Introduction to R software and applications

Marc J. Mazerolle

cef Centre d'étude de la forêt

cfr Centre for Forest Research

Université du Québec en Abitibi-Témiscamingue

What is the R project?

R is a programming and statistical language

- developed in early 1990's by 2 statisticians and programmers from New Zealand (Ihaka and Gentleman)
- dialect of S language (which is proprietary, now integrated into S-PLUS)
- R comes from the need for flexibility in running analyses and high quality graphics without having to defray astronomical costs (i.e., available to anyone)

What is the R project?

Initiative of statisticians and programmers rendering the software and source code freely available.

What this means:

- the source code is open to all and can be modified according to one's needs (for advanced users...)
- new functions can be created by users and made available to all
- updates and upgrades are frequent and free (new version ~ every 6 months)
- no annual licence (site, individual) to be paid to a company that imposes its standards and choices (e.g., SAS)
- a very large community of users worldwide

R uses

Programming language:
related to S and C

Applied mathematics (large calculator):
matrix algebra, derivatives, integrals

Applications in statistics and optimization:
wide array of possibilities

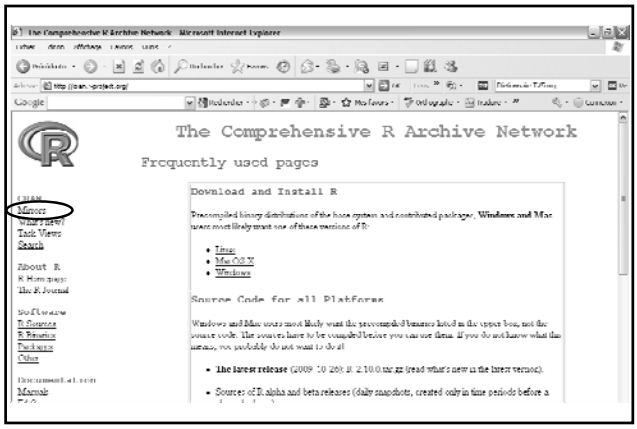
Excellent graphical capability:
wide array of graphics

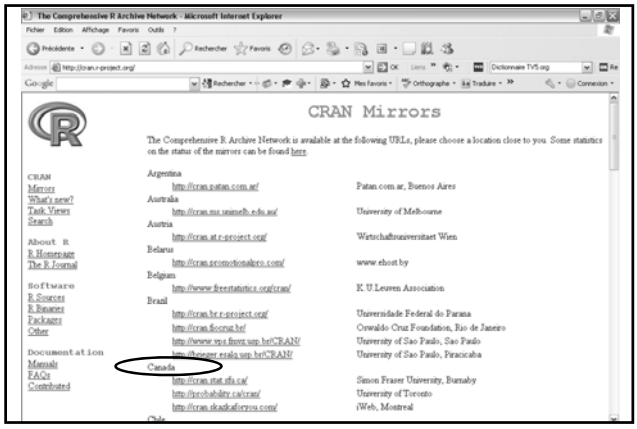
Flexibility:
easy to create new functions,
use loops

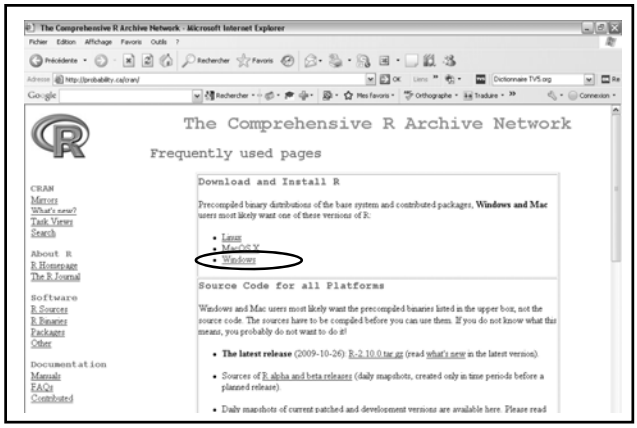
Where to find R?

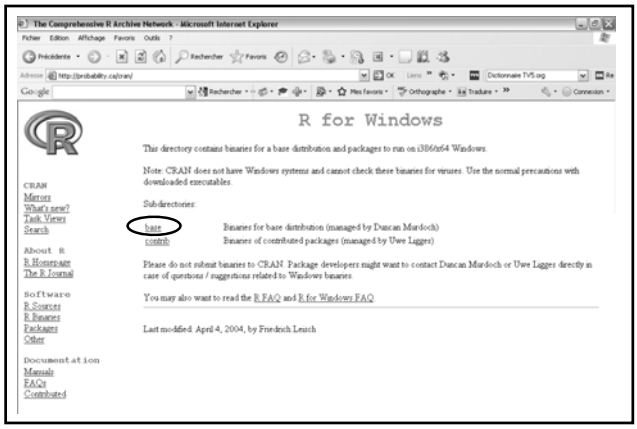
http://cran.r-project.org/

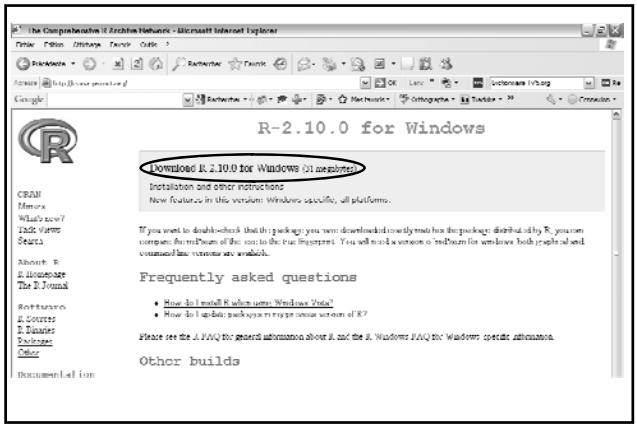
and its numerous mirror sites worldwide




The Comprehensive R Archive Network - Microsoft Internet Explorer
CRAN Mirrors
The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here.
Argentina
http://cran.patan.com.ar/ Patan.com.ar, Buenos Aires
Australia
http://cran.ms.unimelb.edu.au/ University of Melbourne
Austria
http://cran.at.r-project.org/ Wirtschaftsuniversitaet Wien
Belarus
http://cran.promotionalpro.com/ www.ehost.by
Belgium
http://www.freestatistics.org/cran/ K.U.Leuven Association
Brazil
http://cran.br.r-project.org/ Universidade Federal do Parana
http://cran.fiocruz.br/ Oswaldo Cruz Foundation, Rio de Janeiro
http://www.vps.fmvz.usp.br/CRAN/ University of Sao Paulo, Sao Paulo
http://pingger.esalq.usp.br/CRAN/ University of Sao Paulo, Piracicaba
Canada
http://cran.stat.sfu.ca/ Simon Fraser University, Burnaby
http://probability.ca/cran/ University of Toronto
http://cran.skadkaforyou.com/ iWeb, Montreal
CRAN
Mirrors
What's new?
Task Views
Search
About R
R Homepage
The R Journal
Software
R Sources
R Binaries
Packages
Other
Documentation
Manuals
FAQs
Contributed


The Comprehensive R Archive Network - Microsoft Internet Explorer
The Comprehensive R Archive Network
Frequently used pages
Download and Install R
Precompiled binary distributions of the base system and contributed packages, Windows and Mac users most likely want one of these versions of R:
Linux
MacOS X
Windows
Source Code for all Platforms
Windows and Mac users most likely want the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!
The latest release (2009-10-26): R-2.10.0.tar.gz (read what's new in the latest version).
Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
Daily snapshots of current patched and development versions are available here. Please read
CRAN
Mirrors
What's new?
Task Views
Search
About R
R Homepage
The R Journal
Software
R Sources
R Binaries
Packages
Other
Documentation
Manuals
FAQs
Contributed


The Comprehensive R Archive Network - Microsoft Internet Explorer
R for Windows
This directory contains binaries for a base distribution and packages to run on i386/x64 Windows.
Note: CRAN does not have Windows systems and cannot check these binaries for viruses. Use the normal precautions with downloaded executables.
Subdirectories:
base Binaries for base distribution (managed by Duncan Murdoch)
contrib Binaries of contributed packages (managed by Uwe Ligges)
Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch or Uwe Ligges directly in case of questions / suggestions related to Windows binaries.
You may also want to read the R FAQ and R for Windows FAQ.
Last modified: April 4, 2004, by Friedrich Leisch
CRAN
Mirrors
What's new?
Task Views
Search
About R
R Homepage
The R Journal
Software
R Sources
R Binaries
Packages
Other
Documentation
Manuals
FAQs
Contributed

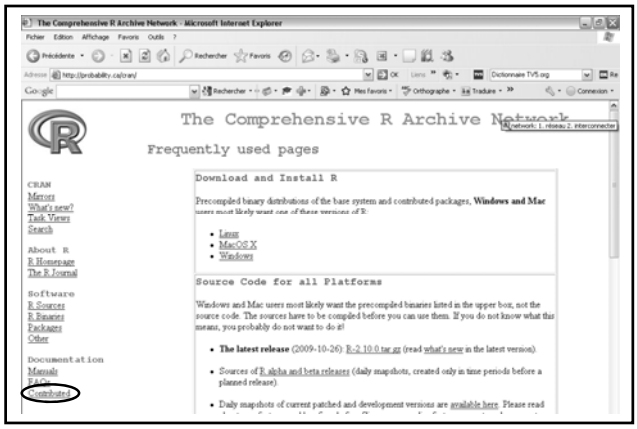


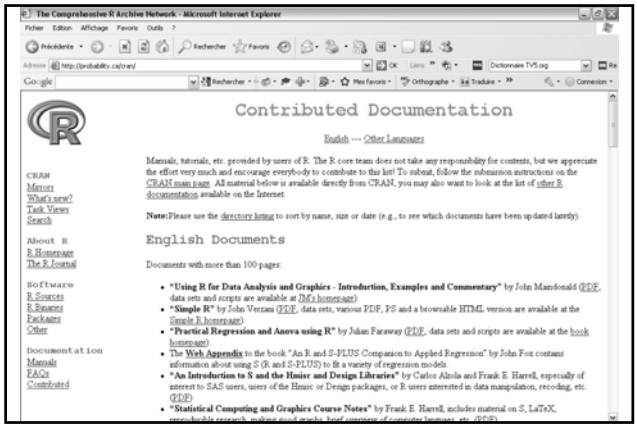
Installation

Run installation file:
`R-2.10.1-win32.exe` (with default options)

The distribution includes user manuals
(a little arid for beginners...)

Go to `Documentation` and `Contributed` on the R page and download other documents

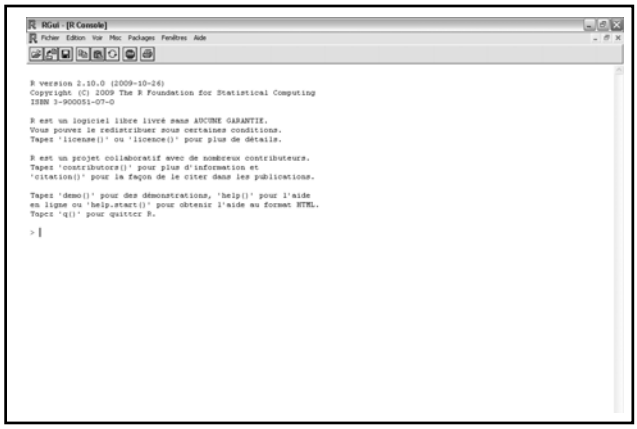




Start me up

Minimal graphical user interface (GUI) for R in Windows in which to use mouse clicks...

(none for Mac and Linux)



Start me up

No fancy GUI for R in Windows

Must communicate by typing commands.

To ease this, one can prepare the code in a text editor (`Word`, `Notepad`, `Emacs`...)

Depends on the operating system (OS)

Editors

Some run on several platforms

`JGR`	R syntax highlighting and interacts with R
`emacs` (with `ess`)	R syntax highlighting and interacts with R
`jEDIT`	R syntax highlighting

Others are platform-dependent:

Mac:

`TextWrangler`	R syntax highlighting
`Aquamacs`	R syntax highlighting and interacts with R

Linux:

`Kate`	R syntax highlighting and interacts with R

Editors

Windows:

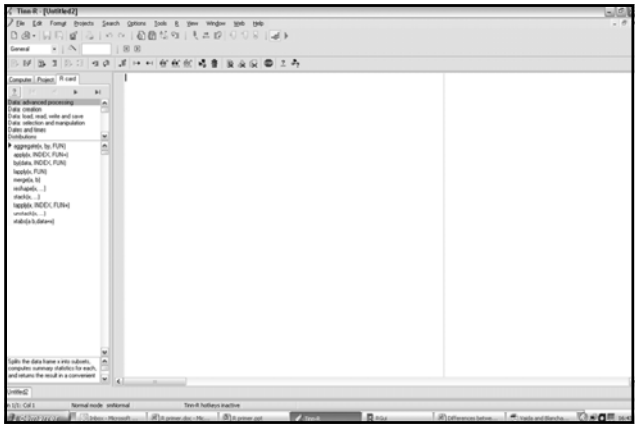
`WinEdt`	R syntax highlighting and interacts with R (shareware)
`Tinn-R`	R syntax highlighting and interacts with R

Suggested editor: `Tinn-R` (version `1.17.2.4`)

not the most recent version, but one that works without any modifications of R settings

recent versions require playing in R preferences

add "--sdi" in target line of shortcut



Tinn-R as an editor

One can write the code in Tinn-R and send the code to R via icons « send selection », « send line » and variations thereof

important: both R and Tinn-R must be open

Possible to assign hot keys and customize Tinn-R to our liking (show line numbers, remove toolbars, etc...)

Tip

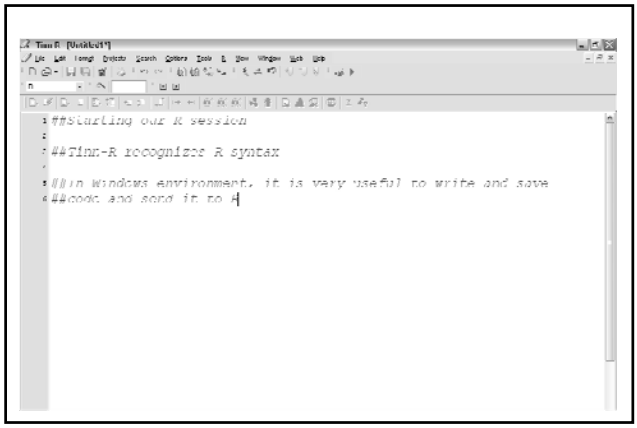
Use an editor to write and save your code

Save the file with the extension .R or .r

once file is opened with a smart editor, it will know your are working in R

The # symbol is reserved for comments
(when encountered, R skips to next line)

Do not be stingy on adding comments in your code!!!



Another tip

All commands in R follow the same strategy:

a name followed by parentheses

e.g., `mean( )` function to compute arithmetic mean

options are specified between parentheses

e.g., `mean(DBH)` compute arithmetic mean of variable DBH

Getting help in R

`help.start( )`

Generates an `html` page from which we can navigate through different themes with hyperlinks

`help.search( )`

Search for specific topics in R help files

the `apropos` argument can be used to specify the search topic

`help.search(apropos = "generalized linear models")`

returns all R functions that deal with the topic

Getting help in R

? Followed by the name of a function

To get help on functions (syntax, arguments, usage)

e.g., `?mean`

Details followed by examples of using the function (most informative)

`RSiteSearch(" ")`

To search R archives on the internet

Opens a window from your web browser directly to site

Object-oriented language

An object can take the form of a numeric variable (vector), a data set, a matrix, a function, a list, output.

Simply assign a name to an object to create it and it remains accessible during the R session.

Vectors and basic operations

Creating a variable (scalar)

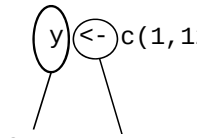
Object — Operator indicates that the value of 15.3 is assigned to object « y ».

the name of an object must begin by a letter and can be a combination of numbers or letters (good practice to keep them short and informative)

the name can contain « _ » or « . » but space is not allowed

Ex. pH_soil, pH.soil NOT pH soil

Creating a variable (vector)



Object — Operator indicates that the values are assigned to object « y ».

c( ) indicates that values should be combined in the vector

Creating a variable (vector)

Beware of upper and lower case

you must use the exact name of the object (match cases)

```
> Y
Error : object "Y" not found

> y
 [1]  1 12 31 14 50 22 13 14 25 26
```

Select values

`y[2]`

selects 2nd value of y

`y[1:5]`

selects first 5 values of y

Creating a vector

To repeat a value

`rep( )`

```
> new_y<-rep(x=5,times=20)
> new_y
 [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

Works with either numeric or string (character) values

```
> new_z<-rep(x=c("new", "old"), times=3)
> new_z
[1] "new" "old" "new" "old" "new" "old"
```

Creating a vector

To create a sequence (series) with a given step

`seq( )`

```
> y2<-seq(from=1, to=10, by=1)
> y2
 [1]  1  2  3  4  5  6  7  8  9 10
```

we can also generate declining sequences

```
y3<-seq(from=10, to=1, by=-2)
> y3
[1] 10  8  6  4  2
```

Checking vector characteristics

Use `class( )` or logical tests to determine characteristics

```
> class(y3)
[1] "numeric"
> is.numeric(y3)
[1] TRUE


> new_z
[1] "new" "old" "new" "old" "new" "old"
> class(new_z)
[1] "character"
> is.numeric(new_z)
[1] FALSE
> is.character(new_z)
[1] TRUE
```

Compute vector length

`length( )` object length

```
> y
 [1]  1 12 31 14 50 22 13 14 25 26

> length(y)
[1] 10
```

Mathematical operators

Mathematical operators

Basic operators:

```
+     -     *     /
> 30+10-2*3
34
```

respects conventions regarding priority of * and / before + or -

```
> y2
 [1]  1  2  3  4  5  6  7  8  9 10
> y
 [1]  1 12 31 14 50 22 13 14 25 26
> y+y2
 [1]  2 14 34 18 55 28 20 22 34 36

> 2-y2
 [1]  1  0 -1 -2 -3 -4 -5 -6 -7 -8
```

the value 2 was recycled for this computation

Mathematical operators

Other basic functions:

`sum( )` sum of all elements

```
> sum(y)
[1] 208
```

`prod( )` product of all elements

```
> prod(y)
[1] 6.77717e+11
```

`^( )` exponentiate - compute value at a given power

```
> 2^3
[1] 8
```

`round( )` round to *x* digits after decimal

```
> round(123.3453, digits=2)
[1] 123.35
```

```
> round(123.3453, digits=0)
[1] 123
```

Mathematical operators

Other basic functions:

`sqrt( )` square root

`exp( )` compute exponential function as y = (e^x)

`log( )` log base *e* (natural or Naperian log)

`log2( )` log base 2

`log10( )` log base 10

`abs( )` absolute value

Mathematical operators

Other basic functions:

`min( )` returns the minimum value

`max( )` returns the maximum value

`range( )` returns the minimum and maximum values

```
> range(y)
[1]  1 50
```

Trigonometric functions:

`sin( ), cos( ), asin( ), tan( ), etc...`

Mathematical operators

Symbolic differential calculus `D( )`

```
> D(expression(x^2),"x")
2 * x
```

derivative of x^2 with respect to x

```
> D(expression(x+(x^2)+(2*z^3)), "z")
2 * (3 * z^2)
```

derivative with respect to z

Integral calculus `integrate( )`

```
> integrate(dnorm, -1.96, 1.96)
0.9500042 with absolute error < 1.0e-11
```

the area under the curve for a standard normal distribution $N(0, 1)$ from -1.96 to 1.96

Matrices and matrix calculus

A new object type: the matrix

Creating a matrix with the function `matrix( )`

```
> mat<-matrix(data=c(1:9), nrow=3, ncol=3)
> mat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

By default, elements are entered one column at a time.

```
> mat<-matrix(data=c(1:9), nrow=3, ncol=3, byrow=TRUE)
> mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

`byrow=TRUE` enters elements one row at a time

Checking matrix characteristics

Checking the nature of an object with `class( )` or logical tests

```
> class(mat)
[1] "matrix"
> is.numeric(mat)
[1] TRUE
> is.vector(mat)
[1] FALSE
```

Determining the number of elements in the matrix with `length( )`

```
> length(mat)
[1] 9
```

Determining matrix dimensions with `dim( )`, `ncol( )`, or `nrow( )`

```
> dim(mat)      > ncol(mat)    > nrow(mat)
[1] 3 3         [1] 3          [1] 3
```

Selecting elements within a matrix

With vectors we use `[ ]`, but for matrices, one must specify two dimensions `[ , ]` (row, column).

```
> mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> mat[1,2]            > mat[3,1]
[1] 2                 [1] 7
```

Selecting elements within a matrix

With vectors we use `[ ]`, but for matrices, one must specify two dimensions `[ , ]` (row, column).

```
> mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

`> mat[,1]` select column 1

```
[1] 1 4 7
```

`> mat[2,]` select row 2

```
[1] 4 5 6
```

Selecting elements within a matrix

With vectors we use `[ ]`, but for matrices, one must specify two dimensions `[ , ]` (row, column).

```
> mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

`> mat[1:2,1]` select elements of column 1 of rows 1 and 2

```
[1] 1 4
```

`> mat[c(1,3),3]` select elements of column 3 of rows 1 and 3

```
[1] 3 9
```

Matrix calculus

Diagonal of matrix `diag( )`

```
> mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> diag(mat)
[1] 1 5 9
```

Transpose of matrix `t( )`

```
> t(mat)
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Matrix calculus

Other functions

`colSums( )` compute sum of each column

```
> colSums(mat)
[1] 12 15 18
```

`colMeans( )` compute mean of each column

`rowSums( )` compute sum of each row

`rowMeans( )` compute mean of each row

Matrix calculus

Other functions

`solve( )` inverse of matrix (if it can be inversed)

```
> mat1<-matrix(data=y[1:9], nrow=3, ncol=3)
> mat1_inv<-solve(mat1)
> mat1_inv
             [,1]         [,2]         [,3]
[1,] -0.067769784  0.004604317  0.032661871
[2,] -0.009640288  0.027194245 -0.010215827
[3,]  0.092517986 -0.029640288  0.008489209
```

Matrix calculus

Multiplication involving matrices %*%

```
> vector<-c(2,4,5)
> mat%*%vector
     [,1]
[1,]   25
[2,]   58
[3,]   91
```

```
> round(mat1%*%mat1_inv, digits=4)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

product of original matrix and its inverse yield an identity matrix (diagonal of 1's)

(definition of inverse of matrix)

Data frames

Another object type: the data frame

Creating a data frame with function `data.frame( )`

```
> Time<-c(1.2, 3.4, 2.1, 5.5)
> Mass<-c(2.5, 4.2, 5.6, 3.4)
> Sex<-c("male", "female", "male", "female")

> dat<-data.frame(Time, Mass, Sex)
> dat
   Time  Mass     Sex
1   1.2   2.5    male
2   3.4   4.2  female
3   2.1   5.6    male
4   5.5   3.4  female
```

Checking characteristics of data frame

```
> class(dat)
[1] "data.frame"
```

```
> length(dat)
[1] 3
```

with a data frame, `length( )` gives the number of variables

```
> dim(dat)
[1] 4 3
```

with a data frame, `dim( )` gives the number of variables and number of observations (rows)

Checking characteristics of data frame

For the structure of the data frame

```
> str(dat)
'data.frame':   4 obs. of  3 variables:
 $ Time: num  1.2 3.4 2.1 5.5
 $ Mass: num  2.5 4.2 5.6 3.4
 $ Sex : Factor w/ 2 levels "female","male": 2 1 2 1
```

With a data frame, `summary( )` yields descriptive stats for each variable

```
> summary(dat)
 Time            Mass            Sex
 Min.   :1.200   Min.   :2.500   female:2
 1st Qu.:1.875   1st Qu.:3.175   male   :2
 Median :2.750   Median :3.800
 Mean   :3.050   Mean   :3.925
 3rd Qu.:3.925   3rd Qu.:4.550
 Max.   :5.500   Max.   :5.600
```

Checking characteristics of data frame

Identify variables in data set

```
> names(dat)
[1] "Time" "Mass" "Sex"
```

To see first observations for each variable

```
> head(dat)
```

To see last observations for each variable

```
> tail(dat)
```

Data frames

Creating a data frame in a single fell swoop

```
> data_set<-data.frame(Speed=c(25, 40, 70, 100),
Type=c(”car", ”truck", ”car", ”truck"))
> data_set
    Speed   Type
1      25    car
2      40  truck
3      70    car
4     100  truck
> Speed
Error : object "Speed" not found
```

How can we access variables???

Access to one or more variables

We have a few options

Option 1: use the $ operator

```
> data_set$Type
[1] car  truck car  truck
Levels: car truck
```

Option 2: use the [,] operator

```
> data_set[,2]
[1] car   truck car   truck
Levels: car truck
```

Type corresponds to column 2

Option 3: attach the data frame with `attach( )`

```
> attach(data_set)
> Speed
[1]  25  40  70 100
```

Enables the direct access to variables, but beware of transformations...

My least favorite...

Importing files

Very unpractical to enter data by hand.

Fortunately, we can easily import data sets (in text format) in R.

We can choose whatever extension we want, but the content must be raw text (not XLS format).

Data structure

Data collected during experiment:

Control	FertA	FertB	FertC
6.1	6.3	7.1	8.1
5.9	6.2	8.2	8.5
5.8	5.8	7.3	7.6
5.4	6.3	6.9	7.8

Appropriate structure for reading and importing files:

Response	Treatment
6.1	Control
5.9	Control
5.8	Control
5.4	Control
6.3	FertA
6.2	FertA
...	...

1 variable = 1 column

this format is required by most of the classic statistical analyses in R

Preparing to import in R

When the file is in the correct format (1 column = 1 variable), we can save it as a text file `.txt` (tab delimited; space delimited) from `MS Excel` or `Calc` in `OpenOffice`.

Very important:

First line of each column must have the header (name of the variable)

No spaces within header names or within data in a given column

Ex.
mean height INCORRECT mean_height or mean.height CORRECT

With missing data, must use `NA` to indicate "Not Available"
(no empty cells allowed in a given column)

Many ways to import data in R, depending on the type of data (e.g., data set, matrix, etc...), the following is for typical scenarios.

Importing in R

To import:

function `read.table( )`

```
worms<-read.table("c:\\path_on_your_computer\\worms.txt",  header=TRUE)
```

(`header = TRUE` tells R that the names of variables are on the first line)

Important points:

Specify the correct path to the file (upper and lower case matter)

Use either \\ or / in the path, but not \ as in Windows

Put the path together with file name between " "

Importing in R

Three equivalent strategies:

1) specify the complete path to the file

```
worms<-read.table(file="c:\\full_path\\worms.txt", header=TRUE)
```

2) use function `file.choose()` to choose a file interactively in a new window

```
worms<-read.table(file=file.choose(), header=TRUE)
```

3) assign a working directory where files are stored at the beginning of the session

```
setwd(dir="c:\\full_path\\")
```

`getwd( )` identifies current working directory

```
worms<-read.table(file= "worms.txt", header=TRUE)
```

Basic manipulations in R

```
> worms[1:10,]
          Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
1        Nashs.Field  3.6    11  Grassland     4.1 FALSE            4
2     Silwood.Bottom  5.1     2     Arable     5.2 FALSE            7
3      Nursery.Field  2.8     3  Grassland     4.3 FALSE            2
4        Rush.Meadow  2.4     5     Meadow     4.9  TRUE            5
5    Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
6           Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
7       Church.Field  3.5     3  Grassland     4.2 FALSE            3
8            Ashurst  2.1     0     Arable     4.8 FALSE            4
9         The.Orchard  1.9     0    Orchard     5.7 FALSE            9
10     Rookery.Slope  1.5     4  Grassland     5.0  TRUE            7
```

Basic manipulations in R

To access variables directly (after all transformations have been made)

Attach object with function `attach( )`

To check the names of variables in the data set

Use function `names ( )`

```
> names(worms)
[1] "Field.Name"   "Area"         "Slope"        "Vegetation"
"Soil.pH"      "Damp"         "Worm.density"
```

Basic manipulations in R

To get a brief summary of each variable (mean, quartiles)

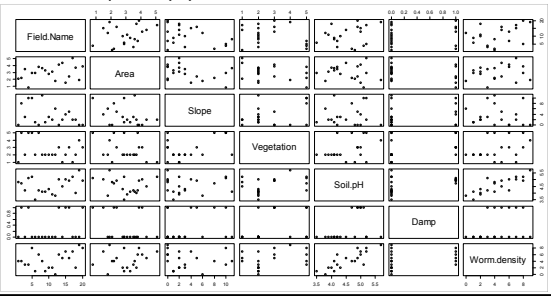
Use function `summary ( )`

```
> summary(worms)
        Field.Name      Area           Slope           Vegetation
 Ashurst     : 1   Min.   :0.800   Min.   : 0.00   Arable   :3
 Cheapside   : 1   1st Qu.:2.175   1st Qu.: 0.75   Grassland:9
 Church.Field: 1   Median :3.000   Median : 2.00   Meadow   :3
 Farm.Wood   : 1   Mean   :2.990   Mean   : 3.50   Orchard  :1
 Garden.Wood : 1   3rd Qu.:3.725   3rd Qu.: 5.25   Scrub    :4
 Gravel.Pit  : 1   Max.   :5.100   Max.   :11.00
 (Other)     :14
    Soil.pH         Damp          Worm.density
 Min.   :3.500   Mode :logical   Min.   :0.00
 1st Qu.:4.100   FALSE:14        1st Qu.:2.00
 Median :4.600   TRUE :6         Median :4.00
 Mean   :4.555                   Mean   :4.35
 3rd Qu.:5.000                   3rd Qu.:6.25
 Max.   :5.700                   Max.   :9.00
```

Basic manipulations in R

To get a matrix of plots for all 2-variable combinations

Use function `plot ( )`



Basic manipulations in R

To select part of a data set

Use indices and `[rows, columns]`

`> worms[1,]` selects first row

`> worms[1:15,]` selects first 15 lines

`> worms[,3]` selects third column

`> worms[,3:5]` selects columns 3 - 5

`> worms[1,3:5]` selects first row of columns 3 - 5

Basic manipulations in R

Logical tests

`>, <, >=, <=, ==, !=`

```
> worms$Slope
 [1] 11  2  3  5  0  2  3  0  0  4 10  1  2  6  0  0  8  2  1 10
```

`> worms$Slope==0` (strictly equal to 0)

```
 [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE
FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
> worms$Slope>0
[1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
TRUE  TRUE TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

`> worms$Slope!=0` (different from 0)

```
 [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
TRUE  TRUE TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

Basic manipulations in R

Logical tests

```
> worms$Vegetation
 [1] Grassland Arable    Grassland Meadow    Scrub     Grassland
Grassland     Arable    Orchard   Grassland Scrub     Grassland
Grassland Grassland Meadow    Meadow    Scrub     Arable
Grassland Scrub
Levels: Arable Grassland Meadow Orchard Scrub
```

With factors (characters), must put value between " "

```
> worms$Vegetation=="Grassland"
 [1]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
```

Basic manipulations in R

Logical tests

`ifelse( )` conditonal test on a vector or string of characters

Create a binary variable

```
> worms$Grassland<-ifelse(worms$Vegetation=="Grassland", 1, 0)
> worms$Grassland
 [1] 1 0 1 0 0 1 1 0 0 1 0 1 1 1 0 0 0 0 1 0
```

Nesting `ifelse( )` to create multiple categories

```
> worms$Slope_cat<-ifelse(worms$Slope >= 0 & worms$Slope < 3, "Low",
ifelse(worms$Slope >= 3 & worms$Slope < 6, "Med",
ifelse(worms$Slope >= 6 & worms$Slope < 9, "High", "Very High")))
> worms$Slope_cat
 [1] "Very High" "Low"       "Med"       "Med"       "Low"  "Low"
 [7] "Med"       "Low"       "Low"       "Med"       "Very High“ “Low"
[13] "Low"       "High"      "Low"       "Low"       "High"      "Low"
[19] "Low"       "Very High"
```

Basic manipulations in R

Logical tests

`which( )` to determine which elements match a given condition

```
> which(worms$Slope==0)
[1]  5  8  9 15 16

> which(worms$Vegetation=="Grassland")
[1]  1  3  6  7 10 12 13 14 19
```

Basic manipulations in R

Using logical tests to select subset of data frame

```
> worms[worms$Vegetation=="Meadow",]
```

selects lines for which `Vegetation` = `Meadow`

```
> worms[worms$Area > 3 & worms$Slope < 3,]
```

selects lines matching both conditions

Equivalently, one can use function `subset( )`

```
> subset(x=worms, subset=worms$Area > 3 & worms$Slope < 3 )
```

Basic manipulations in R

To sort a variable

Use `sort( )` for ascending order

```
> worms$Slope
 [1] 11  2  3  5  0  2  3  0  0  4 10  1  2  6  0  0  8  2  1 10
```

`> sort(worms$Slope)` order values

```
 [1]  0  0  0  0  0  1  1  2  2  2  2  3  3  4  5  6  8 10 10 11
```

Use either `rev(sort( ))` for descending order or `sort( , decreasing = FALSE)`

```
> rev(sort(worms$Slope))
 [1] 11 10 10  8  6  5  4  3  3  2  2  2  2  1  1  0  0  0  0  0
```

Basic manipulations in R

To sort a data frame

Use `order( )` for ascending order

```
> worms[order(worms[,1]), 1:6]
```

sorts columns 1 - 6 by values of column 1 in ascending order

Use either `rev(order( ))` for descending order or `order( , decreasing = FALSE)`

```
> worms[rev(order(worms[,1])), 1:6]
```

sort columns 1 - 6 by values of column 1 in descending order

Useful functions

How can we create a pivot table?

`table( )` summarizes the frequency of each category of one or more factors

```
> table(worms$Vegetation)
Arable Grassland    Meadow   Orchard     Scrub
        3         9         3         1         4

> table(worms$Vegetation, worms$Damp)
            FALSE TRUE
  Arable        3    0
  Grassland     8    1
  Meadow        0    3
  Orchard       1    0
  Scrub         2    2
```

Useful functions

For a better example, let's look at file `Neural.txt` with the pain reduction data

```
> neural<-read.table("C:/Neural.txt", header=TRUE)
> neural[1:10,]
   Reduce Treatment Age Gender Duration
1     1         1  76      M       36
2     1         1  52      M       22
3     0         0  80      F       33
4     0         1  77      M       33
5     0         1  73      F       17
6     0         0  82      F       84
7     0         1  71      M       24
8     0         0  78      F       96
9     1         1  83      F       61
10    1         1  75      F       60
```

Useful functions

```
> tab<-table(neural$Reduce, neural$Treatment, deparse.level=2)
> tab
           neural$Treatment
neural$Reduce 0 1
          0 8 3
          1 0 7
```

`deparse.level=2` adds the names of each table dimension

`as.data.frame( )` coerces the table into a data frame and we can use it later in an subsequent analyses

```
> tab<-as.data.frame(tab)
> tab
  neural.Reduce neural.Treatment Freq
1           0               0    8
2           1               0    0
3           0               1    3
4           1               1    7
```

Useful functions

If a column contains frequencies, we can summarize the frequencies with `xtabs( )`

```
> admissions <- as.data.frame(UCBAdmissions)
> admissions
      Admit Gender Dept Freq
1  Admitted   Male    A  512
2  Rejected   Male    A  313
3  Admitted Female    A   89
4  Rejected Female    A   19
...
> xtabs(Freq~Gender+Admit, data=admissions)
        Admit
Gender   Admitted Rejected
  Male       1198     1493
  Female      557     1278
```

many data sets are already included in R, use `data( )` to see which are available.

Useful functions

`reshape( )` rearranges data in either wide or long formats

```
> Indometh
   Subject time conc
1        1 0.25 1.50
2        1 0.50 0.94
3        1 0.75 0.78
4        1 1.00 0.48
5        1 1.25 0.37
6        1 2.00 0.19
7        1 3.00 0.12
8        1 4.00 0.11
9        1 5.00 0.08
10       1 6.00 0.07
11       1 8.00 0.05
12       2 0.25 2.03
13       2 0.50 1.63
...
>
Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25 conc.2 conc.3 conc.4 conc.5 conc.6 conc.8
      1      1.50     0.94      0.78   0.48      0.37   0.19   0.12   0.11   0.08   0.07   0.05
      2      2.03     1.63      0.71   0.70      0.64   0.36   0.32   0.20   0.25   0.12   0.08
...
```

see also `stack( )` and `unstack( )`

Exporting data frames or other objects from R

`write.table( )` function

```
write.table(x=name_of_object, file="c:\\full_path\\worms.txt",
row.names=FALSE, col.names=TRUE, sep="\t")
```

`row.names=FALSE` indicates that we do not want rownames (if desired, use = `TRUE`)

`col.names=TRUE` indicates that we want to export column names

`sep="\t"` indicates that we want tabs as delimiters

(`sep=" "` for space as delimiter)

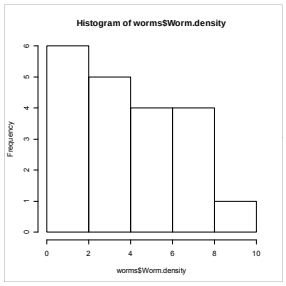
(`sep=","` for comma as delimiter)

Graphics

Graphics in R

histogram

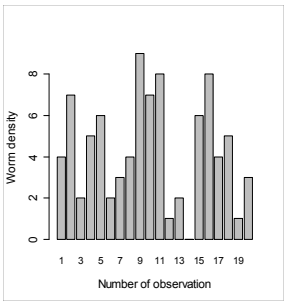
```
> hist(worms$Worm.density)
```



Graphics in R

barplot (this is not a histogram)

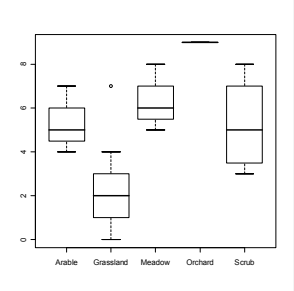
```
> barplot(worms$Worm.density,
xlab="Number of observation",
ylab="Worm density",
names.arg=c(1:20))
```



Graphics in R

boxplot

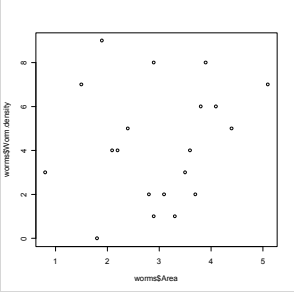
```
> boxplot(worms$Worm.density~
worms$Vegetation)
```



Graphics in R

scatter plot

```
> plot(worms$Worm.density~
worms$Area)
```



Warning:

Some functions (graphics and others) have many arguments

`plot(x, y, ...)`

If each argument is named explicitly, ordering is irrelevant

`plot(x=2, y=10)` or `plot(y=10, x=2)`

If only values are given, the order must be correct or terrible things may happen

`plot(2, 10)` ≠ `plot(10, 2)`

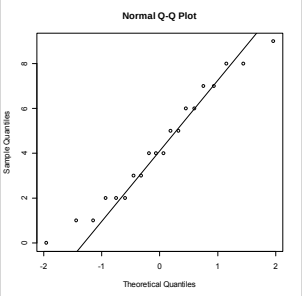
Graphics in R

quantile-quantile plot

```
> qqnorm(worms$Worm.density)
```

to check normality of residuals

```
> qqline(worms$Worm.density)
```



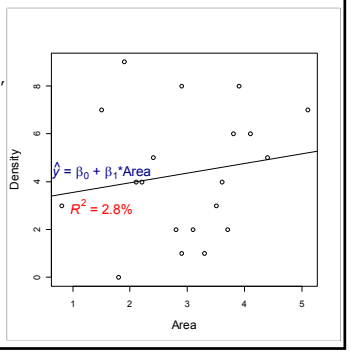
Graphics in R

Run regression, then add line on graph with predictive equation

```
> mod1<-lm(Worm.density~Area, data=worms)

> abline(reg=mod1)

> text(x= , y= , labels= , col= , family= , cex=, ...)
```



Graphics in R

other graphs

`lines( )` to draw lines

`pie( )` to bake pies

To get an idea of the possibilities, consult the R graph gallery:
http://addictedtor.free.fr/graphiques/

134 different types

Graphics in R

Great flexibility in the construction of graphics:
you can change everything

```
> ?par
```

`par` presents most of the available graphical arguments for `plot( )` and many other similar functions

ex.: axes, labels, scales, font, size (characters, labels, etc...), italics, bold, color, Greek letters, exponents, indices, axis styles, symbols, margins, legends, text, etc...

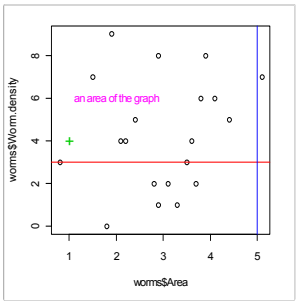
Graphics in R

one of the advantages is that one can easily add elements on the graph once it is drawn

```
> abline(h=3, v=5, col=c("red",
 "blue"))

> points(x=1, y=4, pch="+",
col=3, cex=1.5)

> text(x=2, y=6, labels=
"an area in the graph",
col="magenta")
```



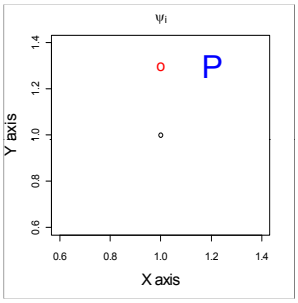
Graphics in R

```
> plot(x=1,y=1,
ylab="Y axis",
xlab="X axis",
cex.lab=1.5)

> mtext(side=3,
text=expression(psi[i]),
line=1, cex=1.3)

> points(x=1, y=1.3, pch="o",
col=2, cex=1.5)

> points(x=1.2,y=1.3, pch="P",
col="blue", cex=3)
```



Graphics in R

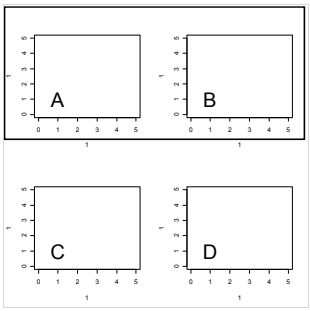
One can easily put several graphs in the same window

`> par(mfrow=c(rows, columns))` fills one row at a time

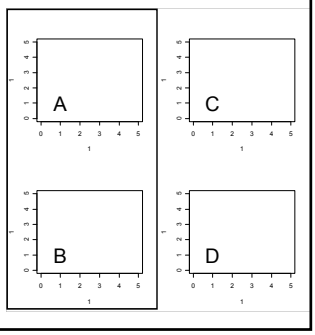
`> par(mfcol=c(rows, columns))` fills one column at a time

Graphics in R

```
> par(mfrow = c(2, 2))
```




```
> par(mfcol = c(2, 2))
```



Graphics in R

In Windows, one can copy and paste directly from R to word processor (as a metafile)

Alternatively, one can save in different formats such as `pdf`, `eps`, `jpeg`, `bitmap`, `png` (see `?Devices`)

to create `pdf` file:

```
> pdf(file="C:\\Fig1.pdf")
> plot(x=c(1,2), y=c(6,5))
> dev.off( )
```

`dev.off( )` tells to R that the figure is complete and to revert to previous graphical device

to create `postcript` file:

```
> postscript(file="C:\\Fig1.ps")
> plot(x=c(1,2), y=c(6,5))
> dev.off( )
```

Graphics in R

to create `eps` file:

1) display graph in default window (screen):

```
> windows( )
> plot(x=c(1,2), y=c(6,5))
```

2) convert graph into `eps`:

```
> dev.copy2eps(file="C:\\Fig1.eps")
```

I strongly recommend R to create your graphs

...be bold and jump in

Basic statistics

Statistical distributions

Functions associated with statistical distributions

`dnorm( )` probability density function

```
> dnorm(x=5,mean=2,sd=3)
[1] 0.08065691
```

density of function when x = 5 for normal distribution with $\mu = 2$ and $\sigma = 3$.

`pnorm( )` cumulative distribution function (distribution function)

```
> 1-pnorm(q=1.96, mean=0, sd=1)
[1] 0.02499790
```

probability of observing x > 1.96 in standard normal distribution

$1 - P(X \leq 1.96) = p$

`qnorm( )` quantile function

```
> qnorm(p=0.975, mean=0, sd=1)
[1] 1.959964
```

quantile associated with probability of 0.025 in a standard normal distribution (inverse of `pnorm( )`).

$1 - P(X \leq x) = 0.025$

Statistical distributions

Functions associated with statistical distributions

`rnorm( )` generate random values coming from normal distribution

```
> var1<-rnorm(n=30, mean=12, sd=3)
```

yields 30 random observations from normal distribution with $\mu = 12$ and $\sigma = 3$.

```
> var1
[1] 11.637296  8.818745  8.548053 10.528768 11.916288 11.852185 11.897473
12.513795 13.869845  8.005414 12.540423 10.237092 10.547166  5.900474 10.276910
17.227563 10.774479  7.130979  9.368063 15.500556 18.154208 12.687525 10.602007
11.039222  9.475898 13.373740 12.765565  9.316305 14.211309 10.649276
```

Many functions for other distributions have same structure

Ex.

X^2	Student 's *t*	*F*	Poisson	binomial	uniform
`dchisq( )`	`dt( )`	`df( )`	`dpois( )`	`dbinom( )`	`dunif( )`
`pchisq( )`	`pt( )`	`pf( )`	`ppois( )`	`pbinom( )`	`punif( )`
`qchisq( )`	`qt( )`	`qf( )`	`qpois( )`	`qbinom( )`	`qunif( )`
`rchisq( )`	`rt( )`	`rf( )`	`rpois( )`	`rbinom( )`	`runif( )`

Descriptive statistics

`mean( )` compute arithmetic mean

```
> mean_var1<-mean(var1)
> mean_var1
[1] 11.37889
```

`sd( )` compute standard deviation

```
> sd_var1<-sd(var1)
> sd_var1
[1] 2.715868
```

} not surprising, since we generated data coming from a normal distribution with $\mu = 12$ and $\sigma = 3$.

`median( )` compute median

```
> median(var1)
[1] 10.90685
```

`quantile( )` compute quantiles (quartiles by default)

```
> quantile(var1)
       0%       25%       50%       75%      100%
 5.900474  9.666196 10.906851 12.650750 18.154208
```

Descriptive statistics

`rank( )` compute ranks

```
> rank(var1)
[1] 17  5  4 11 20 18 19 21 26  3 22  9 12  1 10 29 15  2  7 28 30
23 13 16  8 25 24  6 27 14
```

No function for *SE*

```
> SE_var1<-sd(var1)/sqrt(length(var1))
> SE_var1
[1] 0.4958474
```

we can easily create functions in R

Creating your own R functions

R as a programming language

R includes a plethora of functions ready to use

It is easy to create your own functions with `function( )`

A formula to compute *SE*:
`sd(var1)/sqrt(length(var1))`

Before starting, it is worthwhile to identify the elements needed.

To compute the *SE*, we need:

n (number of observations)
standard deviation

Creating a function in R

A function is just another object type and we assign it a name.

```
> SE_function<-function(x) {
```

`function( )` indicates that we're creating a function, and we specify its arguments between `( )`

here, the argument indicates that we use a variable *x*

the body of the function (i.e., computations or manipulations to perform) is enclosed between `{ }`

Creating a function in R

A function is just another object type and we assign it a name.

```
> SE_function<-function(x) {
  n_x<-length(x)
  sd_x<-sd(x)
  sd_x/sqrt(n_x)
  }

> SE_function(x=var1)
[1] 0.4958474
```

determine *n* for variable *x*

compute *SD* of variable *x*

compute *SE* with elements required

add } to indicate that function ends

use newly created function

All R functions have this structure
(R can also call external functions coded in C, C++, or FORTRAN).

Creating a function in R

One can create all sorts of functions to conduct repetitive tasks.

Another example of simple functions: square root

```
> sq.root<-function(value) {          argument is 'value'
value^(1/2)     compute square root
}
```

Run function

```
> sq.root(16)
[1] 4
```

```
> sq.root(329)
[1] 18.13836
```

Check if it yields the same results as `sqrt( )`

```
> sqrt(16)
[1] 4
```

```
> sqrt(329)
[1] 18.13836
```

Creating a function in R

A function may include many arguments

n^{th} root

```
> root_x<-function(value,level) {
value^(1/level)
}
> root_x(8,3)
[1] 2
```

an argument for the value, and another to request the level

```
> root_x<-function(value,level=2) {
value^(1/level)
}
> root_x(16,2)
[1] 4
> root_x(16)
[1] 4
```

we can also assign default values for any argument (`level = 2` requests square root by default)

Creating a function in R

One can create functions during a session and store them in code.

Even better, save the functions in a separate file and access them with `source( )` when needed.

```
> source(file="C:/path_on_your_computer/my_fonction.txt")
```

(useful when functions span over several lines)

Creating a function in R

A function may return many elements.

Extract sum, min, and max of a variable

```
> sum_max_min<-function(variable){
the_sum<-sum(variable)
the_min<-min(variable)
the_max<-max(variable)
list("Sum_of_observations" = the_sum, "Minimum_value" = the_min,
"Maximum_value" = the_max)
}
```

`list( )` enables us to arrange the results in a list

```
> x<-rnorm(10)     > sum_max_min(x)
                   $Sum_of_observations
                   [1] -0.7677199

                   $Minimum_value
                   [1] -2.700584

                   $Maximum_value
                   [1] 1.031162
```

Displays the values on screen

Creating a function in R

Better to create an object to store the output of the function and extract the elements later (e.g., to graph or use in additional computations)

```
> result<-sum_max_min(x)
> class(result)
[1] "list"
```

a list is another object type

One can check the contents of the list with `names( )`

```
> names(result)
[1] "Sum_of_observations" "Minimum_value"       "Maximum_value"
```

Creating a function in R

To see the results

```
> result
$Sum_of_observations
[1] -0.7677199

$Minimum_value
[1] -2.700584

$Maximum_value
[1] 1.031162
```

To extract the elements, use the name of the object and the $ operator

```
> result$Sum_of_observations  > result$Minimum_value  > result$Maximum_value
[1] -0.7677199                [1] -2.700584           [1] 1.031162
```

Creating a function in R

We can generally use a similar strategy to access the output from analyses in R.

Many functions store output in an object and we can extract the required elements later (no need to run the analysis again)

Loops

Creating loops

Can avoid doing repetitive and boring tasks by hand

Simple example:

generate 100 samples of 30 observations from a normal distribution $N(\mu = 10, \sigma = 1)$, and compute the mean of each sample

first, create an object to store the means computed (output)

```
> output<-rep(x=NA, times=100)
```

the loop *per se*

```
> for (i in 1:100) {
simdata<-rnorm(n=30, mean=10, sd=1)
output[i]<-mean(simdata)
}
```

the computation or manipulation on each iteration is enclosed within { }

generate 1 sample of 30 obs, compute mean, and store value (repeat 100 times)

Creating loops

```
> output
10.187493 10.057872 10.400478 10.150307  9.948904  9.713075 10.050015
 9.799055  9.928758 10.038981  9.775561 10.241103  9.966706 10.058531
10.099403  9.814790  9.884374  9.840617 10.112026  9.886627 10.165894
10.076277  9.505789  9.976422  9.977487  9.741762  9.859750  9.689165
10.421739 10.101839  9.932801  9.985449 10.054757 10.102494 10.121907
10.093705  9.793565 10.159726  9.902479  9.986425 10.311193  9.801151
10.102233  9.943137  9.999567 10.246178  9.798786 10.160431  9.928409
10.119910  9.896694 10.186127  9.690116 10.020563  9.829083  9.890244
 9.996526 10.089126 10.014033  9.466073 10.234593 10.022860  9.917595
10.120968  9.827922  9.961950  9.961035  9.874266 10.256765 10.042088
 9.957012 10.080500 10.117559 10.311178  9.997452  10.038956 10.128459
10.041864 10.044567  9.857874 10.207001 10.004664   9.804787  9.897710
10.256215  9.955179 10.073420  9.935395  9.673318 10.363609  10.016773
10.211726 10.145192 10.032976 10.085238 10.085375  9.843844 10.056572
10.159683  9.921390
> mean(output)
[1] 10.00619
```

Creating loops

Useful to prepare data for analysis

Applies naturally to bootstrap and randomization tests

Can be used in more complex situations:

Ex.

1) import 1000 text files of output from another program,
2) in each file, find lines corresponding to estimates and SE's,
3) extract these values and store them in an object,
4) export the results in a text file for later use and graphing.

Loops can be nested into one another

Can be included in functions

Bootstrapping

Original sample:

```
dbh <- c(3, 22, 8, 50, 9, 22, 26, 46, 28, 16, 33, 45, 18, 34)
```

bootstrap (compute the bootstrap estimate of the *SE*)

`nsims<-1000` determine number of iterations (bootstrap samples)

`boot_out<-rep(NA, nsims)` create an object to store output

`for (i in 1:nsims) {` start loop (`i = 1`)

`bootsample<-sample(x=dbh, size=length(dbh), replace=TRUE)`
create a bootstrap sample from the original sample (`dbh`)

`boot_out[i]<-mean(bootsample)`
compute mean of bootstrap sample `i` and store value in `boot_out[i]`

`}` repeat loop for (`i = 2`)

SE of the statistic (here, the mean) is given by `sd(boot_out)`

Creating loops

Certain functions act like loops:

`tapply( )` applies a function on elements of different levels of a factor

Ex. compute the mean of worm density for each vegetation type

```
> tapply(X=worms$Worm.density, INDEX=worms$Vegetation, FUN=mean)
   Arable Grassland    Meadow   Orchard     Scrub
 5.333333  2.444444  6.333333  9.000000  5.250000
```

Similar functions for different object types:

`apply( )`
`sapply( )`
`lapply( )`

Statistical analyses

χ^2 for frequency data

`chisq.test( )` tests whether proportions are the same according to different levels of a single factor or contingency table

Are the proportions of individuals across three age classes the same?

```
> Freq<-c(124,176, 180)
> chisq.test(Freq, p=c(1/3, 1/3, 1/3))
```

p = proportions conforming to H_0

```
  Chi-squared test for given probabilities

  data:  Freq
  X-squared = 12.2, df = 2, p-value = 0.002243
```

t-test for two groups

```
> data(sleep)
> sleep[1:12,]
   extra group
1    0.7     1
2   -1.6     1
3   -0.2     1
4   -1.2     1
5   -0.1     1
6    3.4     1
7    3.7     1
8    0.8     1
9    0.0     1
10   2.0     1
11   1.9     2
12   0.8     2
```

Use `sleep` data set already in `R` (`?sleep`)

One of two soporific drugs administered to 20 patients (10 of each group) and recording of number of extra hours of sleep relative to each student's baseline before taking drug

t-test for two groups

`t.test( )` tests difference between means of two groups

Many arguments possible:

`alternative = c("two.sided", "less", "greater")` two-tailed or one-tailed

`paired = FALSE` *t*-test for paired or independent groups

`mu = 0` H_0: $\mu_1 - \mu_2 = 0$ ($\mu_1 = \mu_2$)

`var.equal = FALSE` test with equal or unequal variances

`conf.level = 0.95` threshold for confidence interval

t-test for two groups

Run function with data

```
> t.test(extra~group, mu=0, data=sleep)
```

`data=sleep` indicates that data are in object `sleep`

```
 Welch Two Sample t-test

data:  extra by group
t = -1.8608, df = 17.776, p-value = 0.0794
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.3654832  0.2054832
sample estimates:
mean in group 1 mean in group 2
           0.75            2.33
```

By default, assumes variances are unequal (if equal, specify `var.equal = TRUE`)

t-test for two groups

Better to store output in object

```
> result<-t.test(extra~group, mu=0, data=sleep)
```

`names( )` to determine contents

```
> names(result)
[1] "statistic"   "parameter"   "p.value"     "conf.int"
"estimate"    "null.value"  "alternative" "method"      "data.name"
```

One can extract any element

```
> result$statistic
        t
-1.860813
> result$estimate
mean in group 1 mean in group 2
           0.75            2.33
```

Pearson correlation

`cor( )` compute Pearson's product-moment correlation

generate data

```
> x1<-rnorm(n=30, mean=2, sd=1)
> x2<-rpois(n=30, lambda=5)
> cor(x1,x2)
[1] 0.2605317
```

`cor.test( )` compute correlation between variables and test H_0: $\rho = 0$.

```
> cor.test(x1,x2)
        Pearson's product-moment correlation
data:  x1 and x2
t = 1.4279, df = 28, p-value = 0.1644
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1100687  0.5675316
sample estimates:
      cor
0.2605317
```

Linear regression

```
> chocs<-read.table("C:\\chocs_red.dat", header=TRUE)
> chocs
                Size Price
Dark.Bounty     50.0  0.88
Bounty          50.0  0.88
Milo.Bar        40.0  1.15
Viking          80.0  1.54
KitKat.White    45.0  1.15
KitKat.Chunky   78.0  1.40
Cherry.Ripe     55.0  1.28
Snickers        60.0  0.97
Mars            60.0  0.97
Crunchie        50.0  1.28
Tim.Tam         40.0  1.10
Turkish.Delight 55.0  1.28
Mars.Lite       44.5  0.97
Dairy.Milk.King 75.0  1.58
Maltesers       60.0  1.55
MandMs          42.5  1.18
```

Price of Australian chocolate bars as a function of mass.

Linear regression

`lm( )` fit linear models assuming errors following normal distribution (ANOVA, multiple regression, ANCOVA)

Several arguments possible, see `?lm`

Run linear regression

```
> mod_reglin<-lm(Price~Size, data=chocs)
```

if required, we can add explanatory variables and separating them by + in the model formula:

`Price ~ Size + Fat + Size:Fat`

: denotes interaction between two variables

Linear regression

`summary( )` extract summary of output

```
> summary(mod_reglin)
Call:
lm(formula = Price ~ Size, data = chocs)
Residuals:
     Min       1Q   Median       3Q      Max
-0.28031 -0.14368  0.07184  0.12546  0.29969
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.574368   0.213738   2.687  0.01769 *
Size        0.011266   0.003768   2.989  0.00975 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1891 on 14 degrees of freedom
Multiple R-Squared: 0.3896,     Adjusted R-squared: 0.346
F-statistic: 8.937 on 1 and 14 DF,  p-value: 0.009753
```

Linear regression

Contents of output

```
> names(mod_reglin)
 [1] "coefficients"  "residuals"     "effects"       "rank"
"fitted.values" "assign"        "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"


 > mod_reglin$coefficients
 (Intercept)        Size
  0.57436804  0.01126566

> mod_reglin$residuals
    Dark.Bounty          Bounty        Milo.Bar          Viking    KitKat.White
    -0.25765117     -0.25765117      0.12500546      0.06437895      0.06867715
  KitKat.Chunky     Cherry.Ripe        Snickers           Mars        Crunchie
    -0.05308972      0.08602052     -0.28030779     -0.28030779      0.14234883
       Tim.Tam Turkish.Delight       Mars.Lite Dairy.Milk.King       Maltesers
     0.07500546      0.08602052     -0.10569002      0.16070727      0.29969221
         MandMs
     0.12684130
```

Linear regression

There are many generic functions to extract additional elements from the output

`logLik( )` model log-likelihood

```
> logLik(mod_reglin)
'log Lik.' 5.009943 (df=3)
```

`influence.measures( )` leverage, Cook's distance, dfbeta

```
> influence.measures(mod_reglin)
Influence measures of
        lm(formula = Price ~ Size, data = chocs) :
               dfb.1_ dfb.Size  dffit cov.r  cook.d    hat inf
Dark.Bounty    -0.2427  0.16205 -0.416 0.919 0.07970 0.0737
Bounty         -0.2427  0.16205 -0.416 0.919 0.07970 0.0737
Milo.Bar        0.2712 -0.23449  0.303 1.274 0.04766 0.1556
Viking         -0.2013  0.23336  0.262 1.628 0.03646 0.3045   *
...
```

Linear regression

`rstudent( )` Student's residuals (good measure of outliers)

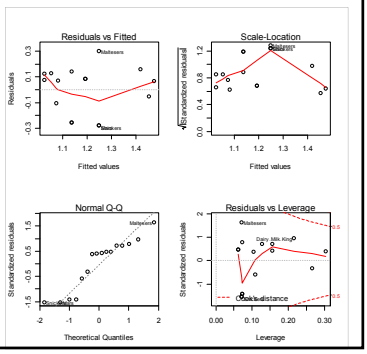
`anova( )` returns ANOVA table for model fit

```
> anova(mod_reglin)
Analysis of Variance Table

Response: Price
          Df  Sum Sq Mean Sq F value   Pr(>F)
Size       1 0.31969 0.31969  8.9369 0.009753 **
Residuals 14 0.50081 0.03577
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Linear regression

`plot( )`
generate 4 diagnostic plots



Linear regression

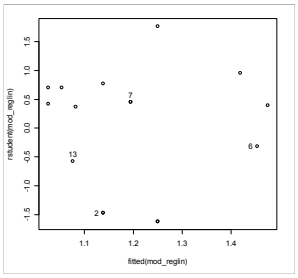
Plot of Student's residuals vs fitted values

```
> plot(rstudent(mod_reglin)~fitted(mod_reglin))
```

One can identify points on the graph with `identify( )`

```
> identify(rstudent(mod_reglin)
~fitted(mod_reglin))
```

left-clicking a point on the plot, will reveal its row in the data frame



ANOVA, ANCOVA, multiple regression

Depending on the nature of explanatory variables, `lm( )` will conduct ANOVA's, ANCOVA's or multiple regressions

to get ANOVA tables, use `anova( )`
(or `summary.aov( )` if design is balanced)

one can extract same elements from all these models

GLM's

`glm( )` fit generalized linear models from a distribution in the exponential family (normal aka Gaussian, Poisson, binomial, gamma)

syntax similar to `lm( )` and extractor functions work similarly

Let's run a logistic regression from the `neural` data set.

GLM's

`neural` data frame (imported previously).

Analysis of the probability of reducing pain in patients with neural disorder after treatment, across age and sex.

```
> mod1<-glm(Reduce~Age+Gender, family=binomial(link=logit), data=neural)
```

`family=binomial` requests a binomial GLM (logistic regression)

`link = logit` requests a logit link (canonical link is the default for all distributions)

GLM's

```
> summary(mod1)
Call:
glm(formula = Reduce ~ Age + Gender, family = binomial(link = logit),
    data = neural)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1760  -0.9603  -0.7633   1.0878   1.8536
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  7.06246    6.59271   1.071    0.284
Age         -0.09679    0.08605  -1.125    0.261
GenderM     -1.22652    1.37535  -0.892    0.373
(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 24.057  on 17  degrees of freedom
Residual deviance: 22.558  on 15  degrees of freedom
AIC: 28.558

Number of Fisher Scoring iterations: 4
```

Other examples of analyses

Linear mixed models and generalized linear mixed models (*GLMM*'s)

GLM extensions for repeated measures (*GEE*'s)

Ordinations (*PCA*, *CCA*, *RDA*) and other multivariate analyses

Randomizations: bootstrap, Monte Carlo

Nonlinear models

Optimization: custom-built models with user-defined likelihood functions solved by maximum likelihood

Packages: functions for various applications

Package = group of functions

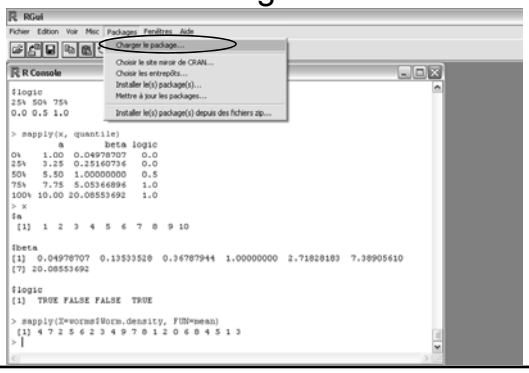
R functions are stored in packages

When starting R, some packages are loaded automatically to enable the use of most popular functions (e.g., packages base, graphics, stats)

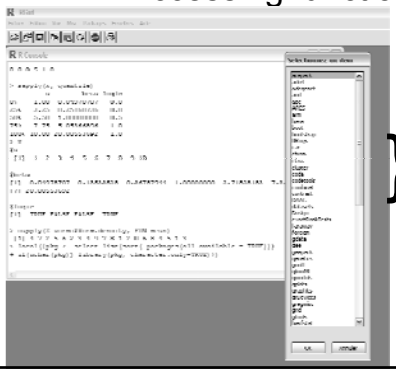
To use functions not included in the packages loaded at startup, one must load the package containing the function(s) of interest (with the default installation, several packages are ready to load)

```
library( )
```

Accessing functions



Accessing functions



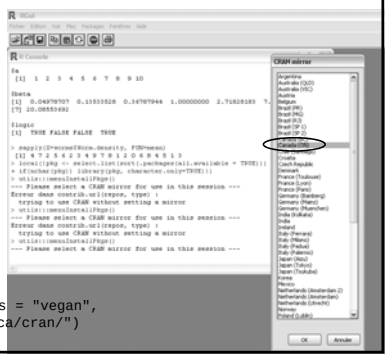
list of packages already installed

Better yet, just issue the `library( )` command with the name of the package between ( )

e.g., `> library(MASS)`

Accessing functions

If the package needed is not installed on your computer, you must download and install it from CRAN.



Better yet, just issue the `install.packages( )` command

e.g., `> install.packages(pkgs = "vegan", repos="http://probability.ca/cran/")`

Trailblazing

Many possibilities with the packages of default installation

More than 2000 packages available on CRAN to run a diverse array of analyses, import/export specific file types, interact with other software, peform tasks, etc...

Analyses such as *CCA*, neural networks, Bayesian analyses, mixed models

Trailblazing

To see a brief summary of the packages, go to CRAN

A few packages of interest

Bootstrap
`boot`, `bootstrap`

Linear mixed models and generalized linear mixed models
`MASS`, `nlme`, `lme4`

Multivariate analyses (*CCA*, *RDA*, cluster)
`vegan`, `ade4`, `cluster`

Interact with other software
`RMARK`, `R2WinBUGS`, `Brugs`, `GRASS`

Model selection and multimodel inference
`AICcmodavg`

Utility functions
`car`, `Design`, `Hmisc`, `gregmisc`, `gmodels`, `foreign`

A few packages of interest

Other applications

spatial analyses: `geoR`, `sp`, `spatial`

time series analyses (ARIMA, etc...): `far`, `fARMA`

Bayesian analyses: `arm`, `bayescount`, `bayesm`, `bayesmix`

genetics: `adegenet`, `ape`, `apTreeshape`, `seqinr`, `genetics`

dendrochronology: `dplR`

A tip

During a session, only load the packages that you need.

Beware of conflicts between packages

Some packages mask R base functions which can result in nasty surprises.

To unload a package, use function `detach( )`

```
> detach(package:nlme)
```

A good idea to update packages to check if new version are available

More tips

New versions of packages are uploaded to CRAN regularly

A good idea to update packages to check if new version are available

either use GUI menu under Packages... update packages

better yet, use function `update.packages( )`

More tips

At the end of an R session, you have the option of saving the session with all the objects currently in memory.

If you save a session modified over a very long time, be sure that you know which objects you are using and that they have not been modified inadvertently (e.g., a subset of a data frame replacing the original data frame)

Check what is stored in memory with `ls( )`

```
> ls()
[1] "i"       "output"  "simdata" "worms"   "x"
```

To delete objects, use `rm( )`

```
> rm(x)
> ls()
[1] "i"       "output"  "simdata" "worms"
```

More tips

At the end of an R session, you have the option of saving the session with all the objects currently in memory.

Instead of saving your R session, save the code (i.e., your script in Tinn-R) associated with the session.

Don 't be stingy on using comments in your R scripts. It is then much easier to dive into your code after a long pause and use the relevant parts in other analyses.

R ressources on the web

CRAN:
http://cran.r-project.org/

R tips:
http://pj.freefaculty.org/R/Rtips.html

R site search:
http://finzi.psych.upenn.edu/search.html

Quick-R:
http://www.statmethods.net/index.html

R Graph Gallery:
http://addictedtor.free.fr/graphiques/allgraph.php

Books with good R introduction

Dalgaard, P. 2002. Introductory Statistics with R. Springer, New York.

Crawley, M. J. 2005. Statistics: An introduction using R. John Wiley and Sons, Chichester, UK.

More advanced books on R

Venables, W. N., and B. D. Ripley. 2002. Modern Applied Statistics with S. Springer-Verlag, New York, USA.

Venables, W. N., and B. D. Ripley. 2002. S Programming. Springer, New York, USA.

More advanced books

Regressions and *GLM*'s

Fox, J. 2002. An R and S-PLUS companion to applied regression. Sage Publications, Inc., London, UK.

Faraway, J. J. 2006. Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models. Chapman and Hall, New York, USA.

Mixed models

Pinheiro, J. C., and D. M. Bates. 2000. Mixed-effect models in S and S-PLUS. Springer Verlag, New York.

Gelman, A., and J. Hill. 2007. Data analysis using regression and multilevel/hierarchical models. Cambride University Press, New York, USA.

Zuur et al. 2009. Mixed effects models and extensions in ecology with R. Springer-Verlag, New York, USA.

More advanced books

Bootstrap and randomizations

Efron, B., and R. J. Tibshirani. 1998. An introduction to the bootstrap. Chapman & Hall/CRC, New York, USA.

Good, P. I. 2005. Introduction to statistics through resampling methods and R/S-PLUS. John Wiley & Sons, Hoboken, NJ, USA.

Multivariate analyses

Everitt, B. 2005. An R and S-PLUS companion to multivariate analysis. Springer, London, UK.

Genetics

Paradis, E. 2006. Analysis of phylogenetics and evolution with R. Springer, New York, USA.

Bayesian analyses

Albert, J. 2007. Bayesian computation with R. Springer, New York.

More advanced books

Modelling/optimization

Bolker, B. M. 2008. Ecological models and data in R. Princeton University Press, Princeton, New Jersey.

Pawitan, Y. 2001. In all likelihood: statistical modelling and inference using likelihood. Oxford University Press, Oxford.

Royle, J. A., and R. M. Dorazio. 2008. Hierarchical modeling and inference in ecology: the analysis of data from populations, metapopulations and communities. Academic Press, New York.

Spatial analyses

Bivand, R. S., E. J. Pebesma, and V. Gómez-Rubio. 2008. Applied spatial data analysis with R. Springer, New York, NY, USA.

Time series analyses

Shumway, R. H., and D. S. Stoffer. 2006. Time series analysis and its applications with R examples, 2nd edition. Springer, New York, NY, USA.

HAPPY PROGRAMMING!!!