The R Environment

FRE6871 & FRE7241, Spring 2025

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

March 28, 2025

NYU | TANDON SCHOOL OF ENGINEERING

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)
> # Equivalent to "help(getwd)"
> ?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.)

```
> # Open the hypertext documentation
> help.start()
```

Introduction to R by Venables and R Core Team.

R Online Help and Documentation

R Cheat Sheets

The R Cheat Sheets are a fast way to find what you want.

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

R Style Guides

DataCamp R style guide

The DataCamp R style guide is very close to what I have adopted:
DataCamp R style guide

Google R style guide

The Google R style guide is similar to DataCamp's:
Google R style guide

Stack Exchange

Stack Overflow

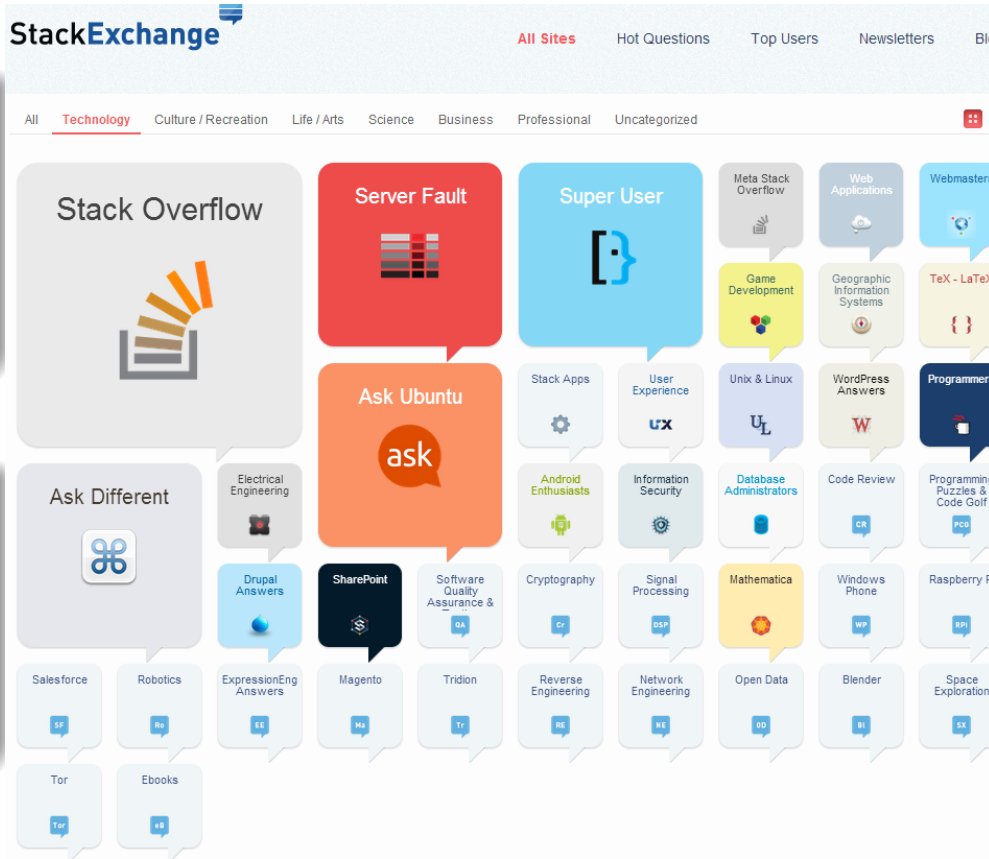
Stack Overflow is a Q&A forum for computer programming, and is part of Stack Exchange

- 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 References

Hadley Wickham book *Advanced R*

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

Cookbook for R by Winston Chang from *RStudio*

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

Efficient R programming by Colin Gillespie and Robin Lovelace

Good tips for fast R programming: https://csgillespie.github.io/efficientR/programming.html

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 Interactive Courses

Datacamp Interactive Courses

Datacamp introduction to R: https://www.datacamp.com/courses/introduction-to-r/

Datacamp list of free courses: https://www.datacamp.com/community/open-courses

Datacamp basic statistics in R: https://www.datacamp.com/community/open-courses/basic-statistics

Datacamp computational finance in R:
https://www.datacamp.com/community/open-courses/computational-finance-and-financial-econometrics-with-r

Datacamp machine learning in R:
https://www.datacamp.com/community/open-courses/kaggle-r-tutorial-on-machine-learning

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/

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/

RStudio Blog

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

GitHub for Hosting Software Projects Online

GitHub is an internet-based online service for hosting repositories of software projects.

GitHub provides version control using *git* (developed by Linus Torvalds).

Most R projects are now hosted on *GitHub*.

Google uses *GitHub* to host its *tensorflow* library for machine learning:

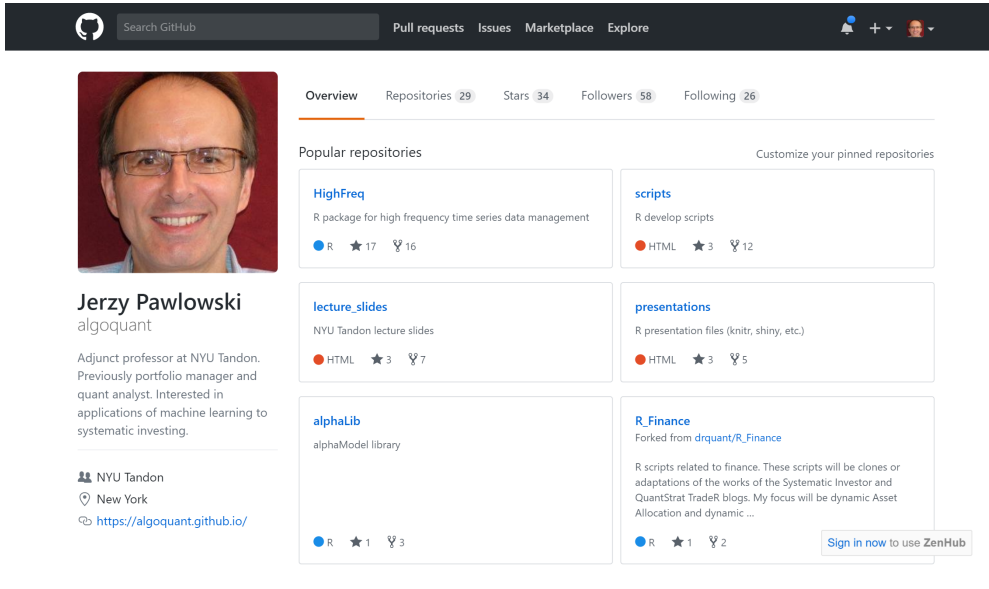
https://github.com/tensorflow/tensorflow

All the *FRE-7241* and *FRE-6871* lectures are hosted on *GitHub*:

https://github.com/algoquant/lecture_slides

https://github.com/algoquant

Hosting projects on *Google* is a great way to advertize your skills and network with experts.



What is R?

- An open-source software environment for statistical computing and graphics.
- An interpreted language, that allows interactive code development.
- A functional language where every operator is an R function.
- A very expressive language that can perform complex operations with very few lines of code.
- A language with metaprogramming facilities that allow programming on the language.
- A language written in C/C++, which can easily call other C/C++ programs.
- Can be easily extended with *packages* (function libraries), providing the latest developments like *Machine Learning*.
- Supports object-oriented programming with *classes* and *methods*.
- Vectorized functions written in C/C++, allow very fast execution of loops over vector elements.

Differences Between R and Python

R was designed for statistics and data science, while Python was designed as a general-purpose programming language.

- R was designed for statistics and data science - Python wasn't.
- R has native date and time objects built in - Python doesn't.
- R has native dataframe objects built in - Python doesn't.
- R has native vector and matrix objects built in - Python doesn't.
- R is designed to be easily extended with C++ code - Python isn't.

Why R is Better Than Python

Why is R More Difficult Than Other Languages?

R is more difficult than other languages because:

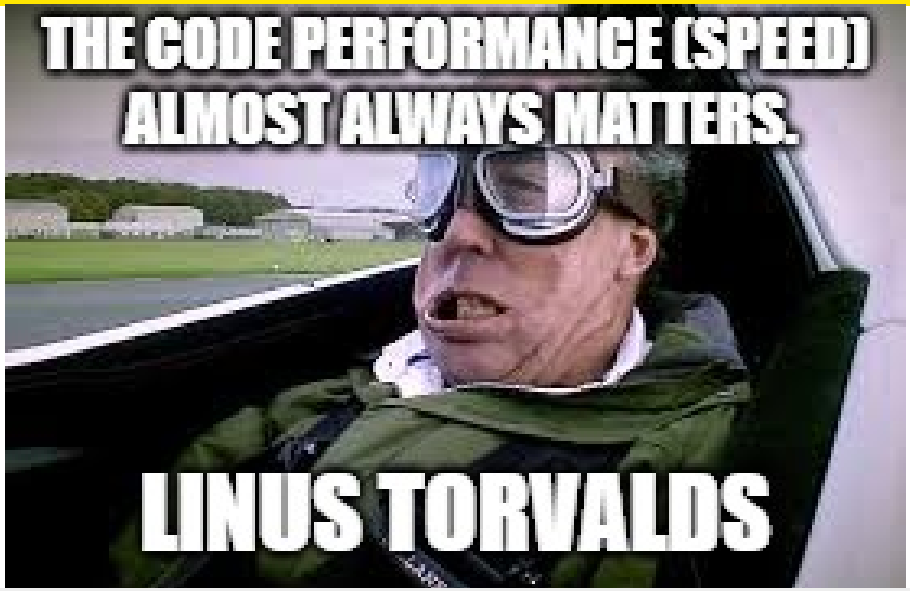
- R is a *functional* language, which makes its syntax unfamiliar to users of procedural languages like C/C++.
- The huge number of user-created *packages* makes it difficult to tell which are the best for particular applications.
- R can produce very cryptic *warning* and *error* messages, because it's a programming environment, so it performs many operations quietly, but those can sometimes fail.
- Fixing errors usually requires analyzing the complex structure of the R programming environment.

This course is designed to teach the most useful elements of R for financial analysis, through case studies and examples,

What are the Best Ways to Use R?

If used properly, R can be fast and interactive:

- Avoid using `apply()` and `for()` loops for large datasets.
- Pre-allocate memory for new objects.
- Avoid using too many R function calls (every command in R is a function).
- Use R as an interface to libraries written in C++, Java, and JavaScript.
- Use R functions which are *compiled* C++ code, instead of using interpreted R code.
- Write C++ functions in *Rcpp* and *RcppArmadillo*.
- Use package *data.table* for high performance data management.
- Use package *shiny* for interactive charts of live models running in R.
- Use package *dygraphs* for interactive time series plots.
- Use package *knitr* for *RMarkdown* documents.




```
> # Calculate cumulative sum of a vector
> vecv <- runif(1e5)
> # Use compiled function
> cumsumv <- cumsum(vecv)
> # Use for loop
> cumsumv2 <- vecv
> for (i in 2:NROW(vecv))
+   cumsumv2[i] <- (vecv[i] + cumsumv2[i-1])
> # Compare the two methods
> all.equal(cumsumv, cumsumv2)
> # Microbenchmark the two methods
> library(microbenchmark)
> summary(microbenchmark(
+   cumsum=cumsum(vecv),
+   loop_alloc={
+     cumsumv2 <- vecv
+     for (i in 2:NROW(vecv))
+ cumsumv2[i] <- (vecv[i] + cumsumv2[i-1])
+   },
+   loop_nalloc={
+     # Doesn't allocate memory to cumsumv3
```

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):

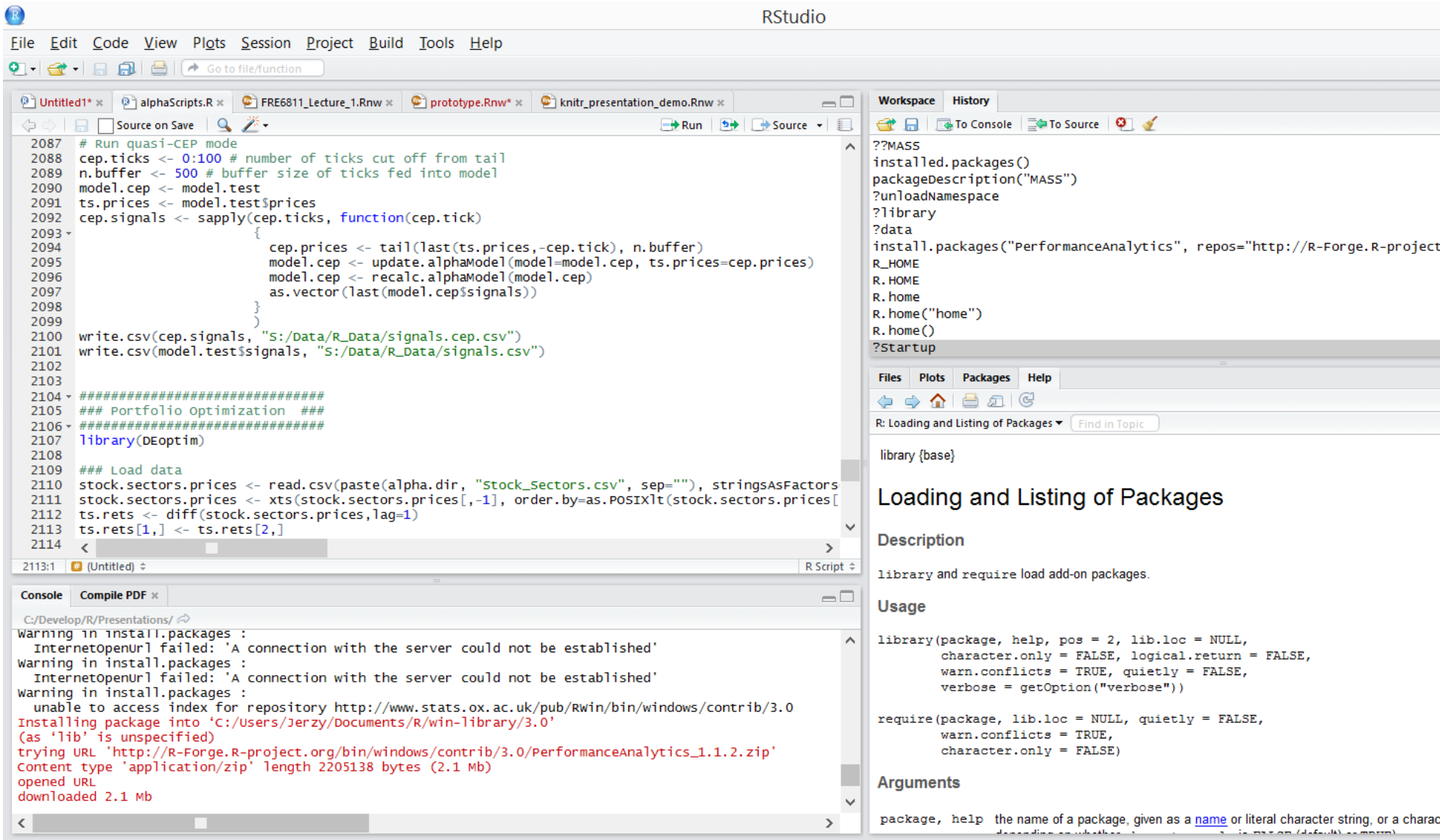
http://cran.r-project.org/

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.

R uses the hash "#" sign to mark text as comments.

All text after the hash "#" sign is treated as a comment, and is not executed as code.

```
> # "<-" and "=" are valid assignment operators
> myvar <- 3
>
> # Typing a symbol or expression evaluates it
> myvar
[1] 3
>
> # Text in quotes is interpreted as a string
> myvar <- "Hello World!"
>
> # Typing a symbol or expression evaluates it
> myvar
[1] "Hello World!"
>
> myvar  # Text after hash is treated as comment
[1] "Hello World!"
```

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("/Users/jerzy/Develop/R")  # Set cwd
> getwd()  # Get cwd
```

Get system date and time

Just the date

```
> Sys.time()  # Get date and time
[1] "2025-03-28 12:34:47 EDT"
>
> Sys.Date()  # Get date only
[1] "2025-03-28"
```

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 (compressed 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
> # List objects starting with "v"
> ls(pattern=glob2rx("v*"))
> # Delete all objects in workspace starting with "v"
> rm(list=ls(pattern=glob2rx("v*")))
> save.image()  # Save workspace to file .RData in cwd
> rm(var1)  # Remove object
> ls()  # List objects
> load(".RData")
> ls()  # List objects
> var2 <- 5  # Define another object
> save(var1, var2,  # Save selected objects
+      file="/Users/jerzy/Develop/lecture_slides/data/my_data.RData"
> rm(list=ls())  # Delete all objects in workspace
> ls()  # List objects
> loadobj <- load(file="/Users/jerzy/Develop/lecture_slides/data/my_
> loadobj
> ls()  # List objects
```

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
R version 4.4.1 (2024-06-14)
Platform: aarch64-apple-darwin20
Running under: macOS Ventura 13.3.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] graphics  grDevices utils     datasets  stats     methods   base

other attached packages:
[1] knitr_1.48       HighFreq_0.1     rutils_0.2       dygraphs_1.1
[5] quantmod_0.4.26  TTR_0.24.4       xts_0.14.0       zoo_1.8-12

loaded via a namespace (and not attached):
 [1] digest_0.6.36     fastmap_1.2.0     xfun_0.46         lattice_
 [5] magrittr_2.0.3    htmltools_0.5.8.1 cli_3.6.3         grid_4.4
 [9] compiler_4.4.1    highr_0.11        tools_4.4.1       rstudioap
[13] curl_6.2.1        evaluate_0.24.0   Rcpp_1.0.13       rlang_1.1
[17] htmlwidgets_1.6.4
```

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="/Users/jerzy/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 method
> # Positive "scipen" values bias towards fixed notation
> # Negative "scipen" values bias towards scientific notation
> 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-confl function
> options(warn=0)
> # One - warnings are printed as they occur
> options(warn=1)
> # 2 or larger - warnings are turned into errors
> options(warn=2)
> # Save all options in variable
> optionv <- options()
> # Restore all options from variable
> options(optionv)
```

Constructing File Paths

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

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

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

```
> # R startup (site) directory
> paste(R.home(), "etc", sep="/")
[1] "/Library/Frameworks/R.framework/Resources/etc"
>
> file.path(R.home(), "etc")  # Better way
[1] "/Library/Frameworks/R.framework/Resources/etc"
>
> # Perform tilde-expansions and convert to readable format
> normalizePath(file.path(R.home(), "etc"), winslash="/")
[1] "/Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/et
>
> normalizePath(R.home("etc"), winslash="/")
[1] "/Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/et
```

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 directory
>
> Sys.getenv("HOME")  # R user HOME directory
>
> setwd("/Users/jerzy/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"), winslash="/")
```

File and Directory Listing Functions

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

The function `list.dirs()` listv the directories in a given directory.

The function `Sys.glob()` listv files matching names obtained from wildcard expansion.

```
> sample(dir(), 5)  # Get 5 file names - dir() lists all files
> sample(dir(pattern="csv"), 5)  # List files containing "csv"
> sample(list.files(R.home()), 5)  # All files in R_HOME directory
> sample(list.files(R.home("etc")), 5)  # All files in "etc" sub-di
> sample(list.dirs(), 5)  # Directories in cwd
> list.dirs(R.home("etc"))  # Directories in "etc" sub-directory
> 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
[1] "/Users/jerzy/Develop/lecture_slides"
```

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 mechanism
>
> # 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*"))
```

draft: 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 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 directory can have its own custom initialization file

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

draft: 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 directory (in that order) every directory can have its own custom initialization file

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

draft: 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")
>
>
```

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.

```
> rm(list=ls())
> # Get base environment
> baseenv()
> # Get global environment
> globalenv()
> # Get current environment
> environment()
> # Get environment class
> class(environment())
> # Define variable in current environment
> globv <- 1
> # Get objects in current environment
> ls(environment())
> # Create new environment
> envv <- new.env()
> # Get calling environment of new environment
> parent.env(envv)
> # Assign Value to Name
> assign("var1", 3, envir=envv)
> # Create object in new environment
> envv$var2 <- 11
> # Get objects in new environment
> ls(envv)
> # Get objects in current environment
> ls(environment())
> # Environments are subset like listv
> envv$var1
> # Environments are subset like listv
> envv[["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.

Rule of Thumb

Be very careful with using `attach()`.

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

```
> search()  # Get search path for R objects
 [1] ".GlobalEnv"        "package:knitr"     "package:graphics"
 [4] "package:grDevices" "package:utils"     "package:datasets"
 [7] "package:HighFreq"  "package:rutils"    "package:dygraphs"
[10] "package:quantmod"  "package:TTR"       "package:xts"
[13] "package:zoo"       "package:stats"     "package:methods"
[16] "Autoloads"         "package:base"
> listv <- list(flowers=c("rose", "daisy", "tulip"),
+       trees=c("pine", "oak", "maple"))
> listv$trees
[1] "pine"  "oak"   "maple"
> attach(listv)
> trees
[1] "pine"  "oak"   "maple"
> search()  # Get search path for R objects
 [1] ".GlobalEnv"        "listv"             "package:knitr"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:HighFreq"  "package:rutils"
[10] "package:dygraphs"  "package:quantmod"  "package:TTR"
[13] "package:xts"       "package:zoo"       "package:stats"
[16] "package:methods"   "Autoloads"         "package:base"
> detach(listv)
> head(trees)  # "trees" is in datasets base package
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
```

Extracting Time Series from Environments

The function `mget()` accepts a vector of strings and returns a list of the corresponding objects extracted from an *environment*.

The extractor (accessor) functions from package *quantmod*: `Cl()`, `Vo()`, etc., extract columns from *OHLC* data.

A list of *xts* series can be flattened into a single *xts* series using the function `do.call()`.

The function `do.call()` executes a function call using a function name and a list of arguments.

`do.call()` passes the list elements individually, instead of passing the whole list as one argument.

The function `eapply()` is similar to `lapply()`, and applies a function to objects in an *environment*, and returns a list.

Time series can also be extracted from an *environment* by coercing it into a `list`, and then subsetting and merging it into an *xts* series using the function `do.call()`.

```
> library(rutils)  # Load package rutils
> # Define ETF symbols
> symbolv <- c("VTI", "VEU", "IEF", "VNQ")
> # Extract symbolv from rutils::etfenv
> pricev <- mget(symbolv, envir=rutils::etfenv)
> # pricev is a list of xts series
> class(pricev)
> class(pricev[[1]])
> # Extract Close prices
> pricev <- lapply(pricev, quantmod::Cl)
> # Collapse list into time series the hard way
> xts1 <- cbind(pricev[[1]], pricev[[2]], pricev[[3]], pricev[[4]])
> class(xts1)
> dim(xts1)
> # Collapse list into time series using do.call()
> pricev <- do.call(cbind, pricev)
> all.equal(xts1, pricev)
> class(pricev)
> dim(pricev)
> # Extract and cbind in single step
> pricev <- do.call(cbind, lapply(
+   mget(symbolv, envir=rutils::etfenv), quantmod::Cl))
> # Or
> # Extract and bind all data, subset by symbolv
> pricev <- lapply(symbolv, function(symbol) {
+     quantmod::Cl(get(symbol, envir=rutils::etfenv))
+ })  # end lapply
> # Same, but loop over etfenv without anonymous function
> pricev <- do.call(cbind,
+   lapply(as.list(rutils::etfenv)[symbolv], quantmod::Cl))
> # Same, but works only for OHLC series - produces error
> pricev <- do.call(cbind,
+   eapply(rutils::etfenv, quantmod::Cl)[symbolv])
```

Managing Time Series

Time series columns can be renamed, and then saved into `.csv` files.

The function `strsplit()` splits the elements of a character vector.

The package *zoo* contains functions `write.zoo()` and `read.zoo()` for writing and reading *zoo* time series from `.txt` and `.csv` files.

The function `eapply()` is similar to `lapply()`, and applies a function to objects in an *environment*, and returns a list.

The function `assign()` assigns a value to an object in a specified *environment*, by referencing it using a character string (name).

The function `save()` writes objects to compressed binary `.RData` files.

```
> # Drop ".Close" from column names
> colnames(pricev)
> do.call(rbind, strsplit(colnames(pricev), split="[.]"))[, 1]
> colnames(pricev) <- do.call(rbind, strsplit(colnames(pricev), spli
> # Or
> colnames(pricev) <- unname(sapply(colnames(pricev),
+     function(colname) strsplit(colname, split="[.]")[[1]][1]))
> tail(pricev, 3)
> # Which objects in global environment are class xts?
> unlist(eapply(globalenv(), is.xts))
> # Save xts to csv file
> write.zoo(pricev,
+   file="/Users/jerzy/Develop/lecture_slides/data/etf_series.csv",
> # Copy prices into etfenv
> etfenv$etf_list <- etf_list
> # Or
> assign("prices", pricev, envir=etfenv)
> # Save to .RData file
> save(etfenv, file="etf_data.RData")
```

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)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
> colnames(trees)
[1] "Girth"  "Height" "Volume"
> mean(Girth)

Error in eval(expr, envir, enclos):  object 'Girth' not found

> mean(trees$Girth)
[1] 13.2
> with(trees,
+      c(mean(Girth), mean(Height), mean(Volume)))
[1] 13.2 76.0 30.2
```

Sourcing R Script Files in an R Session

R commands can be saved into a file, and then executed from an interactive R session using the function `source()`.

The function `source()` executes R commands contained in a file, or in a *URL*.

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

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

```
> script_dir <- "/Users/jerzy/Develop/R/scripts"
> # Execute script file and print the commands
> source(file.path(script_dir, "script.R"),
+  echo=TRUE)
>
> ####################################
> ### Script.R file contains R script to demonstrate sourcing from s
>
> # Print information about this process
> print(paste0("print: This test script was run at: ", format(Sys.ti
> cat("cat: This test script was run at:", format(Sys.time()), "\n")
>
> # Display first 6 rows of cars data frame
> head(cars)
>
> # Define a function
> fun_c <- function(x) x+1
>
> # Read a line from console
> readline("Press Return to continue")
>
> # Plot sine function in x11 window
> x11()
> curve(expr=sin, type="l", xlim=c(-2*pi, 2*pi),
+ xlab="", ylab="", lwd=2, col="orange",
+ main="Sine function")
```

Running R Processes From the Terminal

An interactive R process can be run from the terminal, by simply typing the commands R or Rterm (provided that your *PATH* variable contains the directory of the R executable file).

The command R combined with the option -e can also execute R commands supplied on the command line.

For example the command:

```
R --vanilla -e head(cars) > out.txt
```

executes a single R command, and saves the output to a file.

The option vanilla instructs R to produce minimal output.

The manual *Introduction to R* provides more information about running R processes from the terminal:

https://cran.r-project.org/doc/manuals/R-intro.html#Invoking-R-from-the-command-line

```
# Start an interactive R process
> R

# Get help about running R process
> R --help

# Execute single R command and save output to
# vanilla option to produce minimal output
> R --vanilla -e head(cars) > out.txt
```

Executing R Scripts as Batch Processes

A *batch* process is the execution of a set of commands in a script file, without manual intervention (non-interactive mode).

There are two ways of running an R script file:

- in *interactive* mode from within an R session using the function `source()`,
- in non-interactive *batch* mode from a terminal,

R *batch* processes can be executed using the commands `R`, `R CMD BATCH`, and `Rscript`.

For example the command:

```
Rscript script.R > out.txt
```

executes a *batch* process on a script file containing a plot command and `readline()` for user input, and saves the output to a file.

The command `Rscript` can also execute R commands supplied on the command line, for example:

```
Rscript -e "head(cars)" > out.txt
```

```
> # Get help about running R scripts and batch processes
> ?BATCH
> ?Rscript
```

```
# Execute script file and save output to file
# vanilla option to produce minimal output
> cd /Users/jerzy/Develop/R/scripts
> R --vanilla < script.R > out.txt

# Execute script file and save output to file
# Slave option to produce minimal output
> R CMD BATCH --slave script.R out.txt

# Execute script file and save output to file
> Rscript script.R > out.txt

# Execute single R command from Windows
> Rscript -e "head(cars)" > out.txt

# Execute several R commands and save output to
> Rscript -e "source('script.R'); fun_c(2)" > o
```

Executing R Scripts Using Rscript

The function commandArgs() returns a vector of strings containing the arguments supplied to the R process when called from the command line.

The Rscript command is designed for fast execution of R scripts, and can also accept arguments to the R script supplied on the command line, for example:

```
Rscript --vanilla script_args.R  4 5 6
```

The Rscript command can also accept arguments supplied to R scripts on the command line, for example:

```
Rscript -e "2*as.numeric(commandArgs(TRUE))" 3

Rscript -e "sum(as.numeric(commandArgs(TRUE)))" 4 5 6
```

```
> ### Script_args.R contains R script that accepts arguments
> # Print information about this process
> cat("cat: This script was run at:", format(Sys.time()), "\n")
> # Read arguments supplied on the command line
> arg_s <- commandArgs(TRUE)
> # Print the arguments
> cat(paste0("arguments supplied on command line: ", paste(arg_s, co
> # Return sum of arguments
> sum(as.numeric(arg_s))
```

Plotting to a File From an R Script

A *batch* R process usually fails to produce a plot, because the x11 plot window closes as soon as the R process terminates.

The function readline() doesn't work in batch mode either, because it doesn't wait for user input.

But a *batch* R process can plot to a file by diverting its graphics output to a graphics file.

The functions png(), jpeg(), bmp(), and tiff() divert graphics output to graphics files (text output isn't diverted).

The function dev.off() ends the diversion.

```
> ### Plot_to_file.R
> ### R script to demonstrate plotting to file
>
> # Redirect graphics output to png file
> plot_dir <- "/Users/jerzy/Develop/data"
> png(file.path(plot_dir, "r_plot.png"))
>
> # Plot sine function
> curve(expr=sin, type="l", xlim=c(-2*pi, 2*pi),
+ xlab="", ylab="", lwd=2, col="orange",
+ main="Sine function")
>
> # Turn png output off
> dev.off()
```

```
# Execute script file and save output to file
> Rscript plot_to_file.R > out.txt
```

Interactive Plots in Batch R Processes

Interactive plots don't work in batch R processes, because the attached x11 plot window closes as soon as an R process terminates.

One way to get around this is by pausing the R process using a `while()` loop, to wait until all the x11 plot windows are closed.

The function `dev.list()` returns the number and names of active graphics devices.

```
> ### Plot_interactive.R
> ### R script to demonstrate interactive plotting
>
> # Plot sine function in x11 window
> x11()
> curve(expr=sin, type="l", xlim=c(-2*pi, 2*pi),
+ xlab="", ylab="", lwd=2, col="orange",
+ main="Sine function")
>
> # Wait until x11 window is closed
> while (!is.null(dev.list())) Sys.sleep(1)
```

```
# Execute script file and save output to file
> Rscript plot_interactive.R > out.txt
```