R Packages
FRE6871 & FRE7241, Fall 2022

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

October 30, 2022

R Packages

Types of R Packages

R can run libraries of functions called packages,

R packages can can also contain data,

Most packages need to be *loaded* into R before they can be used,

R includes a number of base packages that are already installed and loaded,

There's also a special package called the base package, which is responsible for all the basic R functionality,

datasets is a base package containing various datasets, for example EuStockMarkets,

The *base* Packages

R includes a number of packages that are pre-installed (often called *base* packages),
Some *base* packages:

- *base* - basic R functionality,
- *stats* - statistical functions and random number generation,
- *graphics* - basic graphics,
- *utils* - utility functions,
- *datasets* - popular datasets,
- *parallel* - support for parallel computation,

Very popular packages:

- *MASS* - functions and datasets for "Modern Applied Statistics with S",
- *ggplot2* - grammar of graphics plots,
- *shiny* - interactive web graphics from R,
- *slidify* - HTML5 slide shows from R,
- *devtools* - create R packages,
- *roxygen2* - document R packages,
- *Rcpp* - integrate C++ code with R,
- *RcppArmadillo* - interface to Armadillo linear algebra library,
- *forecast* - linear models and forecasting,
- *tseries* - time series analysis and computational finance,
- *zoo* - time series and ordered objects,
- *xts* - advanced time series objects,
- *quantmod* - quantitative financial modeling framework,
- *caTools* - moving window statistics for graphics and time series objects,

CRAN Package Views

CRAN view for package *AER*:
http://cran.r-project.org/web/packages/AER/

Note:

- Authors,
- Version number,
- Reference manual,
- Vignettes,
- Dependencies on other packages.

The package source code can be downloaded by clicking on the **package source** link,

---

cran.us.r-project.org/web/packages/AER/

AER: Applied Econometrics with R

Functions, data sets, examples, demos, and vignettes for the book Christian Kleiber and Achim Zeileis (2008), Applie

Version:	1.2-1
Depends:	R (≥ 2.13.0), car (≥ 2.0-1), lmtest, sandwich, survival, zoo
Imports:	stats, Formula (≥ 0.2-0)
Suggests:	boot, dynlm, effects, foreign, ineq, KernSmooth, lattice, MASS, mlogit, nlme, nnet, np, plm, pscl
Published:	2013-11-07
Author:	Christian Kleiber [aut], Achim Zeileis [aut, cre]
Maintainer:	Achim Zeileis <Achim.Zeileis at R-project.org>
License:	GPL-2
NeedsCompilation:	no
Citation:	AER citation info
Materials:	NEWS
In views:	Econometrics, Survival, TimeSeries
CRAN checks:	AER results

Downloads:

Reference manual:	AER.pdf
Vignettes:	Applied Econometrics with R: Package Vignette and Errata
	Sweave Example: Linear Regression for Economics Journals Data
Package source:	AER_1.2-1.tar.gz
MacOS X binary:	AER_1.2-1.tgz
Windows binary:	AER_1.2-1.zip
Old sources:	AER archive

Reverse dependencies:

Reverse depends: ivpack, rdd
Reverse suggests: censReg, glmx, lmtest, micEconCES, mlogit, plm, REEMtree, sandwich

CRAN Task Views

CRAN Finance Task View

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

Note:

- Maintainer,
- Topics,
- List of packages.

CRAN Task View: Empirical Finance

Maintainer: Dirk Eddelbuettel

Contact: Dirk.Eddelbuettel at R-project.org

Version: 2014-01-16

This CRAN Task View contains a list of packages useful for empirical work in Finance,

Besides these packages, a very wide variety of functions suitable for empirical work in Fi
packages on the Comprehensive R Archive Network (CRAN). Consequently, several of
Optimization, Robust, SocialSciences and TimeSeries Task Views.

Please send suggestions for additions and extensions for this task view to the task view m

Standard regression models

- A detailed overview of the available regression methodologies is provided by the
- Linear models such as ordinary least squares (OLS) can be estimated by lm() (fro
  undertaken with the standard optim() function. Many other suitable methods are
  nlme() from the nlme package.
- For the linear model, a variety of regression diagnostic tests are provided by the c
  be of interest as well.

Time series

- A detailed overview of tools for time series analysis can be found in the TimeSerie
- Classical time series functionality is provided by the arima() and KalmanLike()
- The dse and timsac packages provides a variety of more advanced estimation met
- For volatility modeling, the standard GARCH(1,1) model can be estimated with th
  models. The rugarch package can be used to model a variety of univariate GARCH
  methods for fit, forecast, simulation, inference and plotting are provided too. The r
  estimate and simulate the Beta-t-EGARCH model by Harvey. The bayesGARCH
  models, the ccgarch package can estimate (multivariate) Conditional Correlation G
  AutoSEARCH package provides automated general-to-specific model selection c
- Unit root and cointegration tests are provided by tseries, and urca. The Rmetrics p
  unit roots and more. The CADFtest package implements the Hansen unit root test
- MSBVAR provides Bayesian estimation of vector autoregressive models. The dlm
- The vars package offer estimation, diagnostics, forecasting and error decompositio
- The dyn and dynlm are suitable for dynamic (linear) regression models.
- Several packages provide wavelet analysis functionality: rwt, wavelets, waveslim,

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Installing Packages

Most packages need to be *installed* before they can be loaded and used.

Some packages like *MASS* are installed with base R (but not loaded).

Installing a package means downloading and saving its files to a local computer directory (hard disk), so they can be *loaded* by the R system.

The function `install.packages()` installs packages from the R command line.

Most widely used packages are available on the *CRAN* repository:
http://cran.r-project.org/web/packages/

Or on *R-Forge* or *GitHub*:
https://r-forge.r-project.org/
https://github.com/

Packages can also be installed in *RStudio* from the menu (go to Tools and then Install packages),

Packages residing on GitHub can be installed using the devtools packages.

```
> getOption("repos")  # get default package source
> .libPaths()  # get package save directory
> install.packages("AER")  # install "AER" from CRAN
> # install "PerformanceAnalytics" from R-Forge
> install.packages(
+   pkgs="PerformanceAnalytics",  # name
+   lib="C:/Users/Jerzy/Downloads",  # directory
+   repos="http://R-Forge.R-project.org")  # source
> # install devtools from CRAN
> install.packages("devtools")
> # load devtools
> library(devtools)
> # install package "babynamesv" from GitHub
> install_github(repo="hadley/babynamesv")
```

Installing Packages From Source

Sometimes packages aren't available in compiled form, so it's necessary to install them from their source code.

To install a package from source, the user needs to first install compilers and development tools:

For Windows install Rtools:
https://cran.r-project.org/bin/windows/Rtools/

For Mac OSX install XCode developer tools:
https://developer.apple.com/xcode/downloads/

The function `install.packages()` with argument `type="source"` installs a package from source.

The function `download.packages()` downloads the package's installation files (compressed tar format) to a local directory.

The function `install.packages()` can then be used to install the package from the downloaded files.

```
> # install package "PortfolioAnalytics" from source
> install.packages("PortfolioAnalytics",
+   type="source",
+   repos="http://r-forge.r-project.org")
> # download files for package "PortfolioAnalytics"
> download.packages(pkgs = "PortfolioAnalytics",
+   destdir = ".",  # download to cwd
+   type = "source",
+   repos="http://r-forge.r-project.org")
> # install "PortfolioAnalytics" from local tar source
> install.packages(
+   "C:/Users/Jerzy/Downloads/PortfolioAnalytics_0.9.3598.tar.gz",
+   repos=NULL, type="source")
```

Installed Packages

`defaultPackages` contains a list of packages loaded on startup by default.

The function `installed.packages()` returns a matrix of all packages installed on the system.

```
> getOption("defaultPackages")
> # matrix of installed package information
> pack_info <- installed.packages()
> dim(pack_info)
> # get all installed package names
> sort(unname(pack_info[, "Package"]))
> # get a few package names and their versions
> pack_info[sample(x=1:100, 5), c("Package", "Version")]
> # get info for package "xts"
> t(pack_info["xts", ])
```

Package Files and Directories

Package installation files are organized into multiple directories, including some of the following:

- `~/R` containing R source code files,

- `~/src` containing C++ and Fortran source code files,

- `~/data` containing datasets,

- `~/man` containing documentation files,

```
> # list directories in "PortfolioAnalytics" sub-directory
> gsub(
+   "C:/Users/Jerzy/Documents/R/win-library/3.1",
+   "~",
+   list.dirs(
+     file.path(
+       .libPaths()[1],
+       "PortfolioAnalytics")))
 [1] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [2] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [3] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [4] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [5] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [6] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [7] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [8] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
 [9] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
[10] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
[11] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
[12] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
[13] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
[14] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/1
```

Loading Packages

Most packages need to be *loaded* before they can be used in an R session.

Loading a package means attaching the package *namesvpace* to the *search path*, which allows R to call the package functions and data.

The functions `library()` and `require()` load packages, but in slightly different ways.

`library()` produces an *error* (halts execution) if the package can't be loaded.

`require()` returns TRUE if the package is loaded successfully, and FALSE otherwise.

Therefore `library()` is usually used in script files that might be sourced, while `require()` is used inside functions.

```
> # load package, produce error if can't be loaded
> library(MASS)
> # load package, return TRUE if loaded successfully
> require(MASS)
> # load quietly
> library(MASS, quietly=TRUE)
> # load without any messages
> suppressMessages(library(MASS))
> # remove package from search path
> detach(MASS)
> # install package if it can't be loaded successfully
> if (!require("xts")) install.packages("xts")
```

Referencing Package Objects

After a package is *loaded*, the package functions and data can be accessed by name.

Package objects can also be accessed without *loading* the package, by using the double-colon "::" reference operator.

For example, `TTR::VWAP()` references the function `VWAP()` from the package *TTR*.

This way users don't have to load the package *TTR* (with `library(TTR)`) to use functions from the package *TTR*.

Using the "::" operator displays the source of objects, and makes R code easier to analyze.

```
> # calculate VTI volume-weighted average price
> vwapv <- TTR::VWAP(
+   price=quantmod::Cl(rutils::etfenv$VTI),
+   volume=quantmod::Vo(rutils::etfenv$VTI), n=10)
```

Exploring Packages

The package *Ecdat* contains data sets for econometric analysis.

The data frame `Garch` contains daily currency prices.

The function `data()` loads external data or listv data sets in a package.

Some packages provide *lazy loading* of their data sets, which means they automatically load their data sets when they're needed (when they are called by some operation).

The package's data isn't loaded into R memory when the package is *loaded*, so it's not listed using `ls()`, but the package data is available without calling the function `data()`.

The function `data()` isn't required to load data sets that are set up for *lazy loading*.

```
> library()  # list all packages installed on the system
> search()  # list all loaded packages on search path
>
> # get documentation for package "Ecdat"
> packageDescription("Ecdat")  # get short description
> help(package="Ecdat")  # load help page
> library(Ecdat)  # load package "Ecdat"
> data(package="Ecdat")  # list all datasets in "Ecdat"
> ls("package:Ecdat")  # list all objects in "Ecdat"
> browseVignettes("Ecdat")  # view package vignette
> detach("package:Ecdat")  # remove Ecdat from search path
```

```
> library(Ecdat)  # load econometric data sets
> class(Garch)  # Garch is a data frame from "Ecdat"
> dim(Garch)  # daily currency prices
> head(Garch[, -2])  # col 'dm' is Deutsch Mark
> detach("package:Ecdat")  # remove Ecdat from search path
```

Package Namespaces

Package *namesvpaces*:

- Provide a mechanism for calling objects from a package,
- Hide functions and data internal to the package,
- Prevent naming conflicts between user and package names,

When a package is loaded using `library()` or `require()`, its *namesvpace* is attached to the search path.

```
> search()  # get search path for R objects
> library(MASS)  # load package "MASS"
> head(ls("package:MASS"))  # list some objects in "MASS"
> detach("package:MASS")  # remove "MASS" from search path
```

Package Namespaces and the Search Path

Packages may be loaded without their *namesvpace* being attached to the search path.

When packages are loaded, then packages they depend on are also loaded, but their *namesvpaces* aren't necessarily attached to the search path.

The function loadedNamespaces() listv all loaded *namesvpaces*, including those that aren't on the search path.

The function search() returns the current search path for R objects.

search() returns many package *namesvpaces*, but not all the loaded *namesvpaces*.

```
> loadedNamespaces()  # get names of loaded namespaces
>
> search()  # get search path for R objects
```

Not Attached Namespaces

The function `sessionInfo()` returns information about the current R session, including packages that are loaded, but *not attached* to the search path.

`sessionInfo()` listv those packages as "loaded via a *namesvpace* (and not attached)"

```
> # get session info,
> # including packages not attached to the search path
> sessionInfo()
```

Non-Visible Objects

Non-visible objects (variables or functions) are either:

- objects from *not attached namesvpaces*,
- objects *not exported* outside a package,

Objects from packages that aren't attached can be accessed using the double-colon "::" reference operator.

Objects that are *not exported* outside a package can be accessed using the triple-colon ":::" reference operator.

Colon operators automatically load the associated package.

Non-visible objects in namespaces often use the ".*" name syntax.

```
> plot.xts  # package xts isn't loaded and attached
> head(xts::plot.xts, 3)
> methods("cbind")  # get all methods for function "cbind"
> stats::cbind.ts  # cbind isn't exported from package stats
> stats:::cbind.ts  # view the non-visible function
> getAnywhere("cbind.ts")
> library(MASS)  # load package 'MASS'
> select  # code of primitive function from package 'MASS'
```

Exploring Namespaces and Non-Visible Objects

The function `getAnywhere()` displays information about R objects, including non-visible objects.

Objects referenced *within* packages have different search paths than other objects:
Their search path starts in the package *namesvpace*, then the global environment and then finally the regular search path.

This way references to objects from `within` a package are resolved to the package, and they're not masked by objects of the same name in other environments.

```
> getAnywhere("cbind.ts")
```