

R Packages

FRE6871 & FRE7241, Fall 2024

Jerzy Pawlowski jp3900@nyu.edu

NYU Tandon School of Engineering

September 4, 2024



NYU

**TANDON SCHOOL
OF ENGINEERING**

R Packages

Types of R Packages

R can run libraries of functions called packages,

R packages 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,



The screenshot shows the CRAN web page for the 'AER' package. The browser address bar displays 'cran.us.r-project.org/web/packages/AER/'. The page title is 'AER: Applied Econometrics with R'. Below the title, it states 'Functions, data sets, examples, demos, and vignettes for the book Christian Kleiber and Achim Zeileis (2008), Applie'. The page lists various details about the package, including its version (1.2-1), dependencies (R ≥ 2.13.0, car ≥ 2.0-1, lme4, sandwich, survival, zoo), imports (stats, Formula ≥ 0.2-0), and suggests (boot, dplyr, effects, foreign, ineq, KernSmooth, lattice, MASS, mlogit, nlme, rnet, np, plm, pscl). It also lists the author (Christian Kleiber [aut], Achim Zeileis [aut, cre]), maintainer (Achim Zeileis <Achim.Zeileis@R-project.org>), license (GPL-2), and needs compilation (no). The page includes links for citation, news, and views. The download section provides links for the 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), and old sources (AER archive). The reverse dependencies section lists ivpack, rdd, and the reverse suggests section lists censReg, glmx, lme4, micEconCES, mlogit, plm, REEMtree, and sandwich.

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), [lme4](#), [sandwich](#), [survival](#), [zoo](#)

Imports: stats, [Formula](#) (≥ 0.2-0)

Suggests: [boot](#), [dplyr](#), [effects](#), [foreign](#), [ineq](#), [KernSmooth](#), [lattice](#), [MASS](#), [mlogit](#), [nlme](#), [rnet](#), [np](#), [plm](#), [pscl](#)

Published: 2013-11-07

Author: Christian Kleiber [aut], Achim Zeileis [aut, cre]

Maintainer: Achim Zeileis <Achim.Zeileis@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](#), [lme4](#), [micEconCES](#), [mlogit](#), [plm](#), [REEMtree](#), [sandwich](#)

CRAN Task Views

CRAN Finance Task View

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

Note:

- Maintainer,
- Topics,
- List of packages.

← → ↻ cran.us.r-project.org



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](#)

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 Finance are available in R packages on the Comprehensive R Archive Network (CRAN). Consequently, several of the following packages are also available on the [Optimization](#), [Robust](#), [SocialSciences](#) and [TimeSeries](#) Task Views.

Please send suggestions for additions and extensions for this task view to the [task view maintainer](#).

Standard regression models

- A detailed overview of the available regression methodologies is provided by the [lm](#) package.
- Linear models such as ordinary least squares (OLS) can be estimated by `lm()` (from the [stats](#) package) or `lmfit()` (from the [lmfit](#) package). Many other suitable methods are available, e.g., `nlme()` from the [nlme](#) package.
- For the linear model, a variety of regression diagnostic tests are provided by the [car](#) package, which may be of interest as well.

Time series

- A detailed overview of tools for time series analysis can be found in the [TimeSeries](#) Task View.
- Classical time series functionality is provided by the [arima](#)() and [KalmanLike](#)() functions.
- The [dse](#) and [limsac](#) packages provide a variety of more advanced estimation methods.
- For volatility modeling, the standard GARCH(1,1) model can be estimated with the [rugarch](#) package. The [rugarch](#) package can be used to model a variety of univariate GARCH processes. The [rugarch](#) package provides methods for fit, forecast, simulation, inference and plotting as well. The [rugarch](#) package also provides the `simulate()` function to estimate and simulate the Beta-t-EGARCH model by Harvey. The [havesGARCH](#) package provides the `havesGARCH()` function to estimate (multivariate) Conditional Correlation GARCH models. The [ccgarch](#) package can estimate (multivariate) Conditional Correlation GARCH models. The [AutoSEARCH](#) package provides automated general-to-specific model selection.
- Unit root and cointegration tests are provided by [series](#) and [urca](#). The [Rmetrics](#) package provides unit roots and more. The [CADTest](#) package implements the Hansen unit root tests.
- [MSBVAR](#) provides Bayesian estimation of vector autoregressive models. The [dlm](#) package provides the `dlm` package.
- The [vars](#) package offers estimation, diagnostics, forecasting and error decomposition.
- The [dyn](#) and [dynlm](#) are suitable for dynamic (linear) regression models.
- Several packages provide wavelet analysis functionality: [rwt](#), [wavelets](#), [waveslim](#), [wavelet](#), [waveletComp](#), [waveletComp2](#), [waveletComp3](#), [waveletComp4](#), [waveletComp5](#), [waveletComp6](#), [waveletComp7](#), [waveletComp8](#), [waveletComp9](#), [waveletComp10](#), [waveletComp11](#), [waveletComp12](#), [waveletComp13](#), [waveletComp14](#), [waveletComp15](#), [waveletComp16](#), [waveletComp17](#), [waveletComp18](#), [waveletComp19](#), [waveletComp20](#), [waveletComp21](#), [waveletComp22](#), [waveletComp23](#), [waveletComp24](#), [waveletComp25](#), [waveletComp26](#), [waveletComp27](#), [waveletComp28](#), [waveletComp29](#), [waveletComp30](#), [waveletComp31](#), [waveletComp32](#), [waveletComp33](#), [waveletComp34](#), [waveletComp35](#), [waveletComp36](#), [waveletComp37](#), [waveletComp38](#), [waveletComp39](#), [waveletComp40](#), [waveletComp41](#), [waveletComp42](#), [waveletComp43](#), [waveletComp44](#), [waveletComp45](#), [waveletComp46](#), [waveletComp47](#), [waveletComp48](#), [waveletComp49](#), [waveletComp50](#), [waveletComp51](#), [waveletComp52](#), [waveletComp53](#), [waveletComp54](#), [waveletComp55](#), [waveletComp56](#), [waveletComp57](#), [waveletComp58](#), [waveletComp59](#), [waveletComp60](#), [waveletComp61](#), [waveletComp62](#), [waveletComp63](#), [waveletComp64](#), [waveletComp65](#), [waveletComp66](#), [waveletComp67](#), [waveletComp68](#), [waveletComp69](#), [waveletComp70](#), [waveletComp71](#), [waveletComp72](#), [waveletComp73](#), [waveletComp74](#), [waveletComp75](#), [waveletComp76](#), [waveletComp77](#), [waveletComp78](#), [waveletComp79](#), [waveletComp80](#), [waveletComp81](#), [waveletComp82](#), [waveletComp83](#), [waveletComp84](#), [waveletComp85](#), [waveletComp86](#), [waveletComp87](#), [waveletComp88](#), [waveletComp89](#), [waveletComp90](#), [waveletComp91](#), [waveletComp92](#), [waveletComp93](#), [waveletComp94](#), [waveletComp95](#), [waveletComp96](#), [waveletComp97](#), [waveletComp98](#), [waveletComp99](#), [waveletComp100](#), [waveletComp101](#), [waveletComp102](#), [waveletComp103](#), [waveletComp104](#), [waveletComp105](#), [waveletComp106](#), [waveletComp107](#), [waveletComp108](#), [waveletComp109](#), [waveletComp110](#), [waveletComp111](#), [waveletComp112](#), [waveletComp113](#), [waveletComp114](#), [waveletComp115](#), [waveletComp116](#), [waveletComp117](#), [waveletComp118](#), [waveletComp119](#), [waveletComp120](#), [waveletComp121](#), [waveletComp122](#), [waveletComp123](#), [waveletComp124](#), [waveletComp125](#), [waveletComp126](#), [waveletComp127](#), [waveletComp128](#), [waveletComp129](#), [waveletComp130](#), [waveletComp131](#), [waveletComp132](#), [waveletComp133](#), [waveletComp134](#), [waveletComp135](#), [waveletComp136](#), [waveletComp137](#), [waveletComp138](#), [waveletComp139](#), [waveletComp140](#), [waveletComp141](#), [waveletComp142](#), [waveletComp143](#), [waveletComp144](#), [waveletComp145](#), [waveletComp146](#), [waveletComp147](#), [waveletComp148](#), [waveletComp149](#), [waveletComp150](#), [waveletComp151](#), [waveletComp152](#), [waveletComp153](#), [waveletComp154](#), [waveletComp155](#), [waveletComp156](#), [waveletComp157](#), [waveletComp158](#), [waveletComp159](#), [waveletComp160](#), [waveletComp161](#), [waveletComp162](#), [waveletComp163](#), [waveletComp164](#), [waveletComp165](#), [waveletComp166](#), [waveletComp167](#), [waveletComp168](#), [waveletComp169](#), [waveletComp170](#), [waveletComp171](#), [waveletComp172](#), [waveletComp173](#), [waveletComp174](#), [waveletComp175](#), [waveletComp176](#), [waveletComp177](#), [waveletComp178](#), [waveletComp179](#), [waveletComp180](#), [waveletComp181](#), [waveletComp182](#), [waveletComp183](#), [waveletComp184](#), [waveletComp185](#), [waveletComp186](#), [waveletComp187](#), [waveletComp188](#), [waveletComp189](#), [waveletComp190](#), [waveletComp191](#), [waveletComp192](#), [waveletComp193](#), [waveletComp194](#), [waveletComp195](#), [waveletComp196](#), [waveletComp197](#), [waveletComp198](#), [waveletComp199](#), [waveletComp200](#), [waveletComp201](#), [waveletComp202](#), [waveletComp203](#), [waveletComp204](#), [waveletComp205](#), [waveletComp206](#), [waveletComp207](#), [waveletComp208](#), [waveletComp209](#), [waveletComp210](#), [waveletComp211](#), [waveletComp212](#), [waveletComp213](#), [waveletComp214](#), [waveletComp215](#), [waveletComp216](#), [waveletComp217](#), [waveletComp218](#), [waveletComp219](#), [waveletComp220](#), [waveletComp221](#), [waveletComp222](#), [waveletComp223](#), [waveletComp224](#), [waveletComp225](#), [waveletComp226](#), [waveletComp227](#), [waveletComp228](#), [waveletComp229](#), [waveletComp230](#), [waveletComp231](#), [waveletComp232](#), [waveletComp233](#), [waveletComp234](#), [waveletComp235](#), [waveletComp236](#), [waveletComp237](#), [waveletComp238](#), [waveletComp239](#), [waveletComp240](#), [waveletComp241](#), [waveletComp242](#), [waveletComp243](#), [waveletComp244](#), [waveletComp245](#), [waveletComp246](#), [waveletComp247](#), [waveletComp248](#), [waveletComp249](#), [waveletComp250](#), [waveletComp251](#), [waveletComp252](#), [waveletComp253](#), [waveletComp254](#), [waveletComp255](#), [waveletComp256](#), [waveletComp257](#), [waveletComp258](#), [waveletComp259](#), [waveletComp260](#), [waveletComp261](#), [waveletComp262](#), [waveletComp263](#), [waveletComp264](#), [waveletComp265](#), [waveletComp266](#), [waveletComp267](#), [waveletComp268](#), [waveletComp269](#), [waveletComp270](#), [waveletComp271](#), [waveletComp272](#), [waveletComp273](#), [waveletComp274](#), [waveletComp275](#), [waveletComp276](#), [waveletComp277](#), [waveletComp278](#), [waveletComp279](#), [waveletComp280](#), [waveletComp281](#), [waveletComp282](#), [waveletComp283](#), [waveletComp284](#), [waveletComp285](#), [waveletComp286](#), [waveletComp287](#), [waveletComp288](#), [waveletComp289](#), [waveletComp290](#), [waveletComp291](#), [waveletComp292](#), [waveletComp293](#), [waveletComp294](#), [waveletComp295](#), [waveletComp296](#), [waveletComp297](#), [waveletComp298](#), [waveletComp299](#), [waveletComp300](#), [waveletComp301](#), [waveletComp302](#), [waveletComp303](#), [waveletComp304](#), [waveletComp305](#), [waveletComp306](#), [waveletComp307](#), [waveletComp308](#), [waveletComp309](#), [waveletComp310](#), [waveletComp311](#), [waveletComp312](#), [waveletComp313](#), [waveletComp314](#), [waveletComp315](#), [waveletComp316](#), [waveletComp317](#), [waveletComp318](#), [waveletComp319](#), [waveletComp320](#), [waveletComp321](#), [waveletComp322](#), [waveletComp323](#), [waveletComp324](#), [waveletComp325](#), [waveletComp326](#), [waveletComp327](#), [waveletComp328](#), [waveletComp329](#), [waveletComp330](#), [waveletComp331](#), [waveletComp332](#), [waveletComp333](#), [waveletComp334](#), [waveletComp335](#), [waveletComp336](#), [waveletComp337](#), [waveletComp338](#), [waveletComp339](#), [waveletComp340](#), [waveletComp341](#), [waveletComp342](#), [waveletComp343](#), [waveletComp344](#), [waveletComp345](#), [waveletComp346](#), [waveletComp347](#), [waveletComp348](#), [waveletComp349](#), [waveletComp350](#), [waveletComp351](#), [waveletComp352](#), [waveletComp353](#), [waveletComp354](#), [waveletComp355](#), [waveletComp356](#), [waveletComp357](#), [waveletComp358](#), [waveletComp359](#), [waveletComp360](#), [waveletComp361](#), [waveletComp362](#), [waveletComp363](#), [waveletComp364](#), [waveletComp365](#), [waveletComp366](#), [waveletComp367](#), [waveletComp368](#), [waveletComp369](#), [waveletComp370](#), [waveletComp371](#), [waveletComp372](#), [waveletComp373](#), [waveletComp374](#), [waveletComp375](#), [waveletComp376](#), [waveletComp377](#), [waveletComp378](#), [waveletComp379](#), [waveletComp380](#), [waveletComp381](#), [waveletComp382](#), [waveletComp383](#), [waveletComp384](#), [waveletComp385](#), [waveletComp386](#), [waveletComp387](#), [waveletComp388](#), [waveletComp389](#), [waveletComp390](#), [waveletComp391](#), [waveletComp392](#), [waveletComp393](#), [waveletComp394](#), [waveletComp395](#), [waveletComp396](#), [waveletComp397](#), [waveletComp398](#), [waveletComp399](#), [waveletComp400](#), [waveletComp401](#), [waveletComp402](#), [waveletComp403](#), [waveletComp404](#), [waveletComp405](#), [waveletComp406](#), [waveletComp407](#), [waveletComp408](#), [waveletComp409](#), [waveletComp410](#), [waveletComp411](#), [waveletComp412](#), [waveletComp413](#), [waveletComp414](#), [waveletComp415](#), [waveletComp416](#), [waveletComp417](#), [waveletComp418](#), [waveletComp419](#), [waveletComp420](#), [waveletComp421](#), [waveletComp422](#), [waveletComp423](#), [waveletComp424](#), [waveletComp425](#), [waveletComp426](#), [waveletComp427](#), [waveletComp428](#), [waveletComp429](#), [waveletComp430](#), [waveletComp431](#), [waveletComp432](#), [waveletComp433](#), [waveletComp434](#), [waveletComp435](#), [waveletComp436](#), [waveletComp437](#), [waveletComp438](#), [waveletComp439](#), [waveletComp440](#), [waveletComp441](#), [waveletComp442](#), [waveletComp443](#), [waveletComp444](#), [waveletComp445](#), [waveletComp446](#), [waveletComp447](#), [waveletComp448](#), [waveletComp449](#), [waveletComp450](#), [waveletComp451](#), [waveletComp452](#), [waveletComp453](#), [waveletComp454](#), [waveletComp455](#), [waveletComp456](#), [waveletComp457](#), [waveletComp458](#), [waveletComp459](#), [waveletComp460](#), [waveletComp461](#), [waveletComp462](#), [waveletComp463](#), [waveletComp464](#), [waveletComp465](#), [waveletComp466](#), [waveletComp467](#), [waveletComp468](#), [waveletComp469](#), [waveletComp470](#), [waveletComp471](#), [waveletComp472](#), [waveletComp473](#), [waveletComp474](#), [waveletComp475](#), [waveletComp476](#), [waveletComp477](#), [waveletComp478](#), [waveletComp479](#), [waveletComp480](#), [waveletComp481](#), [waveletComp482](#), [waveletComp483](#), [waveletComp484](#), [waveletComp485](#), [waveletComp486](#), [waveletComp487](#), [waveletComp488](#), [waveletComp489](#), [waveletComp490](#), [waveletComp491](#), [waveletComp492](#), [waveletComp493](#), [waveletComp494](#), [waveletComp495](#), [waveletComp496](#), [waveletComp497](#), [waveletComp498](#), [waveletComp499](#), [waveletComp500](#), [waveletComp501](#), [waveletComp502](#), [waveletComp503](#), [waveletComp504](#), [waveletComp505](#), [waveletComp506](#), [waveletComp507](#), [waveletComp508](#), [waveletComp509](#), [waveletComp510](#), [waveletComp511](#), [waveletComp512](#), [waveletComp513](#), [waveletComp514](#), [waveletComp515](#), [waveletComp516](#), [waveletComp517](#), [waveletComp518](#), [waveletComp519](#), [waveletComp520](#), [waveletComp521](#), [waveletComp522](#), [waveletComp523](#), [waveletComp524](#), [waveletComp525](#), [waveletComp526](#), [waveletComp527](#), [waveletComp528](#), [waveletComp529](#), [waveletComp530](#), [waveletComp531](#), [waveletComp532](#), [waveletComp533](#), [waveletComp534](#), [waveletComp535](#), [waveletComp536](#), [waveletComp537](#), [waveletComp538](#), [waveletComp539](#), [waveletComp540](#), [waveletComp541](#), [waveletComp542](#), [waveletComp543](#), [waveletComp544](#), [waveletComp545](#), [waveletComp546](#), [waveletComp547](#), [waveletComp548](#), [waveletComp549](#), [waveletComp550](#), [waveletComp551](#), [waveletComp552](#), [waveletComp553](#), [waveletComp554](#), [waveletComp555](#), [waveletComp556](#), [waveletComp557](#), [waveletComp558](#), [waveletComp559](#), [waveletComp560](#), [waveletComp561](#), [waveletComp562](#), [waveletComp563](#), [waveletComp564](#), [waveletComp565](#), [waveletComp566](#), [waveletComp567](#), [waveletComp568](#), [waveletComp569](#), [waveletComp570](#), [waveletComp571](#), [waveletComp572](#), [waveletComp573](#), [waveletComp574](#), [waveletComp575](#), [waveletComp576](#), [waveletComp577](#), [waveletComp578](#), [waveletComp579](#), [waveletComp580](#), [waveletComp581](#), [waveletComp582](#), [waveletComp583](#), [waveletComp584](#), [waveletComp585](#), [waveletComp586](#), [waveletComp587](#), [waveletComp588](#), [waveletComp589](#), [waveletComp590](#), [waveletComp591](#), [waveletComp592](#), [waveletComp593](#), [waveletComp594](#), [waveletComp595](#), [waveletComp596](#), [waveletComp597](#), [waveletComp598](#), [waveletComp599](#), [waveletComp600](#), [waveletComp601](#), [waveletComp602](#), [waveletComp603](#), [waveletComp604](#), [waveletComp605](#), [waveletComp606](#), [waveletComp607](#), [waveletComp608](#), [waveletComp609](#), [waveletComp610](#), [waveletComp611](#), [waveletComp612](#), [waveletComp613](#), [waveletComp614](#), [waveletComp615](#), [waveletComp616](#), [waveletComp617](#), [waveletComp618](#), [waveletComp619](#), [waveletComp620](#), [waveletComp621](#), [waveletComp622](#), [waveletComp623](#), [waveletComp624](#), [waveletComp625](#), [waveletComp626](#), [waveletComp627](#), [waveletComp628](#), [waveletComp629](#), [waveletComp630](#), [waveletComp631](#), [waveletComp632](#), [waveletComp633](#), [waveletComp634](#), [waveletComp635](#), [waveletComp636](#), [waveletComp637](#), [waveletComp638](#), [waveletComp639](#), [waveletComp640](#), [waveletComp641](#), [waveletComp642](#), [waveletComp643](#), [waveletComp644](#), [waveletComp645](#), [waveletComp646](#), [waveletComp647](#), [waveletComp648](#), [waveletComp649](#), [waveletComp650](#), [waveletComp651](#), [waveletComp652](#), [waveletComp653](#), [waveletComp654](#), [waveletComp655](#), [waveletComp656](#), [waveletComp657](#), [waveletComp658](#), [waveletComp659](#), [waveletComp660](#), [waveletComp661](#), [waveletComp662](#), [waveletComp663](#), [waveletComp664](#), [waveletComp665](#), [waveletComp666](#), [waveletComp667](#), [waveletComp668](#), [waveletComp669](#), [waveletComp670](#), [waveletComp671](#), [waveletComp672](#), [waveletComp673](#), [waveletComp674](#), [waveletComp675](#), [waveletComp676](#), [waveletComp677](#), [waveletComp678](#), [waveletComp679](#), [waveletComp680](#), [waveletComp681](#), [waveletComp682](#), [waveletComp683](#), [waveletComp684](#), [waveletComp685](#), [waveletComp686](#), [waveletComp687](#), [waveletComp688](#), [waveletComp689](#), [waveletComp690](#), [waveletComp691](#), [waveletComp692](#), [waveletComp693](#), [waveletComp694](#), [waveletComp695](#), [waveletComp696](#), [waveletComp697](#), [waveletComp698](#), [waveletComp699](#), [waveletComp700](#), [waveletComp701](#), [waveletComp702](#), [waveletComp703](#), [waveletComp704](#), [waveletComp705](#), [waveletComp706](#), [waveletComp707](#), [waveletComp708](#), [waveletComp709](#), [waveletComp710](#), [waveletComp711](#), [waveletComp712](#), [waveletComp713](#), [waveletComp714](#), [waveletComp715](#), [waveletComp716](#), [waveletComp717](#), [waveletComp718](#), [waveletComp719](#), [waveletComp720](#), [waveletComp721](#), [waveletComp722](#), [waveletComp723](#), [waveletComp724](#), [waveletComp725](#), [waveletComp726](#), [waveletComp727](#), [waveletComp728](#), [waveletComp729](#), [waveletComp730](#), [waveletComp731](#), [waveletComp732](#), [waveletComp733](#), [waveletComp734](#), [waveletComp735](#), [waveletComp736](#), [waveletComp737](#), [waveletComp738](#), [waveletComp739](#), [waveletComp740](#), [waveletComp741](#), [waveletComp742](#), [waveletComp743](#), [waveletComp744](#), [waveletComp745](#), [waveletComp746](#), [waveletComp747](#), [waveletComp748](#), [waveletComp749](#), [waveletComp750](#), [waveletComp751](#), [waveletComp752](#), [waveletComp753](#), [waveletComp754](#), [waveletComp755](#), [waveletComp756](#), [waveletComp757](#), [waveletComp758](#), [waveletComp759](#), [waveletComp760](#), [waveletComp761](#), [waveletComp762](#), [waveletComp763](#), [waveletComp764](#), [waveletComp765](#), [waveletComp766](#), [waveletComp767](#), [waveletComp768](#), [waveletComp769](#), [waveletComp770](#), [waveletComp771](#), [waveletComp772](#), [waveletComp773](#), [waveletComp774](#)

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 "babynames" from GitHub
> install_github(repo="hadley/babynames")
```

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
> packinfo <- installed.packages()
> dim(packinfo)
> # get all installed package names
> sort(unname(packinfo[, "Package"]))
> # get a few package names and their versions
> packinfo[sample(x=1:100, 5), c("Package", "Version")]
> # get info for package "xts"
> t(packinfo["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"))
+   character(0))
```

Loading Packages

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

Loading a package means attaching the package *namespace* 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 *namespaces*:

- 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 *namespace* 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 *namespace* being attached to the search path.

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

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

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

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

```
> 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()` lists those packages as "loaded via a *namespace* (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 namespaces*,
- 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 *namespace*, 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")
```

Package *tseries* for Time Series Analysis

The package *tseries* contains functions for time series analysis and computational finance, such as:

- downloading historical data,
- plotting time series,
- calculating risk and performance measures,
- statistical *hypothesis testing*,
- calibrating models to time series,
- portfolio optimization,

Package *tseries* accepts time series of class "ts" and "zoo", and also has its own class "irts" for irregular spaced time-series objects.

The package *zoo* is designed for managing *time series* and ordered data objects.

The function `zoo::coredata()` extracts the underlying numeric data from a complex data object.

```
> # Get documentation for package tseries
> packageDescription("tseries") # Get short description
>
> help(package="tseries") # Load help page
>
> library(tseries) # Load package tseries
>
> data(package="tseries") # List all datasets in "tseries"
>
> ls("package:tseries") # List all objects in "tseries"
>
> detach("package:tseries") # Remove tseries from search path
```

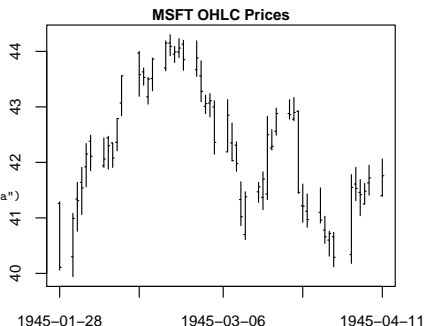
Plotting *OHLC* Time Series Using Package *tseries*

The package *tseries* contains functions for plotting time series:

- `seqplot.ts()` for plotting two time series in same panel.
- `plotOHLC()` for plotting *OHLC* time series.

The function `plotOHLC()` from package *tseries* plots *OHLC* time series.

```
> load(file="/Users/jerzy/Develop/lecture_slides/data/zoo_data.RData")
> # Get start and end dates
> datev <- time(stxts_adj)
> endd <- datev[NROW(datev)]
> startd <- round((4*endd + datev[1])/5)
> # Plot using plotOHLC
> plotOHLC(window(stxts_adj,
+               start=startd,
+               end=endd)[, 1:4],
+         xlab="", ylab="")
> title(main="MSFT OHLC Prices")
```



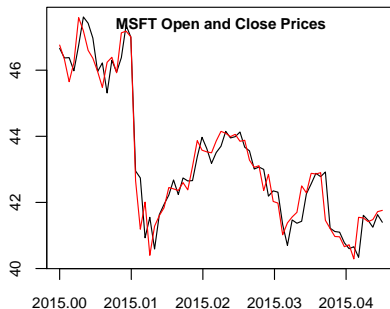
Plotting Two Time Series Using *tseries*

The function `seqplot.ts()` from package *tseries* plots two time series in same panel.

A *ts* time series can be created from a *zoo* time series using the function `ts()`, after extracting the data and date attributes from the *zoo* time series.

The function `decimal_date()` from package *lubridate* converts POSIXct objects into numeric year-fraction dates.

```
> library(lubridate) # Load lubridate
> # Get start and end dates of msft
> startd <- lubridate::decimal_date(start(msft))
> endd <- lubridate::decimal_date(end(msft))
> # Calculate frequency of msft
> tstep <- NROW(msft)/(endd-startd)
> # Extract data from msft
> datav <- zoo::coredata(
+   window(msft, start=as.Date("2015-01-01"),
+     end=end(msft)))
> # Create ts object using ts()
> stxts <- ts(data=datav,
+   start=lubridate::decimal_date(as.Date("2015-01-01")),
+   frequency=tstep)
> seqplot.ts(x=stxts[, 1], y=stxts[, 4], xlab="", ylab="")
> title(main="MSFT Open and Close Prices", line=-1)
```



The function `zoo::coredata()` extracts the underlying numeric data from a complex data object.

Risk and Performance Estimation Using *tseries*

The package *tseries* contains functions for calculating risk and performance:

- `maxdrawdown()` for calculating the maximum drawdown,
- `sharpe()` for calculating the *Sharpe* ratio (defined as the excess return divided by the standard deviation),
- `sterling()` for calculating the *Sterling* ratio (defined as the return divided by the maximum drawdown),

```
> library(tseries) # Load package tseries
> # Calculate maximum drawdown
> maxdrawdown(msft_adj[, "AdjClose"])
> max_drawd <- maxdrawdown(msft_adj[, "AdjClose"])
> zoo::index(msft_adj)[max_drawd$from]
> zoo::index(msft_adj)[max_drawd$to]
> # Calculate Sharpe ratio
> sharpe(msft_adj[, "AdjClose"])
> # Calculate Sterling ratio
> sterling(as.numeric(msft_adj[, "AdjClose"]))
```

Hypothesis Testing Using *tseries*

The package *tseries* contains functions for testing statistical hypothesis on time series:

- `jarque.bera.test()` *Jarque-Bera* test for normality of distribution of returns,
- `adf.test()` *Augmented Dickey-Fuller* test for existence of unit roots,
- `pp.test()` *Phillips-Perron* test for existence of unit roots,
- `kpss.test()` *KPSS* test for stationarity,
- `po.test()` *Phillips-Ouliaris* test for cointegration,
- `bds.test()` *BDS* test for randomness,

```
> msft <- suppressWarnings( # Load MSFT data
+   get.hist.quote(instrument="MSFT",
+                 start=Sys.Date()-365,
+                 end=Sys.Date(),
+                 origin="1970-01-01")
+ ) # end suppressWarnings
> class(msft)
> dim(msft)
> tail(msft, 4)
>
> # Calculate Sharpe ratio
> sharpe(msft[, "Close"], r=0.01)
> # Add title
> plot(msft[, "Close"], xlab="", ylab="")
> title(main="MSFT Close Prices", line=-1)
```

Calibrating Time Series Models Using *tseries*

The package *tseries* contains functions for calibrating models to time series:

- `garch()` for calibrating *GARCH* volatility models,
- `arma()` for calibrating *ARMA* models,

```
> library(tseries) # Load package tseries
> msft <- suppressWarnings( # Load MSFT data
+   get.hist.quote(instrument="MSFT",
+     start=Sys.Date()-365,
+     end=Sys.Date(),
+     origin="1970-01-01")
+ ) # end suppressWarnings
> class(msft)
> dim(msft)
> tail(msft, 4)
>
> # Calculate Sharpe ratio
> sharpe(msft[, "Close"], r=0.01)
> # Add title
> plot(msft[, "Close"], xlab="", ylab="")
> title(main="MSFT Close Prices", line=-1)
```

Portfolio Optimization Using *tseries*

The package *tseries* contains functions for miscellaneous functions:

`portfolio.optim()` for calculating mean-variance efficient portfolios.

```
> library(tseries) # Load package tseries
> msft <- suppressWarnings( # Load MSFT data
+   get.hist.quote(instrument="MSFT",
+     start=Sys.Date()-365,
+     end=Sys.Date(),
+     origin="1970-01-01")
+ ) # end suppressWarnings
> class(msft)
> dim(msft)
> tail(msft, 4)
>
> # Calculate Sharpe ratio
> sharpe(msft[, "Close"], r=0.01)
> # Add title
> plot(msft[, "Close"], xlab="", ylab="")
> title(main="MSFT Close Prices", line=-1)
```


Package *quantmod* for Quantitative Financial Modeling

The package *quantmod* is designed for downloading, manipulating, and visualizing *OHLC* time series data.

quantmod operates on time series of class "xts", and provides many useful functions for building quantitative financial models:

- `getSymbols()` for downloading data from external sources (*Yahoo*, *FRED*, etc.),
- `getFinancials()` for downloading financial statements,
- `adjustOHLC()` for adjusting *OHLC* data,
- `Op()`, `Cl()`, `Vo()`, etc. for extracting *OHLC* data columns,
- `periodReturn()`, `dailyReturn()`, etc. for calculating periodic returns,
- `chartSeries()` for candlestick plots of *OHLC* data,
- `addBBands()`, `addMA()`, `addVo()`, etc. for adding technical indicators (Moving Averages, Bollinger Bands) and volume data to a plot,

```
> # Load package quantmod
> library(quantmod)
> # Get documentation for package quantmod
> # Get short description
> packageDescription("quantmod")
> # Load help page
> help(package="quantmod")
> # List all datasets in "quantmod"
> data(package="quantmod")
> # List all objects in "quantmod"
> ls("package:quantmod")
> # Remove quantmod from search path
> detach("package:quantmod")
```

Plotting *OHLC* Time Series Using `chartSeries()`

The function `chartSeries()` from package *quantmod* can produce a variety of plots for *OHLC* time series, including candlestick plots, bar plots, and line plots.

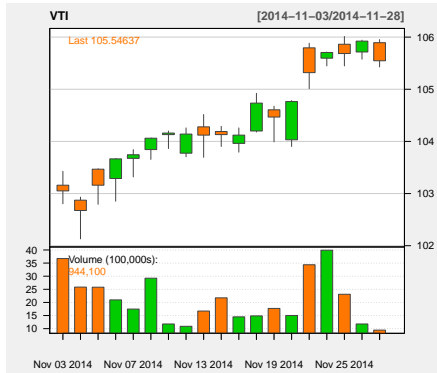
The argument `"type"` determines the type of plot (candlesticks, bars, or lines).

The argument `"theme"` accepts a `"chart.theme"` object, containing parameters that determine the plot appearance (colors, size, fonts).

`chartSeries()` automatically plots the volume data in a separate panel.

Candlestick plots are designed to visualize *OHLC* time series.

```
> # Plot OHLC candlechart with volume
> chartSeries(etfenv$VTI["2014-11"],
+   name="VTI",
+   theme=chartTheme("white"))
> # Plot OHLC bar chart with volume
> chartSeries(etfenv$VTI["2014-11"],
+   type="bars",
+   name="VTI",
+   theme=chartTheme("white"))
```



Each *candlestick* displays one period of data, and consists of a box representing the *Open* and *Close* prices, and a vertical line representing the *High* and *Low* prices.

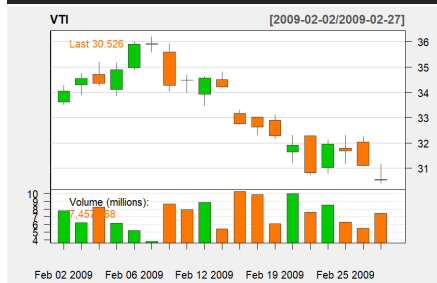
The color of the box signifies whether the *Close* price was higher or lower than the *Open*,

Redrawing Plots Using `reChart()`

The function `reChart()` redraws plots using the same data set, but using additional parameters that control the plot appearance.

The argument "subset" allows subsetting the data to a smaller range of dates.

```
> # Plot OHLC candlechart with volume
> chartSeries(etfenv$VTI["2008-11/2009-04"], name="VTI")
> # Redraw plot only for Feb-2009, with white theme
> reChart(subset="2009-02", theme=chartTheme("white"))
```



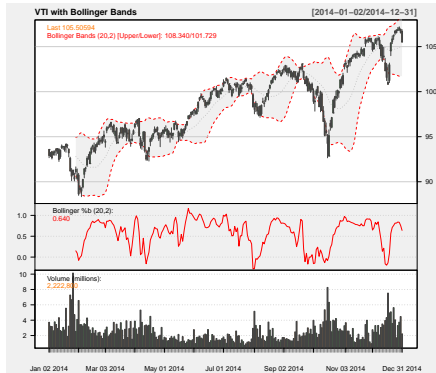
Plotting Technical Indicators Using `chartSeries()`

The argument "TA" allows adding technical indicators to the plot.

The technical indicators are functions provided by the package *TTR*.

The function `newTA()` allows defining new technical indicators.

```
> # Candlechart with Bollinger Bands
> chartSeries(etfenv$VTI["2014"],
+   TA="addBBands(): addBBands(draw='percent'): addVo()",
+   name="VTI with Bollinger Bands",
+   theme=chartTheme("white"))
> # Candlechart with two Moving Averages
> chartSeries(etfenv$VTI["2014"],
+   TA="addVo(): addEMA(10): addEMA(30)",
+   name="VTI with Moving Averages",
+   theme=chartTheme("white"))
> # Candlechart with Commodity Channel Index
> chartSeries(etfenv$VTI["2014"],
+   TA="addVo(): addBBands(): addCCI()",
+   name="VTI with Technical Indicators",
+   theme=chartTheme("white"))
```



Adding Indicators and Lines Using addTA()

The function `addTA()` adds indicators and lines to plots, and allows plotting lines representing a single vector of data.

The `addTA()` function argument `"on"` determines on which plot panel (subplot) the indicator is drawn.

`"on=NA"` is the default, and draws in a new plot panel below the existing plot.

`"on=1"` draws in the foreground of the main plot panel, and `"on=-1"` draws in the background.

```
> ohlc <- rutils::etfenv$VTI["2009-02/2009-03"]
> VTI_close <- quantmod::Cl(ohlc)
> VTI_vol <- quantmod::Vo(ohlc)
> # Calculate volume-weighted average price
> vwapv <- TTR::VWAP(price=VTI_close, volume=VTI_vol, n=10)
> # Plot OHLC candlechart with volume
> chartSeries(ohlc, name="VTI plus VWAP", theme=chartTheme("white"))
> # Add VWAP to main plot
> addTA(ta=vwapv, on=1, col='red')
> # Add price minus VWAP in extra panel
> addTA(ta=(VTI_close-vwapv), col='red')
```



The function `VWAP()` from package *TTR* calculates the Volume Weighted Average Price as the average of past prices multiplied by their trading volumes, divided by the total volume.

The argument `"n"` represents the number of look-back intervals used for averaging,

Shading Plots Using addTA()

addTA() accepts Boolean vectors for shading of plots.

The function addLines() draws vertical or horizontal lines in plots.

```
> # Plot OHLC candlechart with volume
> chartSeries(ohlc, name="VTI plus VWAP shaded",
+             theme=chartTheme("white"))
> # Add VWAP to main plot
> addTA(ta=vwapv, on=1, col='red')
> # Add price minus VWAP in extra panel
> addTA(ta=(VTI_close-vwapv), col='red')
> # Add background shading of areas
> addTA((VTI_close-vwapv) > 0, on=-1,
+ col="lightgreen", border="lightgreen")
> addTA((VTI_adj-vwapv) < 0, on=-1,
+ col="lightgrey", border="lightgrey")
> # Add vertical and horizontal lines at vwapv minimum
> addLines(v=which.min(vwapv), col='red')
> addLines(h=min(vwapv), col='red')
```

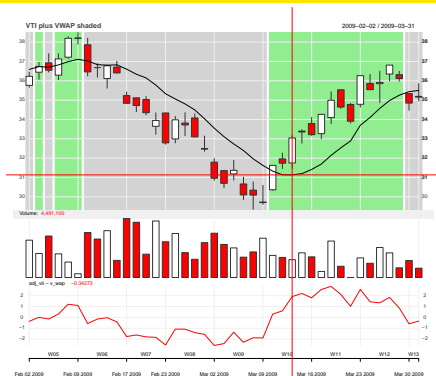


Plotting Time Series Using `chart.Series()`

The function `chart.Series()` from package *quantmod* is an improved version of `chartSeries()`, with better aesthetics.

`chart.Series()` plots are compatible with the base graphics package in R, so that standard plotting functions can be used in conjunction with `chart.Series()`.

```
> # OHLC candlechart VWAP in main plot,
> chart.Series(x=ohlcv, # Volume in extra panel
+             TA="add_Vo(); add_TA(vwapv, on=1)",
+             name="VTI plus VWAP shaded")
> # Add price minus VWAP in extra panel
> add_TA(VTI_adj-vwapv, col='red')
> # Add background shading of areas
> add_TA((VTI_adj-vwapv) > 0, on=-1,
+ col="lightgreen", border="lightgreen")
> add_TA((VTI_adj-vwapv) < 0, on=-1,
+ col="lightgrey", border="lightgrey")
> # Add vertical and horizontal lines
> abline(v=which.min(vwapv), col='red')
> abline(h=min(vwapv), col='red')
```



`chart.Series()` also has its own functions for adding indicators: `add_TA()`, `add_BBands()`, etc.

Note that functions associated with `chart.Series()` contain an underscore in their name,

Plot and Theme Objects of `chart.Series()`

The function `chart.Series()` creates a *plot object* and returns it *invisibly*.

A *plot object* is an environment of class *replot*, containing parameters specifying a plot.

A plot can be rendered by calling, plotting, or printing the *plot object*.

A plot *theme object* is a list containing parameters that determine the plot appearance (colors, size, fonts).

The function `chart.theme()` returns the *theme object*.

`chart.Series()` plots can be modified by modifying *plot objects* or *theme objects*.

Plot and theme objects can be modified directly, or by using accessor and setter functions.

The parameter "plot=FALSE" suppresses plotting and allows modifying *plot objects*.

```
> # Extract plot object
> chobj <- chart.Series(x=ohlc, plot=FALSE)
> class(chobj)
> ls(chobj)
> class(chobj$get_ylim)
> class(chobj$set_ylim)
> # ls(chobj$Env)
> class(chobj$Env$actions)
> plotheme <- chart_theme()
> class(plotheme)
> ls(plotheme)
```


Customizing chart.Series() Plots

chart.Series() plots can be customized by modifying the plot and theme objects.

Plot and theme objects can be modified directly, or by using accessor and setter functions.

A plot is rendered by calling, plotting, or printing the plot object.

The parameter "plot=FALSE" suppresses plotting and allows modifying *plot objects*.

```
> ohlc <- rutils::etfenv$VTI["2010-04/2010-05"]
> # Extract, modify theme, format tick marks "%b %d"
> plotheme <- chart_theme()
> plotheme$format.labels <- "%b %d"
> # Create plot object
> chobj <- chart.Series(x=ohlc, theme=plotheme, plot=FALSE)
> # Extract ylim using accessor function
> ylim <- chobj$get_ylim()
> ylim[[2]] <- structure(range(quantmod::Cl(ohlc)) + c(-1, 1),
+   fixed=TRUE)
> # Modify plot object to reduce y-axis range
> chobj$set_ylim(ylim) # use setter function
> # Render the plot
> plot(chobj)
```



Plotting chart.Series() in Multiple Panels

chart.Series() plots are compatible with the base graphics package, allowing easy plotting in multiple panels.

The parameter "plot=FALSE" suppresses plotting and allows adding extra plot elements.

```
> # Calculate VTI and XLF volume-weighted average price
> vwapv <- TTR::VWAP(price=quantmod::Cl(rutils::etfenv$VTI),
+   volume=quantmod::Vo(rutils::etfenv$VTI), n=10)
> XLF_vwap <- TTR::VWAP(price=quantmod::Cl(rutils::etfenv$XLF),
+   volume=quantmod::Vo(rutils::etfenv$XLF), n=10)
> # Open graphics device, and define
> # Plot area with two horizontal panels
> x11(); par(mfrow=c(2, 1))
> chobj <- chart.Series( # Plot in top panel
+   x=etfenv$VTI["2009-02/2009-04"],
+   name="VTI", plot=FALSE)
> add_TA(vwapv["2009-02/2009-04"], lwd=2, on=1, col='blue')
> # Plot in bottom panel
> chobj <- chart.Series(x=etfenv$XLF["2009-02/2009-04"],
+   name="XLF", plot=FALSE)
> add_TA(XLF_vwap["2009-02/2009-04"], lwd=2, on=1, col='blue')
```

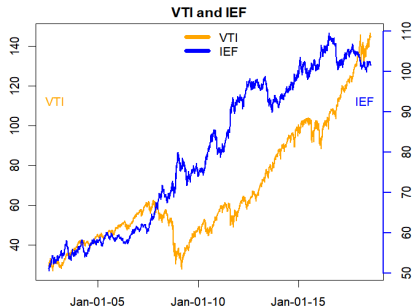


zoo Plots With Two "y" Axes

The function `plot.zoo()` plots time series.

The function `axis()` plots customized axes in an existing plot.

```
> # Open plot window and set plot margins
> x11(width=6, height=4)
> par(mar=c(2, 2, 2, 2), oma=c(1, 1, 1, 1))
> # Plot first time series without x-axis
> zoo::plot.zoo(pricev[, 1], lwd=2, col="orange",
+   xlab=NA, ylab=NA, xaxt="n")
> # Create X-axis date labels and add X-axis
> datev <- pretty(zoo::index(pricev))
> axis(side=1, at=datev, labels=format(datev, "%b-%d-%y"))
> # Plot second time series without y-axis
> par(new=TRUE) # Allow new line on same plot
> zoo::plot.zoo(pricev[, 2], xlab=NA, ylab=NA,
+   lwd=2, yaxt="n", col="blue", xaxt="n")
> # Plot second y-axis on right
> axis(side=4, lwd=2, col="blue")
> # Add axis labels
> mtext(colnamev[1], cex=1.2, lwd=3, side=2, las=2, adj=(-0.5), padj=(-5), col="orange")
> mtext(colnamev[2], cex=1.2, lwd=3, side=4, las=2, adj=1.5, padj=(-5), col="blue")
> # Add title and legend
> title(main=paste(colnamev, collapse=" and "), line=0.5)
> legend("top", legend=colnamev, cex=1.0, bg="white",
+   lty=1, lwd=6, col=c("orange", "blue"), bty="n")
```



Plotting *OHLC* Time Series Using Package *dygraphs*

The function `dygraph()` from package *dygraphs* creates interactive plots for *xts* time series.

The function `dyCandlestick()` creates a *candlestick* plot object for *OHLC* data, and uses the first four columns to plot *candlesticks*, and it plots any additional columns as lines.

The function `dyOptions()` adds options (like colors, etc.) to a *dygraph* plot.

```
> library(dygraphs)
> # Calculate volume-weighted average price
> ohlc <- rutils::etfenv$VTI
> wvavp <- TTR::VWAP(price=quantmod::Cl(ohlc),
+   volume=quantmod::Vo(ohlc), n=20)
> # Add VWAP to OHLC data
> datav <- cbind(ohlc[, 1:4], wvavp)["2009-01/2009-04"]
> # Create dygraphs object
> dyplot <- dygraphs::dygraph(datav)
> # Increase line width and color
> dyplot <- dygraphs::dyOptions(dyplot,
+   colors="red", strokeWidth=3)
> # Convert dygraphs object to candlestick plot
> dyplot <- dygraphs::dyCandlestick(dyplot)
> # Render candlestick plot
> dyplot
> # Candlestick plot using pipes syntax
> dygraphs::dygraph(datav) %>% dyCandlestick() %>%
+   dyOptions(colors="red", strokeWidth=3)
> # Candlestick plot without using pipes syntax
> dygraphs::dyCandlestick(dygraphs::dyOptions(dygraphs::dygraph(datav),
+   colors="red", strokeWidth=3))
```



Each *candlestick* displays one period of data, and consists of a box representing the *Open* and *Close* prices, and a vertical line representing the *High* and *Low* prices.

The color of the box signifies whether the *Close* price was higher or lower than the *Open*,

dygraphs OHLC Plots With Background Shading

The function `dyShading()` adds shading to a *dygraphs* plot object.

The function `dyShading()` requires a vector of dates for shading.

```
> # Create candlestick plot with background shading
> indic <- (quantmod::Cl(datav) > datav[, "VWAP"])
> whichv <- which(rutils::diffit(indic) != 0)
> indic <- rbind(first(indic), indic[whichv, ], last(indic))
> datev <- zoo::index(indic)
> indic <- ifelse(drop(coredata(indic)), "lightgreen", "antiquewhite")
> # Create dygraph object without rendering it
> dyplot <- dygraphs::dygraph(datav) %>% dyCandlestick() %>%
+   dyOptions(colors="red", strokeWidth=3)
> # Add shading
> for (i in 1:(NROW(indic)-1)) {
+   dyplot <- dyplot %>%
+   dyShading(from=datev[i], to=datev[i+1], color=indic[i])
+ } # end for
> # Render the dygraph object
> dyplot
```

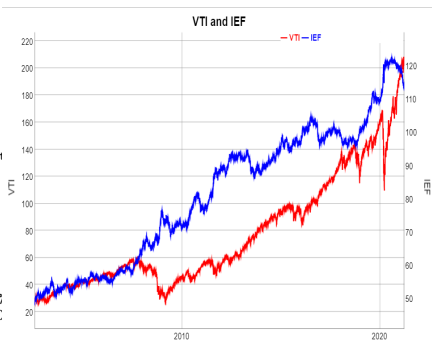


dygraphs Plots With Two "y" Axes

The function `dyAxis()` from package *dygraphs* plots customized axes to a *dygraphs* plot object.

The function `dySeries()` adds a time series to a *dygraphs* plot object.

```
> library(dygraphs)
> # Prepare VTI and IEF prices
> pricev <- cbind(quantmod::Cl(rutils::etfenv$VTI), quantmod::Cl(rut
> pricev <- na.omit(pricev)
> colnamev <- rutils::get_name(colnames(pricev))
> colnames(pricev) <- colnamev
> # dygraphs plot with two y-axes
> library(dygraphs)
> dygraphs::dygraph(pricev, main=paste(colnamev, collapse=" and "))
+ dyAxis(name="y", label=colnamev[1], independentTicks=TRUE) %>%
+ dyAxis(name="y2", label=colnamev[2], independentTicks=TRUE) %>%
+ dySeries(name=colnamev[1], axis="y", strokeWidth=2, col="red") ;
+ dySeries(name=colnamev[2], axis="y2", strokeWidth=2, col="blue") ;
```



draft: Package *qmao* for Quantitative Financial Modeling

The package *qmao* is designed for downloading, manipulating, and visualizing *OHLC* time series data, package *quantmod*

qmao uses time series of class "xts", and provides many useful functions for building quantitative financial models:

- `getSymbols()` for downloading data from external sources (*Yahoo*, *FRED*, etc.),
- `getFinancials()` for downloading financial statements,
- `adjustOHLC()` for adjusting *OHLC* data,
- `Op()`, `Cl()`, `Vo()`, etc. for extracting *OHLC* data columns,
- `periodReturn()`, `dailyReturn()`, etc. for calculating periodic returns,
- `chartSeries()` for candlestick plots of *OHLC* data,
- `addBBands()`, `addMA()`, `addVo()`, etc. for adding technical indicators (Moving Averages, Bollinger Bands) and volume data to a plot,

```
> # Load package qmao
> library(qmao)
> # Get documentation for package qmao
> # Get short description
> packageDescription("qmao")
> # Load help page
> help(package="qmao")
> # List all datasets in "qmao"
> data(package="qmao")
> # List all objects in "qmao"
> ls("package:qmao")
> # Remove qmao from search path
> detach("package:qmao")
```