COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

Price Data and Returns
zoo and **xts** Objects

AMATH 425 (007)

R Programming for Quantitative Finance

References/Reading/Topics

- These slides
- [JV] Ch 4, zoo and xts packages only
- Ang Ch 2, §§ 2.1-2.4 (loosely)
- Hanson, Some Applications of the xts Time Series Package (2014)
  https://blog.revolutionanalytics.com/2014/01/quantitative-finance-applications-in-r.html
- Remark: The slide presentation content is not discussed explicitly in the book

- Topics:
  - Quick intro to object oriented programming
  - The zoo and xts packages
  - Using quantmod to retrieve equity market data from Yahoo Finance
  - More properties of the xts return object
  - Calculating Returns
  - Plotting xts (time series) Objects

Object-Oriented Programming

Paul Drake arrives at LAX, 1960 (Copyright, CBS)

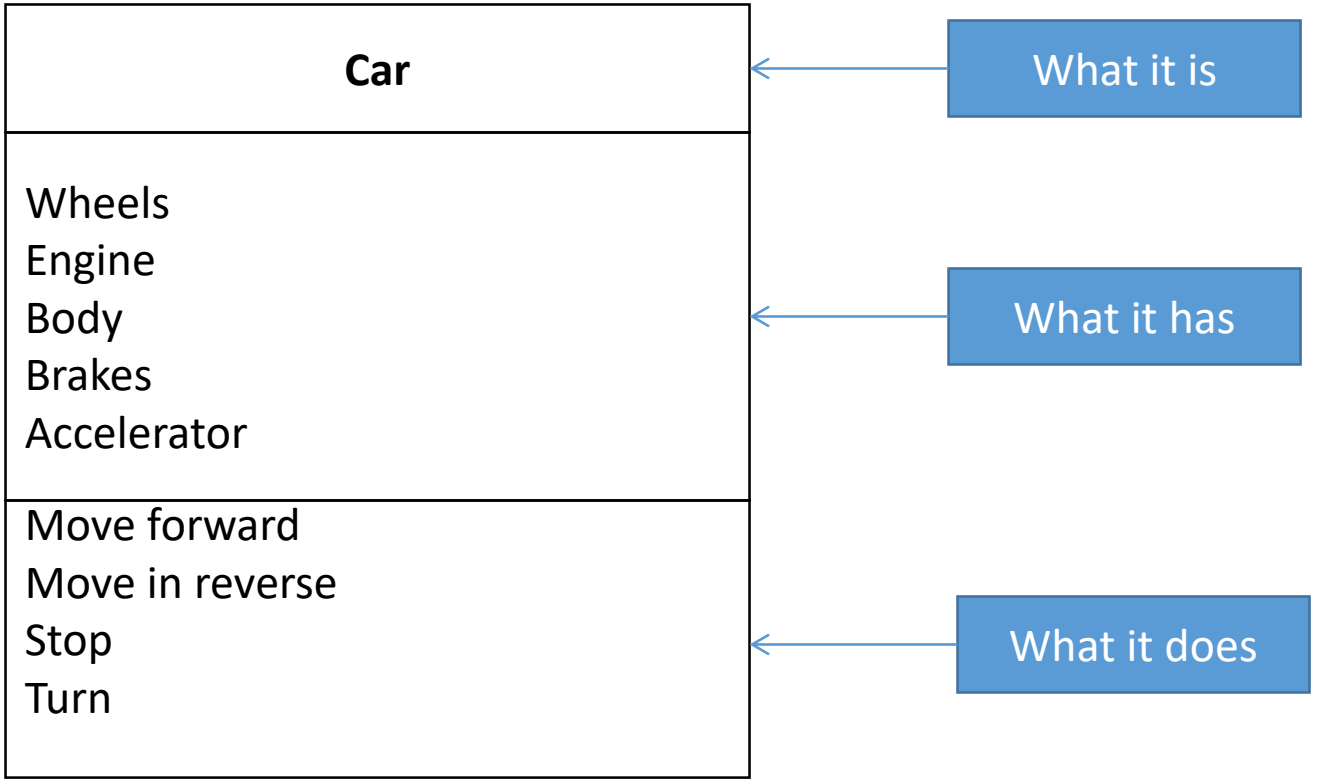
Review: What is a Class? What is an Object?

- A Car (the class)
  - Has wheels, an engine, a steering wheel, a chassis a body, brakes, and an accelerator (member variables)
  - Moves forward, stops, turns (member functions)
- An object is an instance of the class
  - A Dino Ferrari 308 GT4 is an instance of the Car class
  - A Honda Accord is an instance of the Car class

Class (UML) Diagram

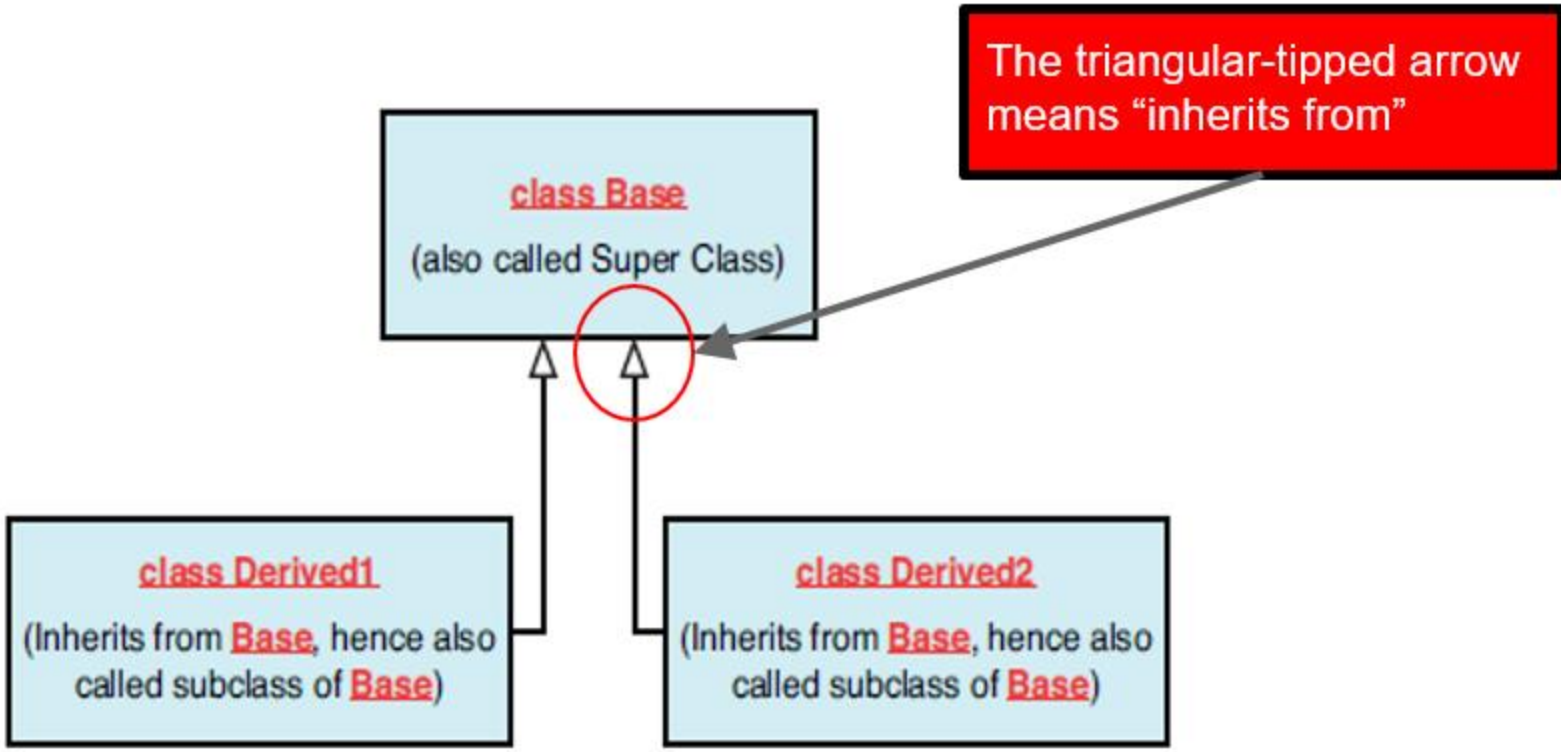
UML =

Unified Markup Language. Used heavily in object-oriented programming



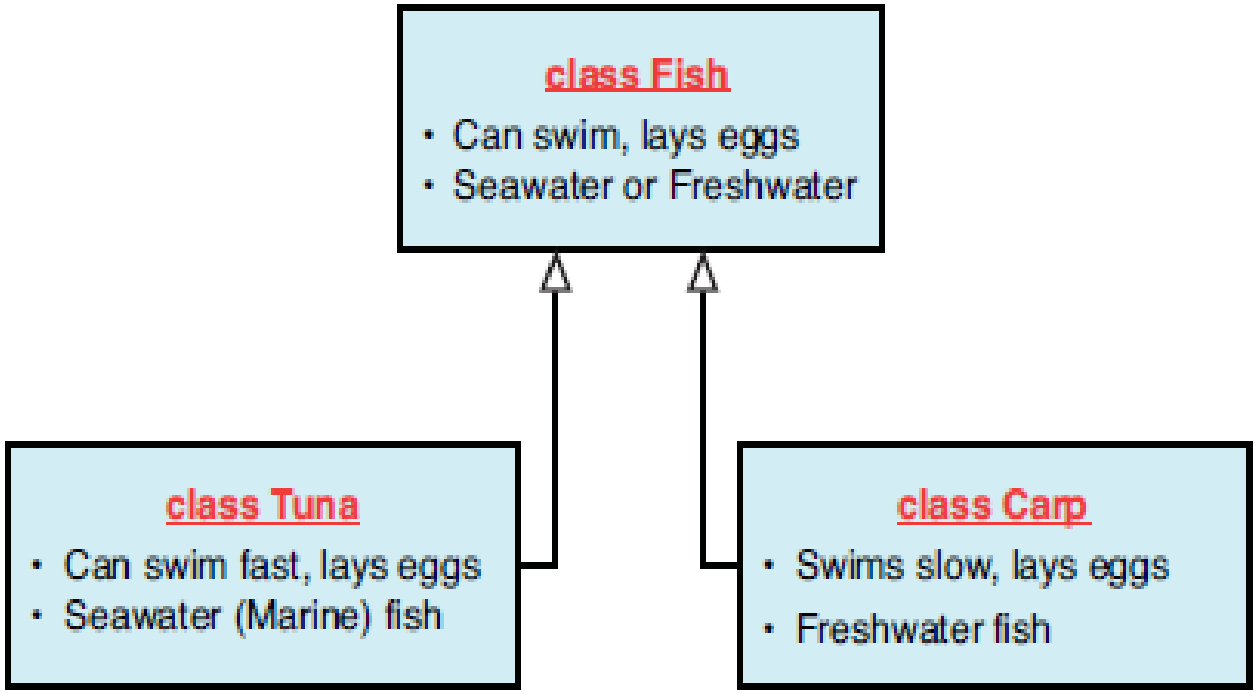
Derived classes inherit from a *base* class

FIGURE 10.1
Inheritance between classes.



Go Fish

FIGURE 10.2
Hierarchical relationship between Tuna, Carp, and Fish.



Function Overloading

- In R, rather than use member functions on a class, object often just contain member variables, and rely on *overloaded* functions
- An overloaded function is one that has the same name, but different instructions depending upon the input argument type(s) and/or object(s)
- Classic example: Two shape classes: Circle and Square

Circle
Radius r
computeArea(.)

Square
Length y
computeArea(.)

Function Overloading

- Classic case: Compute the area of a shape
- Same function name: **`computeArea(.)`**
- Different implementations depending on argument *object type*:
  - **`computeArea(squareObj)`**: return y × y
    - y is a member variable on the Square object
    - **`squareObj.y * squareObj.y`**
  - **`computeArea(circleObj)`**: return πr^2
    - r is a member variable on the Circle object
    - **`pi * circleObj.r * circleObj.r`**

So why do we care?

- The **xts** class extends from the **zoo** class
- We will primarily be concerned with **xts** objects
- However, it can sometimes be convenient to downcast from an xts to a zoo object, in particular for plotting
- The **`plot(.)`** function, in fact, is overloaded for many other classes, and in fact far more often for classes that are not related by inheritance
- Stay tuned…

zoo and **xts** Objects

Sean Connery as James Bond (Copyright United Artists)

The zoo R Package

- From the [JV] book:
  - The **ts** object (introduced earlier in the chapter) has its limitations in representing the time series
  - It cannot be used to represent the daily level stock prices as stock prices are not always equally spaced (weekends, holidays)
  - zoo is flexible and fully equipped to handle unequally spaced data, equally spaced data, and numerically indexed data.
  - Both the zoo and xts packages can be installed with the xts package alone, in the usual way, since xts depends on zoo already

- We will skip coverage of the **ts** package and move on to **zoo** and **xts**
- Recall:
  - dataframe with date column and equity data
  - Have to bind date column with subset of data
  - Can also result in date formatting side effects
- With the xts package, managing and manipulating subsets of time series data become *much easier tasks*

The zoo R Package: Example

- 1st read in the data from the .csv file provided with the data accompanying the textbook [JV]; note (again) that we'll use **read.csv(.)** rather than **read.text(.)**

```
library(zoo)

datadir <- "c:/temp/data/LearningQuantitativeFinancewithR_Code/Chapter04"

# Use read.csv(.) instead of read.table(.) as in the book;
# also, use the whole data set:
StockData <- read.csv(file.path(datadir, "DataChap4.csv"), header = TRUE)
```

- A zoo object is comprised of a date column that determines ordering, plus the data rows associated with each of the dates
- It is therefore constructed with the data in a dataframe in the columns to the right of the dates, and the Date column as the index:

```
Stockdataz <- zoo(x=cbind(StockData$Volume,StockData$Adj.Close), order.by=dt)
```

The zoo R Package: Example

- The **head(.)** function is overloaded for zoo (and xts) objects, so check the top of the dataset:

```
head(Stockdataz, 3)

2016-10-05 1877500 208.46
2016-10-06 4703400 201.00
2016-10-07 3493000 196.61
```

- No column names, so add them in next line – just like for a dataframe – and examine the first three lines again:

```
colnames(Stockdataz) <- c("Volume","Adj.Close")
head(Stockdataz, 3)    # We're cool now

            Volume Adj.Close
2016-10-05 1877500    208.46
2016-10-06 4703400    201.00
2016-10-07 3493000    196.61
```

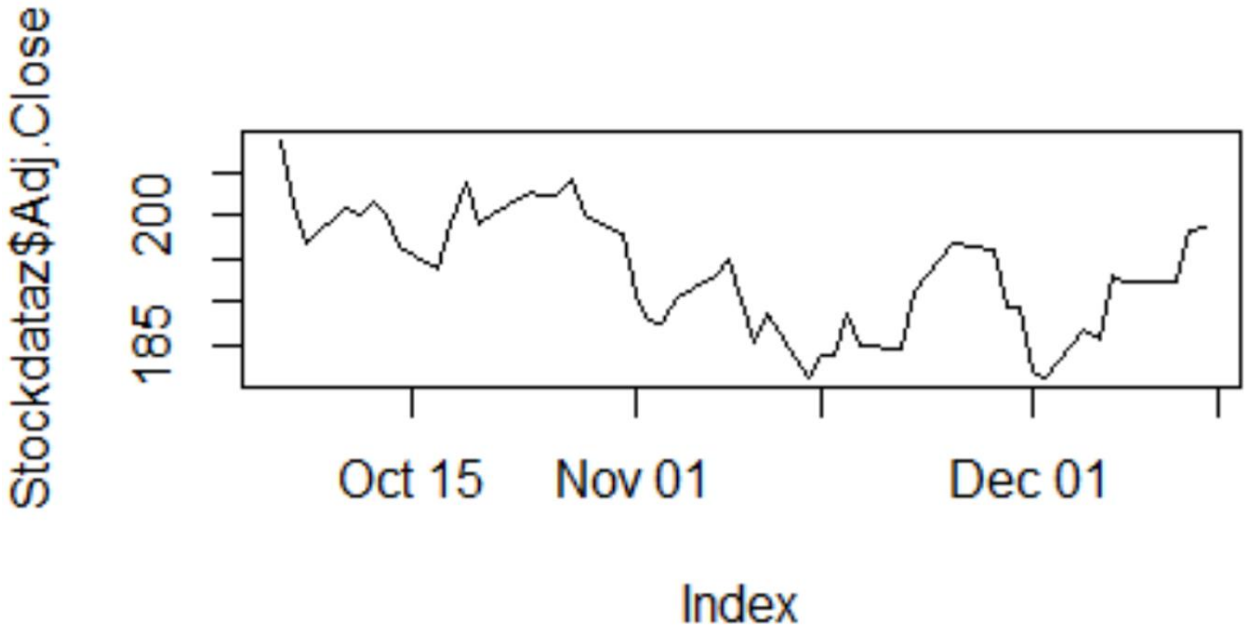
- Note that the dates are set to *yyyy-mm-dd* format, which prevents ambiguity between US and UK date formats (same for xts)

The zoo R Package: Example

- The **plot(.)** function is also overloaded for **zoo** (and **xts**) objects
- In its simplest form (as in the book example, but for a zoo object rather than a dataframe) we write

```
plot(Stockdataz$Adj.Close)
```

- And get

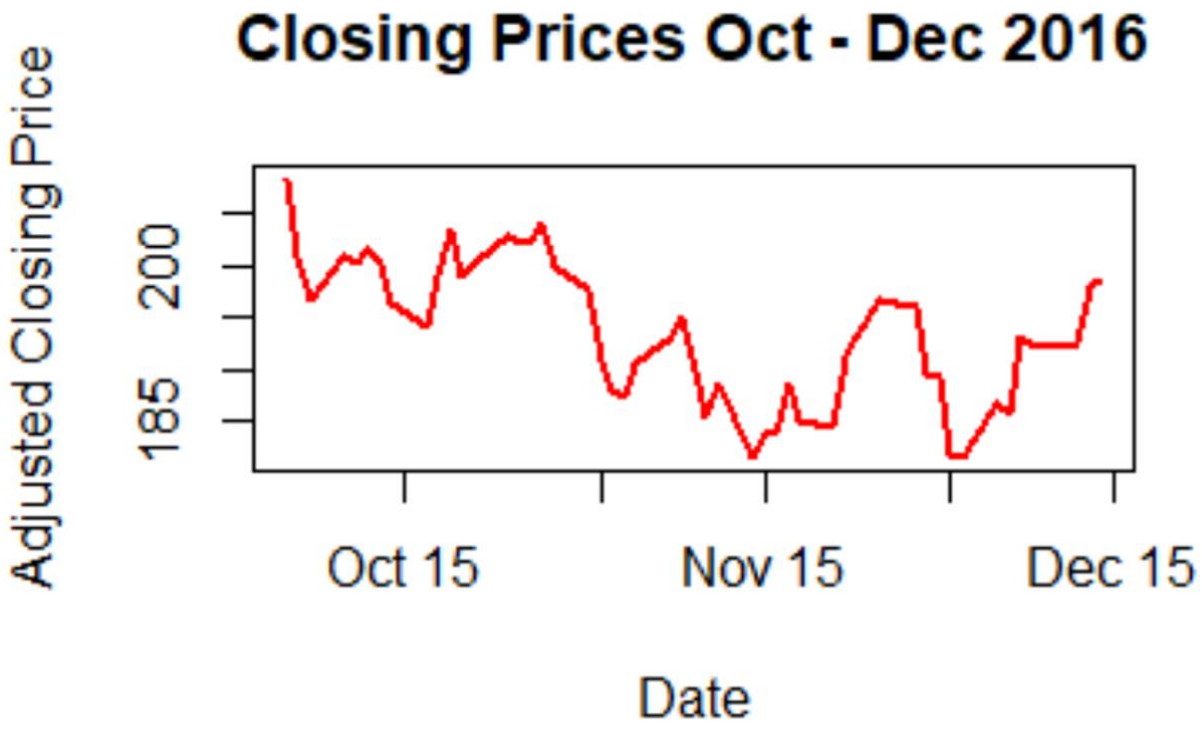


- Note that unlike the dataframe plot in the book, the zoo plot gives us a proper line plot, plus dates along the horizontal axis

The zoo R Package: Example

- We can also go a step further and use some of the same options in Base R **plot(.)** to add color, plot line thickness, axis names, and a plot title:

```
plot(Stockdataz$Adj.Close, col = "red", lwd = 2.0, xlab = "Date",
     ylab = "Adjusted Closing Price",
     main = "Closing Prices Oct - Dec 2016")
```



- Remark: We could also put **plot.zoo(.)** to be explicit, if desired

The xts R Package

- From the [JV] book (paraphrased):
  - **xts** extends a **zoo** object and carries most of its features
  - It consists of a matrix and index (like zoo) but *must* be time-based.
  - There are two ways of constructing xts objects:
    - one is by calling as.xts
    - and another is constructing the xts object from scratch.

- Other fascinating xts characteristics:
  - True time-based index – zoo index can be ordered by anything
  - May use **sapply(.)** on xts objects
  - Time-based subsetting (as we shall see)

The xts R Package

- At this point, Ch 4 of the [JV] book covers some of the theory behind time series analysis
- We will hold this in abeyance and focus more on how to *use* xts objects for managing market data
- This starts with the **quantmod** package, which provides an interface to multiple open sources for financial data
- Datasets returned by quantmod functions are xts objects

Using the quantmod R package

More properties of the xts object

quantmod

- 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
  - FRED (Federal Reserve)
  - csv, RData
- **`chartSeries(.)`** charting tool to create standard financial charts
- Author:
  - Jeffrey Ryan
  - http://www.quantmod.com/

The `getSymbols(.)` Function

- The **`getSymbols(.)`** function loads (downloads) historic price data
- Basic usage:

```
getSymbols(Symbols = vector of symbols as strings, src = "yahoo",
                return.class = 'xts',
                 index.class  = 'Date', from = "2007-01-01",
                 to = Sys.Date(), periodicity = "daily")
```

- Example:

```
getSymbols("AMZN", from = "2010-12-31", to = "2013-12-31")
```

Return from the `getSymbols(.)` Function

- The return type from **`getSymbols(.)`** per the default is an xts object
- This is very convenient for us
- Let's look at some of its properties: Note that it behaves in a manner similar to matrices and dataframes:

```
is.xts(AMZN)
head(AMZN, 3)
tail(AMZN, 3)
names(AMZN)
str(AMZN)
```

The return object has the same name as the ticker symbol (without quotes)

```
> is.xts(AMZN)
[1] TRUE
> head(AMZN, 3)
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2010-12-31    181.96     182.3   179.51     180.00     3451900        180.00
2011-01-03    181.37     186.0   181.21     184.22     5331400        184.22
2011-01-04    186.15     187.7   183.78     185.01     5031800        185.01
> tail(AMZN, 3)
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2013-12-26    401.79    404.52   396.81     404.39     1868500        404.39
2013-12-27    404.65    405.63   396.25     398.08     1986900        398.08
2013-12-30    399.41    399.92   392.45     393.37     2487100        393.37
> names(AMZN)
[1] "AMZN.Open"     "AMZN.High"     "AMZN.Low"      "AMZN.Close"
[5] "AMZN.Volume"   "AMZN.Adjusted"
```

More properties of the xts return object

- First, the **`plot(.)`** function is overloaded for xts objects
- Simple example (we will refine this later):
- Remark: the **`attach(.)`** function is *not* overloaded for xts objects

```
plot(AMZN$AMZN.Close)
```

More properties of the xts return object

- Next, there are additional functions we already know that are overloaded for xts objects:

```
dim(AMZN)                          # Number of rows and columns

summary(AMZN)                      # The usual

rows30 <- AMZN[1:30,]              # 1st 30 rows

rows30 <- rows30[-28:-29, ]        # Remove rows 28 & 29

close <- rows30[, 4]               # Get closing prices

close2 <- rows30[, "AMZN.Close"]   # Same as above, with col name
```

sapply(.) also works with xts objects

```
## sapply also works with xts objects:
sapply(AMZN, FUN = mean)
```

```
AMZN.Open      AMZN.High       AMZN.Low     AMZN.Close    AMZN.Volume AMZN.Adjusted
 237.9515       240.7883       235.0273       238.0934 4322604.5093       238.0934
```

More properties of the xts return object

- Subsetting xts objects on date ranges

```
# Extract the data for 2012 only
xts.2012 <- AMZN['2012']
# We can then then extract out just the adjusted closing data:
xts.2012 <- xts.2012[, 6]    # That's it!

# Can also do the above as a one liner:
xts.2012a <- AMZN['2012', 6]

# Remark: We will typically be more interested in the adjusted
# closing price, as this takes into account dividends and splits:
xts.2012adj <- AMZN['2012', "AMZN.Adjusted"]
# Note: putting "AMZN.Adjusted" instead of 6 is equivalent
```

More properties of the xts return object

- Extracting other periodic data (weekly, monthly, quarterly, yearly)

```
amzn.weekly <- to.weekly(AMZN)
amzn.monthly <- to.monthly(AMZN)
```

- Extracting Remark: For monthly data, the return index is in the form MMM-YYYY. We often will want to see the exact date.
- In this case, use the alternative form:

```
amzn.monthly2 <- to.period(AMZN, period = "months")
```

- Allowable periods (strings must be spelt exactly):

```
"seconds", "minutes", "hours", "days", "weeks", "months", "quarters", "years"
```

More properties of the xts return object

- Custom date subsetting

```
amzn.twoDates <- AMZN['2011-01-05/2011-02-20']

# An alternative syntax is (put '::' in place of '/'):
amzn.twoDatesAlt <- AMZN['2011-01-05::2011-02-20’]
# For specific months:
amzn.months <- AMZN['2012-12/2013-02']

# Use the '::' syntax, and only retrieve
# daily Adjusted prices:
amzn.monthsAdj <- AMZN['2012-12::2013-02', "AMZN.Adjusted"]

# Get Adjusted monthly prices between two months:
amzn.monthsAdjMths <- amzn.monthly2['2012-12/2013-02', 6]

# Adjusted prices from date to end of data period:
amzn.eop <- AMZN['2012-12::', "AMZN.Adjusted"] specific date

# Adjusted prices from beginning of data period to 2013-06-01:
amzn.bop <- AMZN['/2013-06-01', "AMZN.Adjusted"]
```

Calculating Market Returns
Plotting xts Objects

Calculating log returns

- Custom For financial modeling, quants more often than not will rely on the convenience of log returns; viz,

$$r_t = log \frac{S_t}{S_{t-1}}$$

- This is accomplished in R as follows (eg, for monthly returns):

```
amzn.mthRtns <- diff(log(amzn.monthlyAdj), lag = 1)
amzn.mthRtns <- amzn.mthRtns[-1]
```

- The return type is *also an xts object*
- We will adopt this method of calculating returns in the course, unless otherwise specified

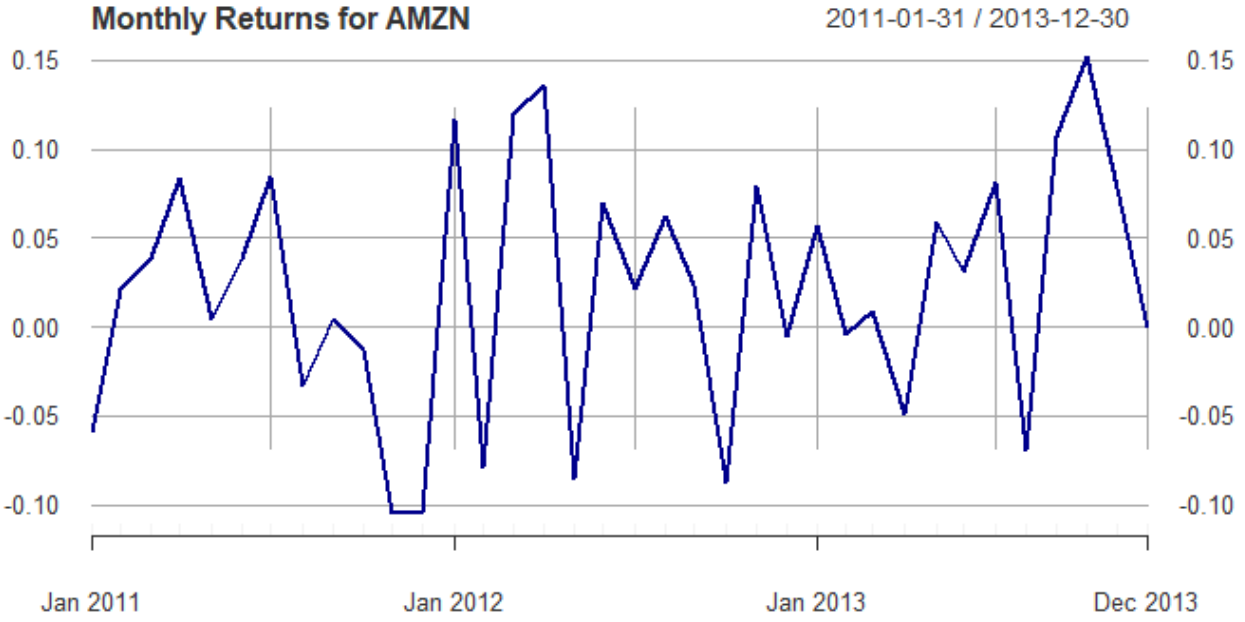
More Plotting

- Plotting of prices and returns, with some additional features to improve the presentations of the results (note some of the same argument names from the base R plot function are used in the xts overloaded version):

```
plot(amzn.adj, main = "Adjusted Closing Daily Prices for AMZN",
     col = "red", major.ticks='years')

# Plot monthly returns:
plot(amzn.mthRtns, main = "Monthly Returns for AMZN",
     col = "darkblue", major.ticks="years")
```

More Plotting



[END]