Az R adatelemzési nyelv alapjai I.

Egészségügyi informatika és biostatisztika

Gézsi András
gezsi@mit.bme.hu

What R is and what it is not

- R is
  - a programming language
  - a statistical package
  - an interpreter
  - Open Source
- R is not
  - SPSS, Statistica, etc.
  - a collection of “black boxes”
  - a spreadsheet software package
  - commercially supported

What R is

- data handling and storage: numeric, textual
- matrix algebra
- regular expressions
- high-level data analytic and statistical functions
- classes (“OO”)
- graphics
- programming language: loops, branching, functions

What R is not

- has no click-point user interfaces
- language interpreter can be very slow, but allows to call own C/C++ code
- no spreadsheet view of data, but connects to Excel/MsOffice
- no professional /commercial support

R and statistics

- Packaging: a crucial infrastructure to efficiently produce, load and keep consistent software libraries from (many) different sources / authors
- Statistics: most packages deal with statistics and data analysis
- State of the art: many statistical researchers provide their methods as R packages

History of R

- Statistical programming language S developed at Bell Labs since 1976 (at the same time as UNIX)
- Intended to interactively support research and data analysis projects
- Exclusively licensed to Insightful ("S-Plus")
- R: Open source platform similar to S developed by R. Gentleman and R. Ihaka (U of Auckland, NZ) during the 1990s
- Since 1997: international "R-core" developing team
- Updated versions available every couple months

Getting started

- To obtain and install R on your computer
  - Go to http://cran.r-project.org/mirrors.html to choose a mirror near you
  - Click on your favorite operating system (Linux, Mac, or Windows)
  - Download and install the “base”
- To install additional packages
  - Start R on your computer
  - Choose the appropriate item from the “Packages” menu

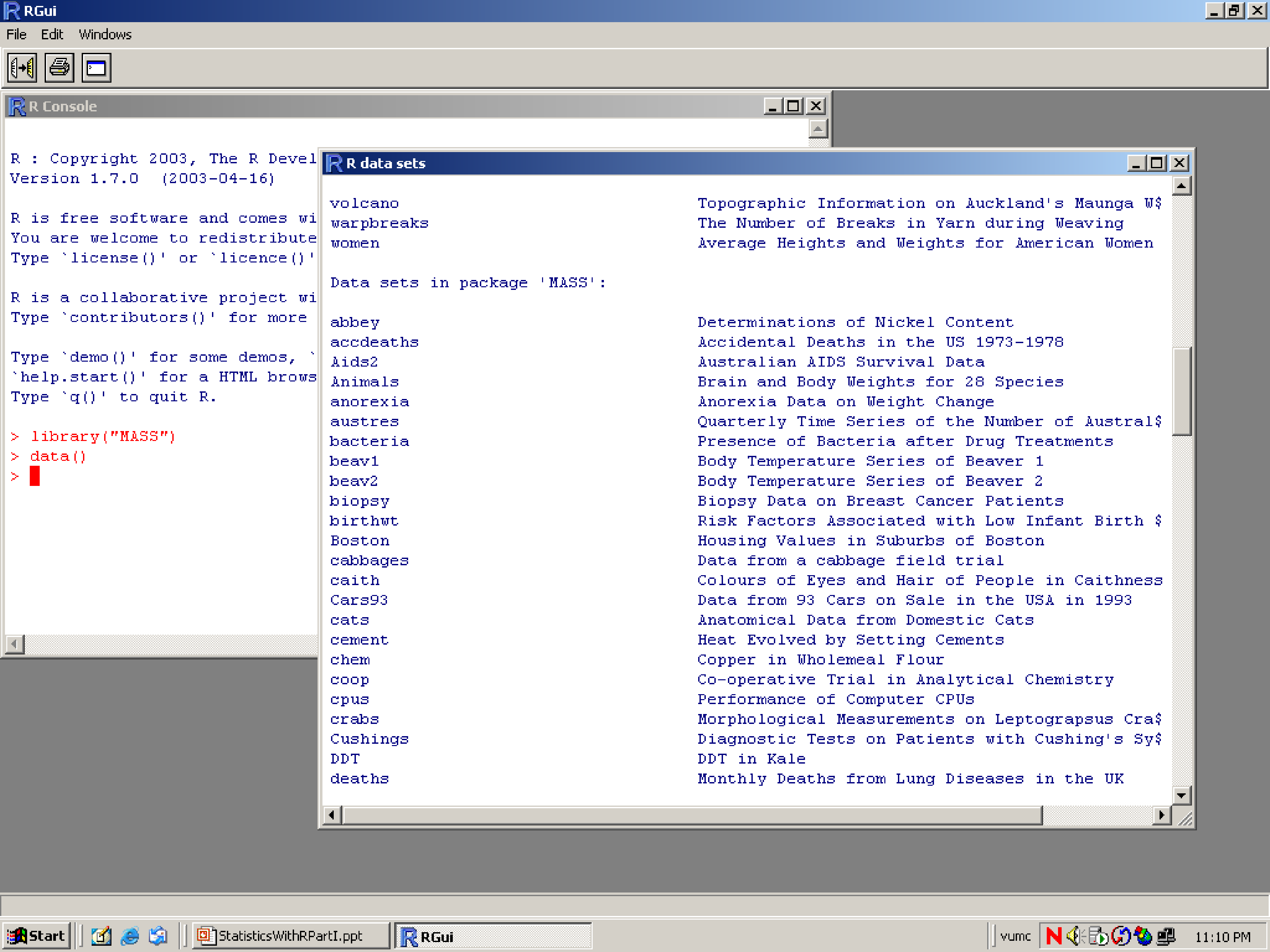
RGui - [R Console]

File Edit Misc Packages Windows Help

```
R : Copyright 2003, The R Development Core Team
Version 1.7.0  (2003-04-16)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type `license()' or `licence()' for distribution details.

R is a collaborative project with many contributors.
Type `contributors()' for more information.

Type `demo()' for some demos, `help()' for on-line help, or
`help.start()' for a HTML browser interface to help.
Type `q()' to quit R.

> library("MASS")
> 
```

R 1.7.0 - A Language and Environment

Start | StatisticsWithRPartI.ppt | RGui - [R Console] | vumc | 11:09 PM


RGui
File Edit Windows
R Console
R : Copyright 2003, The R Devel
Version 1.7.0 (2003-04-16)
R is free software and comes wi
You are welcome to redistribute
Type `license()' or `licence()'
R is a collaborative project wi
Type `contributors()' for more
Type `demo()' for some demos, `
`help.start()' for a HTML brows
Type `q()' to quit R.
> library("MASS")
> data()
>
R data sets
volcano Topographic Information on Auckland's Maunga W$
warpbreaks The Number of Breaks in Yarn during Weaving
women Average Heights and Weights for American Women
Data sets in package 'MASS':
abbey Determinations of Nickel Content
accdeaths Accidental Deaths in the US 1973-1978
Aids2 Australian AIDS Survival Data
Animals Brain and Body Weights for 28 Species
anorexia Anorexia Data on Weight Change
austres Quarterly Time Series of the Number of Austral$
bacteria Presence of Bacteria after Drug Treatments
beav1 Body Temperature Series of Beaver 1
beav2 Body Temperature Series of Beaver 2
biopsy Biopsy Data on Breast Cancer Patients
birthwt Risk Factors Associated with Low Infant Birth $
Boston Housing Values in Suburbs of Boston
cabbages Data from a cabbage field trial
caith Colours of Eyes and Hair of People in Caithness
Cars93 Data from 93 Cars on Sale in the USA in 1993
cats Anatomical Data from Domestic Cats
cement Heat Evolved by Setting Cements
chem Copper in Wholemeal Flour
coop Co-operative Trial in Analytical Chemistry
cpus Performance of Computer CPUs
crabs Morphological Measurements on Leptograpsus Cra$
Cushings Diagnostic Tests on Patients with Cushing's Sy$
DDT DDT in Kale
deaths Monthly Deaths from Lung Diseases in the UK
Start
StatisticsWithRPartI.ppt
RGui
vumc
11:10 PM

RGui - [R Console]

File Edit Misc Packages Windows Help

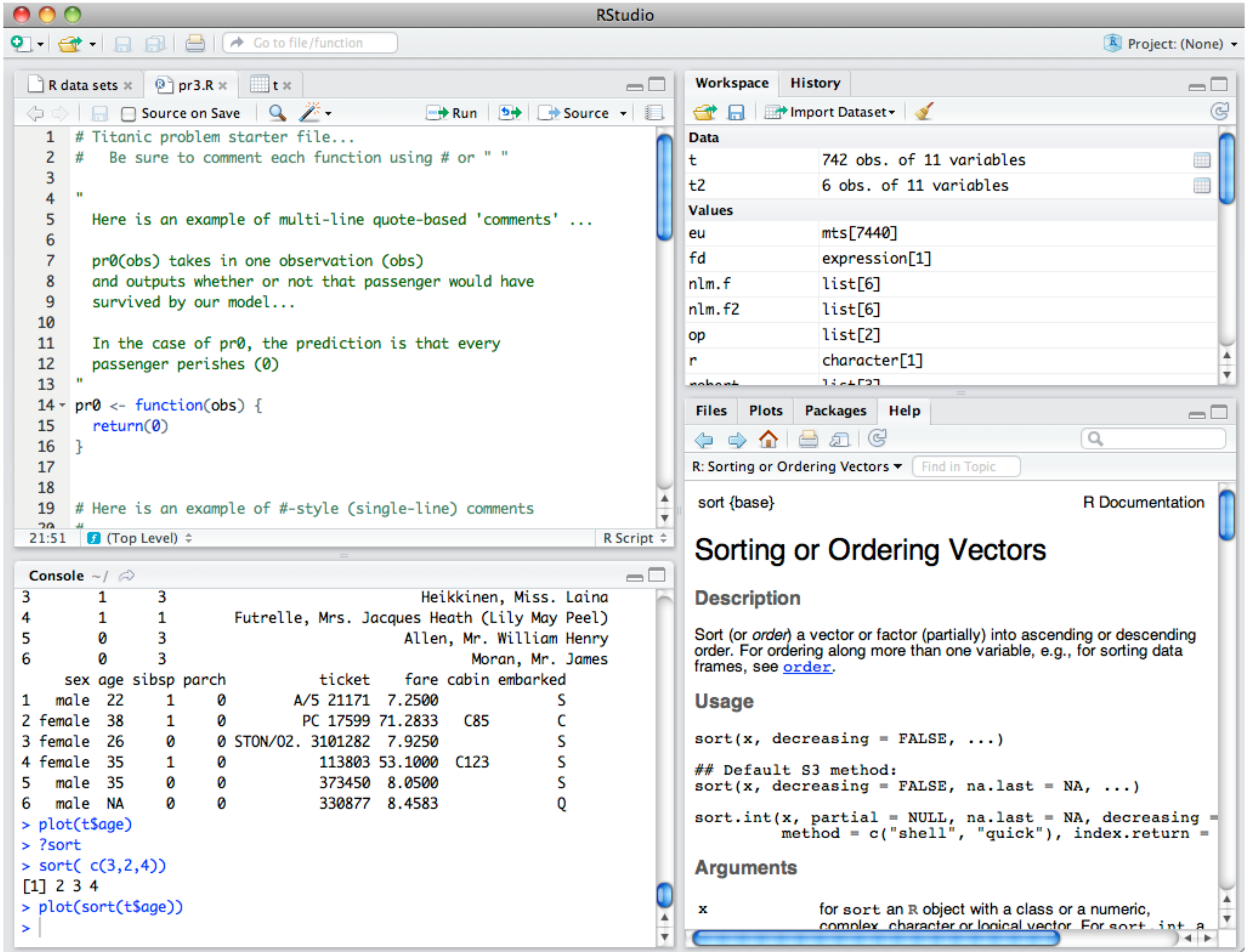
```
R : Copyright 2003, The R Development Core Team
Version 1.7.0  (2003-04-16)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type `license()' or `licence()' for distribution details.

R is a collaborative project with many contributors.
Type `contributors()' for more information.

Type `demo()' for some demos, `help()' for on-line help, or
`help.start()' for a HTML browser interface to help.
Type `q()' to quit R.

> library("MASS")
> data()
> data(Cars93)
> Cars93
   Manufacturer          Model    Type Min.Price Price Max.Price MPG.city
1         Acura        Integra   Small      12.9  15.9      18.8       25
2         Acura         Legend Midsize      29.2  33.9      38.7       18
3          Audi             90 Compact      25.9  29.1      32.3       20
4          Audi            100 Midsize      30.8  37.7      44.6       19
5           BMW           535i Midsize      23.7  30.0      36.2       22
6         Buick        Century Midsize      14.2  15.7      17.3       22
7         Buick        LeSabre   Large      19.9  20.8      21.7       19
8         Buick     Roadmaster   Large      22.6  23.7      24.9       16
9         Buick        Riviera Midsize      26.3  26.3      26.3       19
10     Cadillac        DeVille   Large      33.0  34.7      36.3       16
11     Cadillac        Seville Midsize      37.5  40.1      42.7       16
12    Chevrolet       Cavalier Compact       8.5  13.4      18.3       25
13    Chevrolet        Corsica Compact      11.4  11.4      11.4       25
14    Chevrolet         Camaro  Sporty      13.4  15.1      16.8       19
15    Chevrolet         Lumina Midsize      13.4  15.9      18.4       21
16    Chevrolet     Lumina_APV     Van      14.7  16.3      18.0       18
17    Chevrolet          Astro     Van      14.7  16.6      18.6       15
18    Chevrolet        Caprice   Large      18.0  18.8      19.6       17
```

R 1.7.0 - A Language and Environment

Start | StatisticsWithRPartI.ppt | RGui - [R Console] | vumc | 11:11 PM

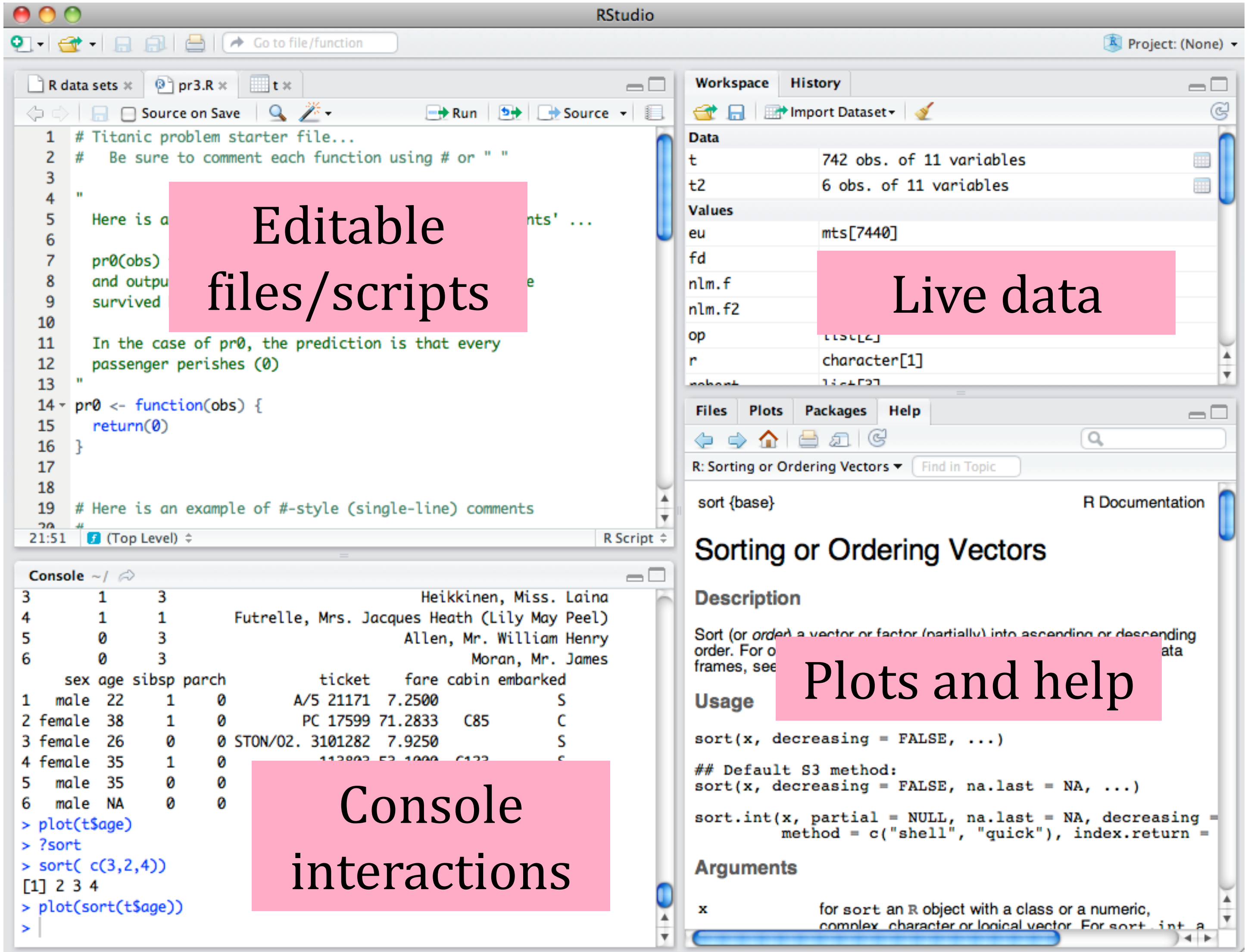
RStudio

An **IDE** that wraps R



RStudio

An **IDE** that wraps R



Getting help… and quitting

- Getting information about a specific command

```
> help(rnorm)
> ?rnorm
```

- Finding functions related to a keyword

```
> help.search("boxplot")
```

- Starting the R installation help pages

```
> help.start()
```

- Quitting R

```
> q()
```

Basic data types

Objects

- variables = objects
- types of objects: **`vector`**, `factor`, `array`, `matrix`, `data.frame`, `ts`, `list`
- attributes
  - mode: integer, numeric, character, complex, logical
  - length: number of elements in object
- creation
  - assign a value
  - create a blank object

Naming Convention

- must start with a letter (A-Z or a-z)

- can contain letters, digits (0-9), and/or
  - periods "."
  - underscore "_"

- case-sensitive
  - `mydata` different from `MyData`

Assignment

- “<-” used to indicate assignment

```
x<-c(1,2,3,4,5,6,7)
x<-c(1:7)
x<-1:7
```

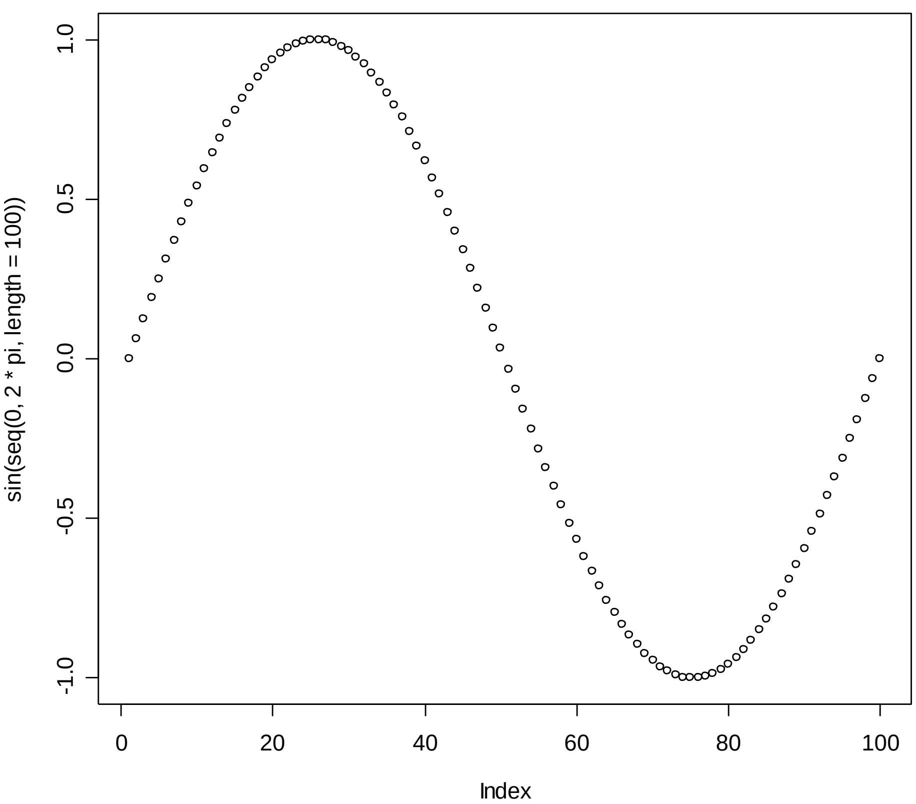
- *note: as of version 1.4 “=“ is also a valid assignment operator*

R as a calculator

```
> 5 + (6 + 7) * pi^2
[1] 133.3049
> log(exp(1))
[1] 1
> log(1000, 10)
[1] 3
> sin(pi/3)^2 + cos(pi/3)^2
[1] 1
> Sin(pi/3)^2 + cos(pi/3)^2
Error: couldn't find function "Sin"
```

R as a calculator

```
> log2(32)
[1] 5

> sqrt(2)
[1] 1.414214

> seq(0, 5, length=6)
[1] 0 1 2 3 4 5
```




```
> plot(sin(seq(0, 2*pi, length=100)))
```

Basic (atomic) data types

- Logical

```
> x <- T; y <- F
> x; y
[1] TRUE
[1] FALSE
```

- Numerical

```
> a <- 5; b <- sqrt(2)
> a; b
[1] 5
[1] 1.414214
```

- Character

```
> a <- "1"; b <- 1
> a; b
[1] "1"
[1] 1
> a <- "character"
> b <- "a"; c <- a
> a; b; c
[1] "character"
[1] "a"
[1] "character"
```

Data Type Conversion

- Type conversions in R work as you would expect. For example, adding a character string to a numeric vector converts all the elements in the vector to character.
- Use is.*foo* to test for data type *foo*. Returns TRUE or FALSE
  Use as.*foo* to explicitly convert it.
- is.numeric(), is.character(), is.vector(), is.matrix(), is.data.frame()
  as.numeric(), as.character(), as.vector(), as.matrix(), as.data.frame)

Vectors, Matrices, Arrays

- Vector
  - Ordered collection of data of the same data type
  - Example:
    - last names of all students in this class
    - Mean intensities of all genes on an oligonucleotide microarray
  - In R, single number is a vector of length 1
- Matrix
  - Rectangular table of data of the same type
  - Example
    - Intensities of all genes measured during a microarray experiment
- Array
  - Higher dimensional matrix

Vectors

- Vector: Ordered collection of data of the same data type

```
> x <- c(5.2, 1.7, 6.3)
> log(x)
[1] 1.6486586 0.5306283 1.8405496
> y <- 1:5
> z <- seq(1, 1.4, by = 0.1)
> y + z
[1] 2.0 3.1 4.2 5.3 6.4
> length(y)
[1] 5
> mean(y + z)
[1] 4.2
```

Vectors

```
> Mydata <- c(2,3.5,-0.2)
> Colors <-
    c("Red","Green","Red")
```

Vector (c="concatenate")

Character vector

```
> x1 <- 25:30
> x1
[1] 25 26 27 28 29 30
```

Number sequences

```
> Colors[2]
[1] "Green"
```

One element (1-index!)

```
> x1[3:5]
[1] 27 28 29
```

Various elements

Operation on vector elements

```
> Mydata
[1] 2 3.5 -0.2
```

```
> Mydata > 0
[1] TRUE TRUE FALSE
```

- Test on the elements

```
> Mydata[Mydata>0]
[1] 2 3.5
```

- Extract the positive elements

```
> Mydata[-c(1,3)]
[1] 3.5
```

- Remove elements

Vector operations

```
> x <- c(5,-2,3,-7)
> y <- c(1,2,3,4)*10
> y
[1] 10 20 30 40
```

Operation on all the elements

```
> sort(x)
[1] -7 -2 3 5
```

Sorting a vector

```
> order(x)
[1] 4 2 3 1
```

Element order for sorting

```
> y[order(x)]
[1] 40 20 30 10
```

Operation on all the components

```
> rev(x)
[1] -7 3 -2 5
```

Reverse a vector

Matrices

- Matrix: Rectangular table of data of the same type

```
> m <- matrix(1:12, 4, byrow = T); m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> y <- -1:2
> m.new <- m + y
> t(m.new)
     [,1] [,2] [,3] [,4]
[1,]    0    4    8   12
[2,]    1    5    9   13
[3,]    2    6   10   14
> dim(m)
[1] 4 3
> dim(t(m.new))
[1] 3 4
```

Matrices

Matrix: Rectangular table of data of the same type

```
> x <- c(3,-1,2,0,-3,6)
> x.mat <- matrix(x,ncol=2)        Matrix with 2 cols
> x.mat
     [,1] [,2]
[1,]    3    0
[2,]   -1   -3
[3,]    2    6
```

```
> x.mat <- matrix(x,ncol=2,
               byrow=T)
> x.mat
     [,1] [,2]
[1,]    3   -1
[2,]    2    0
[3,]   -3    6
```

By row creation

Dealing with matrices

```
> x.mat[,2]
[1] -1 0 6
```

2nd col

```
> x.mat[c(1,3),]

     [,1] [,2]
[1,]    3   -1
[2,]   -3    6
```

1st and 3rd lines

```
> x.mat[-2,]

     [,1] [,2]
[1,]    3   -1
[2,]   -3    6
```

No 2nd line

Dealing with matrices

```
> dim(x.mat)
[1] 3 2
```

Dimension

```
> t(x.mat)
     [,1] [,2] [,3]
[1,]    3    2   -3
[2,]   -1    0    6
```

Transpose

```
> x.mat %*% t(x.mat)

     [,1] [,2] [,3]
[1,]   10    6  -15
[2,]    6    4   -6
[3,]  -15   -6   45
```

Multiplication

`> solve()` solves the equation A %*% X = B for X,

`> eigen()` Eigenvectors and eigenvalues

Missing values

- R is designed to handle statistical data and therefore predestined to deal with missing values
- Numbers that are "not available"

```
> x <- c(1, 2, 3, NA)
> x + 3
[1] 4  5  6 NA
```

- Testing for Missing Values

```
> is.na(x) # returns TRUE if x is missing
> y <- c(1,2,3,NA)
> is.na(y) # returns a vector (F F F T)
```

- "Not a number"

```
> log(c(0, 1, 2))
[1]      -Inf 0.0000000 0.6931472
> 0/0
[1] NaN
```

Missing values

- Excluding Missing Values from Analyses
  - Arithmetic functions on missing values yield missing values.

```
> x <- c(1,2,NA,3)
> mean(x)            # returns NA
> mean(x, na.rm=TRUE) # returns 2
```

  - The function complete.cases() returns a logical vector indicating which cases are complete.

```
# list rows of data that have missing values
> mydata[!complete.cases(mydata),]
```

  - The function na.omit() returns the object with listwise deletion of missing values.

```
# create new dataset without missing data
> newdata <- na.omit(mydata)
```

Subsetting

- It is often necessary to extract a subset of a vector or matrix
- R offers a couple of neat ways to do that

```
> x <- c("a", "b", "c", "d", "e", "f",
"g", "h")
> x[1]
> x[3:5]
> x[-(3:5)]
> x[c(T, F, T, F, T, F, T, F)]
> x[x <= "d"]
> m[,2]
> m[3,]
```

Lists, data frames, and factors

Lists

vector: an ordered collection of data of the same type.

```
> a = c(7,5,1)
> a[2]
[1] 5
```

list: an ordered collection of data of arbitrary types.

```
> doe = list(name="john",age=28,married=F)
> doe$name
[1] "john"
> doe$age
[1] 28
```

Lists 1

- A list is an object consisting of objects called components.
- The components of a list don't need to be of the same mode or type and they can be a numeric vector, a logical value and a function and so on.
- A component of a list can be referred as `aa[[i]]` or `aa$times`, where `aa` is the name of the list and times is a name of a component of `aa`.

Lists 2

- The names of components may be abbreviated down to the minimum number of letters needed to identify them uniquely.
- `aa[[1]]` is the first component of `aa`, while `aa[1]` is the sublist consisting of the first component of `aa` only.
- There are functions whose return value is a List.

Lists are very flexible

```
> my.list <- list(c(5,4,-1),c("X1","X2","X3"))
> my.list
[[1]]:
[1]  5  4 -1

[[2]]:
[1] "X1" "X2" "X3"

> my.list[[1]]
[1]  5  4 -1

> my.list <- list(c1=c(5,4,-1),c2=c("X1","X2","X3"))
> my.list$c2[2:3]
[1] "X2" "X3"
```

Lists 3

```
Empl <- list(employee="Anna", spouse="Fred",
  children=3, child.ages=c(4,7,9))

Empl[[4]]

Empl$child.a

Empl[4]    # a sublist consisting of the 4th component of Empl

names(Empl) <- letters[1:4]

Empl <- c(Empl, service=8)

unlist(Empl)   # converts it to a vector. Mixed types will
  be converted    to character, giving a character vector.
```

More lists

```
> x.mat
     [,1] [,2]
[1,]    3   -1
[2,]    2    0
[3,]   -3    6

> dimnames(x.mat) <- list(c("L1","L2","L3"),
                          c("R1","R2"))

> x.mat
   R1 R2
L1  3 -1
L2  2  0
L3 -3  6
```

Data frames

data frame: represents a spreadsheet.

Rectangular table with rows and columns; data within each column has the same type (e.g. number, text, logical), but different columns may have different types.

Example:

```
> cw = chickwts
> cw
    weight        feed
1     179         horsebean
11    309         linseed
23    243         soybean
37    423         sunflower
...
```

Data frames

Creating a data frame

```
> d <- c(1,2,3,4)
> e <- c("red", "white", "red", NA)
> f <- c(TRUE,TRUE,TRUE,FALSE)
> mydata <- data.frame(d,e,f)
> names(mydata) <- c("ID","Color","Passed")
```

Adding a new column

```
> mydata$Height <- c(100,120,120,130)
> mydata$Shape <- "circle"
```

Subsetting

Individual elements of a vector, matrix, array or data frame are accessed with "[ ]" by specifying their index, or their name

```
> cw = chickwts
> cw
    weight      feed
1     179       horsebean
11    309       linseed
23    243       soybean
37    423       sunflower
...

> cw [3,2]
[1] horsebean
6 Levels: casein horsebean linseed ... sunflower
> cw [3,]
  weight      feed
3    136 horsebean
```

Subsetting

Other ways to subset…

columns 3,4,5 of dataframe

```
> myframe[3:5]
```

columns ID and Age from dataframe

```
> myframe[c("ID","Age")]
```

variable ID in the dataframe

```
> myframe$ID
```

using subset function

```
> subset( myframe, Age < 35, c("ID","Age") )
```

Merging

To merge two dataframes (datasets) horizontally, use the **merge** function. In most cases, you join two dataframes by one or more common key variables (i.e., an inner join).

```
# merge two dataframes by ID
total <- merge(dataframeA,dataframeB,by="ID")

# merge two dataframes by ID and Country
total <- merge(dataframeA,
               dataframeB,
               by=c("ID","Country"))
```

Merging

ADDING ROWS

To join two dataframes (datasets) vertically, use the **rbind** function. The two dataframes **must** have the same variables, but they do not have to be in the same order.

```
total <- rbind(dataframeA, dataframeB)
```

If dataframeA has variables that dataframeB does not, then either:

- Delete the extra variables in dataframeA or
- Create the additional variables in dataframeB and set them to NA (missing) before joining them with rbind.

Aggregating

- **It is relatively easy to collapse data in R using one or more BY variables and a defined function.**
- # aggregate dataframe mtcars by cyl, returning means for numeric variables

```
> attach(mtcars)
> aggdata <- aggregate( mtcars,
                         by=list(cyl),
                         FUN=mean,
                         na.rm=TRUE )
> print(aggdata)
```

Factors

Tell **R** that a variable is **nominal** by making it a factor. The factor stores the nominal values as a vector of integers in the range [1... k] (where k is the number of unique values in the nominal variable), and an internal vector of character strings (the original values) mapped to these integers.

variable gender with 20 "male" entries and 30 "female" entries

```
gender <- c(rep("male",20), rep("female", 30))
gender <- factor(gender)
```

stores gender as 20 1s and 30 2s and associates

1=female, 2=male internally (alphabetically)

R now treats gender as a nominal variable

```
summary(gender)
```

Control structures

Control Structures

Control structures in R allow you to control the flow of execution of the program, depending on runtime conditions. Common structures are

- `if`, `else`: testing a condition
- `for`: execute a loop a fixed number of times
- `while`: execute a loop *while* a condition is true
- `repeat`: execute an infinite loop
- `break`: break the execution of a loop
- `next`: skip an interation of a loop
- `return`: exit a function

Most control structures are not used in interactive sessions, but rather when writing functions or longer expresisons.

Slide credits: *thanks to* JHU's R. D. Peng

Control Structures: if

```
if(<condition>) {
        ## do something
} else {
        ## do something else
}
```

```
if(<condition1>) {
        ## do something
} else if(<condition2>)  {
        ## do something different
} else {
        ## do something different
}
```

if

This is a valid if/else structure.

```
if(x > 3) {
        y <- 10
} else {
        y <- 0
}
```

So is this one.

```
y <- if(x > 3) {
        10
} else {
        0
}
```

```
if(x > 1) {
        print("x is big")
} else if(x > 0) {
        print("x is positive")
} else {
        print("x is negative or zero")
}
```

How are these two conditionals different?

```
if(x > 1) {
        print("x is big")
}
if(x > 0) {
        print("x is positive")
}
print("x is negative or zero")
```

for

`for` loops take an interator variable and assign it successive values from a sequence or vector. For loops are most commonly used for iterating over the elements of an object (list, vector, etc.)

```
for(i in 1:10) {
        print(i)
}
```

This loop takes the `i` variable and in each iteration of the loop gives it values 1, 2, 3, ..., 10, and then exits.

for

These three loops have the same behavior.

```
x <- c("a", "b", "c", "d")

for(i in 1:4) {
        print(x[i])
}

for(i in seq_along(x)) {
        print(x[i])
}

for(letter in x) {
        print(letter)
}

for(i in 1:4) print(x[i])
```

`seq_along` creates a list of indices

while

While loops begin by testing a condition. If it is true, then they execute the loop body. Once the loop body is executed, the condition is tested again, and so forth.

```
count <- 0

while(count < 10) {
        print(count)
        count <- count + 1
}
```

While loops can potentially result in infinite loops if not written properly. Use with care!

while

Sometimes there will be more than one condition in the test.

```
z <- 5

while(z >= 3 && z <= 10) {
        print(z)
        coin <- rbinom(1, 1, 0.5)

        if(coin == 1) {  ## random walk
                z <- z + 1
        } else {
                z <- z - 1
        }
}
```

Conditions are always evaluated from left to right.

Repeat initiates an infinite loop; these are not commonly used in statistical applications but they do have their uses. The only way to exit a `repeat` loop is to call `break`.

```
x0 <- 1
tol <- 1e-8

repeat {
        x1 <- computeEstimate()

        if(abs(x1 - x0) < tol) {
                break
        } else {
                x0 <- x1
        }
}
```

next, return

`next` is used to skip an iteration of a loop

```
for(i in 1:100) {
        if(i <= 20) {
                ## Skip the first 20 iterations
                next
        }
        ## Do something here
}
```

`return` signals that a function should exit and return a given value

Arithmetic Operators

Operator	**Description**
+	addition
-	subtraction
*****	multiplication
/	division
^ **or ****	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Arithmetic Operators

Functions: **`abs(), sign(), log(), log10(), sqrt(),`**
`exp(), sin(), cos(), tan()`
`gamma(), lgamma(), choose()`

Rounding: **`round(x,3)`**

Rounding: **`floor(2.5)`** => 2, **`ceiling(2.5)`** => 3

Vector functions

```
> vec <- c(5,4,6,11,14,19)
> sum(vec)
[1] 59
> prod(vec)
[1] 351120
> mean(vec)
[1] 9.833333
> median(vec)
[1] 8.5
> var(vec)
[1] 34.96667
> sd(vec)
[1] 5.913262
> summary(vec)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  4.000   5.250   8.500   9.833  13.250  19.000 
```

And also: `min()` `max()`
`cummin()` `cummax()`
`cumsum()` `cumprod()`
`range()`

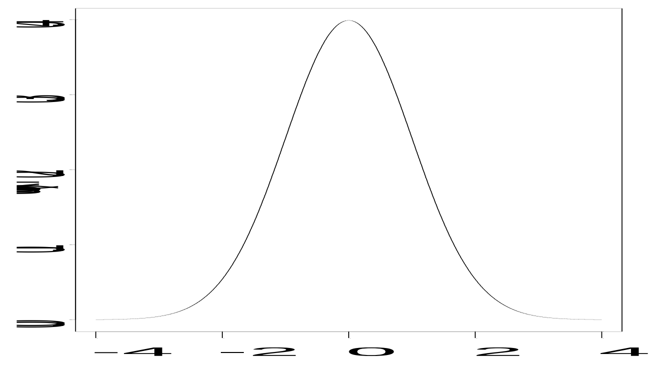
Logical Operators

Operator	**Description**
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if x is TRUE

Statistical functions

Normal distr

$$f(x\,|\,\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



```
> dnorm(2,mean=1,sd=2)
[1] 0.1760327
```

PDF in point 2
for X ~ N(1,4)

```
> qnorm(0.975)
[1] 1.959964
```

Quantile for
the 0.975 for N~ (0,1)

```
> pnorm(c(2,3),mean=2)
[1] 0.5000000 0.8413447
```

= P(X<2) and P(X<3), where X ~ N(2,1)

```
> norm.alea <- rnorm(1000)
```

Pseudo-random normally distributed numbers

```
> summary(norm.alea)
 Min. 1st Qu.  Median     Mean 3rd Qu.  Max.
 -3.418 -0.6625 -0.0429 -0.01797  0.6377 3.153
> sd(norm.alea)
[1] 0.9881418
```

How to remember functions

For a normal distribution, the root is **`norm`**. Then add the letters

`d`	density (**`dnorm()`**)
`p`	probability(**`pnorm()`**)
`q`	quantiles (**`qnorm()`**)
`r`	pseudo-random (**`rnorm()`**)

Distribution	Root	Argument
normal	**`norm`**	**`mean, sd, log`**
t (Student)	**`t`**	**`df, log`**
uniform	**`unif`**	**`min, max, log`**
F (Fisher)	**`f`**	**`df1, df2`**
χ^2	**`chisq`**	**`df, ncp, log`**
Binomial	**`binom`**	**`size, prob, log`**
exponential	**`exp`**	**`rate, log`**
Poisson	**`pois`**	**`lambda, log`**
...		

Function	**Description**

dnorm(_x_**)**	normal density function (by default m=0 sd=1)
	# plot standard normal curve
	x <- pretty(c(-3,3), 30)
	y <- dnorm(x)
	plot(x, y, type='l', xlab="Normal Deviate", ylab="Density", yaxs="i")

pnorm(_q_**)**	cumulative normal probability for q (area under the normal curve to the right of q) pnorm(1.96) is 0.975
qnorm(_p_**)**	normal quantile. value at the p percentile of normal distribution qnorm(.9) is 1.28 # 90th percentile
rnorm(_n_**, m=**0**,sd=**1**)**	n random normal deviates with mean m and standard deviation sd. #50 random normal variates with mean=50, sd=10 x <- rnorm(50, m=50, sd=10)

dbinom(_x_**,** _size_**,** _prob_**)**
pbinom(_q_**,** _size_**,** _prob_**)**
qbinom(_p_**,** _size_**,** _prob_**)**
rbinom(_n_**,** _size_**,** _prob_**)**
	binomial distribution where size is the sample size
	and prob is the probability of a heads (pi)
	# prob of 0 to 5 heads of fair coin out of 10 flips
	dbinom(0:5, 10, .5)
	# prob of 5 or less heads of fair coin out of 10 flips
	pbinom(5, 10, .5)

dpois(_x_**,** _lamda_**)**
ppois(_q_**,** _lamda_**)**
qpois(_p_**,** _lamda_**)**
rpois(_n_**,** _lamda_**)**
	poisson distribution with m=std=lamda
	#probability of 0,1, or 2 events with lamda=4
	dpois(0:2, 4)
	# probability of at least 3 events with lamda=4
	1- ppois(2,4)

dunif(_x_**, min=**0**, max=**1**)** **punif(**_q_**, min=0, max=**1**)** **qunif(**_p_**, min=**0**, max=**1**)** **runif(**_n_**, min=**0**, max=**1**)**	uniform distribution, follows the same pattern as the normal distribution above. #10 uniform random variates x <- runif(10)

Importing/ Exporting Data

Importing/Exporting Data

- Importing data
  - R can import data from other applications
  - Packages are available to import microarray data, Excel spreadsheets etc.
  - The easiest way is to import tab delimited files

```
> SimpleData <- read.table(
file = "http://eh3.uc.edu/SimpleData.txt",
header = TRUE,
quote = "",
sep = "\t",
comment.char="")
```

- Exporting data
  - R can also export data in various formats
  - Tab delimited is the most common

```
> write.table(x, "filename") *)
```

*) make sure to include the path or to first change the working directory

Credits

- Roger D. Peng
- Gilberto Câmara (R – a brief introduction)
- Ralitza Gueorguieva (R Basics)