy.number, issue.age, sex, smoke, face.amount)

Accessing it.

- > ins.df[1,]; ins.df[,1] # access first row, access first column
- > ins.df\$policy.number # access policy number column
- > rownames(ins.df); colnames(ins.df);
- > index.smokers = which(ins.df\$smoke == "S") # row index of smokers
- > ins.df[index.smokers] # access all smokers in the df
- > ins.df\$policy.number[index.smokers] # policy number for smokers

# R classes – Data Frame



## data.frame

Policy Numb	Issue Age	Sex	Smok	F
A00187	74	F	S	420,000
A00300	30	M	N	1,560,000
A00467	68	M	N	960,000
A01226	74	F	N	1,190,000
A01495	62	M	N	1,330,000

Manipulating it.

- > ins.df = rbind(ins.df, c("A01495", 62, "M", "N", 1330))
- > sort.age = sort(ins.df\$issue.age, index=T)
- > ins.df = ins.df[sort.age\$ix,]
- > ins.df[order(ins.df\$issue.age),]
- > ins.df\$visits = c(0,4,2,1,1)
- > drops = c("sex","visits")
- > ins.df[,!(names(ins.df) %in% drops)]
- > ins.df[, "visits"] = c(0,4,2,1,1)
- > carins.df = data.frame(policy.number = c("A01495","A00232","A00187"),  
car.accident = c("Y","N","N"))
- > ins.merged.df = merge(ins.df, carins.df, by = "policy.number")
- > Etc...

# R Classes - Factor

## factor

Qualitative variables that can be included in models.

```
> smoke = c("yes", "no", "yes", "no")  
> smoke.factor = as.factor(smoke)  
> smoke.factor  
> class(smoke)  
> class(smoke.factor)  
> levels(smoke.factor)  
> xtabs(~smoke.factor)
```

# Loops and Conditional Statements

if

## Example

```
> a=9  
> if(a<0){  
  print ("Negative number")  
} else{  
  print ("Non-negative number")  
}
```

# Loops and Conditional Statements

- **for**

```
> z=rep(1,10)
> for (i in 2:10)
    { z[i]=z[i]+exp(1)*z[i-1]
    }
```

- **while**

```
> n=0
> tmp=0
> while(tmp<100)
{
  tmp=tmp+rbinom(1,10,0.5)
  n=n+1
}
```



# Functions!

- My own functions

```
> function.name=function(arg1,arg2,...,argN)
    {
        Body of the function
    }
```

```
> fun.plot=function(y,z){
    y=log(y)*z-z^3+z^2
    plot(z,y)
}
```

```
> z=seq(-11,10)
```

```
> y=seq(11,32)
```

```
> fun.plot(y,z)
```

# Functions! (2)

- The '...' argument
  - Can be used to pass arguments from one function to another
    - Without the need to specify arguments in the header

```
fun.plot=function(y,z,...)
{ y=log(y)*z-z^3+z^2
  plot(z,y,...)
}
```



```
fun.plot(y,z,type="l",col="red")
fun.plot(y,z,type="l",col="red",lwd=4)
```



# Handling data I/O

## Reading from files to a data frame

- > `read.csv("filename.csv")` # reads csv files into a data.frame
- > `read.table("filename.txt")` # reads txt files in a table format to a data.frame

## Writing from a data frame to a file

- > `write(x,filename)` # writes the object x to filename
- > `write.table(x,filename)` # writes the object x to filename in a table format

Note: have in mind additional options such as, `header = TRUE`, `row.names = TRUE`, `col.names = TRUE`, `quotes = TRUE`, etc.

# Plotting!

```
> x.data=rnorm(1000)
> y.data=x.data^3-10*x.data^2
> z.data=-0.5*y.data-90

> plot(x.data,y.data,main="Title of the
  graph",xlab="x label",ylab="y label")
> points(x.data,z.data,col="red")
> legend(-2,2,legend=c("Black points","Red
  points"),col=c("black","red"),pch=1,text.col=c("bl
  ack","red"))
```

# Plotting! (2)

You can export graphs in many formats

png

```
> png("Lab2_plot.png",width=520,height=440)
> plot(x.data,y.data,main="Title of the graph",xlab="x
  label",ylab="y label")
> points(x.data,z.data,col="red")
> legend(-2,2,legend=c("Black points","Red
  points"),col=c("black","red"),pch=1,text.col=c("black","red"))
> dev.off()
```

eps

```
> postscript("Lab2_plot.eps",width=500,height=440)
```

# More Advanced Plot

- » ggplot2

- » e.g. Heatmap

- » 

```
ggplot(melt(Est.error), aes(as.factor(Var2),as.factor(Var1)))  
+geom_tile(aes(fill = value)) + scale_fill_gradient(low = "white", high =  
"red",guide="colourbar")+theme_bw()+xlab("p")+ylab("n")  
+ggtitle("Est error")
```

# Sampling Distributions

## Sampling

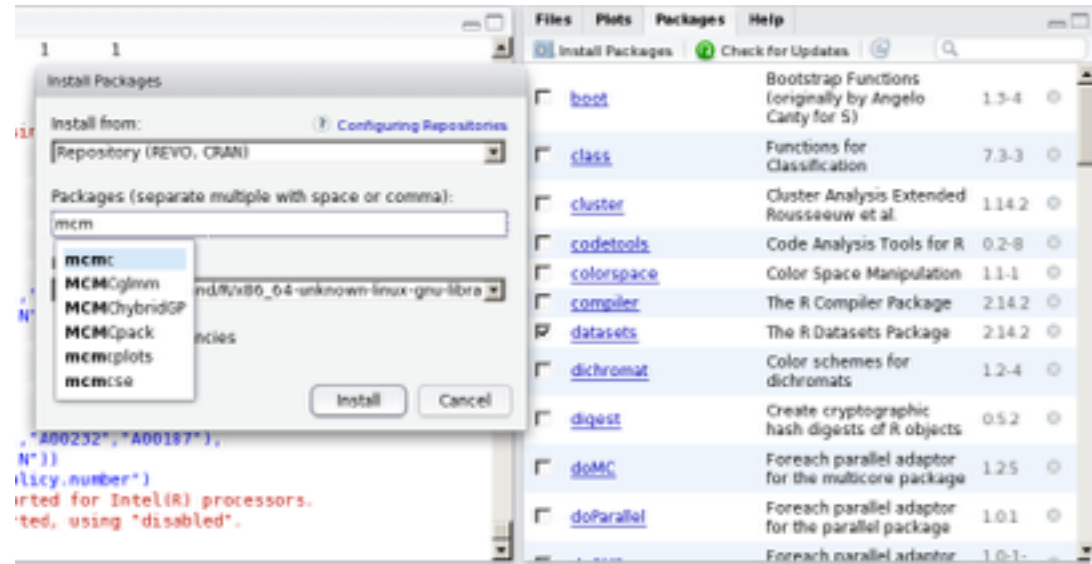
> sample(x,replace  
=TRUE) – put it  
back into the bag!

## Distributions

Distribution	Functions			
<a href="#">Beta</a>	pbeta	qbeta	dbeta	rbeta
<a href="#">Binomial</a>	pbinom	qbinom	dbinom	rbinom
<a href="#">Cauchy</a>	pcauchy	qcauchy	dcauchy	rcauchy
<a href="#">Chi-Square</a>	pchisq	qchisq	dchisq	rchisq
<a href="#">Exponential</a>	pexp	qexp	dexp	rexp
<a href="#">F</a>	pf	qf	df	rf
<a href="#">Gamma</a>	pgamma	qgamma	dgamma	rgamma
<a href="#">Geometric</a>	pgeom	qgeom	dgeom	rgeom
<a href="#">Hypergeometric</a>	phyper	qhyper	dhyper	rhyper
<a href="#">Logistic</a>	plogis	qlogis	dlogis	rlogis
<a href="#">Log Normal</a>	plnorm	qlnorm	dlnorm	rlnorm
<a href="#">Negative Binomial</a>	pnbinom	qnbinom	dnbinom	rnbinom
<a href="#">Normal</a>	pnorm	qnorm	dnorm	rnorm
<a href="#">Poisson</a>	ppois	qpois	dpois	rpois
<a href="#">Student t</a>	pt	qt	dt	rt
<a href="#">Studentized Range</a>	ptukey	qtukey	dtukey	rtukey
<a href="#">Uniform</a>	punif	qunif	dunif	runif
<a href="#">Weibull</a>	pweibull	qweibull	dweibull	rweibull
<a href="#">Wilcoxon Rank Sum Statistic</a>	pwilcox	qwilcox	dwilcox	rwilcox
<a href="#">Wilcoxon Signed Rank Statistic</a>	psignrank	qsignrank	dsignrank	rsignrank

# Libraries!!

Collection of R functions that together perform a specialized analysis.



Install packages from CRAN

```
> install.packages("PackageName")
```

Loading libraries

```
> library(LibraryName)
```

Getting the documentation of a library

```
> library(help=LibraryName)
```

Listing all the available packages

```
> library()
```



# Libraries!!

- [www.bioconductor.org](http://www.bioconductor.org)



- A suite of R packages for Bioinformatics.
- To use Core and Other packages
  - `>source("http://bioconductor.org/biocLite.R")`
  - `>biocLite(c("pkg1", "pkg2", ..., "pkgN"))`

# Online resources

R tutorial:

<http://www.cyclismo.org/tutorial/R/>

Quick-R:

<http://www.statmethods.net/>

Or simply Google “R tutorials”!

# Microarray Analysis

Research article

Open Access

## Loss of p24 function in *Drosophila melanogaster* causes a stress response and increased levels of NF- $\kappa$ B-regulated gene products

Kara A Boltz and Ginger E Carney\*

Address: Department of Biology, Texas A&M University, College Station, TX, USA

Email: Kara A Boltz - kboltz@mail.bio.tamu.edu; Ginger E Carney\* - gcarney@mail.bio.tamu.edu

\* Corresponding author

In simple words, this paper's objective is to compare the gene expression profiles of drosophila females in different tissues - with and without a mutation on the p24 gene logjam (loj).

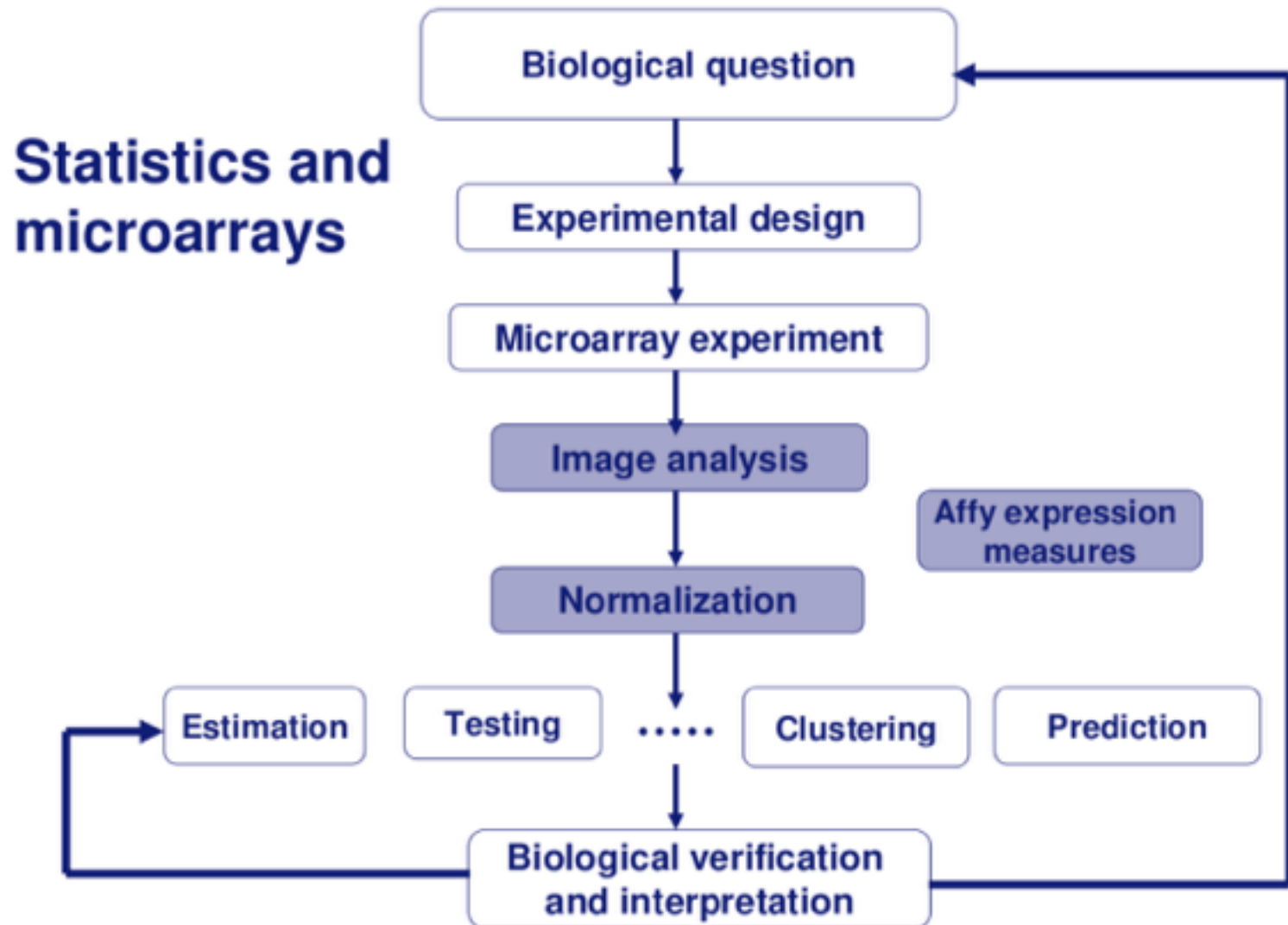
sample name	tissue	experiment
GSM277408	abd	control
GSM277409	abd	control
GSM277410	abd	control
GSM277411	ht	control
GSM277412	ht	control
GSM277413	ht	control
GSM277414	abd	treatment
GSM277415	abd	treatment
GSM277416	abd	treatment
GSM277417	ht	treatment
GSM277418	ht	treatment
GSM277419	ht	treatment

# Exercise – Microarray Analysis

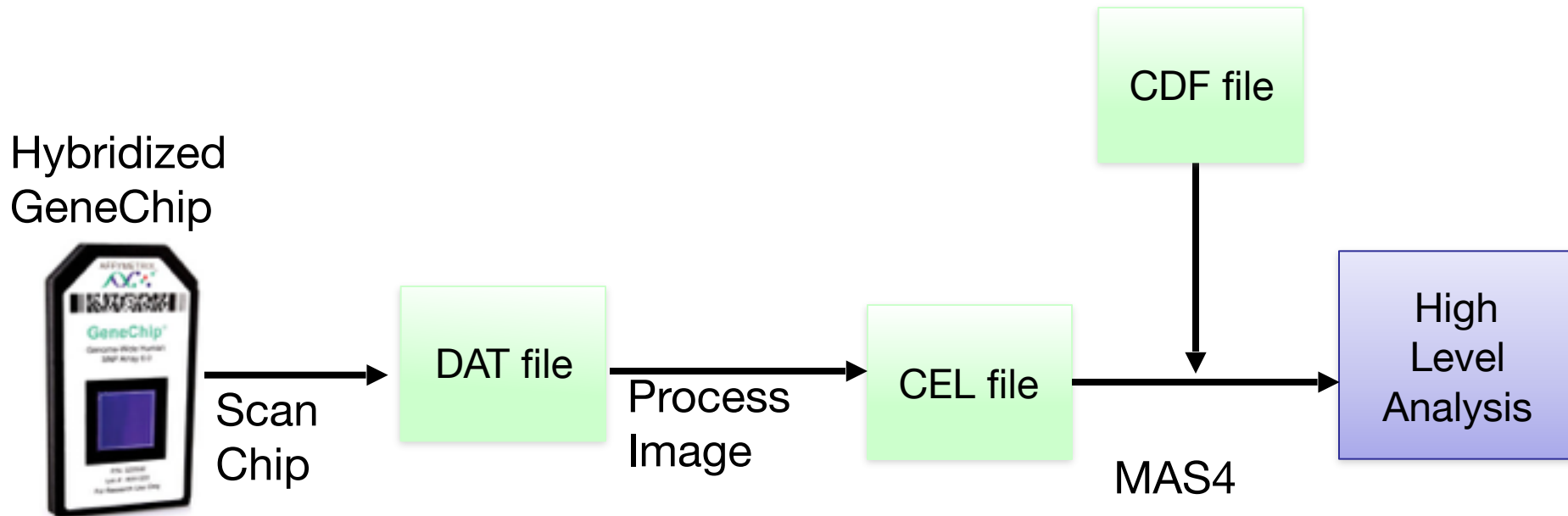
---

- (a) Read the raw microarray data and visualize it.
- (b) Normalize the microarray data using RMA
- (c) Find DE genes

# Microarray data analysis



# Affymetrix Data Flow



CEL — Data file: intensity values of the individual probes.

CDF — Library file: which probes belong to which probe set.

MAS4

MAS5

RMA

Quantile

# Installing Bioconductor packages


- For our analysis, we will need to install the affy, affyPLM, and limma packages.
  - `source("http://bioconductor.org/biocLite.R")`
  - `biocLite("affy")`
  - `biocLite("affyPLM")`
  - `biocLite("limma")`

# Installing Bioconductor packages

- For our analysis, we will need to install the affy, affyPLM, and limma packages.
  - `source("http://bioconductor.org/biocLite.R")`
  - `biocLite("affy")`
  - `biocLite("affyPLM")`
  - `biocLite("limma")`



# Microarray analysis

- Go to  and download the data set:
  - GSE10940
- <http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE10940>
  - Make sure to download the raw CEL files!
- The data set contains 12 .CEL files
  - `library(affy)`
  - `fns <- list.celfiles(path=cellPath,full.names=TRUE)`
  - `data.affy=ReadAffy(filenamees=fns)`
  - What is the name of the CDF file
  - How many probe sets are considered on the arrays?
  - What is the annotation version?

# Looking at RAW data

## Low-level analysis

- MA plot
  - `MAplot(data.affy, pairs = TRUE, which=c(1,2,3,4), plot.method = "smoothScatter")`
- Image of an array
  - `image(data.affy)`
- Density of the log intensities of the arrays
  - `hist(data.affy)`
- Boxplot of the data
  - `boxplot(data.affy, col=seq(2,7,by=1))`

# Normalization

- `data.rma=rma(data.affy)`
- `MAplot(data.rma, pairs = TRUE, which=c(1,2,3,4), plot.method = "smoothScatter")`
- `expr.rma=exprs(data.rma)` # Puts data in a table
- `boxplot(data.frame(expr.rma), col=seq(2,7,by=1))`
- Compare with raw data

# How does RMA (Robust Multiarray Average) work?

- » Background correction
  - »  $PM = \text{Signal} + \text{Background}$
  - » Background correction performed on each array separately
- » Quantile Normalization
- » Summarisation
  - » Sample effect + Probe effect + Noise
  - » Estimate model parameters by Median Polish

# Before moving forward... affy probeset names

➤ `rownames(expr.rma)[1:100]`

- Suffixes are meaningful, for example:
  - `_at` : hybridizes to unique antisense transcript for this chip
  - `_s_at`: all probes cross hybridize to a specified set of sequences
  - `_a_at`: all probes cross hybridize to a specified gene family
  - `_x_at`: at least some probes cross hybridize with other target sequences for this chip
  - `_r_at`: rules dropped
  - and many more...

# Annotation

- **How can we know each probe set mapped to which gene?**

- Refseq, ensembl, Gene symbol, entrez ID...

- **Download Annotation Data from Bioconductor for the chip type you are working on**

» ## for homework ##

» source("https://bioconductor.org/biocLite.R")

» biocLite("hgu133plus2.db")

» ## for lab##

» biocLite("drosophila2.db")

» library(drosophila2.db)

» **Put annotation information in a data frame.**

» **To get a list of available annotation information, run the package name with () at the end**

» my\_frame <- data.frame(expr.rma)

» Annot <- data.frame(REFSEQ=apply(contents(drosophila2REFSEQ), paste, collapse=" ", ),  
SYMBOL=apply(contents(drosophila2SYMBOL), paste, collapse=" ", ),  
DESC=apply(contents(drosophila2GENENAME), paste, collapse=" ", ))

» # Merge data frames together (like a database table join)

» all <- merge(Annot, my\_frame, by.x=0, by.y=0, all=T)

# Custom CDF files

- Genome and transcriptome annotation change over time.
- Oligonucleotide probes on GeneChips are reorganized based on the latest genome and transcriptome information.
- Go to: <http://brainarray.mbni.med.umich.edu/Brainarray/Database/CustomCDF/19.0.0/refseq.asp>

# High-level analysis

- Perform a comparison between the control group and the experimental group
  - **Objective: Obtain the most significant genes with an FDR of 5% and with a fold change of 1**
  - Use information from GEO to identify control and mutant columns
    - `control=c(1,2,3,4,5,6)`
    - `mutants=c(7,8,9,10,11,12)`



- Calculating the fold change for every gene
  - `foldchange=apply(expr.rma, 1, function(x) mean( x[mutants] ) - mean( x[control] ) )`
- Perform a t-test and obtain the p-values
  - `T.p.value=apply(expr.rma, 1, function(x) t.test( x[mutants], x[control], var.equal=T ) $p.value )`
- Calculating the FDR
  - `fdr=p.adjust(T.p.value, method="fdr")`
- **THE GENES**
  - `genes.up=genes.refseq[ which( fdr < 0.05 & foldchange > 0 ) ]`
  - `genes.down=genes.refseq [ which( fdr < 0.05 & foldchange <0 ) ]`

# Differential gene expression with limma

```
➤ library(limma)
```

- Create a design matrix using identified control/mutant columns

```
➤ design <- c(0,0,0,0,0,0,1,1,1,1,1,1)
```

```
➤ design.mat <- model.matrix(~design)
```

- Fit the model

```
➤ fit <- lmFit(expr.rma, design.mat)
```

```
➤ fit <- eBayes(fit)
```

- View results using topTable
  - topTable(fit, coef = 'design')
  - Defaults to top 10 differentially expressed genes
  - Use number to view more genes
    - topTable(fit, coef = 'design', number = 50)

# Two-factor Design

```
» TS = c("WT.U", "WT.S", "Mu.U", "Mu.S", "Mu.S")
» TS <- factor(TS, levels = c("WT.U", "WT.S", "Mu.U", "Mu.S"))
» design <- model.matrix(~0 + TS)
» colnames(design) <- levels(TS)
» fit <- lmFit(eset, design)
» cont.matrix <- makeContrasts(
»   WT.SvsU = WT.S - WT.U,
»   Mu.SvsU = Mu.S - Mu.U,
»   Diff = (Mu.S - Mu.U) - (WT.S - WT.U),
»   levels = design)
» fit2 <- contrasts.fit(fit, cont.matrix)
» fit2 <- eBayes(fit2)
```