MQIM
R Workshop
Part 2

Casey Li

Tuesday, August 21, 2018

Two-day Schedule

Table of Content

- Introduction to R
- The R Language
- Basic Exploratory Data Analysis
- Basic Statistics
- Getting Financial Data in R
- TS (GARCH Model)
- R Objects and Functions
- Data manipulation and Visualization
- R Markdown, R Projects and Github
- Basic Machine Learning using R (KNN)
- Introduction to Python/Malab

Basic Exploratory Data Analysis

Time Series Objects

In finance, we usually work with *time series* data - data that has a specific order or time/date component. R has a variety of time series objects available to users:

- ts
- timeseries
- zoo
- xts

These are objects and associated functions that make working with ordered data (such as financial time series) much more convenient. *xts* is one that we will use extensively in this course, and it has many function for cleaning and manipulating data sets where order is important and dates are used to determine order.

xts Object Example

xts stands for Extensible Time Series - this is an extension of the zoo package.

```
> library(xts)
```

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

We'll look at an example from the xts "vignette" available at https://cran.r-project.org/web/packages/xts/vignettes/xts.pdf

```
> data(sample_matrix)
```

xts Object Example

```
> head(sample_matrix)
```

```
##                Open     High      Low    Close
## 2007-01-02 50.03978 50.11778 49.95041 50.11778
## 2007-01-03 50.23050 50.42188 50.23050 50.39767
## 2007-01-04 50.42096 50.42096 50.26414 50.33236
## 2007-01-05 50.37347 50.37347 50.22103 50.33459
## 2007-01-06 50.24433 50.24433 50.11121 50.18112
## 2007-01-07 50.13211 50.21561 49.99185 49.99185
```

xts Object Example

```
> str(sample_matrix)
```

```
##  num [1:180, 1:4] 50 50.2 50.4 50.4 50.2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:180] "2007-01-02" "2007-01-03" "2007-01-04" "2007-
##   ..$ : chr [1:4] "Open" "High" "Low" "Close"
```

str() returns a list showing the internal structure of an R object - in this case, we see that the sample_matrix xts object has a 180x4 matrix of numeric data as well as an dimnames attribute, which is a list of 2 (the rownames or dates and the column names)

xts Object Example

We won't worry too much about the power of xts so quickly, but you should be aware that having properly formatted time series data in R can often make your work much easier. Consider the problem of extracting the month-end data points from the daily data of the sample_matrix object:

```
> sample.monthly <- apply.monthly(sample_matrix,tail,1)
> head(sample.monthly)
```

```
##                Open     High      Low    Close
## 2007-01-31 50.07049 50.22578 50.07049 50.22578
## 2007-02-28 50.69435 50.77091 50.59881 50.77091
## 2007-03-31 48.95616 49.09728 48.95616 48.97490
## 2007-04-30 49.13825 49.33974 49.11500 49.33974
## 2007-05-31 47.82845 47.84044 47.73780 47.73780
## 2007-06-30 47.67468 47.94127 47.67468 47.76719
```

xts Object Example

Assuming we have proper time/date stamps, we can coerce other data types into xts objects. Let's say we had a matrix of prices in a csv file called "ts.csv" with the associated dates in the first column:

```
> ts <- read.table("data/ts.csv",header=TRUE,sep=",",as.is=TRUE)
> class(ts)
```

```
## [1] "data.frame"
```

```
> head(ts, 2)
```

```
##         Date  Open  High   Low Close Adj.Close Volume
## 1 2018-07-20 40.49 40.58 40.22 40.38     40.38 146400
## 2 2018-07-23 40.39 40.39 39.82 40.27     40.27 114100
```

xts Object Example

We can now convert this data frame into an xts object using the xts() function:

```
> ts <- read.table("data/ts.csv",header=TRUE,sep=",",as.is=TRUE)
> ts.xts <- xts(ts[,-1],order.by=as.Date(ts[,1],
+                                         format="%Y-%m-%d"))
> class(ts.xts)
```

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

```
> head(ts.xts,2)
```

```
##             Open  High   Low Close Adj.Close Volume
## 2018-07-20 40.49 40.58 40.22 40.38     40.38 146400
## 2018-07-23 40.39 40.39 39.82 40.27     40.27 114100
```

Calculating Returns

Linear Returns:

$$L_t = \frac{P_t}{P_{t-1}} - 1$$

If $\omega_1, ..., \omega_n$ are n portfolio weights of the securities in portfolio P, then

$$L_{t,P} = \omega_1 L_{t,1} + ... + \omega_n L_{t,n}$$

But:

$$\frac{P_{t+1}}{P_{t-1}} - 1 \neq L_t + L_{t+1}$$

Calculating Returns

Compounded Returns:

$$C_t = \ln \frac{P_t}{P_{t-1}}$$

If $\omega_1, ..., \omega_n$ are n portfolio weights of the securities in portfolio P, then

$$C_{t,P} \neq \omega_1 C_{t,1} + ... + \omega_n C_{t,n}$$

But:

$$\ln \frac{P_{t+1}}{P_{t-1}} = C_t + C_{t+1}$$

Calculating Returns in R

Linear Returns:

```
> ts.ret.lin <- sample_matrix[-1,]/sample_matrix[-nrow(sample_matri
```

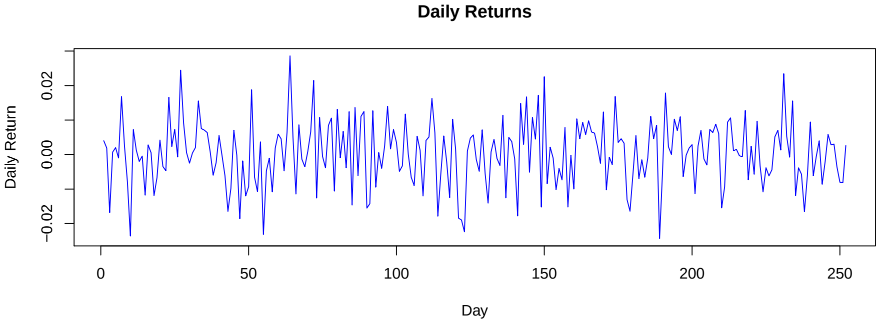
Log Returns:

```
> ts.ret.log <- diff(log(sample_matrix))
> head(ts.ret.log,2)
```

```
##                   Open          High          Low        Close
## 2007-01-03 0.003804009  6.049348e-03 0.0055915300  0.005569091
## 2007-01-04 0.003784530 -1.826194e-05 0.0006694959 -0.001296719
```

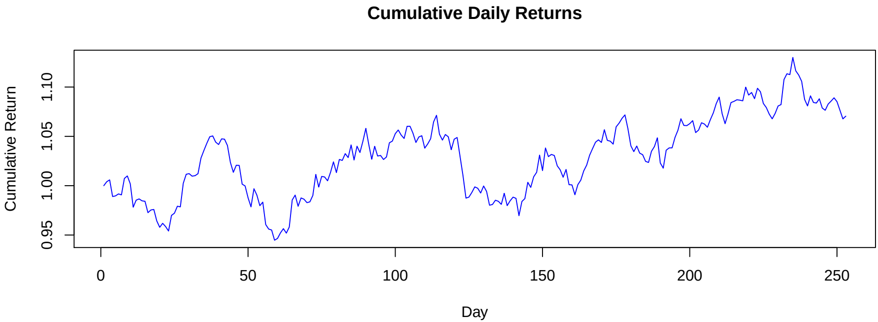
Recovering Prices from Returns

```
> my.returns <- rnorm(252,mean=0.1/252,sd=.16/sqrt(252))
> plot(my.returns,type="l",col="blue",ylab="Daily Return",
+       xlab="Day",main="Daily Returns")
```



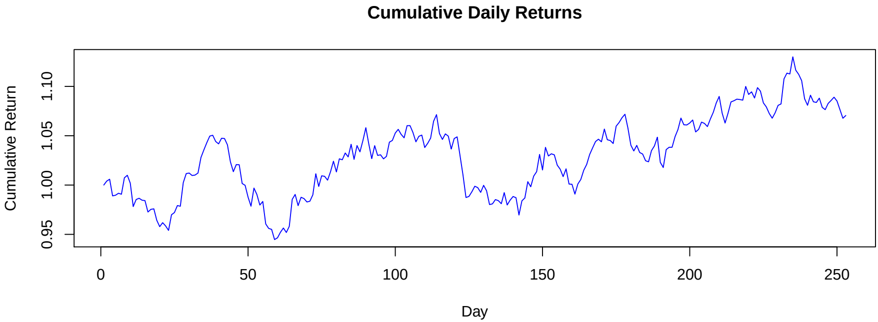
Recovering Prices from Returns

```
> cumulative.returns <- cumprod(1+my.returns)
> plot(c(1,cumulative.returns),type="l",col="blue",xlab="Day",
+       ylab="Cumulative Return",
+       main="Cumulative Daily Returns")
```



Working with Logarithmic Returns

```
> log.returns <- log(1+my.returns)
> cumulative.returns = cumsum(log.returns)
> plot(c(1,exp(cumulative.returns)),type="l",col="blue",xlab="Day",
+      ylab="Cumulative Return",
+      main="Cumulative Daily Returns")
```



Correlations

Let's investigate the correlations of various EDHEC Hedge Fund Indices using data available from the *PerformanceAnalytics* library:

```
> library(PerformanceAnalytics)
```

```
## Warning: package 'PerformanceAnalytics' was built under R versio
```

```
> data(edhec)
> names(edhec)
```

```
##  [1] "Convertible Arbitrage"  "CTA Global"
##  [3] "Distressed Securities"  "Emerging Markets"
##  [5] "Equity Market Neutral"  "Event Driven"
##  [7] "Fixed Income Arbitrage" "Global Macro"
##  [9] "Long/Short Equity"      "Merger Arbitrage"
## [11] "Relative Value"         "Short Selling"
## [13] "Funds of Funds"
```

Correlations

```
> colnames(edhec) = c("CA","CTA","DS","EM","EMN","ED","FIA",
+                     "GM","LS","MA","RV","SS","FoF")
> class(edhec)
```

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

```
> dim(edhec)
```

```
## [1] 152  13
```

Correlations

First 4 rows & first 4 columns of the correlation matrix:

```
> cor(edhec)[1:4,1:4]
```

```
##              CA         CTA          DS          EM
## CA   1.00000000 -0.06819739  0.71322614  0.55476455
## CTA -0.06819739  1.00000000 -0.08052298 -0.01046547
## DS   0.71322614 -0.08052298  1.00000000  0.80268636
## EM   0.55476455 -0.01046547  0.80268636  1.00000000
```

Basic Statistics

Summarizing Data

```
> summary(coredata(edhec[,1:3]))
```

```
##        CA                  CTA                   DS
##  Min.   :-0.123700   Min.   :-0.054300   Min.   :-0.083600
##  1st Qu.: 0.000925   1st Qu.:-0.011500   1st Qu.:-0.000175
##  Median : 0.009200   Median : 0.005250   Median : 0.009700
##  Mean   : 0.006409   Mean   : 0.006489   Mean   : 0.007953
##  3rd Qu.: 0.014600   3rd Qu.: 0.022675   3rd Qu.: 0.018225
##  Max.   : 0.061100   Max.   : 0.069100   Max.   : 0.050400
```

Basic Regression

In R, regression is easily done with the lm() function.

```
> args(lm)
```

function (formula, data, subset, weights, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...) NULL

In its simplest form, this will look like:

```
> my.regression <- lm(dep.var ~ indep.var)
```

Basic Regression

lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula` an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.

`data` an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.

`subset` an optional vector specifying a subset of observations to be used in the fitting process.

`weights` an optional vector of weights to be used in the fitting process. Should be `NULL` or a numeric vector. If non-NULL, weighted least squares is used with weights `weights` (that is, minimizing `sum(w*e^2)`); otherwise ordinary least squares is used. See also 'Details',

`na.action` a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.

`method` the method to be used; for fitting, currently only `method = "qr"` is supported; `method = "model.frame"` returns the model frame (the same as with `model = TRUE`, see below).

`model, x, y, qr` logicals. If `TRUE` the corresponding components of the fit (the model frame, the model matrix, the response, the QR decomposition) are returned.

`singular.ok` logical. If `FALSE` (the default in S but not in **R**) a singular fit is an error.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`.

`offset` this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector or matrix of extents matching those of the response. One or more `offset` terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See `model.offset`.

Figure 1: lm()

Basic Regression

Lets estimate the equity market betas of the EDHEC hedge fund indices over the sample period of the dataset. First, download the history of the S&P 500 from Dec 31 1996 to August 31 2008, store as an xts object, and convert to monthly data:

```
> library(quantmod)
> library(xts)
> getSymbols("^GSPC",src="yahoo",
+            from="1996-12-31",to="2009-08-31")
```

```
## [1] "GSPC"
```

```
> spx.dat = apply.monthly(GSPC[,6],tail,1)
> spx.ret = (exp(diff(log(spx.dat)))-1)[-1,]
> my.df = cbind(coredata(spx.ret), coredata(edhec))
> my.data.xts = xts(my.df,order.by=as.Date(index(spx.ret)))
```

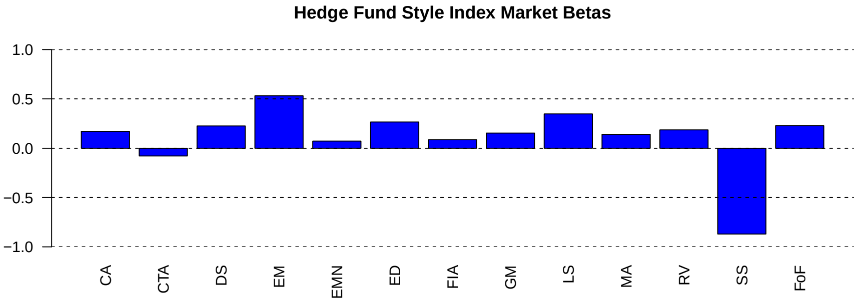
Basic Regression

Regress each EDHEC time series on the S&P 500 returns to estimate the market beta of each hedge fund style over the 1996-2008 period:

```
> betas <- rep(0,ncol(edhec))
> for (i in 1:ncol(edhec)) {
+   betas[i] = lm(my.data.xts[,(i+1)] ~
+                  my.data.xts[,1])$coef[[2]]
+ }
```

Basic Regression

```
> barplot(betas, ylim=c(-1,1),col="blue",
+         names=c("CA","CTA","DS","EM","EMN","ED","FIA",
+                      "GM","LS","MA","RV","SS","FoF"),las=2,
+         main="Hedge Fund Style Index Market Betas")
> abline(h=c(-1,-.5,0,.5,1),lty=2)
```



Getting Financial Data in R

Obtaining Financial Data in R

- Many R packages have built in functionality for downloading financial data from either free sources (Yahoo!Finance, etc.) or commercial vendors (Bloomberg or FactSet, for example).
- We'll primarily use free data sources in this class (or I will provide data sets when free sources won't cover our needs) but for those with Bloomberg access here at UNB or at work, the packages *Rbbg* and *Rblpapi* can be very useful.
- Note that Bloomberg does not suport either of these packages - they appear to be OK with users accessing the API in this way, but will of course offer absolutely no support for Rblpapi users experiencing issues. Do NOT ask the Bloomberg Help Desk for assistance with Rblpapi problems.

Free Sources

- In general, Yahoo!Finance is a pretty good source of equity pricing data, and Quandl is a good source of pricing and fundamentals (both free and non free) for equities and other securities, as well as some economic data.
- Oanda has some FX data, and Google Finance can also be used to obtain some equity pricing.
- The packages we'll use for accessing these sources are *tseries*, *quantmod*, *Quandl*, in addition to *Rblpapi* for Bloomberg data.

tseries

The *tseries* package can download data from Yahoo!Finance or Oanda:

```
> library(tseries)
```

```
## Warning: package 'tseries' was built under R version 3.4.4
```

The function *get.hist.quote()* (refer to *?get.hist.quote* for usage details) can download data into a time series object in R - lets get a few years of S&P 500 prices from Yahoo!Finance:

```
> GSPC <- get.hist.quote("^GSPC","2007-12-31","2014-12-31",
+ provider="yahoo",retclass="zoo")
```

```
> head(GSPC,2) #show first two data points
```

```
##            GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume
## 1996-12-31    753.85    753.95   740.74     740.74   399760000
## 1997-01-02    740.74    742.81   729.55     737.01   463230000
##            GSPC.Adjusted
```

quantmod

quantmod is a package for quantitative modeling of financial data and includes a variety of functions to obtain, process, and plot financial time series from multiple sources. Let's use the *getSymbols* function in the *quantmod* package to get the past 5 years of USDCAD exchange rates:

```
> library(quantmod)
> usdcad <- quantmod::getSymbols("USD/CAD", src="oanda", auto.assign
> tail(usdcad,2) # show final two data points
```

```
##             USD.CAD
## 2018-08-19 1.305886
## 2018-08-20 1.306173
```

Quandl

Quandl has become one of the most full-featured sources of free financial data on the web (www.quandl.com) and have recently started offering non-free premium data as well. Quandl maintains and R package titled *Quandl*:

```
> library(Quandl)
```

For limited usage (less than 50 calls per day), just install the package and use it. If you plan to download larger amounts of data, you will need to register for an authentication token (free) and register that in your R session.

Quandl

The official Quandl documentation has the following example to download a time series of oil prices from the NYSE and save it to an *xts* object:

```
> Quandl::Quandl.api_key("NfEvd1uysf11ToVR7dbh")
> mytimeseries <- Quandl::Quandl("FRED/GDP", type="xts")
> tail(mytimeseries,5)
```

```
##          Date    Value
## 1 2018-04-01 20402.50
## 2 2018-01-01 20041.05
## 3 2017-10-01 19831.83
## 4 2017-07-01 19588.07
## 5 2017-04-01 19359.12
```

Rblpapi

Rblpapi uses the Bloomberg API to allow easy importing of Bloomberg data into R. Load the package with the *library()* command and connect to the Bloomberg API (on a terminal computer) with the *blpConnect* command:

```
> # Not run:
> #
> # library(Rblpapi)
```

Essentially all of the API functionality (whatever you're familiar with from the Bloomberg Excel addin) should be available in Rblpapi - including *bdp*, *bds*, *bdh* functions. Start a connection to the API with

```
> # Not run:
> #
> # blpConnect()
```

Rblpapi - bdp()

Get current data points (prices, fundamentals) for one or more tickers with *bdp()*:

```
> # Not run:
> #
> # tickers <- c("RY CN Equity","TD CN Equity","CM CN Equity","BNS
> # data.items <- c("PX_LAST","DIVIDEND_YIELD","EQY_BETA")
> # bdp(tickers,data.items)
```

```
##                PX_LAST DIVIDEND_YIELD  EQY_BETA
## RY CN Equity   104.440       3.332057 1.1552070
## TD CN Equity    74.955       3.135214 1.0078220
## CM CN Equity   123.540       4.112028 0.9590632
## BNS CN Equity   82.750       3.685801 1.2313290
```

Figure 2:

Rblpapi - bdh()

Get historical data with *bdh()*:

```
> # Not run:
> #
> # historial prices for the XIU etf since Dec 31, 2010
> # my.data <- bdh("XIU CN Equity","PX_LAST",
> #                start.date=as.Date("2010-12-31"),
> #                end.date=as.Date("2015-12-29"))
> # head(my.data)
```

```
##         date PX_LAST
## 1 2010-12-31   19.29
## 2 2011-01-04   19.21
## 3 2011-01-05   19.23
## 4 2011-01-06   19.13
## 5 2011-01-07   19.06
## 6 2011-01-10   19.00
```

Rblpapi - bds()

Download bulk data with *bds()*:

```
> # Not run:
> #
> # current constituents of the s&p/tsx 60 index
> # tsx.memb <- bds("SPTSX INDEX","INDX_MEMBERS")
> # head(tsx.memb)
```

```
##   Member Ticker and Exchange Code
## 1                        AAR-U CT
## 2                          AAV CT
## 3                          ABX CT
## 4                           AC CT
## 5                        ACO/X CT
## 6                           AD CT
```

Figure 4:

Rblpapi

Other functionality exists, including the *getBars()* and *getTicks* functions, as well as a function for searching available fields (*fieldSearch*):

```
> # Not run:
> #
> # search.res <- fieldSearch("volatility")
> # head(search.res, 2)
```

```
##      Id                   Mnemonic                       Descriptio
## 1 RQ295             IVOL_MID_RT  Implied Volatility Mid (Realtime
## 2 RQ438 VOLATILITY_STRIKE_PX_RT Volatility Strike Price (Realtime
```

Figure 5:

Rblpapi

```
> # Not run:
> #
> # dim(search.res)
```

```
## [1] 55  3
```

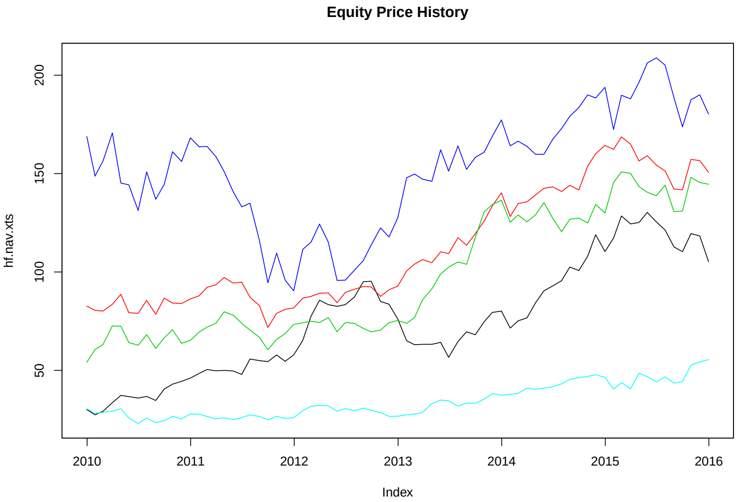
Figure 6:

Note that *fieldSearch()* returns a data frame object.

Importing Data from Files

```
> library(xts)
> hf.nav <- read.table(file = "data/stock_data.csv",
+                      header=T,sep=",",
+                      stringsAsFactors=FALSE)
> hf.nav.xts <- xts(hf.nav[,-1],
+                   order.by=as.Date(hf.nav[,1],
+                   format="%Y-%m-%d"))
> # Not run:
> #
> # plot.zoo(hf.nav.xts,
> #          plot.type='single',
> #          lty=1,
> #          main="Equity Price History")
```

Importing Data from Files



Section Summary

- R has easy and convenient methods for importing data from both free and non-free online sources
- We can also import files (and interact with databases) to import data that we already own
- Many packages are available to make this process easier - *quantmod* is one of the originals and is very full featured, while *Quandl* allows access to Quandl data sets
- Commercial vendors often provide unofficial support (Bloomberg) or official support (FactSet) for development of R packages for subscribers to access their data

Modeling Volatility Using GARCH Model

Volatility Clustering

Time series of financial asset returns often exhibit the volatility clustering property: large changes in prices tend to cluster together, resulting in persistence of the amplitudes of price changes. Some Examples:

Volatility Clustering

```
> library(Rbbg,quietly = TRUE)
> library(xts,quietly = TRUE)
> tickers = c("SPX Index","GBP Curncy",
+              "XAU Curncy", "CL1 Comdty")
> names = c("S&P 500","GBP/USD","Gold","Oil")
> con = blpConnect(verbose=FALSE)
> analysis.date = Sys.Date()
> start.date=analysis.date-3650*2
> my.data = bdh(con,tickers,"px_last",start.date)
> my.px = unstack(my.data,px_last~ticker)
> px.xts = xts(my.px,order.by=
+                 as.Date(my.data[(1:dim(my.px)[1]),2]))
> px.xts=na.omit(px.xts)
> ret=(diff(log(px.xts)))[-1,]
```

Figure 7:

Volatility Clustering

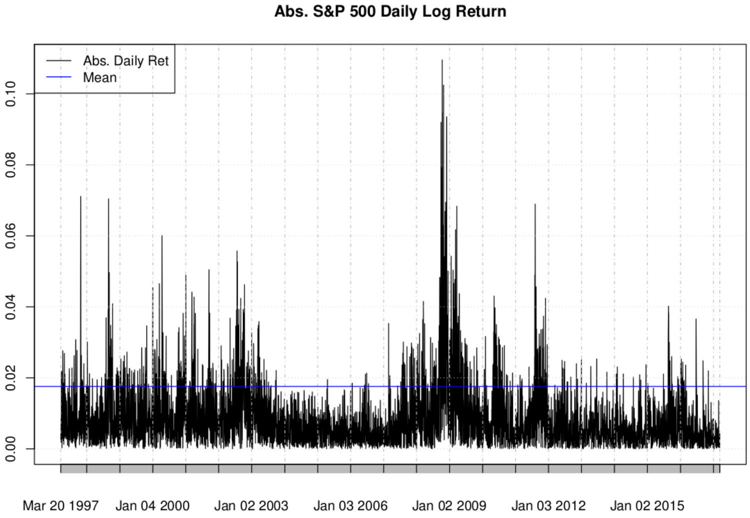


Figure 8:

Volatility Clustering

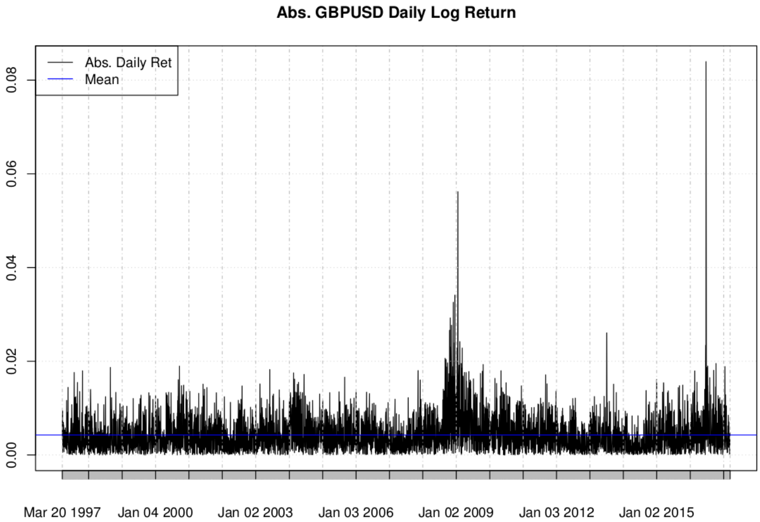


Figure 9:

Volatility Clustering

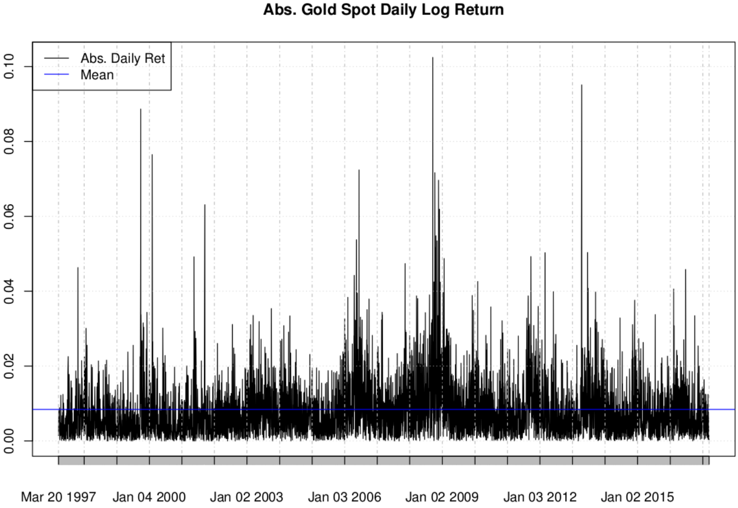


Figure 10:

Volatility Clustering

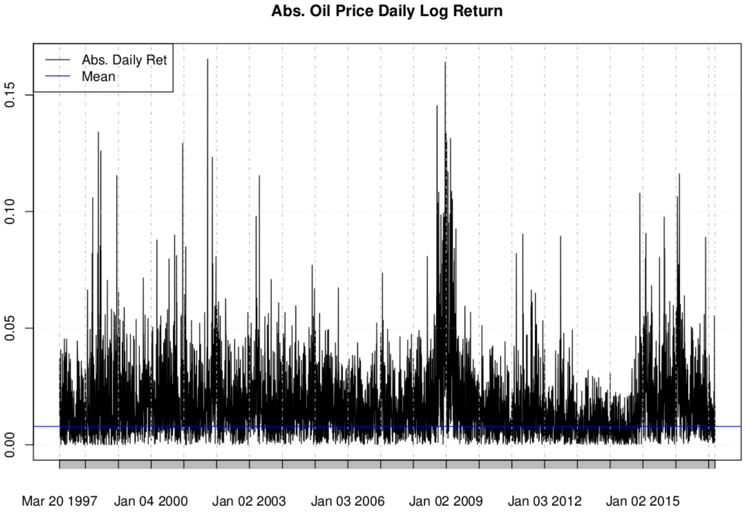


Figure 11:

GARCH Models

- Volatility is highly non-constant. Therefore, we cannot necessarily use the historical standard deviation of stock returns as an optimal forecast of future stock volatiltiy (particularly for short term forecests).
- GARCH models are well known volatility forecasting models that are. They have the properties of very rapid updating of forecasts to reflect recent data, and can often be very useful for short term volatility forecasting.

GARCH Models

GARCH (generalized autoregressive conditional heteroskedasticity) is a method of forecasting the conditional variance of a time series. We will focus on GARCH(1,1) models.

$$\sigma_t = \sqrt{\omega + \alpha_i * a_{t-1}^2 + \beta_i * \sigma_{t-1}^2} \quad (1)$$

$$a = \sigma_t * \epsilon_t \quad (2)$$

Today's volatility forecast σ_t is equal to (the square root of) some long run parameter ω plus a contribution from yesterday's return observation $\alpha_i * a_{t-1}^2$ plus a contribution from yesterday's volatility forecast $\beta_i * \sigma_{t-1}^2$.

GARCH Models

There are many packages available for estimating GARCH models in R. We'll use the *tseries* package, with its *garch()* function.

```
> # Not run:
> #
> # spx.garch = garch(ret[,3],order=c(1,1),trace=FALSE)
> # coef(spx.garch)
```

```
##           a0           a1           b1
## 1.969973e-06 9.696728e-02 8.894952e-01
```

Figure 12:

GARCH Models

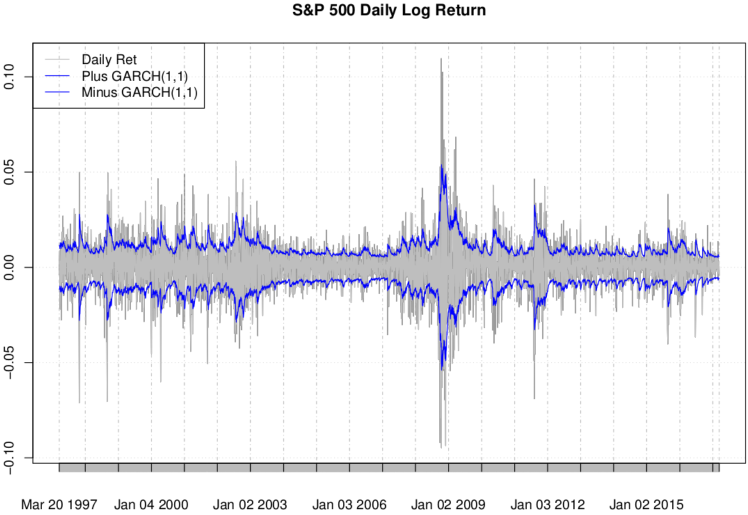


Figure 13: