Basic of R language

Dr. S. DEVAARUL

Learning aims

- Basic use of R and R help
- How to give R commands
- R data structures
- Reading and writing data
- Some more R commands (exercises)

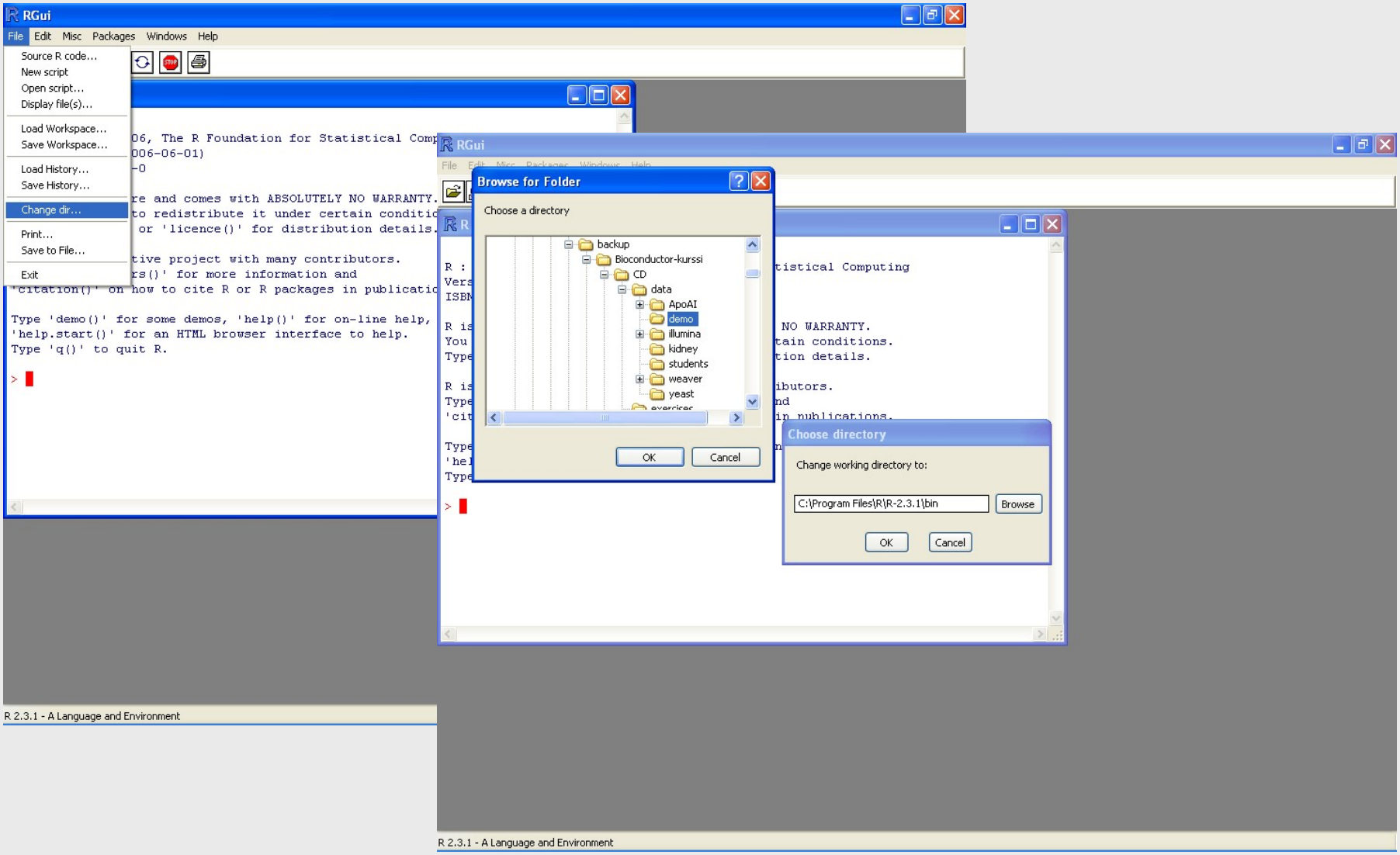
R project

- ”R is a free software environment for statistical computing and graphics” (http://www.r-project.org)
- ”Bioconductor is a software project for the analysis of genomic data” (http://www.bioconductor.org)
  - Currently works as an expansion to R

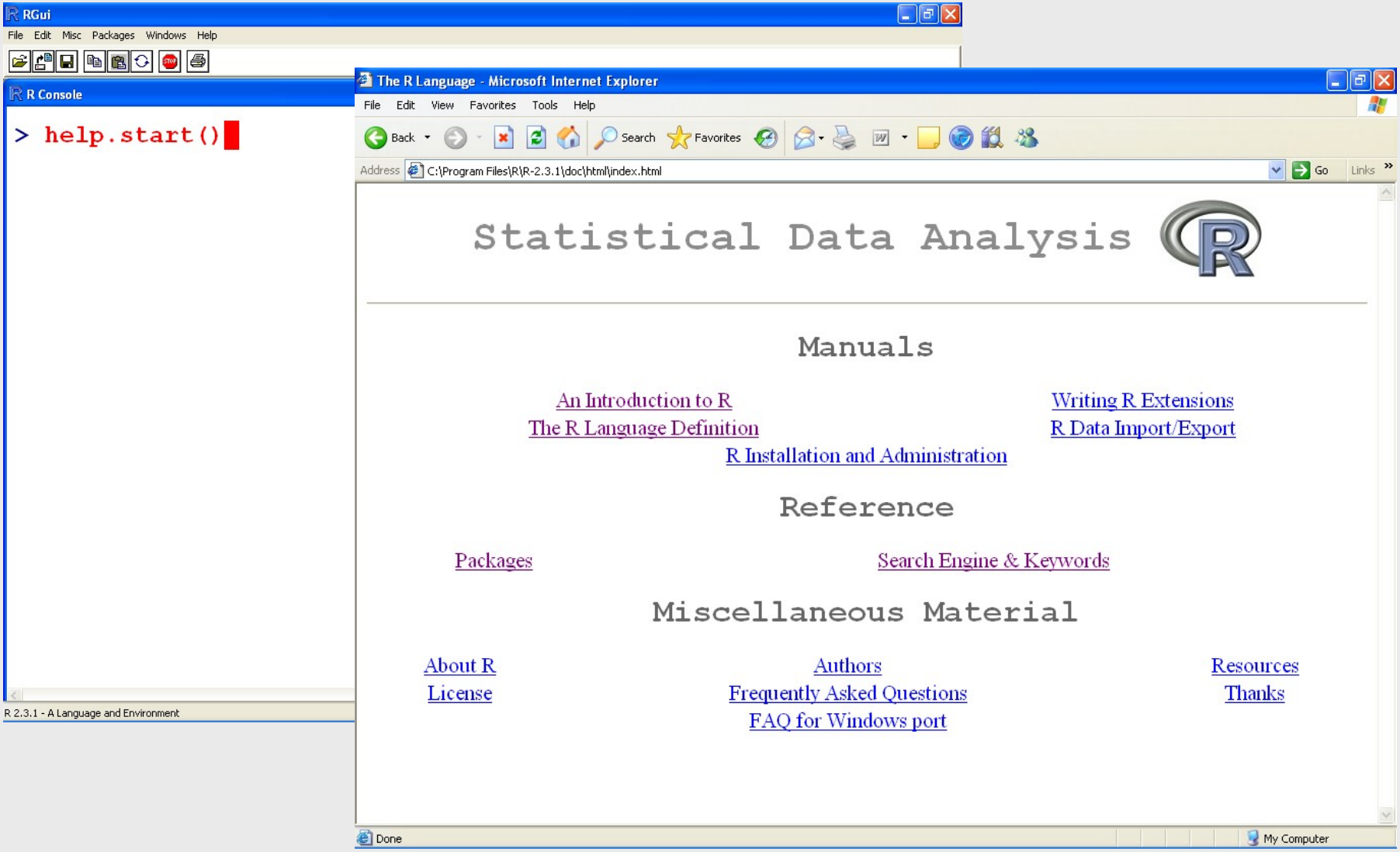
Packages

- R consists of a core and packages.
- Packages contain functions that are not available in the core.
- For example, Bioconductor code is distributed as several dozen of packages for R.
  - Software packages
  - Metadata (annotation) packages

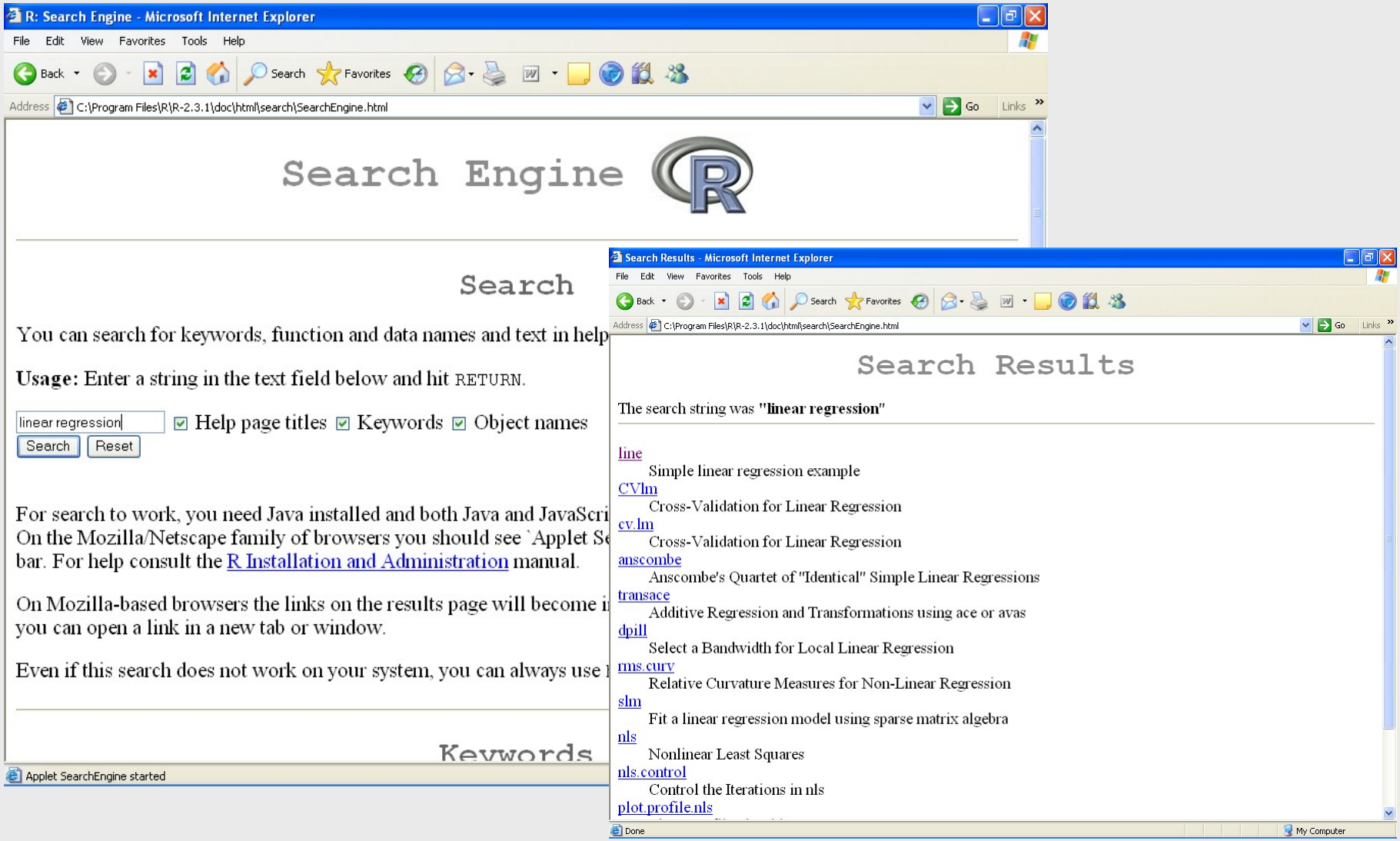
Starting the work with R



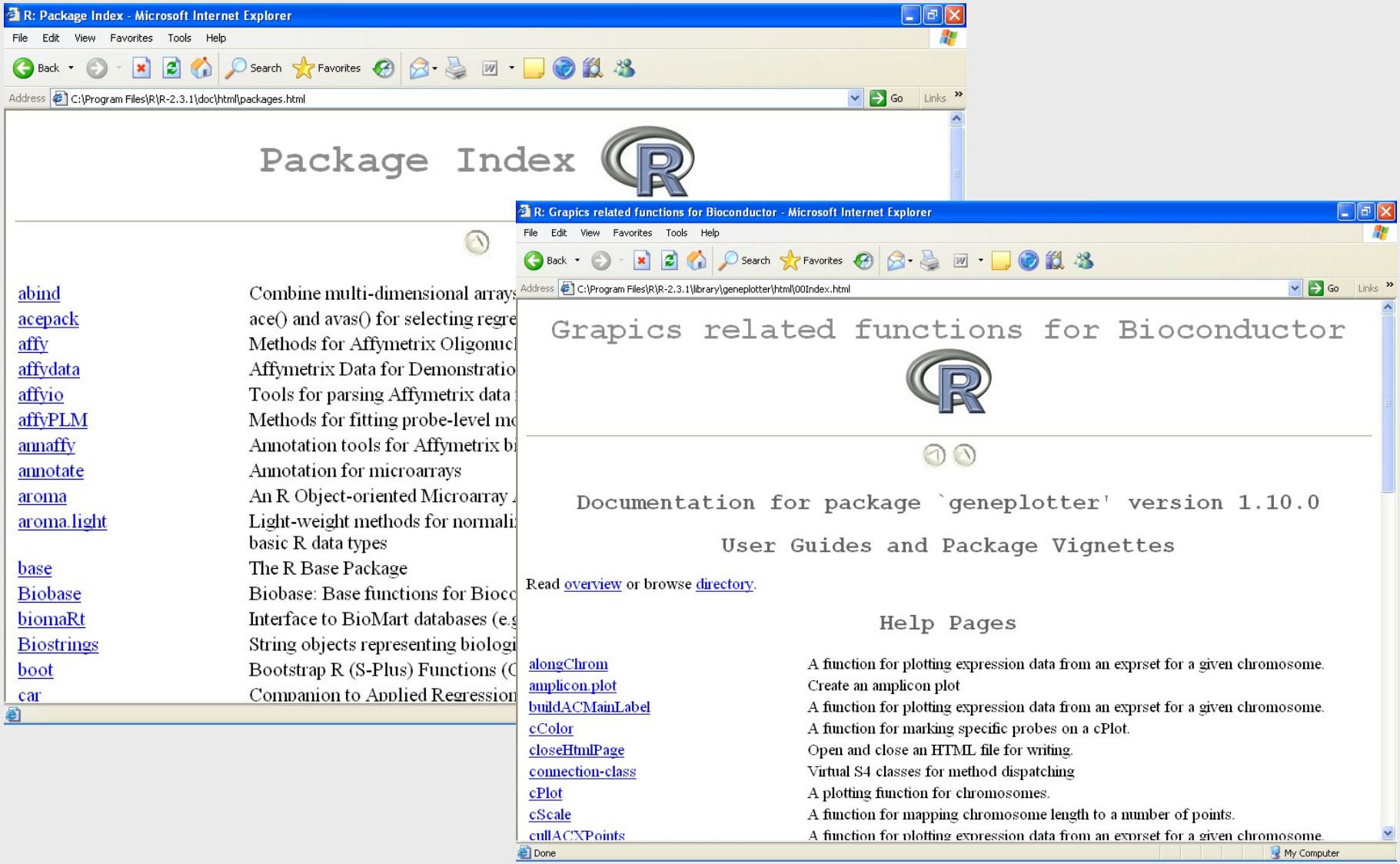
Start help


RGui
File Edit Misc Packages Windows Help
R Console
> help.start()
R 2.3.1 - A Language and Environment
The R Language - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Program Files\R\R-2.3.1\doc\html\index.html
Go Links
Statistical Data Analysis
Manuals
An Introduction to R
The R Language Definition
Writing R Extensions
R Data Import/Export
R Installation and Administration
Reference
Packages
Search Engine & Keywords
Miscellaneous Material
About R
License
Authors
Frequently Asked Questions
FAQ for Windows port
Resources
Thanks
Done
My Computer

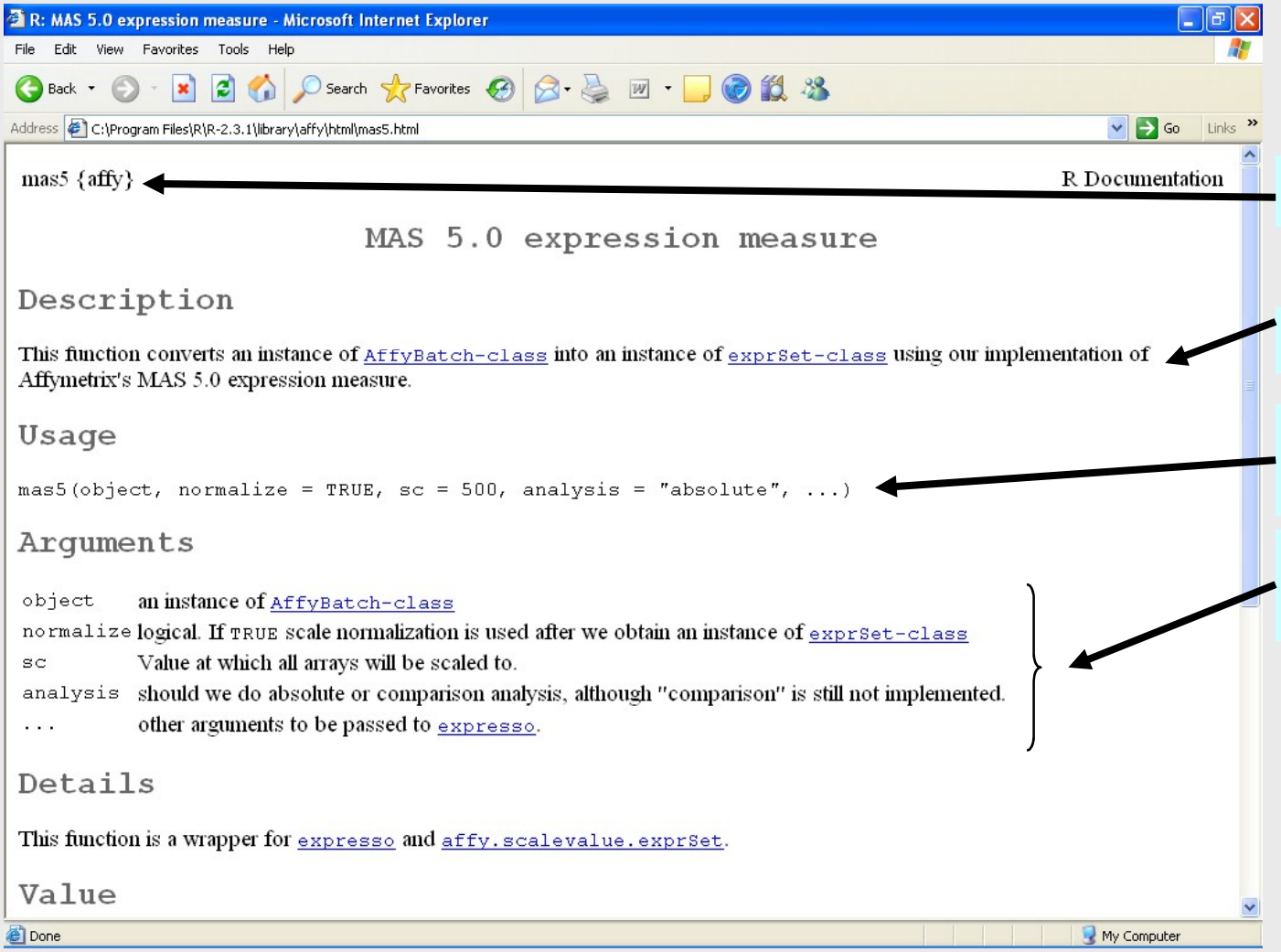
Help - Search engine



Help - packages



Anatomy of a help file 1/2



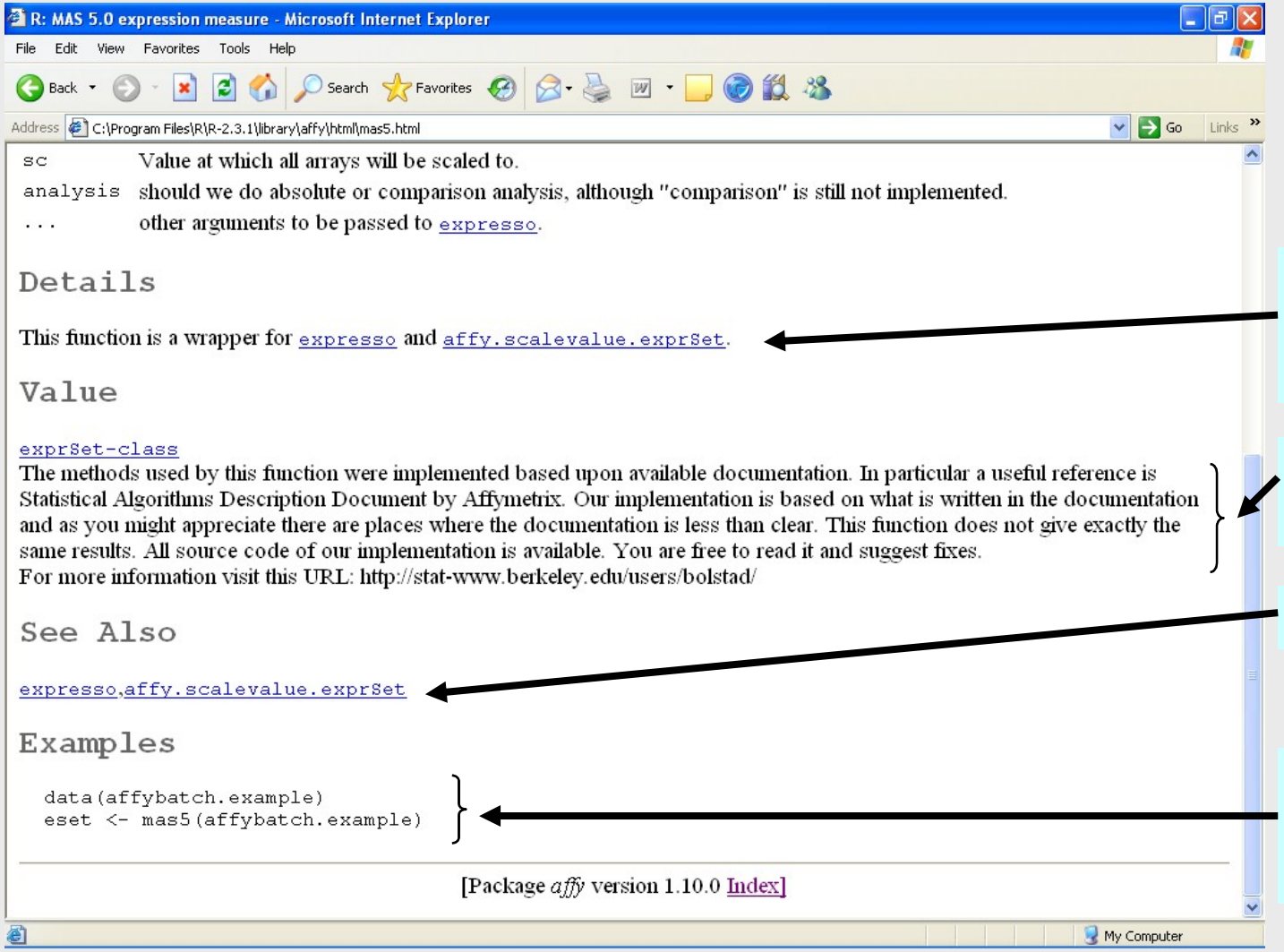
Function {package}

General description

Command and it's argument

Detailed description of arguments

Anatomy of a help file 2/2



Description of how function actually works

What function returns

Related functions

Examples, can be run from R by: `example(mas5)`

Functions or commands in R 1/3

- To use a function in a package, the package needs to be loaded in memory.
- Command for this is library( ), for example:

```
library(affy)
```

- There are three parts in a command:
  - the command
  - brackets
  - Arguments inside brackets (these are not always present)

Functions or commands in R 2/3

- R is case sensitive, so take care when typing in the commands!
  - `library(affy)` works, but `Library(affy)` does not.
- Multiple commands can be written on the same line. Here we first remove missing values from the variable year, and then calculate it's arithmetic average.
  - Writing:
    - na.omit(year)
    - mean(year)
  - Would be the same as
    - mean(na.omit(year))

Functions or commands in R 3/3

- Command can have many arguments. These are always giving inside the brackets.
- Numeric (1, 2, 3…) or logic (T/F) values and names of existing objects are given for the arguments without quotes, but string values, such as file names, are always put inside quotes. For example:
- `mas5(dat3, normalize=T, analysis="absolute")`

Data structures 1/6

- Vector
  - A list of numbers, such as (1,2,3,4,5)
  - R: `a<-c(1,2,3,4,5)`
    - Command c creates a vector that is assigned to object a
- Factor
  - A list of levels, either numeric or string
  - R: `b<-as.factor(a)`
    - Vector a is converted into a factor

Data structures 2/6

- Data frame
  - A table where columns can contain numeric and string values
  - R: `d<-data.frame(a, b)`
- Matrix
  - All columns must contain either numeric or string values, but these can not be combined
  - R: `e<-as.matrix(d)`
    - Data frame d is converted into a matrix e
  - R: `f<-as.data.frame(e)`
    - Matrix e is converted into a dataframe f

Data structures 3/6

- List
  - Contains a list of objects of possibly different types.
  - R: `g<-as.list(d)`
    - Converts a data frame d into a list g
- Class structures
  - Many of the Bioconductor functions create a formal class structure, such as an AffyBatch object.
  - They contain data in slots
  - Slots can be accessed using the @-operator:
    - dat2@cdfName

Data structures 4/6

- Some command need to get, for example, a matrix, and do not accept a data frame. Data frame would give an error message.
- To check the object type:
  - R: `class(d)`
- To check what fields there are in the object:
  - R: `d`
  - R: `str(d)`
- To check the size of the table/matrix:
  - R: `dim(d)`
- To check the length of a factor of vector:
  - R: `length(a)`

Data structures 5/6

- Some data frame related commands:
  - R: `names(d)`
    - Reports column names
  - R: `row.names(d)`
    - Reports row names
- These can also be used for giving the names for the data frame. For example:
  - R: `row.names(d)<-c("a","b","c","d","e")`
    - Letters from a to e are used as the row names for data frame d
    - Note the quotes around the string values!
  - R: `row.names(d)`

Data structures 5/6

- Naming objects:
  - Never use command names as object names!
  - If your unsure whether something is a command name, type to the comman line first. If it gives an error message, you’re safe to use it.
  - Object names can’t start with a number
  - Never use special characters, such as å, ä, or ö in object names.
  - Underscore (_) is not usable, use dot (.) instead:
    - Not acceptable: `good_data`
    - Better way: `good.data`
  - Object names are case sensitive, just like commands

Reading data 1/2

- Command for reading in text files is:

```
read.table(”suomi.txt”, header=T, sep=”\t”)
```

- This examples has one command with three arguments: file name (in quotes), header that tells whether columns have titles, and sep that tells that the file is tab-delimited.

Reading data 2/2

- It is customary to save the data in an object in R. This is done with the assignment operator (<-):

```
dat<-read.table("suomi.txt", header=T, sep="\t")
```

- Here, the data read from file suomi.txt is saved in an object dat in R memory.
- The name of the object is on the left and what is assigned to the object is on the right.
- Command `read.table( )` creates a data frame.

Using data frames

- Individual columns in the data frame can be accessed using one of the following ways:
  - Use its name:
    - `dat$year`
    - dat is the data frame, and year is the header of one of its columns. Dollar sign ($) is an opertaor that accesses that column.
  - Split the data frame into variables, and use the names directly:
    - `attach(dat)`
    - `year`
  - Use subscripts

Subscripts 1/2

- Subscripts are given inside square brackets after the object's name:
  - `dat[,1]`
    - Gets the first column from the object dat
  - `dat[,1]`
    - Gets the first row from the object dat
  - `dat[1,1]`
    - Gets the first row and it's first column from the object dat
- Note that `dat` is now an object, not a command!

Subscripts 2/2

- Subscripts can be used for, e.g., extracting a subset of the data:
  - `dat[which(dat$year>1900),]`
    - Now, this takes a bit of pondering to work out…
    - First we have the object dat, and we are accessing a part of it, because it's name is followed by the square brackets
    - Then we have one command (`which`) that makes an evaluation whether the column year in the object dat has a value higher than 1900.
    - Last the subscript ends with a comma, that tells us that we are accessing rows.
    - So this command takes all the rows that have a year higher 1900 from the object dat that is a data frame.

Writing tables

- To write a table:
  - `write.table(dat, "dat.txt", sep="\t")`
  - Here an object dat is written to a file called dat.txt. This file should be tab-delimited (argument sep).
- To capture what is written on the screen:
  - `sink("output.txt")`
  - `dat`
  - `sink( )`
  - Here, output written on the screen should be written to a file output.txt instead. Contents of the object dat are written to the named file. Last, the file is closed.
  - Note that if you accidentally omit the last command, you'll not be able to see any output on the screen, because output is still redirected to a file!

Quitting R

- Use command `q()` or menu choise File->Exit.
- R asks whether to save workspace image. If you do, all the object currently in R memory are written to a file .Rdata, and all command will be written a file .Rhistory.
- These can be loaded later, and you can continue your work from where you left it.
- Loading can be done after starting R using the manu choises File->Load Workspace and File-> Load History.

In summary 1/2

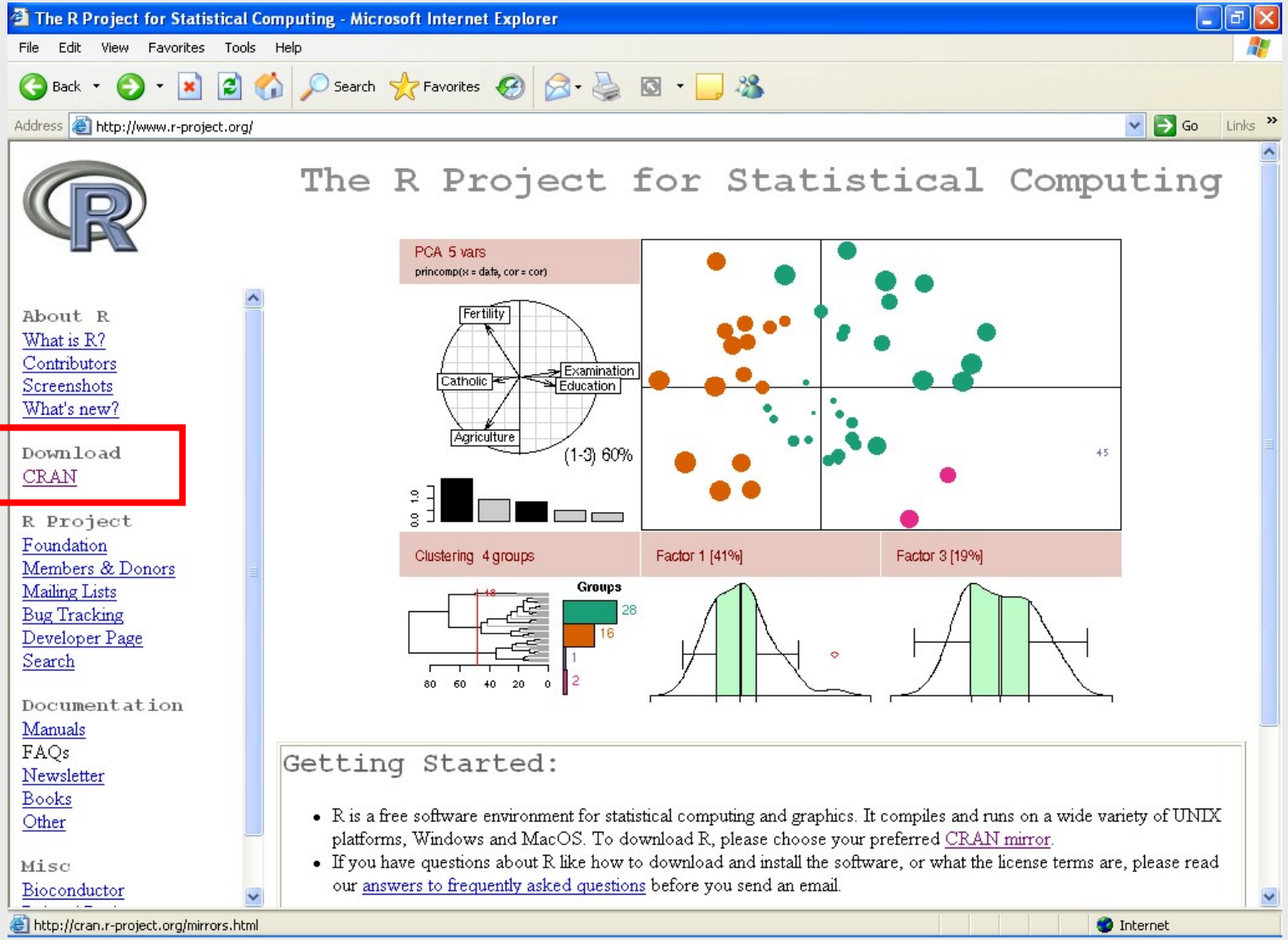
- Commands can be recognized from the brackets ”( )” that follow them. If you calculate how many bracket pairs there are, you’ll be able to identify the number of commands.
  - `pData(dat)<-pd`
- Assignment to an object is denoted by ”<-” or ”->” or ”=”. If you see a notation ”= =”, you’ll looking at a comparison operator.
  - Many other notations can be found from the documentation for the Base package or R.
- Table-like objects are often followed by square brackets ”[ ]”. Square never associate with commands, only objects.
  - `dat[,1]`
- Special characters $ and @ are used denoting individual columns in a data frame or an individual slot in a class type of an object, respectively.
  - `dat$year`
  - `dat2@cdfName`

In summary 2/2

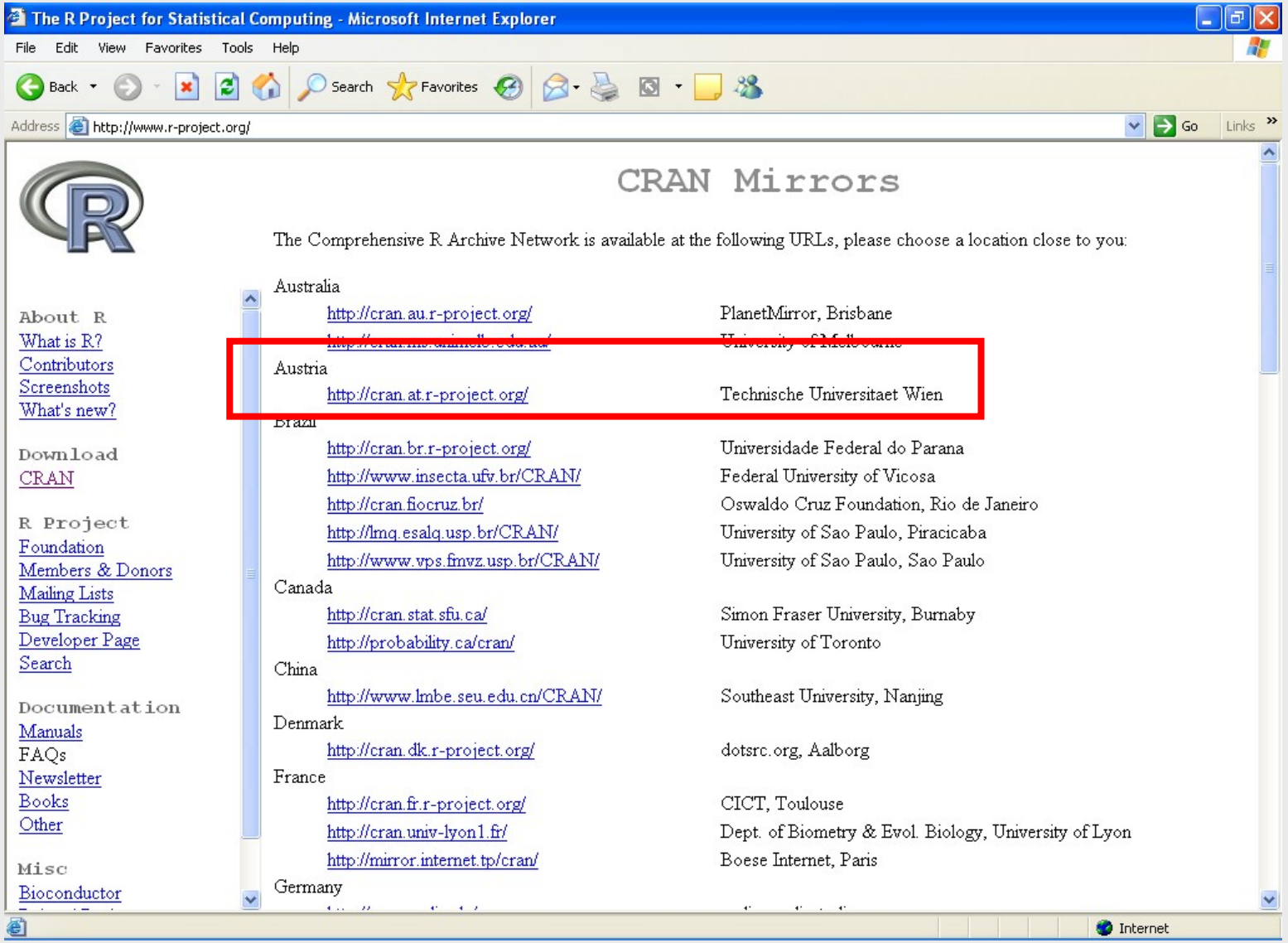
- If you encounter a new command during the exercises, and you'd like to know what it does, please consult the documentation. All R commands are listed nowhere, and the only way to get to know new commands is to read the documentation files, so we'd like you to practise this youself.
- You'll probably see command and notations that were not introduced in this talk. This in intentional, because we thought that these things are best handled on a situational basis. In such cases, please ask for more clarifications if needed.
- If you run into problems, please ask for help from the teachers. That's why we are here!

Installing R

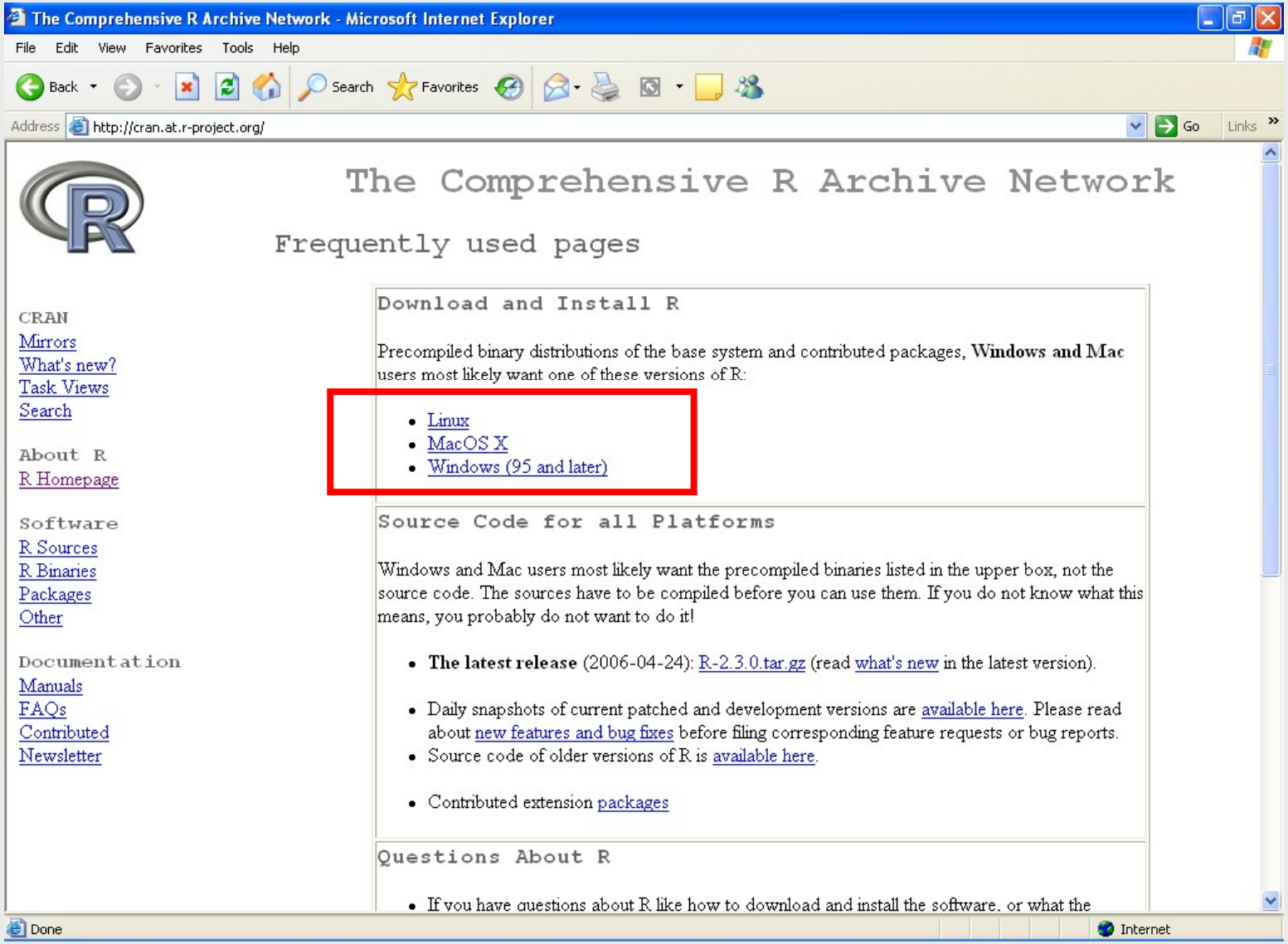
Downloading R



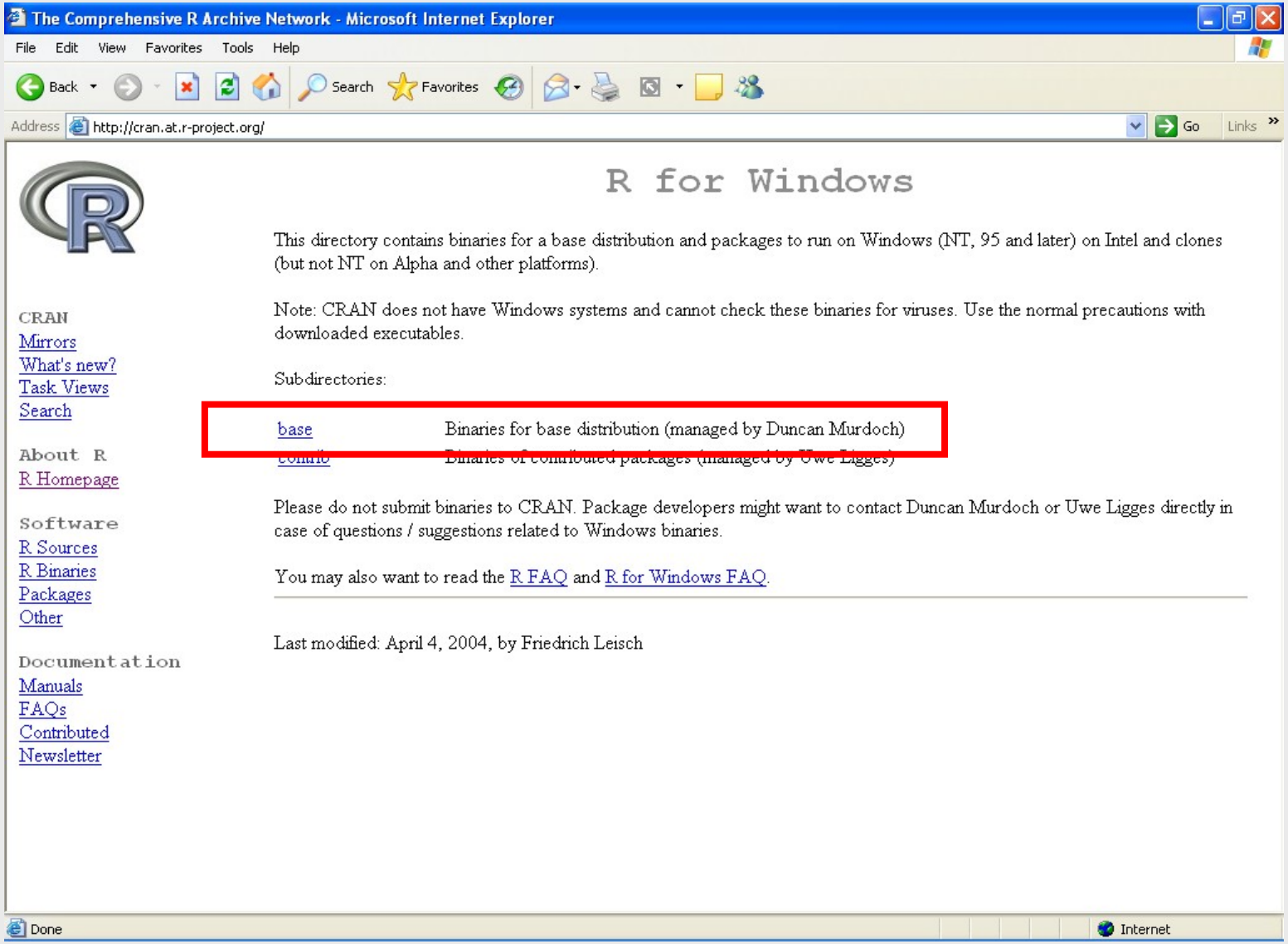
Downloading R



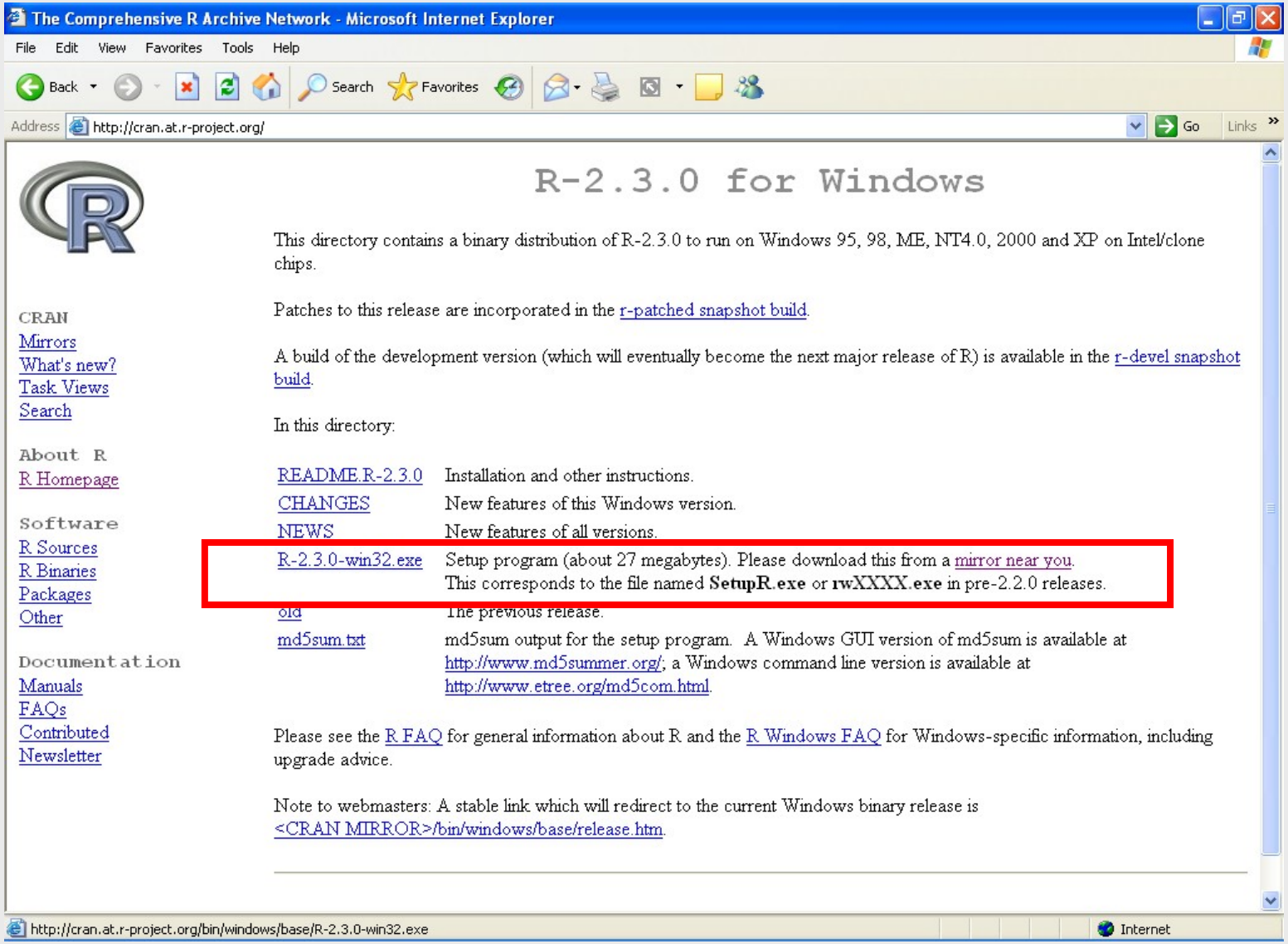
Downloading R



Downloading R



Downloading R



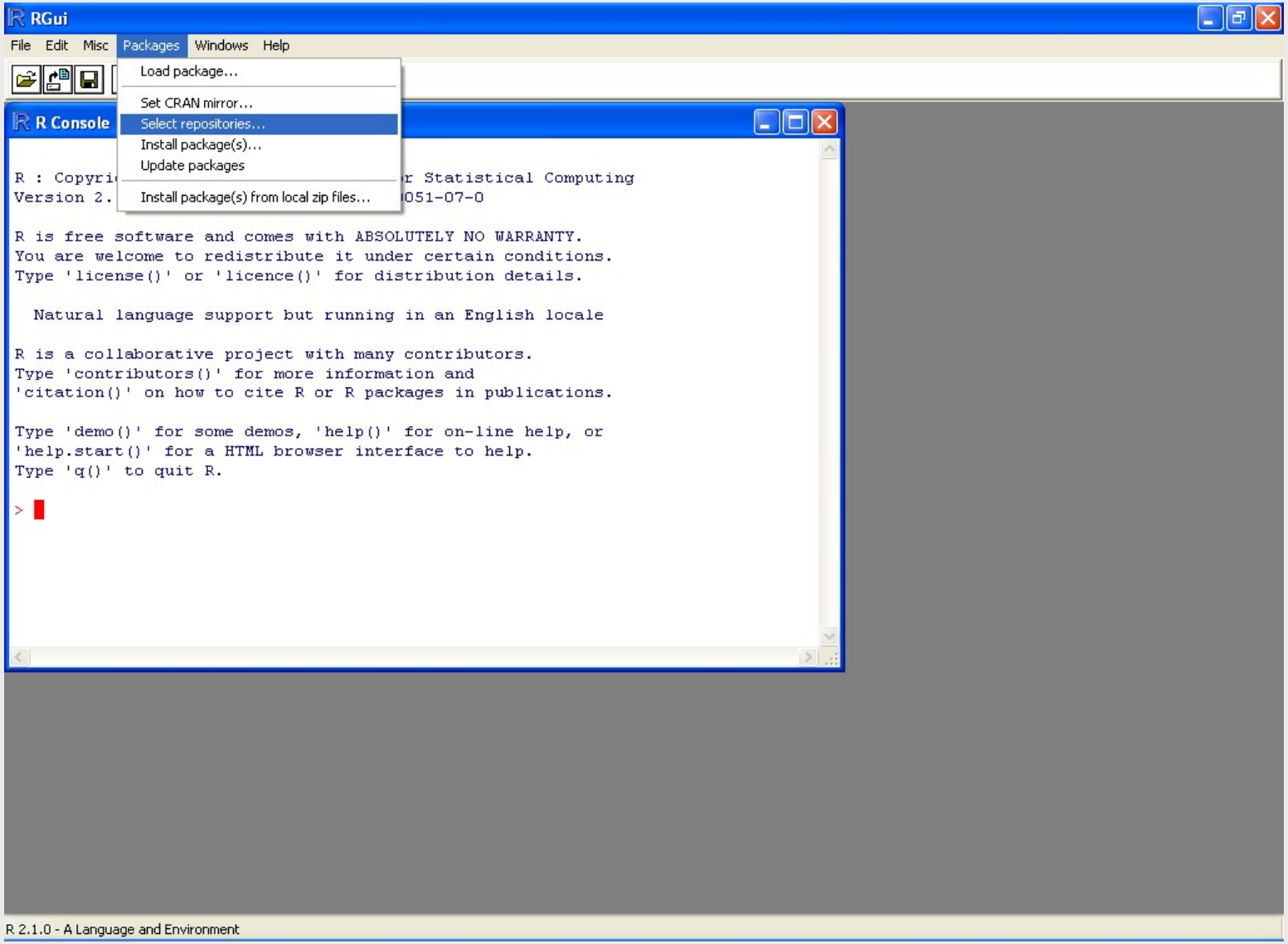
Installing R for Windows

- Execute the R-2.3.0-win32.exe with administrator privileges
- Once the program is installed, run the R program by clicking on its icon

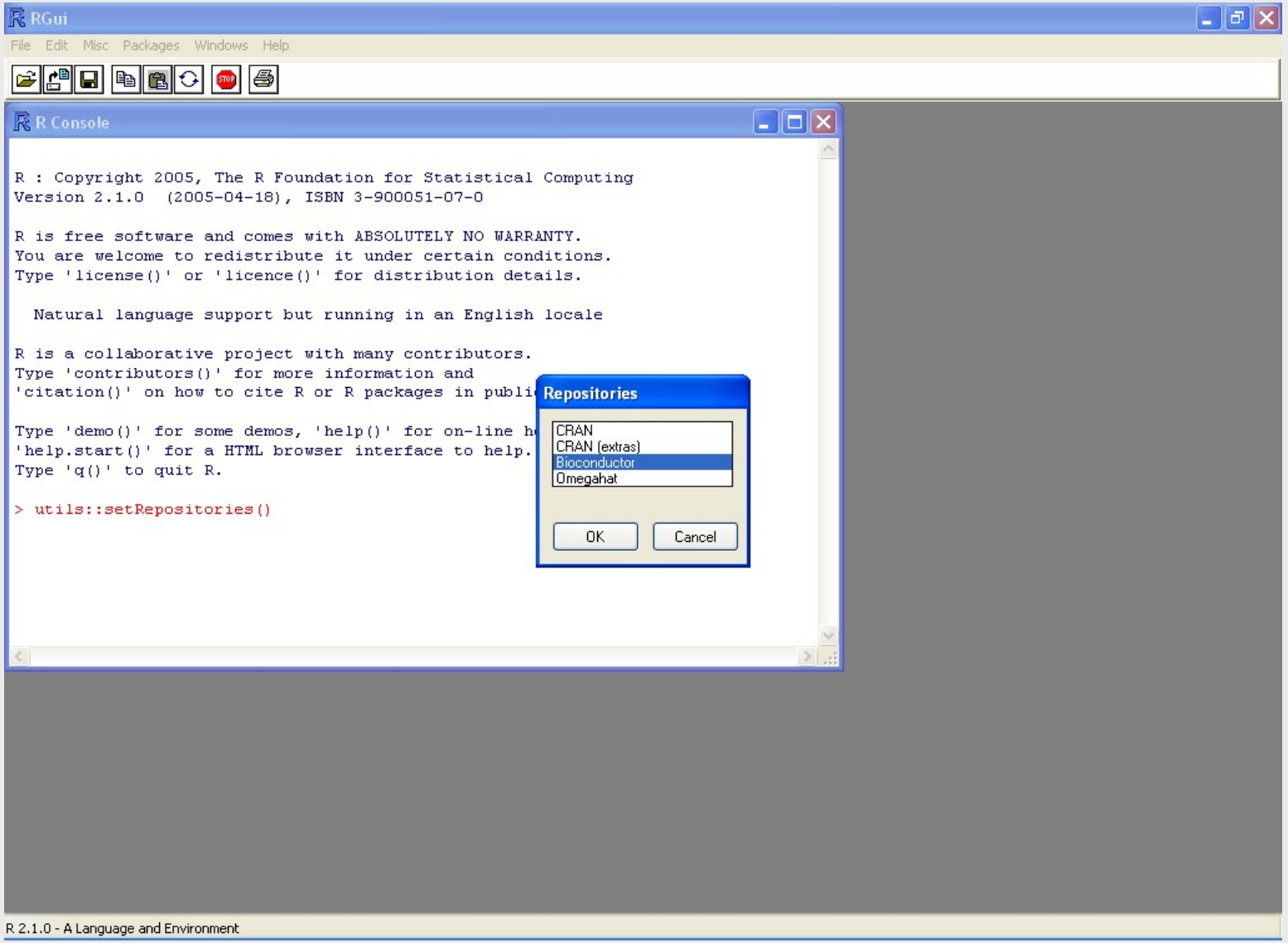
- R 2.2.1 with Bioconductor 1.7.0 is installed on corona.csc.fi, also
- R 2.3.1 is in works

Downloading Bioconductor

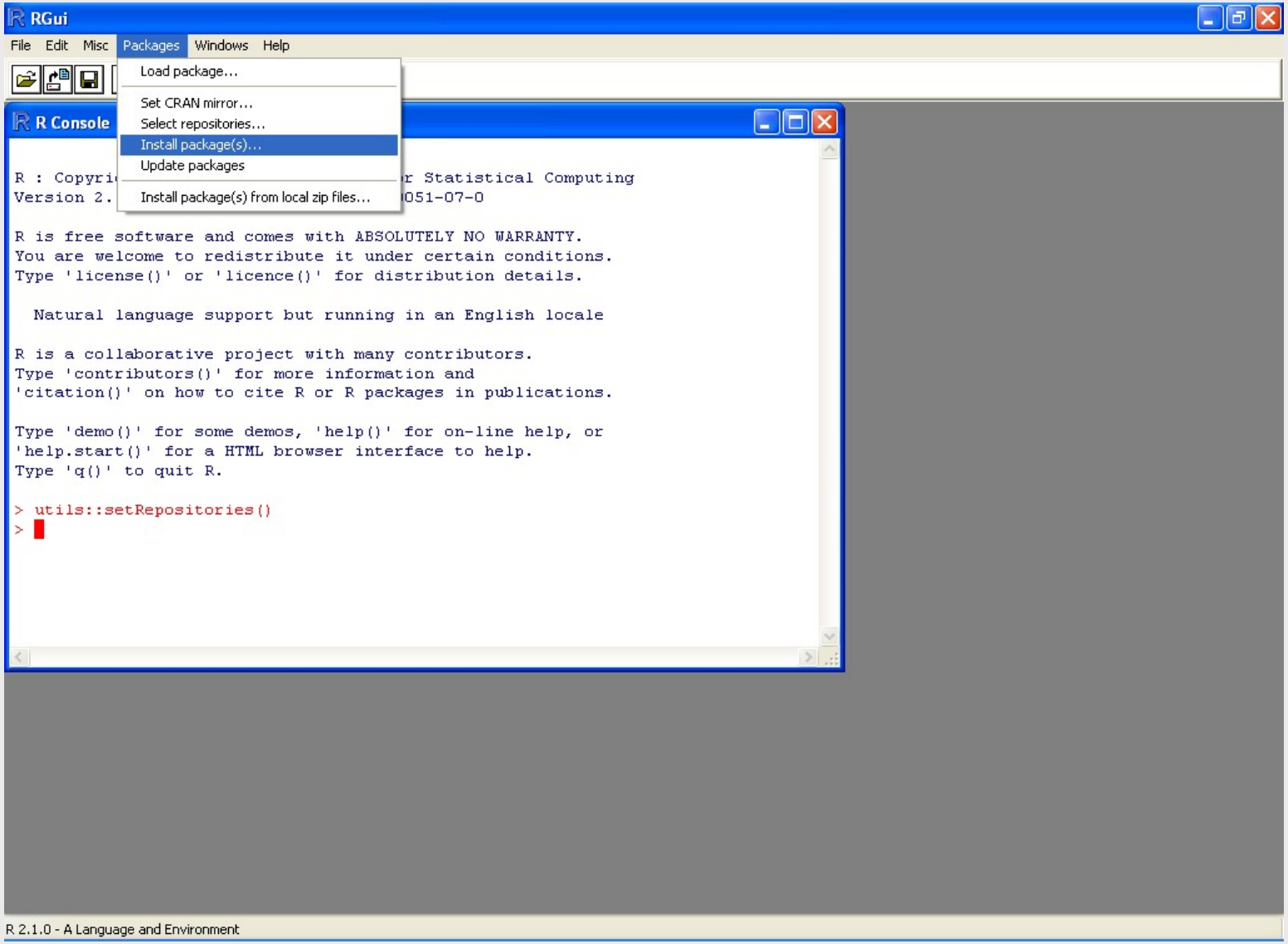
Installing Bioconductor



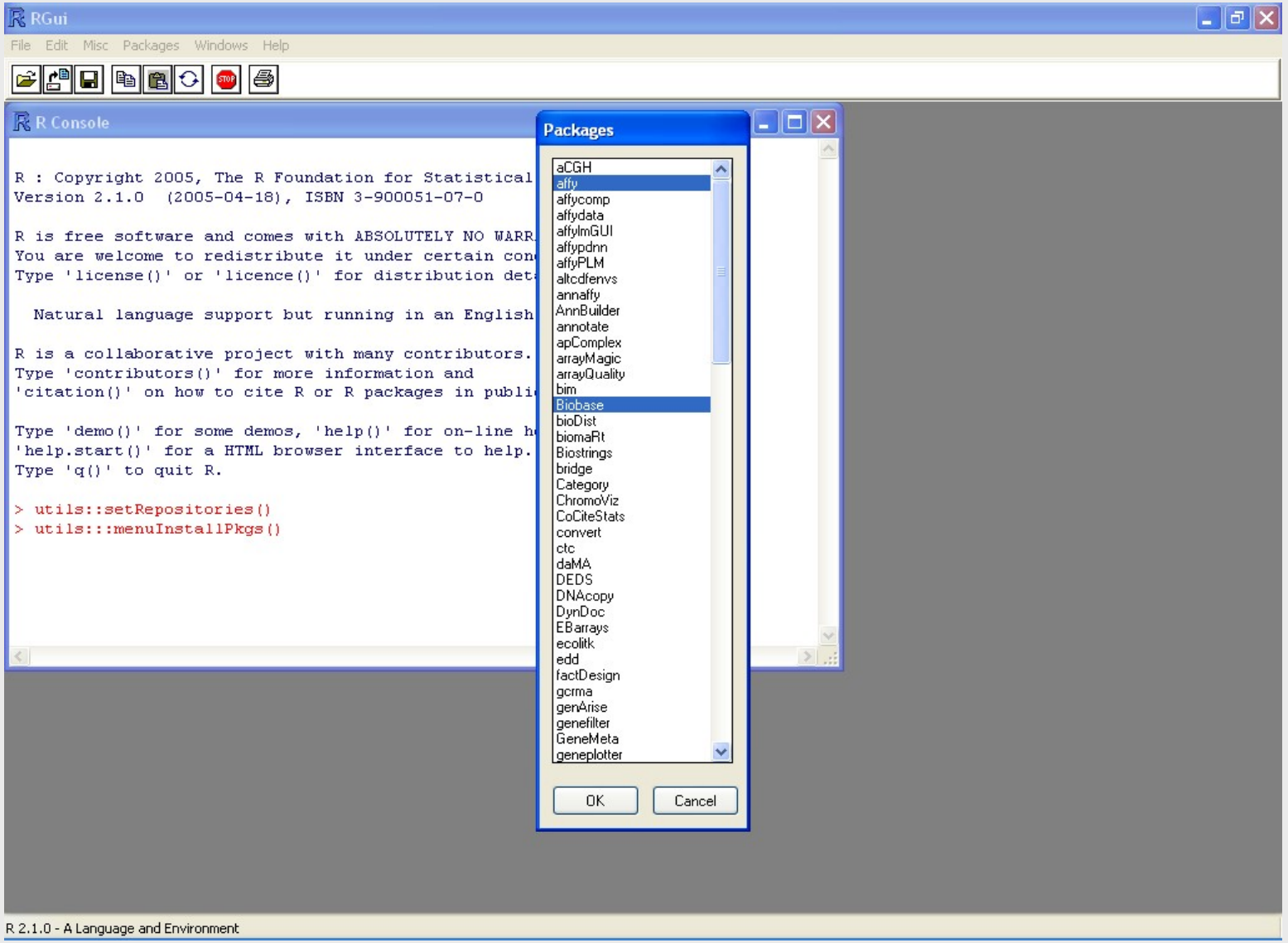
Installing Bioconductor



Installing Bioconductor



Installing Bioconductor



Installing Bioconductor (the best way)

- Alternatively, you can install Bioconductor using a script:

```
source("http://www.bioconductor.org/biocLite.R")
biocLite()

biocLite(c(” "hgu133a", "hgu133acdf",
  "hgu133aprobe", "ygs98", "ygs98cdf",
  "ygs98probe")
```

Linear Models & Descriptive Statistics

- Has functions for all common statistics
- summary() gives lowest, mean, median, first, third quartiles, highest for numeric variables
- stem() gives stem-leaf plots
- table() gives tabulation of categorical variables

Basics

- **Highly Functional**
  - **Everything done through functions**
  - **Strict named arguments**
  - **Abbreviations in arguments OK (e.g. T for TRUE)**
- **Object Oriented**
  - **Everything is an object**
  - **“< -” is an assignment operator**
  - **“X <- 5”: X GETS the value 5**

Data Structures

- Supports virtually any type of data
- Numbers, characters, logicals (TRUE/ FALSE)
- Arrays of virtually unlimited sizes
- Simplest: Vectors and Matrices
- Lists: Can Contain mixed type variables
- Data Frame: Rectangular Data Set

Data Structure in R

	Linear	Rectangular
All Same Type	VECTORS	MATRIX*
Mixed	LIST	DATA FRAME

Reading Data: summary

- Directly using a vector e.g.: x <- c(1,2,3…)
- Using scan and read.table function
- Using matrix function to read data matrices
- Using data.frame to read mixed data
- library(foreign) for data from other programs

Accessing Variables

- edit(<mydataobject>)
- Subscripts essential tools
  - x[1] identifies first element in vector x
  - y[1,] identifies first row in matrix y
  - y[,1] identifies first column in matrix y
- $ sign for lists and data frames
  - myframe$age gets age variable of myframe
  - attach(dataframe) -> extract by variable name

Subset Data

- Using subset function
  - subset() will subset the dataframe
- Subscripting from data frames
  - myframe[,1] gives first column of myframe
- Specifying a vector
  - myframe[1:5] gives first 5 rows of data
- Using logical expressions
  - myframe[myframe[,1], < 5,] gets all rows of the first column that contain values less than 5

Graphics

- Plot an object, like: plot(num.vec)
  - here plots against index numbers
- Plot sends to graphic devices
  - can specify which graphic device you want
    - postscript, gif, jpeg, etc…
    - you can turn them on and off, like: dev.off()
- Two types of plotting
  - high level: graphs drawn with one call
  - Low Level: add additional information to existing graph

Programming in R

- Functions & Operators typically work on entire vectors
- Expressions surrounded by {}
- Codes separated by newlines, “;” not necessary
- You can write your own functions and use them

Statistical Functions in R

- Descriptive Statistics
- Statistical Modeling
  - Regressions: Linear and Logistic
  - Probit, Tobit Models
  - Time Series
- Multivariate Functions
- Inbuilt Packages, contributed packages

Descriptive Statistics

- Has functions for all common statistics
- summary() gives lowest, mean, median, first, third quartiles, highest for numeric variables
- stem() gives stem-leaf plots
- table() gives tabulation of categorical variables

Statistical Modeling

- Over 400 functions
  - lm, glm, aov, ts
- Numerous libraries & packages
  - survival, coxph, tree (recursive trees), nls, …
- Distinction between factors and regressors
  - factors: categorical, regressors: continuous
  - you must specify factors unless they are obvious to R
  - dummy variables for factors created automatically
- Use of data.frame makes life easy

How to model

- Specify your model like this:
  - $y \sim x_i + c_i$, where
  - y = outcome variable, x_i = main explanatory variables, c_i = covariates, + = add terms
  - Operators have special meanings
    - + = add terms, : = interactions, / = nesting, so on…
- Modeling -- object oriented
  - each modeling procedure produces objects

Synopsis of Operators

Operator	Usually means	In Formula means
+ or -	add or subtract	add or remove terms
*	multiplication	main effect and interactions
/	division	main effect and nesting
:	sequence	interaction only
^	exponentiation	limiting interaction depths
%in%	no specific	nesting only

Modeling Example: Regression

carReg <- lm(speed~dist, data=cars)

carReg = becomes an object

to get summary of this regression, we type

summary(carReg)

to get only coefficients, we type

coef(carReg), or carReg$coef

don’t want intercept? add 0, so

carReg <- lm(speed~0+dist, data=cars)

Multivariate Techniques

- Several Libraries available
  - mva, hmisc, glm,
  - MASS: discriminant analysis and multidim scaling
- Econometrics packages
  - dse (multivariate time series, state-space models), ineq: for measuring inequality, poverty estimation, its: for irregular time series, sem: structural equation modeling, and so on…

[http://www.mayin.org/ajayshah/]

Summarizing…

- Effective data handling and storage
- large, coherent set of tools for data analysis
- Good graphical facilities and display
  - on screen
  - on paper
- well-developed, simple, effective programming