COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

# R Introduction

Guy Yollin

Acting Instructor, Applied Mathematics
University of Washington

# Outline

1. Part 1
   - R overview and history
   - R language references
2. Part 2
   - R language and environment basics
   - Data structures, data manipulation, working directory, data files
   - The R help system
   - Web resources for R
   - IDE editors for R
3. Part 3
   - Basic plotting
   - Basic statistics and the normal distribution
   - Working with time series in R
   - Variable scoping in R

# Lecture references


W. N. Venables and D. M. Smith.
*An Introduction to R*.
2011.

J. Adler.
*R in a Nutshell: A Desktop Quick Reference*.
O'Reilly Media, 2010.

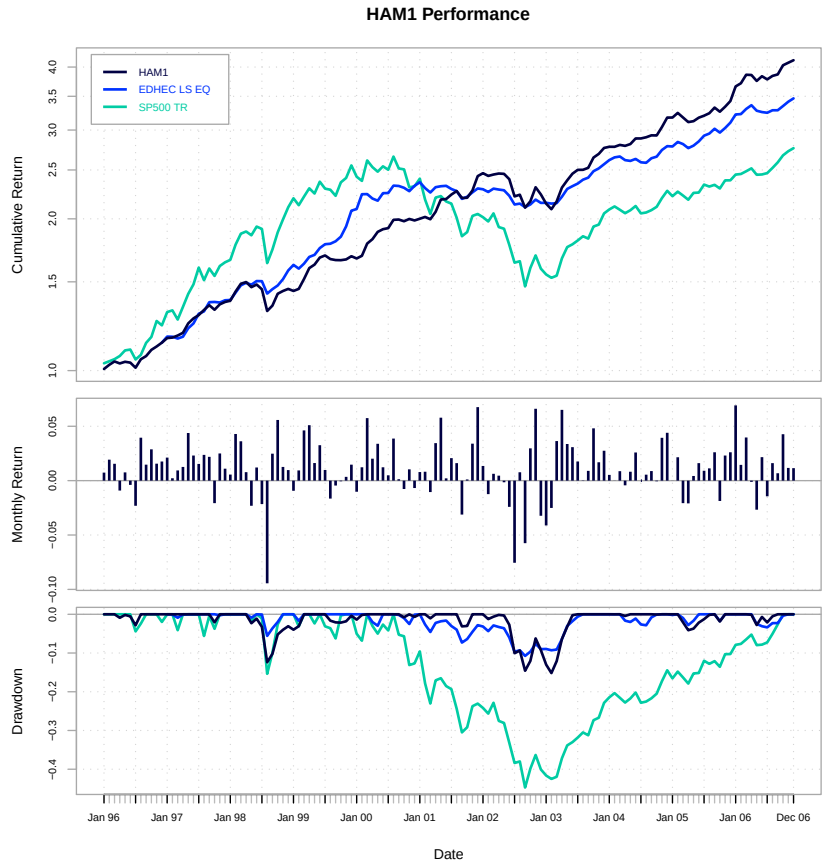
# Outline

# 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

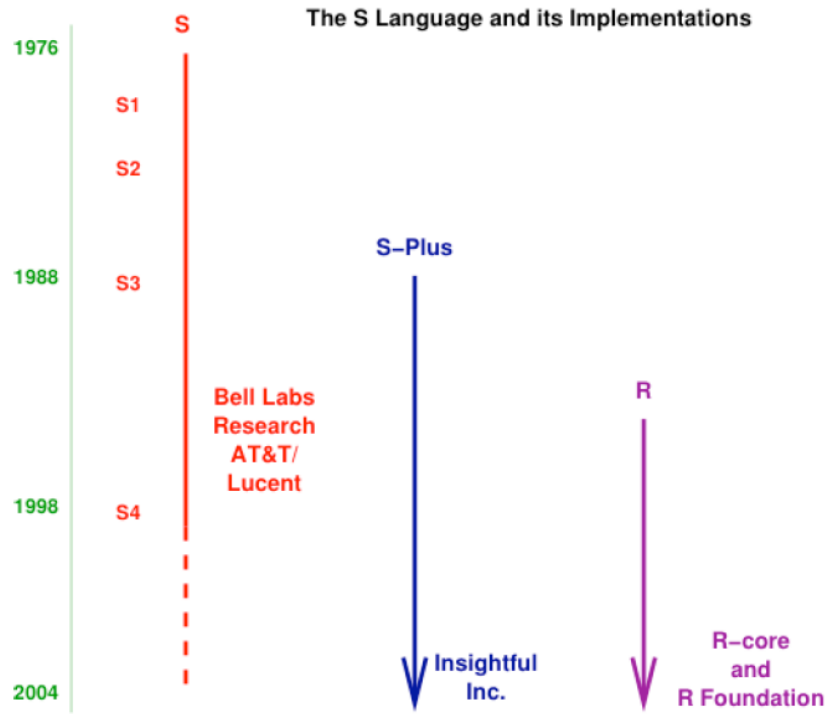
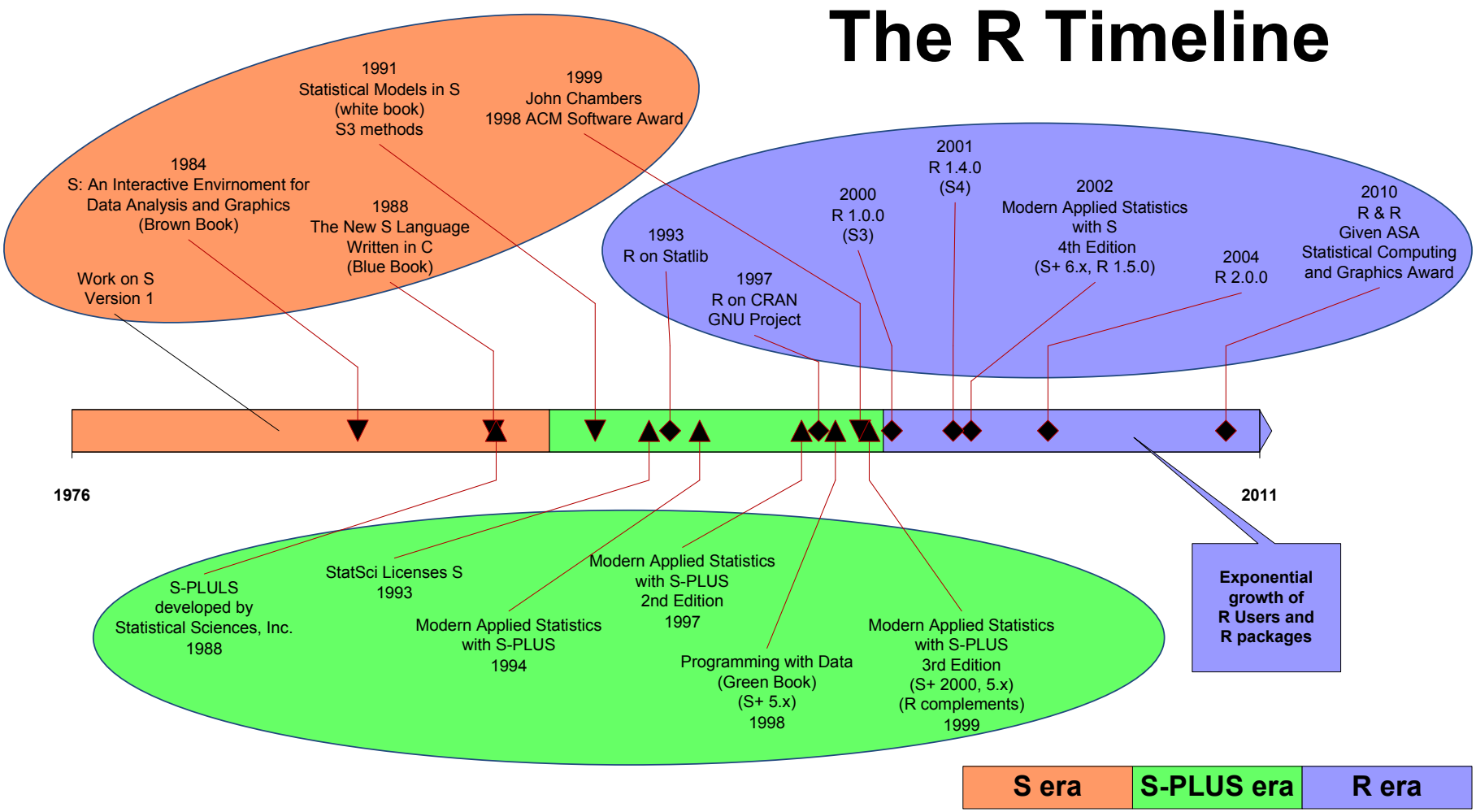



Figure from *The History of S and R, John Chambers, 2006*

[†]Founded by UW Professor Doug Martin, CompFin Program Director

# R timeline


The 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
S era
S-PLUS era
R era

# 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 |

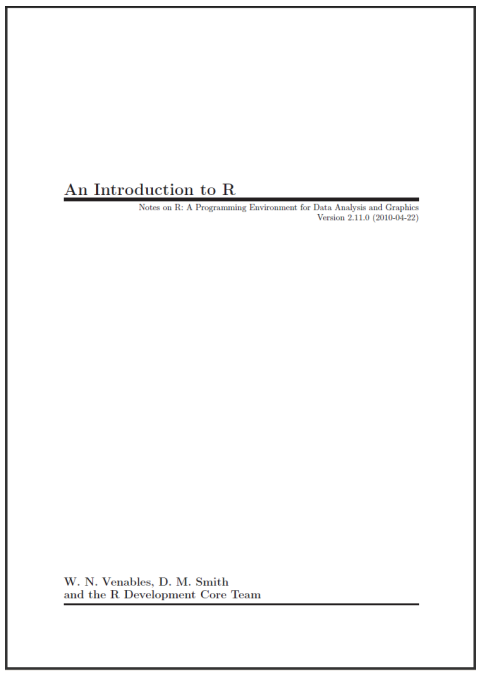
# The R Core Team

- 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
- Uwe Ligges – TU Dortmund University
- Thomas Lumley – University of Auckland
- 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

# 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` is `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 $n^{th}$ 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`

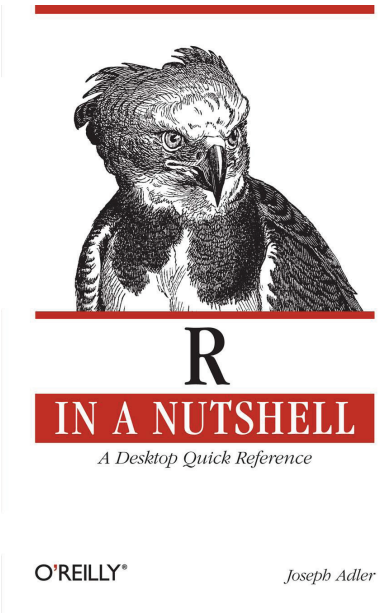
**Data selection and manipulation**

`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

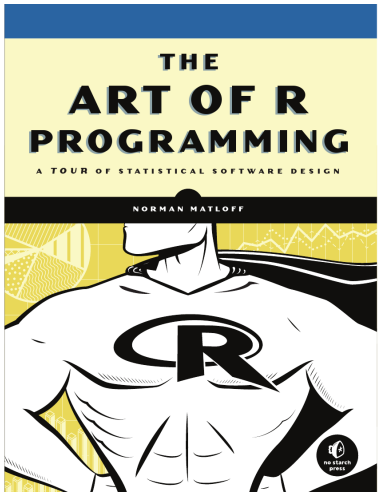
Definitely obtain these PDF files from the R homepage or a CRAN mirror

# R language references

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

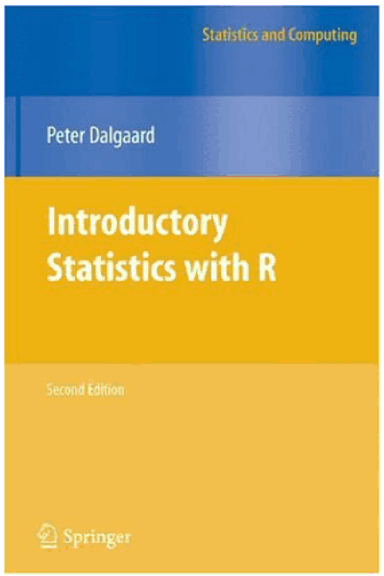



- The Art of R Programming
  - Norman Matloff
  - No Starch Press, 2011

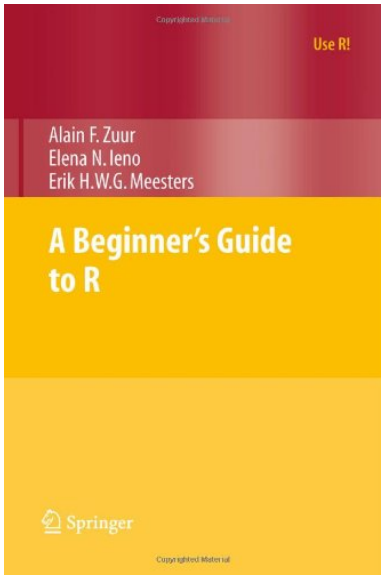


# Other introductory R texts

- Introductory Statistics with R 2nd Edition
  - P. Dalgaard
  - Springer, 2008




- 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`

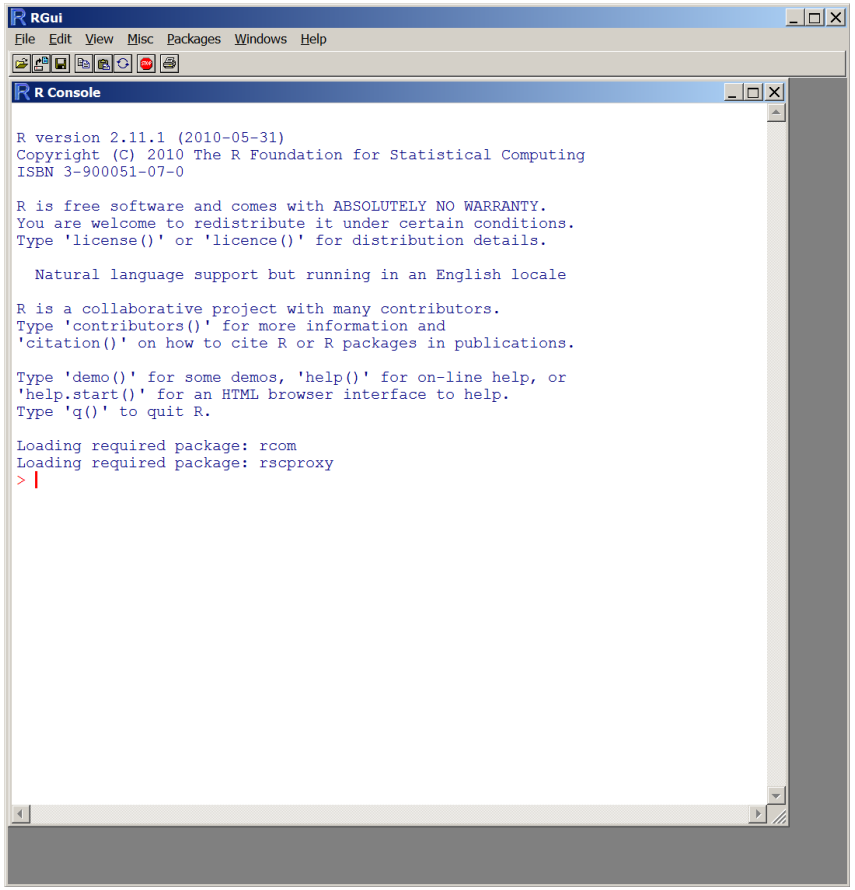
# Outline

# The R GUI

Running R in Windows

- Typically run Rgui.exe

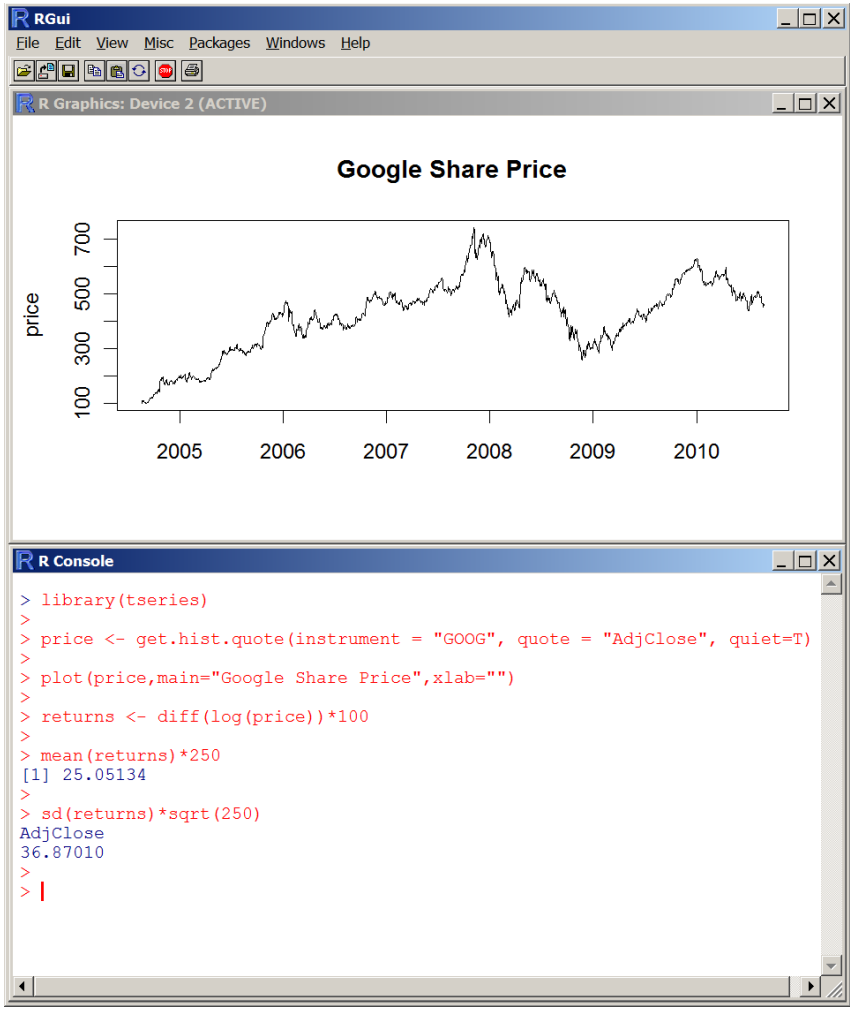

- Can also run R.exe from command prompt
- Or run Rterm.exe in batch mode




The R GUI on a Windows platform

# 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] 0.1323967 0.7617530
```

# 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"        "filename" "m"        "r"        "s"        "tab"
[8] "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,] 1.5250413  0.6412772  2.52021809
[2,] 1.6394314 -0.6197558  0.80289579
[3,] 0.8032637  0.2847256 -0.03179198

> 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
```

| | |
|---|---|
| from | starting value |
| to | 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 to 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.function"      "plot.hclust"        "plot.histogram"
[13] "plot.HoltWinters"   "plot.isoreg"        "plot.lm"
```

- generic functions implement simple polymorphism for S3 objects

# 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

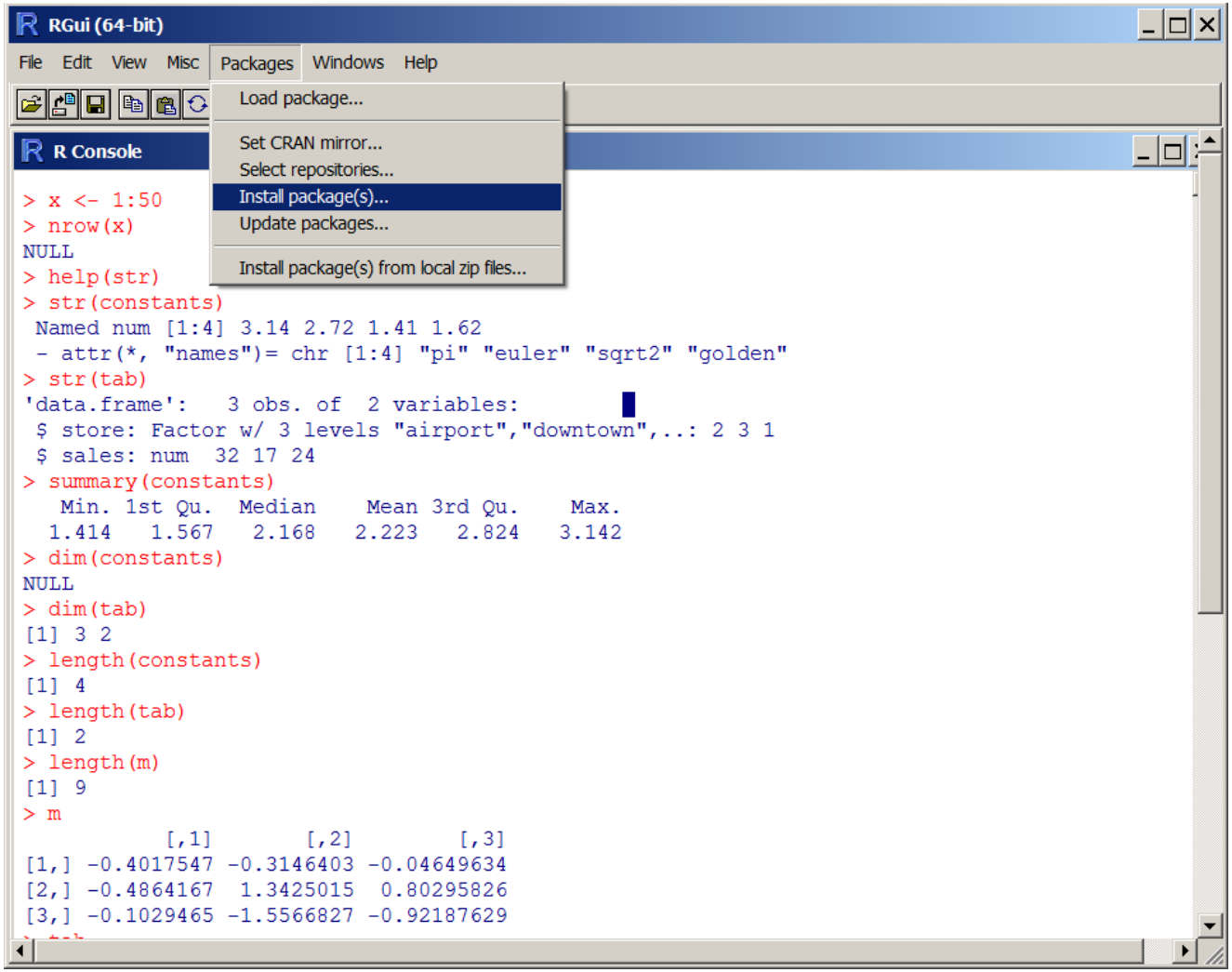
The `install.packages` function can be used to install contributed packages

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

# 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 |

# Writing R functions

One of the strengths of R is that it can easily be extended by writing new R functions; in fact, much of R is written in R

The syntax for defining a new R function is as follows:

```
name <- function(arg_1, arg_2, ..)  expression
```

**R Code: Create a user-defined function**

```
> percentChange <- function(x)
 {
   100*(x[-1]/x[-length(x)]-1)
 }
> sales <- c(100,105,110,105,100)
> percentChange(sales)

[1]  5.000000  4.761905 -4.545455 -4.761905
```

# Outline

# The list object

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

**R Code: Creating lists**

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

[1] "list"

> length(myList)

[1] 3

> myList

$pi
[1] 3.1416

$euler
[1] 2.7183

$golden
[1] 1.618

> diverseList <- list(magic=myList,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**

```
> myList[2]

$euler
[1] 2.7183

> myList[[2]]

[1] 2.7183

> myList[["pi"]]

[1] 3.1416

> myList$golden

[1] 1.618

> diverseList[[3]][2]

[1] "OR"
```

# The data.frame object

- A data.frame is a 2D matrix-like object where the *columns* can be of *different* classes; for example:
  - column with dates
  - column with characters
  - column with integers
  - column with numeric

**R Code: The data.frame object**

```
> data(batting.2008)
> class(batting.2008)

[1] "data.frame"

> dim(batting.2008)

[1] 1384   32

> batting.2008[1:2,1:4]

  nameLast nameFirst weight height
1    Abreu     Bobby    200     72
2     Alou    Moises    190     75

> class(batting.2008[,2])

[1] "character"

> class(batting.2008[,3])

[1] "integer"

> class(batting.2008[,4])

[1] "numeric"
```

# The head and tail functions

**R Code: The head and tail functions**

```
> args(getS3method("head","data.frame"))

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

> data(dow30)
> head(dow30)

  symbol       Date  Open  High   Low Close  Volume Adj.Close
1    MMM 2009-09-21 73.91 74.68 73.91 74.54 2560400     74.54
2    MMM 2009-09-18 75.12 75.25 74.50 74.62 4387900     74.62
3    MMM 2009-09-17 75.34 75.45 74.50 74.89 3371500     74.89
4    MMM 2009-09-16 74.76 75.49 74.50 75.38 2722500     75.38
5    MMM 2009-09-15 74.63 74.88 74.00 74.68 3566900     74.68
6    MMM 2009-09-14 73.72 74.64 73.42 74.56 3466400     74.56

> tail(dow30,3)

     symbol       Date  Open  High   Low Close   Volume Adj.Close
7480    DIS 2008-09-24 32.59 32.59 31.63 31.77 13600300     31.30
7481    DIS 2008-09-23 32.88 33.32 32.15 32.53 13450900     32.05
7482    DIS 2008-09-22 33.85 34.05 32.84 32.91 18394300     32.42
```

# 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
- Since data.frames are stored internally as lists, their columns can be accessed with the $ operator as well

# 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
- Since data.frames are stored internally as lists, their columns can be accessed with the $ operator as well

**R Code: Indexing 2D objects**

```
> dow30[1:4,c("symbol","Date","Close")]

  symbol       Date Close
1    MMM 2009-09-21 74.54
2    MMM 2009-09-18 74.62
3    MMM 2009-09-17 74.89
4    MMM 2009-09-16 75.38

> head(dow30[-1,c(1,2,6)],3)

  symbol       Date Close
2    MMM 2009-09-18 74.62
3    MMM 2009-09-17 74.89
4    MMM 2009-09-16 75.38

> dow30[dow30[,"Volume"]>1.5e9,
   c("symbol","Date","Close","Volume")]

     symbol       Date Close     Volume
2047      C 2009-08-07  3.85 1898814600
2049      C 2009-08-05  3.58 2672492000
2159      C 2009-02-27  1.50 1868209400
```

# Factors

A *factor* is a data type for representing categorial data

**R Code: Create a factor variable**

```
> pet.str <- c("dog","cat","cat","dog","fish","dog","rabbit")
> pets <- as.factor(pet.str)
> pets

[1] dog    cat    cat    dog    fish   dog    rabbit
Levels: cat dog fish rabbit

> as.numeric(pets)

[1] 2 1 1 2 3 2 4

> levels(pets)

[1] "cat"    "dog"    "fish"   "rabbit"
```

- factors are encoded as integers (starting at 1)
- the *levels* of a factor variable contain the categorical labels

# The working directory

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

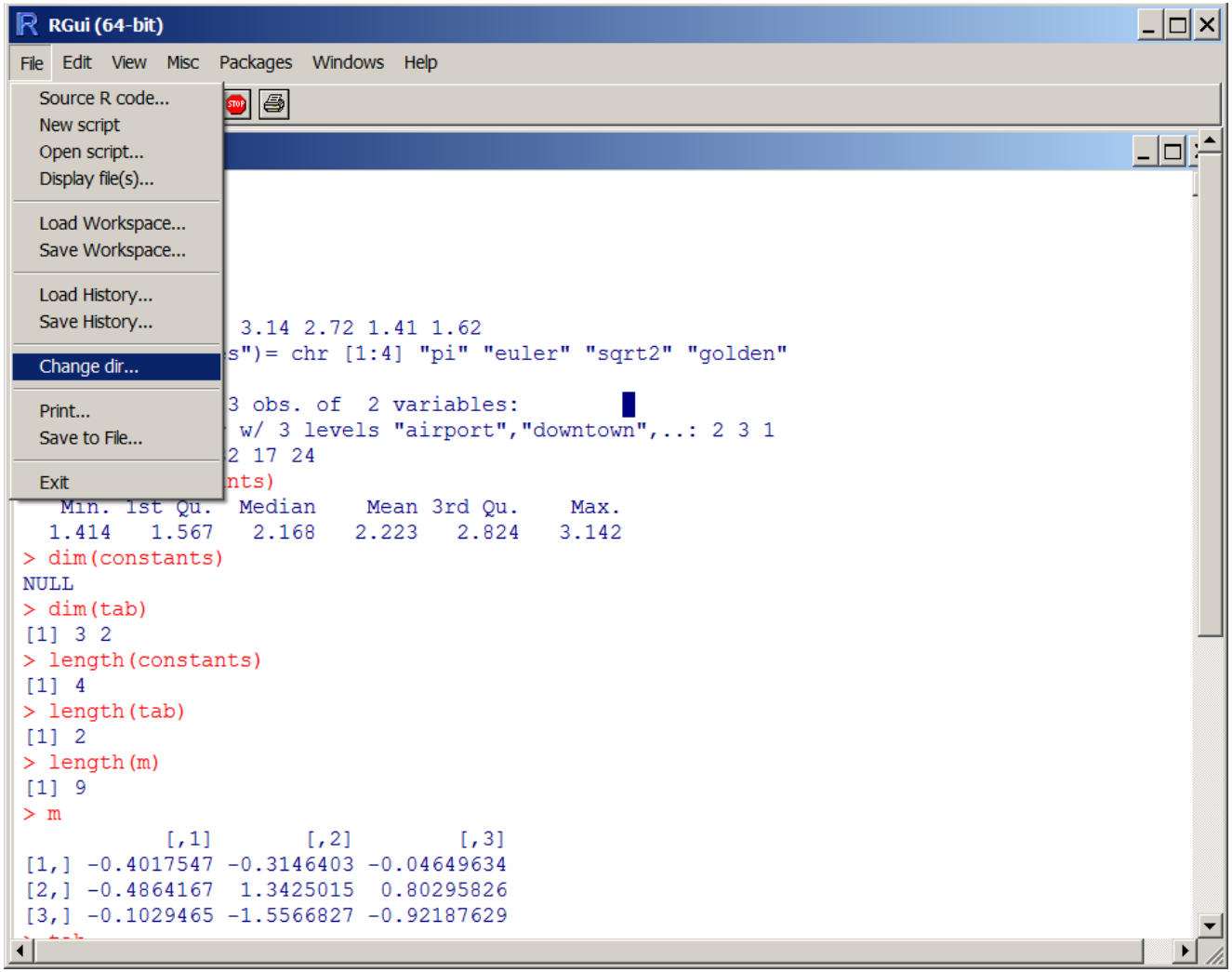
**R Code: Getting and setting the working directory**

```
> (my.wd <- getwd())

[1] "C:/Rprojects/UW/RIntro"

> setwd(R.home())
> getwd()

[1] "C:/R/R-2.15.1"

> setwd(my.wd)
> getwd()

[1] "C:/Rprojects/UW/RIntro"
```

- 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

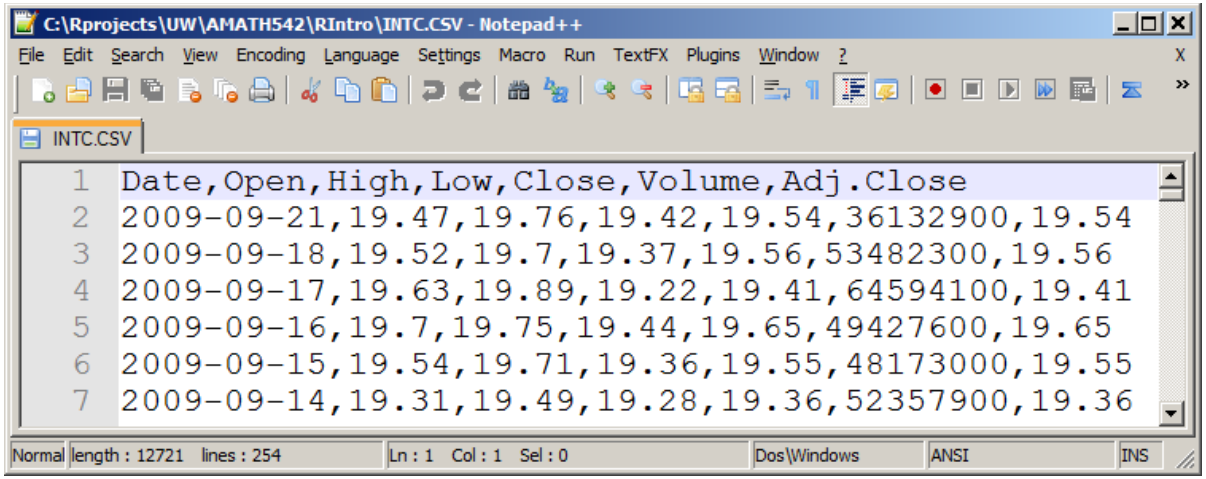
The read.table function is used *extensively* to load data into R

**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", text)
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=TRUE,sep=",",as.is=TRUE)
> dat[1:5,]

        Date  Open  High   Low Close   Volume Adj.Close
1 2009-09-21 19.47 19.76 19.42 19.54 36132900     19.54
2 2009-09-18 19.52 19.70 19.37 19.56 53482300     19.56
3 2009-09-17 19.63 19.89 19.22 19.41 64594100     19.41
4 2009-09-16 19.70 19.75 19.44 19.65 49427600     19.65
5 2009-09-15 19.54 19.71 19.36 19.55 48173000     19.55
```

# Writing text files

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

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

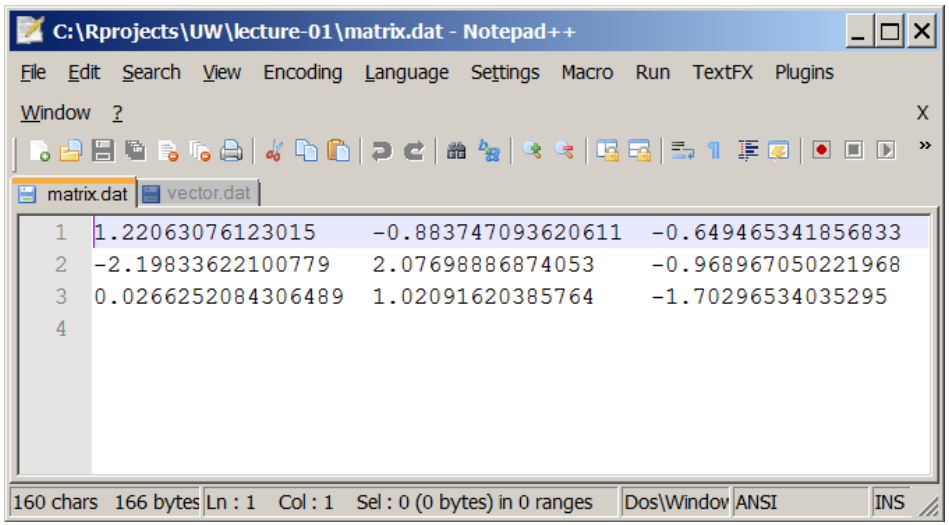
```
> args(write.table)

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
```

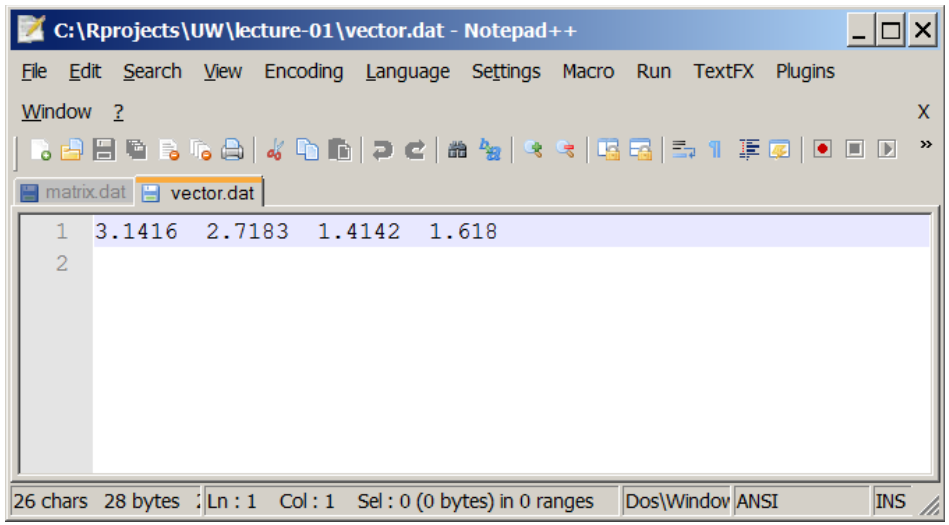
| | |
|---|---|
| `x` | object to be written (data.frame, matrix, vector) |
| `file` | file name (with path if necessary) |
| `sep` | column separation character (e.g. comma or tab) |
| `row.names` | write row names (T/F) |
| `col.names` | write col names (T/F) |

# 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)
> file.info(list.files(pattern="[.][d][a][t]",full.names=T))[,c("size","mtime")]

              size                mtime
./matrix.dat   164 2012-08-31 08:15:15
./vector.dat    28 2012-08-31 08:15:15
```




- note the use of the `list.files` function, the `file.info` function, and the use of regular expressions

# 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 |

# 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

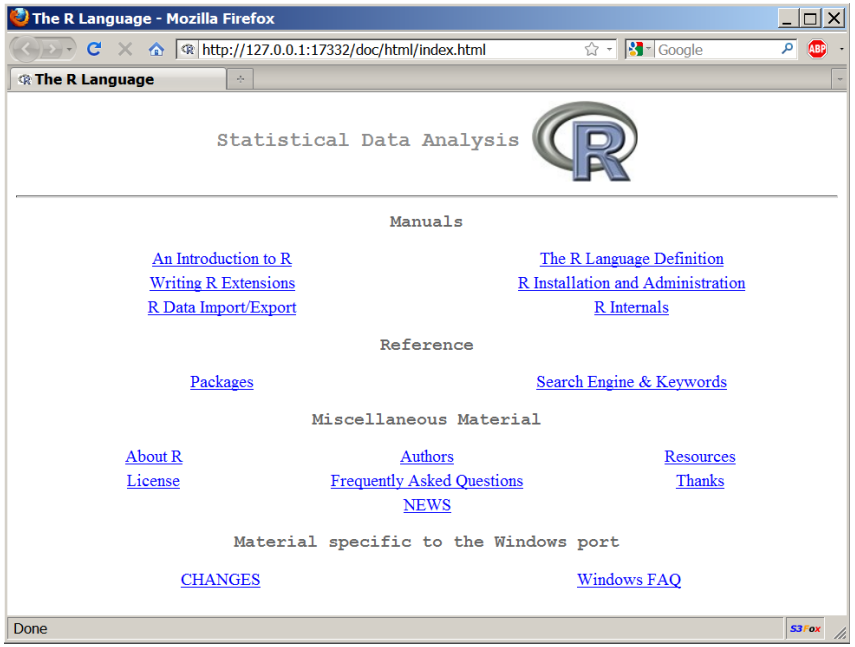
- Data in an S4 class is organized into *slots*; slots can be accessed using:
  - the @ operator: object@name
  - the slot function: slot(object,name)
- Methods for an S4 class can be queried with the showMethods function
  - showMethods(class = "fGARCH")
- Methods can be retrieved/viewed with the getMethod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

# Outline

# 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://127.0.0.1:12534/doc/html/index.html' yourself
```

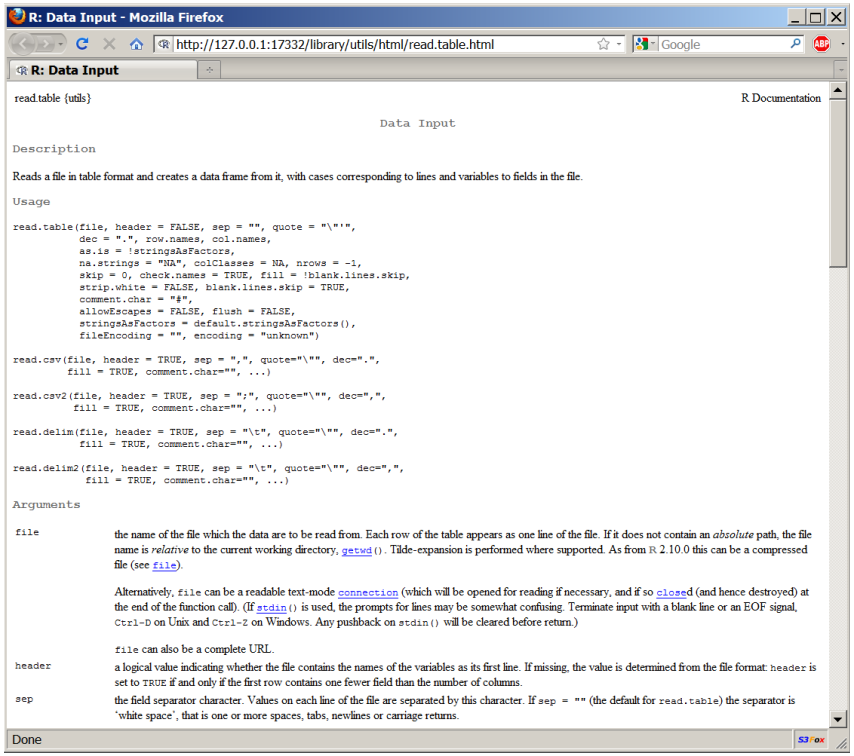
# The help function

Obtain help on a particular topic via the help function

- help(topic)
- ?topic

**R Code: Topic help**

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



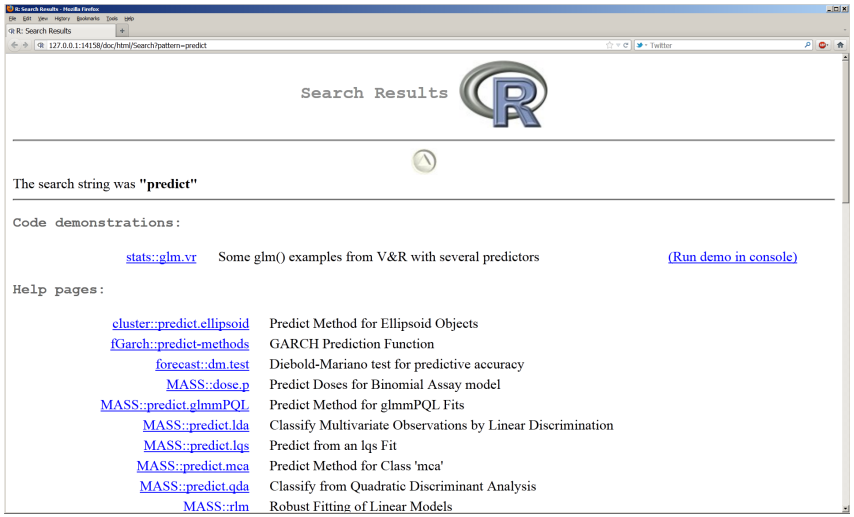
# The help.search function

Search help for a particular topic via the help.search function

- help.search(topic)
- ??topic

**R Code: Search help**

```
> ??predict
```

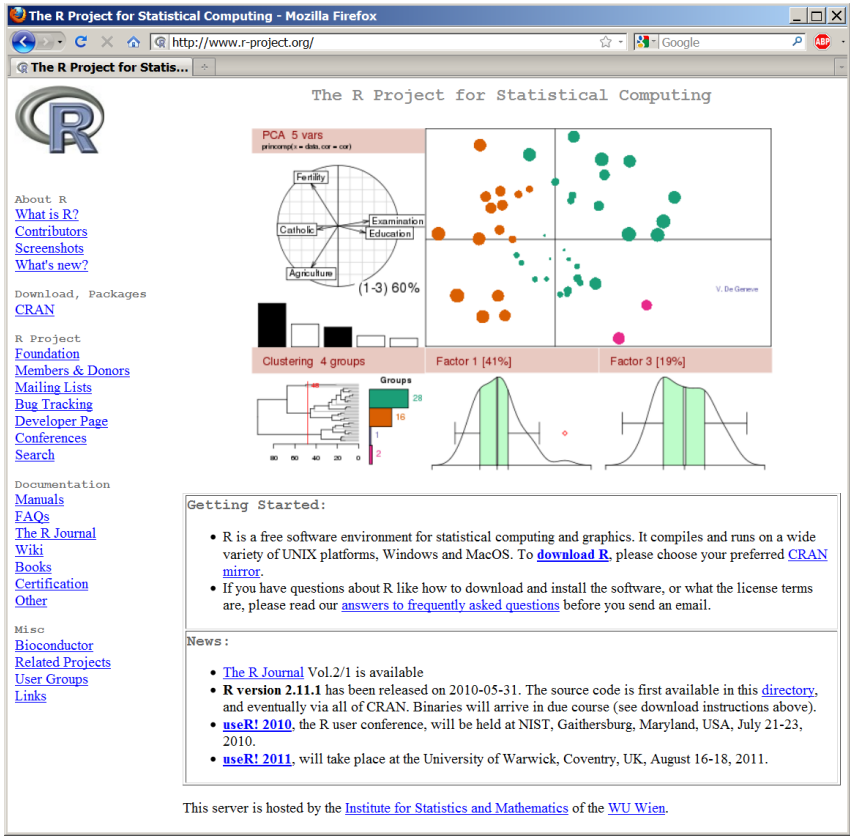


# Outline

# R Homepage

`http://www.r-project.org`

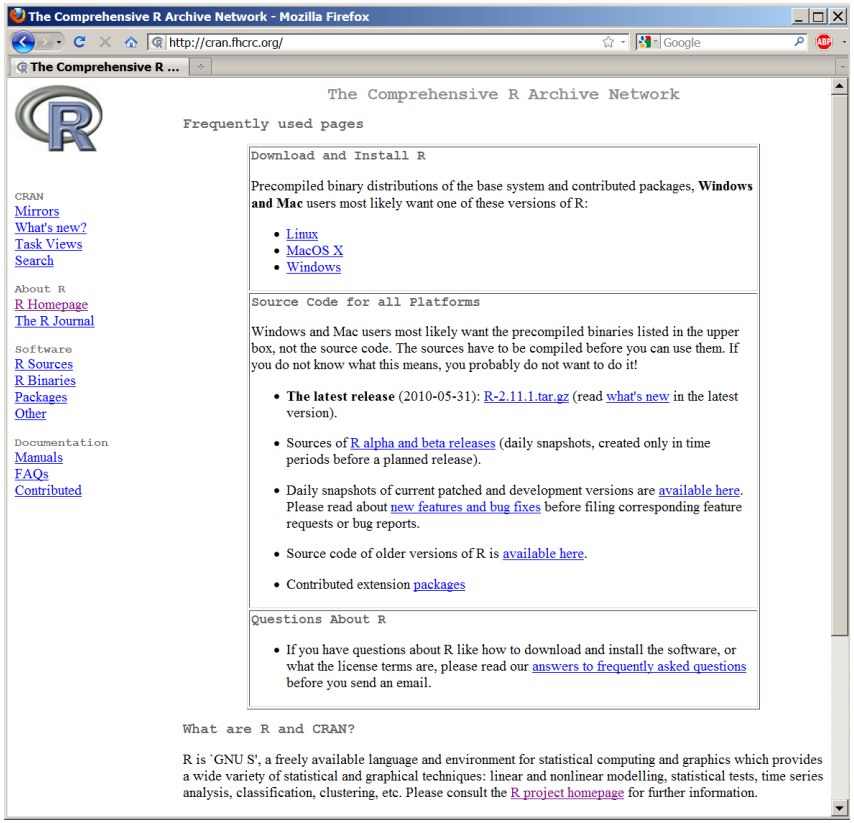
- List of CRAN mirror sites
- Manuals
- FAQs
- Mailing Lists
- Links



# CRAN - Comprehensive R Archive Network

`http://cran.fhcrc.org`

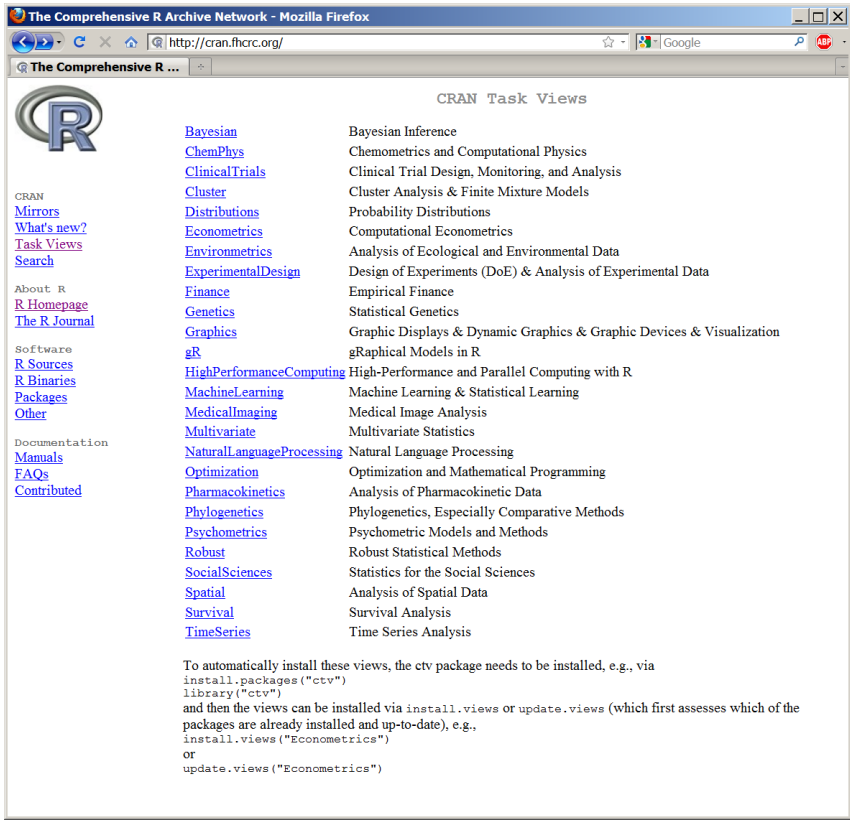
- CRAN Mirrors
  - About 88 sites worldwide
  - About 19 sites in US
- R Binaries
- R Packages
- R Sources
- Task Views



# CRAN Task Views

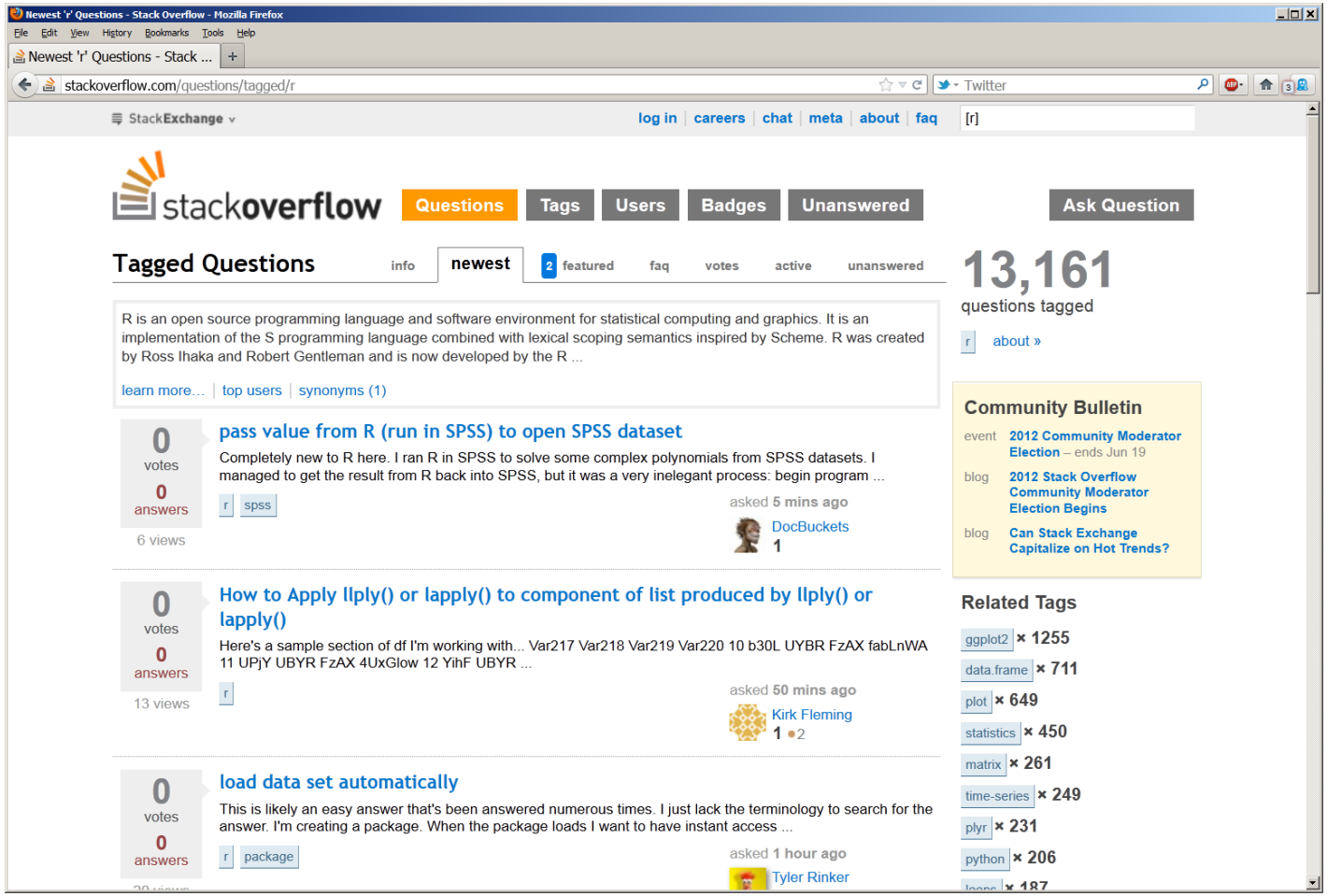
Organizes the 3500+ R packages by application

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



# Stackoverflow

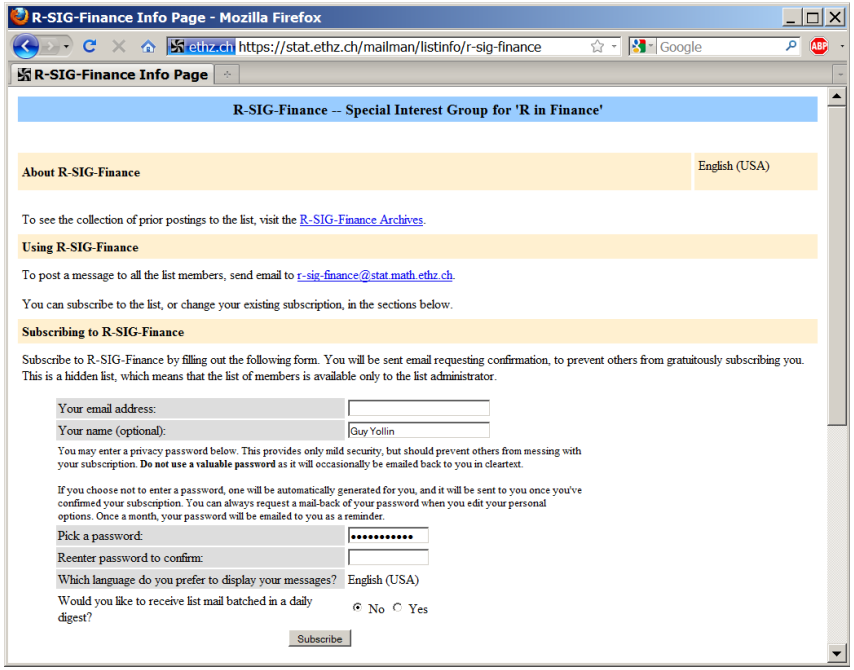
Stackoverflow has become the primary resource for help with R




http://stackoverflow.com/

# R-SIG-FINANCE

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

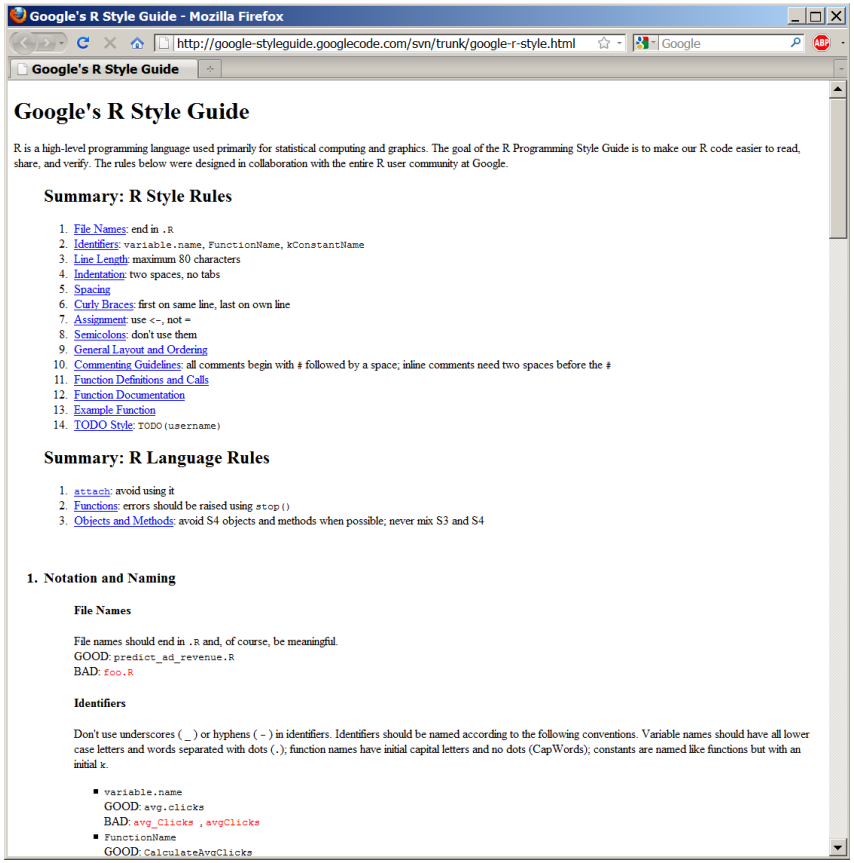



`https://stat.ethz.ch/mailman/listinfo/r-sig-finance`

# Google's R Style Guide

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

- Naming convention
- Coding Syntax
- Program Organization



# 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)

# Programming in R

Online R programming 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

# Statconn

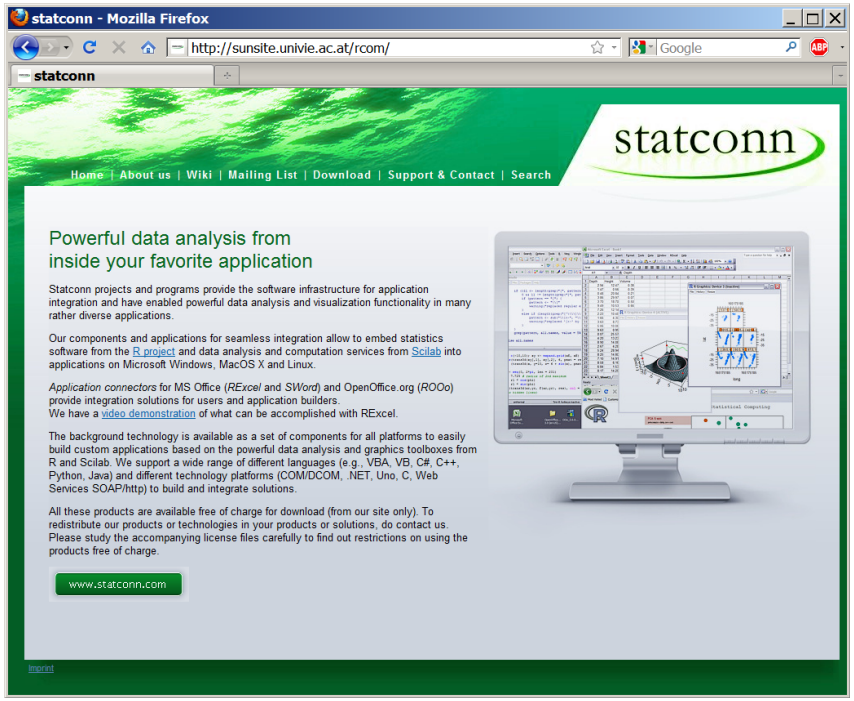
`http://rcom.univie.ac.at`

COM interface for R connectivity

- Excel
- Word
- C#
- VB
- Delphi

Download site for RAndFriends

- R
- Statconn
- Notepad++



# Other useful R sites

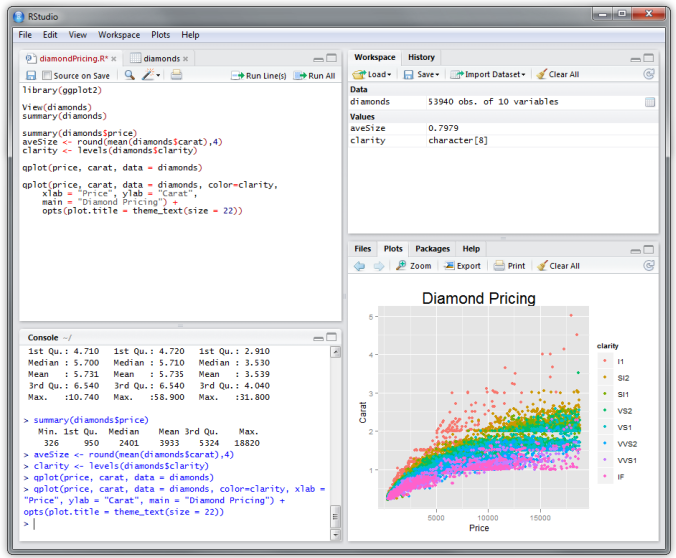
| | |
|---|---|
| R Bloggers | Aggregation of about 290 R blogs<br>• `http://www.r-bloggers.com` |
| R Site Search | Search R function help, vignettes, R-help<br>• `http://finzi.psych.upenn.edu/search.html` |
| R Seek | R specific search site<br>• `http://www.rseek.org/` |
| R Graph Gallery | Examples of many possible R graphs<br>• `http://addictedtor.free.fr/graphiques` |
| Revolution Blog | Blog from David Smith of Revolution<br>• `http://blog.revolutionanalytics.com` |
| Inside-R | R community site by Revolution Analytics<br>• `http://www.inside-r.org` |

# Outline

# RStudio

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

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



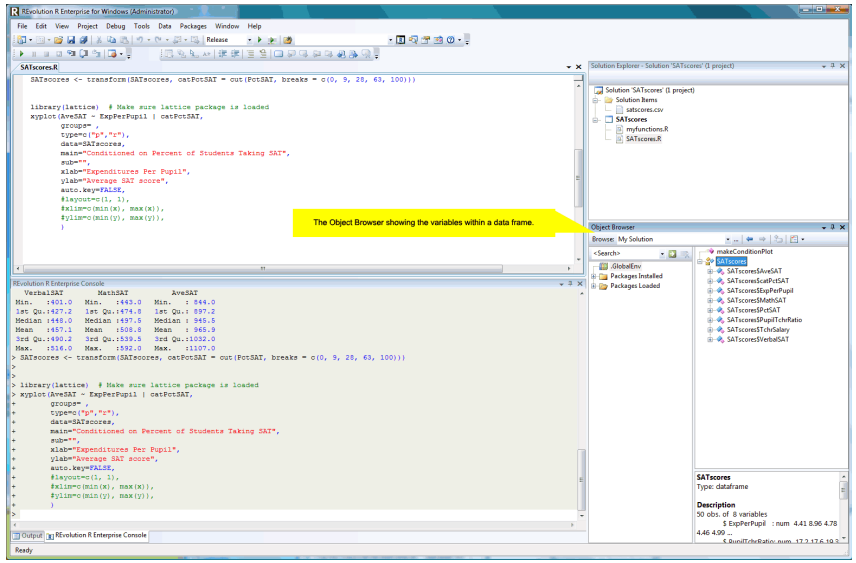

RStudio also provides a server-based version (R running in the cloud)

# 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

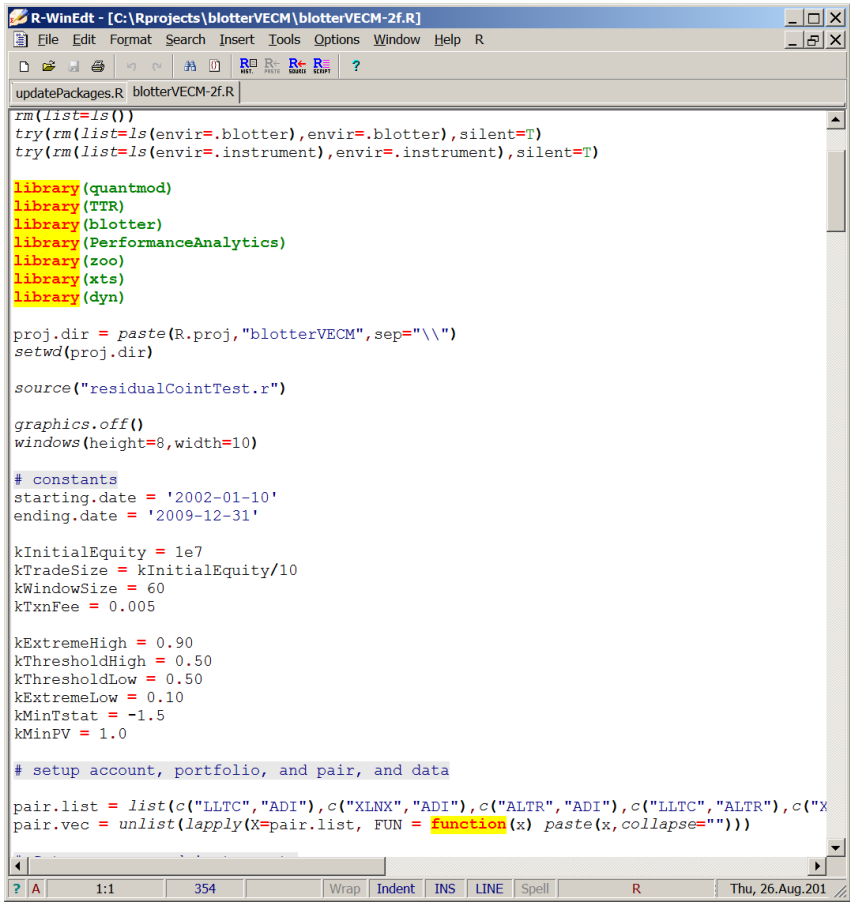



`http://www.revolutionanalytics.com`

# WinEdt and R-Sweave

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

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



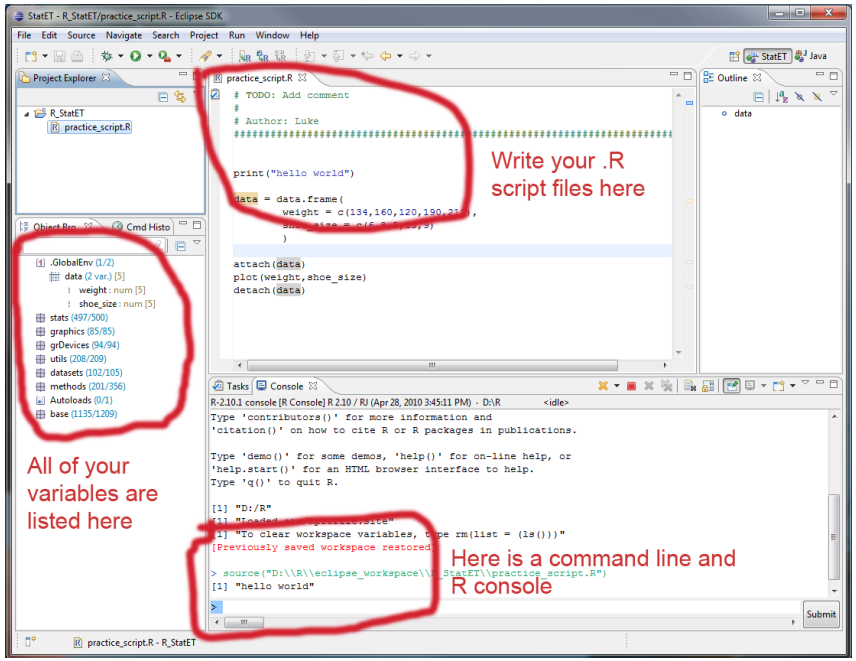

http://www.winedt.com

http://www.winedt.org/Config/modes/R-Sweave.php

# StatET - An Eclipse Plug-In for R

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

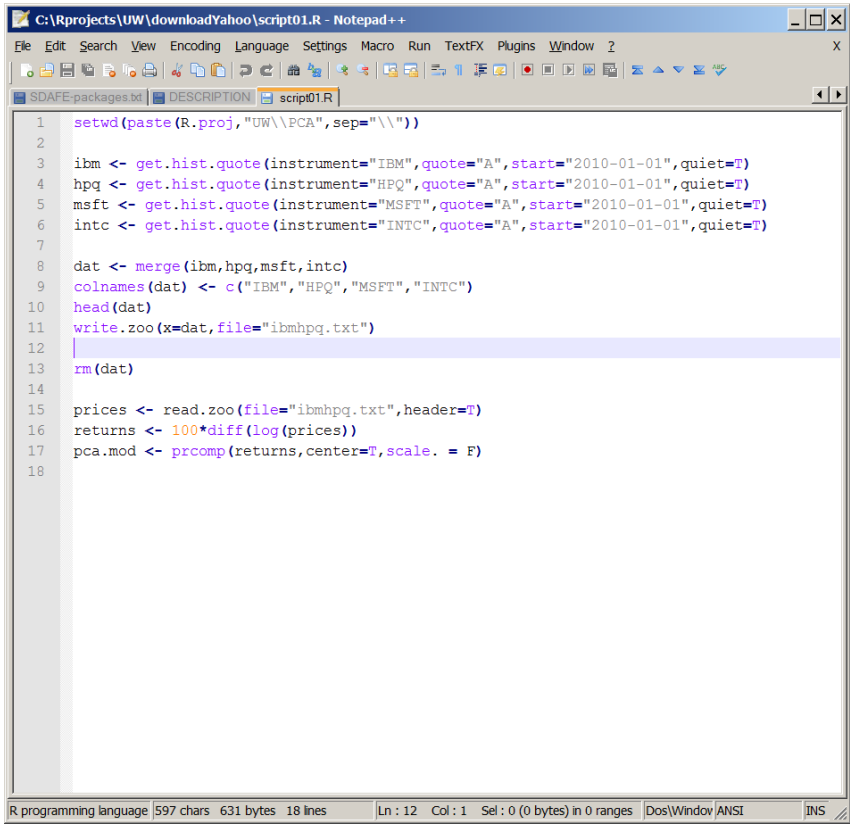
- R language highlighting
- Paste/Source code to R
- Source code debugger
- Supports R in SDI mode
- Excellent documentation by Longhow Lam




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

# 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`

# 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`

# 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**

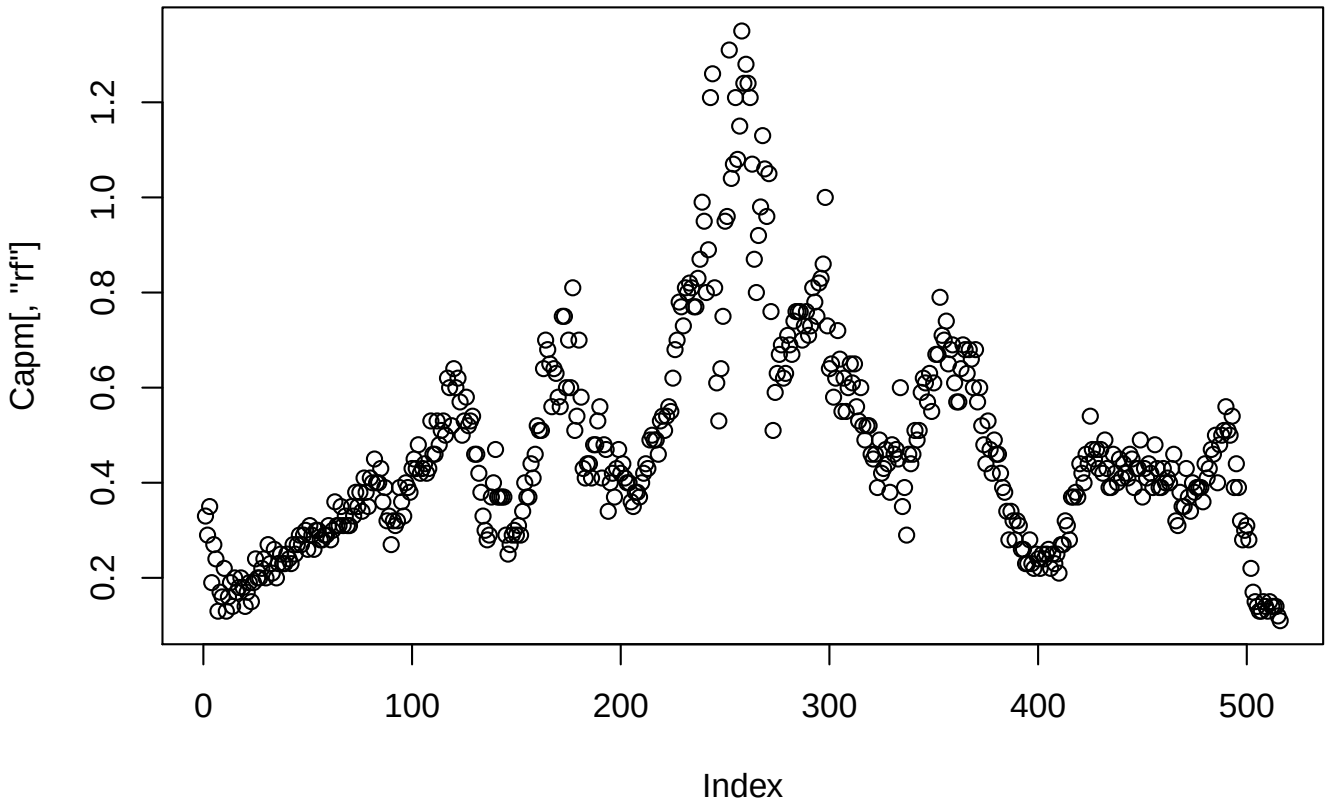
```
> args(plot.default)

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**

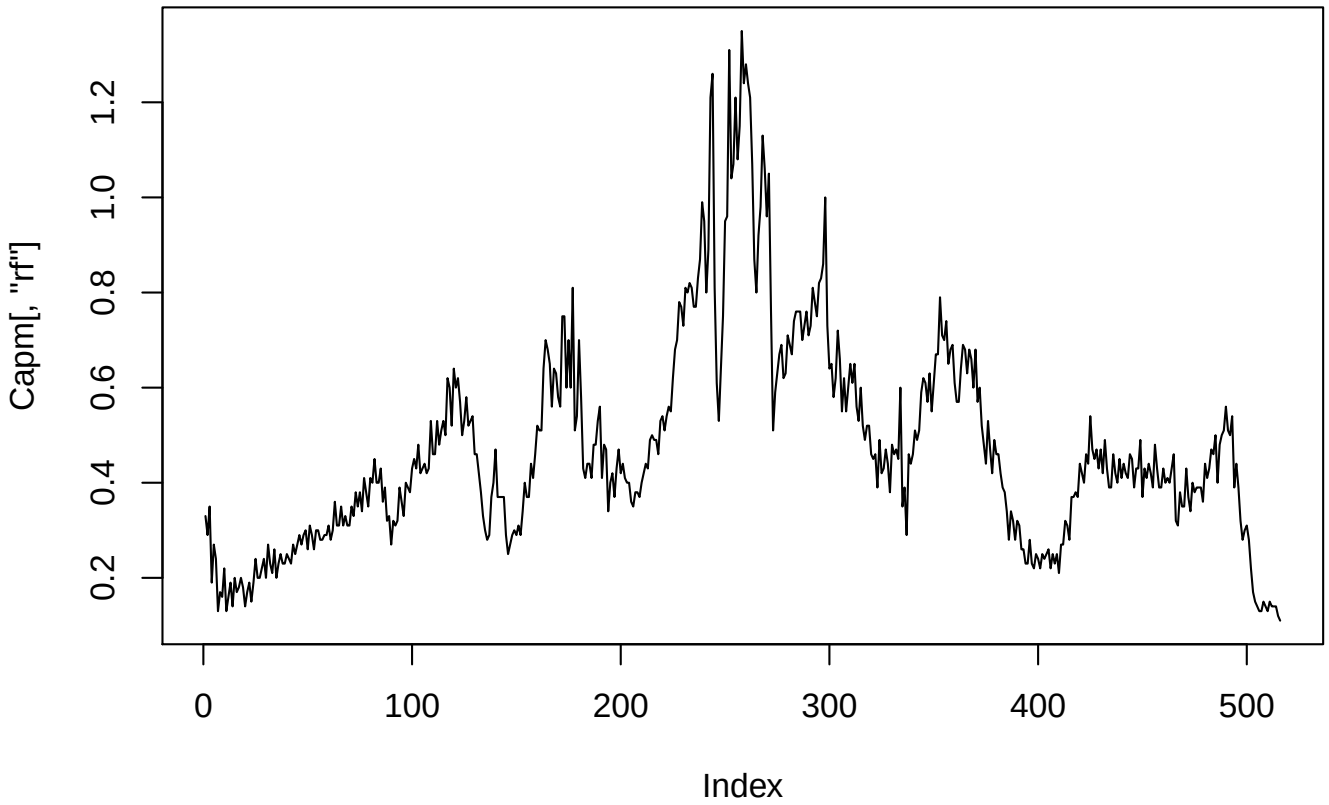
```
> library(Ecdat)
> data(Capm)
> plot(Capm[,"rf"])
```



# The plot function

**R Code: Plot lines**

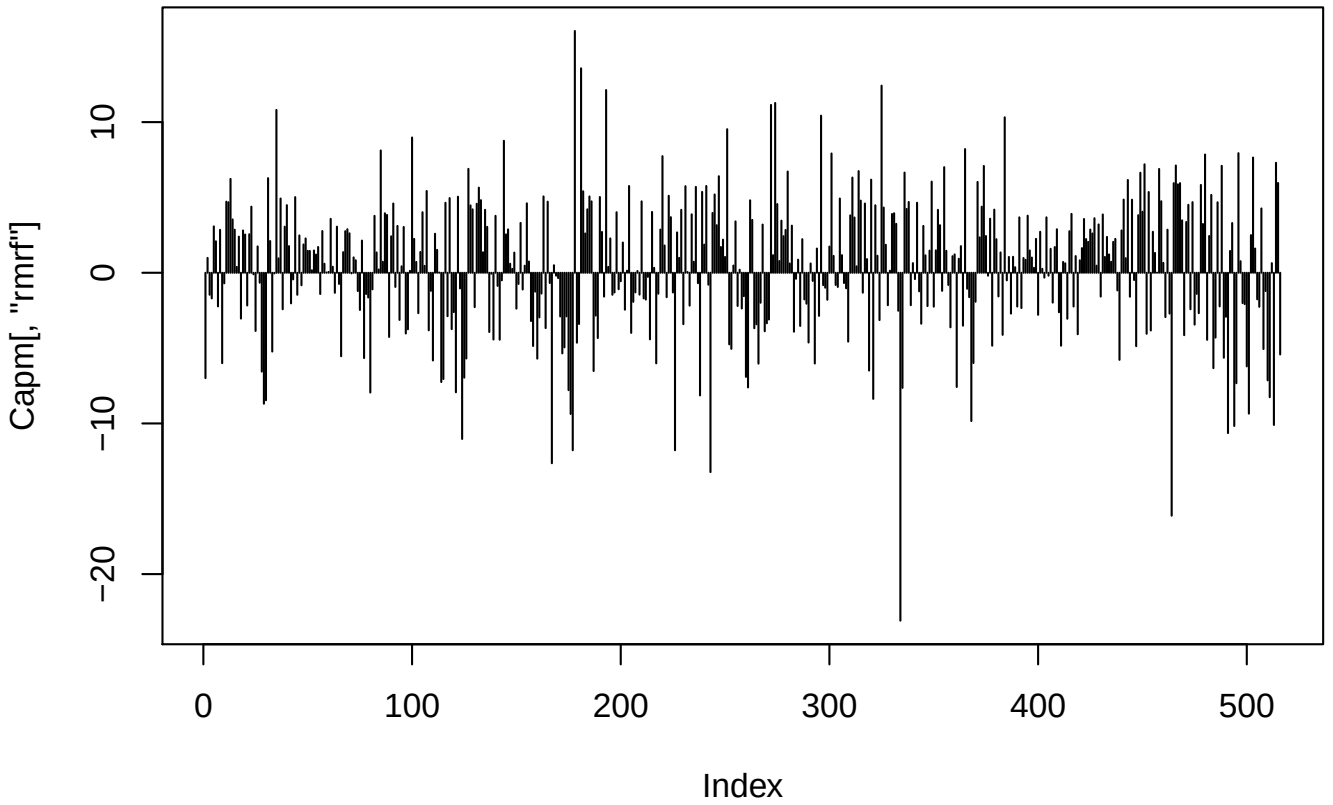
```
> plot(Capm[,"rf"],type="l")
```



# The plot function

**R Code: Plot bars**

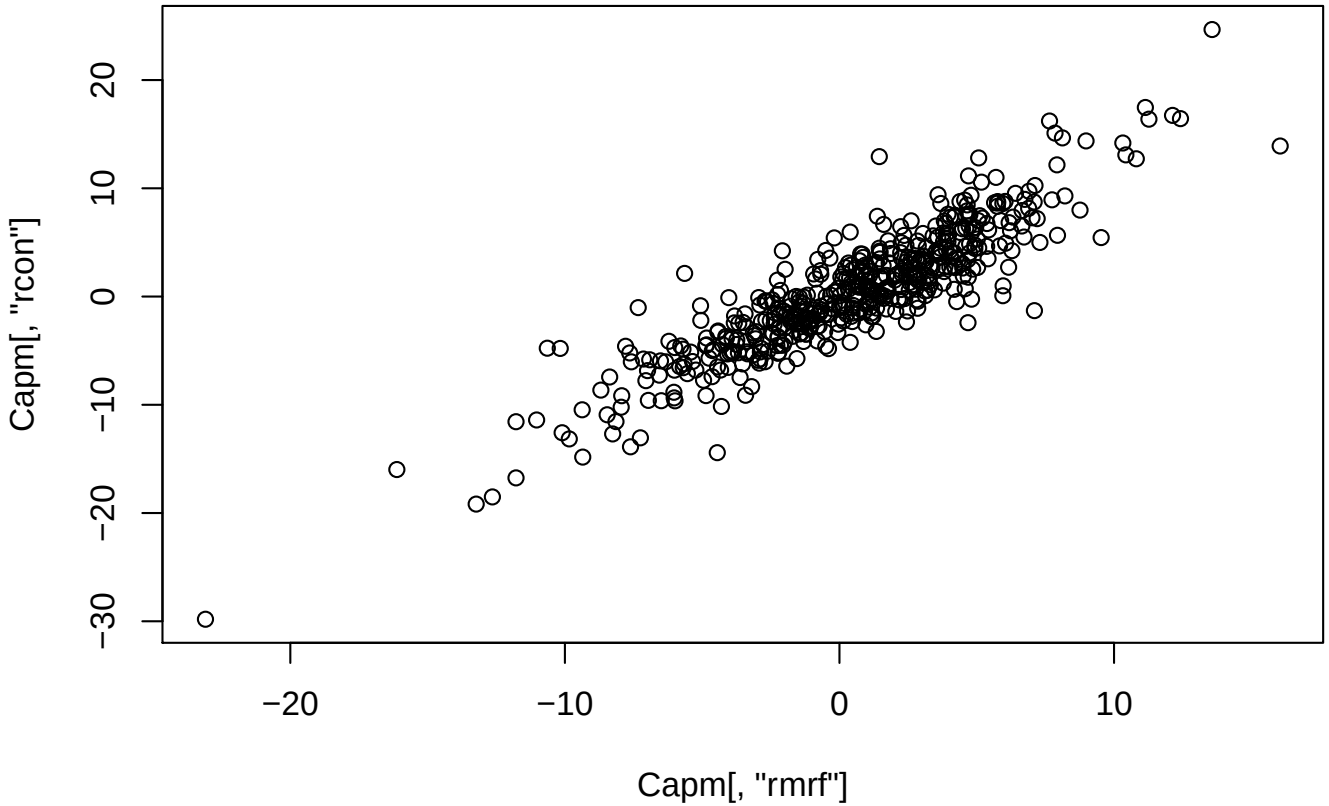
```
> plot(Capm[,"rmrf"],type="h")
```



# The plot function

**R Code: XY plot**

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



# 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**

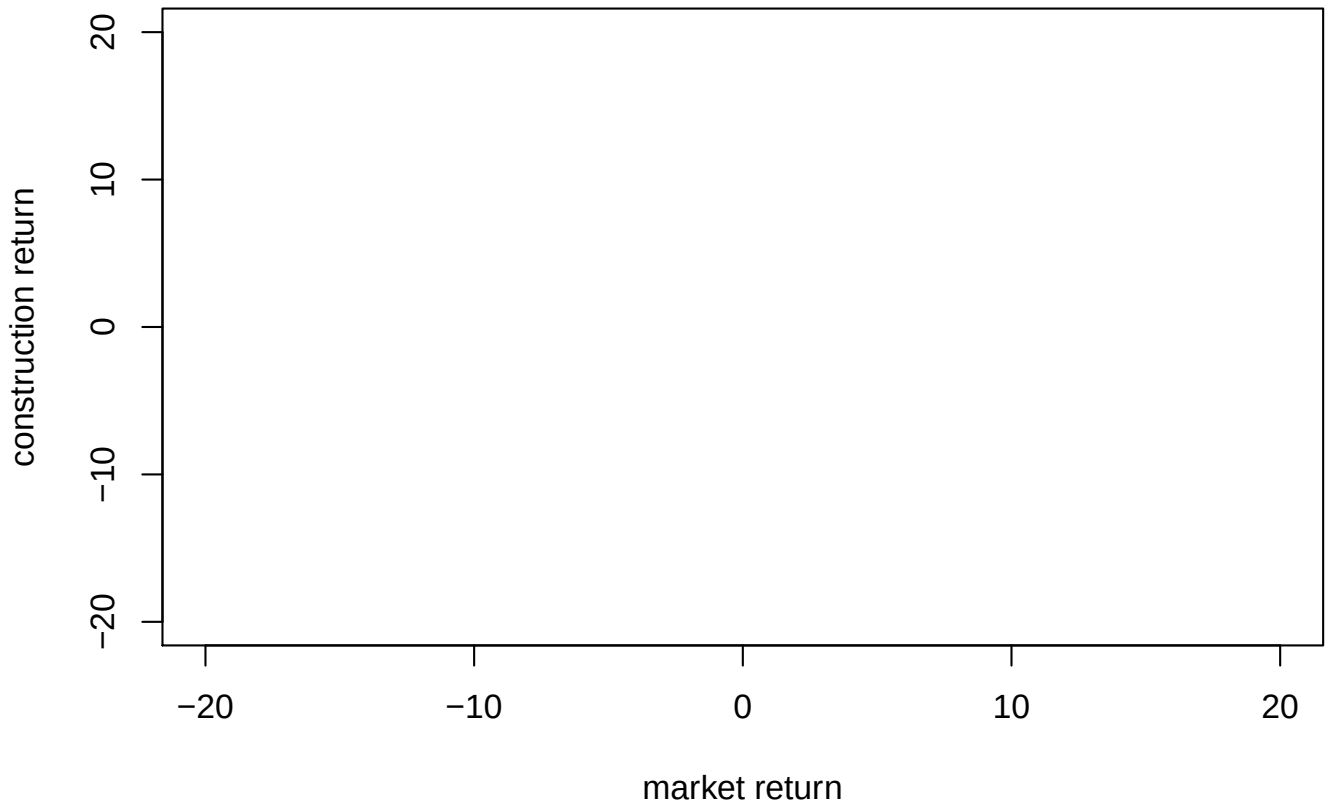
```
> args(text.default)

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**

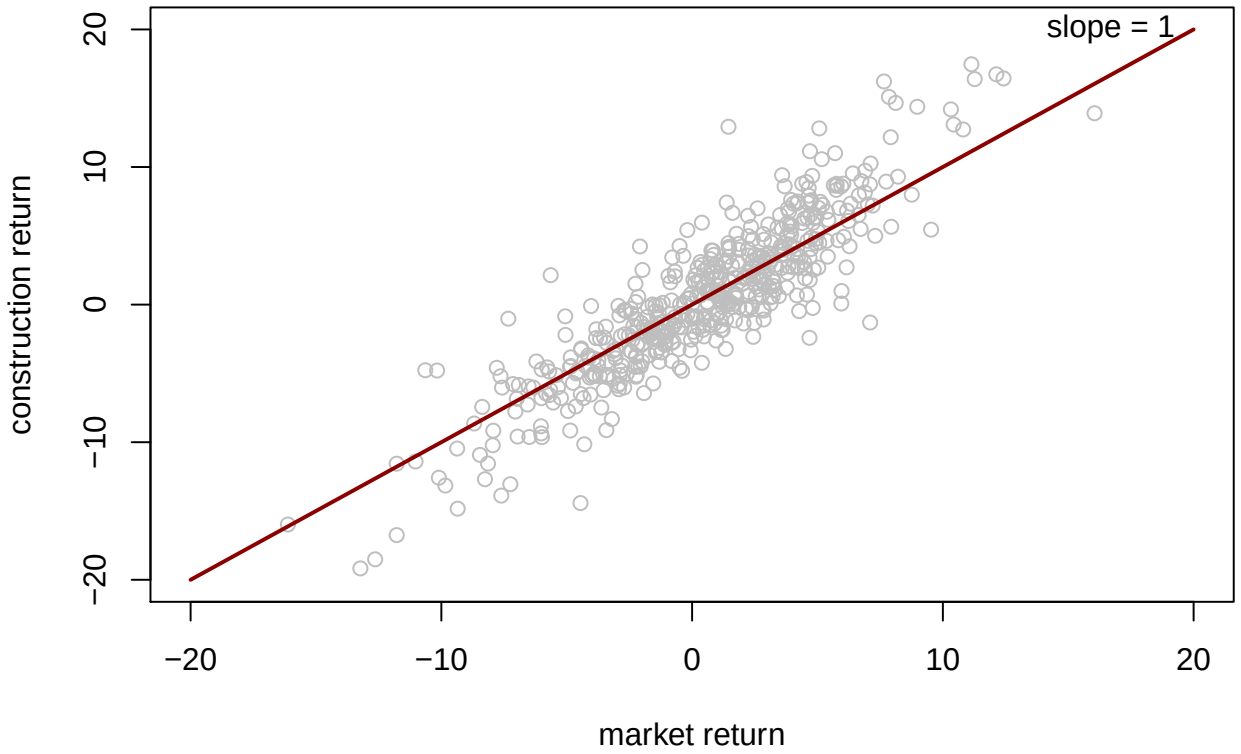
```
> plot(0,xlim=c(-20,20),ylim=c(-20,20),type="n",
   xlab="market return",ylab="construction return")
```



# 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"],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", xname = "x", xlab = xname, 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"      "ylbias"
```

# 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"), text.font = NULL,
    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

# 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

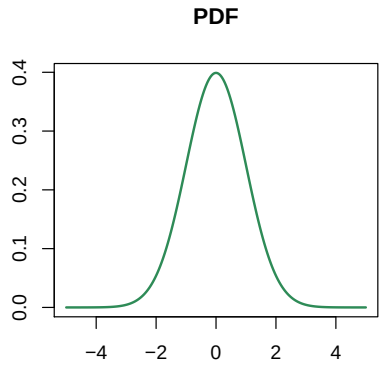
  The set of all possible values of a random variable along with their associated probabilities constitutes a *probability distribution* of the random variable

# PDFs and CDFs

- Probability density function (PDF)

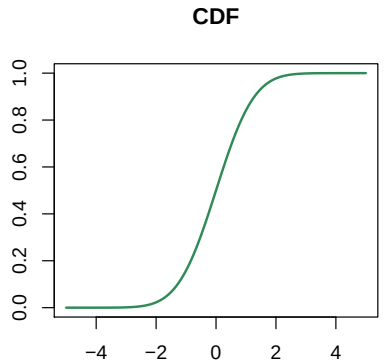
$$Pr(a < Y < b) = \int_a^b f_Y(y)$$

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

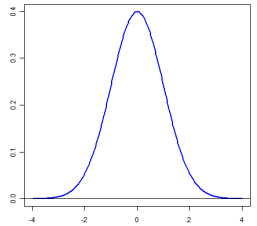
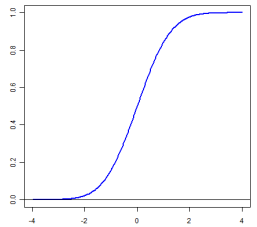
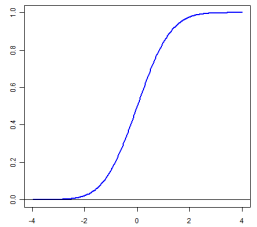



- Cumulative distribution function (CDF)

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



# PDF, CDF, quantile functions

| Function | General Notation | Normal Notation | R | Excel | Graph |
|---|---|---|---|---|---|
| pdf | $f(x)$ | $\phi(z)$ | dnorm | NORMDIST |  |
| cdf | $F(x)$ | $\Phi(z)$ | pnorm | NORMDIST |  |
| quantile | $F^{-1}(x)$ | $\Phi^{-1}(z)$ | qnorm | NORMINV |  |

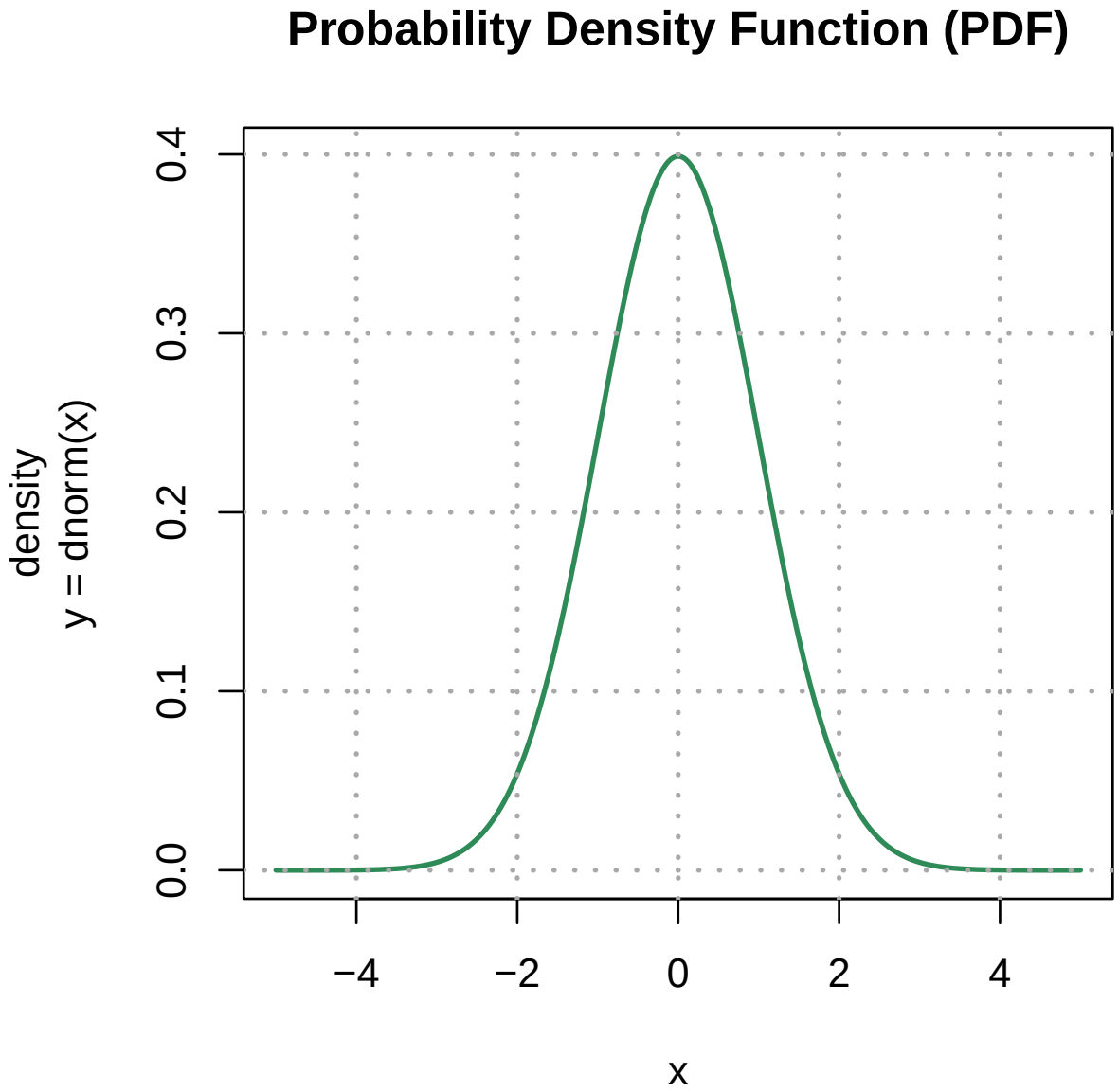
# Normal distribution PDF function: dnorm

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="x",ylab="density\ny = dnorm(x)")
> grid(col="darkgrey",lwd=2)
> title(main="Probability Density Function (PDF)")
```

# Normal distribution PDF function: dnorm




Others:

dt

dstd

dsstd

dged

dsged

dst

dmst

dct

# Normal distribution CDF functions: pnorm and qnorm

pnorm computes the normal CDF:

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

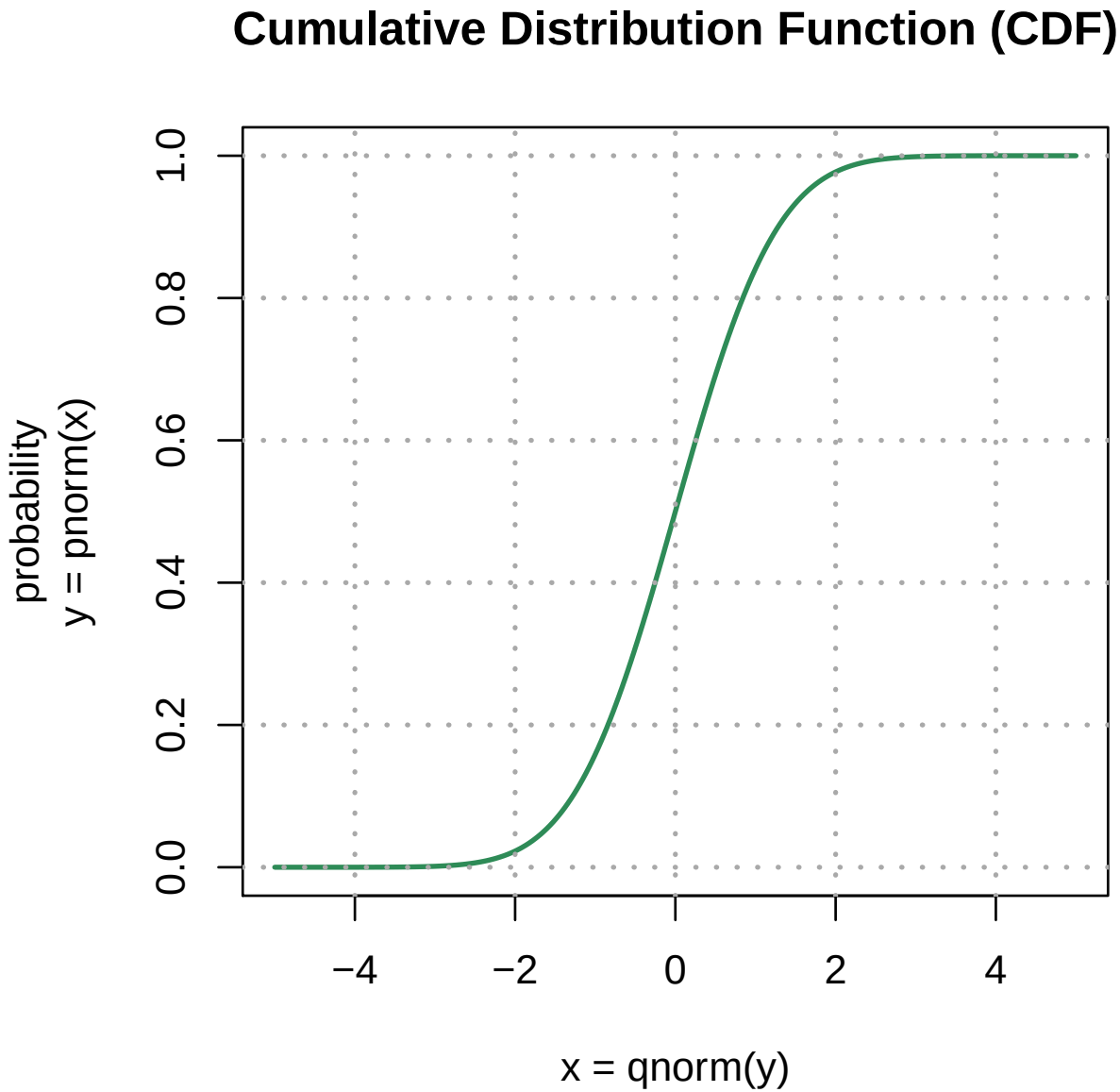
qnorm computes the inverse of the normal CDF (i.e. quantile):

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

**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="x = 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




Others:

pt

pstd

psstd

pged

psged

pst

pmst

pct

# 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**

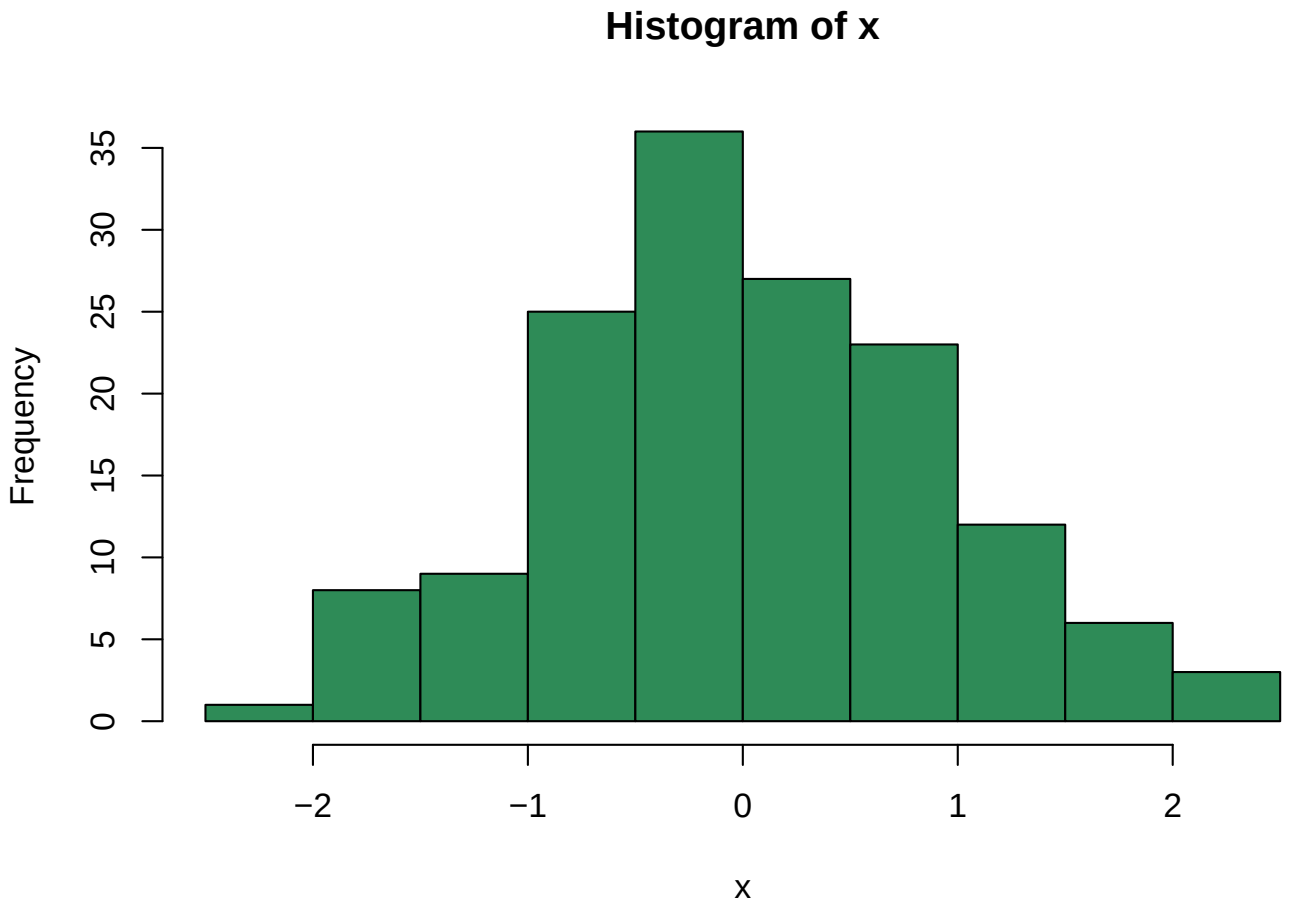
```
> args(hist.default)

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

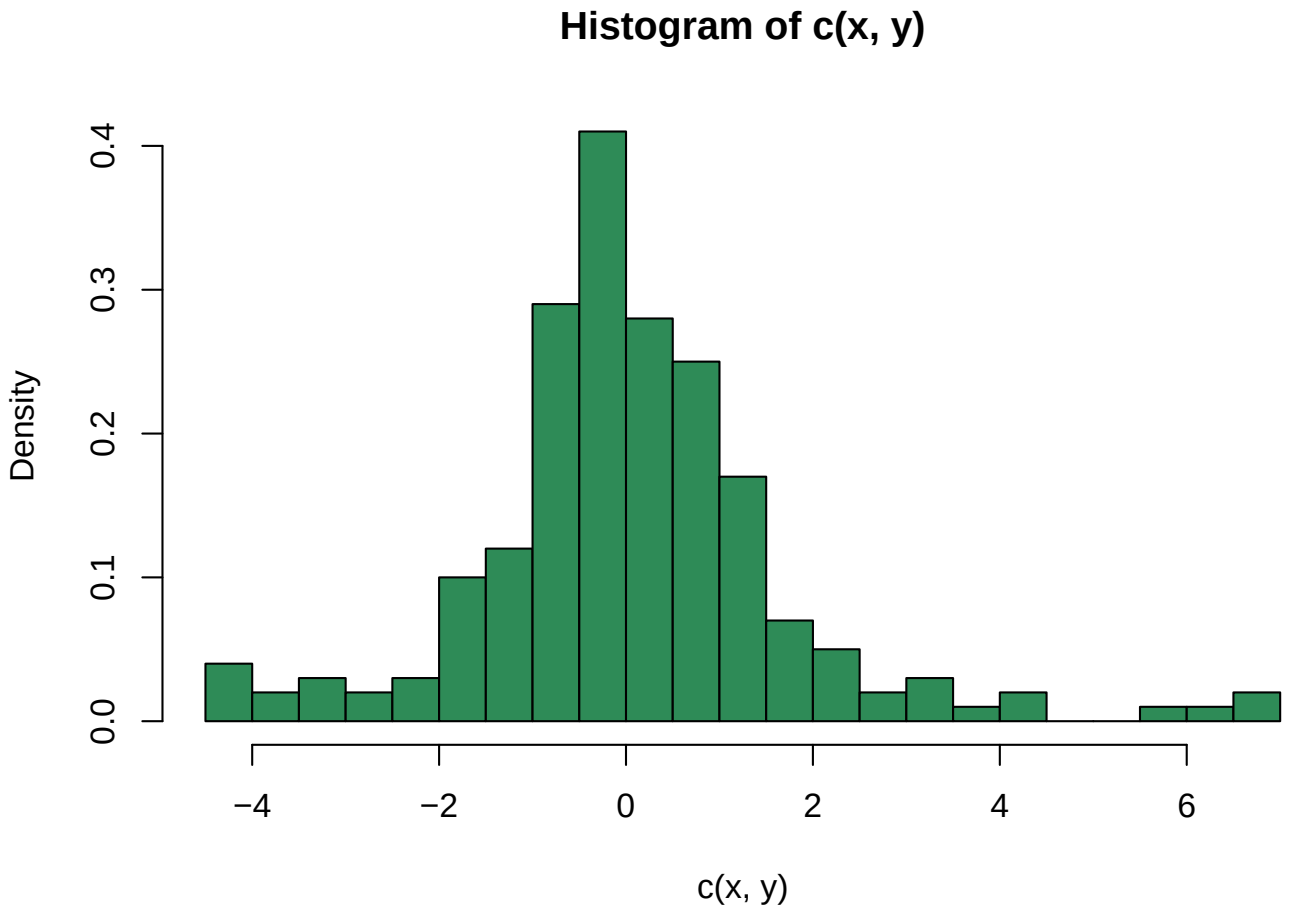
```
> hist(x,col="seagreen")
```



# Plotting histograms

**R Code: Plotting histograms**

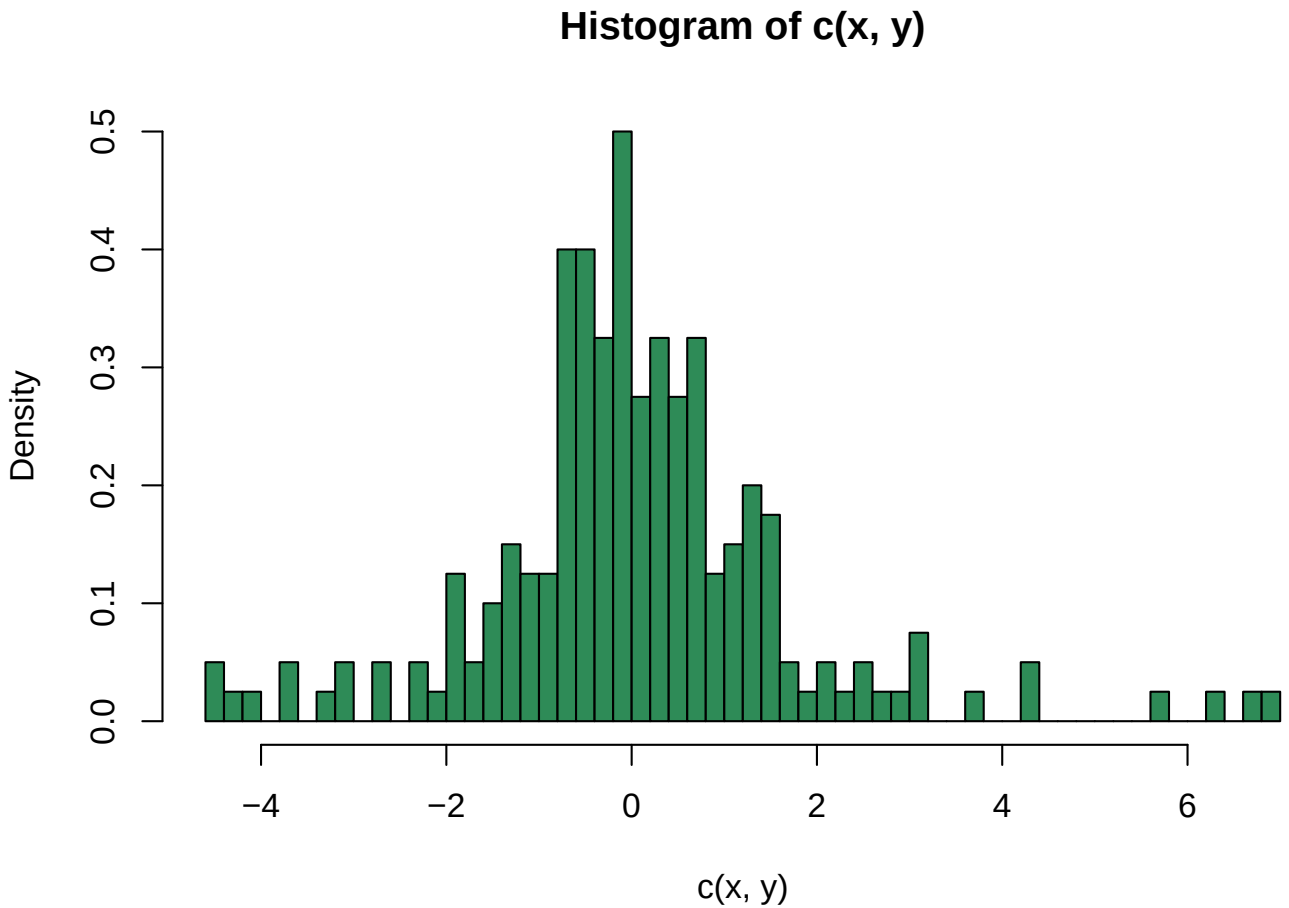
```
> hist(c(x,y),prob=T,breaks="FD",col="seagreen")
```



# 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

# 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 |
| `timeSeries` | rmetrics | default for Rmetrics packages |
| `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("MSFT.CSV", header = TRUE, sep = ",", as.is = TRUE)
> head(msft.df,2)

        Date  Open  High  Low Close   Volume Adj.Close
1 2009-09-21 25.11 25.37 25.1 25.30 28864500     25.30
2 2009-09-18 25.46 25.48 25.1 25.26 68016500     25.26

> 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)

2008-09-22 2008-09-23 2008-09-24 2008-09-25 2008-09-26 2008-09-29
     25.40      25.44      25.72      26.61      27.40      25.01
```

# Inspecting a zoo object

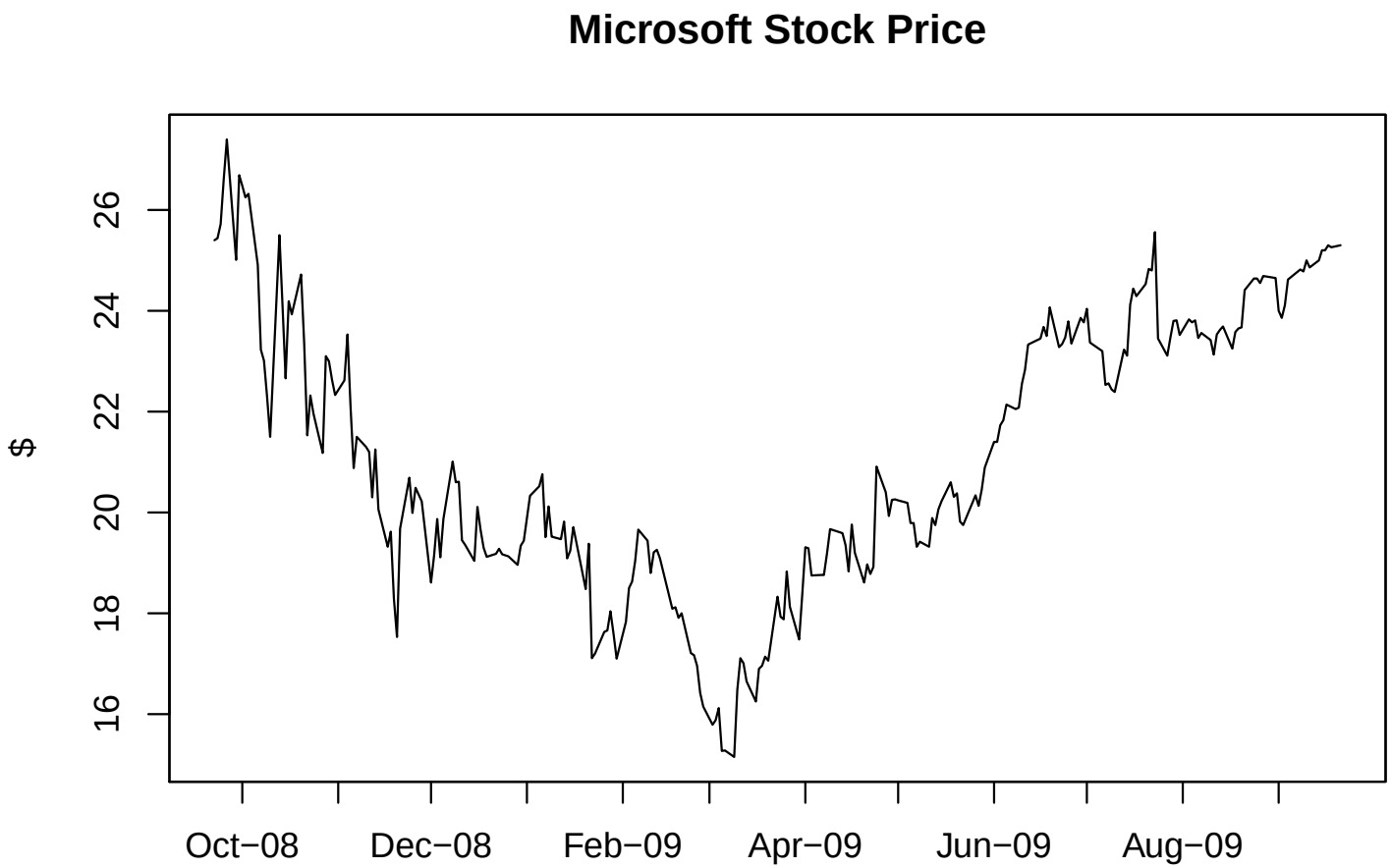
**R Code: Inspecting a zoo object**

```
> class(msft.z)
[1] "zoo"
> start(msft.z)
[1] "2008-09-22"
> end(msft.z)
[1] "2009-09-21"
> 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**

```
> ticks <- as.Date(unique(as.yearmon(index(msft.z))))
> plot(msft.z, xaxt='n',xlab="",ylab="$",main="Microsoft Stock Price")
> axis(side=1, at=ticks, lab=format(ticks,"%b-%y"))
```



# The read.zoo function

**R Code: reading a file into a zoo object**

```
> args(read.zoo)

function (file, format = "", tz = "", FUN = NULL, regular = FALSE,
    index.column = 1, drop = TRUE, FUN2 = NULL, split = NULL,
    aggregate = FALSE, ..., text)
NULL

> soft <- read.zoo(file="MSFT.CSV",header=TRUE,sep=",")
> head(soft,2)

            Open  High   Low Close    Volume Adj.Close
2008-09-22 26.22 26.32 25.32 25.40 105207700     24.76
2008-09-23 25.66 26.17 25.34 25.44  92181300     24.80

> class(soft)

[1] "zoo"

> class(coredata(soft))

[1] "matrix"

> class(index(soft))

[1] "Date"
```

# 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:stats"      "package:graphics"
 [7] "package:grDevices" "package:datasets"   "package:utils"
[10] "Autoloads"          "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
```

COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

http://depts.washington.edu/compfin