MQIM R BOOTCAMP

Basic EDA & Statistics

Monday, Dec 7, 2020

Section 1

Bootcamp Week Schedule

Table of Content

Table of Content

Section 2

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 functions 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 document 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 row names 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
```

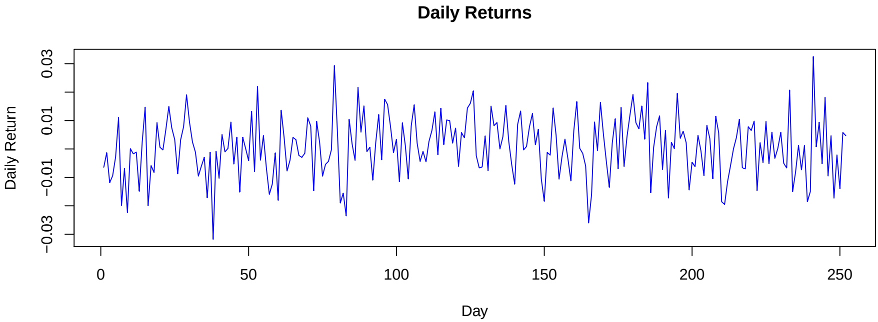
Log Returns:

```
> 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
```

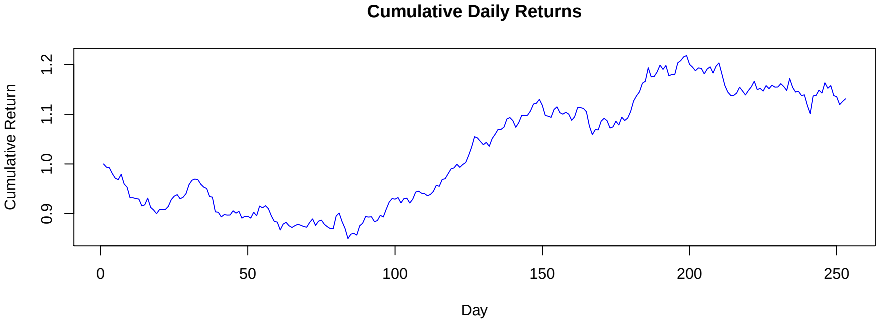
Recovering Prices from Returns

```
> 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")
```



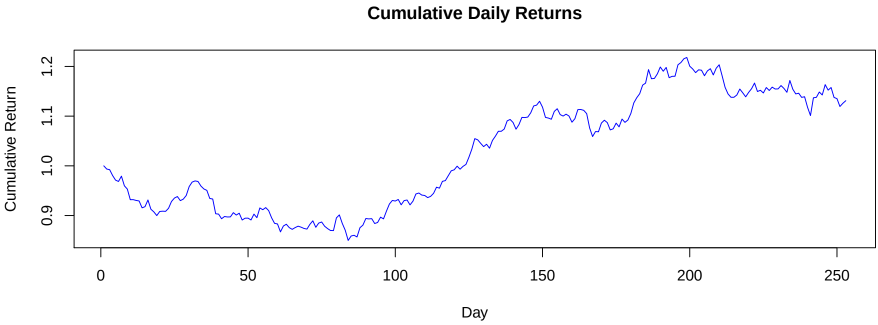
Recovering Prices from Returns

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

```
##  [1] "Convertible Arbitrage"  "CTA Global"              "Distresse
##  [4] "Emerging Markets"       "Equity Market Neutral"   "Event Dri
##  [7] "Fixed Income Arbitrage" "Global Macro"            "Long/Shor
## [10] "Merger Arbitrage"       "Relative Value"          "Short Sel
## [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] 275  13
```

Correlations

Correlation matrix:

```
> cor(edhec)
```

```
##             CA          CTA          DS          EM        EMN
## CA   1.00000000 -0.020397460  0.73056360  0.59581596  0.4945214
## CTA -0.02039746  1.000000000 -0.02066553  0.04076115  0.1979645
## DS   0.73056360 -0.020665526  1.00000000  0.77939305  0.5905784
## EM   0.59581596  0.040761152  0.77939305  1.00000000  0.5148509
## EMN  0.49452139  0.197964540  0.59057839  0.51485086  1.0000000
## ED   0.72719021  0.012591155  0.92272943  0.82251064  0.6233498
## FIA  0.77812105  0.009559439  0.65919584  0.53515775  0.4091773
## GM   0.39842200  0.567719030  0.53364027  0.65577006  0.5602918
## LS   0.60221113  0.104484319  0.77027731  0.80929883  0.6592900
## MA   0.55391291  0.032051520  0.63369670  0.60931248  0.5322269
## RV   0.85007159  0.025297119  0.84793072  0.77635211  0.6360594
## SS  -0.35079353  0.113935995 -0.56542087 -0.65724377 -0.3106873 -
## FoF  0.64603284  0.192717727  0.81442193  0.84885755  0.6886186
##             FIA          GM         LS          MA          RV
## CA   0.778121050  0.3984220  0.6022111  0.55391291  0.85007159 -(
```

Correlations

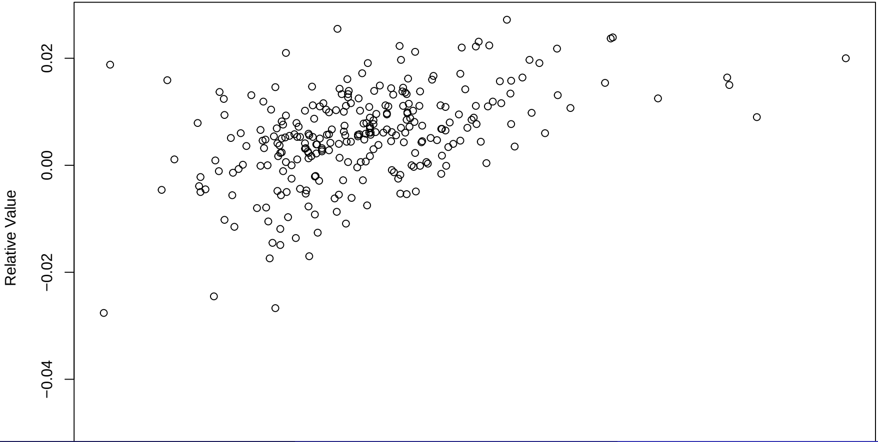
First 4 rows & first 4 columns of the correlation matrix:

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

```
##              CA         CTA          DS         EM
## CA   1.00000000 -0.02039746  0.73056360 0.59581596
## CTA -0.02039746  1.00000000 -0.02066553 0.04076115
## DS   0.73056360 -0.02066553  1.00000000 0.77939305
## EM   0.59581596  0.04076115  0.77939305 1.00000000
```

Plot Two Strategies

```
> plot(coredata(edhec[,8]),coredata(edhec[,10]),
+       xlab="Long/Short Equity",ylab="Relative Value")
```



Section 3

Basic Statistics

Summarizing Data

```r
> summary(coredata(edhec))
```

```
##        CA                  CTA                   DS              
##  Min.   :-0.12370   Min.   :-0.056800   Min.   :-0.083600   Min.  
##  1st Qu.:-0.00005   1st Qu.:-0.011900   1st Qu.:-0.002150   1st Q
##  Median : 0.00640   Median : 0.002000   Median : 0.008600   Media
##  Mean   : 0.00550   Mean   : 0.004158   Mean   : 0.006622   Mean  
##  3rd Qu.: 0.01340   3rd Qu.: 0.020250   3rd Qu.: 0.017500   3rd Q
##  Max.   : 0.06110   Max.   : 0.069100   Max.   : 0.050400   Max.  
##       EMN                  ED                   FIA          
##  Min.   :-0.058700   Min.   :-0.088600   Min.   :-0.086700  
##  1st Qu.: 0.001050   1st Qu.:-0.001450   1st Qu.: 0.001550  
##  Median : 0.004700   Median : 0.008300   Median : 0.005400  
##  Mean   : 0.004356   Mean   : 0.006216   Mean   : 0.004267  
##  3rd Qu.: 0.008100   3rd Qu.: 0.015900   3rd Qu.: 0.009250  
##  Max.   : 0.025300   Max.   : 0.044200   Max.   : 0.036500  
##        GM                  LS                   MA              
##  Min.   :-0.031300   Min.   :-0.06750   Min.   :-0.054400   Min.  
##  1st Qu.:-0.003950   1st Qu.:-0.00475   1st Qu.: 0.000600   1st Q
```

Summarizing Data

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

```
##        CA                  CTA                   DS          
##  Min.   :-0.12370   Min.   :-0.056800   Min.   :-0.083600  
##  1st Qu.:-0.00005   1st Qu.:-0.011900   1st Qu.:-0.002150  
##  Median : 0.00640   Median : 0.002000   Median : 0.008600  
##  Mean   : 0.00550   Mean   : 0.004158   Mean   : 0.006622  
##  3rd Qu.: 0.01340   3rd Qu.: 0.020250   3rd Qu.: 0.017500  
##  Max.   : 0.06110   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[1:152,]))
> 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 1996-2009 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)
```