The R Environment

FRE6871 & FRE7241, Spring 2016

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

July 8, 2016

NYU | TANDON SCHOOL OF ENGINEERING

What is R?

- Is an open-source software environment for statistical computing and graphics,
- Is an interpreted language, allowing interactive code development,
- Is a functional language where every operator is an R function,
- Supports object-oriented programming with *classes* and *methods*,
- Is a very expressive language that allows performing complex operations with very few lines of code,
- Has metaprogramming facilities that allow programming on the language,
- Is written in R itself and in C/C++,
- Has vectorized functions written in C/C++, allowing very fast execution of loops over vector elements,
- Is extended through user-created packages, providing for the latest developments, such as *Machine Learning*,

http://www.r-project.org/
http://en.wikipedia.org/wiki/R_(programming_language)

The R License

R is open-source software released under the GNU General Public License:

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

Some other R packages are released under the Creative Commons Attribution-ShareAlike License:

http://creativecommons.org

Installing R and *RStudio*

Students will be required to bring their laptop computers to all the lectures, and to run the R Interpreter and **RStudio** RStudio during the lecture,

Laptop computers will be necessary for following the lectures, and for performing tests,

Students will be required to install and to become proficient with the R Interpreter,
Students can download the R Interpreter from CRAN (Comprehensive R Archive Network):

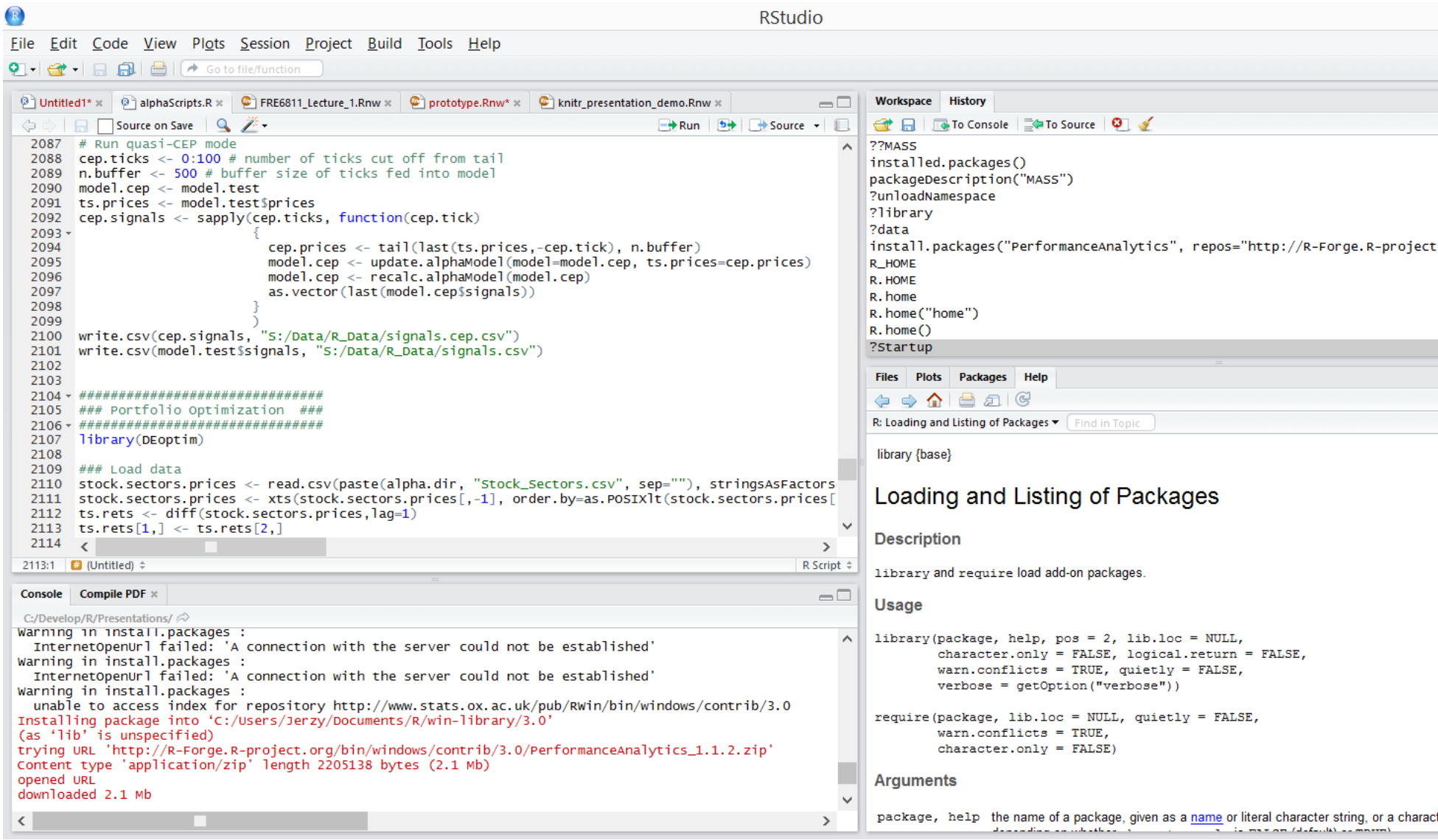
http://cran.r-project.org/

To invoke the RGui interface, click on:
C:/Program Files/R/R-3.1.2/bin/x64/RGui.exe

Students will be required to install and to become proficient with the *RStudio* Integrated Development Environment (*IDE*),

http://www.rstudio.com/products/rstudio/

Using *RStudio*



A First R Session

Variables are created by an assignment operation, and they don't have to be declared,

The standard assignment operator in R is the arrow symbol "<-",

R interprets text in quotes ("") as character strings,

Text that is not in quotes ("") is interpreted as a `symbol` or `expression`,

Typing a `symbol` or `expression` evaluates it,

All text after the "#" symbol is treated as comments,

```
> my_var <- 3  # "<-" and "=" are valid assignment operato
>
> my_var  # typing a symbol or expression evaluates it
>
> my_var <- "Hello World!"  # text in quotes is interprete
>
> my_var  # typing a symbol or expression evaluates it
```

Exploring an R Session

The function `getwd()` returns a vector of length 1, with the first element containing a string with the name of the current working directory (`cwd`),

The function `setwd()` accepts a character string as input (the name of the directory), and sets the working directory to that string,

R is a functional language, and R commands are functions, so they must be followed by parentheses "`()`",

```
> getwd()  # get cwd
> setwd("C:/Develop/R")  # set cwd
> getwd()  # get cwd
```

Get system date and time

Just the date

```
> Sys.time()  # get date and time
>
> Sys.Date()  # get date only
```

The R Workspace

The workspace is the current R working environment, which includes all user-defined objects and the command history,

The function `ls()` returns names of objects in the R workspace,

The function `rm()` removes objects from the R workspace,

The workspace can be saved into and loaded back from an `*.RData` file (binary file format),

The function `save.image()` saves the whole workspace,

The function `save()` saves just the selected objects,

The function `load()` reads data from `*.RData` files, and *invisibly* returns a vector of names of objects created in the workspace,

```
> var1 <- 3  # define new object
> ls()  # list all objects in workspace
[1] "var1"
> # list objects starting with "v"
> ls(pattern=glob2rx("v*"))
[1] "var1"
> save.image()  # save workspace to file .RData in cwd
> rm(var1)  # remove object
> ls()  # list objects
character(0)
> load(".RData")
> ls()  # list objects
[1] "var1"
> var2 <- 5  # define another object
> save(var1, var2,  # save selected objects
+      file="C:/Develop/data/my_data.RData")
> rm(list=ls())  # remove all objects
> ls()  # list objects
character(0)
> load_ed <- load(file="C:/Develop/data/my_data.RData")
> load_ed
[1] "var1" "var2"
> ls()  # list objects
[1] "load_ed" "var1"    "var2"
```

The R Workspace (cont.)

When you quit R you'll be prompted "Save workspace image?"

If you answer *YES* then the workspace will be saved into the `.RData` file in the `cwd`,

When you start R again, the workspace will be automatically loaded from the existing `.RData` file,

```
>   q()  # quit R session
```

The R Workspace (cont.)

When you quit R you'll be prompted "Save workspace image?"

If you answer *YES* then the workspace will be saved into the `.RData` file in the `cwd`,

When you start R again, the workspace will be automatically loaded from the existing `.RData` file,

```
>   q()  # quit R session
```

The function `history()` displays recent commands,

You can also save and load the command history from a file,

```
> history(5)  # display last 5 commands
> savehistory(file="myfile")  # default is ".Rhistory"
> loadhistory(file="myfile")  # default is ".Rhistory"
```

R Session Info

The function `sessionInfo()` returns information about the current R session,

- `R version`,
- `OS platform`,
- `locale` settings,
- list of packages that are loaded and attached to the search path,
- list of packages that are loaded, but *not* attached to the search path,

```
> sessionInfo()  # get R version and other session info
```

Environment Variables

R uses environment variables to store information about its environment, such as paths to directories containing files used by R (startup, history, OS),

For example the environment variables:

- R_USER and HOME store the R user Home directory,
- R_HOME stores the root directory of the R installation,

The functions Sys.getenv() and Sys.setenv() display and set the values environment variables,

Sys.getenv("env_var") displays the environment variable "env_var",

Sys.setenv("env_var=value") sets the environment variable "env_var" equal to "value",

```
> Sys.getenv()[5:7]  # list some environment variables
>
> Sys.getenv("Home")  # get R user HOME directory
>
> Sys.setenv(Home="C:/Develop/data")  # set HOME directory
>
> Sys.getenv("Home")  # get user HOME directory
>
> Sys.getenv("R_home")  # get R_HOME directory
>
> R.home()  # get R_HOME directory
>
> R.home("etc")  # get "etc" sub-directory of R_HOME
```

Global *Options* Settings

R uses a list of global *options* which affect how R computes and displays results,

The function `options()` either sets or displays the values of global *options*,

`options("globop")` displays the current value of option `"globop"`,

`getOption("globop")` displays the current value of option `"globop"`,

`options(globop=value)` sets the option `"globop"` equal to `"value"`,

```
> # ?options  # long list of global options
> # interpret strings as characters, not factors
> getOption("stringsAsFactors")  # display option
> options("stringsAsFactors")  # display option
> options(stringsAsFactors=FALSE)  # set option
> # number of digits printed for numeric values
> options(digits=3)
> # control exponential scientific notation of print metho
> # positive "scipen" values bias towards fixed notation
> # negative "scipen" values bias towards scientific notat
> options(scipen=100)
> # maximum number of items printed to console
> options(max.print=30)
> # warning levels options
> # negative - warnings are ignored
> options(warn=-1)
> # zero - warnings are stored and printed after top-level
> options(warn=0)
> # one - warnings are printed as they occur
> options(warn=1)
> # two or larger - warnings are turned into errors
> options(warn=2)
> # save all options in variable
> op_tions <- options()
> # restore all options from variable
> options(op_tions)
```

Constructing File Paths

Names of *file paths* can be constructed using the function `paste()`,

The function `file.path()` is similar to `paste()`, but automatically uses the correct separator for the platform,

The function `normalizePath()` performs tilde-expansions and displays file paths in user-readable format,

```
> # R startup (site) directory
> paste(R.home(), "etc", sep="/")
>
> file.path(R.home(), "etc")  # better way
>
> # perform tilde-expansions and convert to readable forma
> normalizePath(file.path(R.home(), "etc"), winslash="/")
>
> normalizePath(R.home("etc"), winslash="/")
```

R System Directories under Windows

R uses several different directories to search, read, and store files:

- Windows user personal directory: "~" ("%USERPROFILE%/Documents"),
- R user HOME directory (R_USER and Home),
- cwd current working directory - the default directory for storing and retrieving user files (such as .Rhistory, *.RData, etc.),
- R_HOME root directory of the R installation,
- R startup (site) directory: R_HOME/etc/,

By default, the R user HOME directory is the Windows user personal directory,

The cwd is set to the directory from which R is invoked, or the R user HOME directory,

```
> normalizePath("~", winslash="/")  # Windows user HOME di
>
> Sys.getenv("Home")  # R user HOME directory
>
> setwd("C:/Develop/R")
> getwd()  # current working directory
>
> # R startup (site) directory
> normalizePath(file.path(R.home(), "etc"), winslash="/")
>
> # R executable directory
> normalizePath(file.path(R.home(), "bin/x64"), winslash="
>
> # R documentation directory
> normalizePath(file.path(R.home(), "doc/manual"), winslas
```

File and Directory Listing Functions

The functions `list.files()` and `dir()` return a vector of names of files in a given directory,

`list.dirs()` lists the directories in a given directory,

`Sys.glob()` lists files matching names obtained from wildcard expansion,

```
> setwd("C:/Develop/data")
> sample(dir(), 5)  # get 5 file names - dir() lists all
> sample(dir(pattern="csv"), 5)  # list files containing
> sample(list.files(R.home()), 5)  # all files in R_HOME
> sample(list.files(R.home("etc")), 5)  # all files in "et
> sample(list.dirs(), 5)  # directories in cwd
> list.dirs(R.home("etc"))  # directories in "etc" sub-di
> sample(Sys.glob("*.csv"), 5)
> Sys.glob(R.home("etc"))
```

Invoking an R Session in Windows

An R session can run in several different ways:

- In an R terminal (by invoking R.exe or Rterm.exe),
- In an R RGui (by invoking RGui.exe),
- In an *RStudio* session (or some other IDE),

The initial value of the cwd depends on how the R session is invoked.
If R is invoked:

- from the Windows menu, then cwd is set to the R user HOME directory,
- by clicking on a file (*.R, *.RData, etc.), then cwd is set to the file's directory,
- by typing R.exe or Rterm.exe in the command shell (after setting the PATH), then cwd is set to the directory where the command was typed,

```
> getwd()  # get cwd
```

R Session Startup

At startup R sources (reads) several types of files, in the following order:

- Renviron files defining environment variables,
- Rprofile files containing code executed at R startup,
- RData files containing data to be loaded at R startup,

R sources files from several directories, in the following order:

- R startup directory: Renviron.site and Rprofile.site files,
- cwd directory: .Renviron, .Rprofile, and .RData files,
- HOME user directory (only if no files found in cwd),

The above startup process can be customized by setting environment variables,

```
> # help(Startup)  # description of R session startup mecl
>
> # files in R startup directory directory
> dir(normalizePath(file.path(R.home(), "etc"), winslash="
>
> # *.R* files in cwd directory
> getwd()
> dir(getwd(), all.files=TRUE, pattern="\\.R")
> dir(getwd(), all.files=TRUE, pattern=glob2rx("*.R*"))
```

Customizing the R Environment

users can customize their R environments and workspace by creating custom startup files in different working directories. The Renviron and Rprofile files can be placed in any directory `Renviron` files defining environment variables, `Rprofile` files containing code executed at R startup, If R is invoked from a terminal, then the directory from which it's invoked will be sourced. At startup R searches for startup files in the `cwd` and R home directory, every directory can have its own special initialization file environment files (containing environment variables to be set), and .Rprofile files containing R scripts (code),
startup files may contain environment variables, option settings, and other R scripts startup profile file of R code C:/Program Files/R/R-3.1.2/
to process for setting environment

```
> setwd("C:/Develop/R")
>
> scan(file=".Rprofile", what=character(), sep="\n")
```

The Renviron files

At startup R searches for startup files in the cwd and R home directory, Environment variables can be supplied as "symbol=value" pairs on the command line. environment files (containing environment variables to be set), and .Rprofile files containing R scripts (code),
startup files may contain environment variables, option settings, and other R scripts startup profile file of R code C:/Program Files/R/R-3.1.2/
to process for setting environment variables. executes If no `.Rprofile` file is found in the startup directory, then R looks for a `.Rprofile` file in the user's home directory and uses that (if it exists). The function `getwd()` returns a vector of length 1, with the first element containing a string with the name of the current working directory (cwd), R sources the `.Rprofile` file in the current working directory or in the user's home

```
> cat("sourcing .Rprofile file\n")
>
>
```

The Rprofile files

At startup R searches for startup files in the cwd and R home directory, environment files (containing environment variables to be set), and .Rprofile files containing R scripts (code),
startup files may contain environment variables, option settings, and other R scripts startup profile file of R code C:/Program Files/R/R-3.1.2/
to process for setting environment variables. executes If no .Rprofile file is found in the startup directory, then R looks for a .Rprofile file in the user's home directory and uses that (if it exists). R sources the .Rprofile file in the current working directory or in the user's home directory (in that order) every directory can have its own custom initialization file

```
> cat("sourcing .Rprofile file\n")
>
>
```

Running R Scripts and Batch Processes

.Rprofile files (code), to execute a file foo.R containing R scripts
R CMD BATCH "–args arg1 arg2" foo.R &
Rscript -e "source('functions.txt');f1();f2()" ¿ out.txt
startup files may contain environment variables, option settings, and other R scripts startup profile file of R code C:/Program Files/R/R-3.1.2/
to process for setting environment variables. executes If no `.Rprofile` file is found in the startup directory, then R looks for a `.Rprofile` file in the user's home directory and uses that (if it exists). The function `getwd()` returns a vector of length 1, with the first element containing a string with the name of the current working directory (`cwd`), R sources the `.Rprofile` file in the current working directory or in the user's home directory (in that order) every

```
> # source("script.R", echo=TRUE)  # source script
```

Environments in R

Environments consist of a *frame* (a set of symbol-value pairs) and an *enclosure* (a pointer to an enclosing environment),

There are three system environments:

- `globalenv()` the user's workspace,
- `baseenv()` the environment of the base package,
- `emptyenv()` the only environment without an enclosure,

Environments form a tree structure of successive enclosures, with the empty environment at its root,

Packages have their own environments,

The enclosure of the base package is the empty environment,

```
> # get base environment
> baseenv()
> # get global environment
> globalenv()
> # get current environment
> environment()
> # get environment class
> class(environment())
> # define variable in current environment
> glob_var <- 1
> # get objects in current environment
> ls(environment())
> # create new environment
> new_env <- new.env()
> # get calling environment of new environment
> parent.env(new_env)
> # assign Value to Name
> assign("new_var1", 3, envir=new_env)
> # create object in new environment
> new_env$new_var2 <- 11
> # get objects in new environment
> ls(new_env)
> # get objects in current environment
> ls(environment())
> # environments are subset like lists
> new_env$new_var1
> # environments are subset like lists
> new_env[["new_var1"]]
```

The R Search Path

R evaluates variables using the search path, a series of environments:

- global environment,
- package environments,
- base environment,

The function `search()` returns the search path for R objects,

The function `attach()` attaches objects to the search path,

Using `attach()` allows referencing object components by their names alone, rather than as components of objects,

The function `detach()` detaches objects from the search path,

The function `find()` finds where objects are located on the search path,

```
> search()  # get search path for R objects
> my_list <-
+   list(flowers=c("rose", "daisy", "tulip"),
+        trees=c("pine", "oak", "maple"))
> my_list$trees
> attach(my_list)
> trees
> search()  # get search path for R objects
> detach(my_list)
> head(trees)  # "trees" is in datasets base pac
```

Rule of Thumb

Be very careful with using `attach()`,

Make sure to `detach()` objects once they're not needed,

Referencing Object Components Using with()

The function with() evaluates an expression in an environment constructed from the data,

with() allows referencing object components by their names alone,

It's often better to use with() instead of attach(),

```
> # "trees" is in datasets base package
> head(trees, 3)
> colnames(trees)
> mean(Girth)
> mean(trees$Girth)
> with(trees,
+      c(mean(Girth), mean(Height), mean(Volume)))
```

Writing Text Strings

The function `cat()` concatenates strings and writes them to standard output or to files,

`cat()` interprets its argument character string and its escape sequences ("\"), but doesn't return a value,

The function `print()` doesn't interpret its argument, and simply prints it to standard output and invisibly returns it,

Typing the name of an object in `R` implicitly calls `print()` on that object,

The function `save()` writes objects to a binary file,

```
> cat("Enter\ttab")  # cat() interprets backslash escape s
> print("Enter\ttab")
>
> my_text <- print("hello")
> my_text  # print() returns its argument
>
> # create string
> my_text <- "Title: My Text\nSome numbers: 1,2,3,...\nRpr
>
> cat(my_text, file="mytext.txt")  # write to text file
>
> cat("Title: My Text",  # write several lines to text fil
+     "Some numbers: 1,2,3,...",
+     "Rprofile files contain code executed at R startup,"
+     file="mytext.txt", sep="\n")
>
> save(my_text, file="mytext.RData")  # write to binary fi
```

Displaying Numeric Data

The function `print()` displays numeric data objects, with the number of digits given by the global option `"digits"`,

The function `sprintf()` returns strings formatted from text strings and numeric data,

```
> print(pi)
> print(pi, digits=10)
> getOption("digits")
> foo <- 12
> bar <- "months"
> sprintf("There are %i %s in the year", foo, bar)
```

Reading Text from Files

The function `scan()` reads text or data from a file and returns it as a vector or a list,

The function `readLines()` reads lines of text from a connection (file or console), and returns them as a vector of `character` strings,

The function `readline()` reads a single line from the console, and returns it as a `character` string,

The function `file.show()` reads text or data from a file and displays in editor,

```
> # read text from file
> scan(file="mytext.txt", what=character(), sep="\n")
>
> # read lines from file
> readLines(con="mytext.txt")
>
> # read text from console
> in_put <- readline("Enter a number: ")
> class(in_put)
> # coerce to numeric
> in_put <- as.numeric(in_put)
>
> # read text from file and display in editor:
> # file.show("mytext.txt")
> # file.show("mytext.txt", pager="")
```

Reading and Writing Data Frames from *Text* Files

The functions `read.table()` and `write.table()` read and write data frames from text files,

`write.table()` coerces objects to data frames before it writes them,

`read.table()` returns a data frame, and coerces non-numeric values to `factors` (unless the `stringsAsFactors=FALSE` option is set),

`read.table()` and `write.table()` can be used to read and write matrices from text files, but they have to be coerced back to matrices,

`read.table()` and `write.table()` are inefficient for very large data sets,

```
> # write data frame to text file, and then read it back
> write.table(data_frame, file="florist.txt")
> data_read <- read.table(file="florist.txt")
> data_read  # a data frame
>
> # write matrix to text file, and then read it back
> write.table(mat_rix, file="matrix.txt")
> mat_read <- read.table(file="matrix.txt")
> mat_read  # write.table() coerced matrix to data frame
> class(mat_read)
> # coerce from data frame back to matrix
> mat_read <- as.matrix(mat_read)
> class(mat_read)
```

Copying Data Frames Between the *clipboard* and R

Data frames stored in the *clipboard* can be copied into R using the function `read.table()`,

Data frames in R can be copied into the *clipboard* using the function `write.table()`,

This allows convenient copying of data frames between *Excel* and R,

Data frames can also be manipulated directly in the R spreadsheet-style data editor,

```
> data_frame <- read.table("clipboard", header=TRUE)
>
> write.table(x=data_frame, file="clipboard", sep="\t")
>
> # wrapper function for copying data frame from clipboard
> # by default, data is tab delimited, with a header
> read_clip <- function(file="clipboard", sep="\t",
+                header=TRUE, ...) {
+   read.table(file=file, sep=sep, header=header, ...)
+ }  # end read_clip
>
> data_frame <- read_clip()
>
> # wrapper function for copying data frame from R into cl
> # by default, data is tab delimited, with a header
> write_clip <- function(data, row.names=FALSE,
+                 col.names=TRUE, ...) {
+   write.table(x=data, file="clipboard", sep="\t",
+       row.names=row.names, col.names=col.names, ...)
+ }  # end write_clip
>
> write_clip(data=data_frame)
>
> # launch spreadsheet-style data editor
> data_frame <- edit(data_frame)
```

Reading and Writing Data Frames from *csv* Files

The functions `read.csv()` and `write.csv()` read and write data frames from *csv* format files,

These functions are wrappers for `read.table()` and `write.table()`,

`read.csv()` coerces non-numeric values to `factors`, unless the `stringsAsFactors=FALSE` option is set,

`read.csv()` reads row names as an extra column, unless the `row.names=1` argument is used,

The argument `"row.names"` accepts either the number or the name of the column containing the row names,

The `*.csv()` functions are very inefficient for large data sets,

```
> # write data frame to CSV file, and then read it back
> write.csv(data_frame, file="florist.csv")
> data_read <- read.csv(file="florist.csv",
+                       stringsAsFactors=FALSE)
> data_read  # the row names are read in as extra column
> # restore row names
> rownames(data_read) <- data_read[, 1]
> data_read <- data_read[, -1]  # remove extra column
> data_read
> # read row names from first column
> data_read <- read.csv(file="florist.csv", row.names=1)
> data_read
```

Reading and Writing Data Frames from *csv* Files (cont.)

The functions `read.csv()` and `write.csv()` can read and write data frames from *csv* format files *without using row names*,

Row names can be omitted from the output file by calling `write.csv()` with the argument `row.names=FALSE`,

```
> # write data frame to CSV file, without row names
> write.csv(data_frame, row.names=FALSE, file="florist.csv
> data_read <- read.csv(file="florist.csv")
> data_read  # a data frame without row names
```

Reading and Writing Matrices from *csv* Files

The functions `read.csv()` and `write.csv()` can read and write matrices from *csv* format files,

If row names can be omitted in the output file, then `write.csv()` can be called with argument `row.names=FALSE`,

If the input file doesn't contain row names, then `read.csv()` can be called without the `"row.names"` argument,

```
> # write matrix to csv file, and then read it back
> write.csv(mat_rix, file="matrix.csv")
> mat_read <- read.csv(file="matrix.csv", row.names=1)
> mat_read  # read.csv() reads matrix as data frame
> class(mat_read)
> mat_read <- as.matrix(mat_read)  # coerce to matrix
> identical(mat_rix, mat_read)
> write.csv(mat_rix, row.names=FALSE,
+     file="matrix_ex_rows.csv")
> mat_read <- read.csv(file="matrix_ex_rows.csv")
> mat_read <- as.matrix(mat_read)
> mat_read  # a matrix without row names
```

Reading and Writing Matrices (cont.)

There are several ways of reading and writing matrices from *csv* files, with tradeoffs between simplicity, data size, and speed,

The function `write.matrix()` writes a matrix to a text file, without its row names,

`write.matrix()` is part of package `MASS`,

The advantage of function `scan()` is its speed, but it doesn't handle row names easily,

Removing row names simplifies the reading and writing of matrices,

The function `readLines` reads whole lines and returns them as single strings,

The function `system.time()` calculates the execution time (in seconds) used to evaluate a given expression,

```
> library(MASS)  # load package "MASS"
> # write to CSV file by row - it's very SLOW!!!
> write.matrix(mat_rix, file="matrix.csv", sep=",")
> system.time(  # scan reads faster - skip first line with
+   mat_read <- scan(file="matrix.csv", sep=",",
+             skip=1, what=numeric()))
> col_names <- readLines(con="matrix.csv", n=1)  # read co
> col_names  # this is a string!
> col_names <- strsplit(col_names, s=",")[[1]]  # convert
> mat_read  # mat_read is a vector, not matrix!
> # coerce by row to matrix
> mat_read <- matrix(mat_read, ncol=length(col_names),
+             byrow=TRUE)
> colnames(mat_read) <- col_names  # restore colnames
> mat_read
```

Reading Matrices Containing Bad Data

Very often data that is read from external sources contains elements with bad data,

An example of bad data are `character` strings in `numeric` data,

Columns of numeric data that contain strings are coerced to `character` or `factor`, when they're read by `read.csv()`,

`as.numeric()` coerces strings that don't represent numbers into `NA` values,

```
> # read data from a csv file, including row names
> mat_rix <- read.csv(file="matrix_bad.csv", row.names=1,
+                     stringsAsFactors=FALSE)
> mat_rix
> class(mat_rix)
> # columns with bad data are character or factor
> sapply(mat_rix, class)
> row_names <- row.names(mat_rix)  # copy row names
> # sapply loop over columns and coerce to numeric
> mat_rix <- sapply(mat_rix, as.numeric)
> row.names(mat_rix) <- row_names  # restore row names
> # replace NAs with zero
> mat_rix[is.na(mat_rix)] <- 0
> # matrix without NAs
> mat_rix
```

Reading and Writing zoo Series From *Text* Files

The package zoo contains functions `read.zoo()` and `write.zoo()` for reading and writing zoo objects from *text* and *csv* files,

`read.zoo()` and `write.zoo()` are wrappers for `read.table()` and `write.table()`,

By default these functions read and write data in *space*-delimited format, but they can also read and write data to *comma*-delimited *csv* files by passing the parameter `sep=","`,

```
> # create zoo with Date index
> in_dex <- seq(from=as.Date("2013-06-15"),
+               by="day", length.out=100)
> zoo_series <- zoo(cumsum(rnorm(length(in_dex))),
+               order.by=in_dex)
> tail(zoo_series, 3)
> # write zoo to text file, and then read it back
> write.zoo(zoo_series, file="zoo_series.txt")
> zoo_series <- read.zoo("zoo_series.txt")  # read it back
> tail(zoo_series, 3)
```

Reading and Writing zoo Series With *Date-time* Index

If the index of a zoo series is a *date-time*, then `write.zoo()` writes the date and time fields as separate columns with a *space* between them,

To properly read separate date and time columns from *text* files, `read.zoo()` must be passed arguments `"index.column=list(1,2)"` and `"tz"`,

```
> # create zoo with POSIXct date-time index
> in_dex <- seq(from=as.POSIXct("2013-06-15"),
+               by="hour", length.out=1000)
> zoo_series <- zoo(cumsum(rnorm(length(in_dex))),
+               order.by=in_dex)
> tail(zoo_series, 3)
> # write zoo to text file, and then read it back
> write.zoo(zoo_series, file="zoo_series.txt")
> zoo_series <- read.zoo("zoo_series.txt")  # read it bacl
> # time field was read as a separate column
> tail(zoo_series, 3)
> # read and specify that second column is time field
> zoo_series <- read.zoo(file="zoo_series.txt",
+                        index.column=list(1,2),
+                        tz="America/New_York")
> tail(zoo_series, 3)
```

Reading and Writing zoo Series From *csv* Files

Very often *csv* files contain custom *date-time* formats, which need to be passed as parameters into `read.zoo()` for proper formatting,

The `"FUN"` argument of `read.zoo()` accepts a function for coercing columns of the input data into a *date-time* object suitable for the zoo index,

```
> # write zoo to CSV file, and then read it back
> write.zoo(zoo_series, file="zoo_series.csv", sep=",")
> zoo_series <- read.zoo(file="zoo_series.csv",
+             header=TRUE, sep=",", FUN=as.POSIXct,
+             tz="America/New_York")
> tail(zoo_series, 3)
> # read zoo from CSV file, with custom date-time format
> zoo_frame <- read.table(file="zoo_series2.csv", sep=",")
> tail(zoo_frame, 3)  # date-time format mm/dd/yyyy hh:mm
> zoo_series <- read.zoo(file="zoo_series2.csv",
+             header=TRUE, sep=",", FUN=as.POSIXct,
+             tz="America/New_York",
+             format="%m/%d/%Y %H:%M")
> tail(zoo_series, 3)
```

Passing Arguments to the save() Function

The function save() writes objects to a binary file,

Object names can be passed into save() either through the "..." argument, or the "list" argument,

Objects passed through the "..." argument are not evaluated, so they must be either object names or character strings,

Object names aren't surrounded by quotes "", while character strings that represent object names are surrounded by quotes "",

Objects passed through the "list" argument are evaluated, so they may be variables containing character strings,

```
> var1 <- 1; var2 <- 2
> ls()  # list all objects
> ls()[1]  # list first object
> args(save)  # list arguments of save function
> # save "var1" to a binary file
> save("var1", file="my_data.RData")  # use string
> save(var1, file="my_data.RData")  # use object name
> save(var1, var2, file="my_data.RData")  # mulltiple obje
> # save first list object "var1" by passing it to the ".
> save(ls()[1], file="my_data.RData")  # 'ls()[1]' not eva
> # save first list object "var1" by passing it to the "li
> save(list=ls()[1], file="my_data.RData")
> # save whole list by passing it to the "list" argument
> save(list=ls(), file="my_data.RData")
```

Reading and Writing Lists of Objects

The function `load()` reads data from `*.RData` files, and *invisibly* returns a vector of names of objects created in the workspace,

The vector of names can be used to manipulate the objects in loops, or to pass them to functions,

```
> rm(list=ls())  # remove all objects
> # load objects from file
> load_ed <- load(file="my_data.RData")
> load_ed  # vector of loaded objects
> ls()  # list objects
> # assign new values to objects in  global environment
> sapply(load_ed, function(sym_bol) {
+   assign(sym_bol, runif(1), envir=globalenv())
+ })  # end sapply
> ls()  # list objects
> # assign new values to objects using for loop
> for (sym_bol in load_ed) {
+   assign(sym_bol, runif(1))
+ }  # end for
> ls()  # list objects
> # save vector of objects
> save(list=load_ed, file="my_data.RData")
> # remove only loaded objects
> rm(list=load_ed)
> # remove the object "load_ed"
> rm(load_ed)
```

Saving Output of R to a File

The function `sink()` diverts R *text* output (excluding *graphics*) to a file, or ends the diversion,

Remember to call `sink()` to end the diversion!

The function `pdf()` diverts *graphics* output to a pdf file (text output isn't diverted), in vector graphics format,

The functions `png()`, `jpeg()`, `bmp()`, and `tiff()` divert *graphics* output to graphics files (text output isn't diverted), in pixel graphics format,

The function `dev.off()` ends the diversion,

```
> {
+ sink("sinkdata.txt")# redirect text output to file
+
+ cat("Redirect text output from R\n")
+ print(runif(10))
+ cat("\nEnd data\nbye\n")
+
+ sink()  # turn redirect off
+
+ pdf("Rgraph.pdf", width=7, height=4)  # redirect graphi
+
+ cat("Redirect data from R into pdf file\n")
+ my_var <- seq(-2*pi, 2*pi, len=100)
+ plot(x=my_var, y=sin(my_var), main="Sine wave",
+    xlab="", ylab="", type="l", lwd=2, col="red")
+ cat("\nEnd data\nbye\n")
+
+ dev.off()  # turn pdf output off
+
+ png("Rgraph.png")  # redirect output to png file
+
+ cat("Redirect graphics from R into png file\n")
+ plot(x=my_var, y=sin(my_var), main="Sine wave",
+  xlab="", ylab="", type="l", lwd=2, col="red")
+ cat("\nEnd data\nbye\n")
+
+ dev.off()  # turn png output off
+ }
```

Internal R Help and Documentation

The function `help()` displays documentation on a function or subject,

Preceding the keyword with a single `"?"` is equivalent to calling `help()`,

```
> # display documentation on function "getwd"
> help(getwd)
> ?getwd  # equivalent to "help(getwd)"
```

Internal R Help and Documentation

The function `help()` displays documentation on a function or subject,

Preceding the keyword with a single `"?"` is equivalent to calling `help()`,

```
> # display documentation on function "getwd"
> help(getwd)
> ?getwd  # equivalent to "help(getwd)"
```

The function `help.start()` displays a page with links to internal documentation,

R documentation is also available in RGui under the help tab,

The pdf files with R documentation are also available directly under:
C:/Program Files/R/R-3.1.2/doc/manual/
(the exact path will depend on the R version.)

```
> help.start()  # open the hypertext documentati
```

Internal R Help and Documentation

The function `help()` displays documentation on a function or subject,

Preceding the keyword with a single `"?"` is equivalent to calling `help()`,

```
> # display documentation on function "getwd"
> help(getwd)
> ?getwd  # equivalent to "help(getwd)"
```

The function `help.start()` displays a page with links to internal documentation,

R documentation is also available in RGui under the help tab,

The pdf files with R documentation are also available directly under:
C:/Program Files/R/R-3.1.2/doc/manual/
(the exact path will depend on the R version.)

```
> help.start()  # open the hypertext documentati
```

"Introduction to R" by Venables and R Core Team:
Venables. *An Introduction to R*. . URL: http://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

R Online Help and Documentation

R Programming Wikibook

Wikibooks are crowdsourced textbooks

http://en.wikibooks.org/wiki/R_Programming/

R FAQ

Frequently Asked Questions about R

http://cran.r-project.org/doc/FAQ/R-FAQ.html

R-seek Online Search Tool

R-seek allows online searches specific to the R language

http://www.rseek.org/

R-help Mailing List

R-help is a very comprehensive Q&A mailing list

https://stat.ethz.ch/mailman/listinfo/r-help

R-help has archives of past Q&A - search it before you ask

https://stat.ethz.ch/pipermail/r-help/

GMANE allows searching the R-help archives using a usenet newsgroup style GUI

http://news.gmane.org/gmane.comp.lang.r.general

Stack Exchange

Stack Overflow

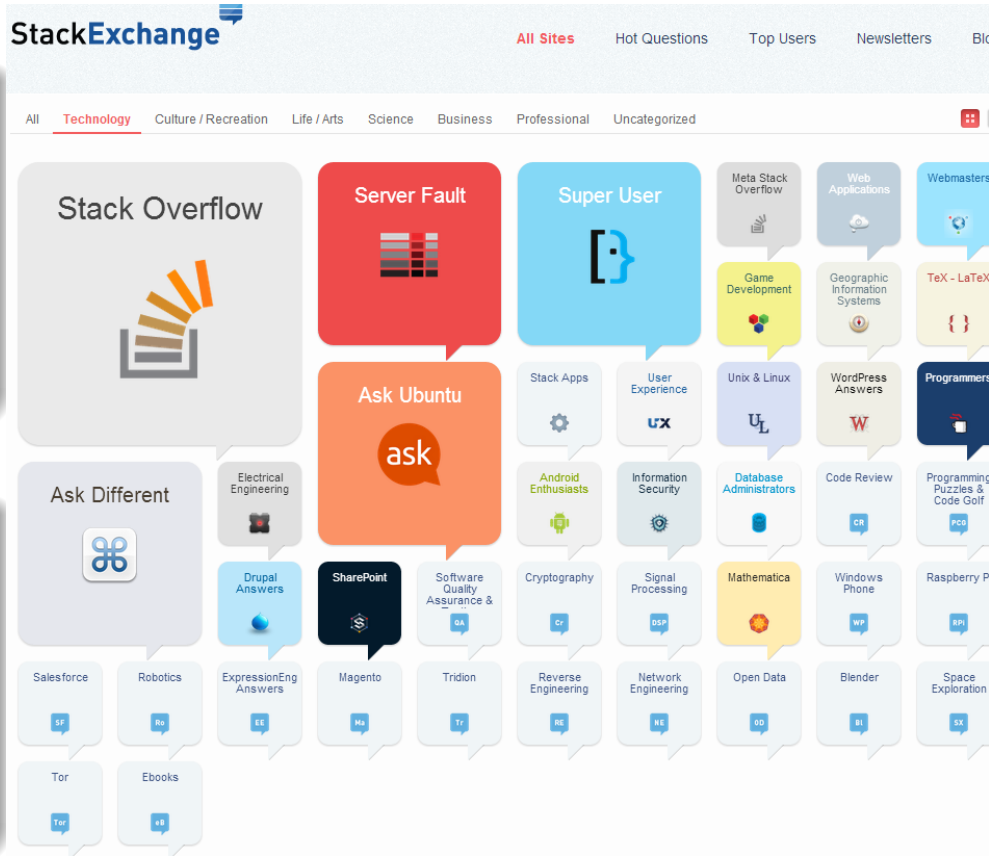
Stack Overflow is a Q&A forum for computer programming, and is part of Stack Exchange

- http://stackoverflow.com
- http://stackoverflow.com/questions/tagged/r
- http://stackoverflow.com/tags/r/info

Stack Exchange

Stack Exchange is a family of Q&A forums in a variety of fields

- http://stackexchange.com/
- http://stackexchange.com/sites#technology
- http://quant.stackexchange.com/



RStudio Support

RStudio has extensive online help, Q&A database, and documentation

https://support.rstudio.com/hc/en-us

https://support.rstudio.com/hc/en-us/sections/200107586-Using-RStudio

https://support.rstudio.com/hc/en-us/sections/200148796-Advanced-Topics

R Online Books and Courses

Companion website to the book *"Advanced R"* by Hadley Wickham - chief scientist at *RStudio*

The best book for learning the advanced features of R: http://adv-r.had.co.nz/

R Online Books and Courses

Companion website to the book *"Advanced R"* by Hadley Wickham - chief scientist at *RStudio*

The best book for learning the advanced features of R: http://adv-r.had.co.nz/

Endmemo web book

Good, but not interactive: http://www.endmemo.com/program/R/

R Online Books and Courses

Companion website to the book *"Advanced R"* by Hadley Wickham - chief scientist at *RStudio*

The best book for learning the advanced features of R: http://adv-r.had.co.nz/

Endmemo web book

Good, but not interactive: http://www.endmemo.com/program/R/

Quick-R by Robert Kabacoff

Good, but not interactive: http://www.statmethods.net/

R Online Books and Courses

Companion website to the book *"Advanced R"* by Hadley Wickham - chief scientist at *RStudio*

The best book for learning the advanced features of R: http://adv-r.had.co.nz/

Endmemo web book

Good, but not interactive: http://www.endmemo.com/program/R/

Quick-R by Robert Kabacoff

Good, but not interactive: http://www.statmethods.net/

R for Beginners by Emmanuel Paradis

good, basic introduction to R: http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

R Online Books and Courses

Companion website to the book *"Advanced R"* by Hadley Wickham - chief scientist at *RStudio*

The best book for learning the advanced features of R: http://adv-r.had.co.nz/

Endmemo web book

Good, but not interactive: http://www.endmemo.com/program/R/

Quick-R by Robert Kabacoff

Good, but not interactive: http://www.statmethods.net/

R for Beginners by Emmanuel Paradis

good, basic introduction to R: http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

Cookbook for R by Winston Chang from *RStudio*

Good plotting, but not interactive: http://www.cookbook-r.com/

R Online Interactive Tutorials

Datacamp introduction to R

Interactive R tutorial, but rather basic: https://www.datacamp.com/courses/introduction-to-r/

R Online Interactive Tutorials

Datacamp introduction to R

Interactive R tutorial, but rather basic: https://www.datacamp.com/courses/introduction-to-r/

Try R

Interactive R tutorial, but rather basic: http://tryr.codeschool.com/

R Blogs and Experts

R-Bloggers

R-Bloggers is an aggregator of blogs dedicated to R

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

Tal Galili is the author of R-Bloggers and has his own excellent blog

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

R Blogs and Experts

R-Bloggers

R-Bloggers is an aggregator of blogs dedicated to R

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

Tal Galili is the author of R-Bloggers and has his own excellent blog

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

Dirk Eddelbuettel

Dirk is a *Top Answerer* for R questions on Stackoverflow, the author of the Rcpp package, and the CRAN Finance View

http://dirk.eddelbuettel.com/

http://dirk.eddelbuettel.com/code/

http://dirk.eddelbuettel.com/blog/

http://www.rinfance.com/

R Blogs and Experts

R-Bloggers

R-Bloggers is an aggregator of blogs dedicated to R

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

Tal Galili is the author of R-Bloggers and has his own excellent blog

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

Dirk Eddelbuettel

Dirk is a *Top Answerer* for R questions on Stackoverflow, the author of the Rcpp package, and the CRAN Finance View

http://dirk.eddelbuettel.com/

http://dirk.eddelbuettel.com/code/

http://dirk.eddelbuettel.com/blog/

http://www.rinfinance.com/

Romain Frangois

Romain is an R Enthusiast and Rcpp Hero

http://romainfrancois.blog.free.fr/

http://romainfrancois.blog.free.fr/index.php?tag/graphgallery

http://blog.r-enthusiasts.com/

More R Blogs and Experts

Revolution Analytics Blog

R blog by Revolution Analytics software vendor
http://blog.revolutionanalytics.com/

More R Blogs and Experts

Revolution Analytics Blog

R blog by Revolution Analytics software vendor
http://blog.revolutionanalytics.com/

RStudio Blog

R blog by *RStudio*
http://blog.rstudio.org/