COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

Introduction to Trading Systems

Guy Yollin

Applied Mathematics
University of Washington

Outline

Outline

Lecture references

- quantmod package
  - Jeffrey Ryan's quantmod website:
    `http://www.quantmod.com/`
  - quantmod project page on R-forge:
    `http://r-forge.r-project.org/projects/quantmod`
- TTR package
  - Joshua Ulrich's Foss Trading blog:
    `http://blog.fosstrading.com/`
  - TTR project page on R-forge:
    `http://r-forge.r-project.org/projects/ttr/`
- xts package
  - xts project page on R-forge:
    `http://r-forge.r-project.org/projects/xts/`

Quantitative analysis package hierarchy

Application Area	R Package
Performance metrics and graphs	**PerformanceAnalytics** - Tools for performance and risk analysis
Portfolio optimization and quantitative trading strategies	**PortfolioAnalytics** - Portfolio analysis and optimization
	quantstrat – Rules-based trading system development
	blotter – Trading system accounting infrastructure
Data access and financial charting	**quantmod** - Quantitative financial modeling framework
	TTR - Technical trading rules
Time series objects	**xts** - Extensible time series
	zoo - Ordered observation

The zoo package

The zoo package provides an infrastructure for regularly-spaced and irregularly-space time series

Key functions:

zoo	create a zoo time series object
merge	merges time series (automatically handles of time alignment)
aggregate	create coarser resolution time series with summary statistics
rollapply	calculate rolling window statistics
read.zoo	read a text file into a zoo time series object

Authors:

- Achim Zeileis
- Gabor Grothendieck

The xts package

The `xts` package extends the zoo time series class with fine-grained time indexes, interoperability with other R time series classes, and user defined attributes

Key functions:

xts	create an xts time series object
align.time	align time series to a coarser resolution
to.period	convert time series data to an OHLC series
[.xts	subset time series

Authors:

- Jeffrey Ryan
- Josh Ulrich

R-forge

- R-forge is a hosting platform for R package development which includes SVN, daily build and check, mailing lists, bug tracking, message boards etc.
- More then 1000 R packages are hosted on R-forge including all of the trading system development packages mentioned earlier
- It is common for new packages to be developed on R-forge and for mature packages to be maintained on R-forge even after being hosted on CRAN

Install trading system development packages

```
#
# install these packages from CRAN (or r-forge)
#
install.packages("xts", dependencies=TRUE)
install.packages("PerformanceAnalytics", dependencies=TRUE)
#
# Install these package from r-forge
#
install.packages("quantmod", repos = "http://R-Forge.R-project.org")
install.packages("TTR", repos = "http://R-Forge.R-project.org")
install.packages("FinancialInstrument", repos = "http://R-Forge.R-project.org")
install.packages("blotter", repos = "http://R-Forge.R-project.org")
install.packages("quantstrat", repos = "http://R-Forge.R-project.org")
```

- R-Forge packages can be installed by setting the repos argument to `http://R-Forge.R-project.org`

Outline

The quantmod package

The `quantmod` package for R is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models.

Key functions:

getSymbols load or download price data

- Yahoo Finance / Google Finance
- FRED
- Oanda
- csv, RData
- MySQL, SQLite

chartSeries charting tool to create standard financial charts

Author:

- Jeffrey Ryan

The getSymbols function

The getSymbols function loads (downloads) historic price data

Usage:

```
getSymbols(Symbols = NULL, env = parent.frame(), src = "yahoo",
  auto.assign = getOption('getSymbols.auto.assign',TRUE), ...)
```

Main arguments:

Symbols	a vector of ticker symbols
env	where to create objects. Setting env=NULL is equal to auto.assign=FALSE
src	source of data (yahoo)
auto.assign	should results be returned or loaded to env
...	additional parameters

Return value:

an object of type return.class depending on env and auto.assign

The getSymbols.yahoo function

The `getSymbols.yahoo` function downloads historic price data from `finance.yahoo.com`

Usage:

```
getSymbols.yahoo(Symbols, env, return.class = 'xts', index.class  = 'Date',
  from = "2007-01-01", to = Sys.Date(), ...)
```

Main arguments:

return.class	class of returned object
index.class	class of returned object index (xts only)
...	additional parameters

Return value:

an object of type `return.class` depending on `env` and `auto.assign`

The getSymbols function

```
library(quantmod)
getSymbols("^GSPC")
ls()

## [1] "filename" "GSPC"

class(GSPC)

## [1] "xts" "zoo"

class(index(GSPC))

## [1] "Date"

dim(GSPC)

## [1] 1933    6
```

- By default, the symbol was *auto-assigned* to the parent environment

The getSymbols function

```
tail(GSPC,4)
```

```
##            GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
## 2014-09-02   2004.07   2006.12  1994.85    2002.28  2819980000       2002.28
## 2014-09-03   2003.57   2009.28  1998.14    2000.72  2809980000       2000.72
## 2014-09-04   2001.67   2011.17  1992.54    1997.65  3072410000       1997.65
## 2014-09-05   1998.00   2007.71  1990.10    2007.71  2818300000       2007.71
```

```
tail(Cl(GSPC),4)
```

```
##            GSPC.Close
## 2014-09-02    2002.28
## 2014-09-03    2000.72
## 2014-09-04    1997.65
## 2014-09-05    2007.71
```

```
tail(Ad(GSPC),4)
```

```
##            GSPC.Adjusted
## 2014-09-02       2002.28
## 2014-09-03       2000.72
## 2014-09-04       1997.65
## 2014-09-05       2007.71
```

- Note that the symbol is prepended to columns names of the xts object; use extractor functions to access column data (e.g. `Cl(GSPC)`)

xts extractor functions

The xts package includes a number of functions to extract specific series from an xts object of market data:

Function	Description
Op(x)	Get Open
Hi(x)	Get High
Lo(x)	Get Low
Cl(x)	Get Close
Vo(x)	Get Volume
Ad(x)	Get Adjusted Close
HLC(x)	Get High, Low, and Close
OHLC(x)	Get Open, High, Low, and Close

The chartSeries function

The chartSeries function creates financial charts

Usage:

```
getSymbols(Symbols = NULL, env = parent.frame(), src = "yahoo",
  auto.assign = getOption('getSymbols.auto.assign',TRUE), ...)
```

Main arguments:

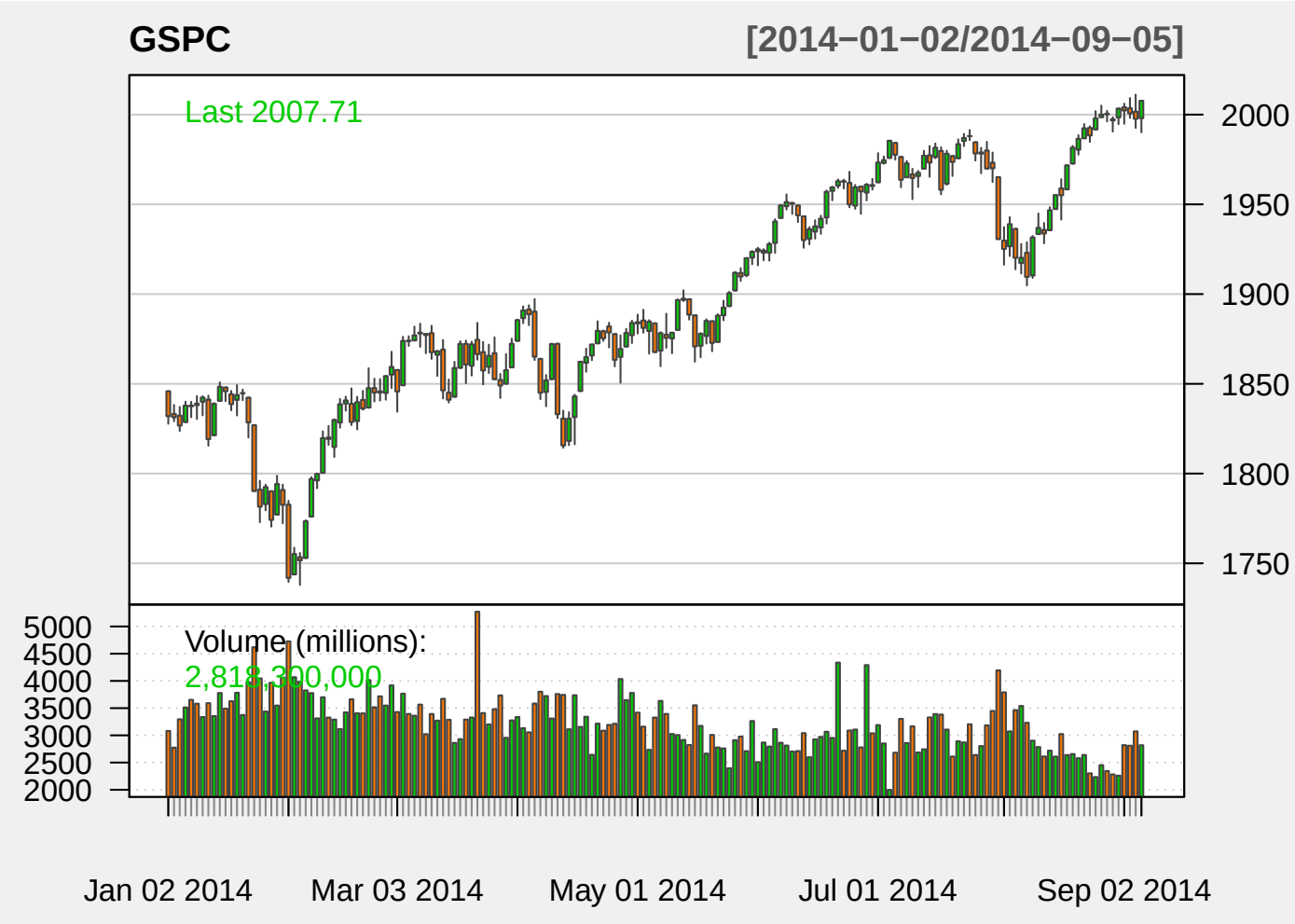
x	an OHLC object
type	style of chart to draw
theme	a chart.theme object
subset	xts style date subsetting argument
TA	a vector of technical indicators and params

Return value:

a chob object

The chartSeries function

```
chartSeries(GSPC,subset="2014",theme="white")
```



Customize a chartSeries plot

```
whiteTheme <- chartTheme("white")
names(whiteTheme)
```

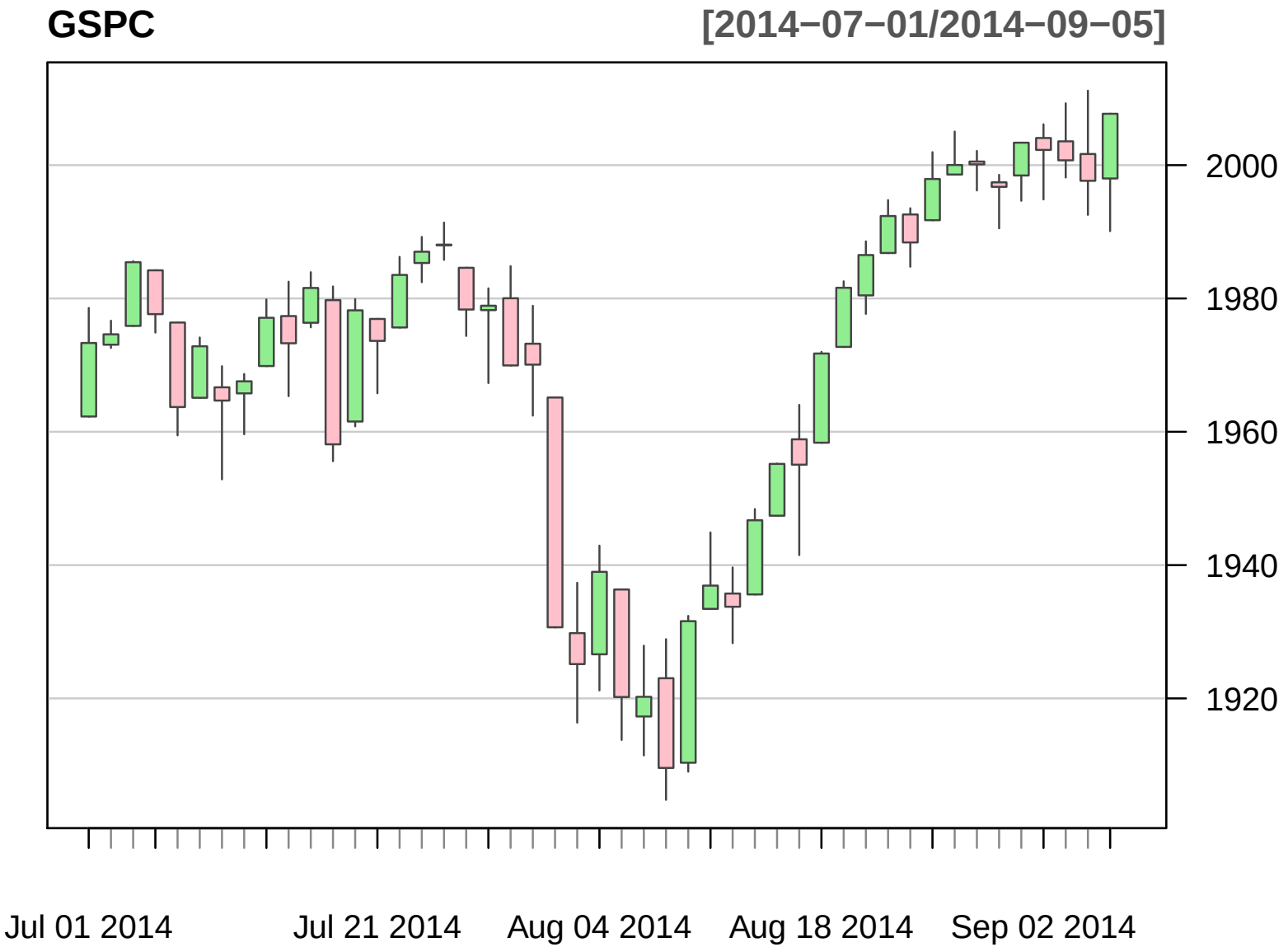
```
##  [1] "fg.col"       "bg.col"       "grid.col"     "border"       "minor.tick"
##  [6] "major.tick"   "up.col"       "dn.col"       "dn.up.col"    "up.up.col"
## [11] "dn.dn.col"    "up.dn.col"    "up.border"    "dn.border"    "dn.up.border"
## [16] "up.up.border" "dn.dn.border" "up.dn.border" "main.col"     "sub.col"
## [21] "area"         "fill"         "Expiry"       "theme.name"
```

```
whiteTheme$bg.col <- "white"
whiteTheme$dn.col <- "pink"
whiteTheme$up.col <- "lightgreen"
whiteTheme$border <- "lightgray"
x <- chartSeries(GSPC,subset="last 3 months",theme=whiteTheme,TA=NULL)
class(x)
```

```
## [1] "chob"
## attr(,"package")
## [1] "quantmod"
```

- subset to last 3 months
- totally white background
- no volume sub-graph (`TA=NULL`)

A chartSeries plot



Outline

Time series data

Almost all data related to trading is a time series:

- market data
- technical analysis indicators
- trades
- position data
- P&L

Time series data

Time series

A *time series* is a sequence of *ordered* data points measured at specific points in time

Time series object

A time series object in R is a *compound data structure* that includes a data matrix as well as a vector of associated time stamps

class	package	overview
`ts`	stats	regularly spaced time series
`mts`	stats	multiple regularly spaced time series
`zoo`	zoo	reg/irreg and arbitrary time stamp classes
`xts`	xts	an extension of the zoo class

Time series methods

Time series classes in R will typically implement the following methods:

`start`	return start of time series
`end`	return end of time series
`frequency`	return frequency of time series
`window`	Extract subset of time series
`index`	return time index of time series
`time`	return time index of time series
`coredata`	return data of time series
`diff`	difference of the time series
`lag`	lag of the time series
`aggregate`	aggregate to lower resolution time series
`cbind`	merge 2 or more time series together
`merge`	merge 2 or more time series together

Components of an xts object

An `xts` object is composed of 3 components:

Index a Data class or Time-Date class used for the time-stamp of observations

Matrix the time series observations (univariate or multivariate)

- can be numeric, character, logical, etc. but must be homogeneous

Attr hidden attributes and user attributes

- class of the index
- format of the index
- time zone

Date class

- A `Date` object is stored internally as the days since 1970-01-01

```
myStr <- "2013-07-04"
class(myStr)
```

```
## [1] "character"
```

```
myDate <- as.Date(myStr)
class(myDate)
```

```
## [1] "Date"
```

```
as.numeric(myDate)
```

```
## [1] 15890
```

```
format(myDate,"%m/%d/%y")
```

```
## [1] "07/04/13"
```

```
as.Date("110704",format="%y%m%d")
```

```
## [1] "2011-07-04"
```

Date-Time classes

- A `POSIXct` object is a Date-Time object internally stored as the number of seconds since 1970-01-01
- A `POSIXlt` object is a Date-Time object internally stored as 9 calendar and time components

```
d <- Sys.time()
class(d)
```

```
## [1] "POSIXct" "POSIXt"
```

```
unclass(d)
```

```
## [1] 1410038854
```

```
sapply(unclass(as.POSIXlt(d)), function(x) x)
```

```
##                sec                min               hour               mday
## "33.5228459835052"              "27"              "14"               "6"
##                mon               year               wday               yday
##                "8"              "114"               "6"              "248"
##              isdst               zone             gmtoff
##                "1"              "PDT"          "-25200"
```

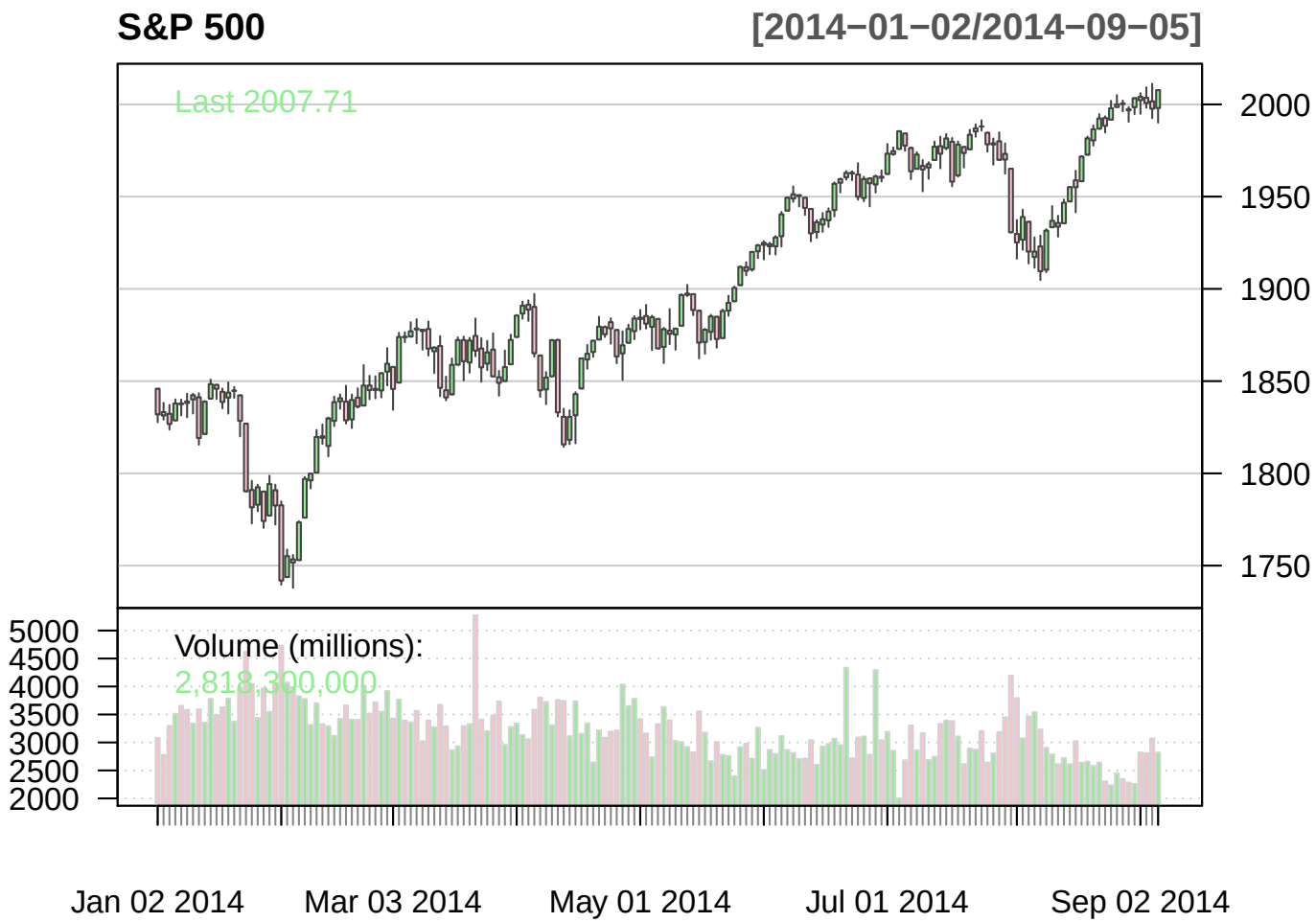
xts subsetting

xts time series objects can be easily subset:

- Date and time organized from most significant to least significant
  - CCYY-MM-DD HH:MM:SS[.s]
- Separators can be omitted
  - CCYYMMDDHHMMSS
- Intervals can be designated with the "/" or "::"
  - 2010/2011
  - 2011-04::2011-07

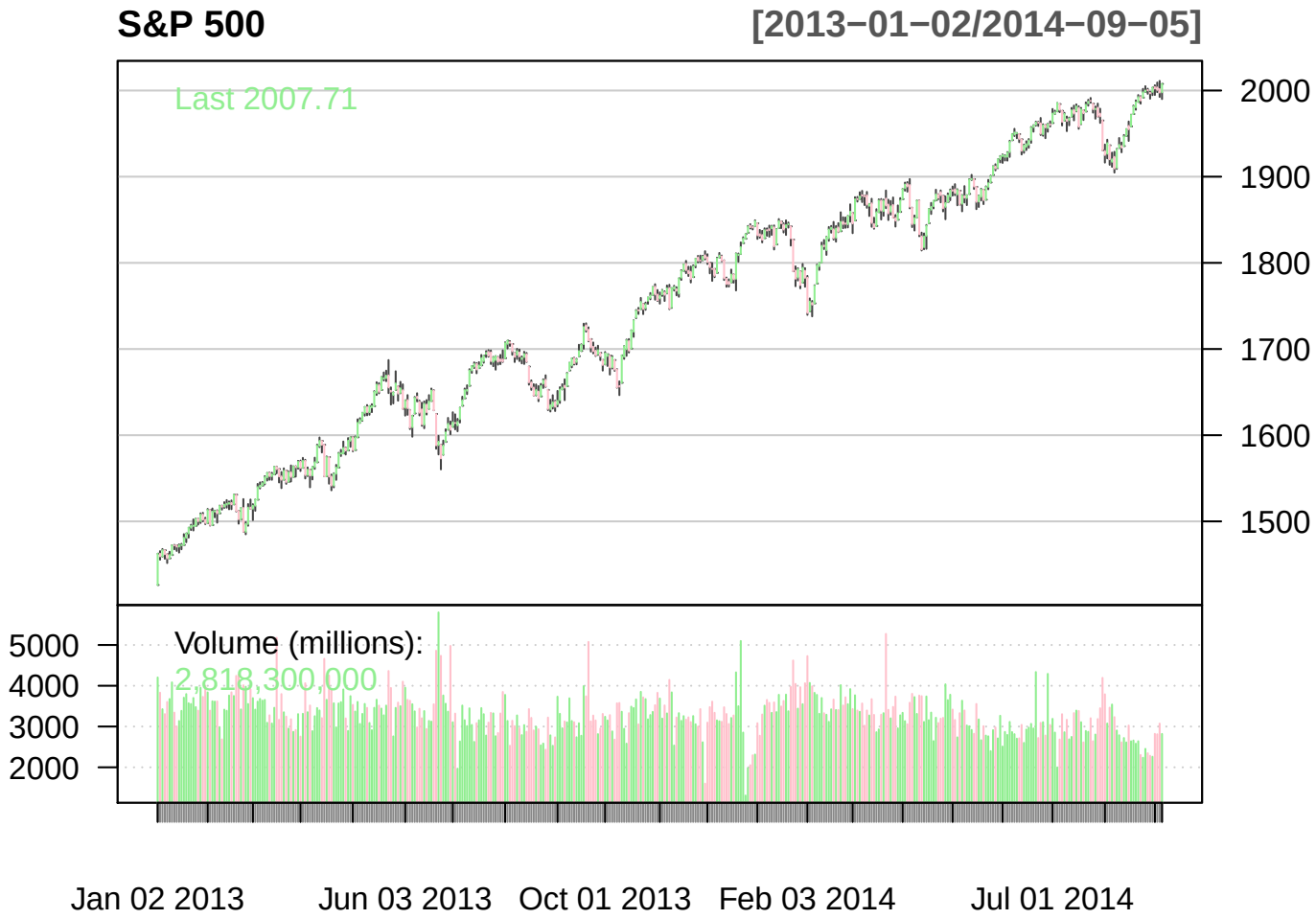
xts subsetting

```
chartSeries(GSPC["2014"],theme=whiteTheme,name="S&P 500")
```



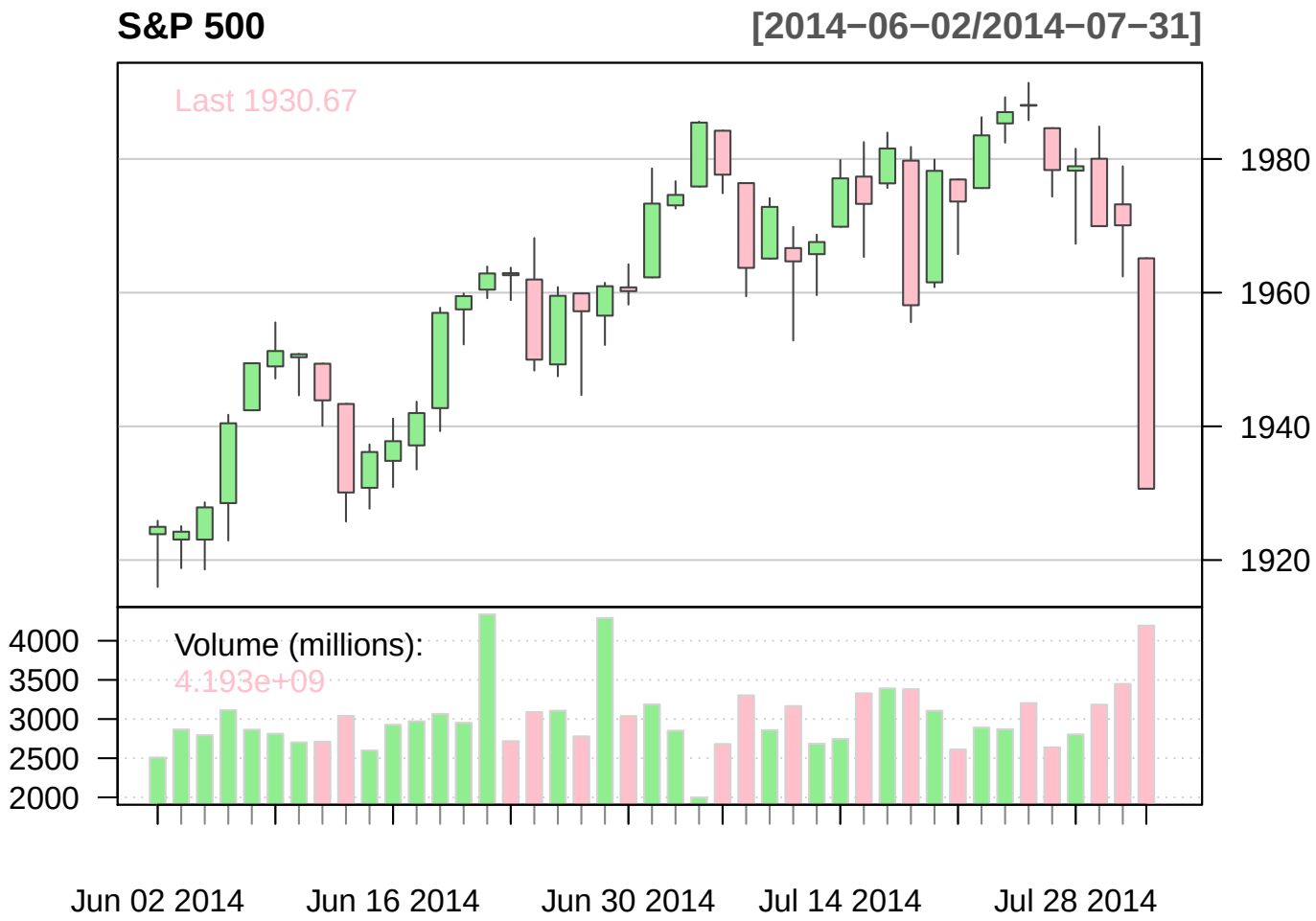
xts subsetting

```
chartSeries(GSPC["2013/2014"],theme=whiteTheme,name="S&P 500")
```



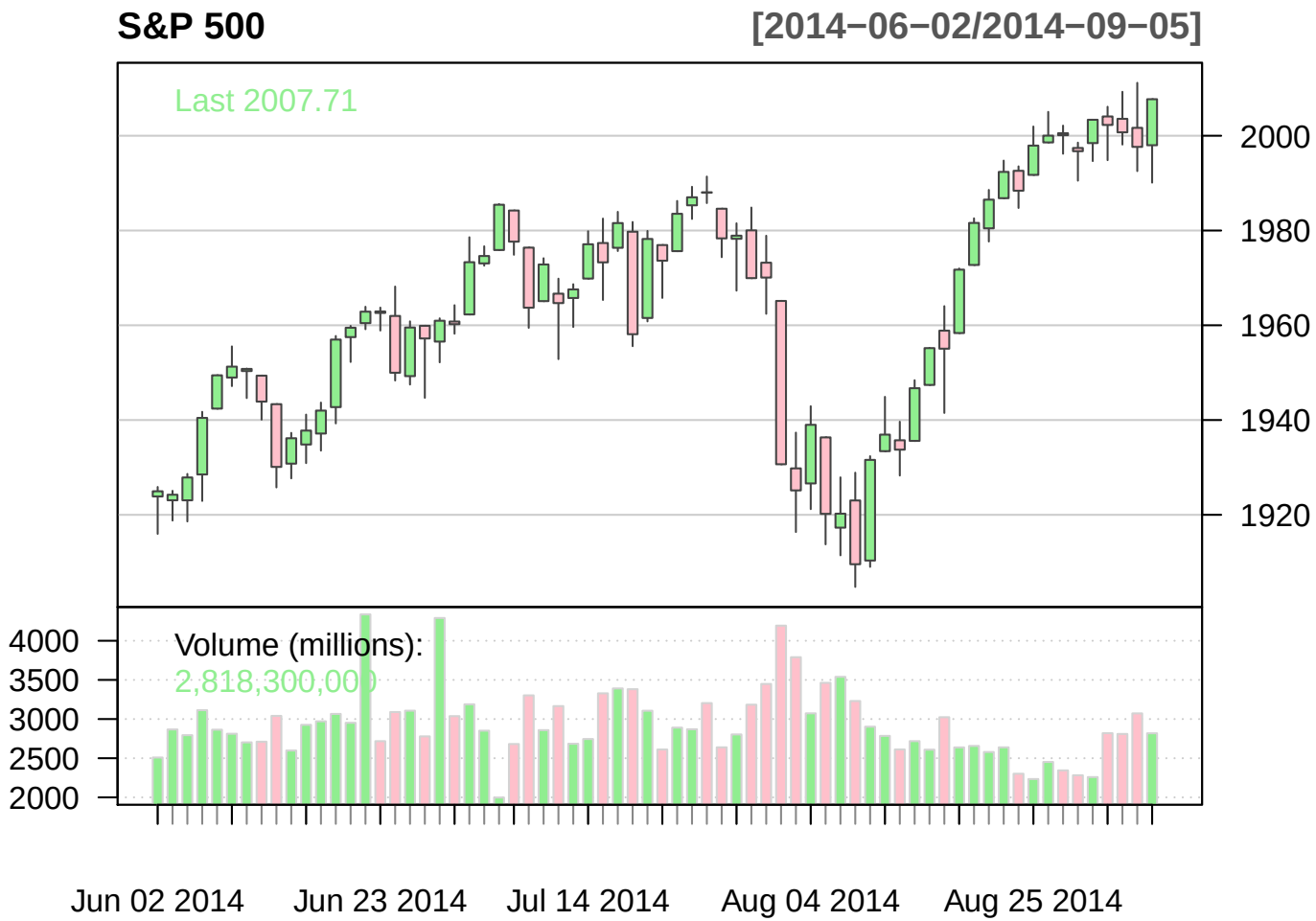
xts subsetting

```
chartSeries(GSPC["2014-06::2014-07"],theme=whiteTheme,name="S&P 500")
```



xts subsetting

```
chartSeries(GSPC["201406::"],theme=whiteTheme,name="S&P 500")
```



Outline

Download historic quotes: specifying the start date

```
getSymbols("SPY",from="2000-01-01")
class(SPY)
```

```
## [1] "xts" "zoo"
```

```
head(SPY)
```

```
##            SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted
## 2000-01-03   148.25   148.25  143.88    145.44    8164300       111.46
## 2000-01-04   143.53   144.06  139.64    139.75    8089800       107.10
## 2000-01-05   139.94   141.53  137.25    140.00   12177900       107.29
## 2000-01-06   139.62   141.50  137.75    137.75    6227200       105.56
## 2000-01-07   140.31   145.75  140.06    145.75    8066500       111.70
## 2000-01-10   146.25   146.91  145.03    146.25    5741700       112.08
```

```
head(index(SPY))
```

```
## [1] "2000-01-03" "2000-01-04" "2000-01-05" "2000-01-06" "2000-01-07" "2000-01-10"
```

```
class(index(SPY))
```

```
## [1] "Date"
```

Download historic quotes: specifying the time stamp class

```
getSymbols("SBUX",index.class="POSIXct",from="2000-01-01")
```

```
class(SBUX)
```

```
## [1] "xts" "zoo"
```

```
head(SBUX)
```

```
##            SBUX.Open SBUX.High SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03     23.88     24.69    23.25      24.66    12116000          5.76
## 2000-01-04     24.06     24.88    23.75      23.88    10782400          5.58
## 2000-01-05     23.94     24.62    23.69      24.19    14103200          5.65
## 2000-01-06     24.00     25.62    24.00      25.06    15412800          5.86
## 2000-01-07     24.75     25.00    24.25      24.94    13022400          5.83
## 2000-01-10     25.88     26.75    25.81      26.00    14515600          6.08
```

```
head(index(SBUX))
```

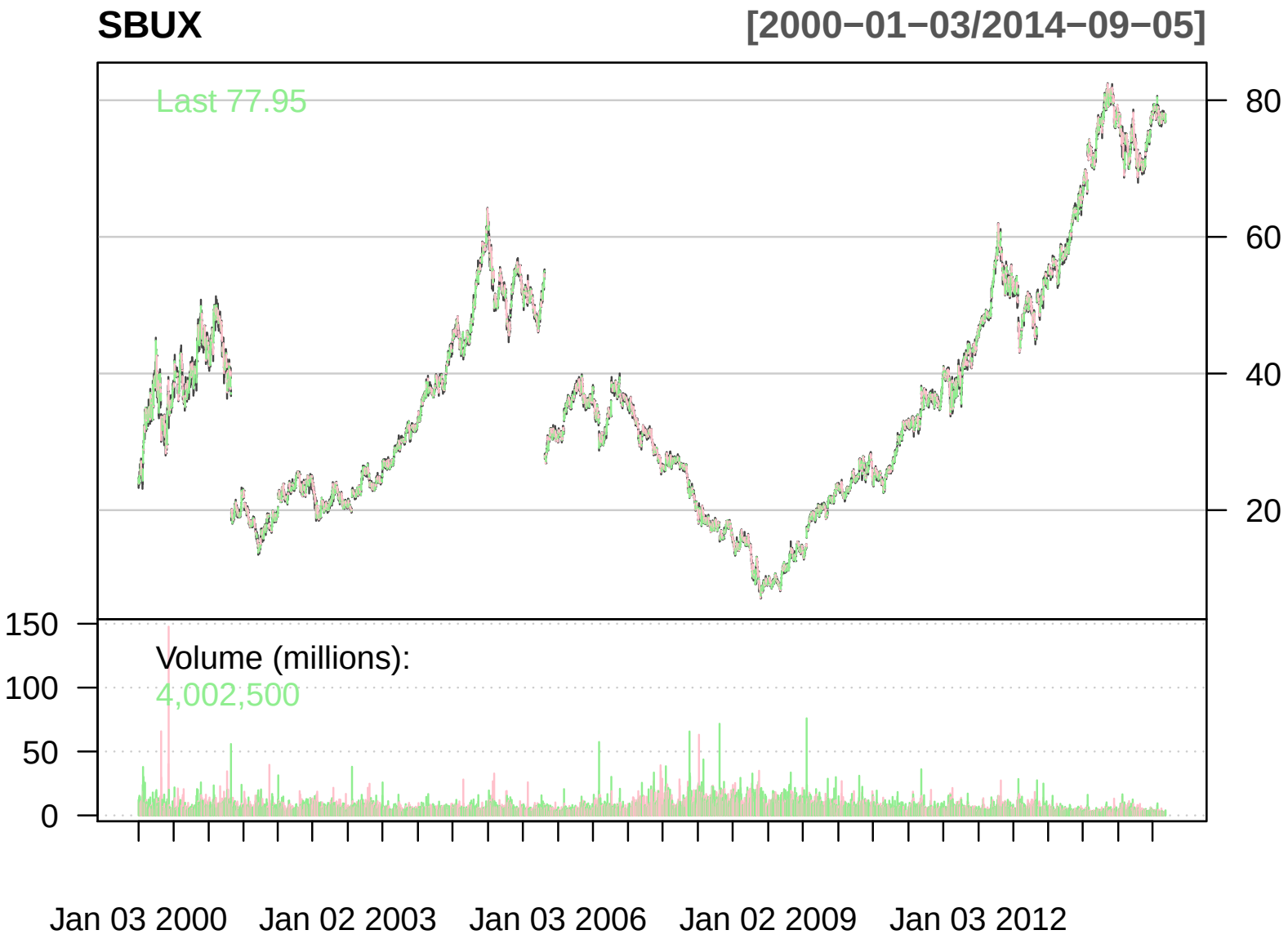
```
## [1] "2000-01-03 UTC" "2000-01-04 UTC" "2000-01-05 UTC" "2000-01-06 UTC"
## [5] "2000-01-07 UTC" "2000-01-10 UTC"
```

```
class(index(SBUX))
```

```
## [1] "POSIXct" "POSIXt"
```

```
chartSeries(SBUX,theme=whiteTheme,minor.ticks=FALSE)
```

Unadjusted SBUX data



Get stock split history

```
(spl <- getSplits("SBUX"))
```

```
##            SBUX.spl
## 1993-09-30      0.5
## 1995-12-04      0.5
## 1999-03-22      0.5
## 2001-04-30      0.5
## 2005-10-24      0.5
```

```
class(spl)
```

```
## [1] "xts" "zoo"
```

Get stock dividend history

```
(div <- getDividends("SBUX"))
```

```
##            [,1]
## 2010-04-05 0.10
## 2010-08-02 0.13
## 2010-11-16 0.13
## 2011-02-07 0.13
## 2011-05-09 0.13
## 2011-08-08 0.13
## 2011-11-15 0.17
## 2012-02-06 0.17
## 2012-05-07 0.17
## 2012-08-06 0.17
## 2012-11-13 0.21
## 2013-02-05 0.21
## 2013-05-07 0.21
## 2013-08-06 0.21
## 2013-11-12 0.26
## 2014-02-04 0.26
## 2014-05-06 0.26
## 2014-08-05 0.26
```

```
class(div)
```

```
## [1] "xts" "zoo"
```

The adjustOHLC function

The adjustOHLC adjusts all columns of an OHLC object for split and dividend

Usage:

```
adjustOHLC(x, adjust = c("split","dividend"), use.Adjusted = FALSE,
 ratio = NULL, symbol.name=deparse(substitute(x)))
```

Main arguments:

x	an OHLC object
use.Adjusted	calculated from dividends and splits, or used Adjusted price column

Return value:

An object of the original class, with prices adjusted for splits and dividends

---

Using use.Adjusted = TRUE will be less precise than the method that employs actual split and dividend information

Adjust for split and dividend

```
head(SBUX)
```

```
##            SBUX.Open SBUX.High SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03     23.88     24.69    23.25      24.66    12116000          5.76
## 2000-01-04     24.06     24.88    23.75      23.88    10782400          5.58
## 2000-01-05     23.94     24.62    23.69      24.19    14103200          5.65
## 2000-01-06     24.00     25.62    24.00      25.06    15412800          5.86
## 2000-01-07     24.75     25.00    24.25      24.94    13022400          5.83
## 2000-01-10     25.88     26.75    25.81      26.00    14515600          6.08
```

```
adj.exact <- adjustOHLC(SBUX)
head(adj.exact)
```

```
##            SBUX.Open SBUX.High  SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 5.5818405 5.7711743 5.4345809  5.7641619    12116000          5.76
## 2000-01-04 5.6239146 5.8155859 5.5514536  5.5818405    10782400          5.58
## 2000-01-05 5.5958652 5.7548121 5.5374288  5.6543016    14103200          5.65
## 2000-01-06 5.6098899 5.9885575 5.6098899  5.8576601    15412800          5.86
## 2000-01-07 5.7851990 5.8436353 5.6683263  5.8296106    13022400          5.83
## 2000-01-10 6.0493313 6.2526898 6.0329691  6.0773807    14515600          6.08
```

- An article that describes how Yahoo calculates the adjusted close can be found here:

  http://help.yahoo.com/kb/index?locale=en_US&y=PROD_ACCT&page=content&id=SLN2311

Compare adjustment methods

```
head(adj.exact)
```

```
##            SBUX.Open SBUX.High  SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 5.5818405 5.7711743 5.4345809  5.7641619    12116000          5.76
## 2000-01-04 5.6239146 5.8155859 5.5514536  5.5818405    10782400          5.58
## 2000-01-05 5.5958652 5.7548121 5.5374288  5.6543016    14103200          5.65
## 2000-01-06 5.6098899 5.9885575 5.6098899  5.8576601    15412800          5.86
## 2000-01-07 5.7851990 5.8436353 5.6683263  5.8296106    13022400          5.83
## 2000-01-10 6.0493313 6.2526898 6.0329691  6.0773807    14515600          6.08
```

```
adj.approx <- adjustOHLC(SBUX, use.Adjusted=TRUE)
head(adj.approx)
```

```
##            SBUX.Open SBUX.High  SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 5.5778102 5.7670073 5.4306569       5.76    12116000          5.76
## 2000-01-04 5.6220603 5.8136683 5.5496231       5.58    10782400          5.58
## 2000-01-05 5.5916081 5.7504341 5.5332162       5.65    14103200          5.65
## 2000-01-06 5.6121309 5.9909497 5.6121309       5.86    15412800          5.86
## 2000-01-07 5.7855854 5.8440257 5.6687049       5.83    13022400          5.83
## 2000-01-10 6.0519385 6.2553846 6.0355692       6.08    14515600          6.08
```

```
chartSeries(adj.exact,theme=whiteTheme,name="SBUX",minor.ticks=FALSE)
```

Adjusted SBUX plot

Download historic quotes: specifying adjusted OHLC

```
getSymbols("SBUX",index.class="POSIXct",from="2000-01-01",adjust=T)
```

```
## [1] "SBUX"
```

```
head(SBUX)
```

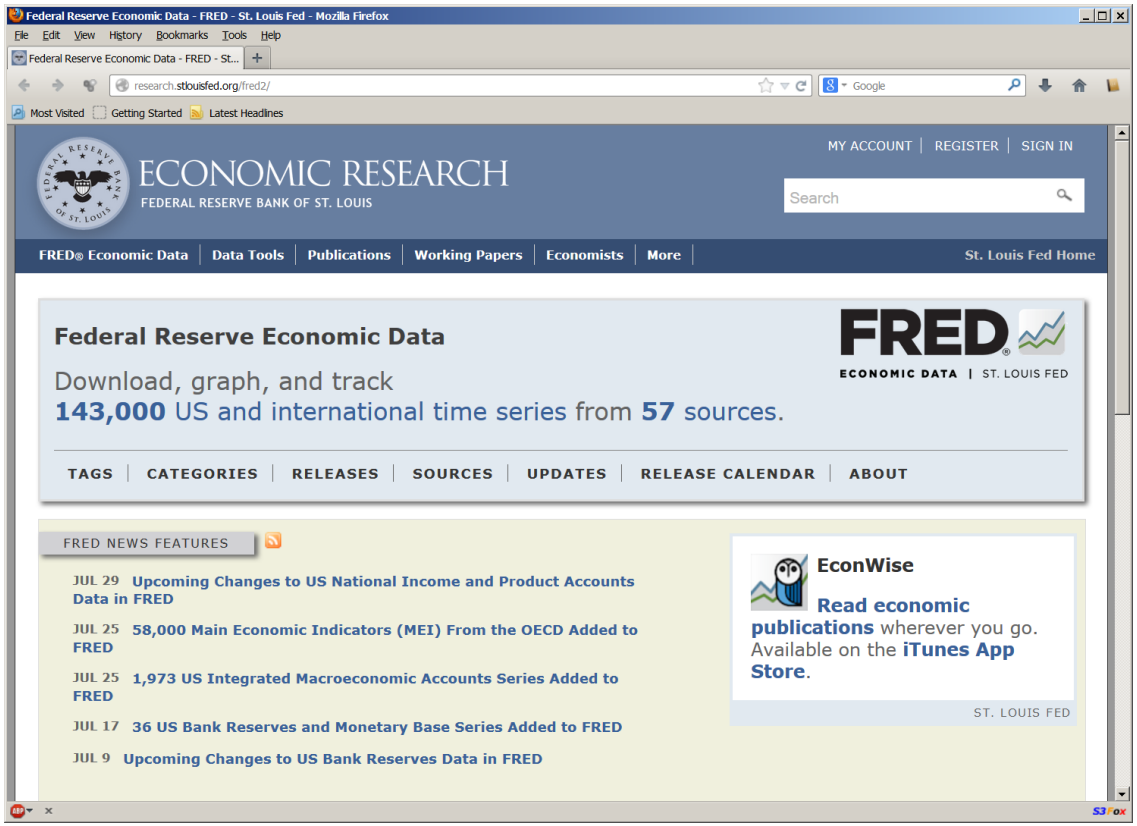
```
##            SBUX.Open SBUX.High  SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 5.5818405 5.7711743 5.4345809  5.7641619    12958543          5.76
## 2000-01-04 5.6239146 5.8155859 5.5514536  5.5818405    11532205          5.58
## 2000-01-05 5.5958652 5.7548121 5.5374288  5.6543016    15083932          5.65
## 2000-01-06 5.6098899 5.9885575 5.6098899  5.8576601    16484602          5.86
## 2000-01-07 5.7851990 5.8436353 5.6683263  5.8296106    13927974          5.83
## 2000-01-10 6.0493313 6.2526898 6.0329691  6.0773807    15525011          6.08
```

```
head(adj.exact)
```

```
##            SBUX.Open SBUX.High  SBUX.Low SBUX.Close SBUX.Volume SBUX.Adjusted
## 2000-01-03 5.5818405 5.7711743 5.4345809  5.7641619    12116000          5.76
## 2000-01-04 5.6239146 5.8155859 5.5514536  5.5818405    10782400          5.58
## 2000-01-05 5.5958652 5.7548121 5.5374288  5.6543016    14103200          5.65
## 2000-01-06 5.6098899 5.9885575 5.6098899  5.8576601    15412800          5.86
## 2000-01-07 5.7851990 5.8436353 5.6683263  5.8296106    13022400          5.83
## 2000-01-10 6.0493313 6.2526898 6.0329691  6.0773807    14515600          6.08
```

Federal reserve economic data

The function `getSymbols` can also be used to access data from the Federal Reserve Economic Data (FRED) database

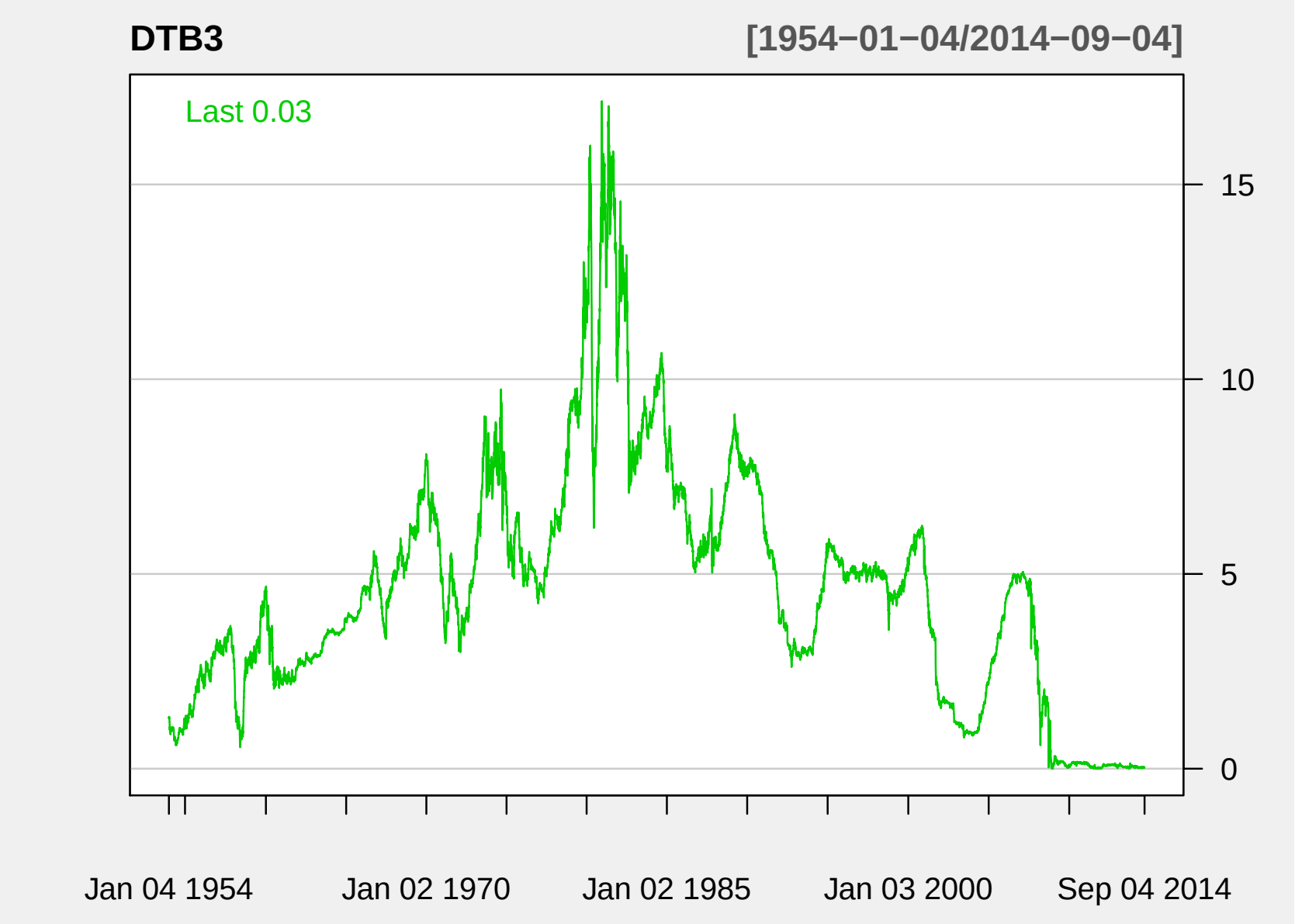


`http://research.stlouisfed.org/fred2/`

Download interest rate data from FRED

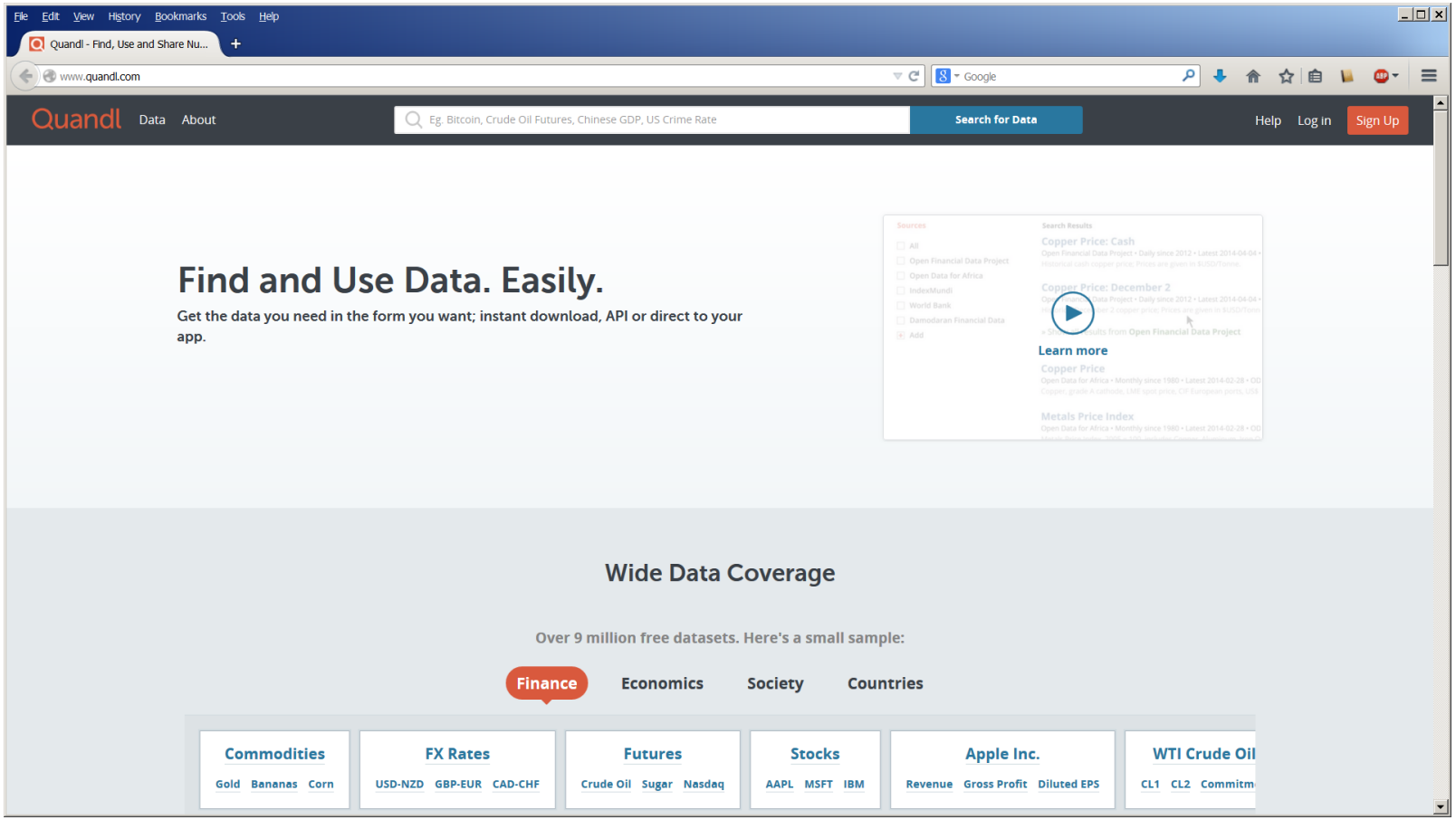
```
getSymbols('DTB3',src='FRED')
first(DTB3,'1 week')

##            DTB3
## 1954-01-04 1.33

last(DTB3,'1 week')

##            DTB3
## 2014-09-01   NA
## 2014-09-02 0.03
## 2014-09-03 0.03
## 2014-09-04 0.03

chartSeries(DTB3,theme="white",minor.ticks=FALSE)
```

Three-month U.S. Treasury bill rate



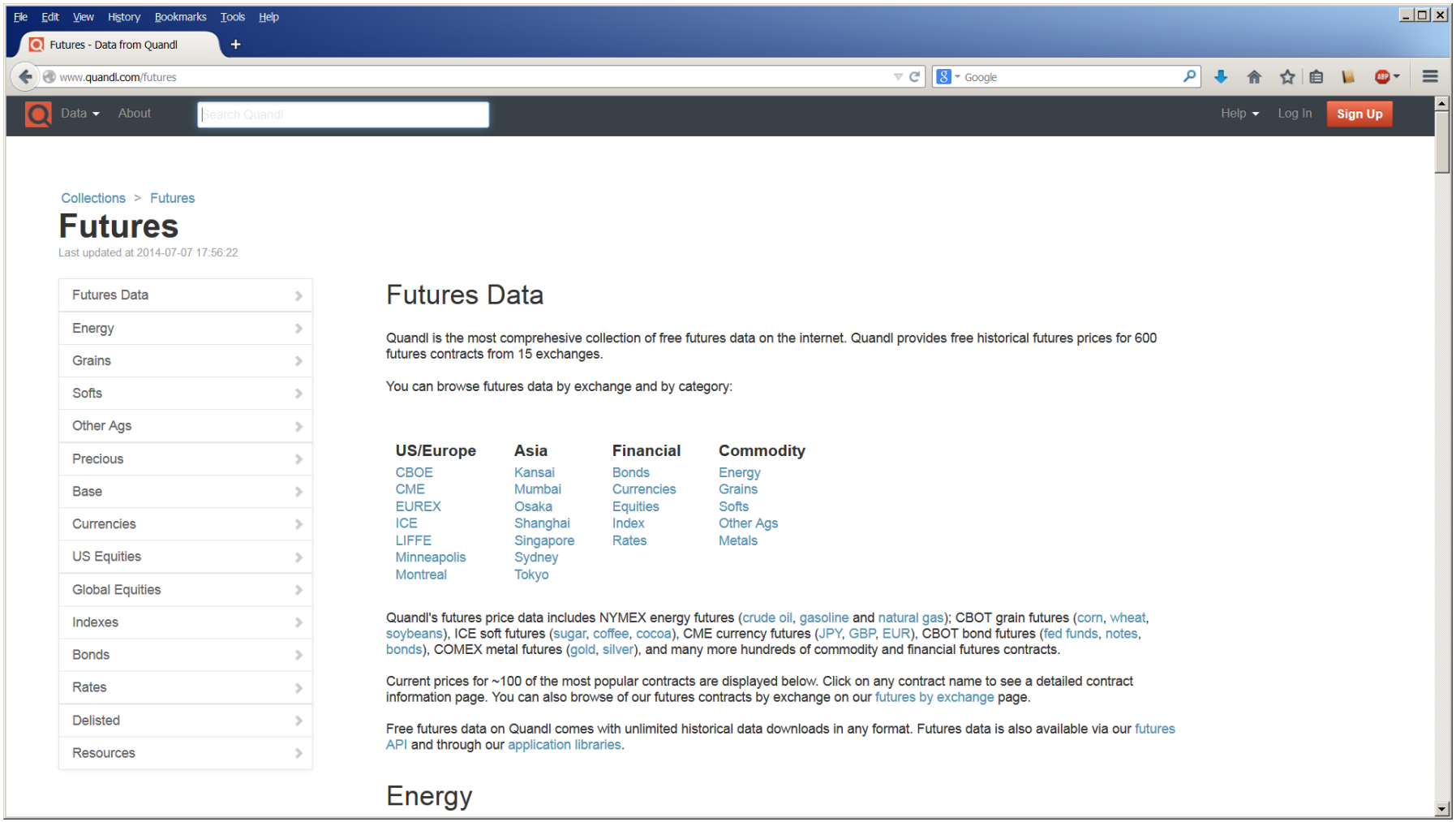
Quandl

Tammer Kamel the founder of `www.quandl.com` wants to do to Bloomberg what Wikipedia did to Britannica.



`http://www.quandl.com/`

Quandl futures



The Quandl package

The Quandl package interacts directly with the Quandl API to offer data in a number of formats usable in R

Key functions:

Quandl	Pulls data from the Quandl API
Quandl.auth	Query or set Quandl API token[†]
Quandl.search	Search the Quandl database

Authors:

- Raymond McTaggart

[†] Anonymous API calls are limited to 50 requests per day; signed up users receive an authorization token that allows them to get 500 API calls per day; see http://www.quandl.com/help/r

The Quandl function

The Quandl function pulls data from the Quandl API

Usage:

```
Quandl(code, type = c("raw", "ts", "zoo", "xts"), start_date, end_date,
 transformation = c("", "diff", "rdiff", "normalize", "cumul", "rdiff_from"),
 collapse = c("", "weekly", "monthly", "quarterly", "annual"),
 sort = c("desc", "asc"), meta = FALSE, authcode = Quandl.auth(), ...)
```

Main arguments:

code	Dataset code on Quandl specified as a string
type	Type of data returned ('raw', 'ts', 'zoo' or 'xts')
start_date	Start date
end_date	End date
collapse	Frequency of data

Return value:

time series data in the specified format

Downloading data from Quandl

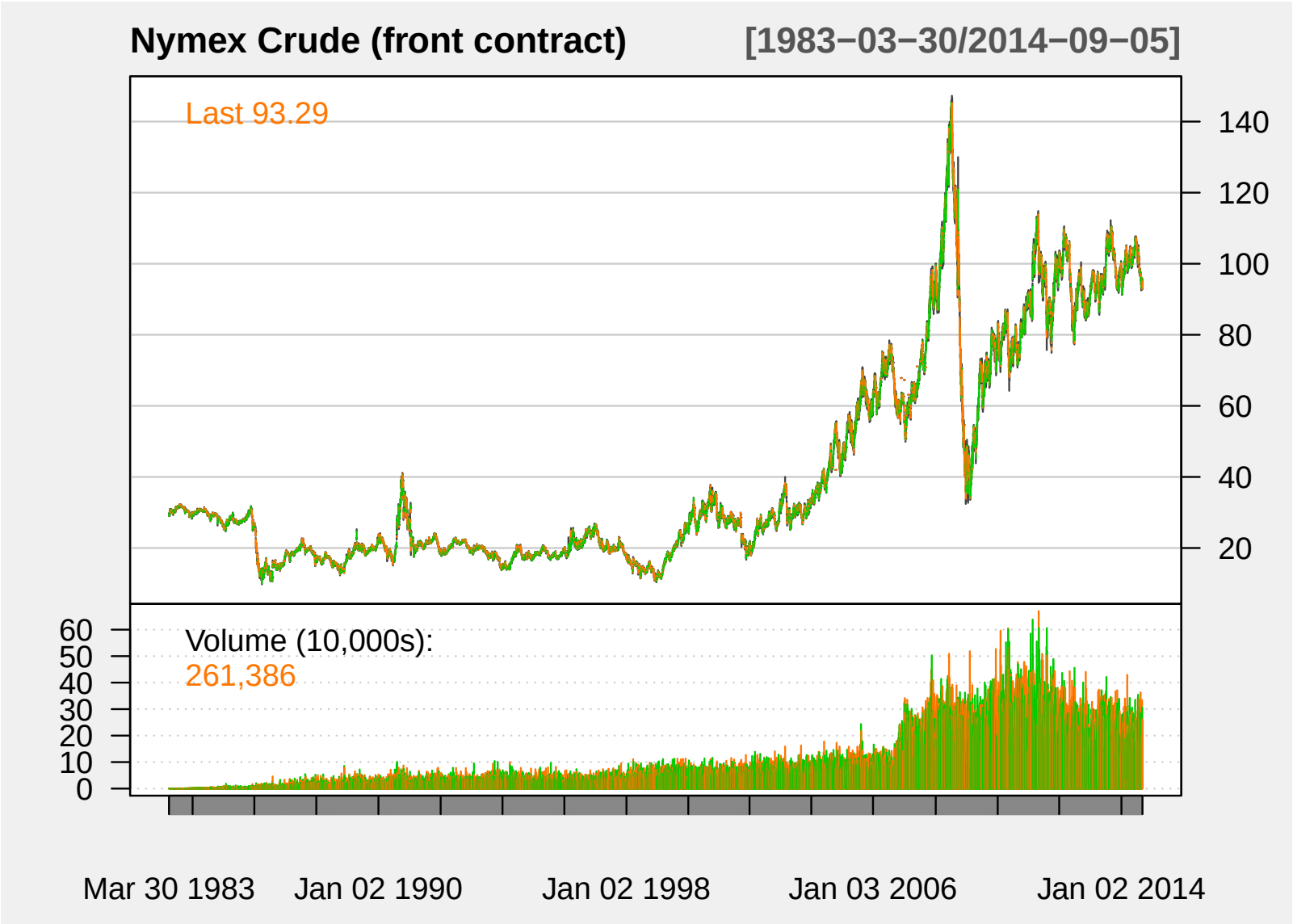
```
library(Quandl)
cl1 = Quandl("OFDP/FUTURE_CL1",type="xts")
class(cl1)

## [1] "xts" "zoo"

class(index(cl1))

## [1] "Date"

first(cl1)

##             Open  High   Low Settle Volume Open Interest
## 1983-03-30 29.01 29.56 29.01   29.4    949           470

last(cl1)

##             Open  High   Low Settle Volume Open Interest
## 2014-09-05 94.57 94.99 92.86  93.29 261386        235848
```

Downloading data from Quandl

```
cl1 <- cl1[,c("Open","High","Low","Settle","Volume")]
colnames(cl1) <-  c("Open","High","Low","Close","Volume")
sum(is.na(coredata(cl1)))

## [1] 0

sum(coredata(cl1)<0.01)

## [1] 17

cl1[cl1 < 0.1] <- NA
cl1 <- na.approx(cl1)
chartSeries(cl1,name="Nymex Crude (front contract)",theme=chartTheme("white"))
```

- Required data cleaning: replace zero values with NA and then use the function `na.approx` to interpolate

Historic futures data from Quandl



Outline

The TTR package

The `TTR` package is a comprehensive collection of technical analysis indicators for R

Key features:

- moving averages
- oscillators
- price channels
- trend indicators

Author:

- Joshua Ulrich

Selected technical analysis indicators in TTR

Function	Description	Function	Description
stoch	stochastic oscillator	ADX	Directional Movement Index
aroon	Aroon indicator	ATR	Average True Range
BBands	Bollinger bands	CCI	Commodity Channel Index
chaikinAD	Chaikin Acc/Dist	chaikinVolatility	Chaikin Volatility
ROC	rate of change	momentum	momentum indicator
CLV	Close Location Value	CMF	Chaikin Money Flow
CMO	Chande Momentum Oscillator	SMA	simple moving average
EMA	exponential moving average	DEMA	double exp mov avg
VWMA	volume weighted MA	VWAP	volume weighed avg price
DonchianChannel	Donchian Channel	DPO	Detrended Price Oscillator
EMV	Ease of Movement Value	volatility	volatility estimators
MACD	MA converge/diverge	MFI	Money Flow Index
RSI	Relative Strength Index	SAR	Parabolic Stop-and-Reverse
TDI	Trend Detection Index	TRIX	Triple Smoothed Exponential Osc
VHF	Vertical Horizontal Filter	williamsAD	Williams Acc/Dist
WPR	William's % R	ZigZag	Zig Zag trend line

see Technical Analysis from A to Z by Steven Achelis

Calculate and plot Bollinger bands

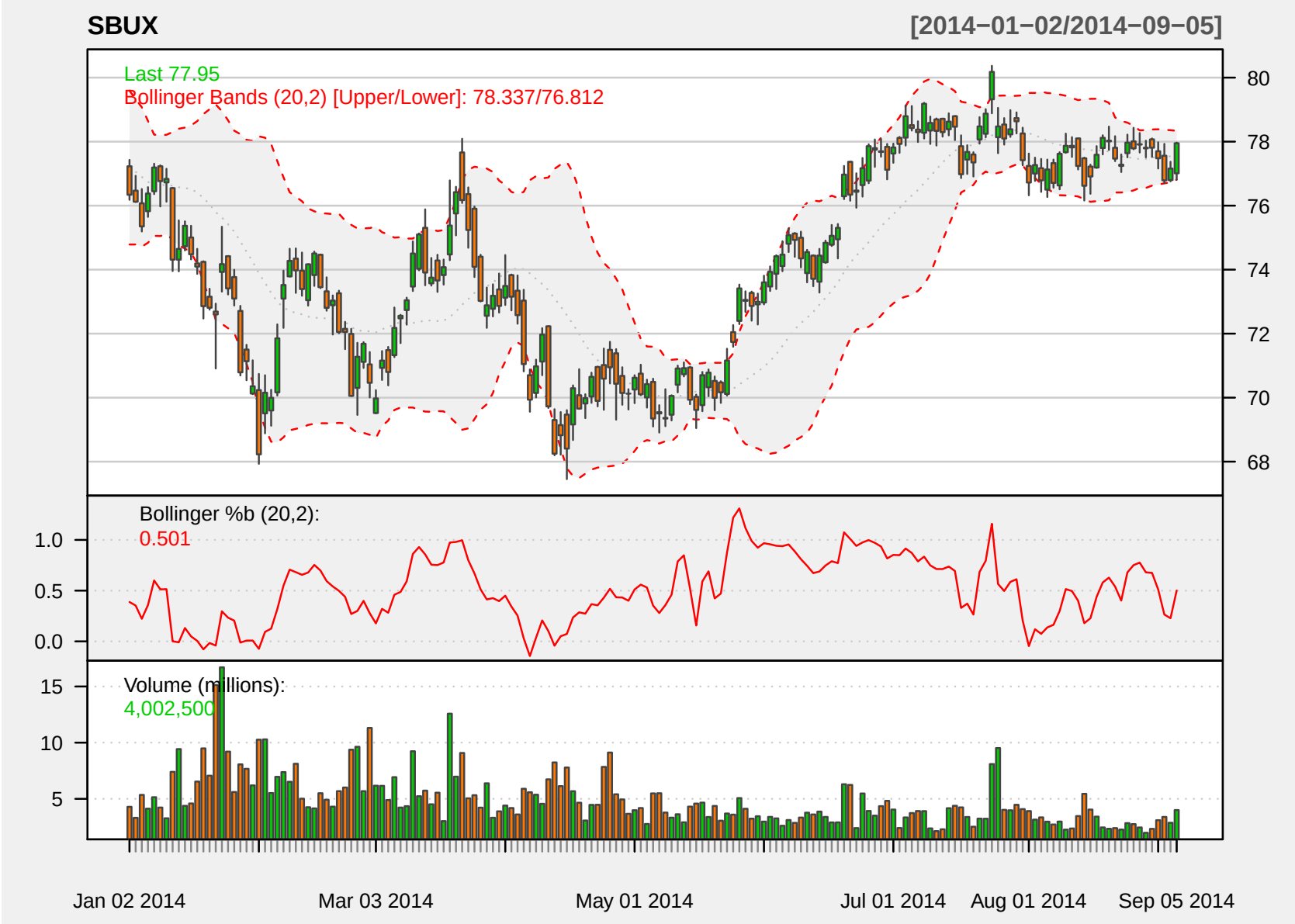
```
b <- BBands(HLC=HLC(SBUX["2014"]), n=20, sd=2)
tail(b,10)
```

```
##                   dn      mavg        up       pctB
## 2014-08-22 76.411021 77.528909 78.646797 0.40208802
## 2014-08-25 76.428539 77.511574 78.594608 0.67932318
## 2014-08-26 76.476472 77.485948 78.495424 0.75131112
## 2014-08-27 76.554083 77.453339 78.352595 0.77615115
## 2014-08-28 76.552982 77.457245 78.361509 0.68030579
## 2014-08-29 76.625643 77.503182 78.380720 0.67481749
## 2014-09-02 76.656180 77.521167 78.386153 0.51088649
## 2014-09-03 76.698147 77.533500 78.368853 0.26447097
## 2014-09-04 76.724373 77.541000 78.357627 0.22794613
## 2014-09-05 76.812251 77.574667 78.337082 0.50131162
```

```
chartSeries(SBUX,TA='addBBands();addBBands(draw="p");addVo()',
  subset='2014',theme="white")
```

$$\text{pctB} = \frac{\text{Close} - \text{LowerBand}}{\text{UpperBand} - \text{LowerBand}}$$

Bollinger bands



Moving Average Convergence-Divergence (MACD)

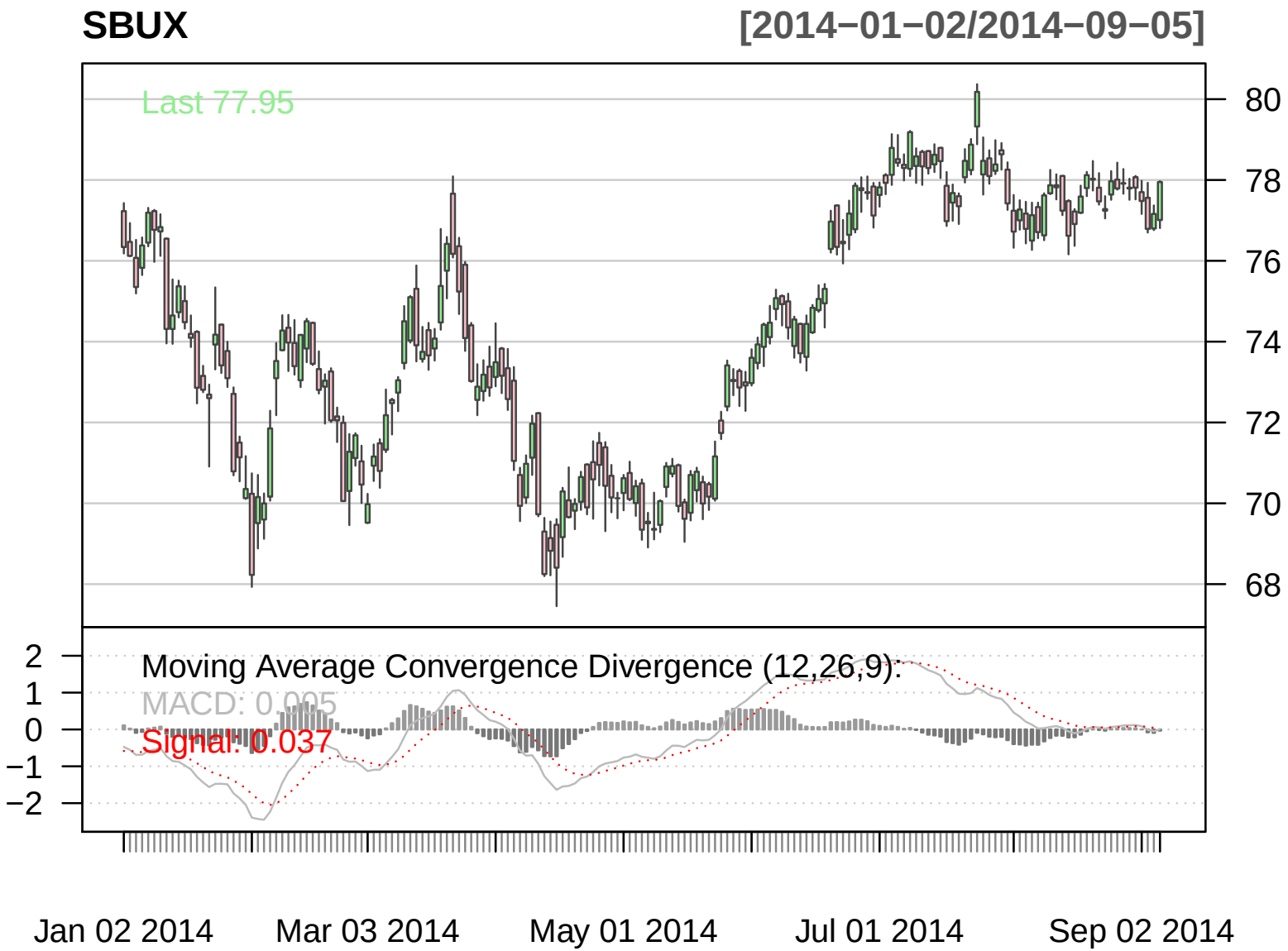
```
macd  <- MACD( Cl(SBUX), 12, 26, 9, maType="EMA" )
tail(macd)

##                    macd      signal
## 2014-08-28  0.1152221277 0.079458718
## 2014-08-29  0.1195390995 0.087474794
## 2014-09-02  0.0876697703 0.087513789
## 2014-09-03 -0.0092470283 0.068161626
## 2014-09-04 -0.0470796190 0.045113377
## 2014-09-05  0.0051187401 0.037114450

chartSeries(SBUX, TA = "addMACD()",subset="2014",theme=whiteTheme)
```

$$\text{MACD line} = \text{EMA(Close,12)} - \text{EMA(Close,26)}$$

$$\text{Signal line} = \text{EMA(MACD line,9)}$$

$$\text{MACD histogram} = \text{MACD line} - \text{Signal line}$$

Moving Average Convergence-Divergence (MACD)



Relative Strength Index (RSI)

```
rsi  <- RSI( Cl(SBUX), n = 14 )
tail(rsi)
```

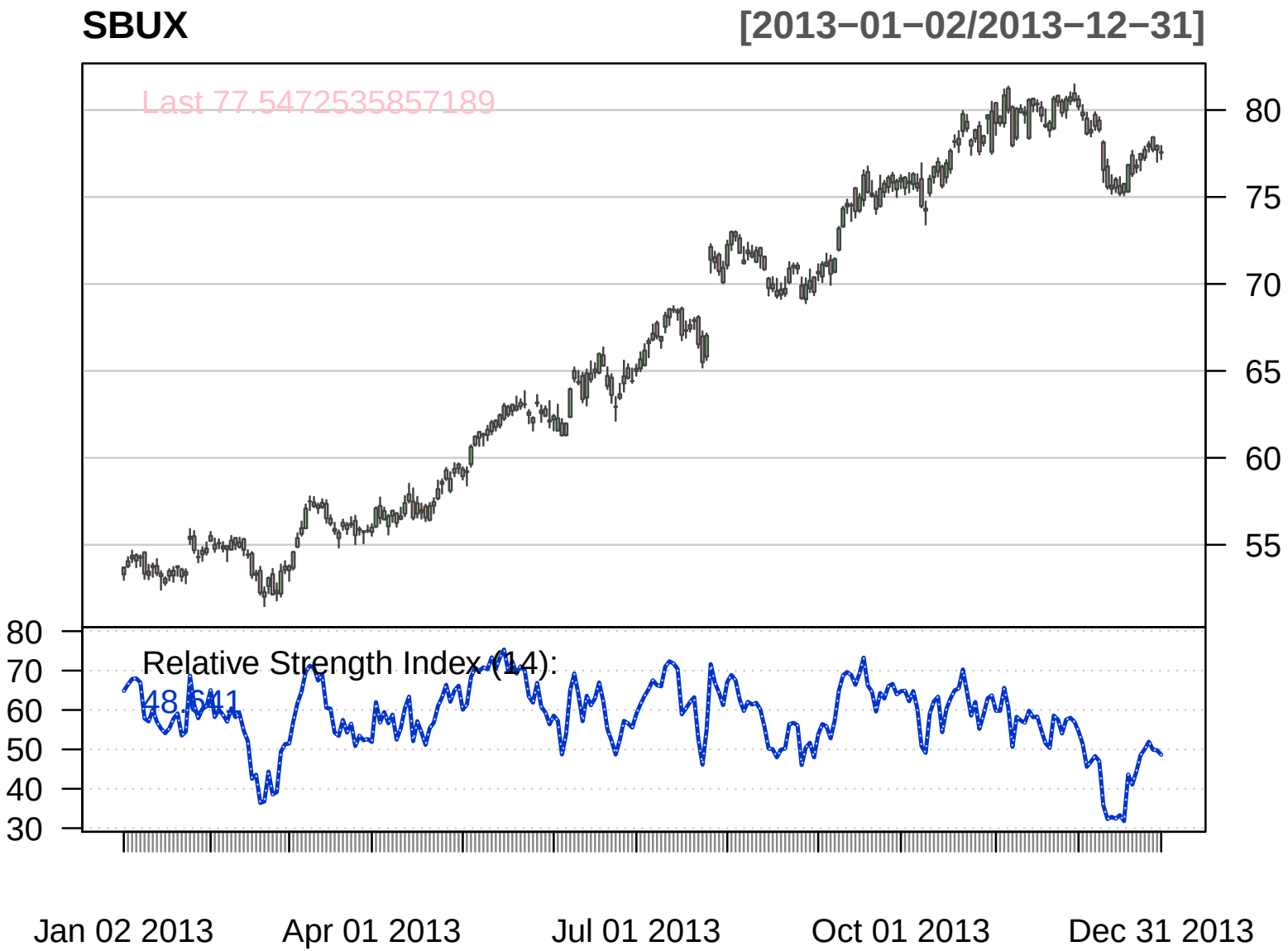
```
##            SBUX.Close.EMA.14
## 2014-08-28         52.010352
## 2014-08-29         52.010352
## 2014-09-02         48.936808
## 2014-09-03         43.189696
## 2014-09-04         46.797814
## 2014-09-05         53.577264
```

```
chartSeries(SBUX, TA = "addRSI()",subset="2013",theme=whiteTheme)
```

$$\text{RSI} = 100 - \frac{100}{1 + \text{RS}}$$

$$\text{RS} = \frac{\text{average of up changes}}{\text{average of down changes}}$$

Relative Strength Index (RSI)



Experimental chart_Series chart

The chartSeries functions are in the process of being updated; quantmod functions incorporating an underscore in their name are experimental version 2 functions:

- chart_Series
- add_TA
- chart_Theme

```
myTheme<-chart_theme()
myTheme$col$up.col<-'lightgreen'
myTheme$col$dn.col<-'pink'
#
chart_Series(SBUX["2013"],TA='add_BBands(lwd=2)',theme=myTheme,name="SBUX")
```

Experimental chart_Series chart

Outline

Working with intra-day data

- For intra-day data indexing by date alone is not enough
  - beyond the capabilities of `zoo` objects with `Date` indexes
    - intra-day bars
    - tick data
- Intra-day time series require formal time-based classes for indexing
- Recommended classes for high-frequency time series:
  - xts class
  - POSIXct/POSIXlt date-time class for indexing

Class	**Daily data**	**Intra-day data**
time series class	`xts`	`xts`
time index class	`Date`	`POSIXlt`

The strptime function

The strptime function converts character strings to POSIXlt date-time objects

Usage:

```
strptime(x, format, tz = "")
```

Main arguments:

x	vector of character strings to be converted to POSIXlt objects
format	date-time format specification
tz	timezone to use for conversion

Return value:

a POSIXlt object

The xts function

The function xts is the constructor function for extensible time-series objects

Usage:

```
xts(x = NULL, order.by = index(x), frequency = NULL,
 unique = TRUE, tzone = Sys.getenv("TZ"), ...)
```

Main arguments:

x	data matrix (or vector) with time series data
order.by	vector of unique times/dates (POSIXct is recommended)

Return value:

an xts object

Read GBPUSD 30-minute bars

```
fn1 <- "GBPUSD.txt"
dat <- read.table(file=fn1,sep=",",header=T,as.is=T)
head(dat)
```

```
##         Date Time   Open   High    Low  Close Up Down
## 1 10/20/2002 2330 1.5501 1.5501 1.5481 1.5482  0    0
## 2 10/21/2002    0 1.5481 1.5483 1.5472 1.5472  0    0
## 3 10/21/2002   30 1.5471 1.5480 1.5470 1.5478  0    0
## 4 10/21/2002  100 1.5477 1.5481 1.5471 1.5480  0    0
## 5 10/21/2002  130 1.5480 1.5501 1.5479 1.5493  0    0
## 6 10/21/2002  200 1.5492 1.5497 1.5487 1.5492  0    0
```

```
tm <- strptime(
  paste(dat[,"Date"], sprintf("%04d",dat[,"Time"])),
  format="%m/%d/%Y %H%M")
class(tm)
```

```
## [1] "POSIXlt" "POSIXt"
```

```
head(tm)
```

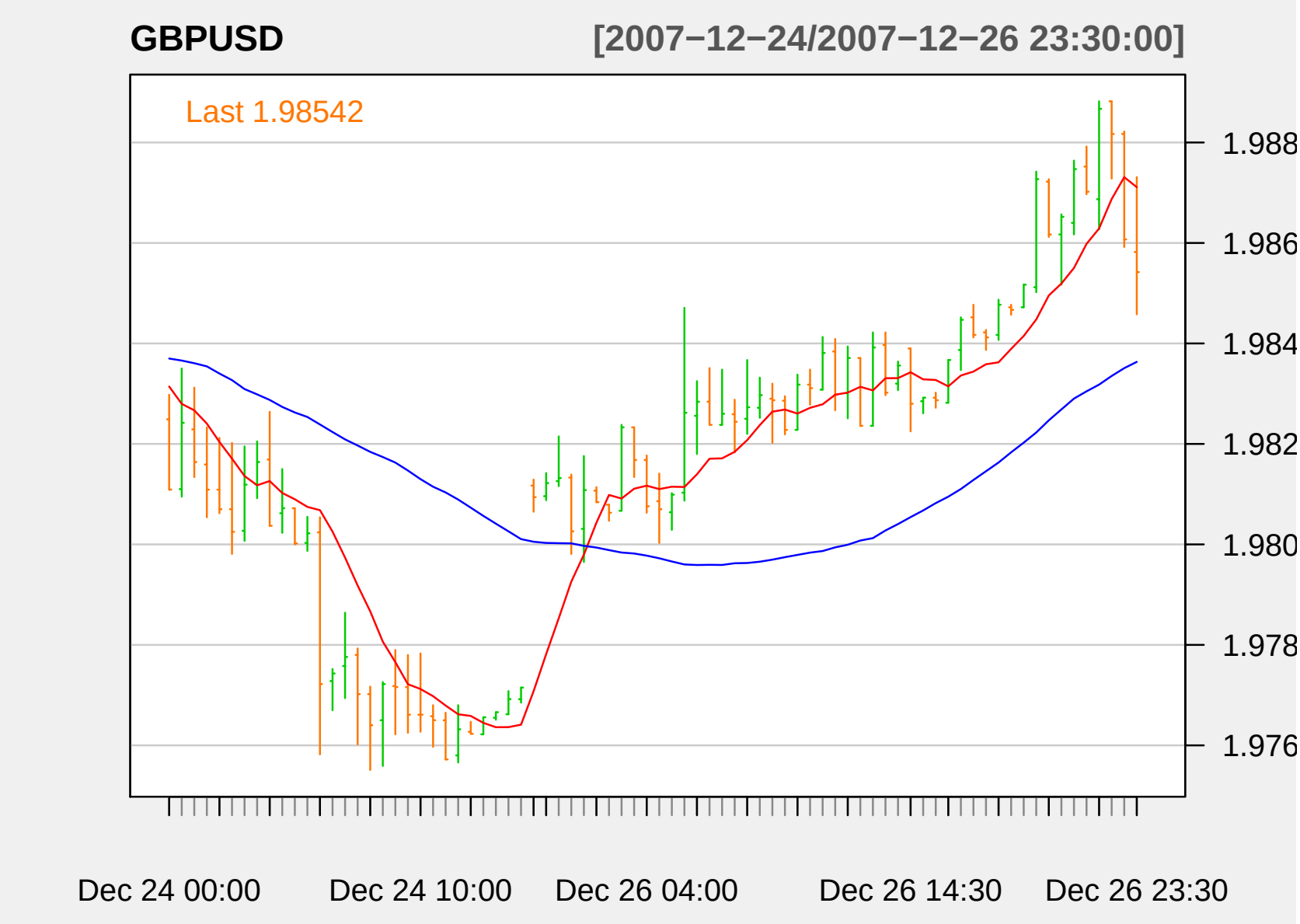
```
## [1] "2002-10-20 23:30:00 PDT" "2002-10-21 00:00:00 PDT" "2002-10-21 00:30:00 PDT"
## [4] "2002-10-21 01:00:00 PDT" "2002-10-21 01:30:00 PDT" "2002-10-21 02:00:00 PDT"
```

- Use `paste` with `sprintf` to format a Date-Time string
- Use the `format` argument of `strptime` to specify the formatting

Create and plot xts object

```
GBP <- xts(x=dat[,c("Open","High","Low","Close")],order.by=tm)
GBP <- GBP['2007']
first(GBP,'4 hours')

##                       Open   High    Low  Close
## 2007-01-01 17:30:00 1.9649 1.9650 1.9644 1.9645
## 2007-01-01 18:00:00 1.9646 1.9648 1.9641 1.9644
## 2007-01-01 18:30:00 1.9645 1.9653 1.9645 1.9650
## 2007-01-01 19:00:00 1.9651 1.9652 1.9647 1.9650
## 2007-01-01 19:30:00 1.9651 1.9658 1.9651 1.9654
## 2007-01-01 20:00:00 1.9655 1.9657 1.9650 1.9654
## 2007-01-01 20:30:00 1.9653 1.9656 1.9651 1.9655

barChart(GBP,TA='addSMA(n = 7, col = "red");addSMA(n = 44, col = "blue")',
  subset='2007-12-24/2007-12-26',theme="white",name="GBPUSD")
```

GBPUSD crossover example



GBPUSD crossover example with annotation

```
# make candle stick plot with moving averages
#
chart_Series(GBP,subset='2007-12-24/2007-12-26',theme=myTheme,name="GBPUSD",
  TA='add_SMA(n=7,col="red",lwd=2);add_SMA(n=44,col="blue",lwd=2)')
#
# find cross-over bar
#
fastMA <- SMA(Cl(GBP),n=7)
slowMA <- SMA(Cl(GBP),n=44)
co <- fastMA > slowMA
x <- which(co['2007-12-24/2007-12-26'])[1]
#
# identify cross-over bar
#
ss <- GBP['2007-12-24/2007-12-26']
add_TA(ss[x,"Low"]-0.0005,pch=17,type="p",col="red", on=1,cex=2)
#
text(x=x,y=ss[x,"Low"]-0.0005,"Crossover\nbar",pos=1)
```

GBPUSD crossover example with annotation

COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

`http://depts.washington.edu/compfin`