Introduction to R

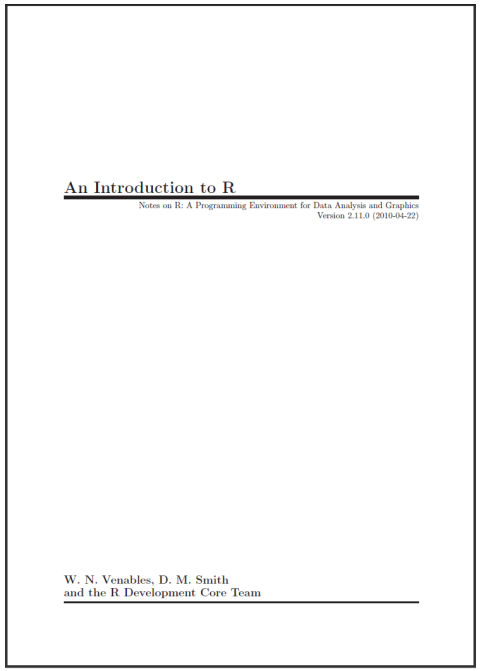
Guy Yollin
Principal Consultant, r-programming.org
Affiliate Instructor, University of Washington

Outline

Outline

Essential web resources

- An Introduction to R
  - W.N. Venables, D.M. Smith
  - R Development Core Team

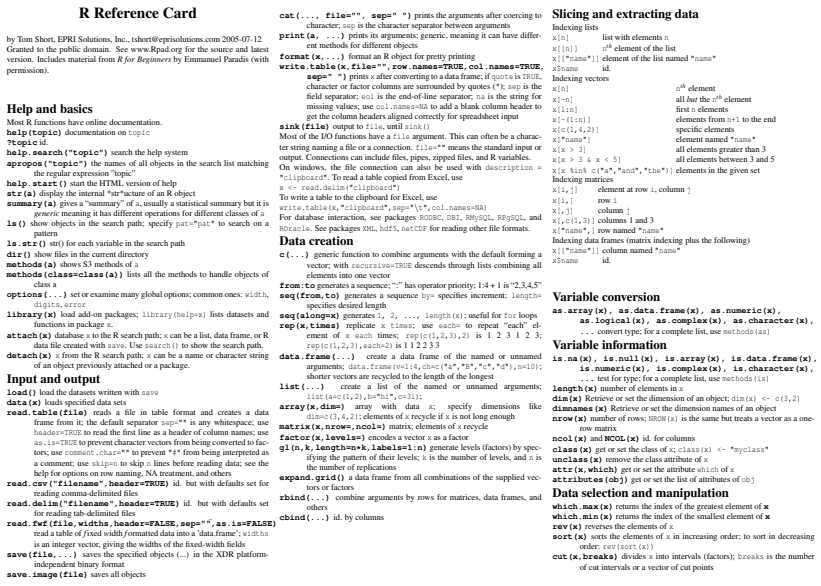


An Introduction to R

Notes on R: A Programming Environment for Data Analysis and Graphics
Version 2.11.0 (2010-04-22)

W. N. Venables, D. M. Smith
and the R Development Core Team

- R Reference Card
  - Tom Short



R Reference Card

by Tom Short, EPRI Solutions, Inc., tshort@eprisolutions.com 2005-07-12
Granted to the public domain. See www.Rpad.org for the source and latest version. Includes material from *R for Beginners* by Emmanuel Paradis (with permission).

Help and basics

Most R functions have online documentation.

help(topic) documentation on `topic`
?topic id.
help.search("topic") search the help system
apropos("topic") the names of all objects in the search list matching the regular expression "topic"
help.start() start the HTML version of help
str(a) display the internal *str*ucture of an R object
summary(a) gives a "summary" of `a`, usually a statistical summary but it is *generic* meaning it has different operations for different classes of `a`
ls() show objects in the search path; specify `pat="pat*"` to search on a pattern
ls.str() str() for each variable in the search path
dir() show files in the current directory
methods(a) shows S3 methods of `a`
methods(class=class(a)) lists all the methods to handle objects of class `a`
options(...) set or examine many global options; common ones: `width`, `digits`, `error`
library(x) load add-on packages; `library(help=x)` lists datasets and functions in package `x`.
attach(x) database `x` to the R search path; `x` can be a list, data frame, or R data file created with `save`. Use `search()` to show the search path.
detach(x) `x` from the R search path; `x` can be a name or character string of an object previously attached or a package.

Input and output

load() load the datasets written with `save`
data(x) loads specified data sets
read.table(file) reads a file in table format and creates a data frame from it; the default separator `sep=""` is any whitespace; use `header=TRUE` to read the first line as a header of column names; use `as.is=TRUE` to prevent character vectors from being converted to factors; use `comment.char=""` to prevent "`#`" from being interpreted as a comment; use `skip=n` to skip `n` lines before reading data; see the help for options on row naming, NA treatment, and others
read.csv("filename",header=TRUE) id. but with defaults set for reading comma-delimited files
read.delim("filename",header=TRUE) id. but with defaults set for reading tab-delimited files
read.fwf(file,widths,header=FALSE,sep="",as.is=FALSE) read a table of *f*ixed width *f*ormatted data into a 'data.frame'; `widths` is an integer vector, giving the widths of the fixed-width fields
save(file,...) saves the specified objects (...) in the XDR platform-independent binary format
save.image(file) saves all objects
cat(..., file="", sep=" ") prints the arguments after coercing to character; `sep` is the character separator between arguments
print(a, ...) prints its arguments; generic, meaning it can have different methods for different objects
format(x,...) format an R object for pretty printing
write.table(x,file="",row.names=TRUE,col.names=TRUE, sep=" ") prints `x` after converting to a data frame; if `quote=TRUE`, character or factor columns are surrounded by quotes ("); `sep` is the field separator; `eol` is the end-of-line separator; `na` is the string for missing values; use `col.names=NA` to add a blank column header to get the column headers aligned correctly for spreadsheet input
sink(file) output to `file`, until `sink()`

Most of the I/O functions have a `file` argument. This can often be a character string naming a file or a connection. `file=""` means the standard input or output. Connections can include files, pipes, zipped files, and R variables.

On windows, the file connection can also be used with `description = "clipboard"`. To read a table copied from Excel, use
`x <- read.delim("clipboard")`
To write a table to the clipboard for Excel, use
`write.table(x,"clipboard",sep="\t",col.names=NA)`

For database interaction, see packages `RODBC`, `DBI`, `RMySQL`, `RPgSQL`, and `ROracle`. See packages `XML`, `hdf5`, `netCDF` for reading other file formats.

Data creation

c(...) generic function to combine arguments with the default forming a vector; with `recursive=TRUE` descends through lists combining all elements into one vector
from:to generates a sequence; ":" has operator priority; `1:4 + 1` is "2,3,4,5"
seq(from,to) generates a sequence `by=` specifies increment; `length=` specifies desired length
seq(along=x) generates `1, 2, ..., length(x)`; useful for `for` loops
rep(x,times) replicate `x times`; use `each=` to repeat "each" element of `x each times`; `rep(c(1,2,3),2)` is `1 2 3 1 2 3`; `rep(c(1,2,3),each=2)` is `1 1 2 2 3 3`
data.frame(...) create a data frame of the named or unnamed arguments; `data.frame(v=1:4,ch=c("a","B","c","d"),n=10)`; shorter vectors are recycled to the length of the longest
list(...) create a list of the named or unnamed arguments; `list(a=c(1,2),b="hi",c=3i)`;
array(x,dim=) array with data `x`; specify dimensions like `dim=c(3,4,2)`; elements of `x` recycle if `x` is not long enough
matrix(x,nrow=,ncol=) matrix; elements of `x` recycle
factor(x,levels=) encodes a vector `x` as a factor
gl(n,k,length=n*k,labels=1:n) generate levels (factors) by specifying the pattern of their levels; `k` is the number of levels, and `n` is the number of replications
expand.grid() a data frame from all combinations of the supplied vectors or factors
rbind(...) combine arguments by rows for matrices, data frames, and others
cbind(...) id. by columns

Slicing and extracting data

Indexing lists

`x[n]`	list with elements `n`
`x[[n]]`	n^{th} element of the list
`x[["name"]]`	element of the list named `"name"`
`x$name`	id.

Indexing vectors

`x[n]`	n^{th} element
`x[-n]`	all *but* the nth element
`x[1:n]`	first `n` elements
`x[-(1:n)]`	elements from `n+1` to the end
`x[c(1,4,2)]`	specific elements
`x["name"]`	element named `"name"`
`x[x > 3]`	all elements greater than 3
`x[x > 3 & x < 5]`	all elements between 3 and 5
`x[x %in% c("a","and","the")]`	elements in the given set

Indexing matrices

`x[i,j]`	element at row `i`, column `j`
`x[i,]`	row `i`
`x[,j]`	column `j`
`x[,c(1,3)]`	columns 1 and 3
`x["name",]`	row named `"name"`

Indexing data frames (matrix indexing plus the following)

`x[["name"]]`	column named `"name"`
`x$name`	id.

Variable conversion

as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use `methods(as)`

Variable information

is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use `methods(is)`
length(x) number of elements in `x`
dim(x) Retrieve or set the dimension of an object; `dim(x) <- c(3,2)`
dimnames(x) Retrieve or set the dimension names of an object
nrow(x) number of rows; `NROW(x)` is the same but treats a vector as a one-row matrix
ncol(x) and **NCOL(x)** id. for columns
class(x) get or set the class of `x`; `class(x) <- "myclass"`
unclass(x) remove the class attribute of `x`
attr(x,which) get or set the attribute `which` of `x`
attributes(obj) get or set the list of attributes of `obj`

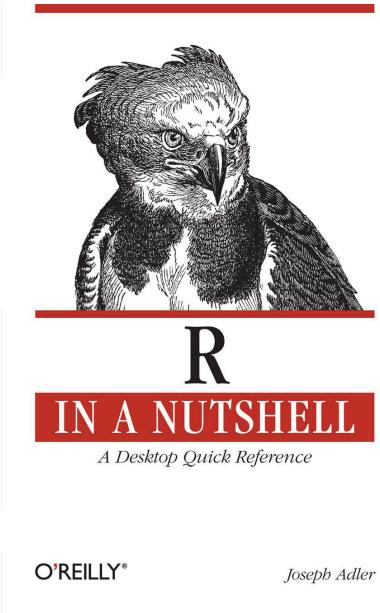
Data selection and manipulation

which.max(x) returns the index of the greatest element of **x**
which.min(x) returns the index of the smallest element of **x**
rev(x) reverses the elements of `x`
sort(x) sorts the elements of `x` in increasing order; to sort in decreasing order: `rev(sort(x))`
cut(x,breaks) divides `x` into intervals (factors); `breaks` is the number of cut intervals or a vector of cut points

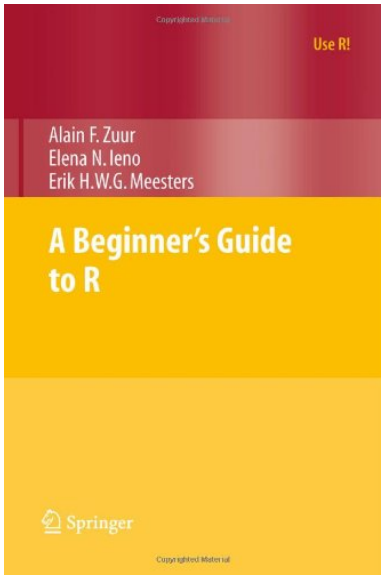
Definitely obtain these PDF files!

Other worthwhile texts

- R in a Nutshell: A Desktop Quick Reference
  - Joseph Adler
  - O'Reilly Media, 2009



- A Beginner's Guide to R
  - Zuur, Ieno, Meesters
  - Springer, 2009



Experience with other statistical computing languages

For those with experience in MATLAB, David Hiebeler has created a MATLAB/R cross reference document:

- `http://www.math.umaine.edu/~hiebeler/comp/matlabR.pdf`

For those with experience in SAS, SPSS, or Stata, Robert Muenchen has written R books for this audience:

- `http://r4stats.com`

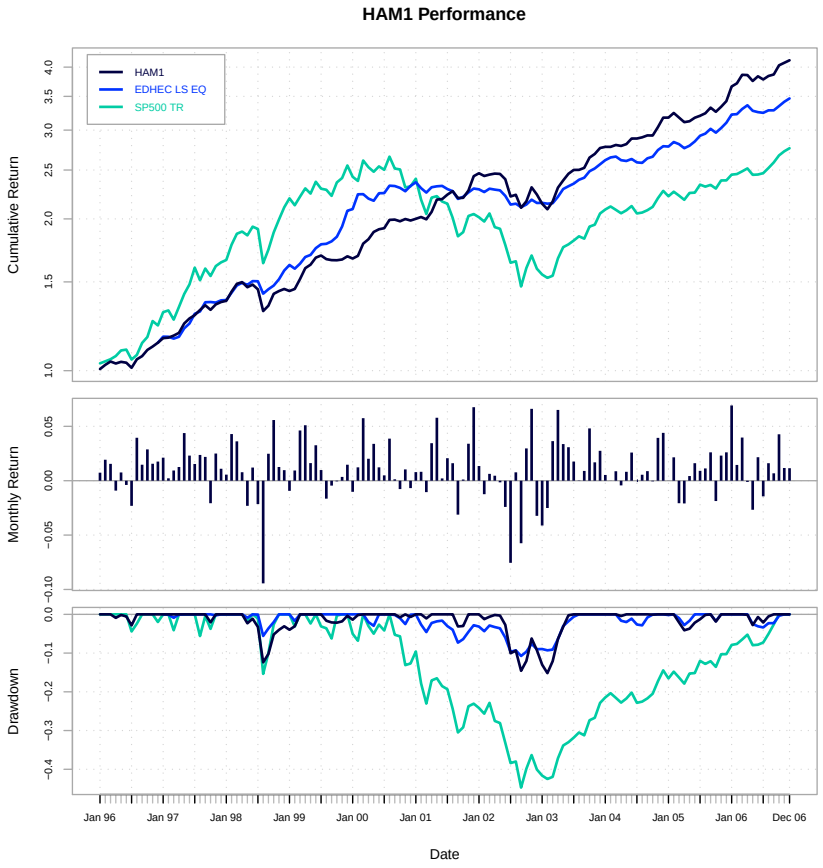
Outline

What is R?

- R is a *language* and *environment* for statistical computing and graphics
- R is based on the *S language* originally developed by John Chambers and colleagues at AT&T Bell Labs in the late 1970s and early 1980s
- R (sometimes called *"GNU S"*) is free open source software licensed under the GNU general public license (GPL 2)
- R development was initiated by Robert Gentleman and Ross Ihaka at the University of Auckland, New Zealand
- R is formally known as The R Project for Statistical Computing
  - `www.r-project.org`

What is R great at?

- Data manipulation
- Data analysis
- Statistical modeling
- Data visualization



Plot from the `PerformanceAnalytics` package

S language implementations

R is the most recent and full-featured implementation of the S language

- Original S - AT & T Bell Labs
- S-PLUS (S plus a GUI)
  - Statistical Sciences, Inc.†
  - Mathsoft, Inc.
  - Insightful, Inc.
  - Tibco, Inc.
- R - The R Project for Statistical Computing

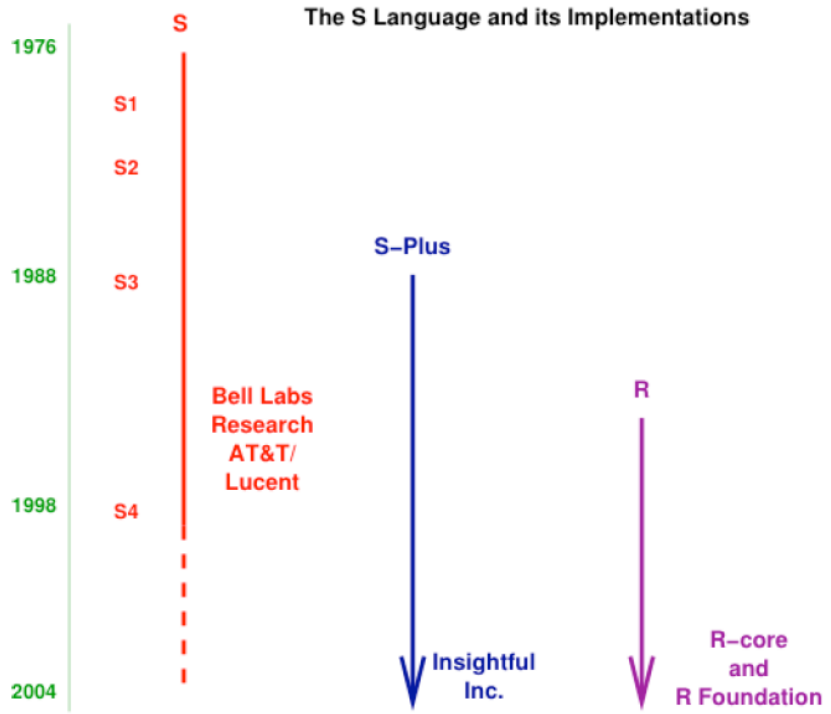
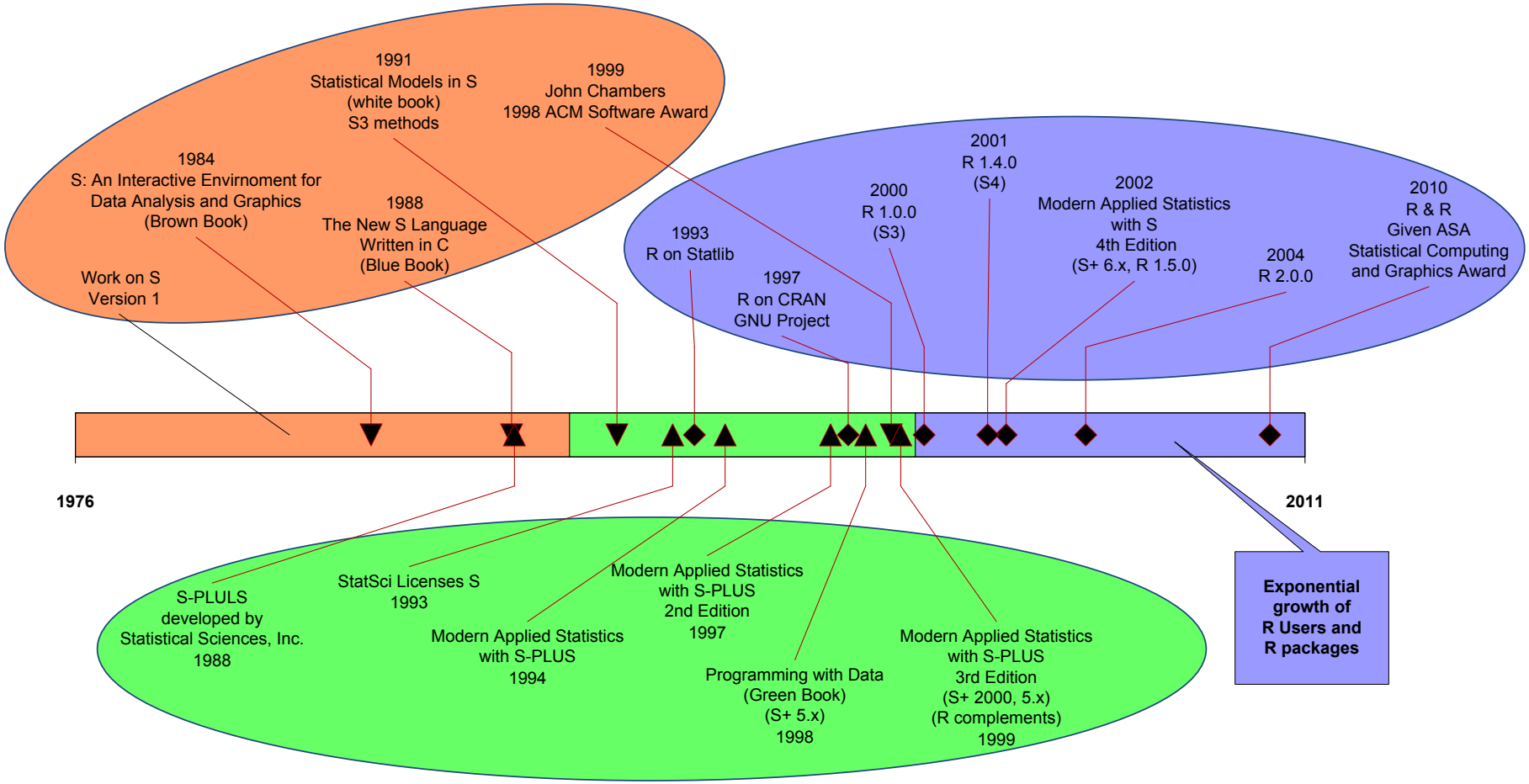


Figure from *The History of S and R, John Chambers, 2006*

†Founded by UW Statistics Professor Doug Martin

R timeline


1991
Statistical Models in S
(white book)
S3 methods
1999
John Chambers
1998 ACM Software Award
1984
S: An Interactive Envirnoment for
Data Analysis and Graphics
(Brown Book)
1988
The New S Language
Written in C
(Blue Book)
Work on S
Version 1
2001
R 1.4.0
(S4)
2000
R 1.0.0
(S3)
2002
Modern Applied Statistics
with S
4th Edition
(S+ 6.x, R 1.5.0)
2010
R & R
Given ASA
Statistical Computing
and Graphics Award
1993
R on Statlib
1997
R on CRAN
GNU Project
2004
R 2.0.0
1976
2011
S-PLULS
developed by
Statistical Sciences, Inc.
1988
StatSci Licenses S
1993
Modern Applied Statistics
with S-PLUS
1994
Modern Applied Statistics
with S-PLUS
2nd Edition
1997
Programming with Data
(Green Book)
(S+ 5.x)
1998
Modern Applied Statistics
with S-PLUS
3rd Edition
(S+ 2000, 5.x)
(R complements)
1999
Exponential
growth of
R Users and
R packages

Recognition of software excellence

Association for Computing Machinery

John Chambers received the 1998 ACM Software System Award

> *Dr. Chambers' work will forever alter the way people analyze, visualize, and manipulate data*

American Statistical Association

Robert Gentleman and Ross Ihaka received the 2009 ASA Statistical Computing and Graphics Award

> *In recognition for their work in initiating the R Project for Statistical Computing*

The R Foundation

The R Foundation is the non-profit organization located in Vienna, Austria which is responsible for developing and maintaining R

- Hold and administer the copyright of R software and documentation
- Support continued development of R
- Organize meetings and conferences related to statistical computing
- Officers

Presidents	Robert Gentleman, Ross Ihaka
Secretary	Friedrich Leisch
Treasurer	Kurt Hornik
At Large	John Chambers
Auditors	Peter Dalgaard, Martin Maechler

The R Core Team

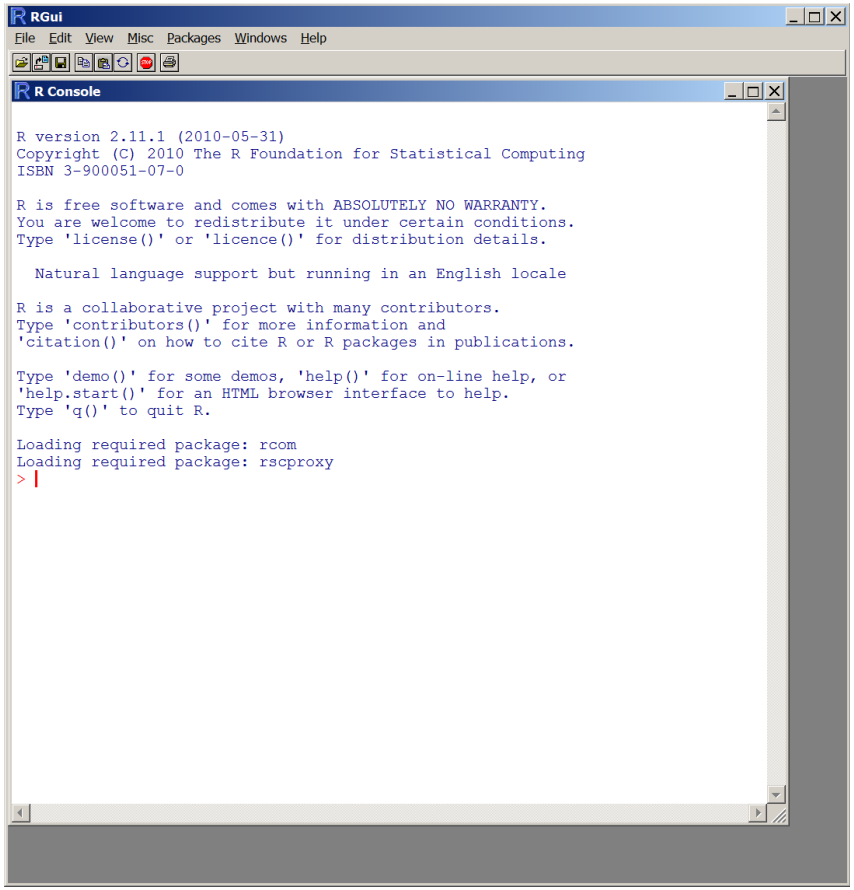
- Douglas Bates – University of Wisconsin Madison
- John Chambers – Stanford University
- Peter Dalgaard – University of Copenhagen
- Seth Falcon – Fred Hutchinson Cancer Research Center
- Robert Gentleman – Genetech
- Kurt Hornik – Vienna University of Economics and Business
- Stefano Iacus – University of Milan
- Ross Ihaka – University of Auckland
- Friedrich Leisch – Ludwig–Maximilians –University Munich
- Thomas Lumley – University of Washington
- Martin Maechler – ETH Swiss Federal Institute of Technology Zurich
- Duncan Murdoch – University of Western Ontario
- Paul Murrell – University of Auckland
- Martyn Plummer – International Agency for Research on Cancer
- Brian Ripley – University of Oxford
- Deepayan Sarkar – Fred Hutchinson Cancer Research Center
- Duncan Temple Lang – University of California Davis
- Luke Tierney – University of Iowa
- Simon Urbanek – AT & T Research Labs

Outline

The R GUI

Running R in Windows†

- Typically run Rgui.exe
- Can also run R.exe from command prompt
- Or run Rterm.exe in batch mode

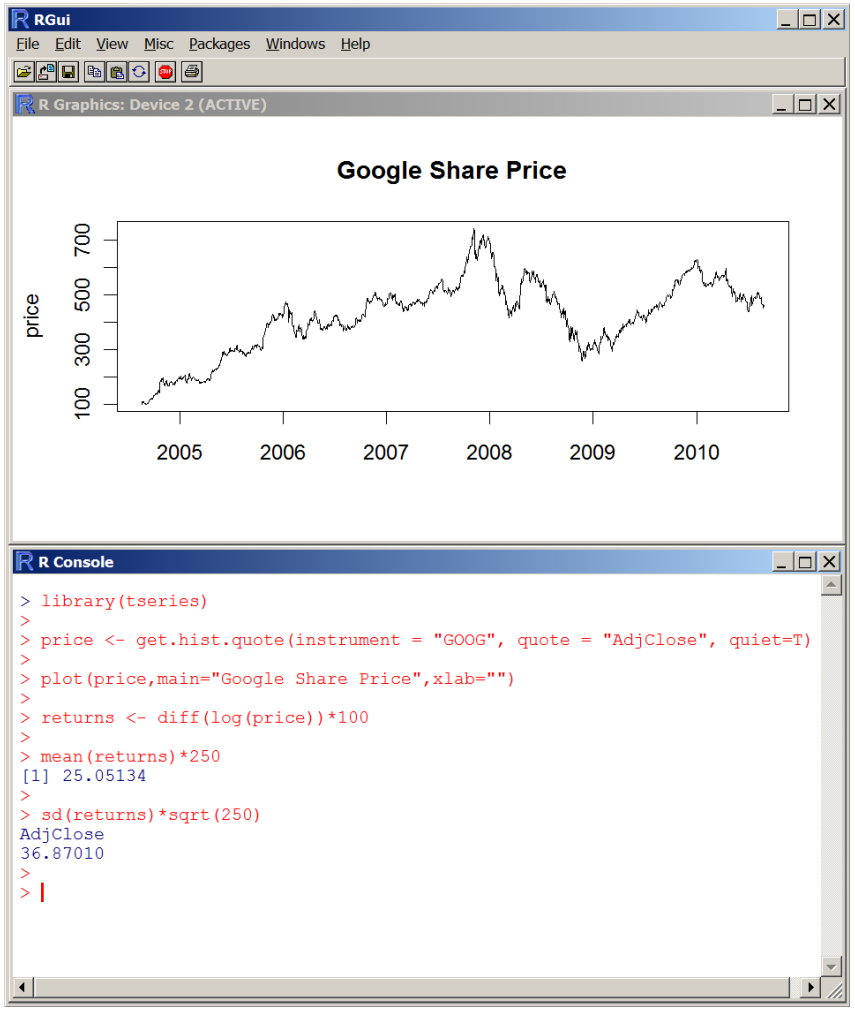


The R GUI on a Windows platform

†see RNUT for info about running R on Linux and Mac

Interactive R session

- R is an *interpreted* language
- The R GUI is an *interactive* command driven *environment*
  - type R commands at the R GUI console
  - Run previously created R scripts (R commands in a text file)



Commands entered interactively into the R console

Assigning values to variables

- Typical variable assignment
  - assignment operator: <-
  - assignment function: `assign`
  - equal sign: =
    - must be used to assign arguments in a function call
- Special purpose assignment
  - global assignment operator: <<-
- Deprecated assignment operator
  - underscore character: _

R Code: Variable assignment

```
> y <- 5
> y

[1] 5

> assign("e",2.7183)
> e

[1] 2.7183

> s = sqrt(2)
> s

[1] 1.414214

> r <- rnorm(n=2)
> r

[1] 0.4296685 0.4046568
```

Object orientation in R

Everything in R is an Object

- Use functions `ls` and `objects` to list all objects in the current workspace

R Code: Listing objects

```
> x <- c(3.1416,2.7183)
> m <- matrix(rnorm(9),nrow=3)
> tab <- data.frame(store=c("downtown","eastside","airport"),sales=c(32,17,24))
> cities <- c("Seattle","Portland","San Francisco")
> ls()

[1] "cities" "e"      "m"      "r"      "s"      "tab"    "x"      "y"
```

Data types

All R objects have a *type* or *storage mode*

- Use function `typeof` to display an object's type
- Common types are:
  - double
  - character
  - list
  - integer

R Code: Object type (storage mode)

```
> x

[1] 3.1416 2.7183

> typeof(x)

[1] "double"

> cities

[1] "Seattle"       "Portland"
[3] "San Francisco"

> typeof(cities)

[1] "character"
```

Object classes

All R objects have a *class*

- Use function `class` to display an object's class
- There are many R classes; basic classes are:
  - numeric
  - character
  - data.frame
  - matrix

R Code: Object class

```
> m

           [,1]       [,2]      [,3]
[1,] -0.6147361 -0.2248133 0.1354078
[2,] -0.7835507  2.3798959 0.8825350
[3,]  1.0156090  1.4605885 0.9470563

> class(m)

[1] "matrix"

> tab

     store sales
1 downtown    32
2 eastside    17
3  airport    24

> class(tab)

[1] "data.frame"
```

Vectors

R is a vector/matrix language

- vectors can easily be created with c, the combine function
- most places where single value can be supplied, a vector can be supplied and R will perform a vectorized operation

R Code: Creating vectors and vector operations

```
> constants <- c(3.1416,2.7183,1.4142,1.6180)
> names(constants) <- c("pi","euler","sqrt2","golden")
> constants

    pi  euler  sqrt2 golden
3.1416 2.7183 1.4142 1.6180

> constants^2

      pi    euler    sqrt2   golden
9.869651 7.389155 1.999962 2.617924

> 10*constants

    pi  euler  sqrt2 golden
31.416 27.183 14.142 16.180
```

Indexing vectors

Vectors indices are placed with square brackets: []

Vectors can be indexed in any of the following ways:

- vector of positive integers
- vector of negative integers
- vector of named items
- logical vector

R Code: Indexing vectors

```
> constants[c(1,3,4)]

    pi  sqrt2 golden
3.1416 1.4142 1.6180

> constants[c(-1,-2)]

 sqrt2 golden
1.4142 1.6180

> constants[c("pi","golden")]

    pi golden
3.1416 1.6180

> constants > 2

    pi  euler  sqrt2 golden
  TRUE   TRUE  FALSE  FALSE

> constants[constants > 2]

    pi  euler
3.1416 2.7183
```

The recycling rule

When 2 vectors of unequal length are involved in an operation, the shorter one is recycled to equal the length of the longer vector

R Code: Illustration of recycling

```
> constants

    pi  euler  sqrt2 golden
3.1416 2.7183 1.4142 1.6180

> constants*2

    pi  euler  sqrt2 golden
6.2832 5.4366 2.8284 3.2360

> constants*c(0,1)

    pi  euler  sqrt2 golden
0.0000 2.7183 0.0000 1.6180

> constants*c(0,1,2)

    pi  euler  sqrt2 golden
0.0000 2.7183 2.8284 0.0000
```

last input generates a warning: longer object length is not a multiple of shorter object length

Sequences

An integer sequence vector can be created with the `:` operator

A general numeric sequence vector can be created with the `seq` function

R Code: `seq` arguments

```
> args(seq.default)

function (from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)
NULL
```

to	starting value
from	ending value
by	increment
len	length of sequence

Sequences

R Code: Creating sequences

```
> 1:5
[1] 1 2 3 4 5
> -5:5
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
> seq(from=0,to=1,len=5)
[1] 0.00 0.25 0.50 0.75 1.00
> seq(from=0,to=20,by=2.5)
[1]  0.0  2.5  5.0  7.5 10.0 12.5 15.0 17.5 20.0
```

Passing arguments to functions

- unnamed arguments are assigned according to their position
- named arguments are assigned according to their name and can be in any position
- partial name matching is performed
- arguments with default values are not required to be passed

R Code: Illustration of flexibility in passing arguments

```
> seq(0,10,2)
[1]  0  2  4  6  8 10
> seq(by=2,0,10)
[1]  0  2  4  6  8 10
> seq(0,10,len=5)
[1]  0.0  2.5  5.0  7.5 10.0
> seq(0,10)
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

The "..." argument

Many functions include in their argument list a ...

R Code: The `plot` function arguments

```
> args(plot.default)

function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
    log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
    ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
    panel.last = NULL, asp = NA, ...)
NULL
```

- This is a mechanism to allow additional arguments to be passed which will subsequently be passed on to a sub-function that the main function will call
- An example of this would be passing graphic parameters (e.g. `lwd=2`) to the `plot` function which will subsequently call and pass these arguments on the `par` function

The rep function

The rep function is used to create (or initialize) vectors

R Code: Examples of rep

```
> rep(0,10)   # initialize a vector
 [1] 0 0 0 0 0 0 0 0 0 0
> rep(1:4, 2) # repeat pattern 2 times
[1] 1 2 3 4 1 2 3 4
> rep(1:4, each = 2)  # repeat each element 2 times
[1] 1 1 2 2 3 3 4 4
> rep(1:4, c(2,1,2,1))
[1] 1 1 2 3 3 4
> rep(1:4, each = 2, len = 10)   # 8 integers plus two recycled 1's.
 [1] 1 1 2 2 3 3 4 4 1 1
> rep(1:4, each = 2, times = 3)  # length 24, 3 complete replications
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

Generic functions

A generic function behaves in a way that is appropriate based on the class of its argument; for example:

- plot
- print
- summary

R Code: Some classes handled by the plot function

```
> methods(plot)[1:15]

 [1] "plot.acf"           "plot.data.frame"    "plot.decomposed.ts"
 [4] "plot.default"       "plot.dendrogram"    "plot.density"
 [7] "plot.ecdf"          "plot.factor"        "plot.formula"
[10] "plot.hclust"        "plot.histogram"     "plot.HoltWinters"
[13] "plot.isoreg"        "plot.lm"            "plot.medpolish"
```

R packages

- All R functions are stored in *packages*
- The standard R distribution includes *core* packages and *recommended* packages:
  - Core R packages
    - base, utils, stats, methods, graphics, grDevices, datasets
  - Recommended packages
    - boot, rpart, foreign, MASS, cluster, Matrix, etc.
  - Additional packages can be downloaded through the R GUI or via the `install.packages` function
- When R is initially loaded, only core R packages are loaded by default
  - Additional packages are loaded via the `library` command
  - Packages datasets are made accessible via the `data` command

Loading packages and data into your R session

The `library` and `data` functions are used to load additional libraries and data into the current R session

R Code: The `library` and `data` function

```
> args(library)

function (package, help, pos = 2, lib.loc = NULL, character.only = FALSE,
    logical.return = FALSE, warn.conflicts = TRUE, quietly = FALSE,
    keep.source = getOption("keep.source.pkgs"), verbose = getOption("verbose"))
NULL

> args(data)

function (..., list = character(), package = NULL, lib.loc = NULL,
    verbose = getOption("verbose"), envir = .GlobalEnv)
NULL

> library(nutshell)
> data(top.bacon.searching.cities)
> top.bacon.searching.cities[1,]

     city rank
1 Seattle  100
```

Installing contributed packages

The `install.packages` function can be used to install contributed packages

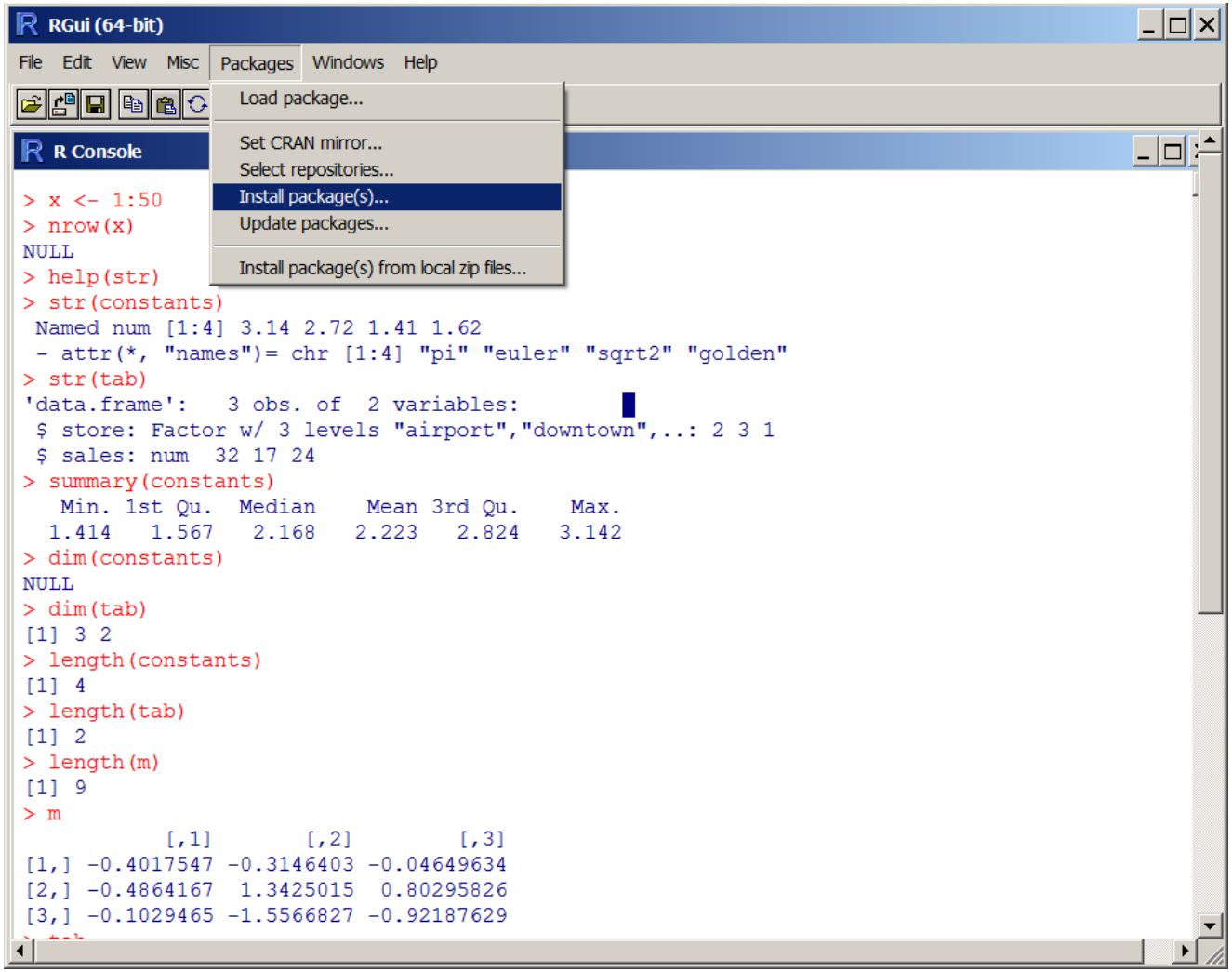
R Code: The `install.packages` function

```
> args(install.packages)

function (pkgs, lib, repos = getOption("repos"), contriburl = contrib.url(repos,
    type), method, available = NULL, destdir = NULL, dependencies = NA,
    type = getOption("pkgType"), configure.args = getOption("configure.args"),
    configure.vars = getOption("configure.vars"), clean = FALSE,
    Ncpus = getOption("Ncpus", 1L), libs_only = FALSE, INSTALL_opts,
    ...)
NULL

> #install.packages("nutshell")
> # or if repos needs to be specified
> #install.packages("nutshell", repos="http://cran.fhcrc.org")
```

see RNUT for more info on installing packages

Installing contributed packages

Packages can also be installed through the R GUI



Packages for basic computational finance

The following R add-on packages are recommended for computational finance:

Package	Description
`zoo`	Time series objects
`tseries`	Time series analysis and computational finance
`PerformanceAnalytics`	Performance and risk analysis
`quantmod`	Quantitative financial modeling framework
`xts`	Extensible time series

Outline

The working directory

Unless overridden by a filename which includes a path, R reads and writes files to the *working directory*

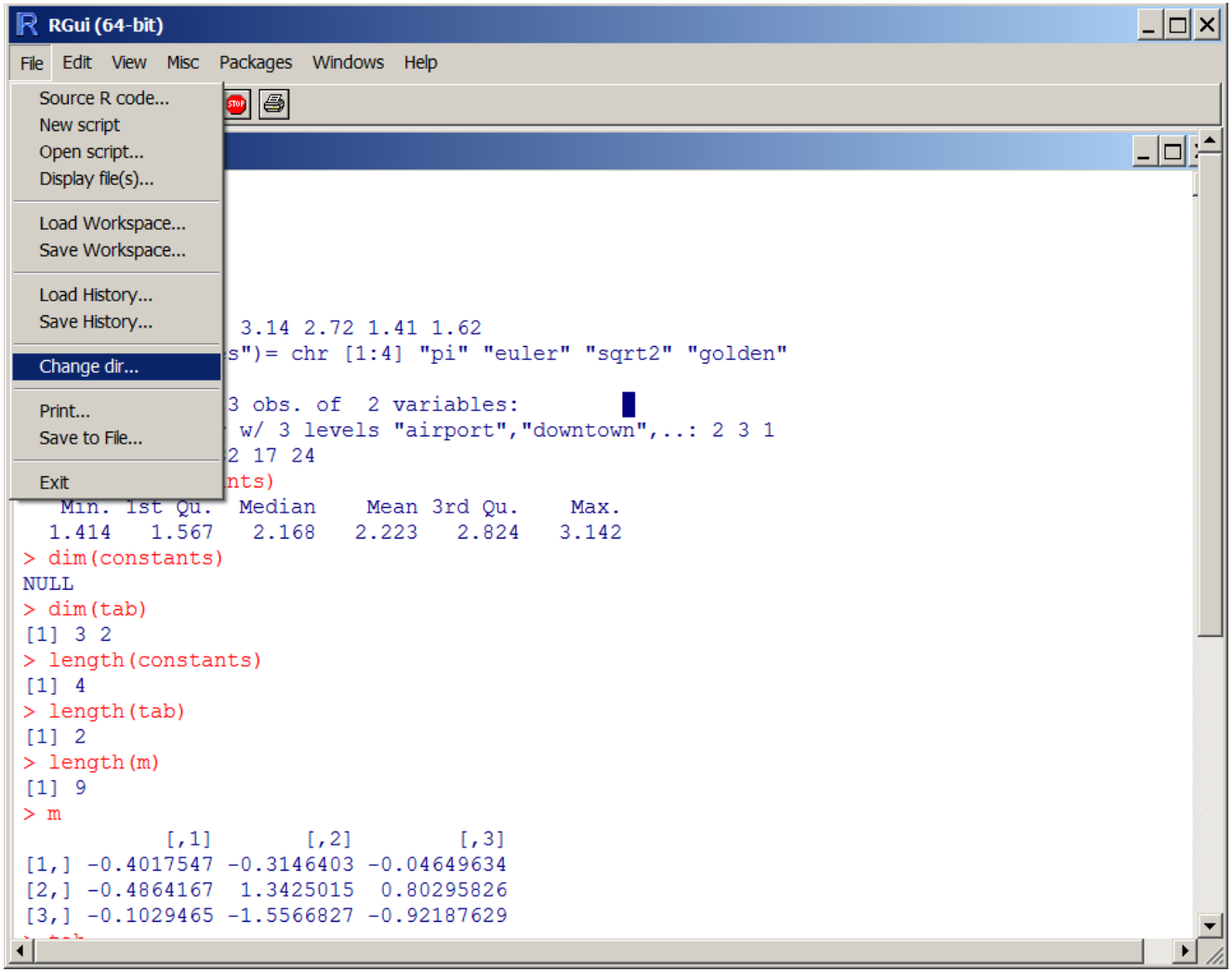
R Code: Getting and setting the working directory

```
> getwd()

[1] "C:/Rprojects/UW/lecture-01"

> setwd("C:\\Rprojects\\PCA")
> getwd()

[1] "C:/Rprojects/PCA"

> setwd("C:/Rprojects/UW/lecture-01")
> getwd()

[1] "C:/Rprojects/UW/lecture-01"
```

- The backslash character "\" in a character string is used to begin an escape sequence, so to use backslash in a string enter it as "\\"
- The forward slash character "/" can also be used as a directory separator on windows systems

The working directory

The working directory can also be changed from the R GUI



The read.table function

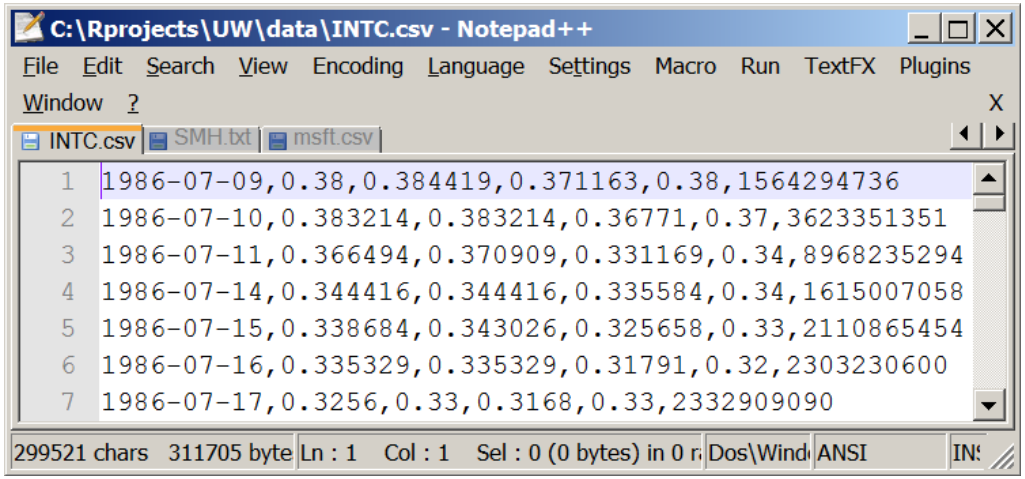
The read.table function is used *extensively* to load data into R

R Code: read.table **arguments**

```
> args(read.table)

function (file, header = FALSE, sep = "", quote = "\"'", dec = ".",
    row.names, col.names, as.is = !stringsAsFactors, na.strings = "NA",
    colClasses = NA, nrows = -1, skip = 0, check.names = TRUE,
    fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
    comment.char = "#", allowEscapes = FALSE, flush = FALSE,
    stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
    encoding = "unknown")
NULL
```

- `file` file name (with path if necessary)
- `header` TRUE/FALSE if there are column names in the file
- `sep` column separation character (e.g. comma or tab)
- `as.is` tells R not to convert strings into factors

Reading a text file



R Code: Read csv file

```
> dat <- read.table("intc.csv",header=FALSE,sep=",",as.is=TRUE)
> dat[1:5,]

          V1       V2       V3       V4   V5         V6
1 1986-07-09 0.380000 0.384419 0.371163 0.38 1564294736
2 1986-07-10 0.383214 0.383214 0.367710 0.37 3623351351
3 1986-07-11 0.366494 0.370909 0.331169 0.34 8968235294
4 1986-07-14 0.344416 0.344416 0.335584 0.34 1615007058
5 1986-07-15 0.338684 0.343026 0.325658 0.33 2110865454
```

The data.frame object

- The `read.table` function returns a `data.frame` object
- A `data.frame` is a 2D matrix-like object where the columns can be of different classes

R Code: The `data.frame` object

```
> dim(dat)

[1] 6092    6

> dat[1:2,1:3]

          V1       V2       V3
1 1986-07-09 0.380000 0.384419
2 1986-07-10 0.383214 0.383214

> typeof(dat)

[1] "list"

> class(dat)

[1] "data.frame"

> class(dat[,1])

[1] "character"

> class(dat[,2])

[1] "numeric"
```

The head and tail functions

R Code: The head and tail functions

```
> args(head.matrix)

function (x, n = 6L, ...)
NULL

> head(dat)

          V1       V2       V3       V4   V5         V6
1 1986-07-09 0.380000 0.384419 0.371163 0.38 1564294736
2 1986-07-10 0.383214 0.383214 0.367710 0.37 3623351351
3 1986-07-11 0.366494 0.370909 0.331169 0.34 8968235294
4 1986-07-14 0.344416 0.344416 0.335584 0.34 1615007058
5 1986-07-15 0.338684 0.343026 0.325658 0.33 2110865454
6 1986-07-16 0.335329 0.335329 0.317910 0.32 2303230600

> tail(dat,3)

             V1    V2    V3    V4    V5        V6
6090 2010-08-30 18.25 18.31 17.94 17.96  73718900
6091 2010-08-31 17.88 17.92 17.60 17.67 111601400
6092 2010-09-01 17.94 18.27 17.89 18.14  73506800
```

Size-related and diagnostic helper functions

R has a number of size related and diagnostic helper functions

Function	Description
`dim`	return dimensions of a multidimensional object
`nrow`	number of rows of a multidimensional object
`ncol`	number of columns of a multidimensional object
`length`	length a vector or list
`head`	display first n rows (elements)
`tail`	display last n rows (elements)
`str`	summarize structure of an object

Indexing data.frames and matrices

R has extremely powerful data manipulation capabilities especially in the area of vector and matrix indexing

- data.frames and matrices can be indexed in any of the following ways
  - vector of positive integers
  - vector of negative integers
  - character vector of columns (row) names
  - a logical vector

R Code: Indexing 2D objects

```
> colnames(dat) <- c("date","open",
 "high","low","close","volume")
> tail(dat[,-1],3)

      open  high   low close    volume
6090 18.25 18.31 17.94 17.96  73718900
6091 17.88 17.92 17.60 17.67 111601400
6092 17.94 18.27 17.89 18.14  73506800

> tail(dat[,c("date","close")],3)

           date close
6090 2010-08-30 17.96
6091 2010-08-31 17.67
6092 2010-09-01 18.14

> dat[dat[,"volume"]>15e9,
   c("date","close","volume")]

           date close      volume
328  1987-10-22  0.61 16717377049
602  1988-11-21  0.52 16116850769
1715 1993-04-19  2.58 21509122170
```

Writing text files

The functions `write.table` and `write` are used to write text files

R Code: `write.table` and `write` arguments

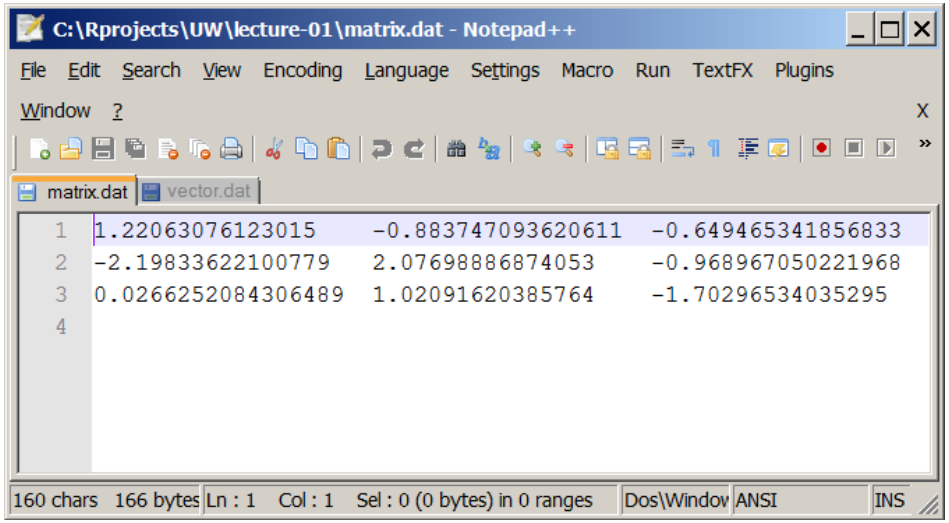
```
> args(write.table)

function (x, file = "", append = FALSE, quote = TRUE, sep = " ",
    eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
    qmethod = c("escape", "double"), fileEncoding = "")
NULL

> args(write)

function (x, file = "data", ncolumns = if (is.character(x)) 1 else 5,
    append = FALSE, sep = " ")
NULL
```

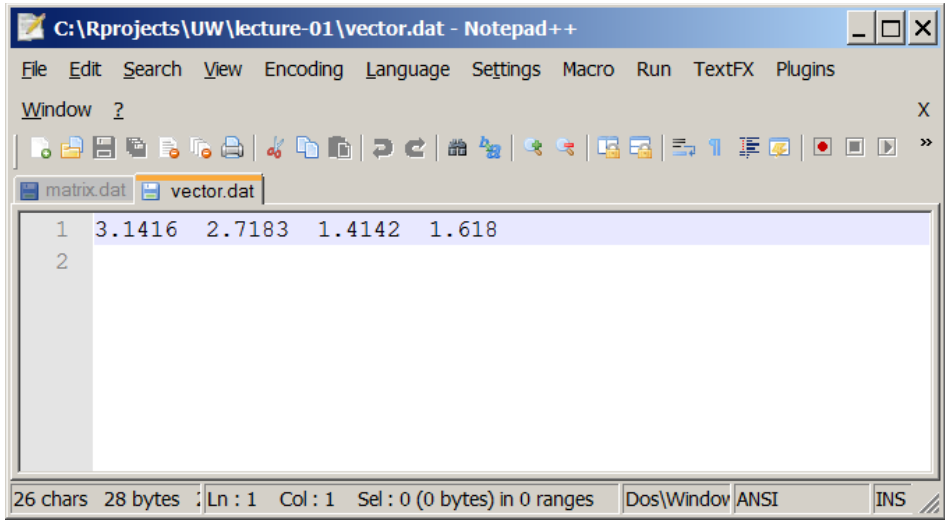
`x`	object to be written (data.frame, matrix, vector)
`file`	file name (with path if necessary)
`sep`	column separation character (e.g. comma or tab)
`row.names`	write row names (T/F)
`col.names`	write col names (T/F)

Writing text files

R Code: Write text files

```
> write(x=constants,file="vector.dat",sep="\t")
> write.table(x=m,file="matrix.dat",sep="\t",row.names=F,col.names=F)
```





The list object

A list object is a container that can hold other objects of different types

R Code: Creating lists

```
> constants <- list(pi=3.1416,euler=2.7183,golden=1.6180)
> class(constants)

[1] "list"

> length(constants)

[1] 3

> constants

$pi
[1] 3.1416

$euler
[1] 2.7183

$golden
[1] 1.618

> diverseList <- list(magic=constants,random=matrix(rnorm(4),ncol=2),
   state=c("WA","OR"))
```

Accessing items in a list

Items in a list can be accessed using `[]`, `[[]]`, or `$` syntax as follows:

- `[]` returns a sublist
  - vector of positive integers
  - vector of named items
  - logical vector
- `[[]]` returns a single element
  - single integer
  - single name
- `$` returns a single element
  - single name

R Code: Indexing lists

```
> constants[2]

$euler
[1] 2.7183

> constants[[2]]

[1] 2.7183

> constants[["pi"]]

[1] 3.1416

> constants$golden

[1] 1.618

> diverseList[[3]][2]

[1] "OR"
```

see RNUT for more info on working with lists

Functions to examine objects and their structures

These functions help to query and unpack an object

`class`	query an objects class
`str`	reports structure of an object
`attributes`	returns list of objects attributes
`attr`	get/set attributes of an object
`names`	gets the names of a list, vector, data.frame, etc.
`dimnames`	gets the row and column names of a data.frame or matrix
`colnames`	column names of a data.frame or matrix
`rownames`	row names of a data.frame or matrix
`dput`	makes an ASCII representation of an object
`unclass`	removes class attribute of an object
`unlist`	converts a list to a vector

see RNUT for more info on these functions

The paste function

The paste function concatenates *(pastes)* strings and numerical values together

- its like a flexible version of sprintf

R Code: The paste function

```
> args(paste)

function (..., sep = " ", collapse = NULL)
NULL

> a <- 2; b <- 2
> paste("We know that: ", a, " + ", b, " = ", a+b, sep ="")

[1] "We know that: 2 + 2 = 4"

> paste("variable",1:5,sep="")

[1] "variable1" "variable2" "variable3" "variable4" "variable5"
```

The apply function

The apply function is an *extremely* useful function that *applies* a given function across the rows and/or columns of a matrix

R Code: The apply function

```
> args(apply)

function (X, MARGIN, FUN, ...)
NULL

> set.seed(1)
> (m <- matrix(sample(9),ncol=3))

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

> apply(m,2,sum)

[1] 17 12 16
```

- There are a number of *apply related* functions; one mark of mastering R is mastering apply related functions

S4 Classes

S4 classes are a more modern implementation of object-oriented programming in R compared to S3 classes

- Data in an S4 class is organized into *slots*; slots can be accessed using:
  - the @ operator: object@name
  - the slot function: slot(object,name)
- Methods for an S4 class can be queried with the showMethods function
  - showMethods(class = "fGARCH")
- Methods can be retrieved/viewed with the getMethod function
  - getMethod("predict","fGARCH")

Outline

Probability distributions

- Random variable

  A *random variable* is a quantity that can take on any of a set of possible values but only one of those values will actually occur

  - *discrete* random variables have a finite number of possible values
  - *continuous* random variables have an infinite number of possible values

- Probability distribution

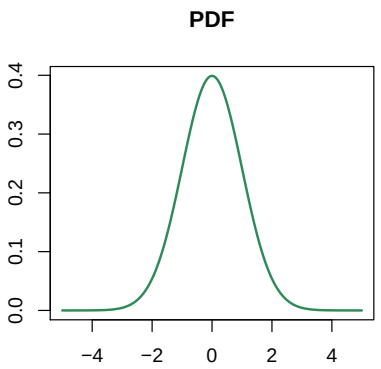
  The set of all possible values of a random variable along with their associated probabilities constitutes a *probability distribution* of the random variable

PDFs and CDFs

- Probability density function (PDF)

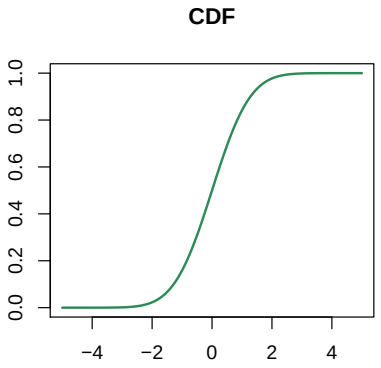
$$Pr(a < Y < b) = \int_a^b f_Y(y)$$

$$\int_{-\infty}^{\infty} f_Y(y)dy = 1$$



- Cumulative distribution function (CDF)

$$F_Y(y) = Pr(Y \leq y) = \int_{-\infty}^{y} f_Y(y)$$



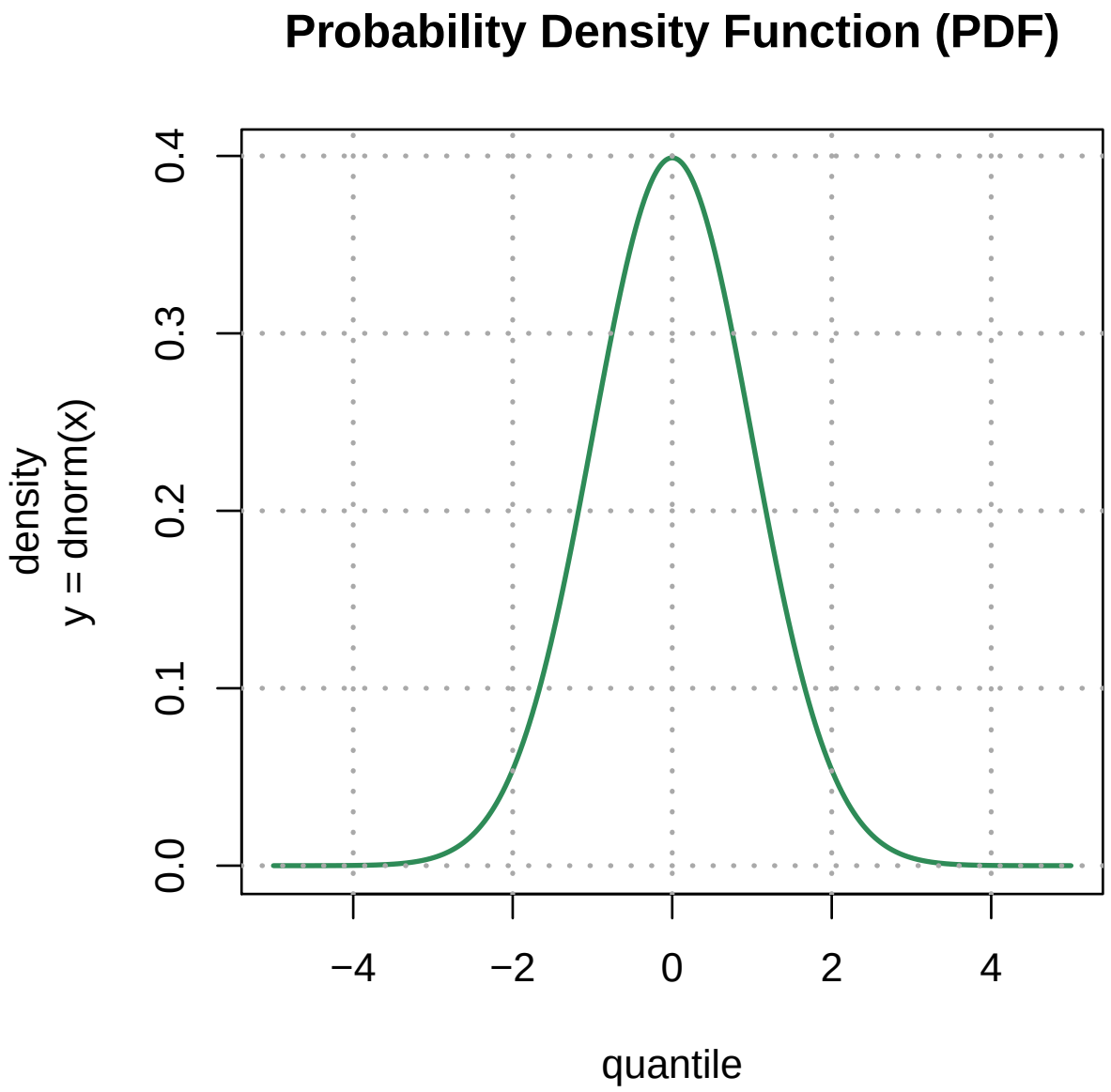
Normal distribution PDF function: dnorm

dnorm computes the normal PDF: $\phi(z)$

R Code: Plot PDF

```
> args(dnorm)

function (x, mean = 0, sd = 1, log = FALSE)
NULL

> x <- seq(from = -5, to = 5, by = 0.01)
> x[1:10]

 [1] -5.00 -4.99 -4.98 -4.97 -4.96 -4.95 -4.94 -4.93 -4.92 -4.91

> y <- dnorm(x)
> y[1:5]

[1] 1.486720e-06 1.562867e-06 1.642751e-06 1.726545e-06 1.814431e-06

> par(mar = par()$mar + c(0,1,0,0))
> plot(x=x,y=y,type="l",col="seagreen",lwd=2,
   xlab="quantile",ylab="density\ny = dnorm(x)")
> grid(col="darkgrey",lwd=2)
> title(main="Probability Density Function (PDF)")
```

Normal distribution PDF function: `dnorm`



Normal distribution CDF functions: pnorm and qnorm

pnorm computes the normal CDF:

$$\Pr(X \leq z) = \Phi(z)$$

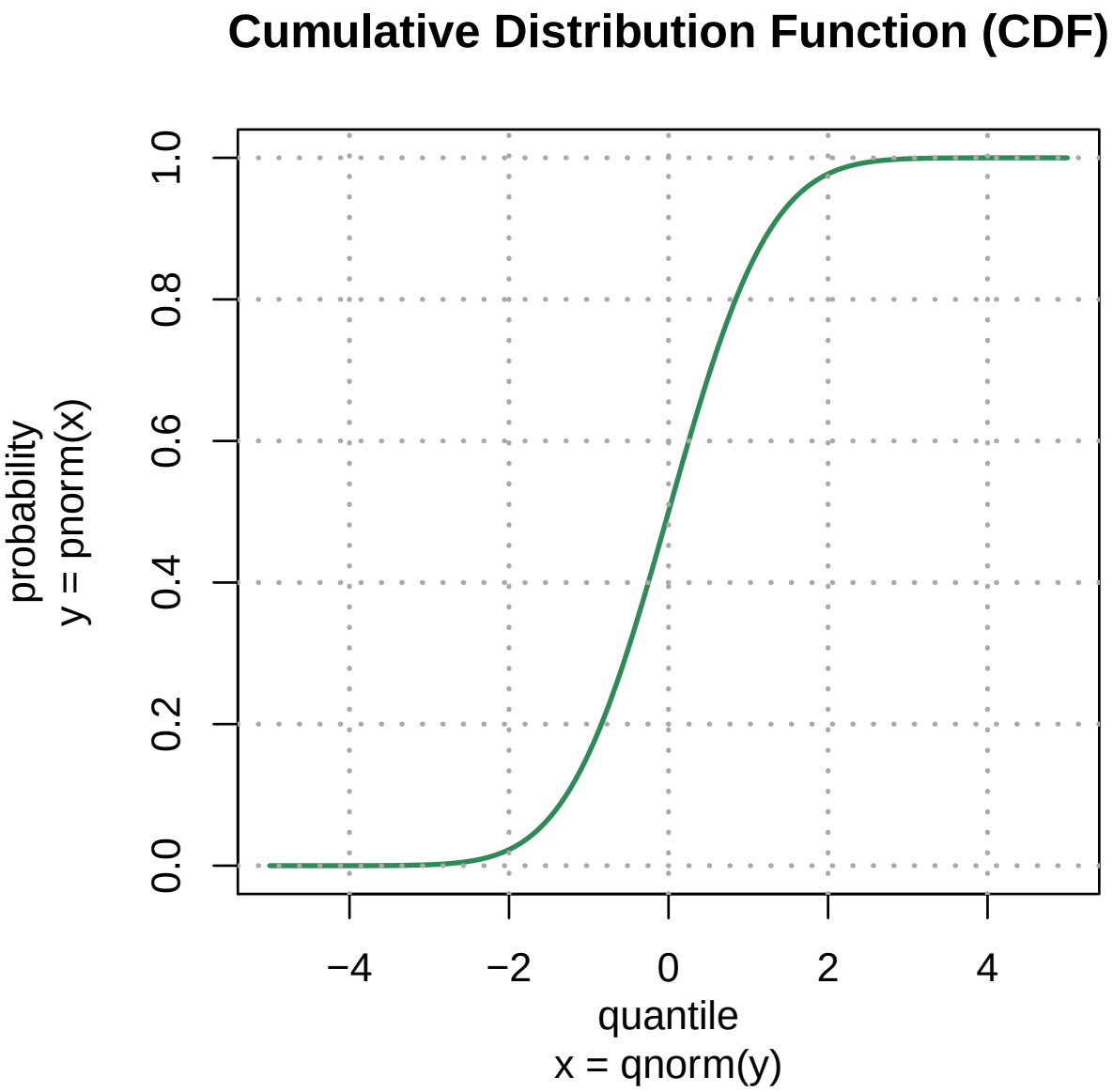
qnorm computes the inverse of the normal CDF (i.e. quantile):

$$z_\alpha = \Phi^{-1}(\alpha)$$

R Code: Plot CDF

```
> args(pnorm)

function (q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
NULL

> args(qnorm)

function (p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
NULL

> y <- pnorm(x)
> par(mar = par()$mar + c(0,1,0,0))
> plot(x=x,y=y,type="l",col="seagreen",lwd=2, xlab="quantile\nx = qnorm(y)",
    ylab="probability\ny = pnorm(x)")  ; grid(col="darkgrey",lwd=2)
> title(main="Cumulative Distribution Function (CDF)")
```

Normal distribution CDF functions: `pnorm` and `qnorm`



Generating normally distributed random numbers

The function `rnorm` generates random numbers from a normal distribution

R Code: `rnorm` arguments

```
> args(rnorm)

function (n, mean = 0, sd = 1)
NULL

> x <- rnorm(150)
> x[1:5]

[1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078

> y <- rnorm(50,sd=3)
> y[1:5]

[1]  1.3505613 -0.0556795 -0.9542051 -2.7880864 -4.4623809
```

`n`	number of observations
`mean`	mean of distribution
`sd`	standard deviation of distribution

Histograms

The generic function `hist` computes a histogram of the given data values

R Code: `hist` arguments

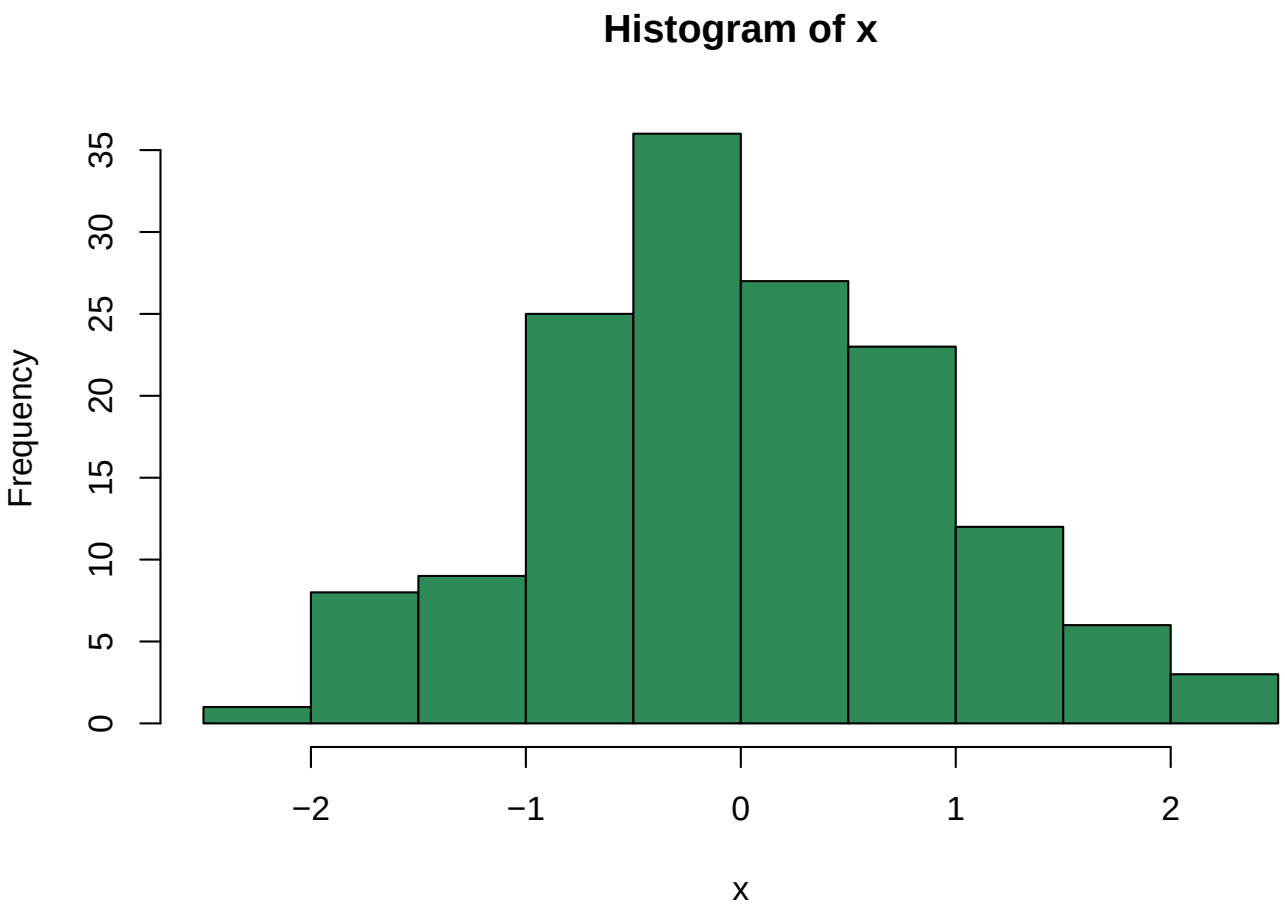
```
> args(hist.default)

function (x, breaks = "Sturges", freq = NULL, probability = !freq,
    include.lowest = TRUE, right = TRUE, density = NULL, angle = 45,
    col = NULL, border = NULL, main = paste("Histogram of", xname),
    xlim = range(breaks), ylim = NULL, xlab = xname, ylab, axes = TRUE,
    plot = TRUE, labels = FALSE, nclass = NULL, warn.unused = TRUE,
    ...)
NULL
```

- `x` vector of histogram data
- `breaks` number of breaks, vector of breaks, name of break algorithm, break function
- `prob` probability densities or counts
- `ylim` y-axis range
- `col` color or bars

Plotting histograms

R Code: Plotting histograms

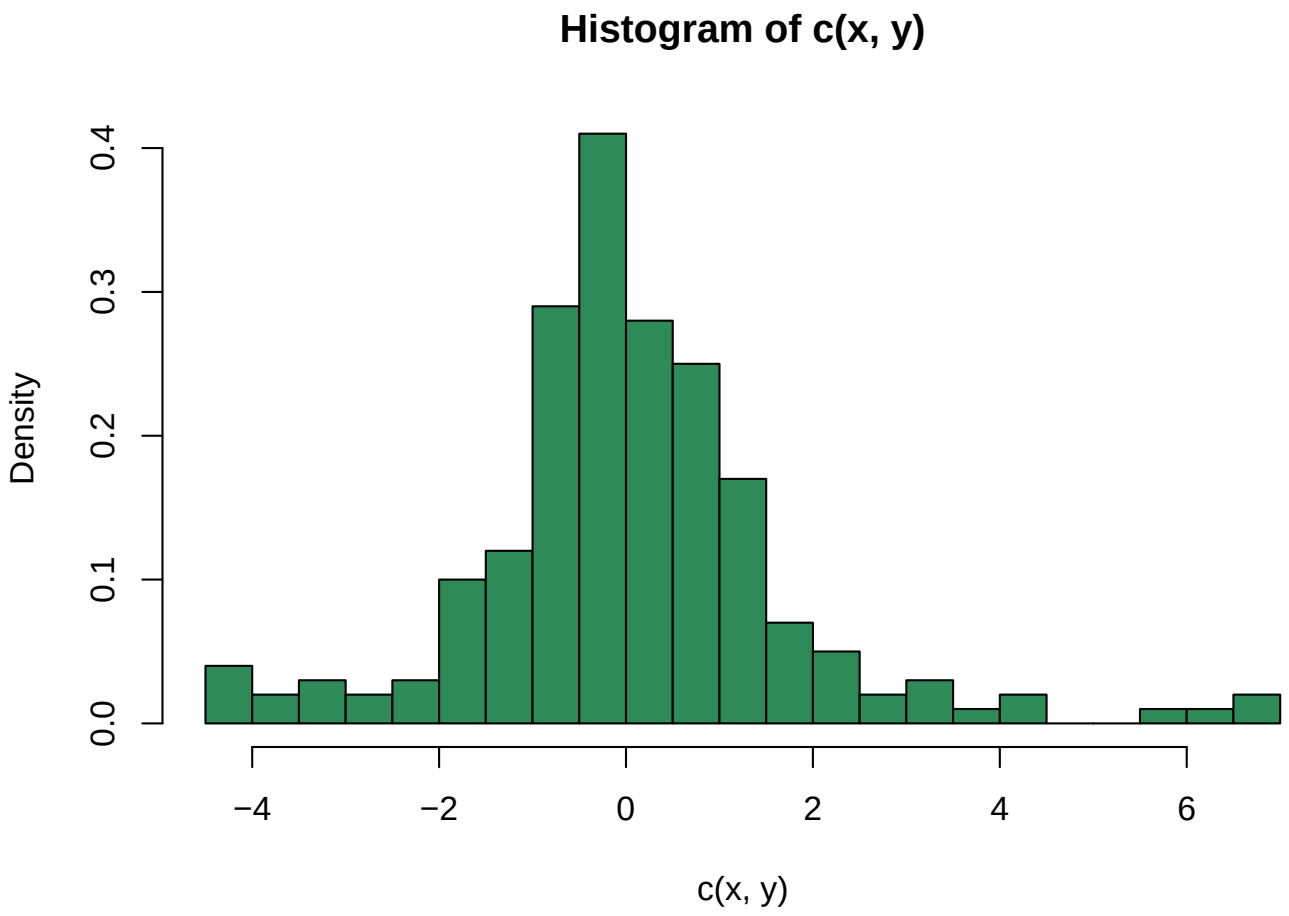
```
> hist(x,col="seagreen")
```



Plotting histograms

R Code: Plotting histograms

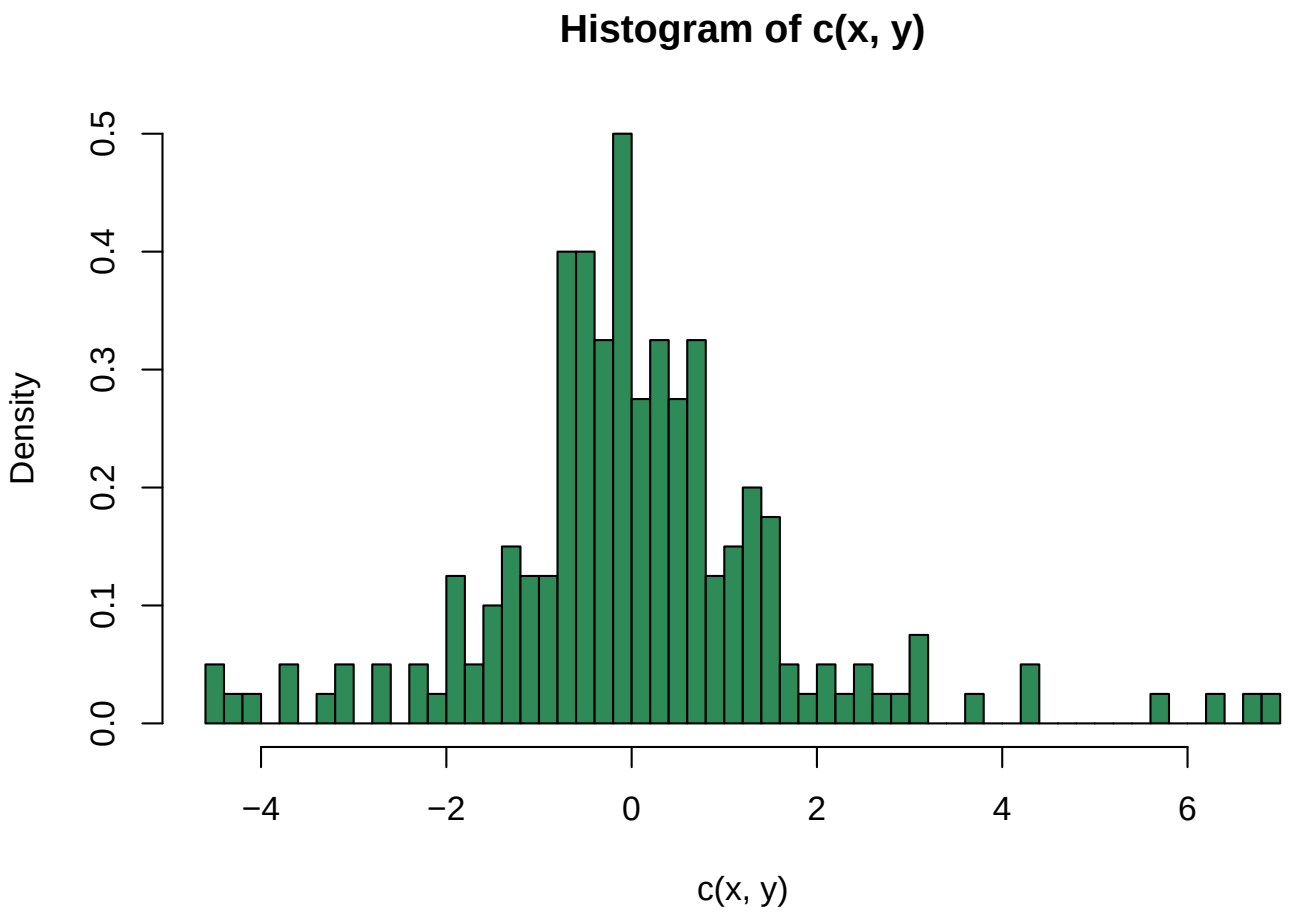
```
> hist(c(x,y),prob=T,breaks="FD",col="seagreen")
```



Plotting histograms

R Code: Plotting histograms

```
> hist(c(x,y),prob=T,breaks=50,col="seagreen")
```



Basic stats functions

Short list of some common statistics and math functions:

`mean`	mean of a vector or matrix
`median`	median of a vector or matrix
`mad`	median absolute deviation of a vector or matrix
`var`	variance of a vector or matrix
`sd`	standard deviation of a vector
`cov`	covariance between vectors
`cor`	correlation between vectors
`diff`	difference between elements in a vector
`log`	log of a vector or matrix
`exp`	exponentiation of a vector or matrix
`abs`	absolute value of a vector or matrix

Outline

Basic plotting functions

Function	Description
`plot`	generic function to plot an R object
`lines`	adds lines to the current plot
`segments`	adds lines line segments between point pairs
`points`	adds points to the current plot
`text`	adds text to the current plot
`abline`	adds straight lines to the current plot
`curve`	plot a function over a range
`legend`	adds a legend to the current plot
`matplot`	plot all columns of a matrix
`par`	sets graphics parameters

The plot function

The plot function is a generic function for plotting of R objects

R Code: plot arguments

```
> args(plot.default)

function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
    log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
    ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
    panel.last = NULL, asp = NA, ...)
NULL
```

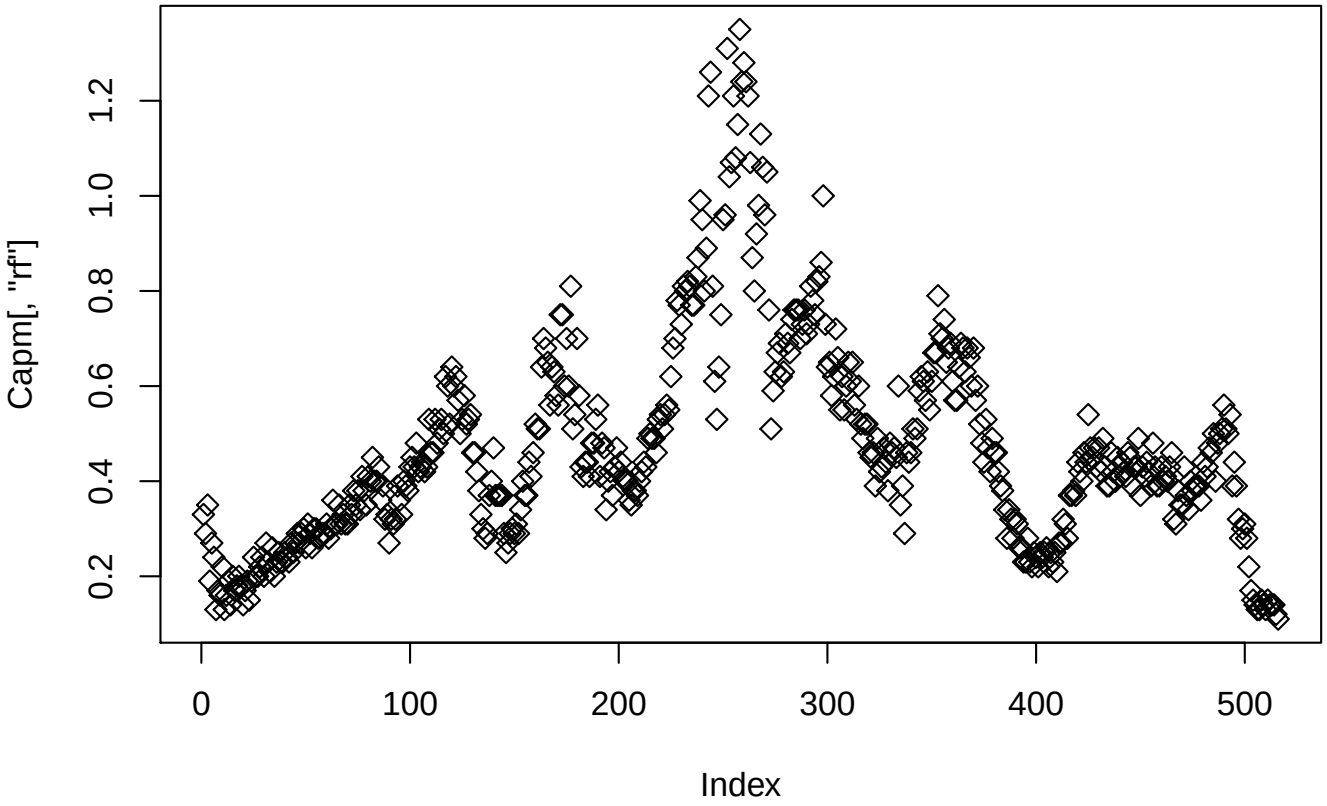
x	vector to be plotted (or index if y given)
y	vector to be plotted
xlim/ylim	x & y limited
xlab/ylab	x & y axis labels
main	plot title (can be done with title function
type	"p" = points (default), "l" = lines, "h" = bars, "n" = no plot
col	color or bars
asp	control the aspect ratio

The plot function

R Code: Plot with defaults

```
> library(Ecdat)
> data(Capm)
> plot(Capm[,"rf"],pch=5)
```

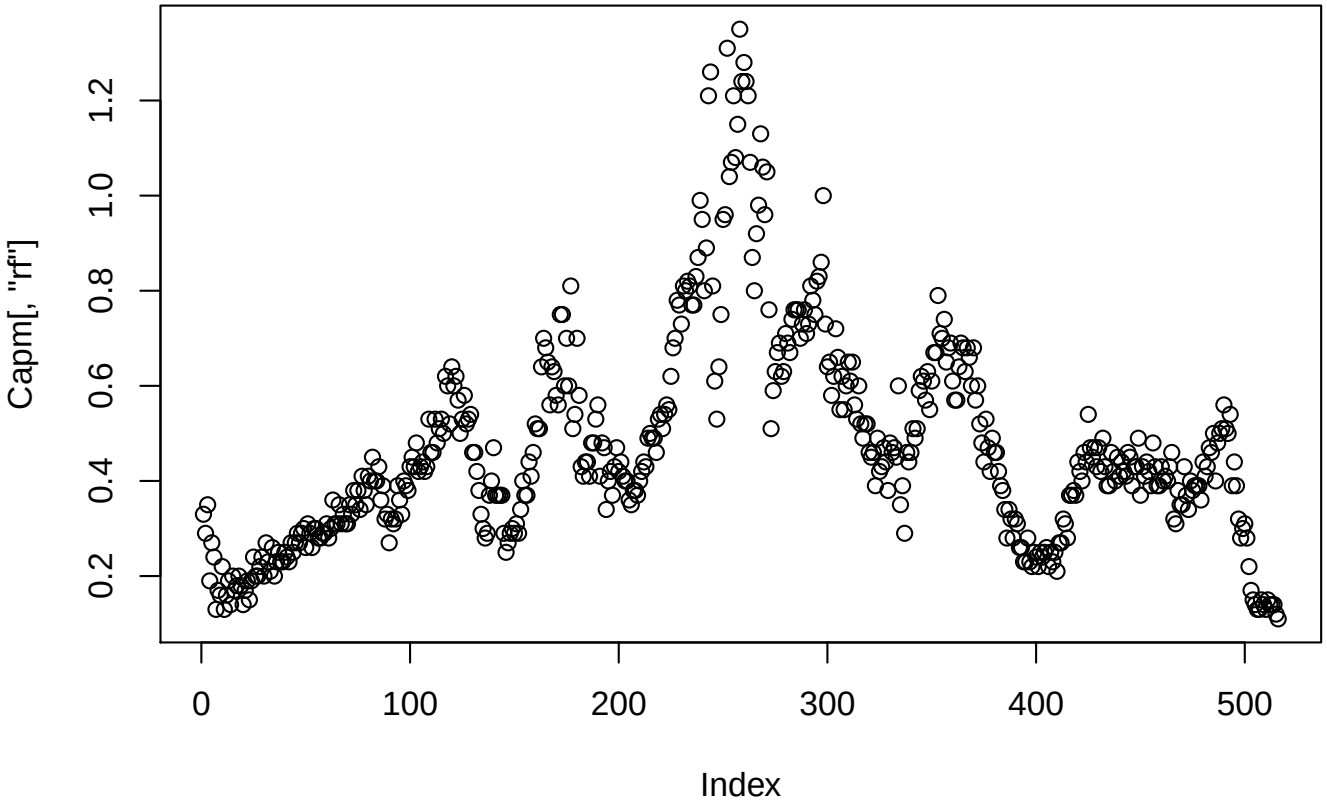
using pch=5 to fix presentation bug



The plot function

R Code: Plot with defaults

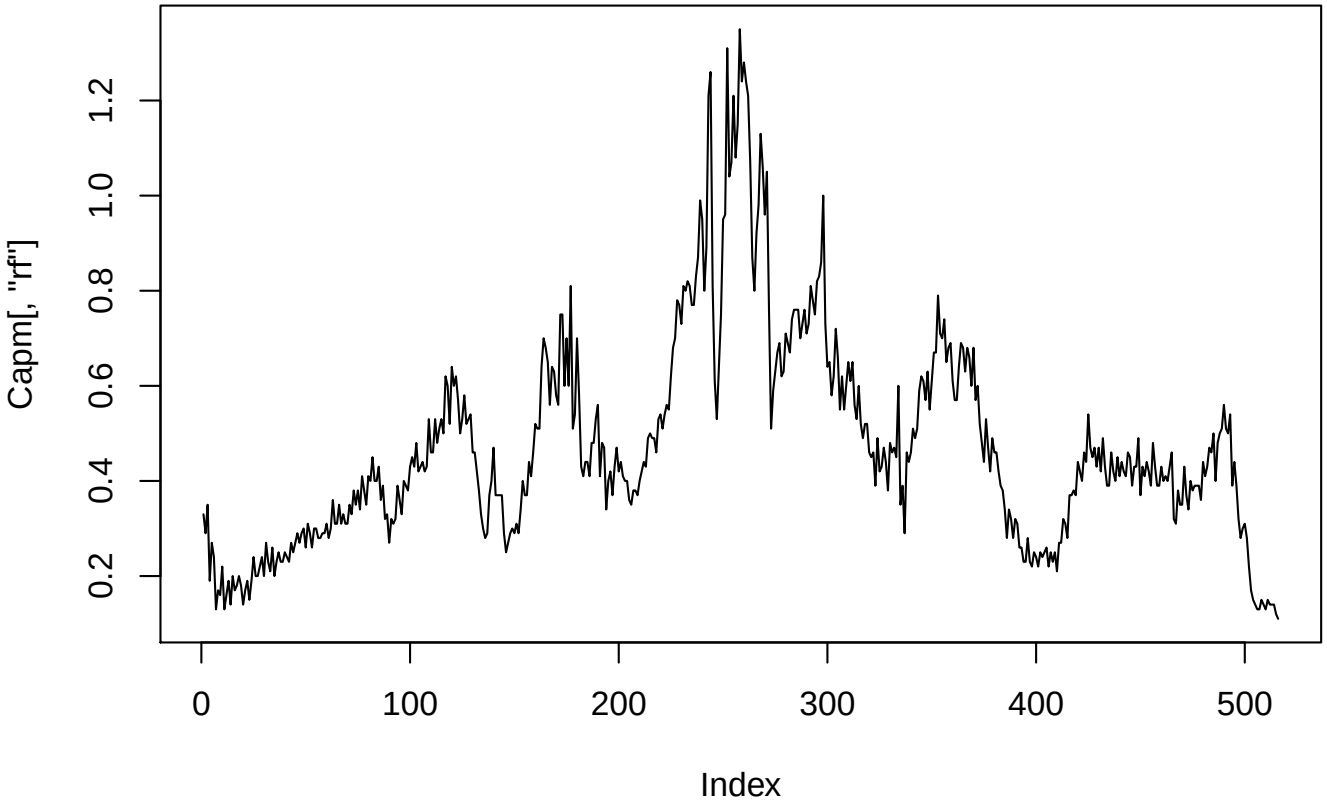
```
> plot(Capm[,"rf"])
```



The plot function

R Code: Plot lines

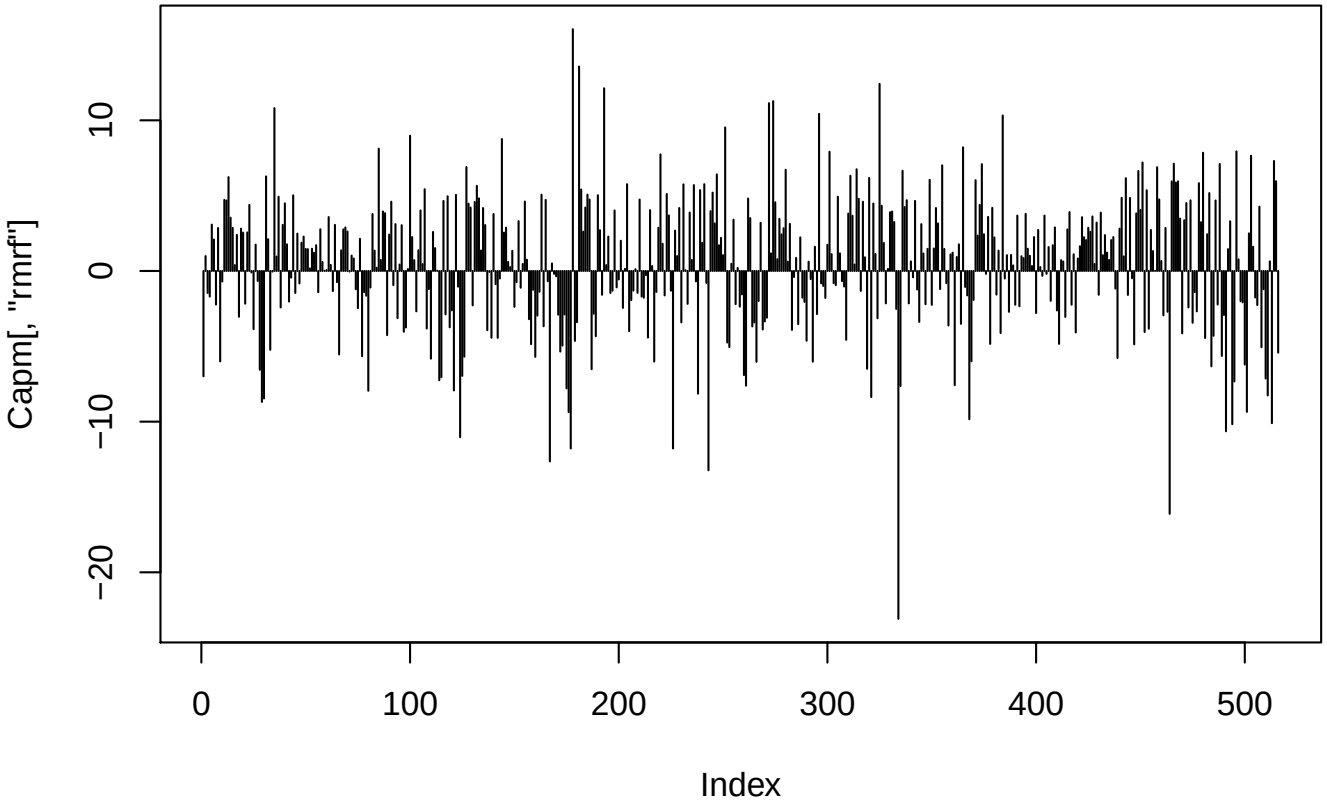
```
> plot(Capm[,"rf"],type="l")
```



The plot function

R Code: Plot bars

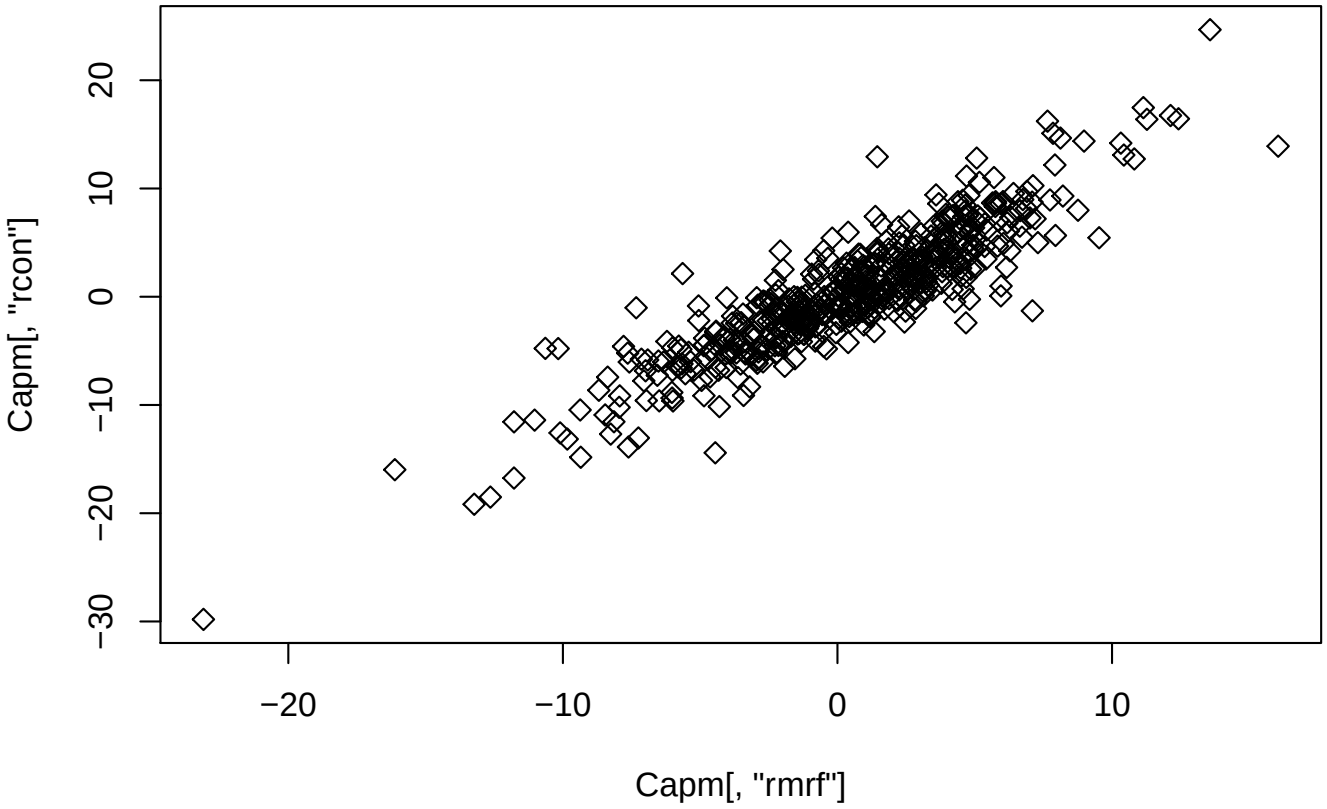
```
> plot(Capm[,"rmrf"],type="h")
```



The plot function

R Code: XY plot

```
> plot(Capm[,"rmrf"],Capm[,"rcon"],pch=5)
```



The points function

The points function adds points to the current plot at the given x, y coordinates

R Code: points arguments

```
> args(points.default)

function (x, y = NULL, type = "p", ...)
NULL
```

- x vector of x coordinates
- y vector of y coordinates

The lines function

The lines function adds connected line segments to the current plot

R Code: lines arguments

```
> args(lines.default)

function (x, y = NULL, type = "l", ...)
NULL
```

- x vector of x coordinates
- y vector of y coordinates

The text function

The text function adds text labels to a plot at given x, y coordinates

R Code: text arguments

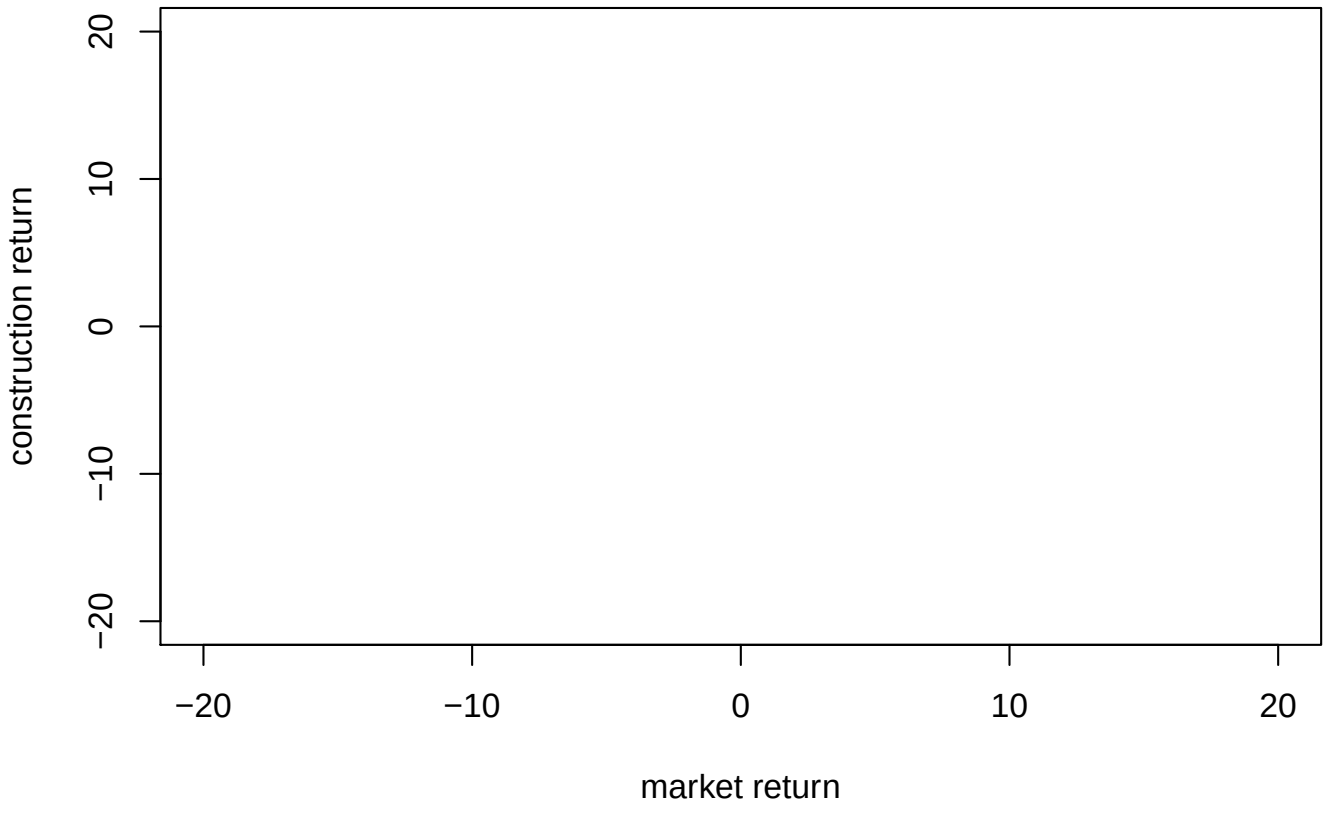
```
> args(text.default)

function (x, y = NULL, labels = seq_along(x), adj = NULL, pos = NULL,
    offset = 0.5, vfont = NULL, cex = 1, col = NULL, font = NULL,
    ...)
NULL
```

x/y	location to place text
labels	text to be display
adj	adjustment of label at x, y location
pos	position of text relative to x, y
offset	offset from pos

Plotting a blank frame

R Code: Plotting a blank frame

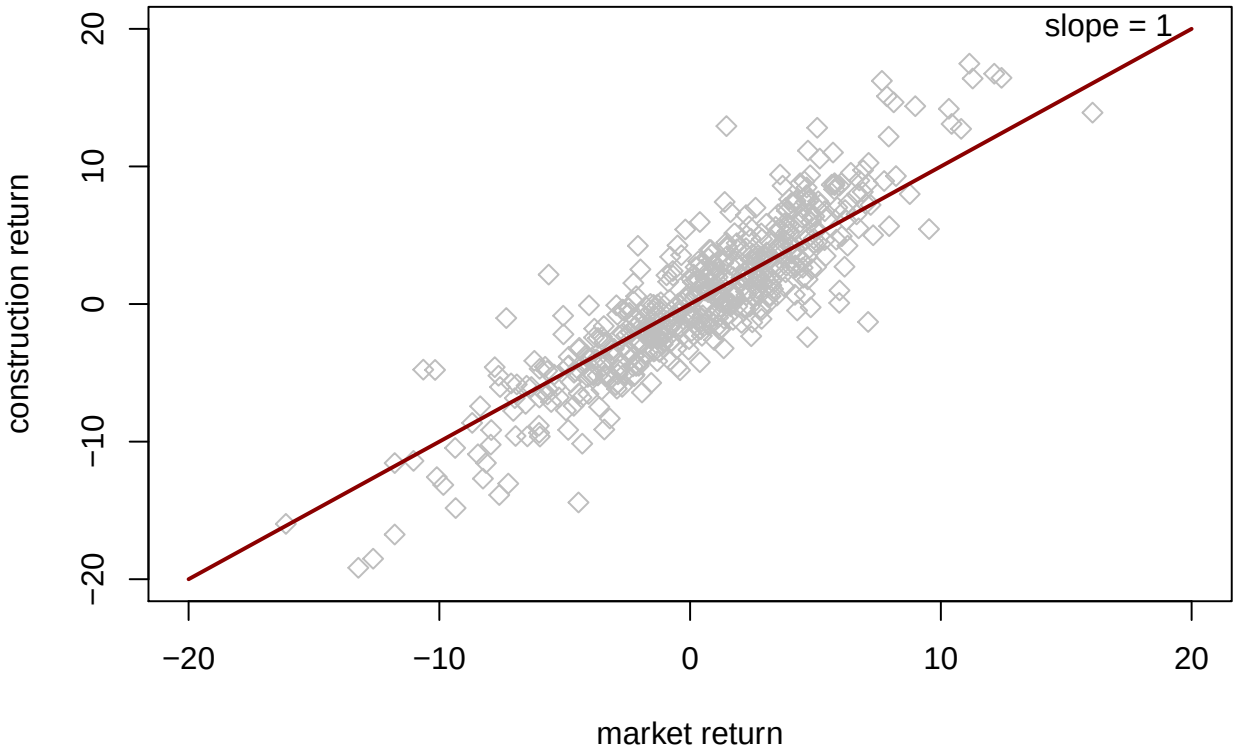
```
> plot(0,xlim=c(-20,20),ylim=c(-20,20),type="n",
   xlab="market return",ylab="construction return")
```



A blank frame with points, lines, and text added

R Code: Adding points, lines, and text to a blank frame

```
> plot(0,xlim=c(-20,20),ylim=c(-20,20),type="n",
   xlab="market return",ylab="construction return")
> points(x=Capm[,"rmrf"],y=Capm[,"rcon"],pch=5,col="gray")
> lines(x=-20:20,y=-20:20,lwd=2,col="darkred")
> text(20,20,labels="slope = 1",pos=2)
```



The segments function

The segments function draws line segments between point pairs

R Code: segments **arguments**

```
> args(segments)

function (x0, y0, x1 = x0, y1 = y0, col = par("fg"), lty = par("lty"),
    lwd = par("lwd"), ...)
NULL
```

- x0, y0 point coordinates from which to draw
- x1, y1 point coordinates to which to draw

The curve function

The curve function draws a curve of a function or expression over a range

R Code: `curve` **arguments**

```
> args(curve)

function (expr, from = NULL, to = NULL, n = 101, add = FALSE,
    type = "l", ylab = NULL, log = NULL, xlim = NULL, ...)
NULL
```

`expr`	function or expression of x
`from`	start of range
`to`	end of range
`n`	number of points over from/to range
`add`	add to current plot (T/F)

The abline function

The abline function adds one or more straight lines through the current plot

R Code: abline **arguments**

```
> args(abline)

function (a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,
    coef = NULL, untf = FALSE, ...)
NULL
```

- h/v vertical or horizontal coordinate of line
- a/b intercept and slope of line

The matplot function

The matplot function plots multiple columns of a matrix versus an index

R Code: matplot **arguments**

```
> args(matplot)

function (x, y, type = "p", lty = 1:5, lwd = 1, lend = par("lend"),
    pch = NULL, col = 1:6, cex = NULL, bg = NA, xlab = NULL,
    ylab = NULL, xlim = NULL, ylim = NULL, ..., add = FALSE,
    verbose = getOption("verbose"))
NULL
```

x/y matrices or vectors to be plotted

Graphical parameters controlled via the par function

R is capable of producing publication quality graphics by allowing (*requiring*) fine-grained control of a number of graphics parameters

R Code: Names of graphical parameters

```
> names(par())

 [1] "xlog"      "ylog"      "adj"       "ann"       "ask"       "bg"
 [7] "bty"       "cex"       "cex.axis"  "cex.lab"   "cex.main"  "cex.sub"
[13] "cin"       "col"       "col.axis"  "col.lab"   "col.main"  "col.sub"
[19] "cra"       "crt"       "csi"       "cxy"       "din"       "err"
[25] "family"    "fg"        "fig"       "fin"       "font"      "font.axis"
[31] "font.lab"  "font.main" "font.sub"  "lab"       "las"       "lend"
[37] "lheight"   "ljoin"     "lmitre"    "lty"       "lwd"       "mai"
[43] "mar"       "mex"       "mfcol"     "mfg"       "mfrow"     "mgp"
[49] "mkh"       "new"       "oma"       "omd"       "omi"       "pch"
[55] "pin"       "plt"       "ps"        "pty"       "smo"       "srt"
[61] "tck"       "tcl"       "usr"       "xaxp"      "xaxs"      "xaxt"
[67] "xpd"       "yaxp"      "yaxs"      "yaxt"
```

Commonly used par parameters

Parameter	Description
`col`	plot color
`lwd`	line width
`lyt`	line type
`mfrow`	set/reset multi-plot layout
`cex.axis`	character expansion - axis
`cex.lab`	character expansion - labels
`cex.main`	character expansion - main
`pch`	point character
`las`	axis label orientation
`bty`	box type around plot or legend

- some parameters can be passed in a `plot` function (e.g. col, lwd)
- some parameters can only be changed by a call to `par` (e.g. mfrow)

The legend function

R Code: `legend` **arguments**

```
> args(legend)

function (x, y = NULL, legend, fill = NULL, col = par("col"),
    border = "black", lty, lwd, pch, angle = 45, density = NULL,
    bty = "o", bg = par("bg"), box.lwd = par("lwd"), box.lty = par("lty"),
    box.col = par("fg"), pt.bg = NA, cex = 1, pt.cex = cex, pt.lwd = lwd,
    xjust = 0, yjust = 1, x.intersp = 1, y.intersp = 1, adj = c(0,
        0.5), text.width = NULL, text.col = par("col"), merge = do.lines &&
        has.pch, trace = FALSE, plot = TRUE, ncol = 1, horiz = FALSE,
    title = NULL, inset = 0, xpd, title.col = text.col, title.adj = 0.5,
    seg.len = 2)
NULL
```

- `x/y` location of the legend (can be give as a position name)
- `legend` vector of labels for the legend
- `col` vector of colors
- `lty` line type
- `lwd` line width
- `pch` character

The barplot function

The barplot function can create vertical or horizontal barplots

R Code: barplot **arguments**

```
> args(barplot.default)

function (height, width = 1, space = NULL, names.arg = NULL,
    legend.text = NULL, beside = FALSE, horiz = FALSE, density = NULL,
    angle = 45, col = NULL, border = par("fg"), main = NULL,
    sub = NULL, xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL,
    xpd = TRUE, log = "", axes = TRUE, axisnames = TRUE, cex.axis = par("cex.axis")
    cex.names = par("cex.axis"), inside = TRUE, plot = TRUE,
    axis.lty = 0, offset = 0, add = FALSE, args.legend = NULL,
    ...)
NULL
```

- **height** vector or matrix (stacked bars or side-by-side bars) of heights
- **names.arg** axis labels for the bars
- **beside** stacked bars or side-by-side if height is a matrix
- **legend** vector of labels for stacked or side-by-side bars

Outline

Time series data

Time Series

A time series is a sequence of *ordered* data points measured at specific points in time

A time series class in R is a *compound data object* that includes a data matrix as well as a vector of associated time stamps

class	**package**	**overview**
`ts`	base	regularly spaced time series
`mts`	base	multiple regularly spaced time series
`its`	tseries	irregularly spaced time series
`timeSeries`	rmetrics	default for Rmetrics packages
`fts`	fts	R interface to tslib (c++ time series library)
`zoo`	zoo	reg/irreg and arbitrary time stamp classes
`xts`	xts	an extension of the zoo class

Time series methods

Time series classes in R will typically implement the following methods:

`start`	return start of time series
`end`	return end of time series
`frequency`	return frequency of time series
`window`	Extract subset of time series
`index`	return time index of time series
`time`	return time index of time series
`coredata`	return data of time series
`diff`	difference of the time series
`lag`	lag of the time series
`aggregate`	aggregate to lower resolution time series
`cbind`	merge 2 or more time series together

Creating a zoo object

R Code: Creating a zoo object

```
> library(zoo)
> msft.df <- read.table("table.csv", header = TRUE, sep = ",", as.is = TRUE)
> head(msft.df,2)

        Date  Open  High   Low Close    Volume Adj.Close
1 2010-09-13 24.20 25.29 24.09 25.11 114606300     25.11
2 2010-09-10 23.98 24.03 23.79 23.85  58284300     23.85

> args(zoo)

function (x = NULL, order.by = index(x), frequency = NULL)
NULL

> msft.z <- zoo(x=msft.df[,"Close"],order.by=as.Date(msft.df[,"Date"]))
> head(msft.z)

2009-01-02 2009-01-05 2009-01-06 2009-01-07 2009-01-08 2009-01-09
     20.33      20.52      20.76      19.51      20.12      19.52
```

Inspecting a zoo object

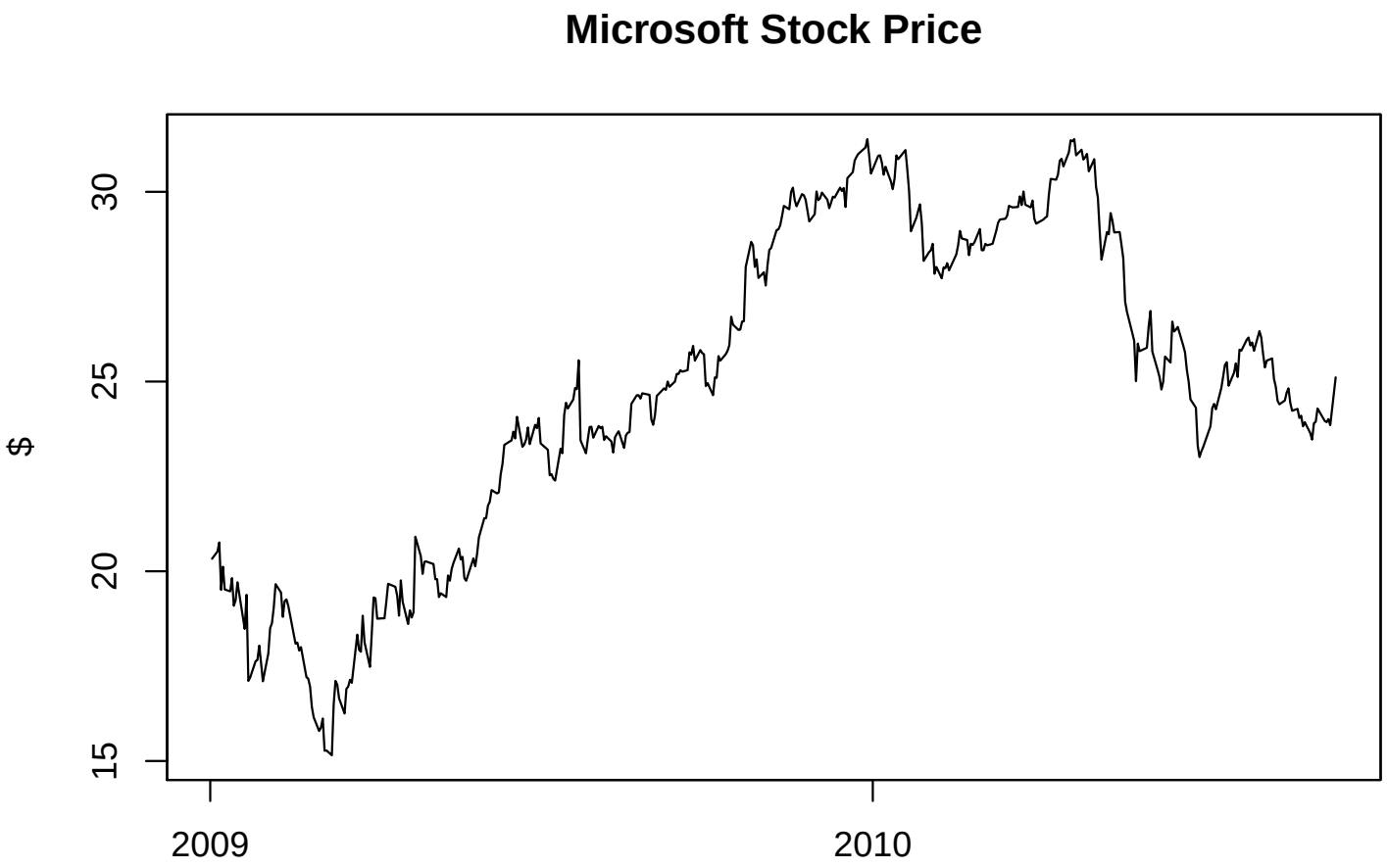
R Code: Inspecting a zoo object

```
> class(msft.z)
[1] "zoo"
> start(msft.z)
[1] "2009-01-02"
> end(msft.z)
[1] "2010-09-13"
> frequency(msft.z)
[1] 1
> class(coredata(msft.z))
[1] "numeric"
> class(time(msft.z))
[1] "Date"
```

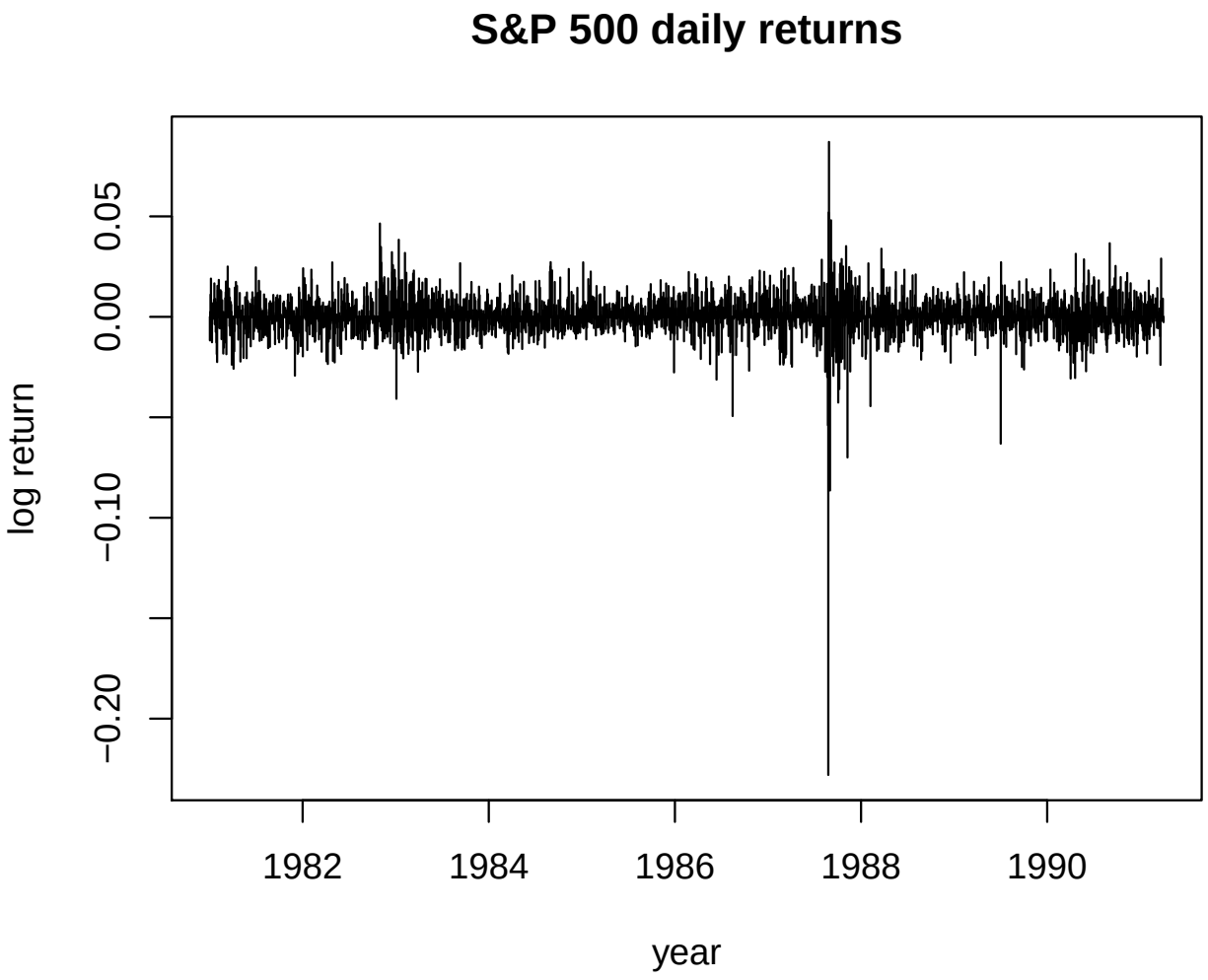
Plotting a zoo object

R Code: Plotting a zoo object

```
> plot(msft.z,xlab="",ylab="$", main="Microsoft Stock Price")
```



S&P 500 Jan-1981 to Apr-1991



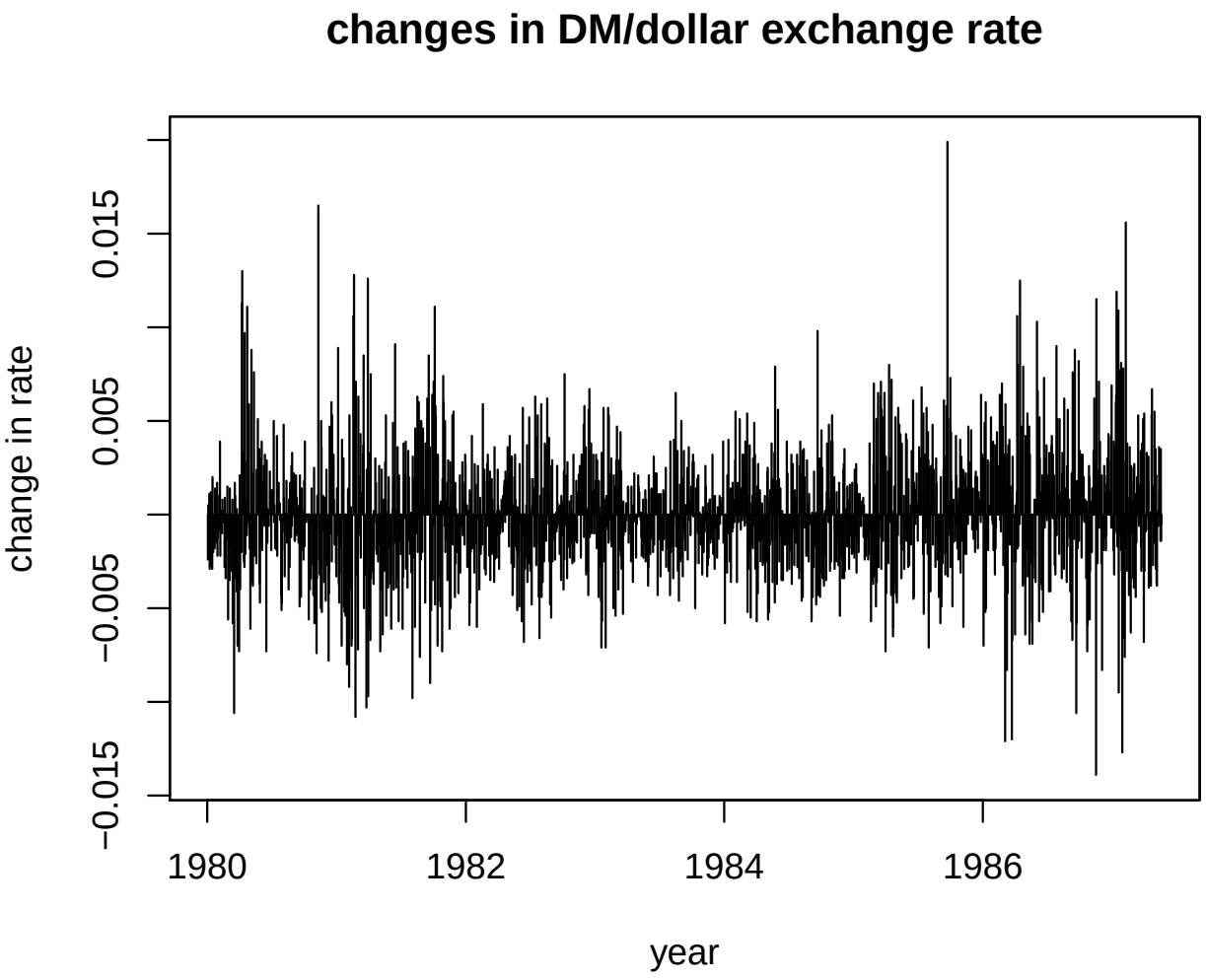
SDAFE Fig 4.1

Plot S&P 500 returns

R Code: Plot SP500 returns

```
> # figure 4.1
> library(Ecdat)
> data(SP500)
> class(SP500)

[1] "data.frame"

> dim(SP500)

[1] 2783    1

> SPreturn = SP500$r500
> head(SPreturn)

[1] -0.0117265  0.0024544  0.0110516  0.0190512 -0.0055657 -0.0043148

> n = length(SPreturn)
> year_SP = 1981 + (1:n)*(1991.25-1981)/n
> head(year_SP)

[1] 1981.004 1981.007 1981.011 1981.015 1981.018 1981.022

> plot(x=year_SP,y=SPreturn,type="h",xlab="year",ylab="log return")
> title("S&P 500 daily returns")
```

Deutsche Mark exchange rate Jan-1980 to May-1987



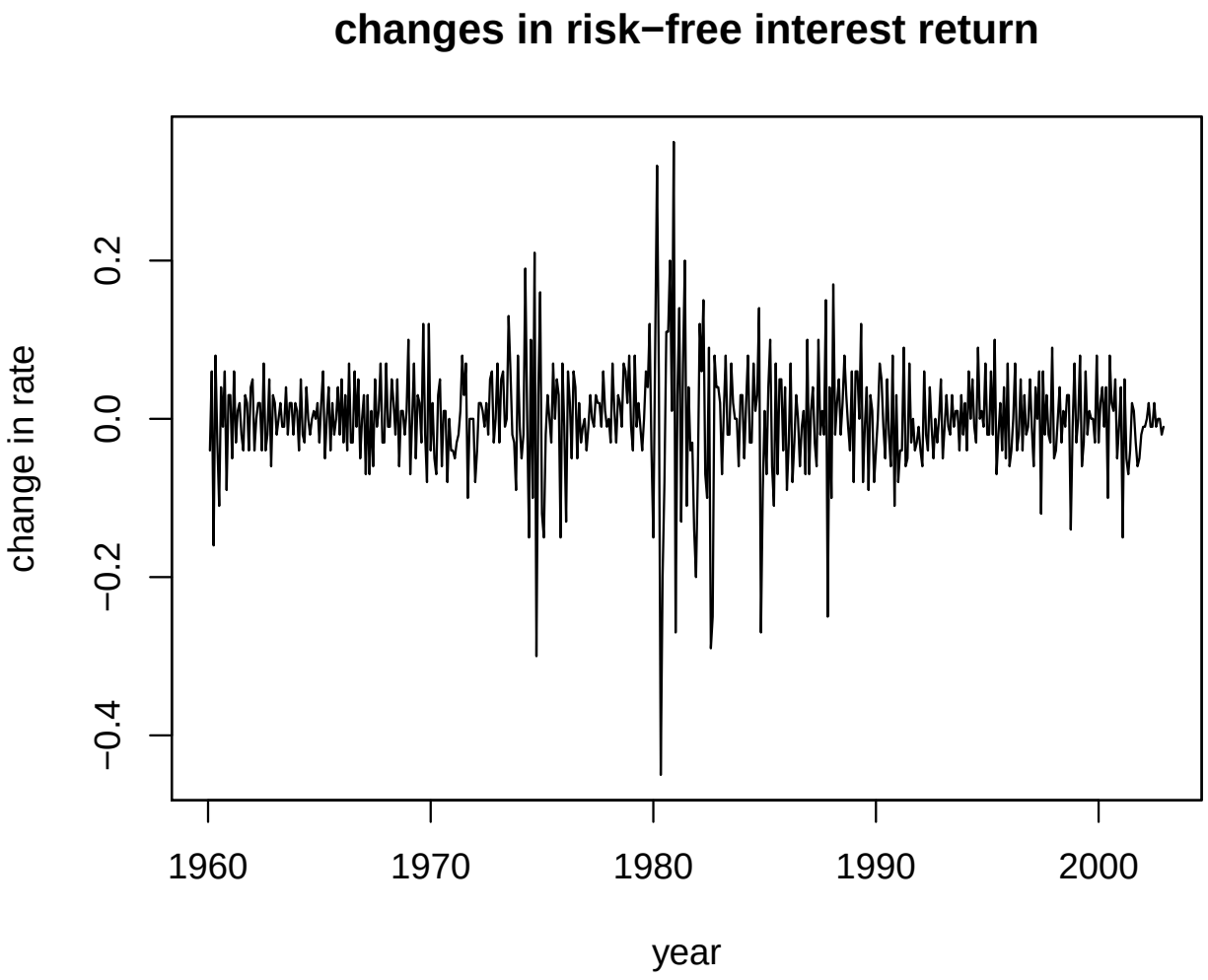
SDAFE Fig 4.2

Plot DM returns

R Code: Plot DM returns

```
> # figure 4.2
> library(zoo)
> data(Garch)
> head(Garch)

    date       day     dm           ddm     bp     cd       dy     sf
1 800102 wednesday 0.5861            NA 2.2490 0.8547 0.004206 0.6365
2 800103  thursday 0.5837 -0.0041032713 2.2365 0.8552 0.004187 0.6357
3 800104    friday 0.5842  0.0008562377 2.2410 0.8566 0.004269 0.6355
4 800107    monday 0.5853  0.0018811463 2.2645 0.8538 0.004315 0.6373
5 800108   tuesday 0.5824 -0.0049670394 2.2560 0.8553 0.004257 0.6329
6 800109 wednesday 0.5834  0.0017155606 2.2650 0.8565 0.004245 0.6349

> dm <- zoo(x=Garch[,"dm"],
   order.by=as.Date(x=as.character(Garch[,"date"]),format="%y%m%d"))
> head(dm)

1980-01-02 1980-01-03 1980-01-04 1980-01-07 1980-01-08 1980-01-09
    0.5861     0.5837     0.5842     0.5853     0.5824     0.5834

> diffdm <- diff(dm)
> plot(diffdm,xlab="year",ylab="change in rate",type="h")
> title("changes in DM/dollar exchange rate")
```

T-bill rate changes Jan-1960 to Dec-2002



SDAFE Fig 4.3

Plot T-bill rate changes

R Code: Plot T-bill rate changes

```
> # figure 4.3
> data(Capm)
> head(Capm)

  rfood  rdur  rcon  rmrf   rf
1 -4.59  0.87 -6.84 -6.99 0.33
2  2.62  3.46  2.78  0.99 0.29
3 -1.67 -2.28 -0.48 -1.46 0.35
4  0.86  2.41 -2.02 -1.70 0.19
5  7.34  6.33  3.69  3.08 0.27
6  4.99 -1.26  2.05  2.09 0.24

> rf <- zooreg(Capm[,"rf"], frequency = 12, start = c(1960, 1),end=c(2002,12))
> head(rf)

1960(1) 1960(2) 1960(3) 1960(4) 1960(5) 1960(6)
   0.33    0.29    0.35    0.19    0.27    0.24

> diffrf <- diff(rf)
> plot(diffrf,xlab="year",ylab="change in rate")
> title("changes in risk-free interest return")
```

Outline

Free variables

In the body of a function, 3 types of symbols may be found:

- formal parameters - arguments passed in the function call
- local variables - variables created in the function
- free variables - variables created outside of the function
  (note, free variables become local variables if you assign to them)

R Code: Types of variables in functions

```
> f <- function(x) {
   y <- 2*x
   print(x) # formal parameter
   print(y) # local variable
   print(z) # free variable
 }
```

Environments

The main workspace in R (i.e. what you are interacting with at the R console) is called the *global environment*

According to the scoping rules of R (referred to a *lexical scoping*), R will search for a free variable in the following order:

1. The environment in which the function was created
   - For functions created in the global environment, this will be the global environment
2. The parent environment of the environment where the function was created
3. The parent of the parent ... up until the global environment is searched
4. The search path of loaded libraries found using the search() function

The search path

The function `search` returns a list of attached packages which will be searched in order (after the global environment) when trying to resolve a free variable

R Code: The search path

```
> search()

 [1] ".GlobalEnv"        "package:zoo"       "package:Ecdat"
 [4] "package:nutshell"  "package:patchDVI"  "package:stats"
 [7] "package:graphics"  "package:grDevices" "package:utils"
[10] "package:datasets"  "package:methods"   "Autoloads"
[13] "package:base"
```

Variable scoping examples

R Code: Variable scoping examples

```
> # example 1
> a <- 10
> x <- 5
> f <- function (x) x + a
> f(2)

[1] 12

> # example 2
> f<- function (x)
 {
   a<-5
   g(x)
 }
> g <- function(y) y + a
> f(2)

[1] 12
```

Variable scoping examples

R Code: Variable scoping examples

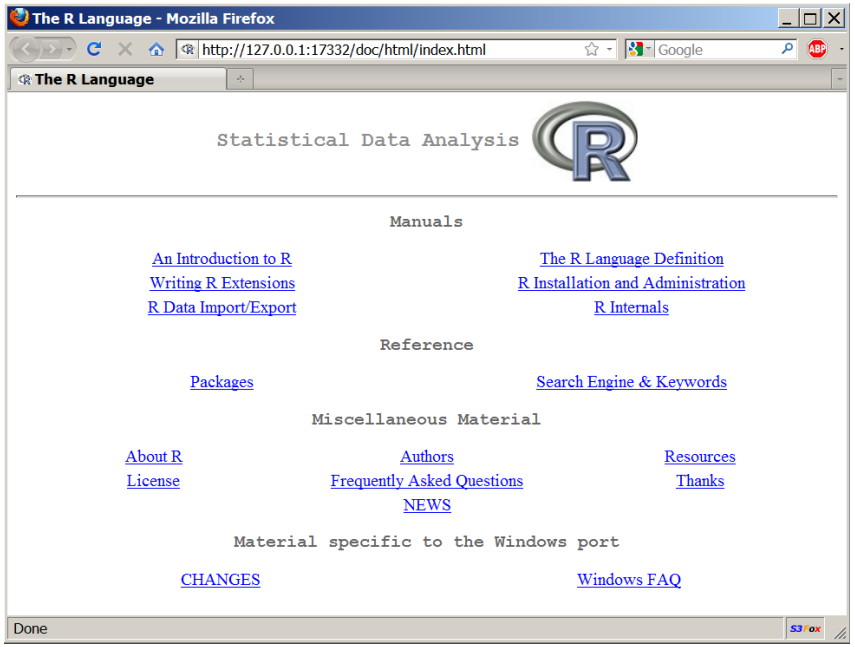
```
> # example 3
> f <- function (x) {
   a<-5
   g <- function (y) y + a
   g(x)
 }
> f(2)

[1] 7

> # example 4
> f <- function (x) {
   x + mean(rivers) # rivers is defined in the dataset package
 }
> f(2)

[1] 593.1844
```

Outline

The HTML help system

R has a comprehensive HTML help facility

- Run the `help.start` function
- R GUI menu item `Help|Html help`



R Code: Starting HTML help

```
> help.start()

If nothing happens, you should open
'http://127.0.0.1:24487/doc/html/index.html' yourself
```

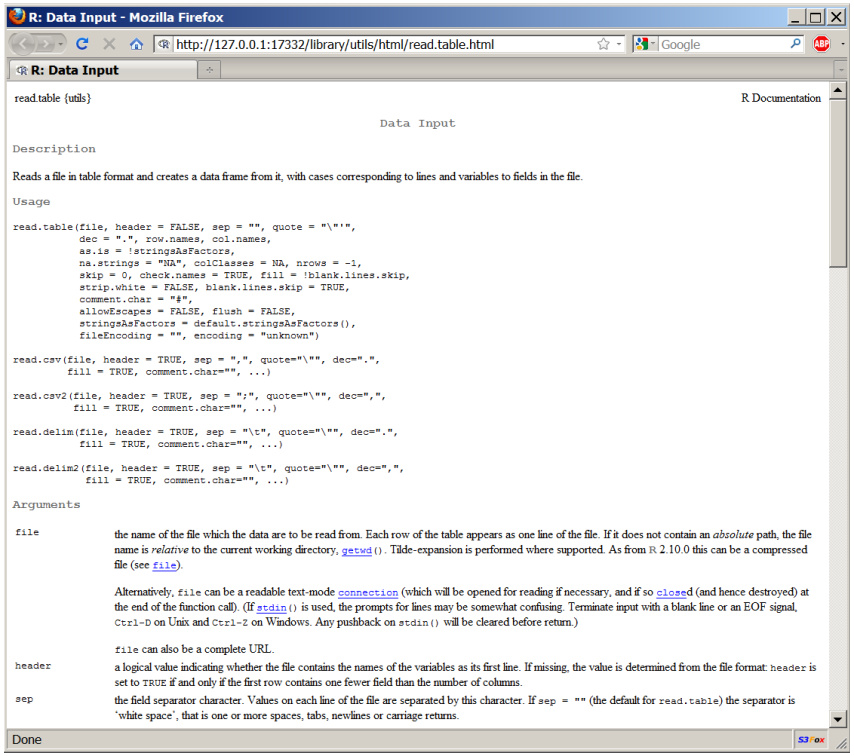
The help function

One can also obtain help on a particular topic via the `help` function

- `help(topic)`
- `?topic`

R Code: Topic help

```
> help(read.table)
```



The RSiteSearch function

The function RSiteSearch can be used to search the R website

- HTML help for all packages
- R-help archives

R Code: Running RSiteSearch

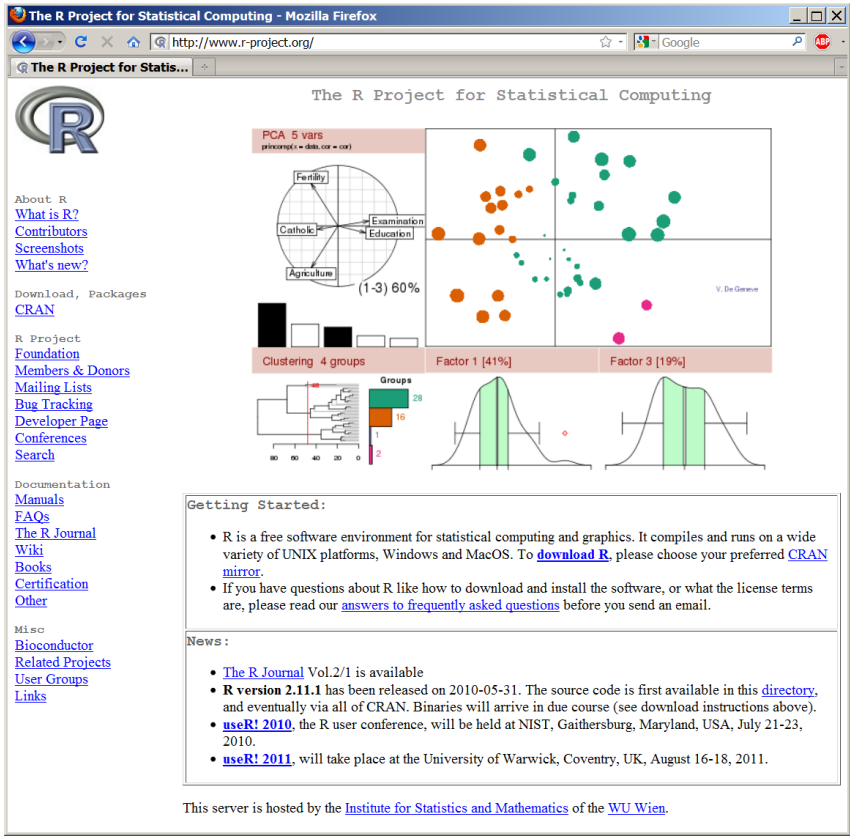
```
> library(RSiteSearch)
> RSiteSearch("ODBC")

A search query has been submitted to http://search.r-project.org
The results page should open in your browser shortly
```

Outline

R Homepage

http://www.r-project.org

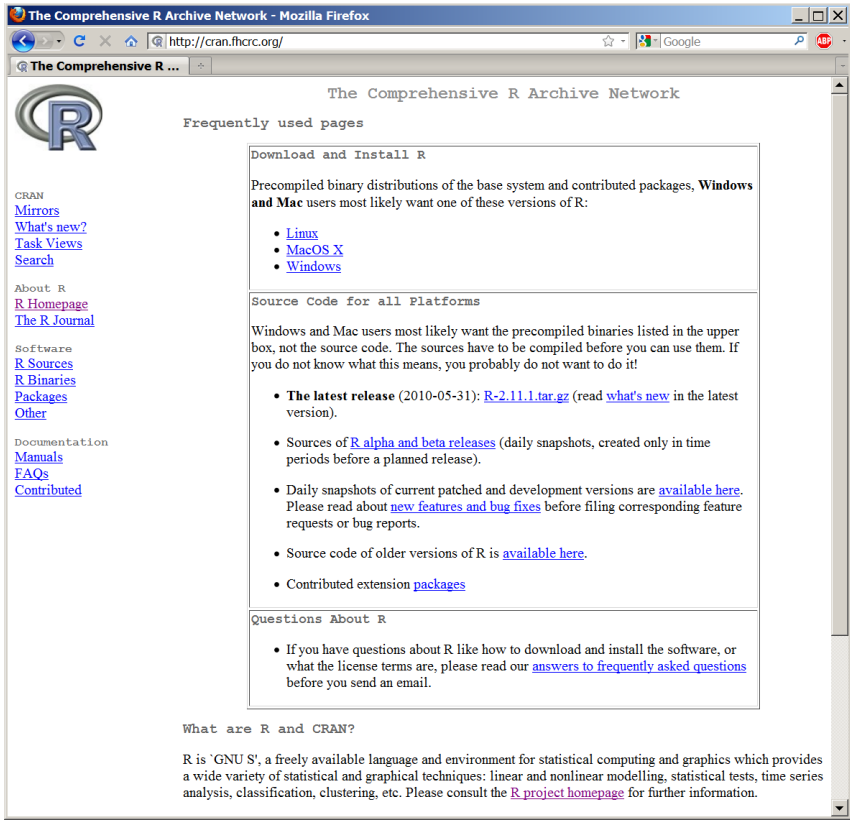
- List of CRAN mirror sites
- Manuals
- FAQs
- Mailing Lists
- Links



CRAN - Comprehensive R Archive Network

`http://cran.fhcrc.org`

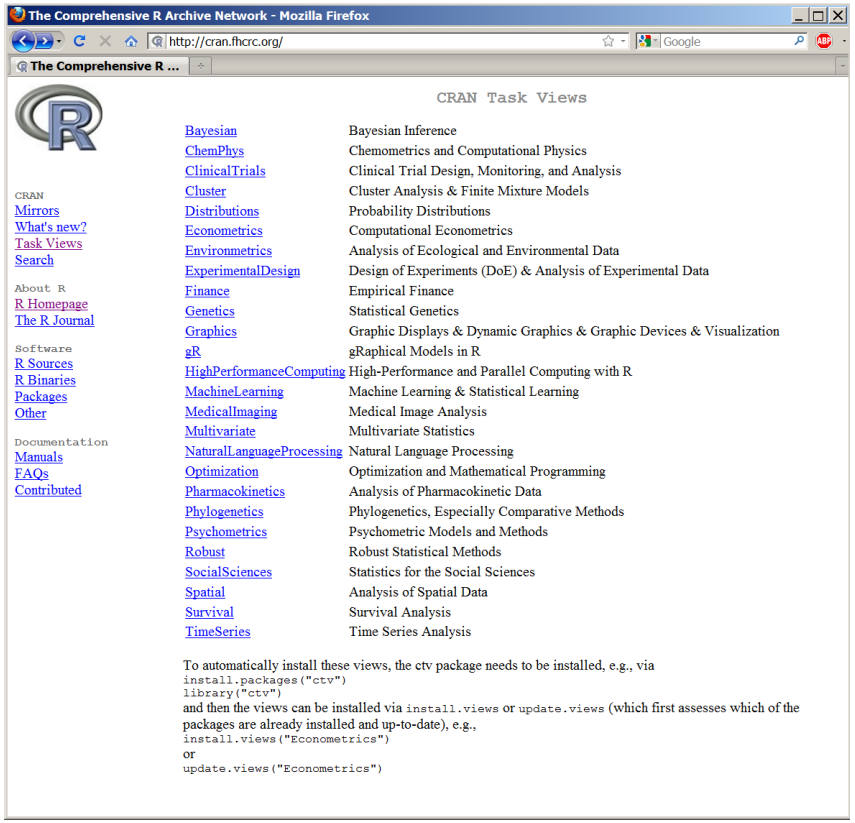
- CRAN Mirrors
  - About 75 sites worldwide
  - About 16 sites in US
- R Binaries
- R Packages
- R Sources
- Task Views



CRAN Task Views

Organizes the 2600+ R packages by application

- Finance
- Time Series
- Econometrics
- Optimization
- Machine Learning



Statconn

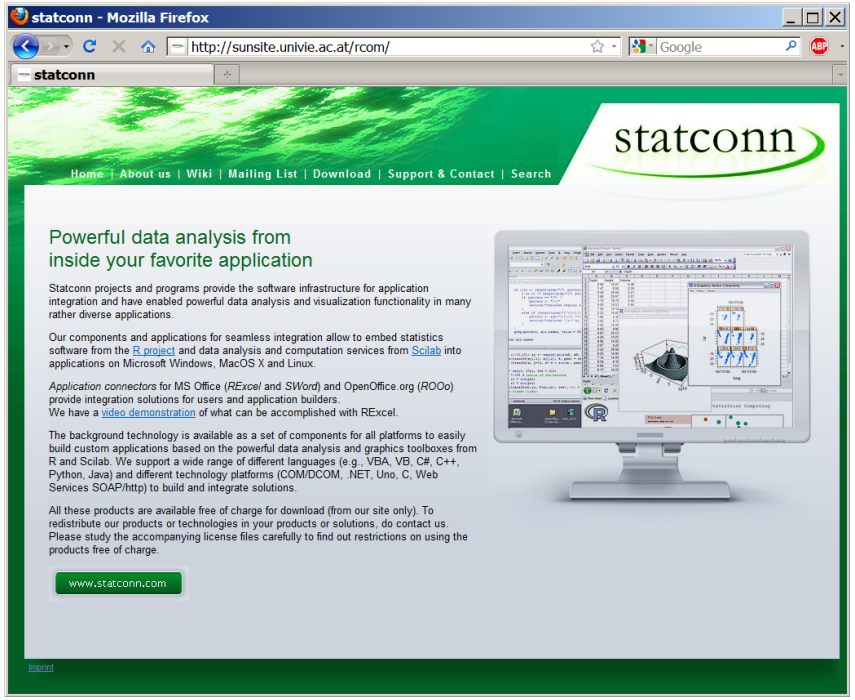
`http://rcom.univie.ac.at`

COM interface for R connectivity

- Excel
- Word
- C#
- VB
- Delphi

Download site for RAndFriends

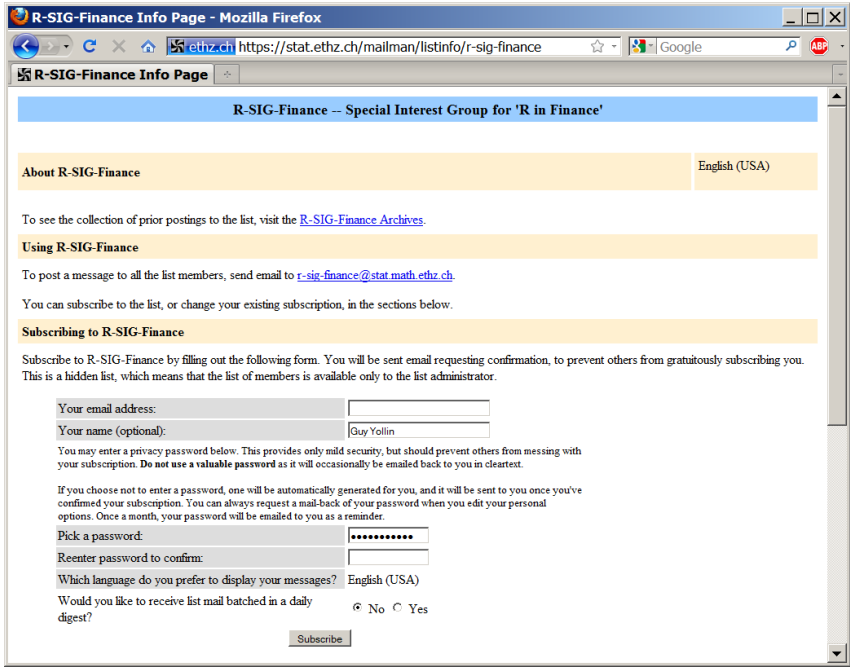
- R
- Statconn
- Notepad++



R-SIG-FINANCE

`https://stat.ethz.ch/mailman/listinfo/r-sig-finance`

- Nerve center of the R finance community
- Daily must read
- Exclusively for Finance-specific questions, not general R questions



Google's R Style Guide

http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html

- Naming convention
- Coding Syntax
- Program Organization

Google's R Style Guide - Mozilla Firefox

http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html

Google's R Style Guide

Google's R Style Guide

R is a high-level programming language used primarily for statistical computing and graphics. The goal of the R Programming Style Guide is to make our R code easier to read, share, and verify. The rules below were designed in collaboration with the entire R user community at Google.

Summary: R Style Rules

1. File Names: end in `.R`
2. Identifiers: `variable.name`, `FunctionName`, `kConstantName`
3. Line Length: maximum 80 characters
4. Indentation: two spaces, no tabs
5. Spacing
6. Curly Braces: first on same line, last on own line
7. Assignment: use `<-`, not `=`
8. Semicolons: don't use them
9. General Layout and Ordering
10. Commenting Guidelines: all comments begin with `#` followed by a space; inline comments need two spaces before the `#`
11. Function Definitions and Calls
12. Function Documentation
13. Example Function
14. TODO Style: `TODO(username)`

Summary: R Language Rules

1. `attach`: avoid using it
2. Functions: errors should be raised using `stop()`
3. Objects and Methods: avoid S4 objects and methods when possible; never mix S3 and S4

1. Notation and Naming

File Names

File names should end in `.R` and, of course, be meaningful.
GOOD: `predict_ad_revenue.R`
BAD: `foo.R`

Identifiers

Don't use underscores (_) or hyphens (-) in identifiers. Identifiers should be named according to the following conventions. Variable names should have all lower case letters and words separated with dots (`.`); function names have initial capital letters and no dots (CapWords); constants are named like functions but with an initial `k`.

- `variable.name`
  GOOD: `avg.clicks`
  BAD: `avg_Clicks` , `avgClicks`
- `FunctionName`
  GOOD: `CalculateAvgClicks`

Quick R

http://www.statmethods.net

Introductory R Lessons

- R Interface
- Data Input
- Data Management
- Basic Statistics
- Advanced Statistics
- Basic Graphs
- Advanced Graphs

R graphics details, colors, and other tech notes

R Graphics and other useful information by Earl Glynn of Stowers Institute for Medical Research

- URL
  `http://research.stowers-institute.org/efg/R/index.htm`

- Features
  - R Color Chart
  - Using Color in R (great presentation)
  - Plot area, margins, multiple figures
  - Mixture models
  - Distance measures and clustering
  - Using Windows Explorer to Start R with Specified Working Directory (under tech notes)

Seven Tips for Surviving R

A presentation from the Bay Area R Users Meetup by John Mount

- Link to presentation
  - `http://www.win-vector.com/dfiles/SurviveR.pdf`
- Link to step-by-step tutorial
  - `http://www.win-vector.com/blog/2009/11/r-examine-objects-tutorial`

Programming in R

Online R programmingn manual from UC Riverside

- URL

  `http://manuals.bioinformatics.ucr.edu/home/programming-in-r`

- Selected Topics
  - R Basics
  - Finding Help
  - Code Editors for R
  - Control Structures
  - Functions
  - Object Oriented Programming
  - Building R Packages

Other useful R sites

R Seek R specific search site

- http://www.rseek.org/

R Bloggers Aggregation of about 100 R blogs

- http://www.r-bloggers.com

Stack Overflow Excellent developer Q&A forum

- http://stackoverflow.com

R Graph Gallery Examples of many possible R graphs

- http://addictedtor.free.fr/graphiques

Revolution Blog Blog from David Smith of Revolution

- http://blog.revolutionanalytics.com

Inside-R R community site by Revolution Analytics

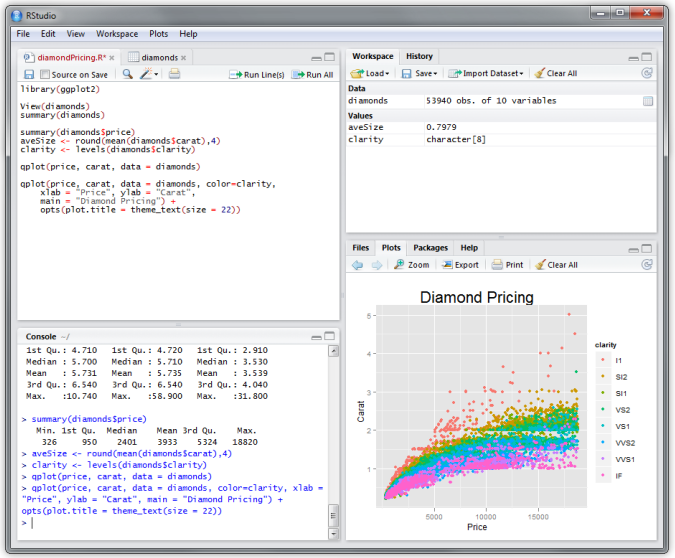
- http://www.inside-r.org

Outline

RStudio

RStudio is an fully-featured open-source IDE for R

- R language highlighting
- Paste/Source code to R
- object explorer
- graphics window in main IDE



RStudio also provides a server-based version (R running in the cloud). Request a free account from `josh@rstudio.org` (be sure to reference the UW Computational Finance program)

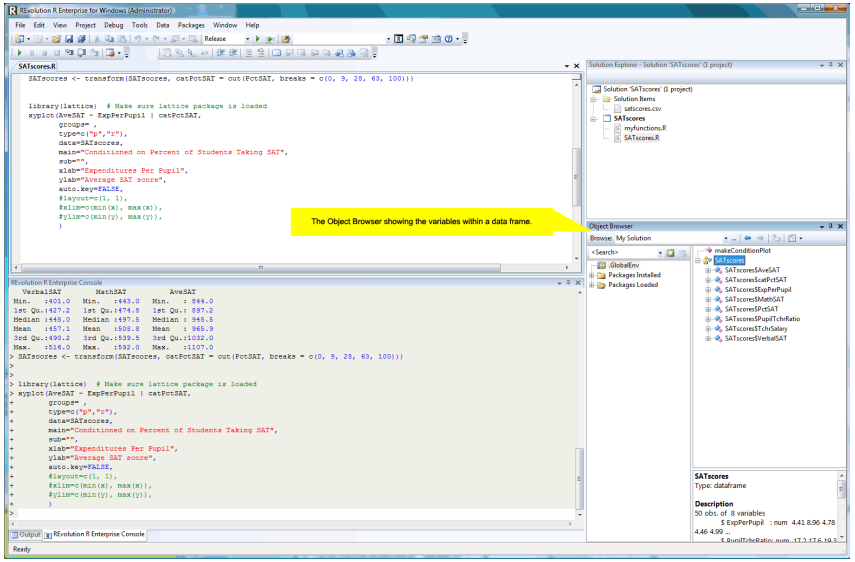
RStudio is highly recommended

Revolution R Enterprize Visual Development Environment

Revolution Analytics is a company that sells a commercial distribution of R including a desktop IDE

Revolution R Enterprize is *free* to academic users

- R language highlighting
- Paste/Source code to R
- object explorer
- runs R in SDI mode

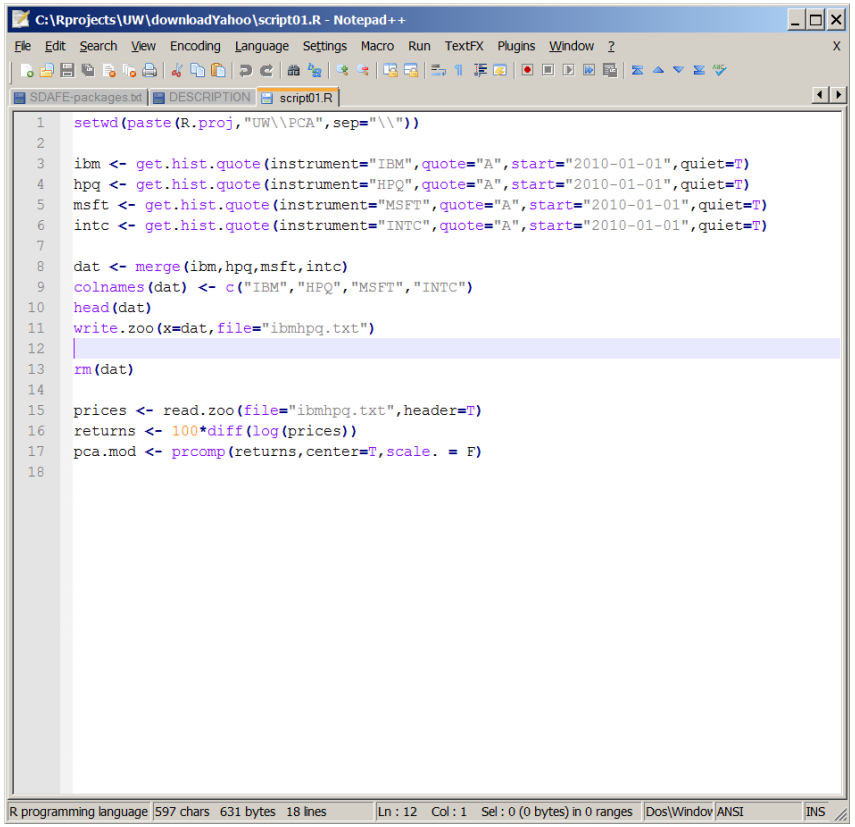


`http://www.revolutionanalytics.com`

Notepad++ and NpptoR

NpptoR is an automation widget (based on AuotHotkey) which allows the very useful program editor Notepad++ to interact with R

- R language highlighting
- Paste/Source code to R
- Supports R in SDI mode
- Can be installed as part of RAndFriends

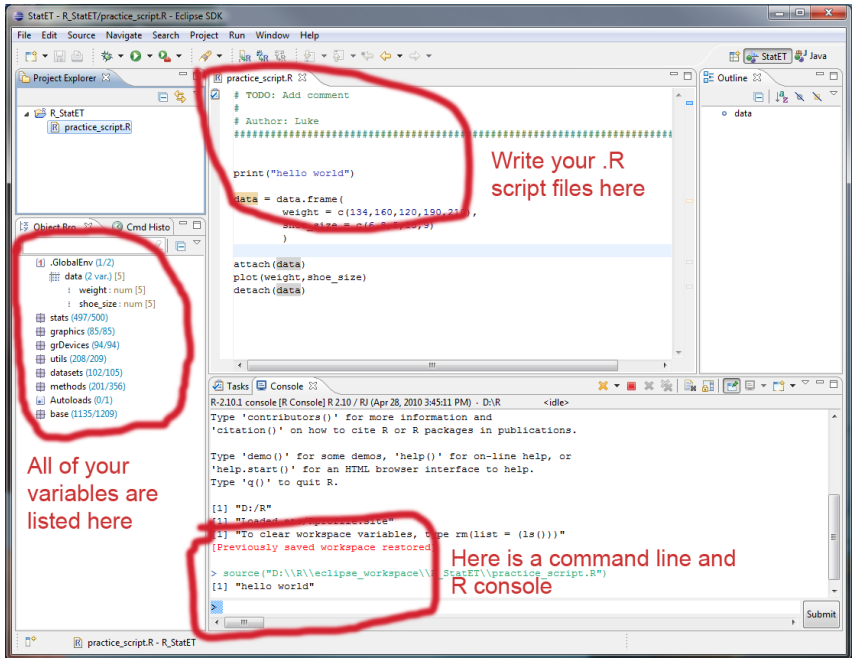


http://notepad-plus-plus.org
http://sourceforge.net/projects/npptor
http://rcom.univie.ac.at/download.html

StatET - An Eclipse Plug-In for R

StatET is a plug-in for the open-source Eclipse development environment

- R language highlighting
- Paste/Source code to R
- Supports R in SDI mode
- Excellent documentation by Longhow Lam

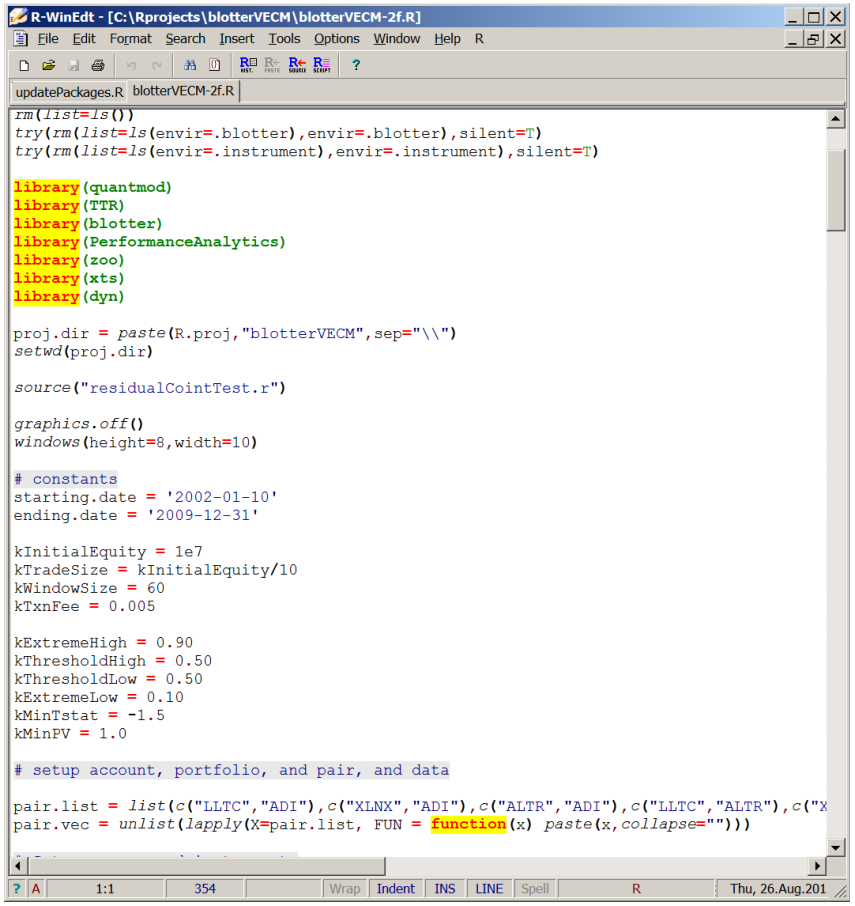


`http://www.walware.de/goto/statet`

R-WinEdit

Based on WinEdt, an excellent shareware editor with support for LaTeXand Sweave development

- R language highlighting
- Paste/Source code to R
- Supports R in MDI mode
- Paste/Source code to S-PLUS



http://www.winedt.com

http://cran.r-project.org/web/packages/RWinEdt

Other R IDE links

Tinn-R Popular R IDE

- http://www.sciviews.org/Tinn-R

ESS Emacs Speaks Statistics

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

other R GUI Projects

- http://www.sciviews.org/_rgui

The End