

## Chapter 2

# Individual Security Returns

In this chapter, we focus on the percentage changes in the price of a security or the security's *return*. From an investments perspective, the return of a security is sometimes a more important measure than the dollar change in the price of our investment. To see why, suppose someone told us they made \$ 500 on a 1-year investment. It is unclear how our response would be. If the initial investment was \$ 1000, making \$ 500 is exceptional as that is equal to a 50 % return in 1 year. However, if the initial investment was \$ 100,000, making \$ 500 is only equal to a 0.5 % return.

When we describe returns as the percentage change in price, we are actually not describing the whole picture. Many securities provide intermediate cash flows, such as dividends for stocks and coupons for bonds. We can reinvest those cash flows back into the security or in some other investment. Therefore, the *total return* we achieve from investing in a security must include both capital gains and the reinvestment of these intermediate cash flows. Because total return is a measure of the return we receive over the entire time we hold the security, this type of return is also known as the *holding period return*. When necessary to make the distinction, we will call the return without the cash flow yield *price return*.

You may think that getting the dividend and reinvesting that back into the stock may seem impractical. As such, you may ask, how realistic is achieving the total return for the individual investor in practice? The answer is that this is very simple to implement. In many instances, it is as simple as checking a box in your brokerage account notifying your broker that you want to reinvest dividends on a particular investment in your portfolio.

We begin this chapter by showing how to calculate price returns and total returns. Then, for statistical and programming considerations, we will show how to calculate logarithmic total returns. We then show how to cumulate daily returns into multi-day returns using both the arithmetic (i.e., non-logarithmic) returns and logarithmic returns. We then show that for dividend paying stocks, the total return over a long period of time can be much higher than the price return over the same period.

Extending our discussion of weekly and monthly prices to returns, we show how to calculate weekly and monthly returns. One common application of weekly or monthly returns is in the calculation of betas used in the cost of capital calculation, which we will demonstrate in a subsequent chapter.

Finally, we show how to compare the performance of securities using total returns. This is similar to the indexing approach we used using the closing prices in the prior chapter, but this time we perform all the calculations using the adjusted prices from Yahoo Finance.

## 2.1 Price Returns

The dollar change based solely on the closing price of a security is called *capital gains* and the percentage price in the closing price of a security is its *price return*. The price return is measured over some investment horizon (e.g., 1 day, 1 month, 1 year, etc.). The length of the period depends on the application, but the return calculations should be consistent such that cumulating all the 1-day returns in a given year should equal the annual return.

The daily price return is the percentage change in the price of a security today relative to its price yesterday. That is,

$$PRet_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1, \quad (2.1)$$

where  $PRet_t$  is the price return on day  $t$ ,  $P_t$  is the price of the security on day  $t$ , and  $P_{t-1}$  is the price of the security the trading day before.

In this section, we will use IBM stock as the example because IBM pays dividends so we can compare the results for price returns and total returns. For our example, suppose we want to calculate the daily price returns for IBM stock from 2011 to 2013.

**Step 1: Import IBM Data from Yahoo Finance** At the start of each chapter, we should load the `quantmod` and `xts` packages. Then, using techniques we discussed in Chap. 1, we read-in the **IBM Yahoo.csv** file. Recall that this file contains IBM data from December 31, 2010 to December 31, 2013.

```
> library(quantmod)
> library(xts)
> data.IBM<-read.csv("IBM Yahoo.csv",header=TRUE)
> date<-as.Date(data.IBM$Date,format="%Y-%m-%d")
> data.IBM<-cbind(date, data.IBM[,-1])
> data.IBM<-data.IBM[order(data.IBM$date),]
> data.IBM<-xts(data.IBM[,2:7],order.by=data.IBM[,1])
> names(data.IBM)<-
+   paste(c("IBM.Open","IBM.High","IBM.Low",
+   "IBM.Close","IBM.Volume","IBM.Adjusted"))
> data.IBM[c(1:3,nrow(data.IBM)),]
```

	IBM.Open	IBM.High	IBM.Low	IBM.Close	IBM.Volume	IBM.Adjusted
2010-12-31	146.73	147.07	145.96	146.76	2969800	139.23
2011-01-03	147.21	148.20	147.14	147.48	4603800	139.91
2011-01-04	147.56	148.22	146.64	147.64	5060100	140.06
2013-12-31	186.49	187.79	186.30	187.57	3619700	187.57

**Step 2: Subset the Data to Only Include the Closing Price** The price return is calculated off the stock's closing price. In `data.IBM`, this is `IBM.Close` or Column 4. We then subset the data to only include the IBM closing price.

```
> IBM.prc.ret<-data.IBM[,4]
> IBM.prc.ret[c(1:3,nrow(IBM.prc.ret)),]
      IBM.Close
2010-12-31    146.76
2011-01-03    147.48
2011-01-04    147.64
2013-12-31    187.57
```

**Step 3: Calculate IBM's Price Return** We can apply Eq. (2.1) to the IBM closing price to calculate the price return. This equation is implemented in R using the `Delt` command. For example, the price return on January 3, 2011 is equal to 0.49 %  $[(147.48/146.76) - 1]$ .

```
> IBM.prc.ret$IBM.prc.ret<-Delt(IBM.prc.ret$IBM.Close)
> IBM.prc.ret[c(1:3,nrow(IBM.prc.ret)),]
      IBM.Close IBM.prc.ret
2010-12-31    146.76      NA
2011-01-03    147.48 0.004905969
2011-01-04    147.64 0.001084893
2013-12-31    187.57 0.006222842
```

**Step 4: Clean up Data Object** The output above shows that December 31, 2010 has a return of NA and the first return in the series is on January 3, 2011, which is the first trading day of 2011. As such, we can delete the first observation in `IBM.prc.ret`. In addition, since we will be dealing with returns, we do not need `IBM.Close` and the first column can also be deleted.

```
> options(digits=3)
> IBM.prc.ret<-IBM.prc.ret[-1,2]
> IBM.prc.ret[c(1:3,nrow(IBM.prc.ret)),]
      IBM.prc.ret
2011-01-03    0.00491
2011-01-04    0.00108
2011-01-05   -0.00400
2013-12-31    0.00622
> options(digits=7)
```

### A Note on Stock Splits and the Use of Closing Prices from Yahoo Finance

Note that the price return calculation above blindly calculates a return off the closing price of the company's security. The closing price of a company's security is affected by stock splits and there are no adjustments to the closing price that reflects this. For example, Colgate-Palmolive declared a 2-for-1 stock split on March 7, 2013, which was payable on May 15, 2013. This means that

on May 16, 2013, each shareholder of Colgate-Palmolive would get two shares for every one share they own but the price of each share is cut in half. This means that the value of the investors' holdings in Colgate-Palmolive is unchanged based purely on the act of splitting the stock. Colgate-Palmolive's closing price on May 15, 2013, the day before the split, was \$ 124.55 and the closing price on May 16, 2013 was \$ 62.38. Using the closing price to calculate the price return would yield a *negative* 49.9 % return. However, the correct price return is a *positive* 0.10 %, which is calculated by dividing \$ 124.76 [= \$ 62.39 \* 2] by \$ 124.55 then subtracting one from the result.

As such, looking at the closing price alone may not be sufficient to properly make inferences from the data. When we see large returns when using the close price, we have to investigate what caused the stock price to move by that much. It is certainly possible for stocks to move by a large amount, but it is also possible that such large price movements are caused by stock splits or reverse stock splits and investigation is warranted.

## 2.2 Total Returns

The return to investors from holding shares of stock are not limited to changes in the price of the security. For companies that pay dividends, the shareholders holding shares prior to the ex-dividend date receive cash payments that they are able to reinvest. In fact, automatically reinvesting dividends may be an option for some securities. Therefore, the total return a shareholder can receive from a stock that pays dividends includes both the change in the price of the shares she owns as well as any income generated from the dividends and the reinvestment of those dividends on the ex-date. This is known as the *holding period return* or *total return*.

The total return from holding a security is calculated as follows:

$$R_t = \frac{P_t + CF_t + P_{t-1}}{P_{t-1}} = \underbrace{\left[ \frac{P_t}{P_{t-1}} - 1 \right]}_{\text{Capital Appreciation}} + \underbrace{\frac{CF_t}{P_{t-1}}}_{\text{CF Yield}}, \quad (2.2)$$

where  $CF_t$  is the cash flow (e.g., dividend) payment on day  $t$  and all other terms are defined the same way as Eq. (2.1). The first term represents the capital appreciation while the second term represents the dividend yield. The decomposition in Eq. (2.2) can also help identify the source of the return, such that investors who prefer capital gains or dividends (e.g., for tax purposes) can make a better assessment of the attractiveness of the investment for their particular investment objective.

For a daily total return calculation, the dividend yield is zero on non-ex dividend dates. This fact has two implications. First, on most days, the price return and the total return are the same because we only have the changes in the capital appreciation

on those dates. Second, for a non-dividend paying stock, the price return and total return across time is the same.

Yahoo Finance data provides us with an adjusted closing price variable. In `data.IBM` this is the sixth column labeled `IBM.Adjusted`. Not only does the adjusted closing price incorporate adjustments for dividend payments, it also incorporates adjustments for stock splits. As such, the problem above we discussed with using the closing price variable for Colgate-Palmolive would not have been present had we used the adjusted close price. The adjusted closing price variable makes these adjustments retroactively, such that the closing price and adjusted closing price after the last stock split or dividend payment are the same. The difference can be observed when looking at the pre-split or pre-dividend closing and adjusting closing prices.

### Why Bother with the Closing Price?

The use of the closing price seems more trouble than it is worth, so why bother dealing with the closing price? There are at least two reasons why we need the closing price. First, if we are interested in knowing the capital appreciation or price return, such as for the decomposition of total returns between capital appreciation and dividend yield, we need to be able to calculate the correct price return. Second, if we were to use any sort of calculation based off the closing price, the adjusted close price may not reflect the appropriate price that is compatible with those securities. For example, option prices have strike prices that are compared to the closing price. Therefore, any analyses that requires the use of historical close prices that cross an ex-dividend date would have an adjusted closing price that is not consistent with the option strike prices.

Continuing from our IBM example from the previous section, we now calculate IBM's total returns from January 2011 to December 2013. As of the writing of this book, the last time IBM declared dividends was on October 29, 2013, which was payable on December 10, 2013. However, the important dividend date for calculating returns is the *ex-dividend date* or the *ex-date*, which was on November 6, 2013. The *ex-date* means that IBM shares will be trading without the dividend beginning on November 6, 2013, such that those who purchase IBM shares on November 6, 2013 and after would not be entitled to the dividend declared on October 29, 2013. The output below shows that the closing price and adjusted closing price for IBM on or after November 6, 2013 are the same, but those two variables have different values prior to November 6, 2013.

```
> data.IBM[715:720,]
      IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adjusted
2013-11-01  179.81  180.34  178.88   179.23   3644500    178.27
2013-11-04  179.90  180.80  179.34   180.27   3483300    179.31
2013-11-05  179.54  179.80  177.71   177.85   6096800    176.90
2013-11-06  177.91  179.75  177.78   179.19   4560700    179.19
2013-11-07  179.60  181.39  179.60   180.00   5219500    180.00
2013-11-08  178.83  180.08  177.35   179.99   6275000    179.99
```

Since anyone who purchased shares on November 6, 2013 would not be entitled to the dividend, this implicitly means that holders of those shares would not be entitled to a portion of the cash (assets) of the firm that will be used to pay the dividends. Because the value of the assets of the firm that the investor is buying on November 6, 2013 is lower by the amount of dividend per share (i.e., cash of \$ 1 has a market value of \$ 1), we should observe a corresponding drop in the equity value of IBM. This reduction in equity value of IBM will translate into a drop in the IBM stock price holding all else constant.

Comparing the close price and adjusted close price on November 5, 2013, we see that the close price is higher than the adjusted close price. This implies that the price return would be smaller than the total return, which is what we would expect given that the total return is equal to the price return plus the dividend yield.

We now turn to calculating IBM's daily total return.

**Step 1: Import Adjusted Closing Price Data** Since we previously called-in the IBM data, we only need to check that it is still in the R memory. If so, we then keep the adjusted close price or Column 6 in `data.IBM`.

```
> IBM.ret<-data.IBM[, 6]
> IBM.ret[c(1:3,nrow(IBM.ret)),]
      IBM.Adjusted
2010-12-31      139.23
2011-01-03      139.91
2011-01-04      140.06
2013-12-31      187.57
```

**Step 2: Calculate Total Return** We apply the `Delt` command on the adjusted close price to calculate the total return. For example, the total return for January 3, 2011 is equal to 0.49 % [= (139.91/139.23) – 1]. Note that due to rounding of the adjusted close price, it would appear that the January 3, 2011 total return is lower than the price return calculated in the prior section. However, logically, the total return is at least as large as the price return because total return equals price return on non ex-dividend days and the total return is higher than the price return on ex-dividend days. We will see this distinction graphically later in this section.

```
> IBM.ret$IBM.tot.ret=Delt(IBM.ret$IBM.Adjusted)
> IBM.ret[c(1:3,nrow(IBM.ret)),]
      IBM.Adjusted IBM.tot.ret
2010-12-31      139.23      NA
2011-01-03      139.91 0.004884005
2011-01-04      140.06 0.001072118
2013-12-31      187.57 0.006222842
```

**Step 3: Clean up the Data** To output the data, we only show the total return column and limit the number of decimals on display using the `digits=3` option. We do not delete the December 31, 2010 value here because we will use this data object in a later section.

```

> options(digits=3)
> IBM.log.ret<-IBM.log.ret[,2]
> IBM.log.ret[c(1:3,nrow(IBM.log.ret)),]
      IBM.log.ret
2010-12-31      NA
2011-01-03    0.00487
2011-01-04    0.00107
2013-12-31    0.00620
> options(digits=7)

```

### Note on Potential Discrepancy in Total Return Calculations From Later Downloads of IBM Data from Yahoo Finance

The adjusted close price for prior periods changes every time a stock split or dividend is paid. As such, the adjusted close price values we observe from a later data pull of IBM data may be different from what we report in this text. In addition, the adjusted closing price reported on Yahoo Finance only has two decimals. All these may result in slight variations in the total return calculation. However, the normal variations in calculated returns based on modifications to the adjusted close prices should not be orders of magnitude different from what we have calculated above.

## 2.3 Logarithmic Total Returns

The returns calculated in the preceding section using Eq. (2.2) and the `Delta` command are called *arithmetic returns* or simple returns. In this section, we show how to calculate *logarithmic returns* or log returns. Logarithmic returns are used extensively in derivatives pricing, among other areas of finance. In addition, when we calculate multi-period returns later in this chapter, we show that calculating cumulative returns is relatively easier using logarithmic returns.

The logarithmic return,  $r_t$ , is calculated as

$$r_t = \ln \left( \frac{P_t}{P_{t-1}} \right) = \ln(1 + R_t) = \ln P_t - \ln P_{t-1}, \quad (2.3)$$

where  $\ln$  is the natural logarithm operator and the rest of the variables are defined the same as in Eq. (2.2). Therefore, we can take the *difference* of the *log* prices to calculate *log* returns.

We now calculate the logarithmic total return for IBM stock from January 2011 to December 2013.

**Step 1: Import Adjusted Closing Price Data** Since we are calculating logarithmic total returns, we still need the adjusted closing price data as our base data source.

```

> IBM.ret<-data.IBM[,6]
> IBM.ret[c(1:3,nrow(IBM.ret)),]
      IBM.Adjusted
2010-12-31      139.23
2011-01-03      139.91
2011-01-04      140.06
2013-12-31      187.57

```

**Step 2: Calculate Log Returns** We use a combination of the `diff` and `log` commands to calculate logarithmic returns. This corresponds to the right-most definition of logarithmic returns in Eq. (2.3). That is, the log returns are calculated as the differences (`diff`) between the natural logarithm (`log`) of the adjusted closing prices.

```

> IBM.log.ret$IBM.log.ret<-diff(log(IBM.log.ret$IBM.Adjusted))
> IBM.log.ret[c(1:3,nrow(IBM.log.ret)),]
      IBM.Adjusted IBM.log.ret
2010-12-31      139.23      NA
2011-01-03      139.91 0.004872117
2011-01-04      140.06 0.001071543
2013-12-31      187.57 0.006203560

```

**Step 3: Clean up the Data** We then clean up `AMZN.log.ret` to delete the first column and keep only the logarithmic return series.

```

> options(digits=3)
> IBM.log.ret<-IBM.log.ret[,2]
> IBM.log.ret[c(1:3,nrow(IBM.log.ret)),]
      IBM.log.ret
2010-12-31      NA
2011-01-03    0.00487
2011-01-04    0.00107
2013-12-31    0.00620
> options(digits=7)

```

**Compare Log Returns with Arithmetic Returns** Below we show how to combine the two total return calculations using the `cbind` command. We see that the differences on each day are fairly small and hard to spot by visual inspection. The output below shows that, on an absolute value basis, the largest difference between the two return measures is 0.36 % and the smallest difference is virtually zero. Note that we used the `na.rm=TRUE` option so that R still calculates a `max` and a `min` even though there is an NA in the data (i.e., the December 31, 2010 value is an NA and would cause an error in R without the `na.rm=TRUE` option).



```

> options(digits=3,scipen=100)
> tot.rets<-cbind(IBM.ret,IBM.log.ret)
> tot.rets[c(1:3,nrow(tot.rets)),]
      IBM.tot.ret IBM.log.ret
2010-12-31      NA      NA
2011-01-03    0.00488    0.00487
2011-01-04    0.00107    0.00107
2013-12-31    0.00622    0.00620
> max(abs(tot.rets$IBM.tot.ret-tot.rets$IBM.log.ret),na.rm=TRUE)
[1] 0.00363
> min(abs(tot.rets$IBM.tot.ret-tot.rets$IBM.log.ret),na.rm=TRUE)
[1] 0.00000000126
> options(digits=7,scipen=0)

```

Note we used the `scipen=100` option to increase the threshold before R converts the output into scientific notation. This allows us to read the minimum difference above in decimals rather than having to interpret the results in scientific notation, which may be harder to understand. After we are done, we revert the `scipen` option back to zero so that the output of subsequent analyses will revert back to the default display options.

## 2.4 Cumulating Multi-Day Returns

When evaluating investments, we are typically concerned with how our investment has performed over a particular time horizon. Put differently, we are interested in cumulative *multi-day returns*. We could be interested in knowing the returns of our investment over the past week or over the past month or over the past year. To fully capture the effects of being able to reinvest dividends, we should calculate daily returns and string those returns together for longer periods. Otherwise, if we simply apply Eq. (2.2) using prices at the beginning and end of the investment horizon and add the dividend, we are assuming that the dividends are received at the end of the period and no additional returns on those dividends are earned. If the assumption is that dividends were not reinvested and were kept under the mattress, then this calculation may be appropriate. However, we would argue that the more plausible alternative is for us to re-invest the dividend in a security that is of similar risk-return profile as our initial investment, and a security that satisfies that is the same security that generated those dividends. In other words, when we receive the dividend, we would reinvest that amount back into the stock. The returns of that stock going forward determines whether the reinvested dividend earned a positive or negative return.

Let us now walk through an example of cumulating returns. Suppose we are interested in knowing how much would an investment in Amazon have made through the end of 2013 if we purchased AMZN shares at the closing price on December 31, 2010. We show how to implement this calculation using arithmetic returns and logarithmic returns, and show that, when consistently implemented, both types of returns yield the same result. However, some may find the programming for logarithmic returns slightly easier.

### 2.4.1 Cumulating Arithmetic Returns

To string together multiple days of arithmetic returns, we have to take the product of the daily gross returns. The gross return is one plus the net return  $R_t$ . That is, for a 2-day cumulative return, we take  $(1 + R_1) \times (1 + R_2)$ . For a 3-day cumulative return, we take  $(1 + R_1) \times (1 + R_2) \times (1 + R_3)$ . Therefore, we can generalize this calculation over a  $T$ -day investment horizon as

$$R_{1 \text{ to } T} = (1 + R_1) \times (1 + R_2) \times \cdots \times (1 + R_T). \quad (2.4)$$

**Step 1: Import Data and Calculate Arithmetic Returns** Since we have already calculate the daily arithmetic total returns above, we call `IBM.ret` to see that we have the correct data still available in the R memory. If not, then we can run the code that generates `IBM.ret` above.

```
> IBM.acum<-IBM.ret
> IBM.acum[c(1:3,nrow(IBM.acum)),]
      IBM.tot.ret
2010-12-31      NA
2011-01-03 0.004884005
2011-01-04 0.001072118
2013-12-31 0.006222842
```

**Step 2: Set First Day Total Return Value to Zero** Because we assume that we are making the investment on December 31, 2010, we should set the return on that day equal to zero.

```
> IBM.acum[1,1]<-0
> IBM.acum[c(1:3,nrow(IBM.acum)),]
      IBM.tot.ret
2010-12-31 0.000000000
2011-01-03 0.004884005
2011-01-04 0.001072118
2013-12-31 0.006222842
```

**Step 3: Calculate Gross Daily Returns** We then create a new variable for the *gross return*, which is simply one plus the net total return.

```
> IBM.acum$GrossRet<-1+IBM.acum$IBM.tot.ret
> IBM.acum[c(1:3,nrow(IBM.acum)),]
      IBM.tot.ret GrossRet
2010-12-31 0.000000000 1.000000
2011-01-03 0.004884005 1.004884
2011-01-04 0.001072118 1.001072
2013-12-31 0.006222842 1.006223
```

**Step 4: Calculate Cumulative Gross Returns** We use the `cumprod` command to take the cumulative product of the gross return. `cumprod` takes the product of all the gross returns from December 31, 2010 to the valuation date. For example, the `GrossCum` from December 31, 2010 to January 4, 2011 is equal to 1.005961, which

is the product of 1.00000, 1.004884, and 1.001072 gross returns from December 31, 2010, January 3, 2011, and January 4, 2011, respectively.

```
> IBM.acum$GrossCum<-cumprod(IBM.acum$GrossRet)
> IBM.acum[c(1:3,nrow(IBM.acum)),]
      IBM.tot.ret GrossRet GrossCum
2010-12-31 0.000000000 1.000000 1.000000
2011-01-03 0.004884005 1.004884 1.004884
2011-01-04 0.001072118 1.001072 1.005961
2013-12-31 0.006222842 1.006223 1.347195
```

**Step 5: Convert Cumulative Gross Returns to Cumulative Net Returns** Note that the above value is still a gross return number. So the last step would require us to subtract one from the `GrossCum` to calculate the *net cumulative return* or `NetCum`. The `NetCum` number is interpreted as the percentage return from the investment date, December 31, 2010. This means that an investment made in IBM stock at the end of 2010 would have returned 34.7% by the end of 2013.

```
> IBM.acum$NetCum<-IBM.acum$GrossCum-1
> IBM.acum[c(1:3,nrow(IBM.acum)),]
      IBM.tot.ret GrossRet GrossCum      NetCum
2010-12-31 0.000000000 1.000000 1.000000 0.000000000
2011-01-03 0.004884005 1.004884 1.004884 0.004884005
2011-01-04 0.001072118 1.001072 1.005961 0.005961359
2013-12-31 0.006222842 1.006223 1.347195 0.347195288
```

### 2.4.2 Cumulating Logarithmic Returns

An alternative way to calculate multi-period returns is to take the sum of the daily logarithmic returns. That is,

$$\begin{aligned} r_{1 \text{ to } T} &= \ln((1 + R_1) \times (1 + R_2) \times \cdots \times (1 + R_T)) \\ &= r_1 + r_2 + \cdots + r_T \\ &= \sum_{t=1}^T r_t \end{aligned} \quad (2.5)$$

**Step 1: Import Data and Calculate Logarithmic Returns** Since we have already calculated the daily log total returns above, we call `IBM.logret` to see that we have the correct data still available in the R memory. If not, then we can run the code that generates `IBM.logret` above.

```
> IBM.logcum<-IBM.log.ret
> IBM.logcum[c(1:3,nrow(IBM.logcum)),]
      IBM.log.ret
2010-12-31      NA
2011-01-03 0.004872117
2011-01-04 0.001071543
2013-12-31 0.006203560
```

**Step 2: Set the First Log Return to Zero** We also set the return on December 31, 2010 to zero as we are calculating returns as if we purchased the IBM shares at the close on December 31, 2010.

```
> IBM.logcum[1,1]<-0
> IBM.logcum[c(1:3,nrow(IBM.logcum)),]
      IBM.log.ret
2010-12-31 0.000000000
2011-01-03 0.004872117
2011-01-04 0.001071543
2013-12-31 0.006203560
```

**Step 3: Take the Sum of all Logarithmic Returns During the Investment Period** We add up the values in `IBM.log.ret` variable using the `sum` command.

```
> logcumret=sum(IBM.logcum$IBM.log.ret)
> logcumret
[1] 0.2980249
```

**Step 4: Convert Log Return Back to Arithmetic Return** Unlike the arithmetic cumulative return, the logarithmic cumulative return may not have any practical interpretation. Therefore, we would need to convert the cumulative logarithmic return to a cumulative arithmetic return. We do this by taking the exponential of the logarithmic return using the `exp` command.

```
> cumret=exp(logcumret)-1
> cumret
[1] 0.3471953
```

The logarithmic return calculated a cumulative return of 34.7 %, which is identical to the cumulative return calculated using the arithmetic return. However, we can see that it takes fewer and simpler steps to calculate multi-period returns using logarithmic returns than it is using arithmetic returns. However, if we need to show daily cumulative values (as we will have to in the next section), we would be better off following the technique discussed when we cumulated arithmetic returns.

### 2.4.3 Comparing Price Return and Total Return

Using IBM stock as an example and the technique to calculate multi-period arithmetic returns discussed above, we now show that the total return yields higher returns than price returns for a stock that pays dividends.

**Step 1: Import Data and Calculate Price and Total Returns** Since we already have combined the price and total returns data previously in `tot.rets`, we only need to call-in the data at this stage. Then, we rename the data to better identify the variables as `prc.ret` and `tot.ret`.

```
> IBM.Ret<-cbind(IBM.prc.ret, IBM.ret)
> names(IBM.Ret)<-c("prc.ret", "tot.ret")
> IBM.Ret[c(1:3,nrow(IBM.Ret)), ]
      prc.ret      tot.ret
2010-12-31      NA      NA
2011-01-03 0.004905969 0.004884005
2011-01-04 0.001084893 0.001072118
2013-12-31 0.006222842 0.006222842
```

**Step 2: Set First Returns to Zero** We then set the December 31, 2011 price and total return to zero.

```
> IBM.Ret$prc.ret[1]<-0
> IBM.Ret$tot.ret[1]<-0
> IBM.Ret[c(1:3,nrow(IBM.Ret)), ]
      prc.ret      tot.ret
2010-12-31 0.000000000 0.000000000
2011-01-03 0.004905969 0.004884005
2011-01-04 0.001084893 0.001072118
2013-12-31 0.006222842 0.006222842
```

**Step 3: Calculate Gross Returns** Then, we calculate the gross price return `gross.prc` and gross total return `gross.tot`.

```
> IBM.Ret$gross.prc<-1+IBM.Ret$prc.ret
> IBM.Ret$gross.tot<-1+IBM.Ret$tot.ret
> IBM.Ret[c(1:3,nrow(IBM.Ret)), ]
      prc.ret      tot.ret gross.prc gross.tot
2010-12-31 0.000000000 0.000000000 1.000000 1.000000
2011-01-03 0.004905969 0.004884005 1.004906 1.004884
2011-01-04 0.001084893 0.001072118 1.001085 1.001072
2013-12-31 0.006222842 0.006222842 1.006223 1.006223
```

**Step 4: Cumulate the Gross Returns** Lastly, we calculate the cumulative price return `cum.prc` and cumulative total return `cum.tot` by using the `cumprod` command.

```
> IBM.Ret$cum.prc<-cumprod(IBM.Ret$gross.prc)
> IBM.Ret$cum.tot<-cumprod(IBM.Ret$gross.tot)
> IBM.Ret[c(1:3,nrow(IBM.Ret)), ]
      prc.ret      tot.ret gross.prc gross.tot cum.prc cum.tot
2010-12-31 0.000000000 0.000000000 1.000000 1.000000 1.000000 1.000000
2011-01-03 0.004905969 0.004884005 1.004906 1.004884 1.004906 1.004884
2011-01-04 0.001084893 0.001072118 1.001085 1.001072 1.005996 1.005961
2013-12-31 0.006222842 0.006222842 1.006223 1.006223 1.278073 1.347195
```

**Step 5: Plot the Two Return Series** We then plot the `cum.prc` and `cum.tot` variables. We first plot the `cum.tot` variable, and then plot the `cum.prc` variable. Using the `abline` command, we add a horizontal line at \$ 1 to make it easy for us to interpret whether the investment is making money or losing money. From the output above, we can see that by the end of 2013, reinvesting dividends yields a return of 34.7 % while price appreciation alone yields 27.8 %.

```

> y.range<-range(IBM.Ret[,5:6])
> y.range
[1] 1.000000 1.526826
> plot(IBM.Ret$cum.tot,
+      type="l",
+      auto.grid=FALSE,
+      xlab="Date",
+      ylab="Value of Investment ($)",
+      ylim=y.range,
+      minor.ticks=FALSE,
+      main="IBM Stock Performance Based On
+          Total Returns and Price Returns
+          December 31, 2010 - December 31, 2013")
> lines(IBM.Ret$cum.prc,
+      type="l",
+      lty=3)
> abline(h=1,col="black")
> legend("topleft",
+      col=c("black","black"),
+      lty=c(1,3),
+      c("Value Based on Total Return",
+      "Value Based on Price Return"))

```

As Fig. 2.1 shows, the value of an investment based on IBM's total return (solid line) is equal to or greater than the value on an investment based on IBM's price return (dashed line). The two lines overlap each other at the start of the chart in early 2011 because no ex-dividend dates occurred over that time period. IBM pays quarterly dividends, so it would have four ex-dates each year. Hence, the gap between the black line and gray line increases over time as more dividend payments are made by IBM.

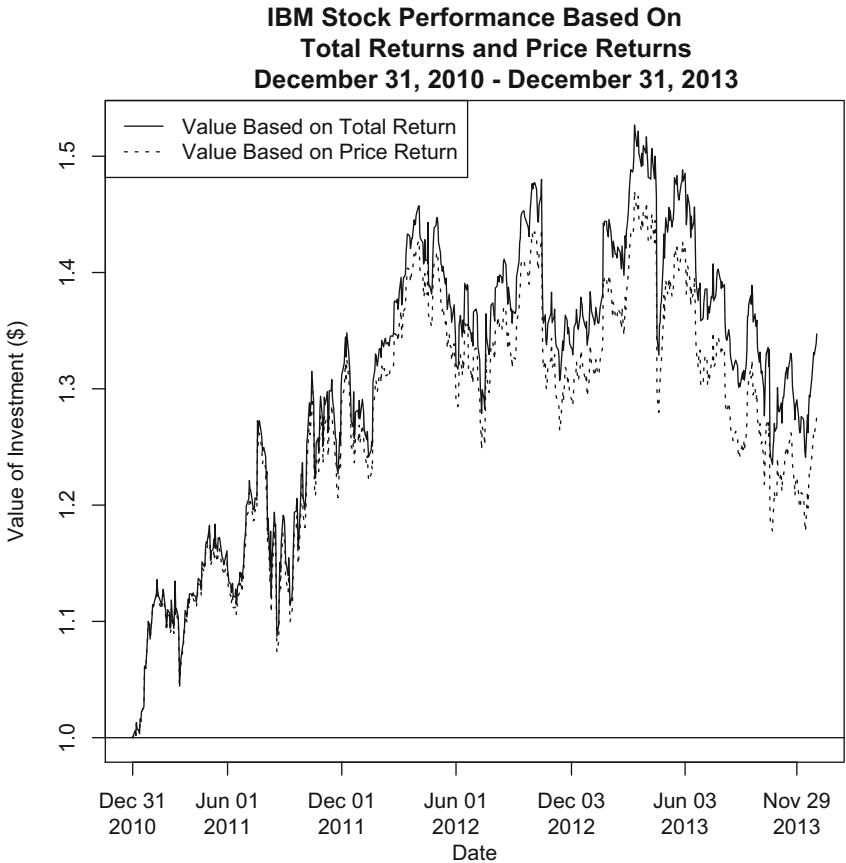
## 2.5 Weekly Returns

In the previous sections, we used daily returns. However, there are applications in which we may have to use returns of lower frequency, such as *weekly returns*. In this section, we will revert back to using AMZN stock data. As such, we have to import AMZN data. We need the `quantmod` package and `xts` package for our calculations in this section. As `data.AMZN` is an `xts` object, we can easily change the daily data to weekly using the `to.weekly` command.

```

> data.AMZN<-read.csv("AMZN Yahoo.csv",header=TRUE)
> date<-as.Date(data.AMZN$Date,format="%Y-%m-%d")
> data.AMZN<-cbind(date, data.AMZN[, -1])
> data.AMZN<-data.AMZN[order(data.AMZN$date), ]
> data.AMZN<-xts(data.AMZN[, 2:7],order.by=data.AMZN[, 1])
> names(data.AMZN)<-
+ paste(c("AMZN.Open", "AMZN.High", "AMZN.Low",
+ "AMZN.Close", "AMZN.Volume", "AMZN.Adjusted"))
> data.AMZN[c(1:3,nrow(data.AMZN)), ]

```



**Fig. 2.1** Effect on investment performance of reinvesting dividends. Reproduced with permission of CSI ©2013. Data Source: CSI [www.csidata.com](http://www.csidata.com)

	AMZN.Open	AMZN.High	AMZN.Low	AMZN.Close	AMZN.Volume	AMZN.Adjusted
2010-12-31	181.96	182.30	179.51	180.00	3451900	180.00
2011-01-03	181.37	186.00	181.21	184.22	5331400	184.22
2011-01-04	186.15	187.70	183.78	185.01	5031800	185.01
2013-12-31	394.58	398.83	393.80	398.79	1996500	398.79

```
> class(data.AMZN)
[1] "xts" "zoo"
```

We now demonstrate how to calculate weekly returns from the daily AMZN data above.

**Step 1: Import Data into R** For this calculation, we use the `data.AMZN` data object we previous created. We save the data into a new data object labeled `wk`.

```
> wk<-data.AMZN
> wk[c(1:3,nrow(data.AMZN)),]
      AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2010-12-31    181.96    182.30    179.51     180.00     3451900         180.00
2011-01-03    181.37    186.00    181.21     184.22     5331400         184.22
2011-01-04    186.15    187.70    183.78     185.01     5031800         185.01
2013-12-31    394.58    398.83    393.80     398.79     1996500         398.79
```

**Step 2: Convert to Daily Data Data to Weekly Data** Using the `to.weekly` function, we convert the daily price data to weekly price data.

```
> AMZN.weekly<-to.weekly(wk)
> AMZN.weekly[c(1:3,nrow(AMZN.weekly)),]
      wk.Open wk.High wk.Low wk.Close wk.Volume wk.Adjusted
2010-12-31    181.96    182.30    179.51     180.00     3451900         180.00
2011-01-07    181.37    188.45    181.21     185.49     22183400         185.49
2011-01-14    185.04    188.94    182.51     188.75     15899000         188.75
2013-12-31    399.41    399.92    392.45     398.79      4483600         398.79
```

### Programming Tip

Note that in Step 1, we used a short name for the data object. The reason for this is because the `to.weekly` command uses the data object name as the prefix of the new variable names as shown above. Had we used `data.AMZN`, we would have ended up with, say, an adjusted close price label of `data.AMZN.Adjusted` and is less meaningful (i.e., that label does not tell you about the weekly nature of the data).

**Step 3: Clean up Weekly Data to Keep Only the Adjusted Closing Prices at the End of Each Week** We then clean up the data object to only keep the adjusted close data, which is the sixth column in `AMZN.weekly`.

```
> AMZN.weekly<-AMZN.weekly[,6]
> AMZN.weekly[c(1:3,nrow(AMZN.weekly)),]
      wk.Adjusted
2010-12-31      180.00
2011-01-07      185.49
2011-01-14      188.75
2013-12-31      398.79
```

**Step 4: Calculate Weekly Returns** Then, using the `Delt` command, we calculate the weekly return by calculating the percentage change in adjusted price from the Friday of a particular week and the adjusted price from the Friday of the prior week. As such, what we calculate is a Friday-to-Friday total return. Note that if Friday is a non-trading day, the weekly return will be calculated based on the price of the last trading day of this week relative to the price of the last trading day of the prior week.



```
> AMZN.weekly$Ret<-Delt(AMZN.weekly$wk.Adjusted)
> AMZN.weekly[c(1:3,nrow(AMZN.weekly)),]
      wk.Adjusted      Ret
2010-12-31      180.00      NA
2011-01-07      185.49 0.030500000
2011-01-14      188.75 0.017575071
2013-12-31      398.79 0.001783561
```

Note that the December 31, 2013 return is not a full week's return. The last Friday prior to the end of the year is on December 27, 2013.

**Step 5: Cleanup Weekly Returns Data by Deleting the First Observation** Note that the first observation has an NA, which we do not want to retain in our final data. In addition, the only variable we need is the return variable. As such, we should delete the first row *and* keep the second column.

```
> AMZN.weekly<-AMZN.weekly[-1,2]
> AMZN.weekly[c(1:3,nrow(AMZN.weekly)),]
      Ret
2011-01-07 0.030500000
2011-01-14 0.017575071
2011-01-21 -0.060026490
2013-12-31 0.001783561
```

### Verify Weekly Return Calculations

It is a good habit to test our calculations to make sure that we did not make a mistake in programming. Since we have calculated the cumulative returns above in `AMZN.acum`, we can output the data for January 7, 14, and 21, 2011.

```
> AMZN.acum[c(6,11,15),]
      AMZN.tot.ret GrossRet GrossCum NetCum
2011-01-07 -0.001990746 0.9980093 1.0305000 0.03050000
2011-01-14 0.017355684 1.0173557 1.0486111 0.04861111
2011-01-21 -0.024950539 0.9750495 0.9856667 -0.01433333
```

The output above shows the the `NetCum` of 3.05 % for January 7, 2011 is identical to the weekly return for January 7, 2011 in `AMZN.weekly`. That is straightforward to determine because it is the first `NetCum` observation.

For the subsequent returns, we have to perform a little arithmetic to verify that the two series yields identical results (except for rounding differences). Let us look at the January 14, 2011 `GrossCum` of 1.0486111. To figure out the weekly return for that week, we have to divide the `GrossCum` on January 14, 2011 by the `GrossCum` on January 7, 2011 of 1.0305000. This yields a gross weekly return of 1.017575061 [= 1.046111/1.0305000] or a net weekly return of 1.76 %. This matches the return for the week of January 14, 2011 in `AMZN.weekly`, except for numbers beginning the eighth decimal place.

Next, let us verify the return for the week of January 21, 2011. In `AMZN.acum`, we find a gross weekly return of 0.939973552 or a net weekly return of  $-6.0\%$ . This again matches the weekly return for January 21, 2011 in `AMZN.weekly` except for numbers beginning the eighth decimal place.

## 2.6 Monthly Returns

Another option for lesser frequency returns are *monthly returns*. The implementation is similar to calculating weekly returns.

**Step 1: Import Data into R** We again use `data.AMZN` we imported in the earlier section as a starting point for our monthly return calculation.

```
> mo<-data.AMZN
> mo[c(1:3,nrow(data.AMZN)),]
      AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2010-12-31    181.96    182.30    179.51     180.00     3451900         180.00
2011-01-03    181.37    186.00    181.21     184.22     5331400         184.22
2011-01-04    186.15    187.70    183.78     185.01     5031800         185.01
2013-12-31    394.58    398.83    393.80     398.79     1996500         398.79
```

**Step 2: Convert Daily Data to Monthly Data.** Using the `to.monthly` command, we convert the daily data to monthly data.

```
> AMZN.monthly<-to.monthly(mo)
> AMZN.monthly[c(1:3,nrow(AMZN.monthly)),]
      mo.Open mo.High mo.Low mo.Close mo.Volume mo.Adjusted
Dec 2010    181.96    182.30    179.51     180.00     3451900         180.00
Jan 2011    181.37    191.60    166.90     169.64    113611300         169.64
Feb 2011    170.52    191.40    169.51     173.29     95776400         173.29
Dec 2013    399.00    405.63    379.50     398.79     55638100         398.79
Warning message:
timezone of object (UTC) is different than current timezone ().
```

Notice the warning message that appears every time we use the `to.monthly` command. This is because the object's time appears to be in Coordinated Universal Time (UTC). UTC is commonly referred to as Greenwich Mean Time (GMT). For our purposes, this should not affect our results.

**Step 3: Clean up Data to Include Only Adjusted Closing Prices for the End of Each Month** We subset the data to only show the adjusted close price, because that is the only variable we need to calculate monthly returns.

```
> AMZN.monthly<-AMZN.monthly[,6]
> AMZN.monthly[c(1:3,nrow(AMZN.monthly)),]
      mo.Adjusted
Dec 2010      180.00
Jan 2011      169.64
Feb 2011      173.29
Dec 2013      398.79
Warning message:
timezone of object (UTC) is different than current timezone ().
```

**Step 4: Calculate Monthly Returns** Using the `Delt` command, we can calculate the percentage change in the end-of-the-month adjusted close price. As such, we get a month-end to month-end return using this approach.

```
> AMZN.monthly$Ret<-Delt(AMZN.monthly$mo.Adjusted)
> AMZN.monthly[c(1:3,nrow(AMZN.monthly)),]
      mo.Adjusted      Ret
Dec 2010      180.00      NA
Jan 2011      169.64 -0.05755556
Feb 2011      173.29  0.02151615
Dec 2013      398.79  0.01313450
```

**Step 5: Cleanup Monthly Data Object** Since December 2010 has NA for its return and since we only need the return column, we subset the data by deleting the first row and keeping the second column.

```
> AMZN.monthly<-AMZN.monthly[-1,2]
> AMZN.monthly[c(1:3,nrow(AMZN.monthly)),]
      Ret
Jan 2011 -0.05755556
Feb 2011  0.02151615
Mar 2011  0.03947141
Dec 2013  0.01313450
```

## 2.7 Comparing Performance of Multiple Securities: Total Returns

Total returns capture both the returns from capital appreciation and dividends. Therefore, it is a better measure of investment performance than price returns. In the last chapter, we showed how to create normalized price charts based on price returns. In that example, we retrieved data from Yahoo Finance for Amazon (AMZN), S&P 500 Index (^GSPC), Yahoo (YHOO), and IBM from December 31, 2013 to December 31, 2013. AMZN and YHOO have not paid dividends, but IBM has. The S&P 500 Index data we use does not include dividends. Therefore, the index value for AMZN, YHOO, and GSPC would be the same regardless of whether we use price returns or total returns. However, for IBM, the index value using total returns will be higher than the index value using price returns.

**Step 1: Importing Price Data** The programming technique used in this section will be identical to the programming technique used to generate the normalized price chart in the previous chapter. We start by calling in the Yahoo Finance data for AMZN, GSPC, YHOO, and IBM. Since we have already called-in AMZN's and IBM's data in this chapter, we simply check to make sure that the data is still available in R's memory and that we did not inadvertently erase it. For GSPC and YHOO, we read in the files **GSPC Yahoo.csv** and **YHOO Yahoo.csv** that contains the Yahoo Finance data for these the two symbols.

```
> data.AMZN[c(1:3,nrow(data.AMZN)),]
      AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2010-12-31    181.96    182.30    179.51    180.00    3451900    180.00
2011-01-03    181.37    186.00    181.21    184.22    5331400    184.22
2011-01-04    186.15    187.70    183.78    185.01    5031800    185.01
2013-12-31    394.58    398.83    393.80    398.79    1996500    398.79
>
> data.IBM[c(1:3,nrow(data.IBM)),]
      IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adjusted
2010-12-31    146.73    147.07    145.96    146.76    2969800    139.23
2011-01-03    147.21    148.20    147.14    147.48    4603800    139.91
2011-01-04    147.56    148.22    146.64    147.64    5060100    140.06
2013-12-31    186.49    187.79    186.30    187.57    3619700    187.57
>
> # GSPC Data
> data.GSPC<-read.csv("GSPC Yahoo.csv",header=TRUE)
> date<-as.Date(data.GSPC$Date,format="%Y-%m-%d")
> data.GSPC<-cbind(date, data.GSPC[,~1])
> data.GSPC<-data.GSPC[order(data.GSPC$date),]
> data.GSPC<-xts(data.GSPC[,2:7],order.by=data.GSPC[,1])
> names(data.GSPC)[1:6]<-
+   paste(c("GSPC.Open", "GSPC.High", "GSPC.Low",
+   "GSPC.Close", "GSPC.Volume", "GSPC.Adjusted"))
> data.GSPC[c(1:3,nrow(data.GSPC)),]
      GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
2010-12-31    1256.76    1259.34    1254.19    1257.64    1799770000    1257.64
2011-01-03    1257.62    1276.17    1257.62    1271.87    4286670000    1271.87
2011-01-04    1272.95    1274.12    1262.66    1270.20    4796420000    1270.20
2013-12-31    1842.61    1849.44    1842.41    1848.36    2312840000    1848.36
>
```

```
> # Yahoo Data
> data.YHOO<-read.csv("YHOO Yahoo.csv",header=TRUE)
> date<-as.Date(data.YHOO$Date,format="%Y-%m-%d")
> data.YHOO<-cbind(date, data.YHOO[,~1])
> data.YHOO<-data.YHOO[order(data.YHOO$date),]
> data.YHOO<-xts(data.YHOO[,2:7],order.by=data.YHOO[,1])
> names(data.YHOO)[1:6]<-
+   paste(c("YHOO.Open", "YHOO.High", "YHOO.Low",
+   "YHOO.Close", "YHOO.Volume", "YHOO.Adjusted"))
> data.YHOO[c(1:3,nrow(data.YHOO)),]
      YHOO.Open YHOO.High YHOO.Low YHOO.Close YHOO.Volume YHOO.Adjusted
2010-12-31    16.74    16.76    16.47     16.63     7754500         16.63
2011-01-03    16.81    16.94    16.67     16.75    17684000         16.75
2011-01-04    16.71    16.83    16.57     16.59    11092800         16.59
2013-12-31    40.17    40.50    40.00     40.44     8291400         40.44
```

**Step 2: Combine Data** To make the calculations easier, we will combine the four data objects containing the price data into one data object `multi` that holds only the adjusted closing price data for the four securities. To create `multi`, we call-in the `AMZN` adjusted closing price in `data.AMZN` first, then for each of the next three securities, we use the `merge` command to combine what is in the `multi` data object with the adjusted closing price of the three securities.

```
> multi<-data.AMZN[,6]
> multi<-merge(multi,data.GSPC[,6])
> multi<-merge(multi,data.YHOO[,6])
> multi<-merge(multi,data.IBM[,6])
> multi[c(1:3,nrow(multi)),]
      AMZN.Adjusted GSPC.Adjusted YHOO.Adjusted IBM.Adjusted
2010-12-31      180.00      1257.64        16.63       139.23
2011-01-03      184.22      1271.87        16.75       139.91
2011-01-04      185.01      1270.20        16.59       140.06
2013-12-31      398.79      1848.36        40.44       187.57
```

**Step 3: Converting Data into a data.frame Object** The next set of steps creates a `data.frame` object with a workable `date` variable and shortened names for the four adjusted closing price variables. For more details on the explanation of the code, please refer to the discussion of subsetting data using dates or generating a normalized price chart using price returns in Chap. 1.

```
> multi.df<-cbind(data.frame(index(multi)),
+   data.frame(multi))
> names(multi.df)<-paste(c("date", "AMZN", "GSPC", "YHOO", "IBM"))
> multi.df[c(1:3,nrow(multi.df)),]
      date AMZN GSPC YHOO IBM
2010-12-31 2010-12-31 180.00 1257.64 16.63 139.23
2011-01-03 2011-01-03 184.22 1271.87 16.75 139.91
2011-01-04 2011-01-04 185.01 1270.20 16.59 140.06
2013-12-31 2013-12-31 398.79 1848.36 40.44 187.57
```

**Step 4: Constructing Normalized Values for Each Security** We now turn to indexing each security's adjusted closing price to its adjusted closing price on December 31, 2010. We do this by dividing each security's adjusted closing price by its adjusted closing price on December 31, 2010. For example, for AMZN, its December 31, 2010 index value of 1.00000 is equal to its December 31, 2010 price of \$ 180.00 divided by its December 31, 2010 price of \$ 180. Then, its January 3, 2011 index value of 1.023444 is equal to its January 3, 2011 value of \$ 184.22 divided by its December 31, 2010 value of \$ 180.00. AMZN's December 31, 2013 index value of 2.215500 is equal to its December 31, 2013 price of \$ 398.79 by its December 31, 2010 price of \$ 180.00.

```
> multi.df$AMZN.idx<-multi.df$AMZN/multi.df$AMZN[1]
> multi.df$GSPC.idx<-multi.df$GSPC/multi.df$GSPC[1]
> multi.df$YHOO.idx<-multi.df$YHOO/multi.df$YHOO[1]
> multi.df$IBM.idx<-multi.df$IBM/multi.df$IBM[1]
> multi.df[c(1:3,nrow(multi.df)),6:9]
      AMZN.idx GSPC.idx YHOO.idx IBM.idx
2010-12-31 1.000000 1.000000 1.000000 1.000000
2011-01-03 1.023444 1.011315 1.0072159 1.004884
2011-01-04 1.027833 1.009987 0.9975947 1.005961
2013-12-31 2.215500 1.469705 2.4317498 1.347195
```

We only show columns 6 through 9 above, but the first five columns would be identical to the output in Step 3.

### Interpreting the Index Values

Note that we can easily read off the performance of the investment from December 31, 2010 using the index values in `multi.df`. For example, by December 31, 2013, Yahoo's price had increased by approximately 143.2 % from its December 31, 2010 price, which is equal to the `YHOO.idx` value of 2.432 on December 31, 2013  $- 1$ . The  $- 1$  is because the index values are equivalent to gross returns.

**Step 5: Plotting the Index Values of Each Security** To generate the normalized price chart based on total returns, we follow a similar programming code as what we used when generating the normalized price chart based on price returns in Chap. 1. The two significant features to note when we create this chart are the following. First, we create a `y.range` variable to make sure that the chart generated encompasses the largest and smallest index values. Second, we differentiate the four securities by using lines of different color (e.g., choosing black or different shades of gray in the `col` command), thickness (choosing option 1 or 2 in the `lwd` command), and type (e.g., choosing option 1 (solid) or 2 (dashed) in the `lty` command).

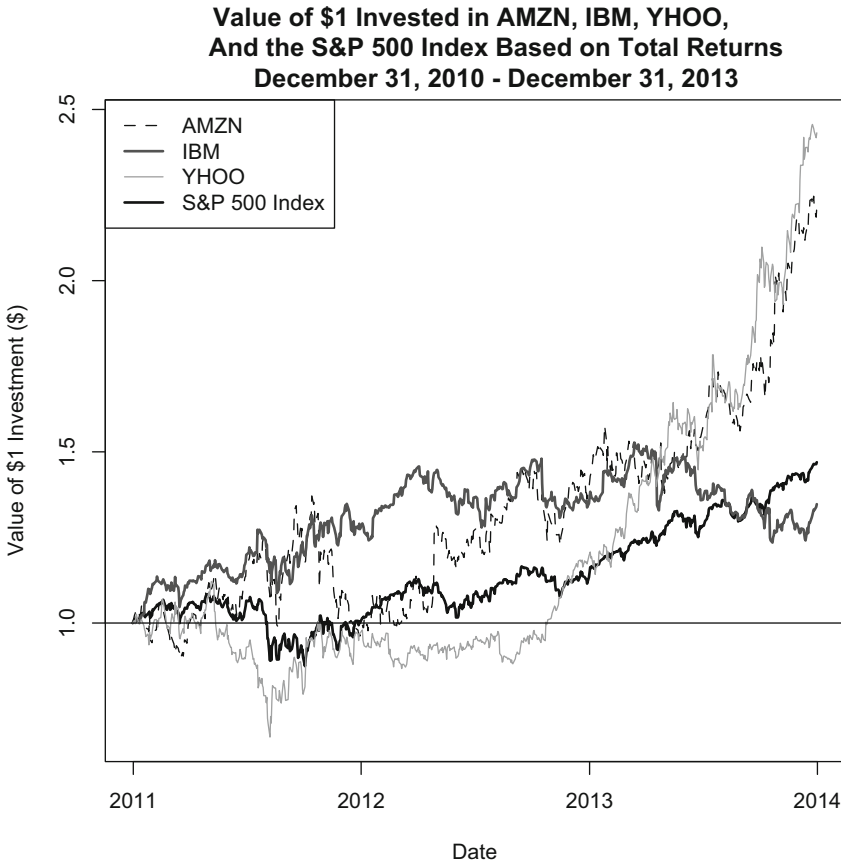
```

> y.range<-range(multi.df[,6:9])
> y.range
[1] 0.6668671 2.4564041
> par(mfrow=c(1,1))
> plot(x=multi.df$date,
+      xlab="Date",
+      y=multi.df$GSPC.idx,
+      ylim=y.range,
+      ylab="Value of $1 Investment ($)",
+      type="l",
+      col="black",
+      lty=1,
+      lwd=2,
+      main="Value of $1 Invested in AMZN, IBM, YHOO,
+           And the S&P 500 Index Based on Total Returns
+           December 31, 2010 - December 31, 2013")
> lines(x=multi.df$date,
+       y=multi.df$AMZN.idx,
+       col="black",
+       lty=2,
+       lwd=1)
> lines(x=multi.df$date,
+       y=multi.df$IBM.idx,
+       col="gray40",
+       lty=1,
+       lwd=2)
> lines(x=multi.df$date,
+       y=multi.df$YHOO.idx,
+       col="gray60",
+       lty=1,
+       lwd=1)
> abline(h=1,lty=1,col="black")
> legend("topleft",
+       c("AMZN", "IBM", "YHOO", "S&P 500 Index"),
+       col=c("black", "gray40", "gray60", "black"),
+       lty=c(2, 1, 1, 1),
+       lwd=c(1, 2, 1, 2))

```

The last few lines of the code shows that we add two elements to the chart. First, we add a horizontal line to represent \$ 1 using the `abline` command. This makes interpreting the investment performance easier, as anytime the line is above the \$ 1 line the investment is making money and anytime the line is below the \$ 1 line the investment is losing money. Second, we add a legend at the top-left portion of the graph, and specify the name, line type, line thickness, and color of each of the securities in the order in which we want them to appear on the legend. Note that the legend is independent of how the chart was created. Put differently, we can re-arrange the order of the lines in the legend. We only need to make sure that the order of securities in the legend itself is consistent.

Figure 2.2 shows the output of the normalized chart. As the chart shows, Yahoo outperformed the other three securities with a return of over 140 % from 2011 to 2013. Amazon.com came in second with over 120 % return from 2011 to 2013.



**Fig. 2.2** Performance of AMZN, YHOO, IBM, and the S&P 500 index based on total returns, December 31, 2010 to December 31, 2013. Reproduced with permission of CSI ©2013. Data Source: CSI [www.csidata.com](http://www.csidata.com)

These two securities beat the market, as represented by the S&P 500 Index, which only increased approximately 47 %. The laggard of the group is IBM, which although increased by approximately 35 % from 2011 to 2013, did not beat the market's return during the period.



<http://www.springer.com/978-3-319-14074-2>

Analyzing Financial Data and Implementing Financial  
Models Using R

Ang, C.

2015, XVI, 351 p. 60 illus., Hardcover

ISBN: 978-3-319-14074-2