COMPUTATIONAL FINANCE & RISK MANAGEMENT
UNIVERSITY of WASHINGTON
Department of Applied Mathematics

A Brief History of R
R Setup and Environment

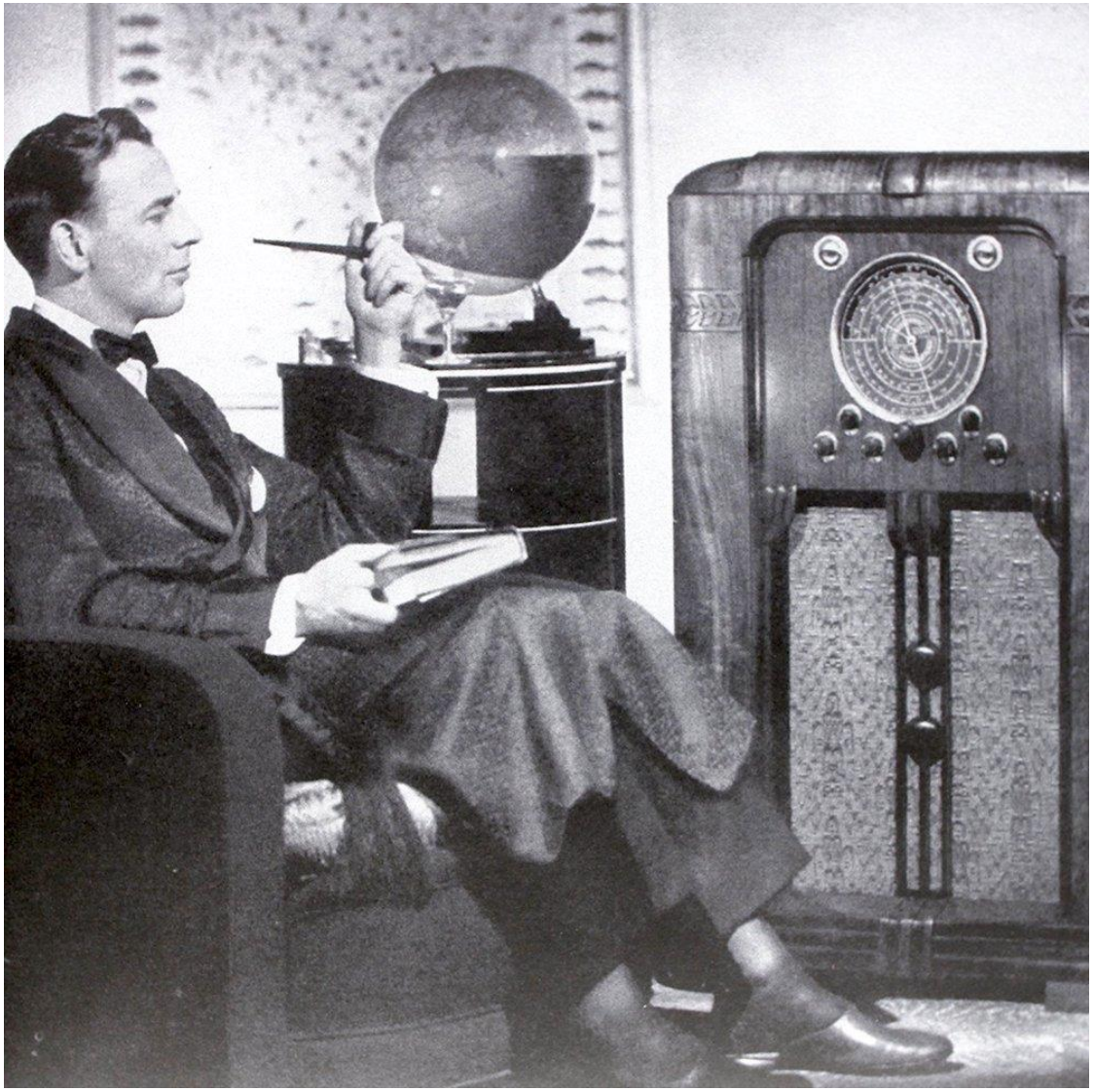
CFRM 425 (002)

R Programming for Quantitative Finance

References/Reading

- Reading: Jeet & Vats, Ch 1
- Additional Reference: Guy Yollin lecture notes, CFRM short course on R, 2014

R Overview and History



The R programming language

- 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 in the 1990s
- R is formally known as The R Project for Statistical Computing

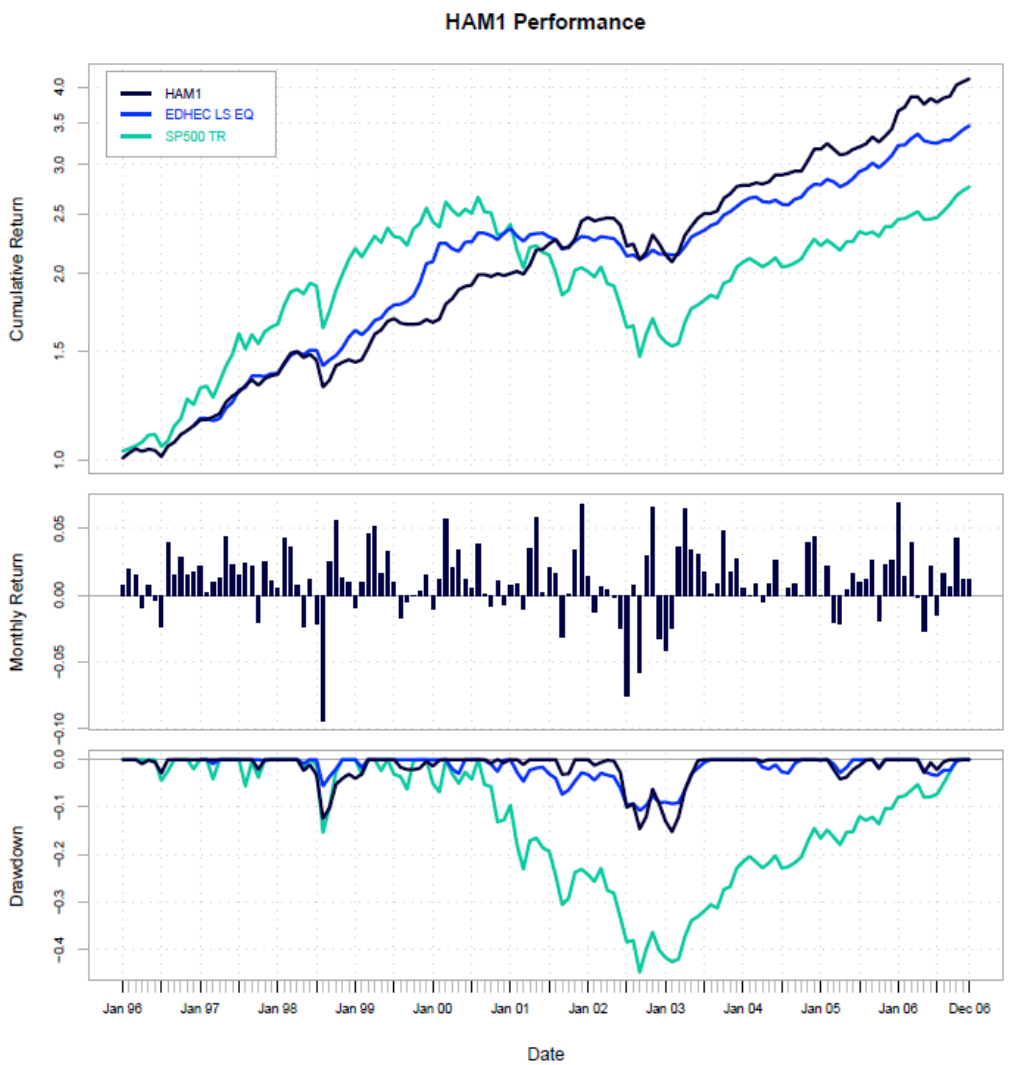
www.r-project.org

_____

Guy Yollin notes

Strengths of the R programming language

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



Plot from the PerformanceAnalytics package

Guy Yollin notes

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

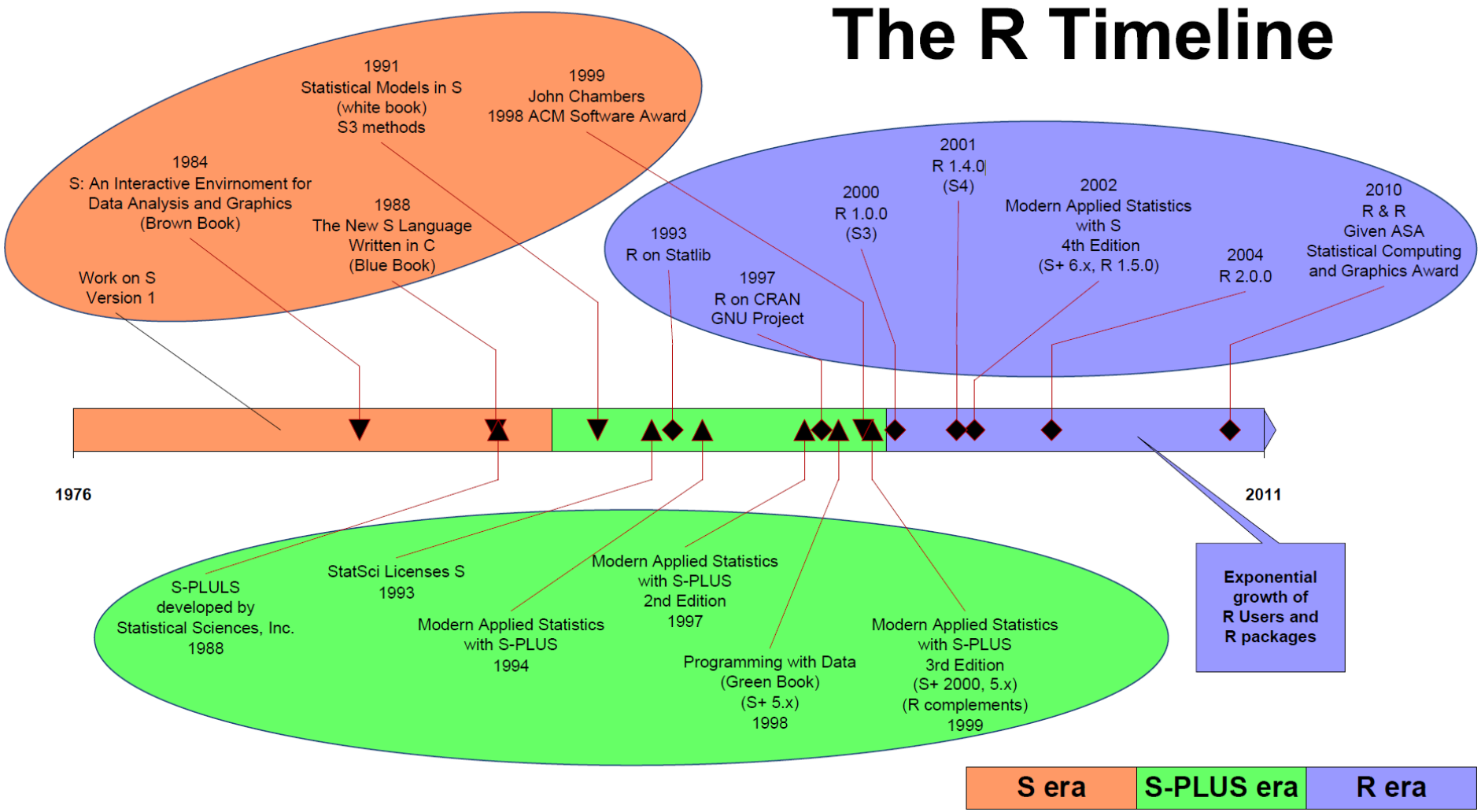
---

† Founded by UW Professor Doug Martin, CFRM Program Founder and former Director

Guy Yollin notes

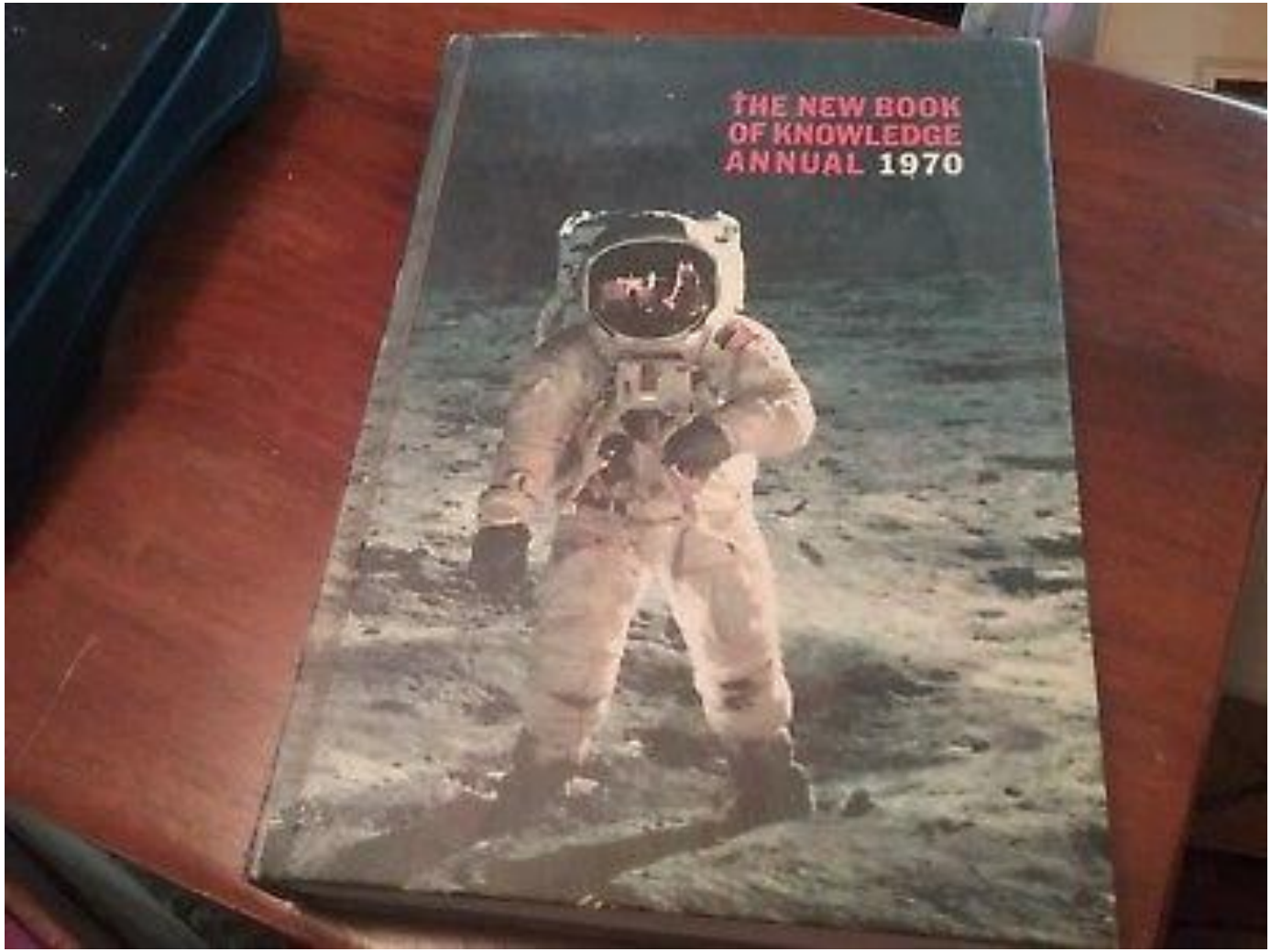
Revolution Analytics

- Founded in 2007
- Venture funded
- Former S-PLUS members
- Company focus
  - "Red Hat for R"
  - Higher performance version of CRAN R, by linking with the MKL libraries
  - Commercial package – RevoScaleR
    - Competitor of SAS
    - Handled large data sets without loading entire set into memory first
    - Core library written in C++
    - High-performance machine learning algorithms
      - General linear models (particularly logistic regression and multiple regression)
      - Tree-based models: random trees, random forest, boosted trees
      - Naïve Bayes
      - Web-based data visualization
      - Others
  - Consulting and education services for clients using Base R or Revo R
  - Wrote and maintained several well-known open R packages
- Acquired by Microsoft, 2015



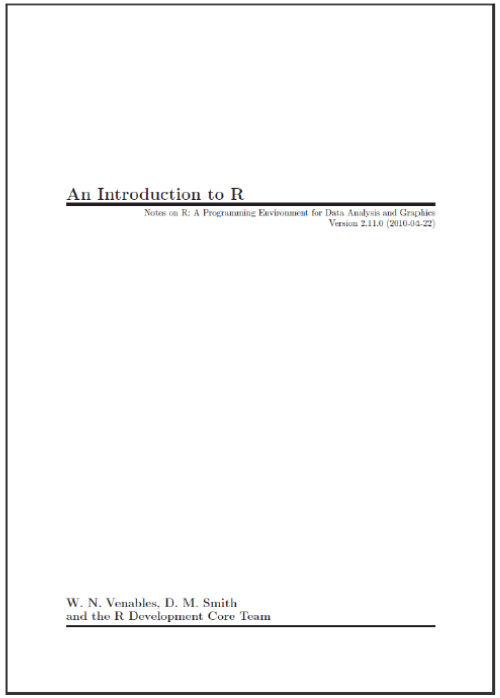
Guy Yollin notes

R Language References



Essential web resources

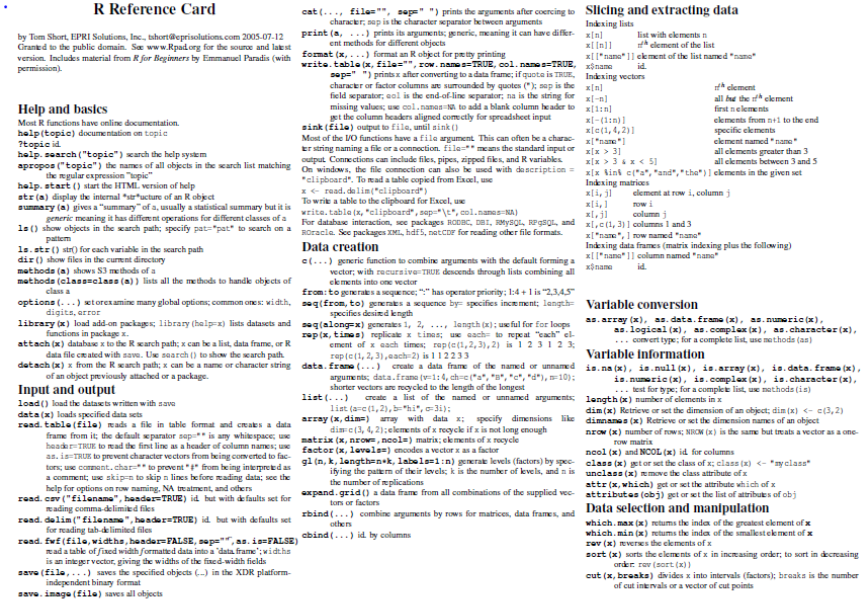
- An Introduction to R
  - W.N. Venables, D.M. Smith
  - R Development Core Team
- R Reference Card 2.0
  - Baggott & Short



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

by Tom Short, EPRI Solutions, Inc., tshort@eprisolutions.com 2005-07-12
Granted to the public domain. See www.RpadGrg 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 fixed width formatted 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$name` id.
Indexing vectors
`x[n]` n^{th} element
`x[-n]` all *but* the nth element
`x[1:n]` first n elements
`x[-(1:n)]` elements from n+1 to the end
`x[c(1,4,2)]` specific elements
`x["name"]` element named "name"
`x[x > 3]` all elements greater than 3
`x[x > 3 & x < 5]` all elements between 3 and 5
`x[x %in% c("a","and","the")]` elements in the given set
Indexing matrices
`x[i,j]` element at row i, column j
`x[i,]` row i
`x[,j]` column j
`x[,c(1,3)]` columns 1 and 3
`x["name",]` row named "name"
Indexing data frames (matrix indexing plus the following)
`x[["name"]]` column named "name"
`x$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 from the R homepage or a CRAN mirror

Guy Yollin notes

Experience with other statistical computing languages

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

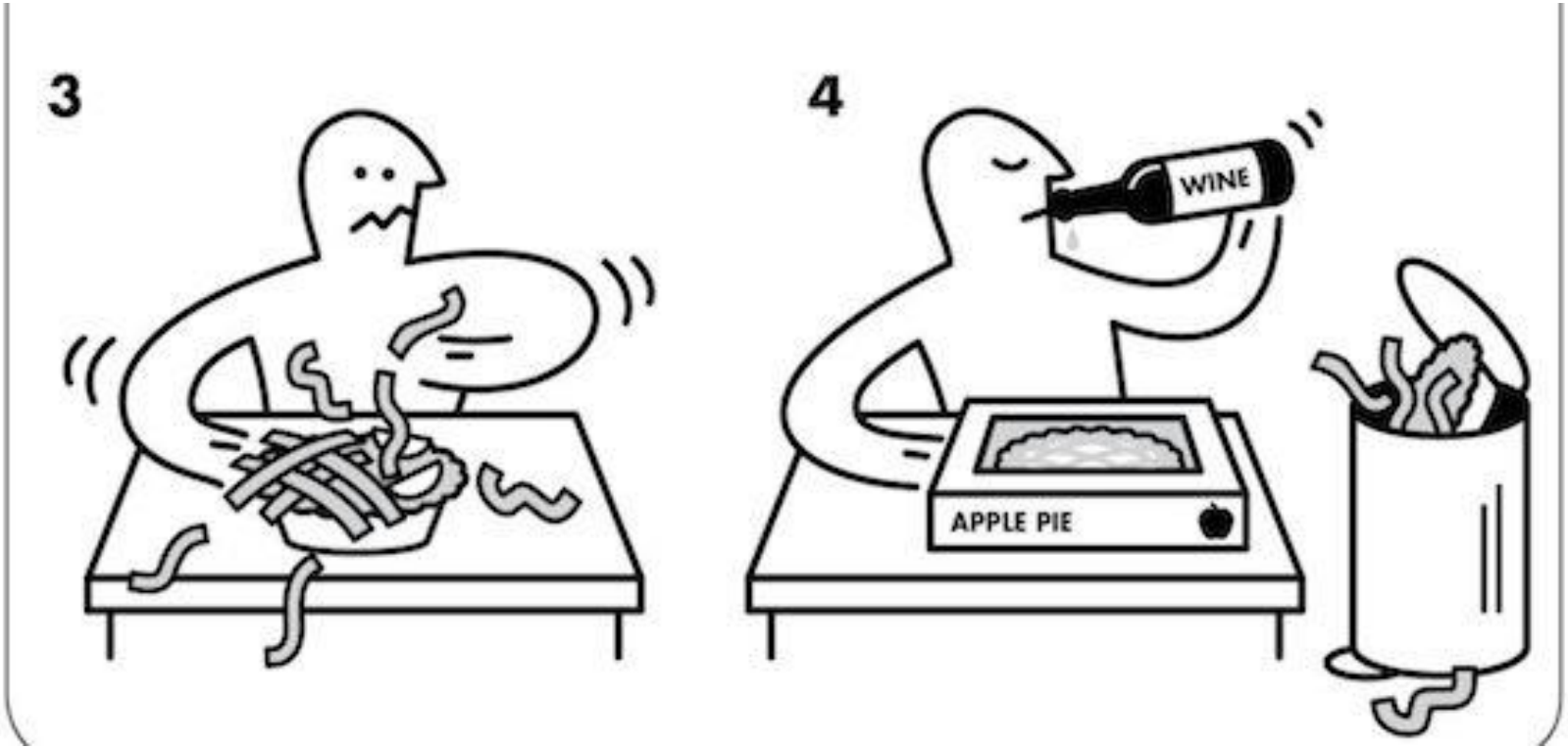
  http://www.math.umaine.edu/~hiebeler/comp/matlabR.pdf

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

  http://r4stats.com

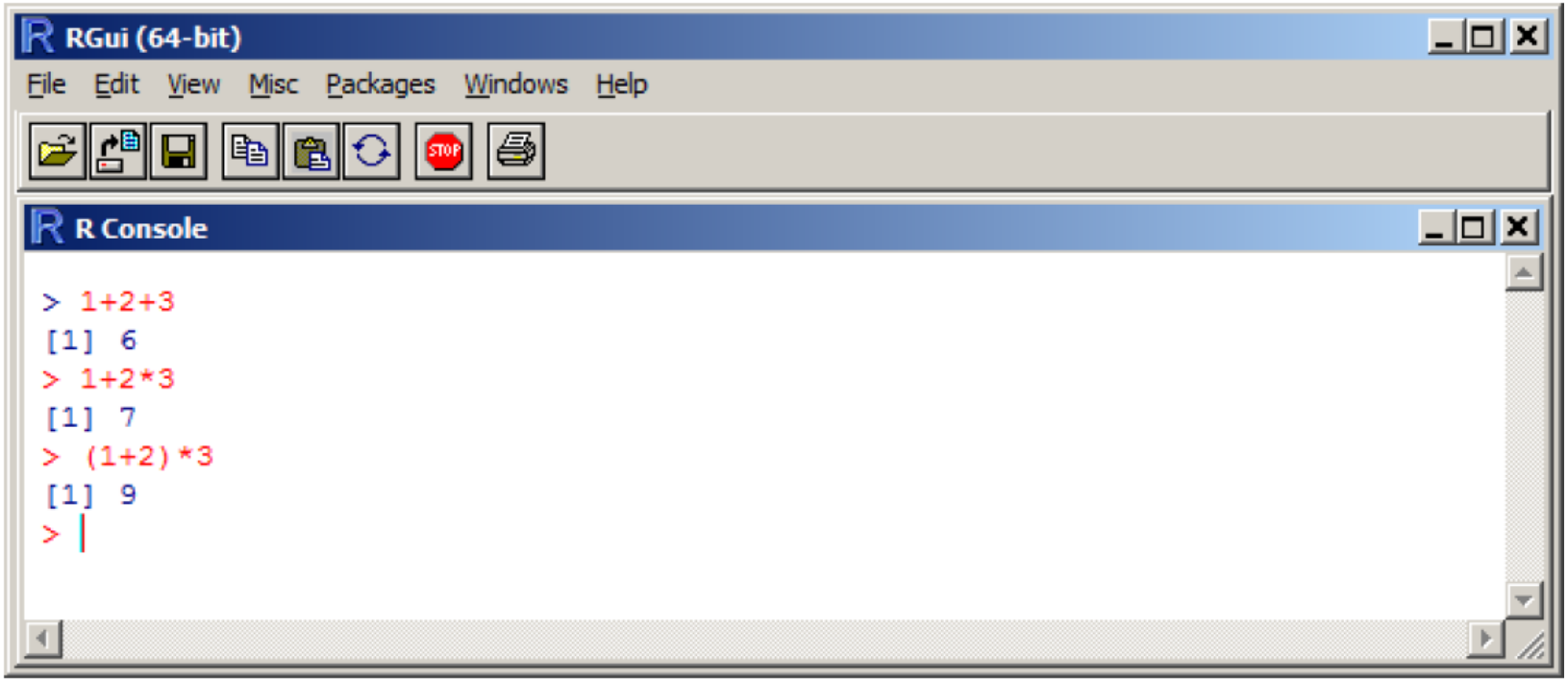
Guy Yollin notes

A Short R Tutorial



Interacting with R

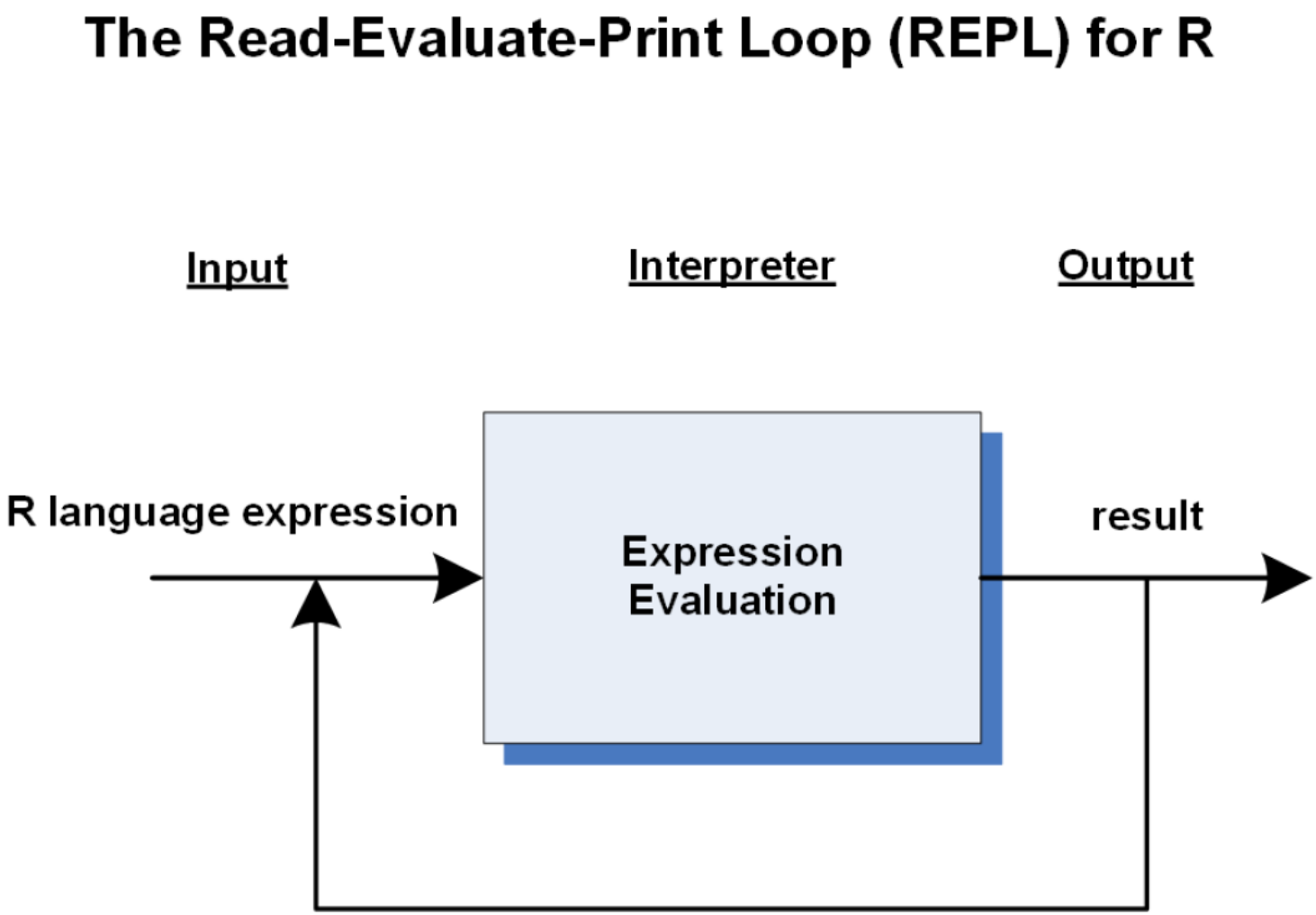
- R is an interpreted language (not compiled)
- An R interpreter must be running in order to evaluate R commands or execute R scripts
  - RGui which includes an R Console window (below)
  - RStudio which includes an R Console window (*preferred*)



Guy Yollin notes

R expression evaluation (REPL)

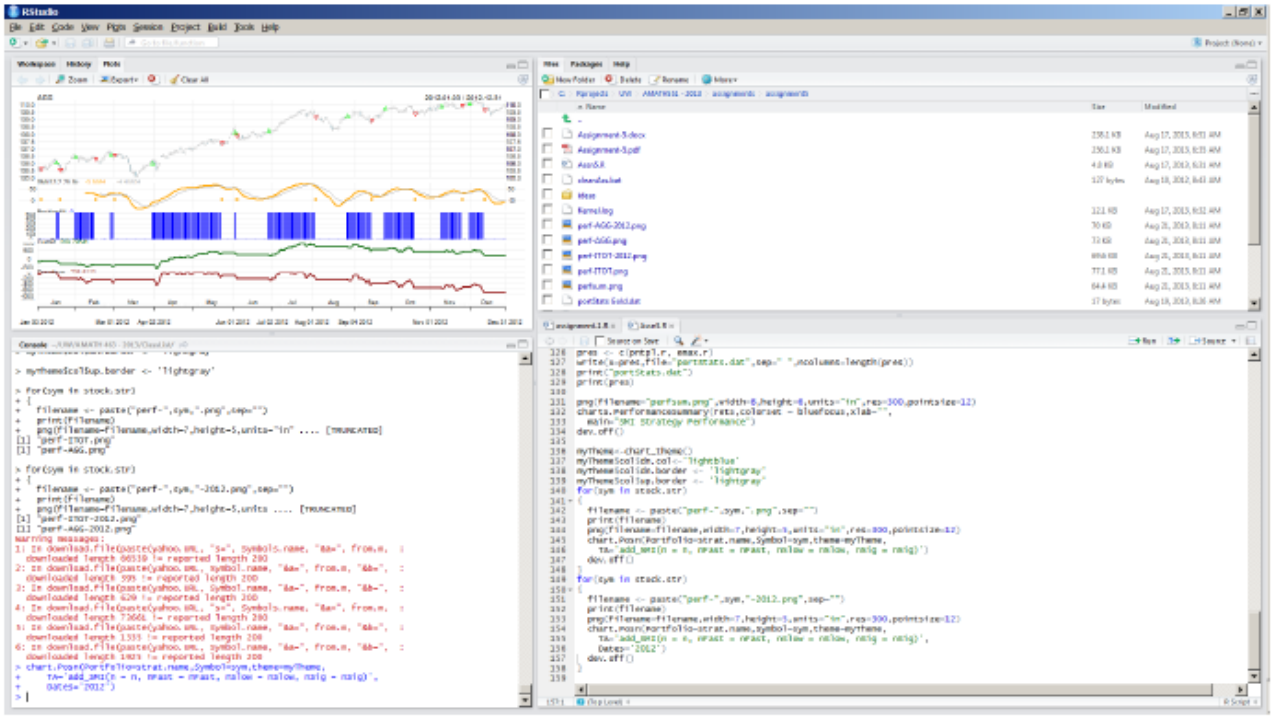
- R expressions are often processed via R's Read-eval-print loop*:



*http://en.wikipedia.org/wiki/Read-eval-print_loop

Interacting with RStudio

- The RStudio is an Integrated Development Environment (IDE) for R:
  - Embedded R Console: RStudio runs an R interpreter automatically
  - Program editor for R
  - Plot window
  - File browser
  - Integrated version control
  - R debugger



Guy Yollin notes

RStudio includes an embedded R Console

Calling R Functions

- R makes extensive use of
  - built-in functions
  - user-defined functions (UDF)
- Functions can be defined to take zero or more arguments
- Functions typically return a value, but this is not required (aka a *void* function, in C++/Java parlance)
- Functions are called by name with any arguments enclosed in parentheses even if the function has no arguments the parentheses are required

```
y <- 5
y
```

```
## [1] 5
```

```
assign("e",2.7183)
e
```

```
## [1] 2.7183
```

```
s = sqrt(2)
s
```

```
## [1] 1.4142136
```

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

```
## [1] -1.0067110533 -0.0020828847
```

```
s*e+y
```

```
## [1] 8.8442567
```

Guy Yollin notes

Classes and Objects (Object Oriented Programming)

- Everything in R is an Object
- All R objects have a class
- The class of an object determines what it can do and what you can do with it
- Use function `class(.)` to display an object's class
- There are many R classes; basic classes are:
  - `numeric`
  - `character`
  - `data.frame`
  - `matrix`
- Two other useful functions:
  - `names(.)`
  - `str(.)`

```
m

##              [,1]         [,2]        [,3]
## [1,]  0.374352397  0.586864810 -0.73778598
## [2,] -0.071532765 -0.262264339 -0.19904931
## [3,]  0.790144078  0.012603635  1.96472235

class(m)

## [1] "matrix"

tab

##      store sales
## 1 downtown    32
## 2 eastside    17
## 3  airport    24

class(tab)

## [1] "data.frame"
```

Guy Yollin notes

Vectors

- R is a vector/matrix programming language (also known as an array programming language)
- vectors can easily be created with `c(.)`, the ***c**ombine* function
- most places where single value can be supplied, a vector can be supplied and R will perform a vectorized operation

```
my.vector <- c(2, 4, 3, 7, 10)
my.vector

## [1]  2  4  3  7 10

my.vector^2

## [1]   4  16   9  49 100

sqrt(my.vector)

## [1] 1.4142136 2.0000000 1.7320508 2.6457513 3.1622777
```

Guy Yollin notes

Creating vectors with the c(.) function

```
constants <- c(3.1416,2.7183,1.4142,1.6180)
constants

## [1] 3.1416 2.7183 1.4142 1.6180

my.labels <- c("pi","euler","sqrt2","golden")
my.labels

## [1] "pi"     "euler"  "sqrt2"  "golden"

names(constants) <- my.labels
constants

##     pi  euler  sqrt2 golden
## 3.1416 2.7183 1.4142 1.6180
```

- The **names(.)** function assigns character names identifying each element of a vector
- The [1] in the above output is labeling the first element of the vector
- The **c(.)** function can be used to create character vectors, numeric vectors, as well as other types of vectors

Guy Yollin notes

Indexing vectors

- Vector 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 condition

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

```
##     pi  sqrt2 golden
## 3.1416 1.4142 1.6180
```

```r
constants[c(-1,-2)]
```

```
##  sqrt2 golden
## 1.4142 1.6180
```

```r
constants[c("pi","golden")]
```

```
##     pi golden
## 3.1416 1.6180
```

```r
constants > 2
```

```
##     pi  euler  sqrt2 golden
##   TRUE   TRUE  FALSE  FALSE
```

```r
constants[constants > 2]
```

```
##     pi  euler
## 3.1416 2.7183
```

Guy Yollin notes

Creating integer sequences with the a:b operator

- The sequence operator will generate a vector of integers between a and b, using the colon (:) operator
- Sequences of this type are particularly useful for indexing vectors, matrices, dataframes etc

```
1:5

## [1] 1 2 3 4 5

-(1:4)

## [1] -1 -2 -3 -4

letters[1:15]

##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"

letters[16:26]

##  [1] "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"

letters[-(1:15)]

##  [1] "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

Guy Yollin notes

Comparing vector and non-vector computing

```
# vectorized operation
# taking the log of each element in a vector
x <- c(97.87,96.18,95,86.39,88.18,90.8,86.06,82.27,83.32,85.3,83.25,82.13,78.54)
log(x)

##  [1] 4.5836401 4.5662214 4.5538769 4.4588719 4.4793802 4.5086593 4.4550447
##  [8] 4.4100065 4.4226886 4.4461745 4.4218481 4.4083034 4.3636080

# non-vectorized computation
# taking the log of each element in a vector
n <- length(x)
y <- rep(0,n)
for( i in 1:n )
  y[i] <- log(x[i])
y

##  [1] 4.5836401 4.5662214 4.5538769 4.4588719 4.4793802 4.5086593 4.4550447
##  [8] 4.4100065 4.4226886 4.4461745 4.4218481 4.4083034 4.3636080
```

Guy Yollin notes

Comparing vector and non-vector computing

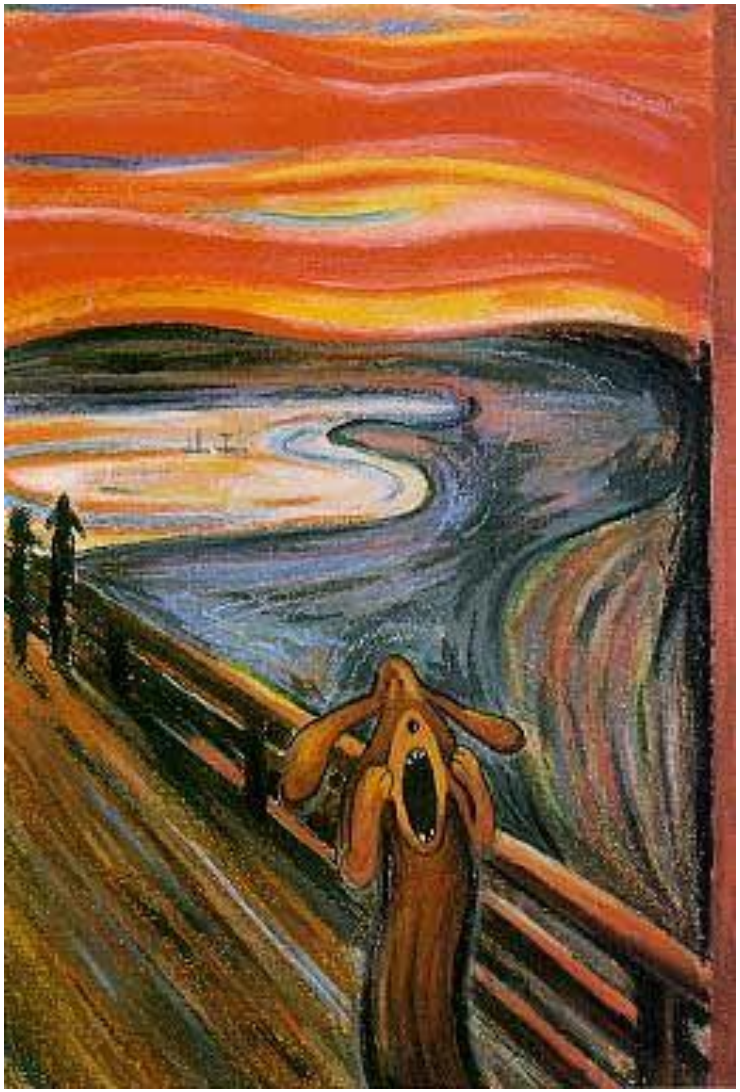
```
# vectorized operation
# taking the log of each element in a matrix
x <- matrix(c(2,9,4,7,5,3,6,1,8),nrow=3)
x^2

##      [,1] [,2] [,3]
## [1,]    4   49   36
## [2,]   81   25    1
## [3,]   16    9   64

# non-vectorized computation
# taking the log of each element in a matrix
y <- x
for( i in 1:nrow(x) )
  for( j in 1:ncol(x) )
    y[i,j] <- x[i,j]^2
y

##      [,1] [,2] [,3]
## [1,]    4   49   36
## [2,]   81   25    1
## [3,]   16    9   64
```

Guy Yollin notes

Comparing vector and non-vector computing

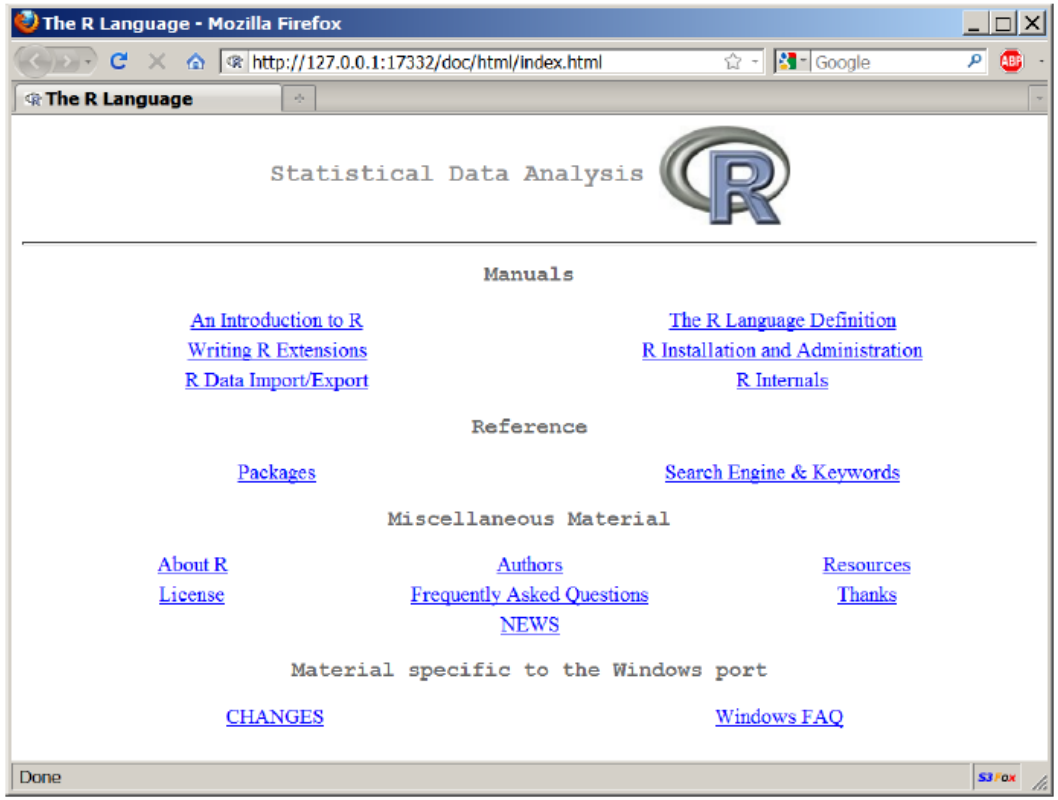
- *Always* prefer vectorized operations over loops in R
- Loops are a huge hit on performance in an interpreted language (vs a compiled language)
- Loops also usually make the code *less* maintainable and *more* error prone

The R Help System



The HTML Help System

- R has a comprehensive HTML help facility
  - Run the **help.start** function
  - R GUI menu item




```
help.start()

## If nothing happens, you should open
## 'http://127.0.0.1:28913/doc/html/index.html' yourself
```

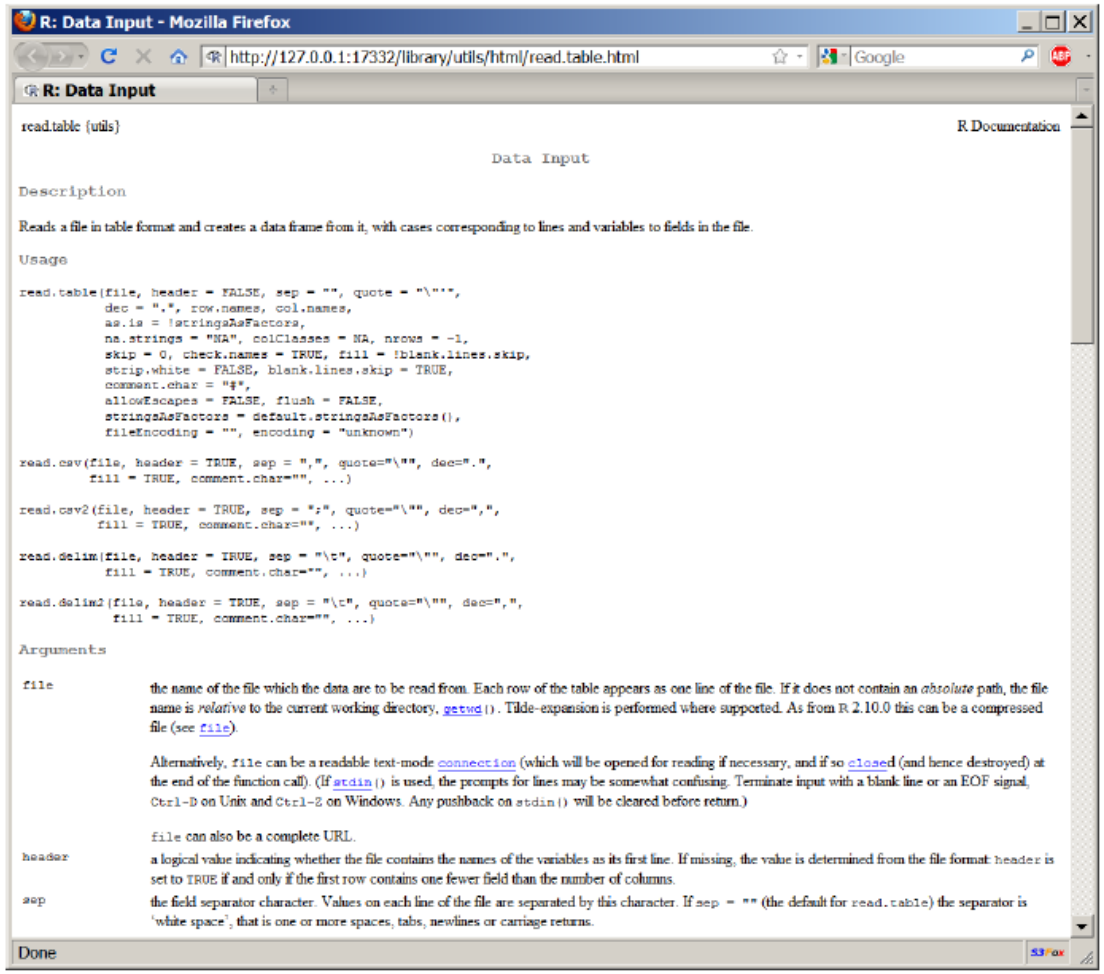
Guy Yollin notes

The `help` function

- Obtain help on a particular topic via the help function
  - `help(topic)`
  - `?topic`

- For example (these do the same thing):

```
help(read.table)
?read.table
```



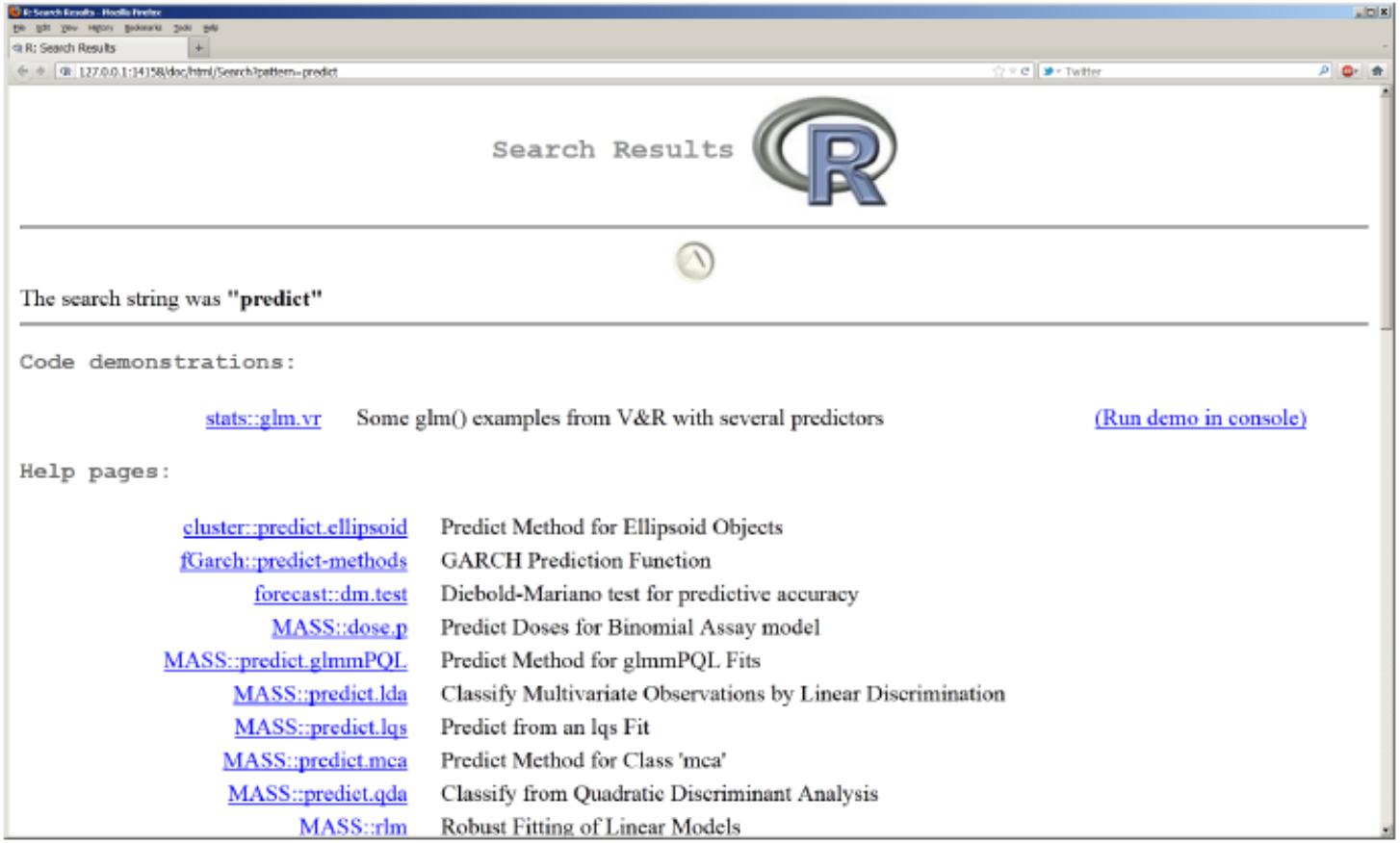
Guy Yollin notes

The `help.search` function

- Search help for a particular topic via the help.search function (eg, if you don't know the particular function or class name)
  - `help.search(topic)`
  - `??topic`

- For example (these do the same thing):

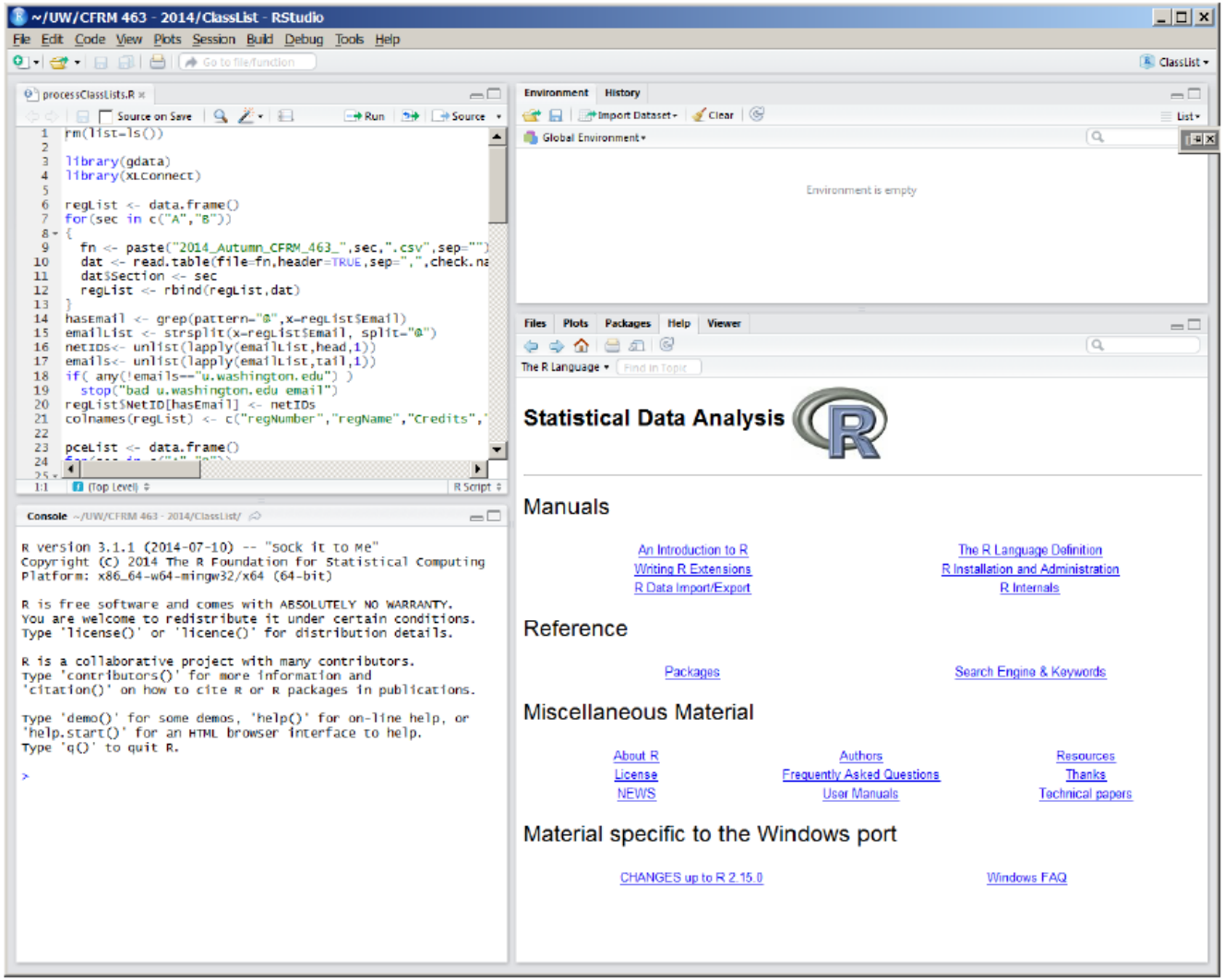
```
help(predict)
??predict
```



Guy Yollin notes

Help tab in RStudio (Recommended)

- RStudio incorporates a dedicated help tab which facilitates accessing the R HTML help system
- This is your instructor's 1st choice



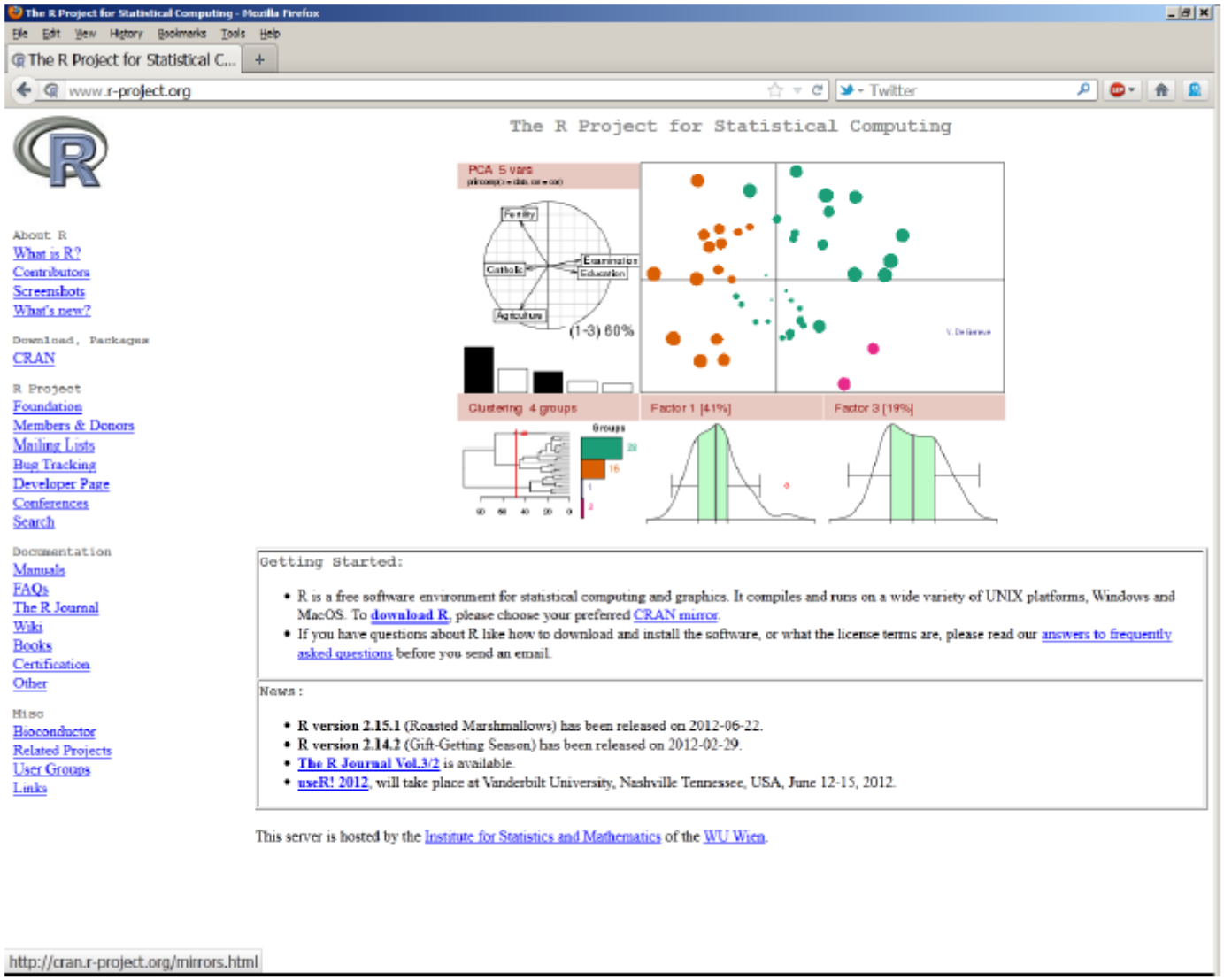
_____

Guy Yollin notes

Web Resources for R

R Homepage

- http://www.r-project.org
  - List of CRAN mirror sites – Download R
  - Manuals
  - FAQs
  - Site seach
  - Mailing lists
  - Links



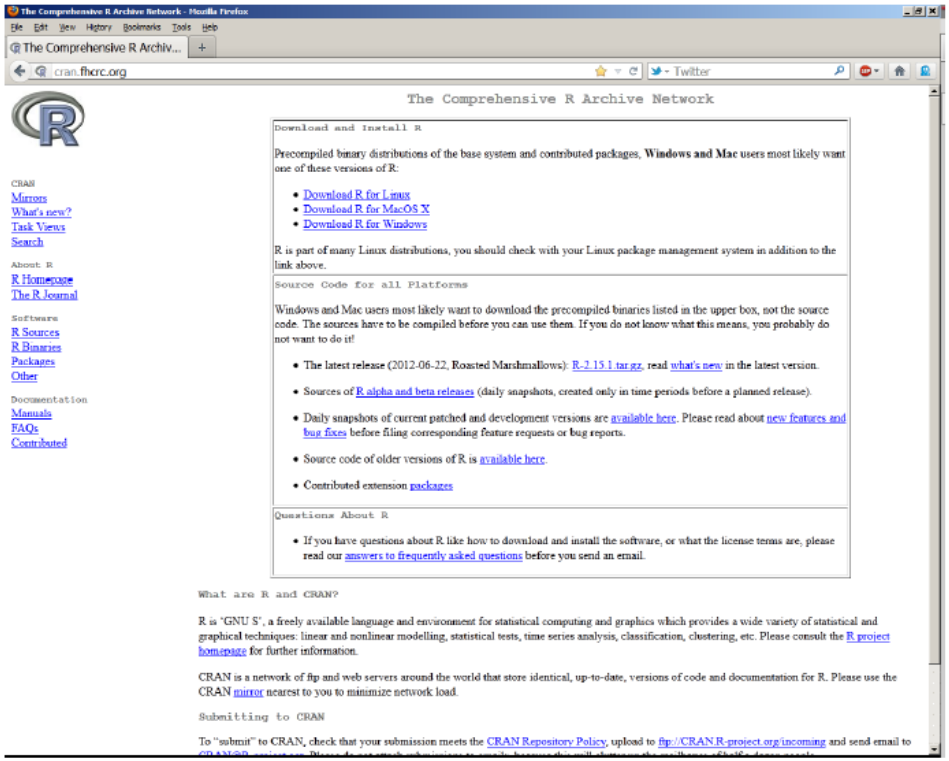
_____

Guy Yollin notes

CRAN - Comprehensive R Archive Network

- CRAN Mirrors
  - About 45 countries
  - About 100 sites worldwide
  - About 15 sites in US
- R Binaries (Base R installation)
- R Packages
  - 5800+ packages
- Use your closest CRAN mirror site
- Closest to UW Seattle is

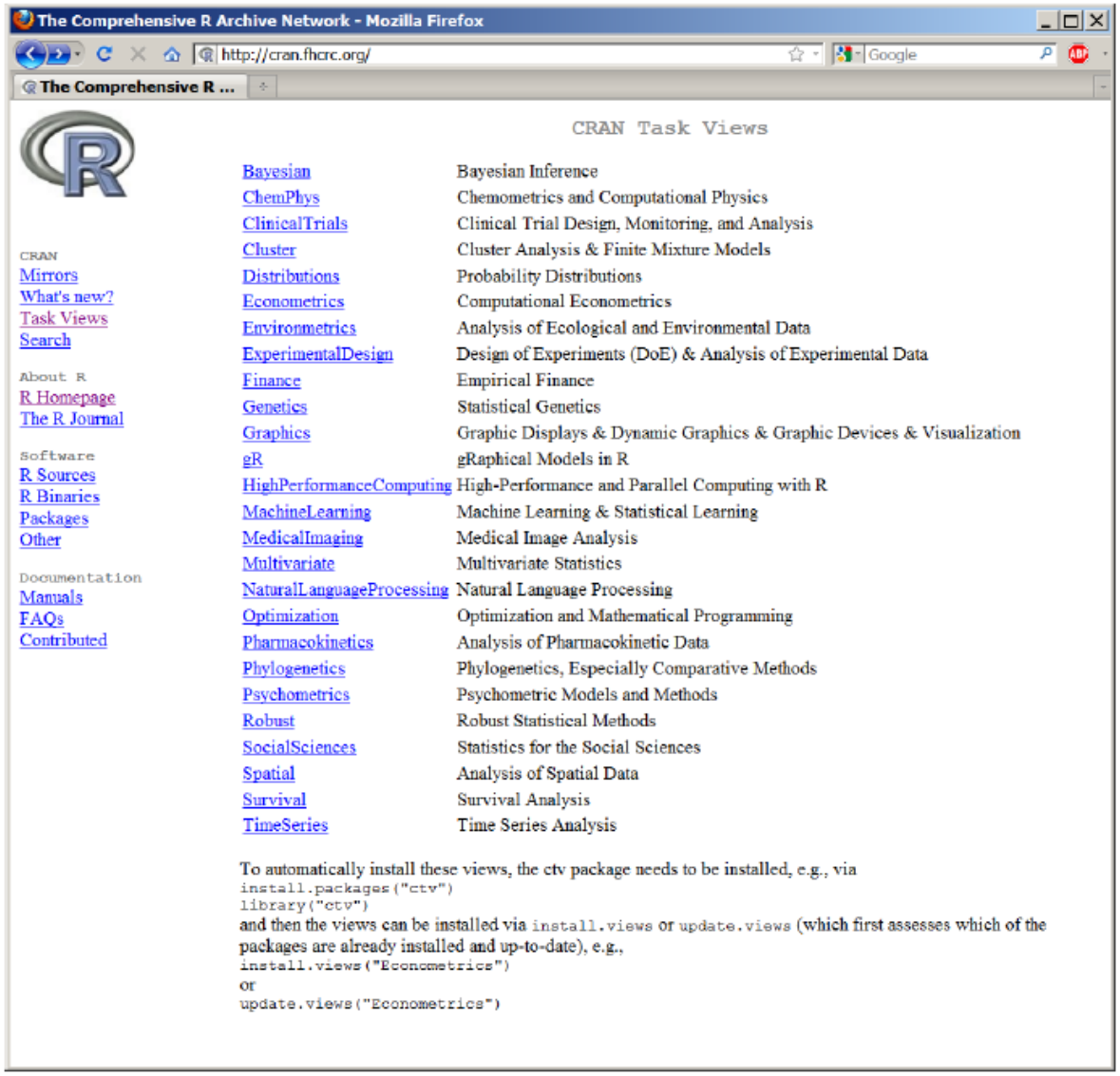
  http://cran.fhcrc.org



Guy Yollin notes

CRAN Task Views

- Organizes the 5800+ R packages by application
  - Finance
  - Time Series
  - Econometrics
  - Optimization
  - Machine Learning

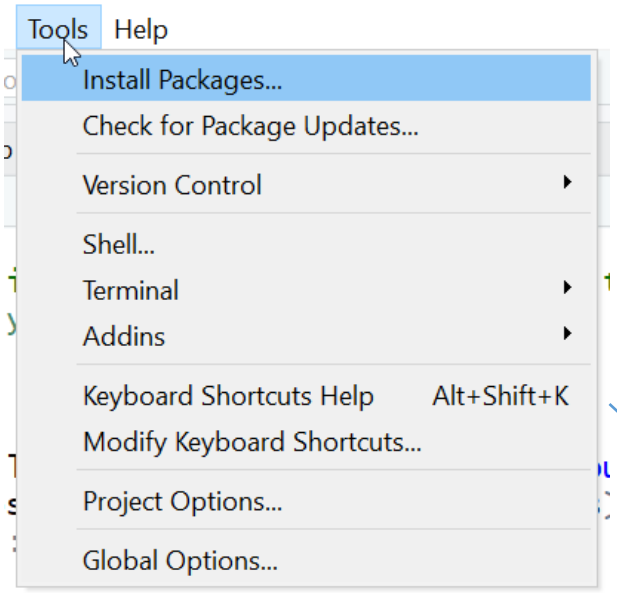


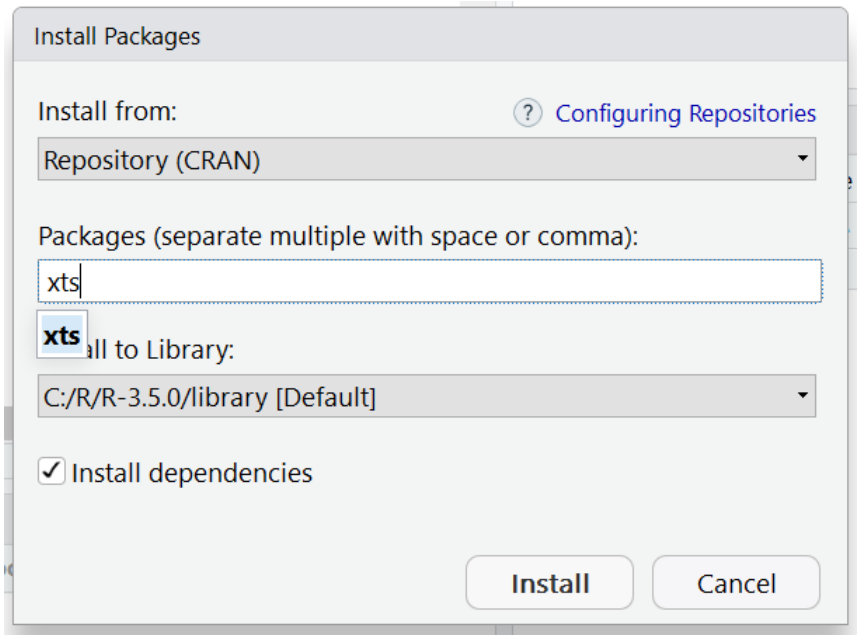
_____

Guy Yollin notes

Loading R Packages with RStudio

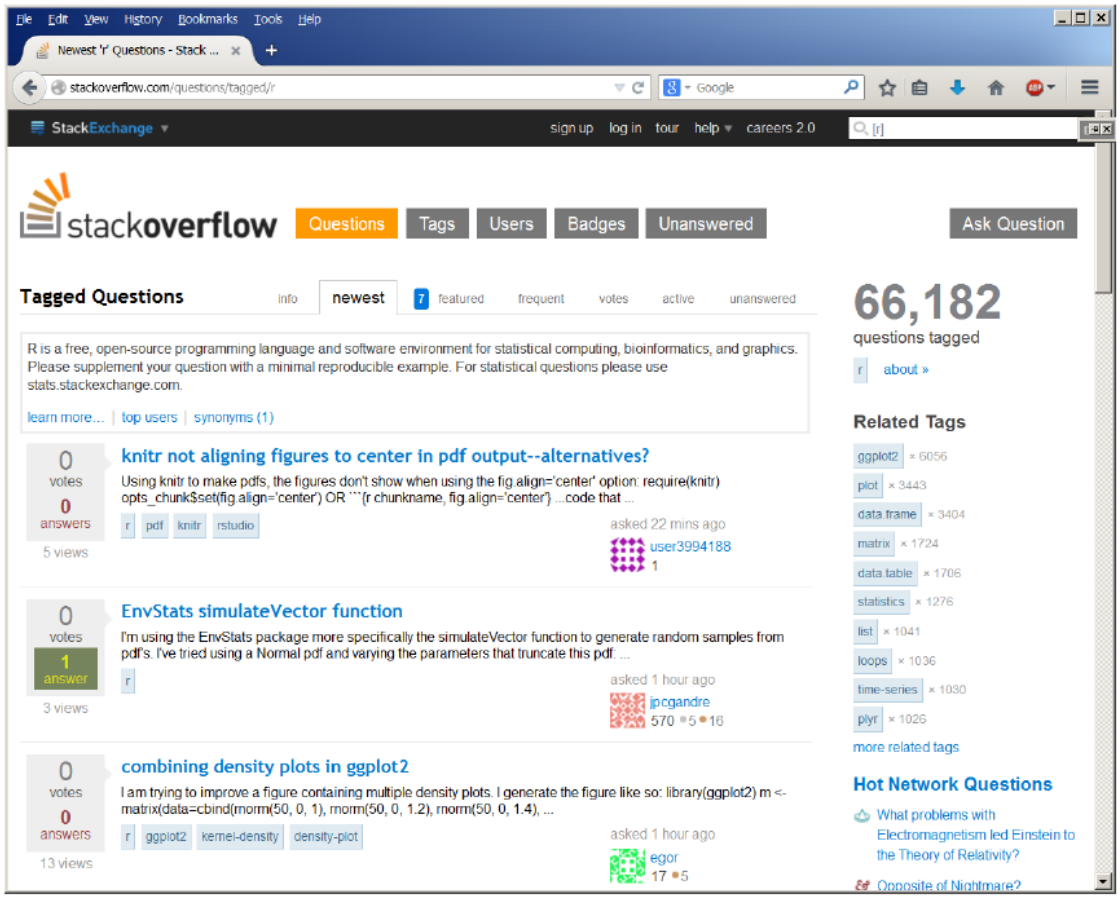
- The easiest and most straightforward method of loading R packages from CRAN
- Will be the latest versions available (by default)





Stackoverflow

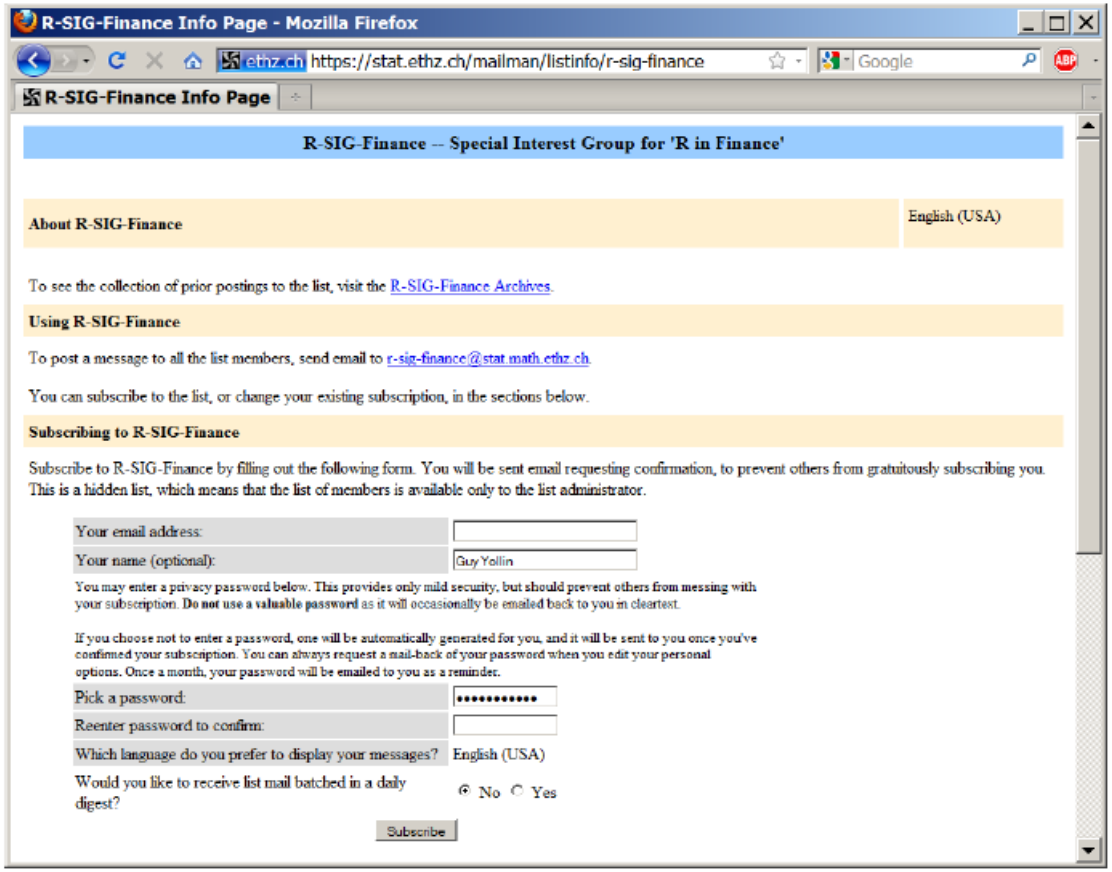
- Stackoverflow has become the primary resource for help with R



http://stackoverflow.com/

R-SIG-FINANCE

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

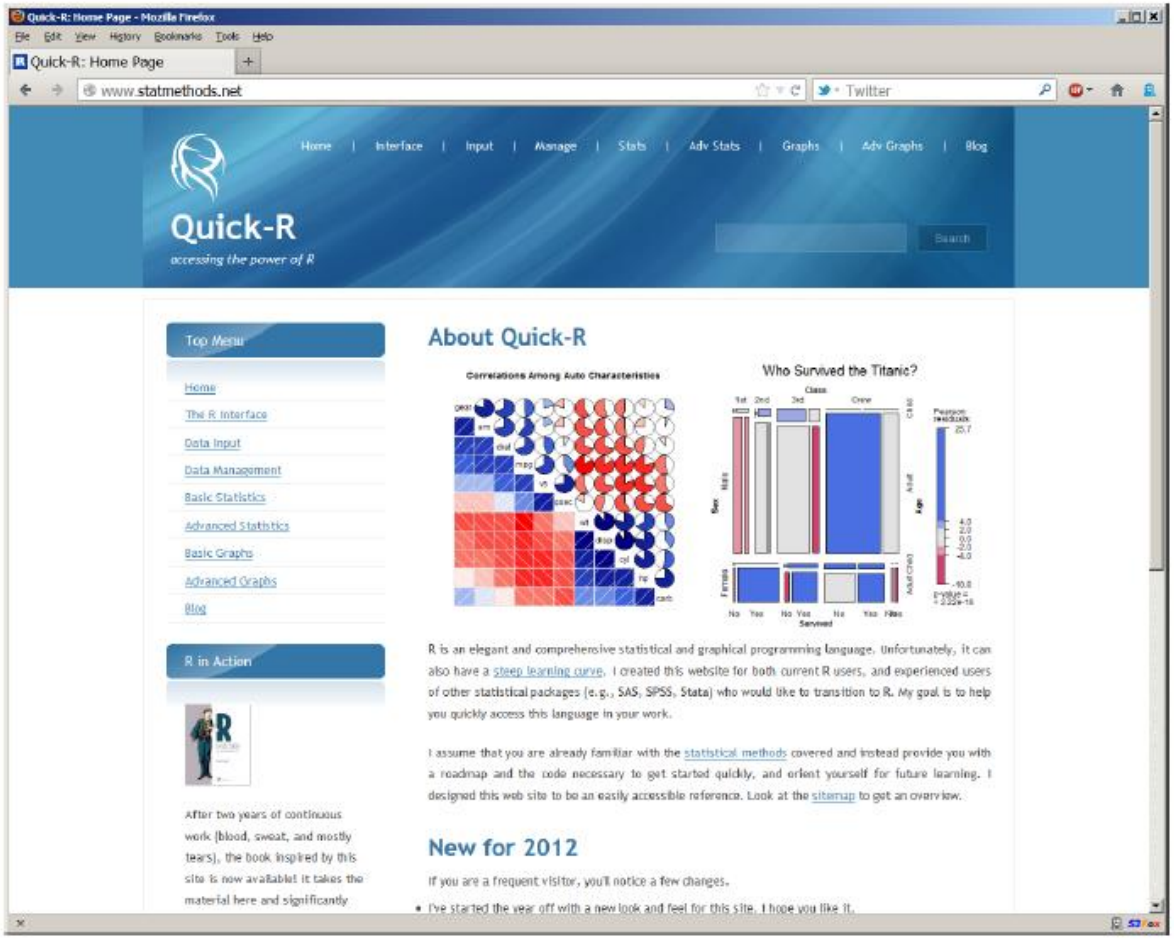


https://stat.ethz.ch/mailman/listinfo/r-sig-finance

Quick R

- http://www.statmethods.net

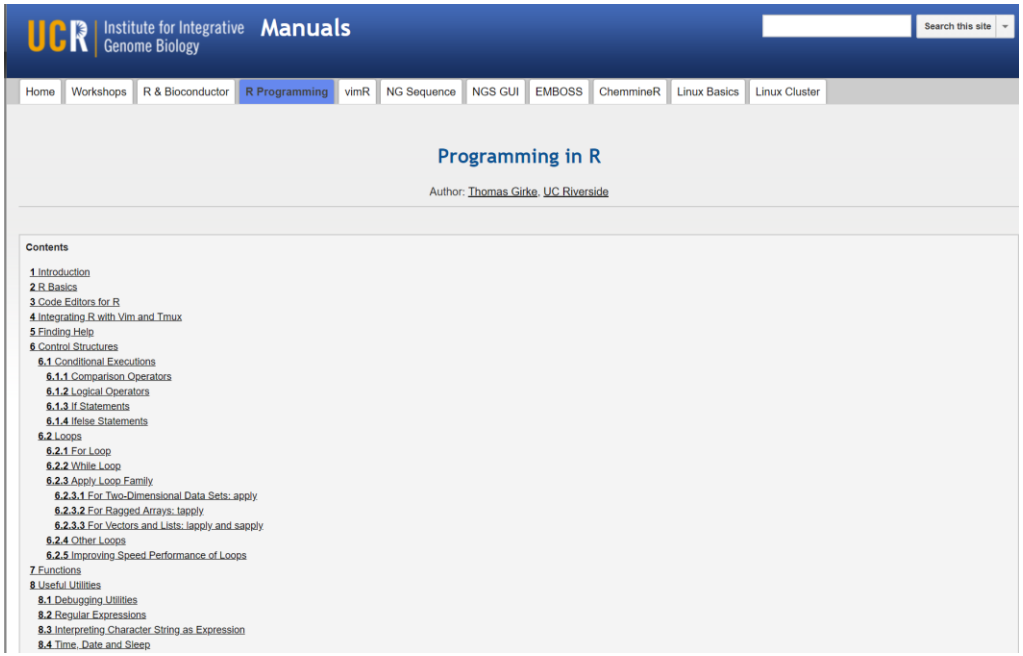
- Introductory R Lessons
  - R Interface
  - Data Input
  - Data Management
  - Basic Statistics
  - Advanced Statistics
  - Basic Graphs
  - Advanced Graphs



- Site maintained by Robert Kabacoff, author of R in Action

Online R programming manual from UC Riverside:

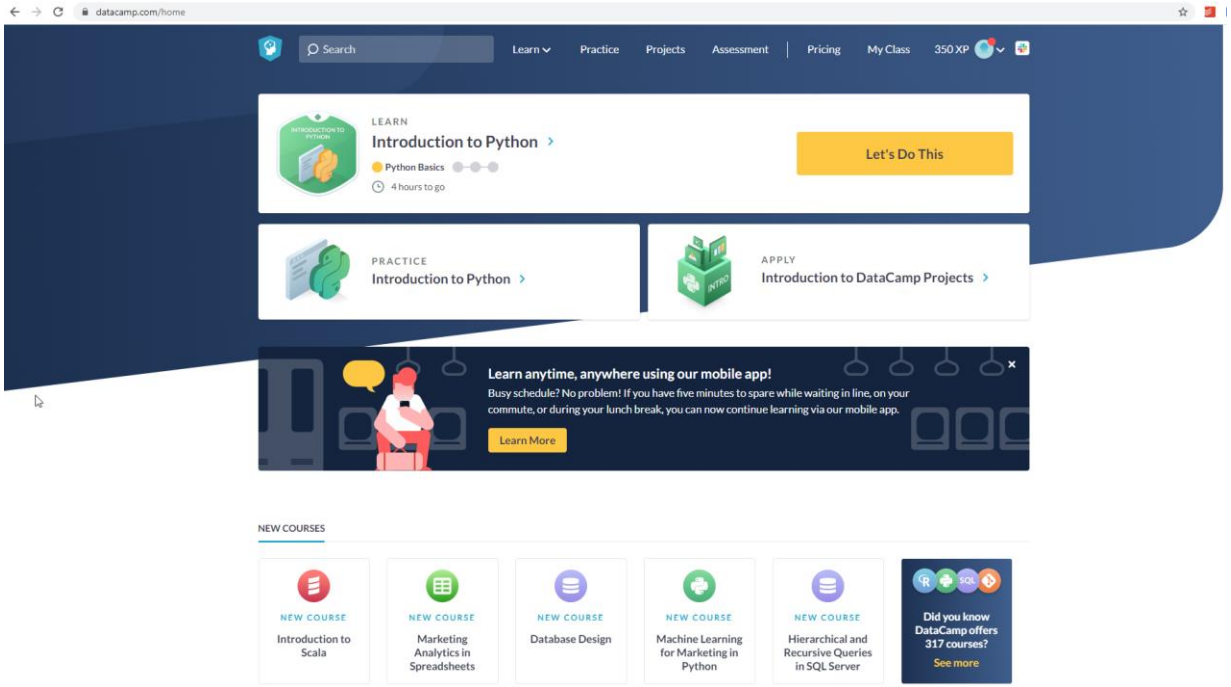
- R Basics
- Finding Help
- Code Editors for R
- Control Structures
- Functions
- Object Oriented Programming
- Building R Packages



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

DataCamp

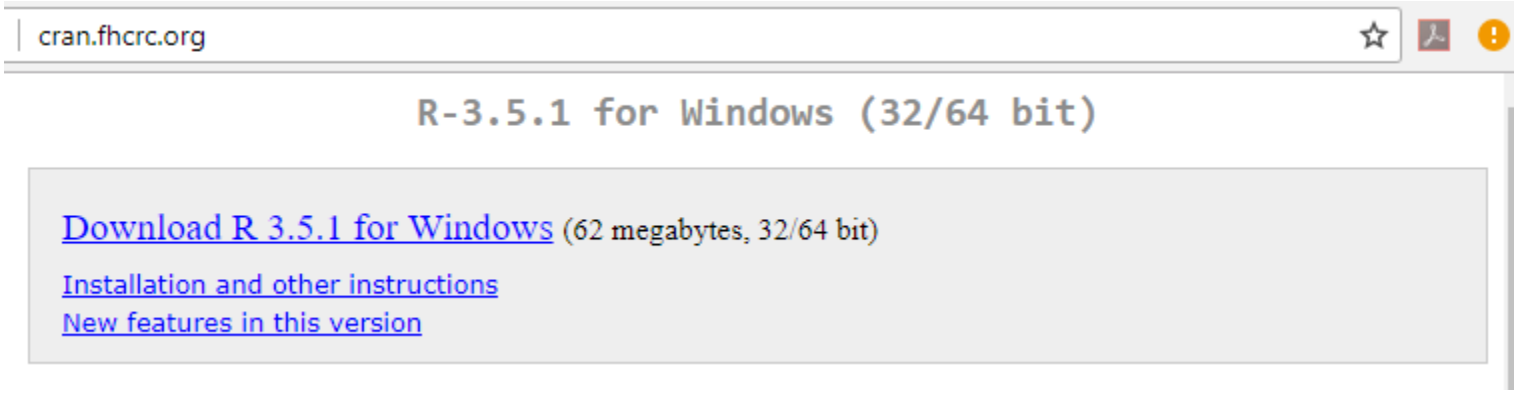
- You will receive a free six-month subscription with this course
- After receiving email confirmation, set up UN/PW
- Courses cover a wide variety of programming and database topics related to data science
  - R
  - Python
  - SQL
  - Spark/Hadoop/Scala
  - Domain applications
    - ➢ Finance
    - ➢ Machine Learning
    - ➢ Genetics
    - ➢ Others



Downloading and Installing R

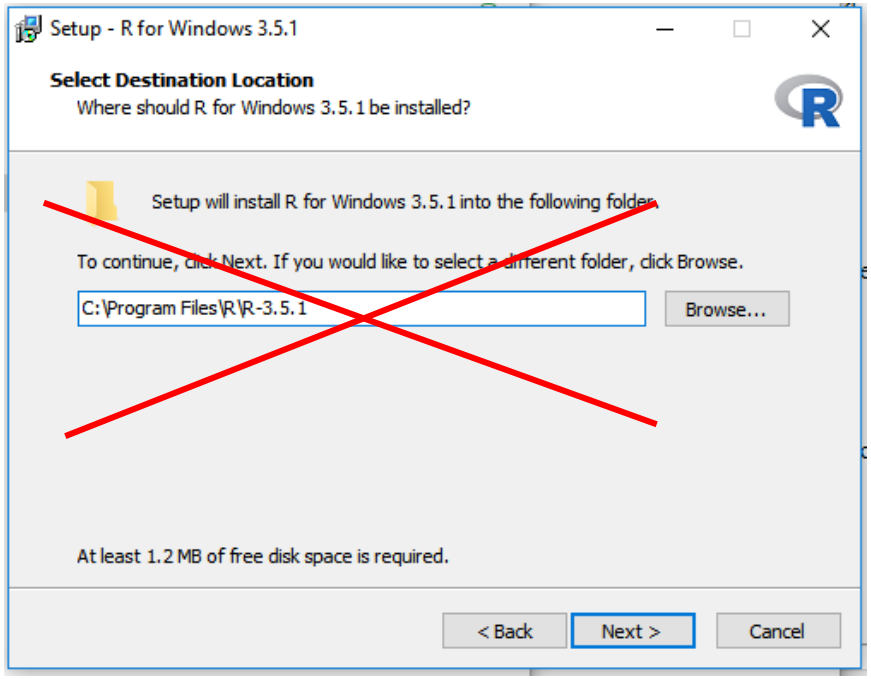
Downloading Base R from CRAN

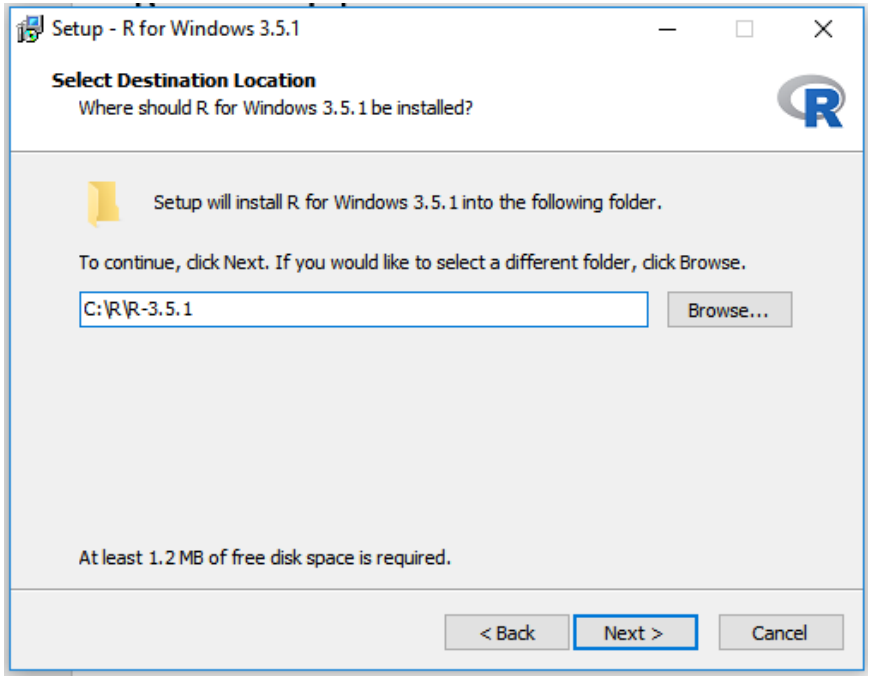
- CRAN:
  - Comprehensive R Archive Network
  - https://www.r-project.org/
  - Choose a mirror from https://cran.r-project.org/mirrors.html
  - Or, for closest mirror in Seattle area, use http://cran.fhcrc.org (not listed on CRAN site)
  - Latest version is R 3.**6**.1
  - Use 64-bit version



Installing Base R from CRAN

- On Windows, your instructor prefers installation in an R directory under root, rather than Program Files:

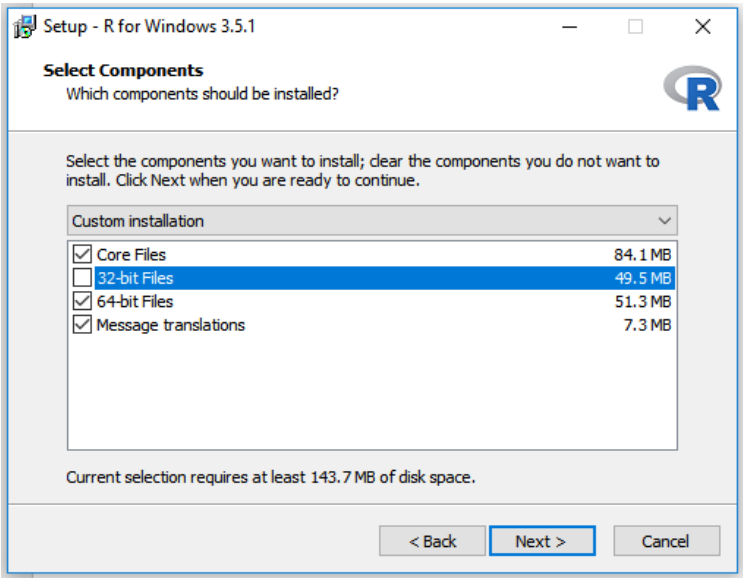




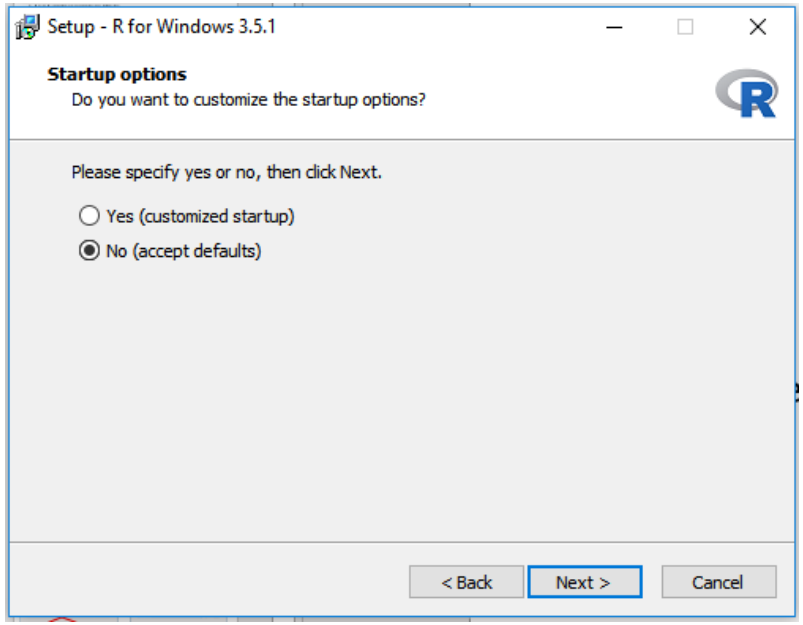
- Alternatively, create a subdirectory called **`Program_Files`**
- In general, recommended that you ***<u>do not</u>*** use a directory with split words (for more advanced extensions you may wish to use down the road)

Installing Base R from CRAN

- Make your life simpler by unchecking the 32-bit version:

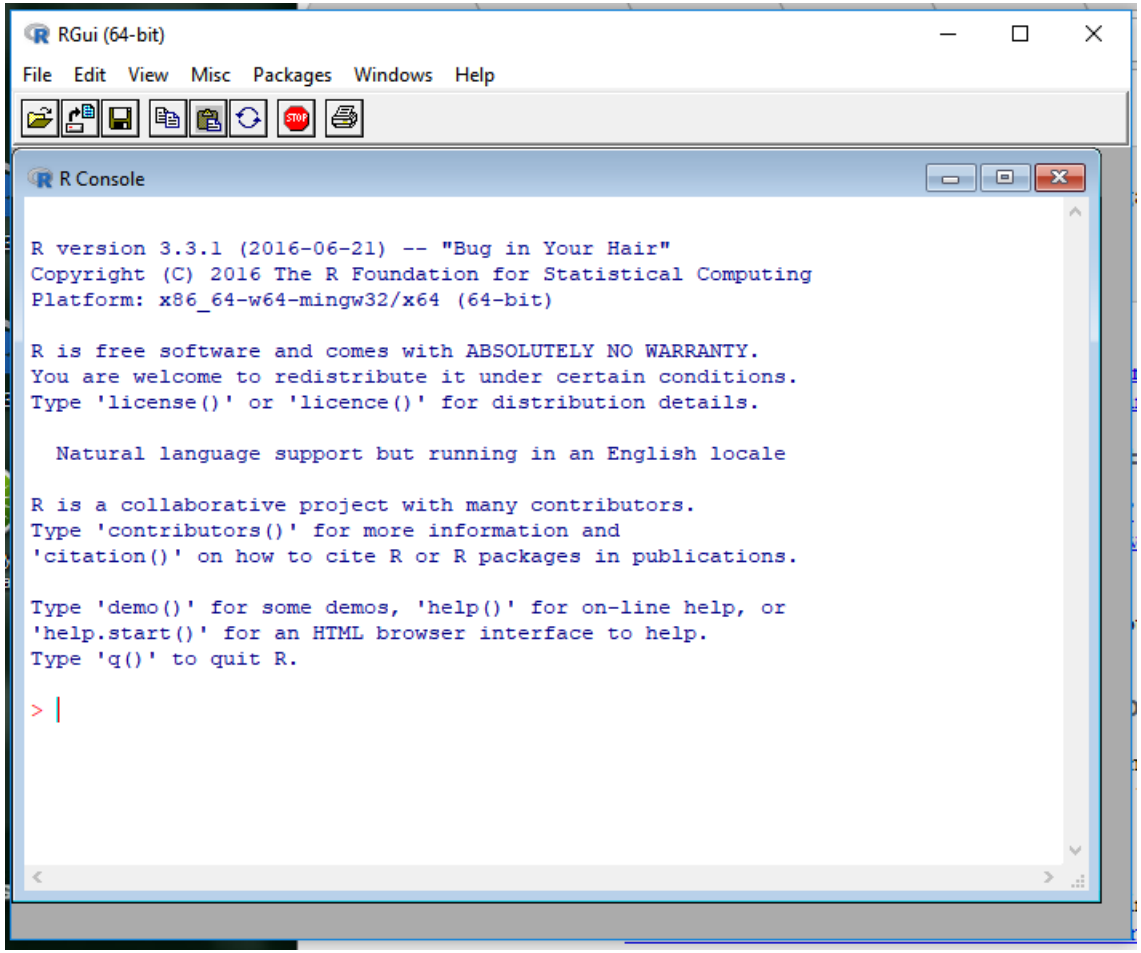


- And, accept the defaults otherwise:
  - Two more installation screens after this
  - Accept those defaults also



Run Base R

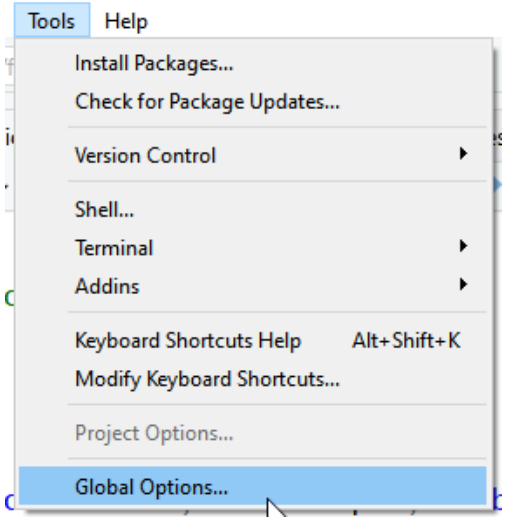
- Base R comes with its own GUI:

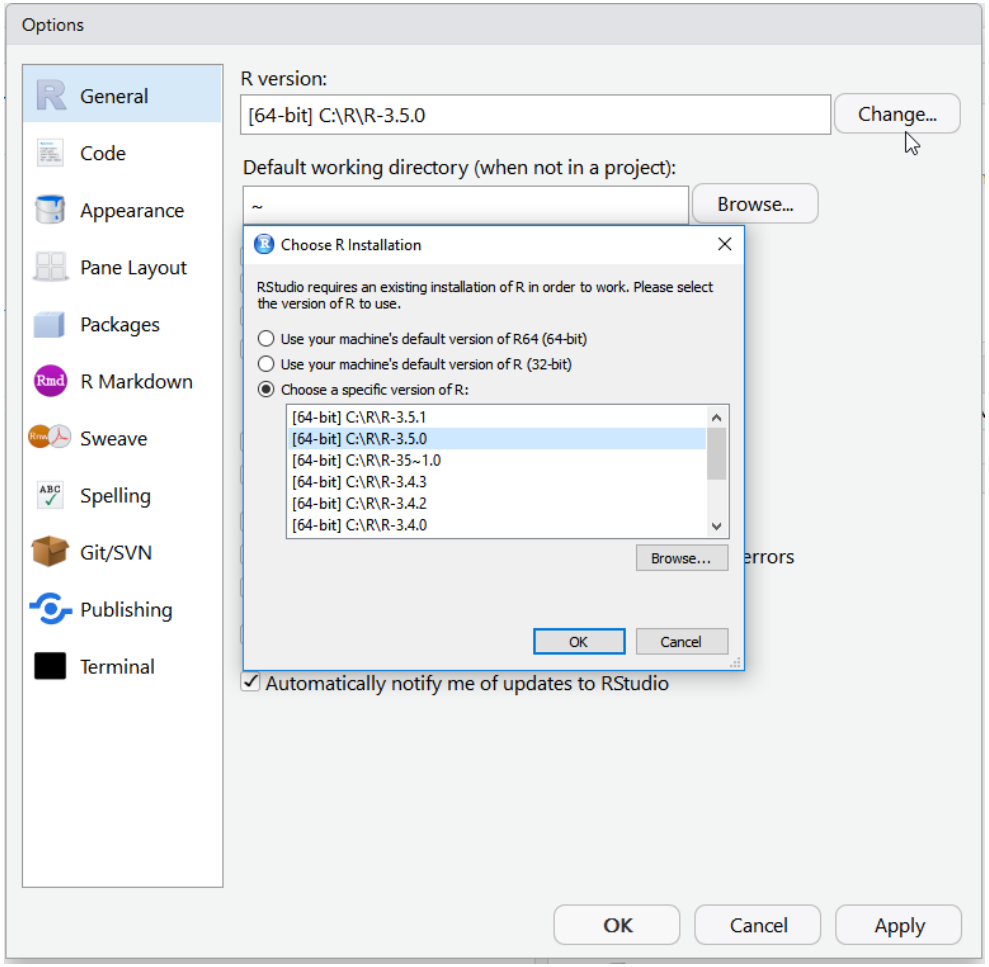


- However, we will use the more robust RStudio IDE

RStudio

- Download from https://www.rstudio.com/products/rstudio/#Desktop
- If CRAN R is already installed, RStudio should find it
- If not, choose Tools/Global Options… from the menus at top:
- Then, select the version of R you wish to use:





Help System in RStudio
Installing and Loading R Packages
Quitting an R Session

Accessing Help in RStudio

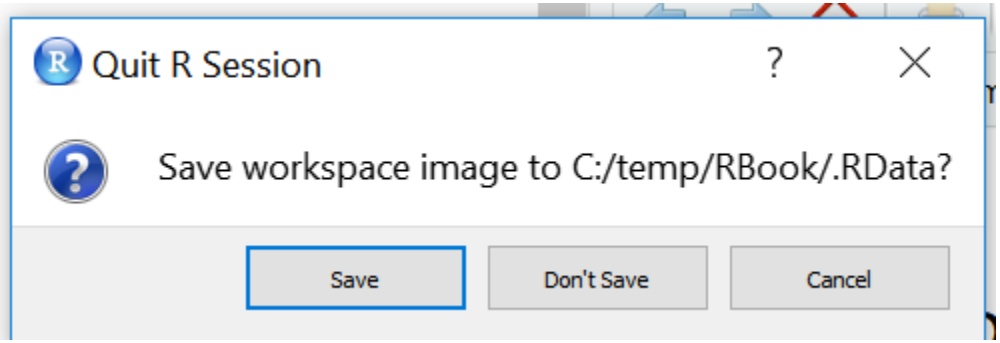
- In the previous slides, we saw that we can get help in R as follows:
  - `?specific_function_or_topic`
  - `??general_topic_involving_a_search`
- These will also work in the RStudio console
- However, a very convenient tool in RStudio is its own local set of help files that don't involve a web search
- Let's look at the help files for
  - **`pi`**
  - **`sin(.)`**
  - **`xyplot(.)`**
  - **`merge(.)`** (and the overloaded merge in the xts package)
- See examples in class
- Note the examples at the end of the help files that can be run in your R session
- Help files in R follow a standard format

Installing and Loading an R Package

- One of the packages we will use a lot in this class is the quantmod package
- It is used for accessing and loading market data
- In RStudio, go to the Tools menu, and choose Install Packages… from the drop-down selections
- See the in-class example

Quitting an R Session

- When quitting a R session in RStudio (and similar in the base R GUI), you will get a message as follows:



- Just **choose "Don't Save".** It won't hurt you if you choose "Save", but we do not need to save the workspace for anything, at least yet

Phew – that was a long one!

[END]