Clifford S. Ang

Analyzing Financial Data and Implementing Financial Models Using R

Springer Texts in Business and Economics

More information about this series at http://www.springer.com/series/10099

Clifford S. Ang

Analyzing Financial Data and Implementing Financial Models Using R

Springer

Clifford S. Ang
Compass Lexecon
Chicago, Illinois
USA

To Claudine, Cole, and Cody.

Preface

This is a financial modeling book aimed at the relative beginner to R. The student does not need to have prior financial modeling background, but having gone through a corporate finance and investments course would be helpful. The goal of this text is for the student to be able to obtain raw data, manipulate and analyze that data, implement financial models, and generate the output required for the analysis.

There are three main features of this book. First, we use a R as the program of choice because it is free and has a large on-line community that can provide support to programmers of all levels. This means that the student can gain familiarity with software that is being used by many people and does not cost the students anything to acquire and update. In contrast, many texts use commercial software that the student has to end up paying hundreds, if not thousands, of dollars in acquisition or renewal costs to use after they get out of school.

Second, the examples in the book only use real-world data available for free to the student for their personal use. We will primarily use data obtained from Yahoo Finance and the Federal Reserve Electronic Database. Unlike typical textbook examples in which students see sample data that are sanitized for a particular purpose, real-world data comes in a generic format that will likely not be suited for a specific analysis. Although by using traditional textbook examples the student may be able to say that they have learned to "analyze" data or "implemented" models, the lack of experience using real world data will likely make the student feel challenged when applying such analyses in practice.

Finally, the discussion in this text handholds the student through every step of the way. The examples take the student from obtaining the raw data to manipulating that data to performing the analysis and ends by showing how to generate a typical output for that particular analysis. In addition, we also present intermediate output, so students can quickly identify which portion of their code contains the error and should get them back on track sooner.

Now that I have discussed what this book is about, let me briefly go through what you will not see in this book. First, although this book teaches students how to program in R, this is not a technical programming book. As such, I will be loose with programming terminology. I will also sacrifice efficiency in writing code. The primary reason is that the data used in our examples is relatively small, so the entire

code runs in a matter of seconds. Therefore, we would not make any practical gains from writing efficient code and the student may end up losing the intuition gained by laying out each step.

Next, unlike some financial modeling books, this text will not come with program codes available for download. In my opinion, giving the student an option to copy and paste code will defeat the purpose of learning how to program. Programming is one of those skills that students cannot learn without spending a decent amount of time getting their hands dirty. This text shows the students all the code and also shows them the intermediate output. As such, the very design of the book is to help the student not get lost along the way. This book goes almost all the way to the end but essentially stops short of typing the code for the student.

Structure of the Book

This book is broken up into nine chapters. Each chapter is pretty much self contained. It is recommended that two packages are installed at the start of each chapter. These are quantmod and xts. I also suggest to write code using the R Editor, so each chapter's code can be saved in one file. Although you can skip around to different chapters, I recommend going through the chapters linearly as I likely will provide fuller explanations the first time certain techniques are used or issues appear.

Chapter 1 is about security prices and introduces the student to basic data manipulation techniques. In addition, we show examples of how to perform technical analysis in R.

In Chaps. 2 and 3, we demonstrates how to calculate returns for individual securities and portfolios. Specifically, we show how to calculate arithmetic returns, logarithmic returns, price returns, total returns, daily returns, weekly returns, and monthly returns. We also go through the construction of equal-weighted and value-weighted portfolio returns with quarterly rebalancing.

Chapter 4 deals with risk, which is the other side of the risk-return trade-off. We show how to measure individual security risk and portfolio risk using variance or standard deviation as the risk measure. We also implement other measures of risk, namely Value-at-Risk or VaR, Expected Shortfall, and the risk measures developed by Parkinson, Garman-Klass, Rogers, Satchell, & Yoon, and Yang & Zhang.

In Chap. 5, we analyze factor models, which are models that explain the variation in expected stock returns using various proxies. We demonstrate how to implement the most popular of these models, which is the Capital Asset Pricing Model (CAPM), as well as a commonly-used alternative model developed by Eugene Fama and Kenneth French (i.e., the Fama-French Three Factor Model). We end this chapter with a discussion of a widely-used application of factor models called the "event study," which empirically analyzes the reaction of securities to the disclosure of value-relevant information.

To achieve higher returns, we have to take on more risk. In Chap. 6, we demonstrate how to calculate various commonly-used risk-adjusted portfolio performance

measures, namely the Sharpe Ratio, Roy's Safety First, Treynor Ratio, Sortino Ratio, and the Information Ratio. These risk-adjusted return measures allow us to rank different investments by their risk-return profile.

Chapter 7 discusses mean-variance optimization based on the work of Harry Markowitz. The basic idea is for us to find portfolios that provide the highest expected return for a given level of risk. We demonstrate the intuition of identifying mean-variance efficient portfolios and the construction of the mean-variance efficient frontier through a simple two-asset example. We then show how to use quadratic programming to extend the two-asset portfolio to a multi-asset portfolio. We end the chapter by showing how allowing short selling impacts the calculation of the efficient frontier.

In Chap. 8, we cover fixed income securities. We first show how to analyze economic and fixed income market data. Then, we demonstrate how to implement basic fixed income valuation models as well as the calculation of duration and convexity.

We end the book in Chap. 9 with showing how to analyze options data. We first go through the implementation of the Black-Scholes-Merton options pricing model (OPM) and the related Greeks. Then, we demonstrate how to implement the Binomial OPM.

Acknowledgments

Supplemental Website

Supplemental material for this book can be accessed at http://cliffordang.com.

Chicago, Illinois   Clifford Ang
November 2014

Contents

Chapter 1
Prices

The most fundamental analysis we undertake when investing revolves around the prices of *securities*, which is a term I will use throughout this book to refer to financial assets such as stocks, bonds, and options. At this point, an important distinction must be made between the *price* of a security and the *value* of a security. The price of a security is the amount of money we pay when we purchase a security or the amount of money we receive when we sell a security. In contrast, the value of a security is how much the security is worth.

Although the price of a security is the amount one investor is willing to forego to obtain the security and is an equivalent amount another investor is willing to receive to part with the security, the value of the security to those two investors may be different. These diverging valuations lead to trading among market participants. Specifically, investors purchase securities they think are undervalued and sell securities they think are overvalued. When there is sufficient trading in a particular security (i.e., markets are efficient), the observed market price of that security can be considered the best estimate of the value of that security.

The price of a security is at the core of investment analysis. For example, we can use the price of a security or securities as a benchmark when making investment decisions. From a fundamental value standpoint, investors would be inclined to buy (sell) the security if its price is sufficiently lower (higher) than the security's *intrinsic value* or the value of the security based on its fundamentals. Alternatively, from a *relative value* standpoint, investors would be inclined to buy or sell the security if its price is misaligned with the historical relationship between comparable securities and the investor believes the security's price will revert back to some average price.

Given the importance of prices in investments, we begin this book by analyzing prices. We show how to obtain and analyze raw security price data and how we can manipulate such raw data to fit some basic analysis using prices. In the first few chapters, we will focus on using stock data and exchange traded funds (ETF) data, which we obtain from Yahoo Finance. This allows us to gain familiarity with a reliable source of securities data that is widely-accessible. We then close the chapter by performing some commonly-used analyses that use security prices.

1.1 Importing Daily Stock Price Data

Before we can analyze the price data, we have to first obtain the data from a reliable source. One such source that is widely-accessible to the public is data on Yahoo Finance.[1] The data on Yahoo Finance provides us with the open price, high price, low price, close price, and trade volume data.[2]

We can import Yahoo Finance data several ways into R. However, throughout this book, we will use the most stable approach, which is to download the data into a CSV file and upload the CSV file into R. Alternatively, we can use an R package such as `quantmod` package and the `getSymbols` command to retrieve the data directly into R. However, this second approach is less stable and may be unusable when Yahoo Finance changes some specifications of the data. For example, in late 2013, Yahoo Finance modified the location of the data, so using commands such as `getSymbols` as well as other ways to retrieve data directly from within R was not working for some time. Only after a patch was created did the command work again. As such, to avoid such an issue, this book uploads data from a CSV obtained from Yahoo Finance. However, we manipulate the raw data into the identical form as what we would obtain had we used the `getSymbols` command. Put differently, if you decide to use the `getSymbols` command instead of the CSV upload code used in this book, you should be able to run the programs smoothly.

1.2 Importing Price Data from Yahoo Finance

To begin our analysis, we start by importing price data from Yahoo Finance. For purposes of our analysis, we will primarily use Amazon.com price data. There is no particular reason why we use Amazon.com price data, but I had to choose one stock as an example and Amazon.com seemed like a good choice.

Step 1: Import CSV File from Yahoo Finance To obtain data from Yahoo Finance, one would need to know either the company name or the security's ticker symbol. In this example, we are downloading data for Amazon.com with ticker symbol AMZN. So that we can replicate the results in this book on future dates, we are going to download historical stock price data for Amazon.com. In particular, for most of the security analysis we will do in this book, we will use the 3-year period from

---

[1] Yahoo Finance data is provided by third party data vendors that also provide fee-based data services used by investors. In particular, data on equities and ETFs are provided by CSI (http://www.csidata.com) and data on options are provided by Interactive Data Real-time Services (http://www.interactivedata. com).

[2] In addition, the Yahoo Finance data also provides us with an adjusted closing price variable, which allows us to calculate returns that incorporate the effects of dividends or what are sometimes referred to as *total returns*. We will use total returns in subsequent chapters, so using Yahoo Finance data allows us to gain familiarity with a single data source that we will use throughout this book.

December 31, 2010 to December 31, 2013. Using the same date range allows us to compare the results from our R program to the results presented in this book.

The following general steps lay out how to retrieve a CSV file of the historical data for AMZN. Depending on the browser we use, the steps or terms that appear on the screen may be slightly different.

1. On your web browser, **ENTER** the following website address: **http://finance. yahoo.com**.
2. **ENTER** the ticker symbol **AMZN** in the "Quote Lookup" or "Enter Symbol" box. This will take us to the page of the latest AMZN price quote.
3. On the page with the latest AMZN price quote, **CLICK** on the **Historical Prices** link. This is likely on the panel on the left-hand side of the screen.
4. This **Historical Prices** page allows us to set the date range for the data we want to download, which, for our purposes, we **ENTER** the date range December 31, 2010 to December 31, 2013.
5. When the correct date range is loaded, the first data in the table will December 31, 2013 (note that the data appears in reverse chronological order. **CLICK** on the **Last** link to verify the earliest data in the table is December 31, 2010.)
6. **RIGHT CLICK** the link at the bottom of the sheet that says **Download to Spreadsheet**, and choose the option to **Save target as...** or **Save Link As...** onto the R working directory with the label **AMZN Yahooo.csv**.To identify what the R working directory is, type the `getwd()` command at the command prompt in the R console. Place all R code we create and any data we need to call from within R in this R working directory.

The critical step above is No. 6, as deviating from that may result in confusion and errors while importing the CSV file. Note that the above approach saves the file directly *without* opening the CSV file in Excel. The reason for this is that Excel modifies the format of the date variable upon opening, which then makes the code below incompatible with the modified date format. If you have not downloaded Yahoo Finance data before, it may be tempting to open it at least for some files but not for others. Figure 1.1 shows a screen shot of the header row, the first five observations, and last five observations of the CSV file we retrieved. This is how the data would look like when viewed in Excel, so hopefully that may satisfy your curiosity. However, if we did open the CSV file in Excel and problems uploading the data occur, the easiest fix may be to re-download the data from Yahoo Finance. Alternatively, I describe in a note below how to modify the program to properly read the modified date format.

Tip to Download Yahoo Finance Data for Other Symbols
Since most examples from Yahoo Finance used in this book use the same date range of December 31, 2010 through December 31, 2013, we can retrieve *all* the equity and ETF data we need from Yahoo Finance in one step. After No. 6

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Volume	Adj Close
2	12/31/2013	394.58	398.83	393.8	398.79	1996500	398.79
3	12/30/2013	399.41	399.92	392.45	393.37	2487100	393.37
4	12/27/2013	404.65	405.63	396.25	398.08	1986900	398.08
5	12/26/2013	401.79	404.52	396.81	404.39	1868500	404.39
6	12/24/2013	402.52	403.72	396.37	399.2	1380400	399.2
7	12/23/2013	403.69	405	399.2	402.92	2659500	402.92
751	1/7/2011	187.88	188.45	183.74	185.49	5221700	185.49
752	1/6/2011	186.5	187.41	185.25	185.86	3179700	185.86
753	1/5/2011	184.1	187.45	184.07	187.42	3418800	187.42
754	1/4/2011	186.15	187.7	183.78	185.01	5031800	185.01
755	1/3/2011	181.37	186	181.21	184.22	5331400	184.22
756	12/31/2010	181.96	182.3	179.51	180	3451900	180

Fig. 1.1 Screenshot of CSV file of AMZN data retrieved from Yahoo Finance. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

above, we can find on the upper right side of the **Historical Prices** screen a box labeled **Get Historical Prices for:**. In that box, we **ENTER** the symbol of another security we will be using. This will keep us on the same page and allow us to download the new security's data *for the same date range*. Otherwise, we would have to go through Steps 1 through 6 again before we can pull the data we need. The other equity and ETF securities we will use in this book that use the same date range are as follows: IBM (IBM), S&P 500 Index (^GSPC), SPDR S&P 500 ETF (SPY), S&P 600 Small Cap ETF (SLY), SPDR Barclays Aggregate Bond ETF (LAG), SPDR Barclays High Yield Bond ETF (JNK), SPDR MSCI All Country World Index ex USA ETF (CWI), Tesla (TSLA), and Yahoo (YHOO). For our implementation of event studies, we use data for Netflix (NFLX) and SPDR S&P 500 ETF (SPY) from July 20, 2012 to July 23, 2013. All data retrieved from Yahoo Finance for the above securities, including data for AMZN, are reproduced with permission of CSI. Data Source: ©2013 CSI www.csidata.com.

We import the raw data from Yahoo Finance prior to doing any data manipulation. The reason for this is that we want to preserve the actual data from the source and any changes we make can be cataloged in the program we write. In this manner, we can keep track of what we did with the raw data so that we can replicate it at a later date. Moreover, this approach will help us identify any potential issues or errors and we can trace those problems as either a programming error or problem with the raw data. Yes, problems with data do exist unlike what typical textbook examples show you. Later in this chapter we demonstrate some techniques to help us spot some types of data problems.

> **Using the** R **Editor for Writing Programs**
>
> For convenience, we should use the R Editor to write and execute the codes in this book. The R Editor can be opened using the **CTRL + N** (Windows) or **Command + N** (OS X). More details about the R Editor can be found in Appendix A. We will type all the codes that appear in this text in the R Editor so the codes can easily be modified and re-run at any point in time.
>
> In the output below, lines that are preceded by a command prompt ($>$) or plus sign ($+$) are lines of code. We type those into the R editor without the command prompt and plus sign. The command prompt denotes the start of a line of code. Multiple lines of code that belong together but broken up into separate lines are identified by the lines that begin with a plus sign. Specifically, the first line of code will be the line that has the command prompt and subsequent but related lines of code will be the line or lines that start with the plus sign.
>
> We can highlight the portion of code we want to run and type **CTRL + R** (Windows) or **CTRL + ENTER** (OS X) to execute. The output in the R console when we run specific lines of code should be identical to the output we report in this text. Lines that do not begin with a command prompt or plus sign will be output of the code we run.

Step 2: Import Data into R To upload the CSV file into R, we will use the `read.csv` command.[3] From Fig. 1.1, we see the CSV file has a header row with variables. As such, we have to make sure we add the argument `header=TRUE` to the code so that R knows the first row should be read-in as variable names. We can then look at the first six observations of `data.AMZN` using the `head` command. The last six observations can be reported using the `tail` command.

```
> data.AMZN<-read.csv("AMZN Yahoo.csv",header=TRUE)
> head(data.AMZN)
        Date   Open   High    Low  Close  Volume Adj.Close
1 2013-12-31 394.58 398.83 393.80 398.79 1996500    398.79
2 2013-12-30 399.41 399.92 392.45 393.37 2487100    393.37
3 2013-12-27 404.65 405.63 396.25 398.08 1986900    398.08
4 2013-12-26 401.79 404.52 396.81 404.39 1868500    404.39
5 2013-12-24 402.52 403.72 396.37 399.20 1380400    399.20
6 2013-12-23 403.69 405.00 399.20 402.92 2659500    402.92
> tail(data.AMZN)
          Date   Open   High    Low  Close  Volume Adj.Close
750 2011-01-07 187.88 188.45 183.74 185.49 5221700    185.49
751 2011-01-06 186.50 187.41 185.25 185.86 3179700    185.86
752 2011-01-05 184.10 187.45 184.07 187.42 3418800    187.42
753 2011-01-04 186.15 187.70 183.78 185.01 5031800    185.01
754 2011-01-03 181.37 186.00 181.21 184.22 5331400    184.22
755 2010-12-31 181.96 182.30 179.51 180.00 3451900    180.00
```

---

[3] Note that terms in `teletype` font are R-related terms. These terms include commands, packages, or object names.

Comparing the output to Fig. 1.1, we can see that the data are identical except for the format of the date. When we import data, we should always make sure the data is uploaded correctly. Just because the data looks like it is correct, does not necessarily mean that it is correct. If the data we use is incorrect, any output of our analysis will also be incorrect (i.e., garbage-in garbage-out).

Step 3: Convert the date Variable from a `Factor` **to a** `Date` Although the `Date` variable from the output above looks like a date, R is not treating it as a date. Instead, R treats that variable as a `Factor`. Using the `class` command, we can show how R treats the `Date` variable. Note that to tell R we are only interested in `Date`, we added a dollar sign ($) after the data object name, i.e., `data.AMZN$Date`.

```
>class(data.AMZN$Date)
[1]"factor"
```

A `Factor` is also known as a *categorical variable*, which is a type of variable used to bucket data into groups. For example, if we want to bucket days into *weekdays* and *weekends*. We can have a categorical variable created that lets us know whether a particular day is a weekday or if it is a weekend.

Since we are not using `Date` as a categorical variable, we convert it to a variable with class `Date` using the `as.Date` function. When using the `as.Date` function, we have to look at how the values in `Date` are formatted to make sure we tell R how to read-in the data properly. From the output above, we can see the format is *year-month-day*, so we use the format `%Y-%m-%d`. The capital `Y` for the year lets R know it is a four-digit year. A small `y` means a two-digit year.[4]

```
> date<-as.Date(data.AMZN$Date,format="%Y-%m-%d")
> head(date)
[1] "2013-12-31" "2013-12-30" "2013-12-27" "2013-12-26" "2013-12-24"
[6] "2013-12-23"
> tail(date)
[1] "2011-01-07" "2011-01-06" "2011-01-05" "2011-01-04" "2011-01-03"
[6] "2010-12-31"
```

> **If Yahoo Finance Data is Saved into CSV from Within Excel**
> If we opened the CSV file of Yahoo Finance data and saved it, the format of the `Date` variable would be different from the one described above. It would follow the format shown in Fig. 1.1 and will be imported into R looking like the output below. Notice that the values of the `Date` variable are in **Month/Day/2-digit Year** format.

---

[4] Table A.1 in Appendix A at the end of this book contains a table of the various date formats in R and their corresponding symbols.

```
> head(data.AMZN)
      Date   Open   High    Low  Close  Volume Adj.Close
1 12/31/13 394.58 398.83 393.80 398.79 1996500    398.79
2 12/30/13 399.41 399.92 392.45 393.37 2487100    393.37
3 12/27/13 404.65 405.63 396.25 398.08 1986900    398.08
4 12/26/13 401.79 404.52 396.81 404.39 1868500    404.39
5 12/24/13 402.52 403.72 396.37 399.20 1380400    399.20
6 12/23/13 403.69 405.00 399.20 402.92 2659500    402.92
> tail(data.AMZN)
       Date   Open   High    Low  Close  Volume Adj.Close
750  1/7/11 187.88 188.45 183.74 185.49 5221700    185.49
751  1/6/11 186.50 187.41 185.25 185.86 3179700    185.86
752  1/5/11 184.10 187.45 184.07 187.42 3418800    187.42
753  1/4/11 186.15 187.70 183.78 185.01 5031800    185.01
754  1/3/11 181.37 186.00 181.21 184.22 5331400    184.22
755 12/31/10 181.96 182.30 179.51 180.00 3451900    180.00
```

When we re-format the `Date` variable from `Factor` class to a `Date` class,
we have to change the format to `%m/%d/%y`. This will allow `R` to read-in the
data correctly.

```
> date<-as.Date(data.AMZN$Date,format="%m/%d/%y")
> head(date)
[1] "2013-12-31" "2013-12-30" "2013-12-27" "2013-12-26" "2013-12-24"
[6] "2013-12-23"
> tail(date)
[1] "2011-01-07" "2011-01-06" "2011-01-05" "2011-01-04" "2011-01-03"
[6] "2010-12-31"
```

To check the class of the variable, we use the `class` command on the `date` object.
As we can see, this object is now of class `Date`.

```
> class(date)
[1] "Date"
```

Extra Care Must Be Taken When Using Real-World Data
At this early stage it is worth stressing that real-world data is a lot messier than
what is typically presented in textbooks. We must be careful when using such
data and have to know that we can use the data for our specific purpose. There
are many potential problems that could arise when dealing with real-world
data. As the example above shows, real-world data may come in one form
but depending on the way we save the data we may end up modifying it from
its raw form. Therefore, we must exercise a reasonable amount of care when
using real-world data

Step 3: Combining `date` **and** `data.AMZN` The next step would be to replace the date variable in `data.AMZN` with the `date`. We know that `data.AMZN` has six variables and hundreds of columns. One of these variables is the original `Date` variable that is a `Factor` (i.e., Column 1), so we do not really need that column anymore after we combine the data with `date`. We also know that we want `date` to come in as a column. Because we want to combine the two objects as columns, we use the `cbind` command. To remember this command, just think of the `c` as standing for "column."

```
> data.AMZN<-cbind(date, data.AMZN[,-1])
> head(data.AMZN)
        date   Open   High    Low  Close  Volume Adj.Close
1 2013-12-31 394.58 398.83 393.80 398.79 1996500    398.79
2 2013-12-30 399.41 399.92 392.45 393.37 2487100    393.37
3 2013-12-27 404.65 405.63 396.25 398.08 1986900    398.08
4 2013-12-26 401.79 404.52 396.81 404.39 1868500    404.39
5 2013-12-24 402.52 403.72 396.37 399.20 1380400    399.20
6 2013-12-23 403.69 405.00 399.20 402.92 2659500    402.92
> tail(data.AMZN)
          date   Open   High    Low  Close  Volume Adj.Close
750 2011-01-07 187.88 188.45 183.74 185.49 5221700    185.49
751 2011-01-06 186.50 187.41 185.25 185.86 3179700    185.86
752 2011-01-05 184.10 187.45 184.07 187.42 3418800    187.42
753 2011-01-04 186.15 187.70 183.78 185.01 5031800    185.01
754 2011-01-03 181.37 186.00 181.21 184.22 5331400    184.22
755 2010-12-31 181.96 182.30 179.51 180.00 3451900    180.00
```

Notice that to delete the original `Date` variable in `data.AMZN` we used [,-1]. We will discuss this technique in more detail below when we discuss subsetting data. For now, it is sufficient to know that the square brackets after the data object name allows us to identify rows (by entering terms before the comma) and columns (by entering terms after the comma) we either want to keep or delete. In this case, -1 implies we want to *delete* Column 1, which was the column number of the original `Date` variable in `data.AMZN`.

In addition, the order with which we apply `cbind` is important. This tells R which object goes on the left and which object goes on the right. Had we flipped the order above, we would have the `date` variable follow the `Adj.Close` variable. We can still call the variables by its variable name, but we will not be able to use the same column numbers to call the data. In R, there are instances when variable names can and cannot be used. Therefore, we have to be careful with the positioning of the variables going forward. For purposes of this book, the codes are written to match the output presented in the text. As such, it is best to make sure that the output you get matches the intermediate output reported in the text.

Step 4: Sort Data in Chronological Order Raw data imported from Yahoo Finance appears in reverse chronological order. Since it is more typical to work with data in chronological order, we *sort* the data in R using the `order` command. We want to re-sort the rows of `data.AMZN` using `date` as the sorting variable. Recall when

using the square brackets, anything to do with rows is to the left of the comma and anything to do with columns is to the right of the comma. Therefore, we type `[order(data.AMZN$date),]`.

```
> data.AMZN<-data.AMZN[order(data.AMZN$date),]
> head(data.AMZN)
          date   Open   High    Low  Close  Volume Adj.Close
755 2010-12-31 181.96 182.30 179.51 180.00 3451900    180.00
754 2011-01-03 181.37 186.00 181.21 184.22 5331400    184.22
753 2011-01-04 186.15 187.70 183.78 185.01 5031800    185.01
752 2011-01-05 184.10 187.45 184.07 187.42 3418800    187.42
751 2011-01-06 186.50 187.41 185.25 185.86 3179700    185.86
750 2011-01-07 187.88 188.45 183.74 185.49 5221700    185.49
> tail(data.AMZN)
        date   Open   High    Low  Close  Volume Adj.Close
6 2013-12-23 403.69 405.00 399.20 402.92 2659500    402.92
5 2013-12-24 402.52 403.72 396.37 399.20 1380400    399.20
4 2013-12-26 401.79 404.52 396.81 404.39 1868500    404.39
3 2013-12-27 404.65 405.63 396.25 398.08 1986900    398.08
2 2013-12-30 399.41 399.92 392.45 393.37 2487100    393.37
1 2013-12-31 394.58 398.83 393.80 398.79 1996500    398.79
```

Step 5: Convert from `data.frame` Object to `xts` Object `data.AMZN` is structured as a `data.frame` class.

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

`data.frame` is a flexible data class, and we will do some of our analyses when an object is a `data.frame`. However, since we are mainly dealing with financial time series, we will do a lot of our analyses also on data that is of `xts` class. `xts` stands for eXtensible Time Series and allows us to perform many functions useful when analyzing time series data (e.g., creating lag variables).[5]

To convert a `data.frame` into an `xts` object, we need to use the `xts` package. To load a package in R, we use the `library` command.

```
> library(xts)
Loading required package: zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric
```

[5] Jeffrey A. Ryan and Joshua M. Ulrich (2013). xts: eXtensible Time, Series. R package version 0.9-7. http://CRAN.R-project.org/package=xts.

Once the `xts` package has been loaded, we can now use the `xts` command to convert `data.AMZN` into an `xts` object. The `xts` command takes on two arguments that denote which are the data columns, which are Columns 2–7, and which is the index column. The data columns are the columns of `data.AMZN` that contains the data. The index column will be the `date` column that will be used as an index in a time series, which is in Column 1. The index column is preceded by the argument `order.by=` that means we want the order of the data to follow the `date` column.

```
> data.AMZN<-xts(data.AMZN[,2:7],order.by=data.AMZN[,1])
> head(data.AMZN)
             Open   High    Low  Close  Volume Adj.Close
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
2011-01-05 184.10 187.45 184.07 187.42 3418800    187.42
2011-01-06 186.50 187.41 185.25 185.86 3179700    185.86
2011-01-07 187.88 188.45 183.74 185.49 5221700    185.49
> tail(data.AMZN)
             Open   High    Low  Close  Volume Adj.Close
2013-12-23 403.69 405.00 399.20 402.92 2659500    402.92
2013-12-24 402.52 403.72 396.37 399.20 1380400    399.20
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
2013-12-31 394.58 398.83 393.80 398.79 1996500    398.79
>
> class(data.AMZN)
[1] "xts" "zoo"
```

As the output shows, `data.AMZN` is now an `xts` object. The output also shows that this object is also a `zoo` object. An `xts` object is a variant of a `zoo` object, so `xts` and `zoo` share many of the same characteristics.[6]

Step 6: Rename Variables We then rename the variables to conform to the variable names obtained from the `getSymbols` command. These variables names will be preceded by the ticker symbol (i.e., AMZN) and will be Open, High, Low, Close, Volume, and Adjusted. In R, we can call the variable names of an object using the `names` command.

```
> names(data.AMZN)
[1] "Open"      "High"      "Low"       "Close"     "Volume"    "Adj.Close"
```

Using the `names` command, we can make the variable names a target of whatever command we want to implement. For renaming variable names, the command we want to implement is the `paste` command, which tells R the variable names in `data.AMZN` will be getting new value. We then use the `c(...)` operator to string

---

[6] Achim Zeileis and Gabor Grothendieck (2005). zoo: S3 Infrastructure for Regular and Irregular Time Series. Journal of Statistical Software, 14(6), 1–27. URL http://www.jstatsoft.org/v14/i06/.

together the six new variable names, where each variable name is in double quotation marks and the six variables are separated by comma.

```
> names(data.AMZN)<-
+    paste(c("AMZN.Open","AMZN.High","AMZN.Low",
+    "AMZN.Close","AMZN.Volume","AMZN.Adjusted"))
> head(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
2011-01-05    184.10    187.45   184.07     187.42     3418800        187.42
2011-01-06    186.50    187.41   185.25     185.86     3179700        185.86
2011-01-07    187.88    188.45   183.74     185.49     5221700        185.49
> tail(data.AMZN)
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2013-12-23    403.69    405.00   399.20     402.92     2659500        402.92
2013-12-24    402.52    403.72   396.37     399.20     1380400        399.20
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
2013-12-31    394.58    398.83   393.80     398.79     1996500        398.79
```

We now have the Amazon.com stock data structure we will use. In later instances of using the same technique to import data, we will suppress the intermediate output to save space. Once we are able to get this code to work, we only need to change the symbols and data object names and we should not encounter any problems executing the code.

Retrieving Yahoo Finance Data Directly Using `getSymbols`
Although not recommended, there are alternative ways to import Yahoo Finance into R. One approach is to use the `getSymbols` command in the `quantmod` package. Since some people may prefer to use this approach, the code above to import the Yahoo Finance CSV file is designed to match the output of this alternative approach.

   To retrieve the Amazon.com data for the period December 31, 2010 to December 31, 2013 from Yahoo Finance using the `getSymbols` command, we can use the following code:

```
> library(quantmod)
>
> alt.data.AMZN<-getSymbols("AMZN",from="2010-12-31",
+    to="2013-12-31",auto.assign=FALSE)
>
> class(alt.data.AMZN)
[1] "xts" "zoo"
```

The argument `auto.assign=FALSE` is important because you will not be able to see any output when you use the `head` or `tail` commands if you do not include this string. The `class` command verifies that the class of the data object is also `xts` and `zoo`, just like the output in Step 5 above.

Finally, if you use the `head` and `tail` commands on `alt.data.AMZN`, you will get identical results as our preferred approach.

1.3   Checking the Data

Since we are using real-world data, we have to perform at least some basic tests to make sure we have correct and complete data. Although not comprehensive, this section discusses some simple checks we can implement to make sure that the data is not bad on its face. We do not need to do all these tests for every single data series we use nor do we have to do these checks in order. Even though we have been using a data source regularly, it is still good practice to perform some basic tests on the data to ensure its continued reliability or to identify potential peculiarities with specific data series that we may not have been aware of.

1.3.1   Plotting the Data

The most basic test we need to perform is to make sure we have complete data. One of the simplest things we can do to test this is to do a quick and dirty chart of the price data. Since this is an internal test, which will not be shared as part of a presentation, there is no need to spend time making this chart pretty. To chart the data, we use the `plot` command. To plot the Amazon.com closing price (`AMZN.Close`), we use the following code:

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

Figure 1.2 shows a quick and dirty chart of the AMZN closing price from our data. This chart does not show anything obviously suspicious with the AMZN closing price data. There appears to be trades everyday as the stock price bounces up and down with no obvious flat spots. Note that the Yahoo Finance data excludes trading holidays and weekends, so there should be no visible flat spots when we are looking at 3 years of daily data, which would be a sign of missing observations and that data is interpolated over these missing data points.

As an exercise to show how plotting the closing price can be helpful, let us construct a hypothetical data series of Amazon.com prices in which we delete observation numbers 400–500. To identify a range of observations, we use a colon (:) between

data.AMZN$AMZN.Close
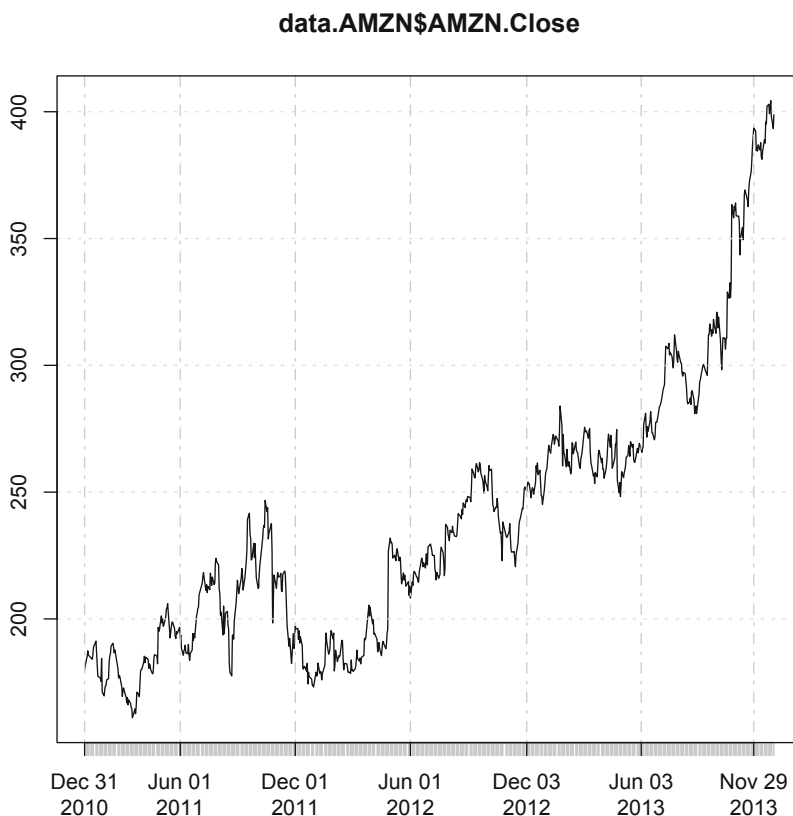
Fig. 1.2 AMZN stock price, 12/31/10 - 12/31/13. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

the two numbers. Since we are *deleting* rows, we put a negative sign in front of the numbers (i.e., [−400 : −500,]). Also, since we are deleting *rows*, the arguments we enter are to the left of the comma inside the square brackets.

```
> data.missing<-data.AMZN[-400:-500,]
> plot(data.missing$AMZN.Close)
```

We now plot a quick and dirty chart of `data.missing`, which is shown in Fig. 1.3. We can immediately see from this chart that there is a straight line in the middle of the chart, which is very suspicious for a highly-traded stock like Amazon.com. Had this outcome been observed from a raw data pull, this would have been a signal that we should investigate the raw data to determine whether the source files itself did not contain the observations or we should look for the causes as to why such data became missing (i.e., did we inadvertently delete the data).

data.missing$AMZN.Close
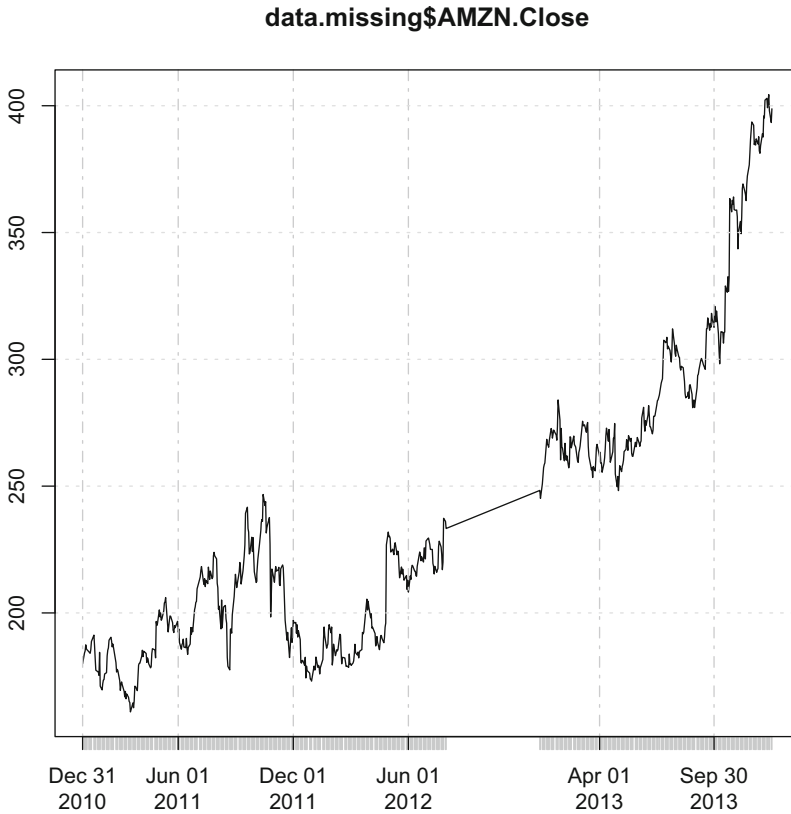
Fig. 1.3 AMZN stock price after removing 100 observations in the middle of the period. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

1.3.2   Checking the Dimension

We can learn a lot about the completeness of the data also by looking at the number of columns and rows or the *dimension* of the object. For Yahoo Finance data, the number of columns tells us how many variables are in the data, which we know should consist of six variables. If the number of variables is not equal to six, then we should investigate what the additional or missing variables are. In a time series, the number of rows represent the dates of the observation. Most stock price data will show only trading days. Therefore, a good rule of thumb is to assume approximately 252 trading days in a year and we can use this estimate to determine how much data we expect to observe over our entire date range. In our example, we are using 3 years of data. This means that we should expect to see 756 [= 252 trading days * 3 years] observations.

In R, we use the `dim` command to output the dimension of the object. The output below shows we have 755 rows (the number on the left in the output), which is one less than the 756 rows we expected. Note, however, that 252 trading days in a year is merely an approximation and the actual number of trading days in a particular year fluctuates primarily due to when holidays fall. In other words, having one less observation than what we expected would not in and of itself raise a red flag. As for the number of columns (the number on the right in the output), the output shows we have six columns in the data object, which is the exact number we expected.

```
> dim(data.AMZN)
[1] 755    6
```

1.3.3 Outputting Summary Statistics

Another useful check is to look at the *summary statistics* of the data. Using the `summary` command, R can output for each variable in the data object that variable's minimum and maximum values, the interquartile range (25th percentile and 75th percentile), mean, and median. These metrics give a sense of whether there is anything out of the ordinary with the data. For example, a negative price or volume or an absurdly high price or volume may be a sign that there is something odd with the data and that we should investigate further.

```
> summary(data.AMZN)
     Index               AMZN.Open       AMZN.High       AMZN.Low
 Min.   :2010-12-31   Min.   :161.2   Min.   :163.5   Min.   :160.6
 1st Qu.:2011-09-29   1st Qu.:192.6   1st Qu.:195.4   1st Qu.:190.2
 Median :2012-06-29   Median :226.6   Median :230.7   Median :224.6
 Mean   :2012-06-30   Mean   :238.2   Mean   :241.0   Mean   :235.2
 3rd Qu.:2013-04-03   3rd Qu.:266.7   3rd Qu.:269.4   3rd Qu.:263.8
 Max.   :2013-12-31   Max.   :404.6   Max.   :405.6   Max.   :399.2
   AMZN.Close       AMZN.Volume       AMZN.Adjusted
 Min.   :161.0   Min.   :  984400   Min.   :161.0
 1st Qu.:193.3   1st Qu.: 2660750   1st Qu.:193.3
 Median :227.3   Median : 3706200   Median :227.3
 Mean   :238.3   Mean   : 4319401   Mean   :238.3
 3rd Qu.:266.4   3rd Qu.: 5160950   3rd Qu.:266.4
 Max.   :404.4   Max.   :24134200   Max.   :404.4
```

1.3.4 Checking the Ticker Symbol

The ticker symbol for a company can be a tricky issue for several reasons. First, the ticker symbol is not a unique identifier that is constant through time. Firms change ticker symbols for various reasons and old ticker symbols get recycled from

one company to another. Second, ticker symbols may not be derived from the most obvious variation of the company name. For example, the technology company Hewlett-Packard is often called "HP" but the company's ticker symbol is actually "HPQ." If we blindly plugged in "HP" we would be obtaining data for an oil and gas drilling company Helmerich & Payne. Third, ticker symbols may not be the same across data sources. For example, the ticker symbol for the S&P 500 Index on Yahoo Finance is "^GSPC" but it is "SPX" on Bloomberg. Inputting "SPX" on Yahoo Finance pulls up data for SPX Corporation, which has ticker symbol "SPW."

Although checking for the correct ticker symbol is often neglected, using the wrong ticker symbol in the analysis will clearly result in incorrect results. Therefore, it is good practice to make sure we visually check the data we observe from trusted data sources that provide the same data. For example, many publicly-traded firms have an Investor Relations section on their website in which stock prices are sometimes reported or obtainable. We can then validate the data we pull with what the company reports. Alternatively, we can pull the data from another source or look at the firm's SEC filings, which may give you some data on the performance of the firm's stock price over some period of time.

1.4   Basic Data Manipulation Techniques

Importing raw data often means reading in more data than you need. This could be in the form of too many observations or too many variables. In our example of Amazon.com stock data obtained from Yahoo Finance, data.AMZN contains 755 observations with six variables. In some applications, we would not need all 755 observations or six variables. Having extra observations or variables sometimes makes the analysis messier. As such, we may want to manipulate the data in order for us to end up with only the data we need for the specific application we have in mind.

1.4.1   Keeping and Deleting One Row

Subsetting data in R requires the addition of square brackets after the data object name (i.e., data.AMZN[,]). The terms inside the square bracket are separated by a comma. To the left of the comma are arguments we use to control the rows of the data and to the right of the comma are argument we use to control the columns of the data.

To keep or delete one row, we place the row number to the left of the comma inside the square brackets. A negative sign in front of the row number will tell R to delete that row. For example, if we want to keep only the first row, we type [1,]. If we want to delete the first row, we type [−1,].

Suppose we want to create a data object called `AMZN.onlyFirst` that contains information on the first observation in the `data.AMZN`. A potential application of this is when we want to know the stock price on the first day of our period. In our example, this would mean keeping the data for December 31, 2010. We would then want to type `data.AMZN[1,]`. As the output of `AMZN.onlyFirst` confirms, there is only one observation in the data and that is for December 31, 2010.

```
> AMZN.onlyFirst<-data.AMZN[1,]
> AMZN.onlyFirst
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2010-12-31    181.96     182.3   179.51        180     3451900           180
```

Now, suppose we want to chart the data from January 2011 through December 2013 and need to drop the first observation, which is the data for December 31, 2010. To do this, we would type `data.AMZN[-1,]`. Then, using the `head` command to look at the first six observations of `AMZN.delFirst`, we see that the data now begins on January 3, 2011, which is the first trading day in 2011.

```
> AMZN.delFirst<-data.AMZN[-1,]
> head(AMZN.delFirst)
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
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
2011-01-05    184.10    187.45   184.07     187.42     3418800        187.42
2011-01-06    186.50    187.41   185.25     185.86     3179700        185.86
2011-01-07    187.88    188.45   183.74     185.49     5221700        185.49
2011-01-10    185.04    185.29   182.51     184.68     3375900        184.68
```

1.4.2   Keeping First and Last Rows

In some instances, we may want to know what the first and last observation of the data object is. For example, if we want to see the change in Amazon.com's stock price in the 3-year period from December 31, 2010 to December 31, 2013, we do not need to look at all 755 observations but only the first and last observation. To implement this, we need to use the `c(...)` operator. We can put as many row numbers inside the `c(...)` operator as long as we separate those row numbers by commas. As shown above, the first row is identified by a 1. However, to identify the last row of the data, we could type 755 because we already know how many observations in total `data.AMZN` has. However, we can also use the `nrow` command to return the number of rows in any data object. As such, we type `c(1,nrow(data.AMZN))` to the left of the comma and `R` will interpret this as to keep the first and the last row.

```
> data.AMZN[c(1,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
2013-12-31    394.58    398.83   393.80     398.79     1996500        398.79
```

1.4.3   Keeping Contiguous Rows

In the prior example, we kept the first and last rows, which are likely to be observations that are far apart. However, suppose we want to output the observations in the first week of January 2011, which would be the second through sixth row. We can use the c(...) operator and type [c(2,3,4,5,6),]. Although this may be workable for five observations, this may not be practical for much larger number of observations.

To identify consecutive rows, we can use a colon (:) to tell R that you want to keep all observations between the row number to the left of the colon and the row number to the right of the colon. In this case, we would type [2:6,].

```
> AMZN.first.week<-data.AMZN[2:6,]
> AMZN.first.week
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
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
2011-01-05    184.10    187.45   184.07     187.42     3418800        187.42
2011-01-06    186.50    187.41   185.25     185.86     3179700        185.86
2011-01-07    187.88    188.45   183.74     185.49     5221700        185.49
```

We can also combine multiple commands together for more complicated subsetting applications. Suppose we want to run a volume-weighted average price (VWAP) calculation over the last 30 closing prices in the data. We know that we have to use a colon (:) to denote that we are going to subset a block of rows. To the left of the colon, we type ((nrow(data.AMZN)-29)). The nrow command returns the number of rows in the data and then we subtract 29 from that number to get the 30th trading day from the end of 2013. The dimension of the output below confirms that subtracting 29 gives us a total of 30 observations. To the right of the colon, we type nrow(data.AMZN), which returns the total number of observations in the data. Combining the two results in the last 30 observations. This whole argument must then be entered to the left of the comma inside the square brackets.

```
> AMZN.last30<-data.AMZN[((nrow(data.AMZN)-29)):nrow(data.AMZN),]
> AMZN.last30
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
2013-11-18    370.28    373.49   364.67     366.18     2737400        366.18
2013-11-19    365.82    368.78   362.50     364.94     1882000        364.94
2013-11-20    367.56    367.56   360.45     362.57     1771500        362.57
2013-11-21    364.05    369.25   363.30     368.92     1964600        368.92
2013-11-22    370.00    374.50   366.31     372.31     2965500        372.31
2013-11-25    373.82    377.79   373.18     376.64     2972300        376.64
2013-11-26    377.61    382.50   374.82     381.37     2724400        381.37
2013-11-27    383.50    387.00   382.61     386.71     2267600        386.71
2013-11-29    389.10    394.10   388.62     393.62     2406000        393.62
2013-12-02    399.00    399.00   389.10     392.30     4714000        392.30
2013-12-03    390.11    390.95   383.10     384.66     3702900        384.66
2013-12-04    383.50    389.69   381.49     385.96     2355300        385.96
2013-12-05    386.65    386.65   381.37     384.49     1874400        384.49
2013-12-06    388.35    388.35   383.83     386.95     1984700        386.95
2013-12-09    388.11    388.21   382.57     384.89     2761800        384.89
2013-12-10    383.74    389.06   383.02     387.78     2736800        387.78
2013-12-11    387.34    388.98   382.00     382.19     2436000        382.19
2013-12-12    381.26    385.00   379.50     381.25     2122500        381.25
2013-12-13    385.32    389.42   383.80     384.24     3025000        384.24
2013-12-16    385.03    391.70   385.00     388.97     2251700        388.97
2013-12-17    390.65    391.36   386.50     387.65     2343900        387.65
2013-12-18    389.23    396.30   383.10     395.96     3489100        395.96
2013-12-19    394.27    397.29   392.60     395.19     2427200        395.19
2013-12-20    396.55    404.72   395.78     402.20     5033900        402.20
2013-12-23    403.69    405.00   399.20     402.92     2659500        402.92
2013-12-24    402.52    403.72   396.37     399.20     1380400        399.20
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
2013-12-31    394.58    398.83   393.80     398.79     1996500        398.79
> nrow(AMZN.last30)
[1] 30
```

1.4.4   Keeping First Three Rows and Last Row

In this book, we will show intermediate data output to guide us when implementing the models. We could use the `head` and `tail` commands, but that would unnecessarily use up too much space as each command would generate six observations. As an alternative, we will show the first three observations and the last observation of the data.

The reason for this is that the first and last observation tells us the range of dates we use. Then, the first three observations collectively tell us, in most cases, the frequency of the data (e.g., daily, weekly, monthly). Please note that from this point forward, we show Yahoo Finance output using this technique when possible and as long as we do not lose any relevant information in doing so.

To implement this, we need to use the `c(...)` operator because we are stringing together two things. First, we are saying we want to keep rows 1–3 (i.e., 1:3). Next, we are saying we also want to keep the last row (i.e., `nrow(data.AMZN)`).

```
> 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
```

Note that the above data shows that we are using daily stock price data. January 1 and 2, 2011 are weekends and are, therefore, non-trading days.

1.4.5 Keeping and Deleting One Column

Now we turn to showing examples of subsetting the columns of our data. For our purposes, columns represent variable names. Using the `names` command, we can see what the names of the variables are. Notice there is a [1] and [5] on the left-side of the output. These represent the variable number of the first observation on each line. How many variable names will show up on each line depends on how wide the R console is. The wider the R console, the more variables we can fit in one line. Therefore, what we observe in our R console may be different from what appears in the output below even though the variable names should be identical. In any case, the total number of variables should be six as there should be six columns in `data.AMZN` for us to work with.

```
> names(data.AMZN)
[1] "AMZN.Open"     "AMZN.High"     "AMZN.Low"      "AMZN.Close"
[5] "AMZN.Volume"   "AMZN.Adjusted"
```

As an example of keeping one column, suppose we want to plot only the AMZN closing price. We can do so by typing in `[,4]`. Note that since we are subsetting columns, we enter 4 to the right of the comma.

```
> AMZN.onlyPrice<-data.AMZN[,4]
> AMZN.onlyPrice[c(1:3,nrow(AMZN.onlyPrice)),]
           AMZN.Close
2010-12-31     180.00
2011-01-03     184.22
2011-01-04     185.01
2013-12-31     398.79
```

An alternative way to call one column is, instead of the square brackets following the data object name, we specify its variable name preceded by a dollar sign ($). For example, the closing price is `AMZN.Close` so we type `data.AMZN$AMZN.Close`. The output below shows we get identical results to identifying the variable by its column number.

```
> AMZN.onlyPrice2<-data.AMZN$AMZN.Close
> AMZN.onlyPrice2[c(1:3,nrow(AMZN.onlyPrice2)),]
           AMZN.Close
2010-12-31     180.00
2011-01-03     184.22
2011-01-04     185.01
2013-12-31     398.79
```

We can now look at an example of when we want to delete one column. Suppose we want to create an open, high, low, close (OHLC) price chart with volume (we will implement this later in this chapter). The sixth observation in our data is the adjusted close, which we will not need for such an application. Therefore, we can put a *negative* sign in front of the column number to *delete* the specific column. That is, we type `[,-6]`.

```
> AMZN.delAdjPrice<-data.AMZN[,-6]
> AMZN.delAdjPrice[c(1:3,nrow(AMZN.delAdjPrice)),]
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume
2010-12-31    181.96    182.30   179.51     180.00     3451900
2011-01-03    181.37    186.00   181.21     184.22     5331400
2011-01-04    186.15    187.70   183.78     185.01     5031800
2013-12-31    394.58    398.83   393.80     398.79     1996500
```

1.4.6 Keeping Non-Contiguous Columns

In most applications, we likely will need more than one variable. Suppose we want to compare the daily open and close prices for AMZN. We would then need to keep the open price (Column 1) and close price (Column 4). We then use the `c(...)` operator to combine the two (i.e., `[,c(1,4)]`).

```
> AMZN.OpenClose<-data.AMZN[,c(1,4)]
> AMZN.OpenClose[c(1:3,nrow(AMZN.OpenClose)),]
           AMZN.Open AMZN.Close
2010-12-31    181.96     180.00
2011-01-03    181.37     184.22
2011-01-04    186.15     185.01
2013-12-31    394.58     398.79
```

1.4.7 Keeping Contiguous Columns

Sometimes we may want to keep variables that are next to each other. For example, suppose we want to analyze Amazon's daily closing price (Column 4) and volume (Column 5). We can use the colon (:) to denote that these columns are contiguous (i.e., `[,4:5]`).

```
> AMZN.PriceVol<-data.AMZN[,4:5]
> AMZN.PriceVol[c(1:3,nrow(AMZN.PriceVol)),]
           AMZN.Close AMZN.Volume
2010-12-31     180.00     3451900
2011-01-03     184.22     5331400
2011-01-04     185.01     5031800
2013-12-31     398.79     1996500
```

1.4.8 Keeping Contiguous and Non-Contiguous Columns

In some instances, we may end up wanting to keep several columns that are not all contiguous but some are. To do this, we use the `c(...)` operator. For example, suppose we want to keep the daily open price (Column 1), close price (Column 4), and volume (Column 5). Therefore, to the right of the comma we can type `c(1,4:5)`.

```
> AMZN.OpenCloseVol<-data.AMZN[,c(1,4:5)]
> AMZN.OpenCloseVol[c(1:3,nrow(AMZN.OpenCloseVol)),]
           AMZN.Open AMZN.Close AMZN.Volume
2010-12-31     181.96     180.00     3451900
2011-01-03     181.37     184.22     5331400
2011-01-04     186.15     185.01     5031800
2013-12-31     394.58     398.79     1996500
```

Alternatively, we can replicate the results above by deleting the columns we do not want (i.e., deleting Columns 2, 3, and 6).

```
> AMZN.OpenCloseVol<-data.AMZN[,c(-2:-3,-6)]
> AMZN.OpenCloseVol[c(1:3,nrow(AMZN.OpenCloseVol)),]
           AMZN.Open AMZN.Close AMZN.Volume
2010-12-31     181.96     180.00     3451900
2011-01-03     181.37     184.22     5331400
2011-01-04     186.15     185.01     5031800
2013-12-31     394.58     398.79     1996500
```

The choice of whether to keep rows that we want or to delete rows we do not want is a matter of preference and, sometimes, convenience. For example, if we have data with many variables and we would like to delete more than half of the variables, choosing to keep the variables we need may be more practical. Conversely, if more than half of the variables would be retained, choosing to delete variables may be more practical.

1.4.9 Subsetting Rows and Columns

So far we have either subset rows only or subset columns only. It is likely that in many applications we would have to subset both rows and columns. For example, suppose we want to calculate the volume-weighted average price (VWAP) over the last 30 trading days of 2013. What we would need in this analysis is the last 30 days of data in 2013 and AMZN's close price (Column 4) and volume (Column 5). We essentially are going to combine what we did when we created `AMZN.last30` and `AMZN.PriceVol` above. We put the former on the left side of the comma and the latter on the right side of the comma.

```
> data.vwap<-data.AMZN[((nrow(data.AMZN)-29)):nrow(data.AMZN),c(4,5)]
> data.vwap[c(1:3,nrow(data.vwap)),]
           AMZN.Close AMZN.Volume
2013-11-18     366.18     2737400
2013-11-19     364.94     1882000
2013-11-20     362.57     1771500
2013-12-31     398.79     1996500
```

Consistent with `AMZN.last30`, we only have data from November 18, 2013 to December 31, 2013. Consistent with `AMZN.PriceVol`, we only show the `AMZN.Close` and `AMZN.Volume` variables.

1.4.10 Subsetting Using Dates

In many financial applications, we will deal with *time series* data. In this particular case, the term *time series* is used in the more general sense as in data that can be indexed by some time interval, such as daily, weekly, monthly, quarterly, or annual. As such, it is often easier to subset data using dates as these are easier to remember as they hold some tangible meaning. For example, it would be easier to use dates to subset data in 2012 rather than figuring out we need to keep rows 254–503 in `data.AMZN`. Moreover, the latter date range may not be applicable if our data does not start on December 31, 2010.

Consider the case that we want to chart AMZN stock price in 2012. We take two general approaches in this book, depending on the application. The approach depends on whether the data is an `xts` object or a `data.frame` object. Recall that the starting point of our Yahoo Finance data will be an `xts` object. However, when we perform some analyses, we would need to convert the data into a `data.frame`. As such, showing how to subset using dates under both classes of data will be useful going forward.

Data is an `xts` **Object** The main command we use to subset the data is the `subset` command. For an `xts` object, we identify the date using the `index` command as the date is in the index of the `xts` object. Since we are only concerned with plotting

the closing stock price in this example, we also limit the data we subset to only Column 4.

```
> class(data.AMZN)
[1] "xts" "zoo"
> xts.2012<-subset(data.AMZN[,4],
+    index(data.AMZN) >= "2012-01-01" &
+    index(data.AMZN) <= "2012-12-31")
> xts.2012[c(1:3,nrow(xts.2012))]
          AMZN.Close
2012-01-03    179.03
2012-01-04    177.51
2012-01-05    177.61
2012-12-31    250.87
```

Data is a `data.frame` **Object** Before we can show how to subset the data when we have a data frame, we have to first convert the data into a `data.frame` object. One of the reasons why we would want to convert the data into a `data.frame` object is because we have a workable column of dates. The workable column of dates is created by applying the `index` command to `data.AMZN`. Before we can combine the workable column of dates with the close price, we need to convert the close price into a `data.frame` object also. Then, we can combine the two data objects using the `cbind` command.

```
> AMZN.2012<-cbind(index(data.AMZN),
+    data.frame(data.AMZN[,4]))
> AMZN.2012[c(1:3,nrow(AMZN.2012)),]
          index(data.AMZN) AMZN.Close
2010-12-31       2010-12-31     180.00
2011-01-03       2011-01-03     184.22
2011-01-04       2011-01-04     185.01
2013-12-31       2013-12-31     398.79
> class(AMZN.2012)
[1] "data.frame"
```

The output shows this is now a `data.frame` object. Note that the first two lines of the output above (i.e., the first line starts with a command prompt and the second line starts with a plus sign) are considered one line of code. We split the code into two separate lines to make it easier to read from a programming standpoint. As we can see, the first argument in `cbind` command is in the first line. The comma that ends the first line splits the two arguments of the `cbind` command. The second argument is in the next line. We could also use one line of code and the result should be the same, but splitting up long or complicated lines of code into multiple lines makes it easier to follow.

There are two issues with `AMZN.2012` above. First, the variable name for the date is not meaningful and too hard to use on a regular basis. Second, we already have a date variable, so the index of dates is redundant. Therefore, we would want to rename the first variable and then change the index into an identifier for the observation number. Since we are only renaming the first variable, we can add `[1]`

after we apply the `names` command to `AMZN.2012`. To change the index, we use the `rownames` command. What we substitute for the dates is a sequence of numbers using the `seq` command. The `seq` command takes on three arguments, which are separated by commas. The first argument is the starting value, the second argument is the ending value (i.e., the number of rows of `AMZN.2012`), and the third argument is the increment (i.e., an increment of 1 means that the sequence goes up by 1 until it reaches the 755, the number of rows in `AMZN.2012`).

```
> names(AMZN.2012)[1]<-paste("date")
> rownames(AMZN.2012)<-seq(1,nrow(AMZN.2012),1)
> AMZN.2012[c(1:3,nrow(AMZN.2012)),]
          date AMZN.Close
1   2010-12-31     180.00
2   2011-01-03     184.22
3   2011-01-04     185.01
755 2013-12-31     398.79
```

Now we are ready to subset the data. We also use the `subset` command. However, we now explicitly call the `date` variable using the dollar sign ($) (i.e., `AMZN.2012$date`) instead of using `index(AMZN.2012)`. As we can see, when we subset the data, the index values still maintain the original index values. Therefore, we know that the data in 2012 is from observation number 254–503 of the original data.

```
> AMZN.2012<-subset(AMZN.2012,
+    AMZN.2012$date >= "2012-01-01"&
+    AMZN.2012$date <= "2012-12-31")
> AMZN.2012[c(1:3,nrow(AMZN.2012)),]
          date AMZN.Close
254 2012-01-03     179.03
255 2012-01-04     177.51
256 2012-01-05     177.61
503 2012-12-31     250.87
```

1.4.11 Converting Daily Prices to Weekly and Monthly Prices

The data we downloaded from Yahoo Finance was daily stock price data. There are certain applications that would require us to convert to daily data to data of lesser frequency, such as weekly or monthly data. As an `xts` object, we can easily convert the data in `data.AMZN` to weekly data or monthly data.

Converting to Weekly Prices To convert data into weekly data, we use the `to.weekly` command. Note that we created a new data object `wk`. The reason for this is the `to.weekly` command takes the entire name of the data object and puts it in the variable name. As the output shows, the variable names of `data.weekly` all have the prefix `wk`.

```
> wk<-data.AMZN
> data.weekly<-to.weekly(wk)
> data.weekly[c(1:3,nrow(data.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
```

Note that the December 31, 2010 values are only based on one day, December 31, 2010. This is not a "true" weekly data unlike the rest of of the observations.

   To check that the data was actually converted properly by the `to.weekly` command, let us look closer at the details of the first week of January 2011, which is the second observation and is the week that ends on January 7, 2011. The data below, which shows the output from the original data, shows that the open price of the week was $ 181.37 on January 3, 2011 and the close/adjusted close prices at the end of the week was $ 185.49 on January 7, 2013. The high price for the week is the January 7, 2011 high price of $ 188.45 and the low price for the week is the January 3, 2011 low price of $ 181.21. The last line of code confirms the volume for the week is the sum of the volume for the entire week of 22.183 million shares traded.

```
> data.AMZN[2:6,]
           AMZN.Open AMZN.High AMZN.Low AMZN.Close AMZN.Volume AMZN.Adjusted
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
2011-01-05    184.10    187.45   184.07     187.42     3418800        187.42
2011-01-06    186.50    187.41   185.25     185.86     3179700        185.86
2011-01-07    187.88    188.45   183.74     185.49     5221700        185.49
> sum(data.AMZN[2:6,5])
[1] 22183400
```

Converting to Monthly Prices To convert data into monthly data, we use the `to.monthly` command. The rest of the code is similar to that of the conversion to weekly data. Again, we should ignore the December 2010 observation because that is not true monthly data and is based solely on a single observation in December 2010.

```
> mo<-data.AMZN
> data.monthly<-to.monthly(mo)
> data.monthly[c(1:3,nrow(data.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 current purpose, this should not affect our results and could be ignored.

Yahoo Finance Weekly and Monthly Volume is the Average Volume for the Period and Not the Total Volume for the Period
There is an option to download weekly or monthly data from Yahoo Finance. However, the weekly and monthly volume data reported on Yahoo Finance is the average volume for the period. That is, Yahoo Finance reports the average weekly volume or average monthly volume when we choose to download Yahoo Finance data of a lesser frequency. Unfortunately, we cannot simply scale up the average volume by a fixed number as the number of trading days is not constant for each week or each month due to holidays.

Plotting a Candlestick Chart Using Monthly Data   One common way of presenting weekly or monthly data is to use a *candlestick chart*. The `chartSeries` function has a variety of built-in charting options. One of those options is to generate a candlestick chart. However, we would need to first convert the data into an open-high-low-close (OHLC) object. To do this, we have to load the `quantmod` package.[7].

```
> library(quantmod)
Loading required package: Defaults
Loading required package: TTR
Version 0.4-0 included new data defaults. See ?getSymbols.
```

Recall that the first time we load a package, a series of messages may pop up. This is normal.

We then convert the data to class `OHLC` using the `as.quantmod.OHLC` command. We have to tell R what the column names in the data are. Note that to avoid any confusion, we dropped the adjusted close from `data.monthly` when we started as well as the December 2010 observation.

---

[7] Jeffrey A. Ryan (2013). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-0. http://CRAN.R-project.org/package=quantmod.

```
> OHLC<-data.monthly[-1,-6]
> AMZN.ohlc<-as.quantmod.OHLC(OHLC,
+     col.names=c("Open","High","Low","Close","Volume"))
> class(AMZN.ohlc)
[1] "quantmod.OHLC" "zoo"
>
> AMZN.ohlc[c(1:3,nrow(AMZN.ohlc)),]
         OHLC.Open OHLC.High OHLC.Low OHLC.Close OHLC.Volume
Jan 2011    181.37    191.60   166.90     169.64   113611300
Feb 2011    170.52    191.40   169.51     173.29    95776400
Mar 2011    173.53    181.57   160.59     180.13   118979100
Dec 2013    399.00    405.63   379.50     398.79    55638100
```

Comparing the output of `AMZN.ohlc` with that of `data.monthly`, the only visible difference from the two outputs are the renaming of the variable names and the date variable.

   We now plot the OHLC data using the `chartSeries` function. The default theme for the plot is a black background with orange and green bars. However, this type of output may look better on a monitor but not when presented in other formats. As such, we will change the `theme` to `white.mono`, such that we get a white background with black bars. Notice that in `chartSeries`, the title of the chart is controlled by the `name` argument. In contrast, recall that when using the `plot` command, the title of the chart is controlled by the `main` argument.

```
> chartSeries(AMZN.ohlc,
+     theme="white.mono",
+     name="AMZN OHLC")
```

The candlestick chart output is shown in Fig. 1.4. A candlestick chart shows us that there are some months in which the high and low prices are within a narrow range, such as May 2011 in which AMZN's price traded within a $ 15.51 range between $ 190.88 and 206.39, whereas the range is wide in other months, such as November 2013 in which AMZN's price traded within a $ 52.22 range between $ 341.88 and 394.10.

1.5   Comparing Capital Gains of Multiple Securities Over Time

In this section, we begin with our first application of the price data and we will use many of the techniques we previous discussed to manipulate the data. Prior to beginning this section, we need to make sure we clear the R memory of the data we have created previously as most of those have nothing to do with our current application. Cleaning the R memory helps avoid inadvertently using any old data objects that have been previously created.

   We can take a look at what objects are in the R memory using the `ls` command. The output below shows we have 14 data objects in memory.
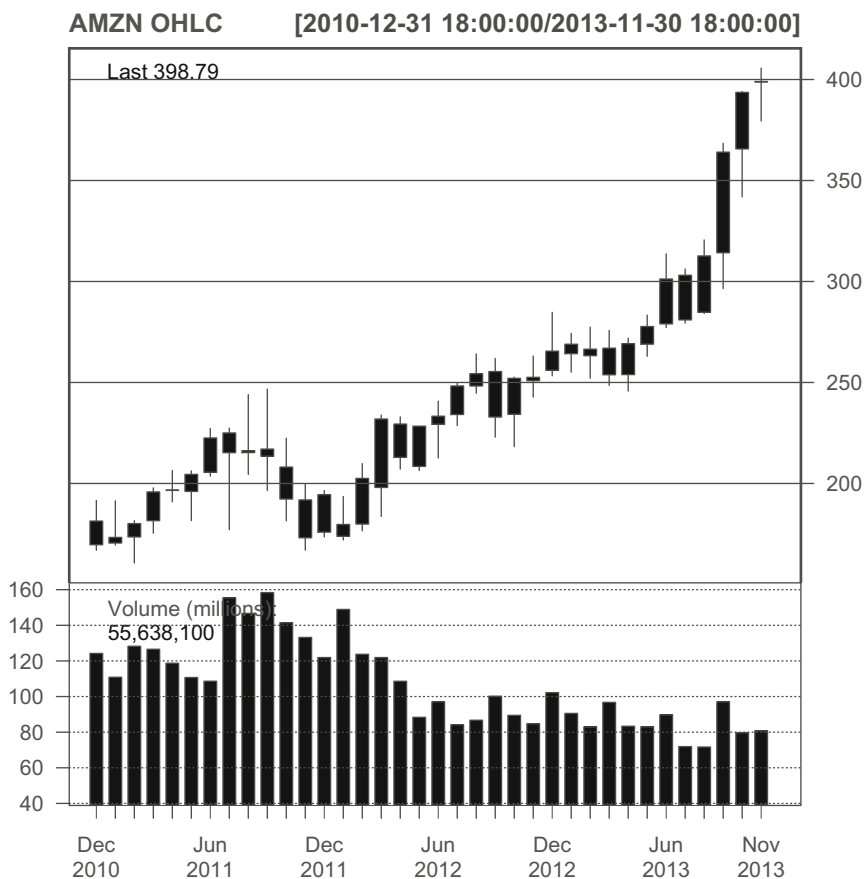
Fig. 1.4 AMZN candlestick chart using monthly prices, 2010–2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

```
> ls()
 [1]"AMZN.2012"       "AMZN.delAdjPrice" "AMZN.delFirst"
 [4]"AMZN.first.week" "AMZN.last30"      "AMZN.onlyFirst"
 [7]"AMZN.onlyPrice"  "AMZN.OpenClose"   "AMZN.OpenCloseVol"
[10]"AMZN.PriceVol"   "data.AMZN"        "data.missing"
[13]"date"            "xts.2012"
```

To delete all these objects, we use the `rm(list=ls())` command. After this command, using the `ls` command again shows that there are is no data in the R memory.

```
> rm(list=ls())
> ls()
character(0)
```

Now that we have cleared the memory, we can now begin with comparing the capital gains of multiple securities over time. A common question when investing is to compare how our investment performed compared to other securities over a certain time horizon. For example, suppose we made an investment at the closing price on December 31, 2010 in Amazon.com (AMZN), IBM (IBM), Yahoo (YHOO), and the S&P 500 Index (ˆGSPC). Note that the text inside the parentheses are Yahoo Finance tickers for these securities. We now pose the question, which of these investments performed better from December 31, 2010 through December 31, 2013 based solely on price appreciation?

To implement this analysis, we would need to track the changes in the price of each security from the same starting value (i.e., starting from their respective closing prices on December 31, 2013). For the starting value, we decide to use $ 1 as the starting point, so it is easier to interpret the values in the chart as a percentage change in the price of the security. This chart is sometimes known as a *normalized price chart*.

Step 1: Import Data for Each of the Four Securities   We go to Yahoo Finance and download the data for each of the four securities from December 31, 2010 to December 31, 2013 into the following CSV files: **AMZN Yahoo.csv**, **IBM Yahoo.csv**, **YHOO Yahoo.csv**, and **GSPC Yahoo.csv**. We then import each of these files into R using the technique we described earlier.

We import the AMZN data again but do not show all the intermediate output. However, we can compare the output below to the output we reported earlier and see that the two outputs are identical.

```
> 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)),]
           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
```

Similarly, we import the Yahoo data with ticker symbol YHOO.

```
> 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)<-
+    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
```

Next, we import the data for IBM with ticker symbol IBM.

```
> 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
```

Finally, we import the S&P 500 Index data with ticker symbol ˆGSPC.

```
> 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)<-
+    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
```

Step 2: Combine Data into One Data Object   For our analysis, we only need the
close price from each of these data objects. As such, we keep only the variables with
the Close suffix. We start by saving AMZN.Close into a new data object labeled

Close.Prices. Then, we use the `cbind` command to string together the close prices for the other three securities.

```
> Close.Prices<-data.AMZN$AMZN.Close
> Close.Prices<-cbind(Close.Prices,data.GSPC$GSPC.Close,
+    data.YHOO$YHOO.Close,data.IBM$IBM.Close)
> Close.Prices[c(1:3,nrow(Close.Prices)),]
           AMZN.Close GSPC.Close YHOO.Close IBM.Close
2010-12-31     180.00    1257.64      16.63    146.76
2011-01-03     184.22    1271.87      16.75    147.48
2011-01-04     185.01    1270.20      16.59    147.64
2013-12-31     398.79    1848.36      40.44    187.57
```

Note that using the `cbind` command here works seamlessly as these securities have prices on each trading day. Therefore, we know that all the dates will be aligned. In general, however, it is good practice to apply the techniques we learned earlier in the chapter to make sure we are combining the data correctly.

Step 3: Convert Data into a `data.frame` In this step, we follow closely the method we implement when we subset the data using dates. We convert the data into a `data.frame` in the same step that we create a workable column of dates. We then rename all the variable names in the `multi.df` to make it simpler to use. We also change the index to denote the observation number.

```
> multi.df<-cbind(index(Close.Prices),
+    data.frame(Close.Prices))
> names(multi.df)<-paste(c("date","AMZN","GSPC","YHOO","IBM"))
> rownames(multi.df)<-seq(1,nrow(multi.df),1)
> multi.df[c(1:3,nrow(multi.df)),]
          date   AMZN    GSPC  YHOO     IBM
1   2010-12-31 180.00 1257.64 16.63  146.76
2   2011-01-03 184.22 1271.87 16.75  147.48
3   2011-01-04 185.01 1270.20 16.59  147.64
755 2013-12-31 398.79 1848.36 40.44  187.57
```

Step 4: Calculate Normalized Values for Each Security When we think of capital appreciation, we normally would think about calculating the change in price every day. We would then cumulate the daily price changes through time and that should be what we need to compare the performance of the four securities. However, when implementing programs, we do not need to think that linearly and we can think of ways to get to the same end result using the simplest technique.

The equivalent way to implement this calculation is to create an index for each security with values that equal the price of the security on each day divided by the security's price on December 31, 2010. To see why this is equivalent, we look at the price on January 3, 2011, the first trading day after December 31, 2010 for AMZN. AMZN price on January 3, 2011 is \$ 184.22, which is 2.34 % higher than the December 31, 2010 price of \$ 180.00. Now, the January 4, 2011 price of \$ 185.01 is 0.04 % higher than the January 3, 2011 price. Cumulatively, that is a 2.78 % increase in price from December 31, 2010 to January 4, 2011. We can get to that same result

by dividing the January 4, 2011 price of $ 185.01 by the December 31, 2010 price of $ 180.00. That is, we can calculate the cumulative percentage price change through some date by dividing the price on that day by the price on December 31, 2010.

```
> 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]
> options(digits=5)
> multi.df[c(1:3,nrow(multi.df)),]
          date   AMZN   GSPC  YHOO     IBM AMZN.idx GSPC.idx YHOO.idx IBM.idx
1   2010-12-31 180.00 1257.6 16.63 146.76   1.0000   1.0000  1.00000  1.0000
2   2011-01-03 184.22 1271.9 16.75 147.48   1.0234   1.0113  1.00722  1.0049
3   2011-01-04 185.01 1270.2 16.59 147.64   1.0278   1.0100  0.99759  1.0060
755 2013-12-31 398.79 1848.4 40.44 187.57   2.2155   1.4697  2.43175  1.2781
> options(digits=7)
```

Reducing the Number of Decimals R Outputs
Note that in the above output we used the `digits=5` option here to reduce the number of decimals that R outputs. This option does not round the values to five decimal places, but displays results with the smallest number being displayed having at least four digits following the first non-zero digit (i.e., 4 digits plus 1 non-zero digit equals the 5 in `digits=5`). In the above example for YHOO.idx, we have the first non-zero digit in the tenths place, which means we will have at least four additional digits following that.

This option remains in effect until we end the R session or until we use the `digits` option again and specify a new number. To return the value to its default value, we type `options(digits=7)`.

Step 5: Plot the Capital Appreciation of Each Security It is often easier to visualize results, so we now plot the values of the variables that have the `idx` suffix. To do this, we use the `plot` command to chart the data. The plot requires an `x` and `y` variable. For the `x` variable, we use the `date`. For the `y` variable, we use the S&P 500 Index. There is nothing special with plotting the S&P 500 Index first except that the first security we use will be at the lowest layer (i.e., later lines will be on top of the S&P 500 Index line). We also choose a line chart (`type="l"`). We change the x-axis labels using the `xlab` argument and the y-axis labels using the `ylab` argument. We can change the color of the line using the `col` argument. The default is a black line so we did not really have to include the `col` option in our example. We then choose the line type `lty`, which can be solid (1) or dashed (2 or 3). We also make the line width (`lwd`) thicker by choosing 2 (default is 1). Then, the `main` argument controls the title of the chart. By using a hard return (i.e., hitting return or enter on the keyboard) at the end of "in" and "Index" we break the title into three rows.

```
> plot(x=multi.df$date,
+     y=multi.df$GSPC.idx,
+     type="l",
+     xlab="Date",
+     ylab="Value of Investment ($)",
+     col="black",
+     lty=1,
+     lwd=2,
+     main="Value of $1 Investment in
+         AMZN, IBM, YHOO, and the S&P 500 Index
+         December 31, 2010 - December 31, 2013")
```

The next steps add lines for each of the other three securities. We do this by using
the `lines` command. The `x` variable all use the `date` variable. The `y` variable is
the index data for AMZN, IBM, and YHOO. We then choose the color, line type,
and line width for each of the different securities. In this book, we will use black and
shades of gray for our charts, but for other purposes we can use other colors to make
the charts more lively.[8]

```
> 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="gray",
+     lty=2,
+     lwd=1)
> lines(x=multi.df$date,
+     y=multi.df$YHOO.idx,
+     col="gray",
+     lty=1,
+     lwd=1)
```

Since the value on December 31, 2010 has been normalized to $ 1, we can add a
horizontal line to the chart that denotes the starting investment value of $ 1. This line
helps us visually separate out periods in which the security generated a capital gain
and periods in which the security generated a capital loss. The horizontal line at $ 1
is created using the `abline` command with `h=1` option.

```
> abline(h=1,lty=1,col="black")
```

---

[8] We can use various common color choices, such as `red`, `blue`, or `green`. However, `R` has many
more color options too choose from. Professor Tian Zhang of Columnbia University has provided a
very helpful document showing the various colors we can use in `R`. This document can be accessed
at the following URL: http://www.stat.columbia.edu/ tzheng/files/Rcolor.pdf.

Finally, we add a legend to the chart using the `legend` command. The block of code below are all related to the `legend` command. The first argument tells R to put the legend at the top-left portion of the chart. The second line identifies the names of the securities. We can order the securities to appear in the legend independent from the order we used to plot the lines. However, the order used in the legend has to be consistent for all arguments used in the legend. Put differently, when we identify the color, line type, and line width, the order has to follow the order of the securities in the second line. Notice that the names, color, line type, and line width all have to use the `c(...)` operator.

```
> legend("topleft",
+    c("AMZN","IBM","YHOO","S&P 500 Index"),
+    col=c("black","gray","gray","black"),
+    lty=c(2,2,1,1),
+    lwd=c(1,1,1,2))
```

Step 6: Fix the y-axis to Encompass Range of Normalized Values for All Securities The output of the previous `plot` command is shown in Fig. 1.5. The output of this chart cuts off the values for some securities. The reason is that the range of the y-axis, by default, follows the range of the first security we plot. In our example, we plotted the S&P 500 Index first and so the y-axis range is based on the S&P 500 Index's normalized values. To fix the y-axis range, we use the `range` command to identify the minimum and maximum values of all four securities, which are in Columns 6 through 9.

```
> y.range<-range(multi.df[,6:9])
> y.range
[1] 0.6668671 2.4564041
```

The `y.range` goes into an argument inside the `plot` command called `ylim`. We add this argument as the fifth line in the `plot` command.

```
> plot(x=multi.df$date,
+     y=multi.df$GSPC.idx,
+     type="l",
+     xlab="Date",
+     ylim=y.range,
+     ylab="Value of Investment ($)",
+     col="black",
+     lty=1,
+     lwd=2,
+     main="Value of $1 Investment in
+        AMZN, IBM, YHOO, and the S&P 500 Index
+        December 31, 2010 - December 31, 2013")
> lines(x=multi.df$date,
+     y=multi.df$AMZN.idx,
+     col="black",
+     lty=2,
+     lwd=1)
> lines(x=multi.df$date,
+     y=multi.df$IBM.idx,
+     col="gray",
+     lty=2,
+     lwd=1)
> lines(x=multi.df$date,
+     y=multi.df$YHOO.idx,
+     col="gray",
+     lty=1,
+     lwd=1)
> abline(h=1,lty=1,col="black")
> legend("topleft",c("AMZN","IBM","YHOO","S&P 500 Index"),
+     col=c("black","gray","gray","black"),
+     lty=c(2,2,1,1),
+     lwd=c(1,1,1,2))
```

Making this change to the our code for plotting above generates a better looking, non-truncated chart, as shown in Fig. 1.6.

Because the original chart used the y-axis for GSPC, we did not see how well AMZN and YHOO performed over the 2011 to 2013 time period. The beauty of these normalized price charts is that they are relatively easy to interpret. For example, YHOO's indexed value of around 2.5 means that over 2011–2013 YHOO returned approximately 150 %, which we can calculate by subtracting 1.0 (the starting index value) from 2.5 (the approximate ending index value). In fact, that growth is more impressive as YHOO's indexed value dipped around the middle of 2011. The chart also shows that AMZN's return is not too far behind at around 125 %. IBM, on the other hand, although it had positive returns over the period, did not outperform the market. As such, investing in the overall market would have given an investor higher realized returns than IBM over the 2011–2013 period.
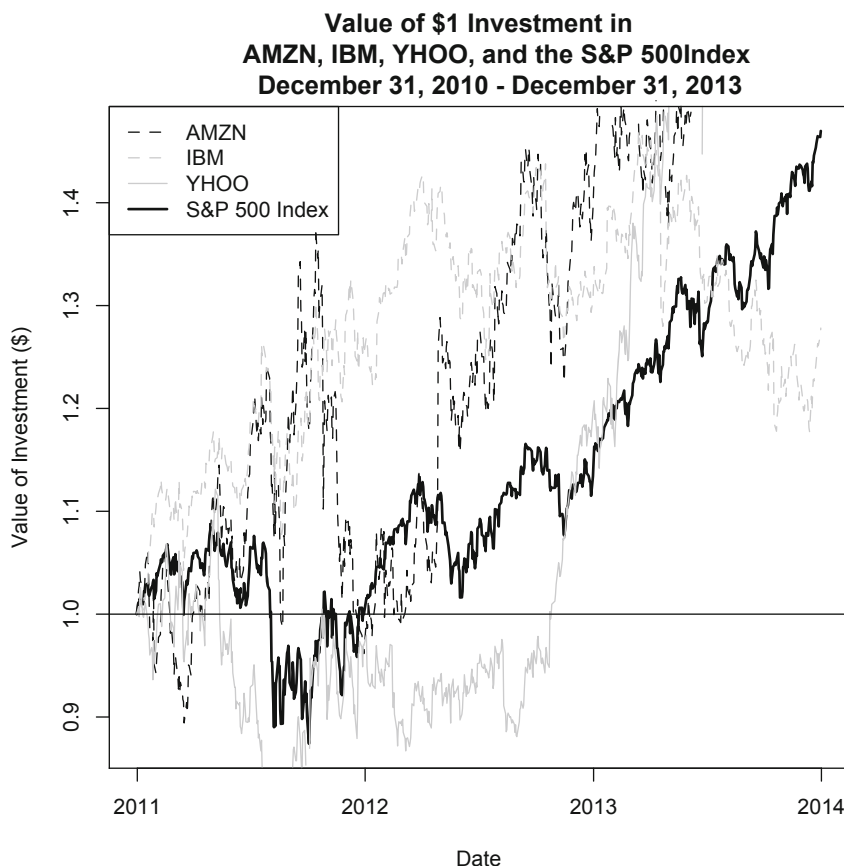
Fig. 1.5 Capital appreciation of AMZN, YHOO, IBM, and the S&P 500 Index, December 31, 2010 to December 31, 2013. Note that this chart truncates higher values of some securities because the range of values for the y-axis is based on the values of the S&P 500 Index, which underperformed some of the other securities on this chart. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

1.5.1   Alternative Presentation of Normalized Price Chart

Our chart in Fig. 1.6 is not the only way we can present the data. An alternative way could be to separate each of the four securities into four mini-charts. In each chart, we can highlight one security by having the line for that security in a different color, while the other three securities all have the same color. We can then plot all four of the mini-charts into one big chart, so we do not lose any information. The resulting chart will look like Fig. 1.7.

To implement this chart, we essentially create four mini-charts. However, we need to let R know that we will be doing this. We will be using mostly the data from the prior section.

Value of $1 Investment in
AMZN, IBM, YHOO, and the S&P 500 Index
December 31, 2010 - December 31, 2013
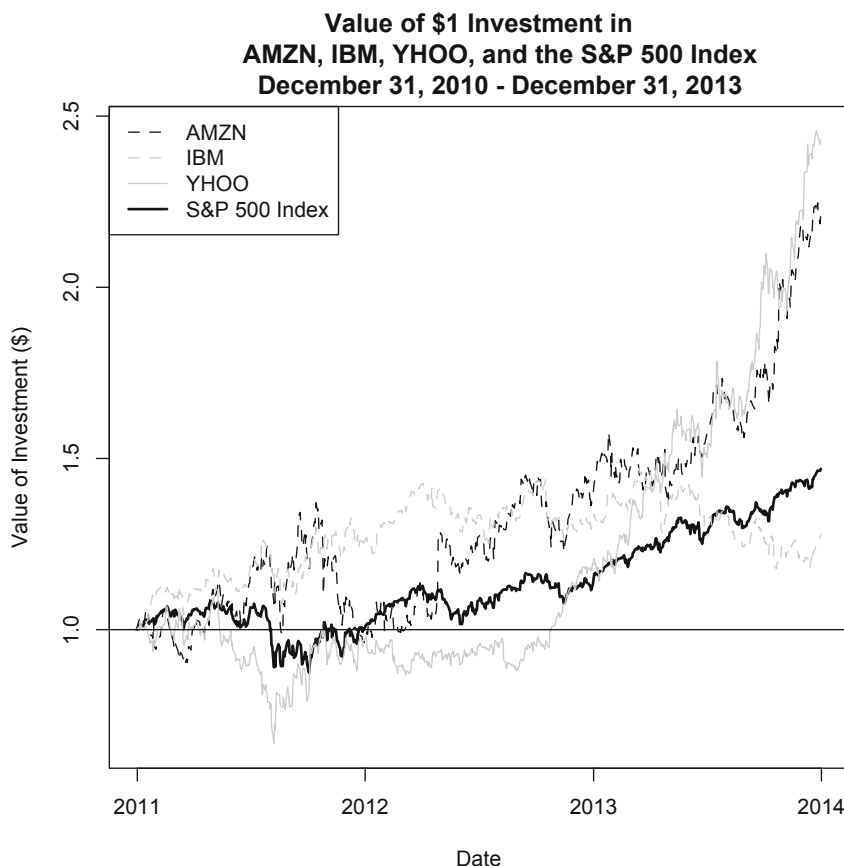
Fig. 1.6 Capital appreciation of AMZN, YHOO, IBM, and the S&P 500 Index after fixing the y-axis range, December 31, 2010 to December 31, 2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

Step 1: Setup Chart Layout in R We want to make sure we have enough space on the outside margin to add a global title to the chart. For this, we use the `par` command using the `oma` option. The third argument in `oma` represents the top margin. By trial-and-error, we enter "3" in that spot, while keeping the others at zero.

```
> par(oma=c(0,0,3,0))
```

Step 2: Let R **Know We Will Be Plotting 4 Charts with 2 Charts in Each Column and 2 Charts in Each Row** Using the `par` command, we use the `mfrow` option to tell R that we want a 2 (rows) × 2 (columns) layout for the next four plots we create.

```
> par(mfrow=c(2,2))
```

Value of $1 Invested in Amazon, IBM, Yahoo, and the Market
December 31, 2010 - December 31, 2013
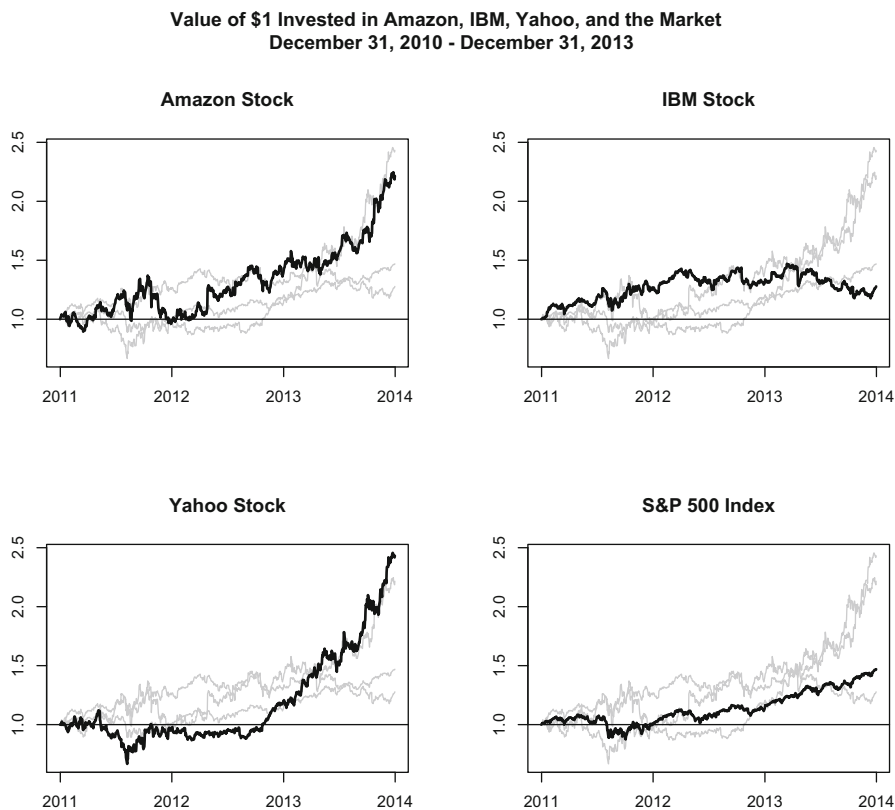
Fig. 1.7 Alternative presentation of performance of AMZN, YHOO, IBM, and the S&P 500 Index, December 31, 2010 to December 31, 2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

Step 3: Create the 4 Plots In this step, we simply create four plots. For each plot, we plot the subject stock last. For example, the first plot is about Amazon.com stock. Therefore, the last `lines` command has AMZN's index value, which is going to be drawn as a black line that is relatively thicker than the other lines. The other three lines are all colored gray. Similarly, the second plot focuses on IBM, the third plot focuses on Yahoo, and the fourth plot focuses on the S&P 500 Index.

```
> plot(x=multi.df$date,
+     xlab="",
+     y=multi.df$YHOO.idx,
```

```
+     ylim=y.range,
+     ylab="",
+     type="l",
+     col="gray",
+     main="Amazon Stock")
>     lines(x=multi.df$date,y=multi.df$GSPC.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$IBM.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$AMZN.idx,col="black",lwd=2)
>     abline(h=1)
> plot(x=multi.df$date,
+     xlab="",
+     y=multi.df$YHOO.idx,
+     ylim=y.range,
+     ylab="",
+     type="l",
+     col="gray",
+     main="IBM Stock")
>     lines(x=multi.df$date,y=multi.df$AMZN.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$GSPC.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$IBM.idx,col="black",lwd=2)
>     abline(h=1)
> plot(x=multi.df$date,
+     xlab="",
+     y=multi.df$GSPC.idx,
+     ylim=y.range,
+     ylab="",
+     type="l",
+     col="gray",
+     main="Yahoo Stock")
>     lines(x=multi.df$date,y=multi.df$AMZN.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$IBM.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$YHOO.idx,col="black",lwd=2)
>     abline(h=1)
> plot(x=multi.df$date,
+     xlab="",
+     y=multi.df$YHOO.idx,
+     ylim=y.range,
+     ylab="",
+     type="l",
+     col="gray",
+     main="S&P 500 Index")
>     lines(x=multi.df$date,y=multi.df$AMZN.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$IBM.idx,col="gray")
>     lines(x=multi.df$date,y=multi.df$GSPC.idx,col="black",lwd=2)
>     abline(h=1)
```

Step 4: Create a Global Title for the Charts Each mini-chart above has its own title, but it would be more informative if we added a global title to the chart. We do this by using the `title` command and the `outer=T` option. The latter makes sure that we apply the title to the outer margins, which we have setup above in Step 1.

```
> title1="Value of $1 Invested in Amazon, IBM, Yahoo, and the Market"
> title2="December 31, 2010 - December 31, 2013"
> title(main=paste(title1,"\n",title2),outer=T)
```

Note that this is an alternative way to create a title. The `title1` and `title2` simply contain text, which are then applied to the `main` argument using the `paste` command. The `\n` creates the effect of a hard return, which moves the text in `title2` to the second row of the title.

1.6   Technical Analysis Examples

Technical analysis is the use of charts to study stock price and volume data for the purpose of forecasting future trends. Those who follow technical analysis are using stock price and volume data as an indication of the supply and demand for the stock. For example, a rising stock price may indicate that demand exceeds supply while a falling stock price may indicate that supply exceeds demand. As a trading strategy, profiting from technical analysis relies on being able to identify trends and the ability to catch on during the early stages of the trend. Moreover, the same technical indicators may be interpreted differently by different chartists depending on their investment philosophies (e.g., trend follower or contrarian).

There are three broad groups of technical indicators: (i) trend indicator, (ii) volatility indicator, and (iii) momentum indicator. In this chapter, we go through an example each type starting with simple moving average crossover (trend), Bollinger Bands (volatility), and relative strength index (momentum). There are many more technical indicators. Note that the above examples as well as many more technical indicators can be implemented using the `chartSeries` function in `R`, which we used earlier to create the Open-High-Low-Close (OHLC) chart. In constructing trading strategies, combining indicators is often used. For example, we can use the relative strength index to confirm signals identified by the simple moving average. Therefore, we can view the examples as building blocks to more complex strategies.

1.6.1   Trend: Simple Moving Average Crossover

A common technical analysis trend indicator is the Simple Moving Average (SMA) crossover. In an SMA, the moving average is calculated by taking the average of a firm's stock price over a certain number of days. The term "simple" comes from the fact that this type of average treats all days equally, regardless of how near or far those days are from the present. This is called an SMA "crossover" because we will use two SMA lines, a shorter-term and a longer-term, and make trading decisions when the lines cross.

In our example, we will use an average over 50 (200) days for the shorter (longer) term. Since we are interested in making decisions during the present, an extremely long time series of data may not be necessary. As such, we will demonstrate how to implement an SMA crossover for Amazon.com stock from 2012 to 2013.

Step 1: Obtain Closing Prices for Amazon.com Stock Since we have already imported the Amazon.com stock price data into `data.AMZN`, we only need to extract the closing price from that data object (i.e., the fourth column).

```
> AMZN.sma<-data.AMZN[,4]
> AMZN.sma[c(1:3,nrow(AMZN.sma)),]
           AMZN.Close
2010-12-31     180.00
2011-01-03     184.22
2011-01-04     185.01
2013-12-31     398.79
```

Step 2: Calculate the Rolling 50-Day and 200-Day Average Price To calculate the rolling or moving average, we use the `rollmeanr` function and choose the window length as `k=50` for the 50-day moving average and `k=200` for the 200-day moving average. Note that the first three observations below under `sma50` and `sma200` are `NA` as R only reports data beginning the 50th and 200th observation, respectively.

```
> AMZN.sma$sma50<-rollmeanr(AMZN.sma$AMZN.Close,k=50)
> AMZN.sma$sma200<-rollmeanr(AMZN.sma$AMZN.Close,k=200)
> AMZN.sma[c(1:3,nrow(AMZN.sma)),]
           AMZN.Close     sma50    sma200
2010-12-31     180.00        NA        NA
2011-01-03     184.22        NA        NA
2011-01-04     185.01        NA        NA
2013-12-31     398.79  372.9076  306.0776
```

This is because the first 50 observations are required to calculate the first 50-day moving average (i.e., the first 49 observations for `sma50` will be `NA`).

```
> AMZN.sma[48:52,]
           AMZN.Close     sma50  sma200
2011-03-10     166.14        NA      NA
2011-03-11     168.07        NA      NA
2011-03-14     166.73  179.2034      NA
2011-03-15     165.08  178.9050      NA
2011-03-16     164.70  178.5146      NA
```

Similarly, the first 200 observations are required to calculate the first 200-day moving average (i.e., the first 199 observations for `sma200` will be `NA`).

Step 3: Subset to Only Show 2012 and 2013 Data Note that the first observation in the data object is currently December 31, 2010. We want to use the `subset` command to start the data on January 1, 2012.

```
> AMZN.sma[198:202,]
          AMZN.Close     sma50    sma200
2011-10-12    236.81 212.7294        NA
2011-10-13    236.15 213.2532        NA
2011-10-14    246.71 214.1578 195.3837
2011-10-17    242.33 214.9504 195.6953
2011-10-18    243.88 215.9540 195.9936

> AMZN.sma2012<-subset(AMZN.sma,
+    index(AMZN.sma)>="2012-01-01")
> AMZN.sma2012[c(1:3,nrow(AMZN.sma2012)),]
          AMZN.Close     sma50    sma200
2012-01-03    179.03 197.8458 201.5382
2012-01-04    177.51 196.7004 201.6031
2012-01-05    177.61 195.5004 201.6782
2013-12-31    398.79 372.9076 306.0776
```

Step 4: Plot the SMA To make sure we have the full range in the y-axis, we use the
`range` command to find the minimum and maximum values in the `AMZN.sma2012`
data. Note that we used the `na.rm=TRUE` option so that the `range` command
ignores any `NA` values.

```
> y.range<-range(AMZN.sma2012,na.rm=TRUE)
> y.range
[1] 175.93 404.39
> par(mfrow=c(1,1))
> plot(x=index(AMZN.sma2012),
+    xlab="Date",
+    y=AMZN.sma2012$AMZN.Close,
+    ylim=y.range,
+    ylab="Price ($)",
+    type="l",
+    main="Amazon - Simple Moving Average
+       January 1, 2012 - December 31, 2013")
> lines(x=index(AMZN.sma2012),y=AMZN.sma2012$sma50)
> lines(x=index(AMZN.sma2012),y=AMZN.sma2012$sma200,lty=2)
> legend("topleft",
+    c("Amazon Price","50-Day Moving Average","200-Day Moving Average"),
+    lty=c(1,1,2))
```

The output of the chart is shown as Fig. 1.8. If the 50-day moving average cross
above the 200-day moving average, which is called a *bullish crossover*, this may
be taken as an indicator to buy the stock. Conversely, if the 50-day moving average
crosses below the 200-day moving average, which is known as a *bearish crossover*,
this may be taken as an indication to sell the stock.

Amazon - Simple Moving Average
January 1, 2012 - December 31, 2013
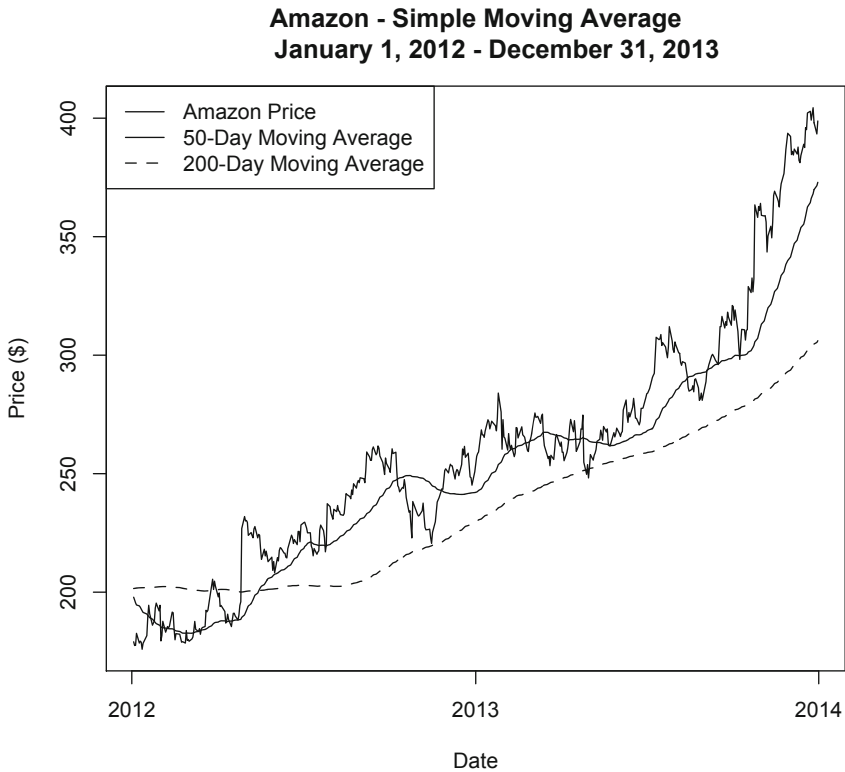
Fig. 1.8 50-day and 200-day simple moving average of Amazon stock, 2012–2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

1.6.2 Volatility: Bollinger Bands

A frequently used technical analysis volatility indicator are *Bollinger Bands*. The Bollinger Bands have three components. The first component is a 20-day simple moving average (SMA). The second component is an upper band, which is two standard deviations above the 20-day SMA. The third component is a lower band, which is two standard deviations below the 20-day SMA. Bollinger Bands are considered as volatility indicators because the Bollinger Bands widen (narrow) with more (less) volatility in the stock. When the bands narrow, it may be used as an indication that volatility is about to rise. Below we demonstrate how to apply Bollinger Bands to Amazon.com stock in 2013.

Step 1: Obtain Closing Prices for Amazon.com Stock Since we have already imported the Amazon.com stock price data into `data.AMZN`, we only need to extract the closing price from that data object (i.e., the fourth column).

```
> AMZN.bb<-data.AMZN[,4]
> AMZN.bb[c(1:3,nrow(AMZN.bb)),]
          AMZN.Close
2010-12-31     180.00
2011-01-03     184.22
2011-01-04     185.01
2013-12-31     398.79
```

Step 2: Calculate Rolling 20-Day Mean and Standard Deviation Similar to our calculation in the SMA, we use the `rollmeanr` command with `k=20` to calculate the 20-day moving average. For the standard deviation, we use the `rollapply` command, which allows us to apply a function on a rolling basis. Here, the `FUN=sd` tells R that the function we want to apply is the standard deviation and the `width=20` tells R that it is a 20-day standard deviation that we want to calculate.

```
> AMZN.bb$avg<-rollmeanr(AMZN.bb$AMZN.Close,k=20)
> AMZN.bb$sd<-rollapply(AMZN.bb$AMZN.Close,width=20,FUN=sd,fill=NA)
> AMZN.bb[c(1:3,nrow(AMZN.bb)),]
          AMZN.Close       avg       sd
2010-12-31     180.00        NA       NA
2011-01-03     184.22        NA       NA
2011-01-04     185.01        NA       NA
2013-12-31     398.79 391.4565 7.519363
```

Since we are using a 20-day moving average and rolling standard deviation, we will have `NA` under `avg` and `sd`. The 20th observation is January 28, 2011.

```
> AMZN.bb[18:22,]
          AMZN.Close       avg       sd
2011-01-26     175.39        NA       NA
2011-01-27     184.45        NA       NA
2011-01-28     171.14 182.8705 5.061730
2011-01-31     169.64 182.3525 5.841057
2011-02-01     172.11 181.7470 6.250598
```

Step 3: Subset to Only Show 2013 Data We now subset the data to what we want to show, which is only the data for 2013.

```
> AMZN.bb2013<-subset(AMZN.bb,
+    index(AMZN.bb)>="2013-01-01")
> AMZN.bb2013[c(1:3,nrow(AMZN.bb2013)),]
          AMZN.Close       avg       sd
2013-01-02     257.31 253.1670 4.457978
2013-01-03     258.48 253.4665 4.608764
2013-01-04     259.15 253.7260 4.780912
2013-12-31     398.79 391.4565 7.519363
```

Step 4: Calculate the Bollinger Bands We now calculate the bands that are two standard deviations around the average. Using January 2, 2013 as an example, this means that the upper Bollinger Band is equal to 262.0830 (= 253.1670 +

(2×4.457978)) and the lower Bollinger Band is equal to 244.2510 (= $253.1670 - (2 \times 4.457978)$). Note that the difference is that the $2 \times$ Standard Deviation is added to or subtracted from the mean.

```
> AMZN.bb2013$sd2up<-AMZN.bb2013$avg+2*AMZN.bb2013$sd
> AMZN.bb2013$sd2down<-AMZN.bb2013$avg-2*AMZN.bb2013$sd
> AMZN.bb2013[c(1:3,nrow(AMZN.bb2013)),]
           AMZN.Close      avg       sd   sd2up  sd2down
2013-01-02     257.31 253.1670 4.457978 262.0830 244.2510
2013-01-03     258.48 253.4665 4.608764 262.6840 244.2490
2013-01-04     259.15 253.7260 4.780912 263.2878 244.1642
2013-12-31     398.79 391.4565 7.519363 406.4952 376.4178
```

Step 5: Plot the Bollinger Bands As typical when we have multiple lines to plot, we would need to determine the range of possible values for the y-axis. Note that we do not want to include in the calculation of the range the values in the sd column. Therefore, we exclude Column 3 when we use the range function.

```
> y.range<-range(AMZN.bb2013[,-3],na.rm=TRUE)
> y.range
[1] 242.3923 406.4952
> plot(x=index(AMZN.bb2013),
+     xlab="Date",
+     y=AMZN.bb2013$AMZN.Close,
+     ylim=y.range,
+     ylab="Price ($)",
+     type="l",
+     lwd=3,
+     main="Amazon - Bollinger Bands (20 days, 2 deviations)
+        January 1, 2013 - December 31, 2013")
> lines(x=index(AMZN.bb2013),y=AMZN.bb2013$avg,lty=2)
> lines(x=index(AMZN.bb2013),y=AMZN.bb2013$sd2up,col="gray40")
> lines(x=index(AMZN.bb2013),y=AMZN.bb2013$sd2down,col="gray40")
> legend("topleft",
+     c("Amazon Price","20-Day Moving Average","Upper Band","Lower Band"),
+     lty=c(1,2,1,1),
+     lwd=c(3,1,1,1),
+     col=c("black","black","gray40","gray40"))
```

The resulting plot is shown in Fig. 1.9. Assuming a normal distribution, two standard deviations in either direction from the mean should cover pretty much the majority of the data. As the chart shows, for 2013, most of the closing prices fell within the Bollinger Bands. For a trend follower, when Amazon's stock price was right around the upper band as in July 2013 and November 2013, this may be taken as an indication that the stock is *overbought*. Conversely, when Amazon.com's stock price moved right around the lower band, as in August 2013, this may be taken as an indication that the stock is *oversold*.

Amazon - Bollinger Bands (20 days, 2 deviations)
January 1, 2013 - December 31, 2013
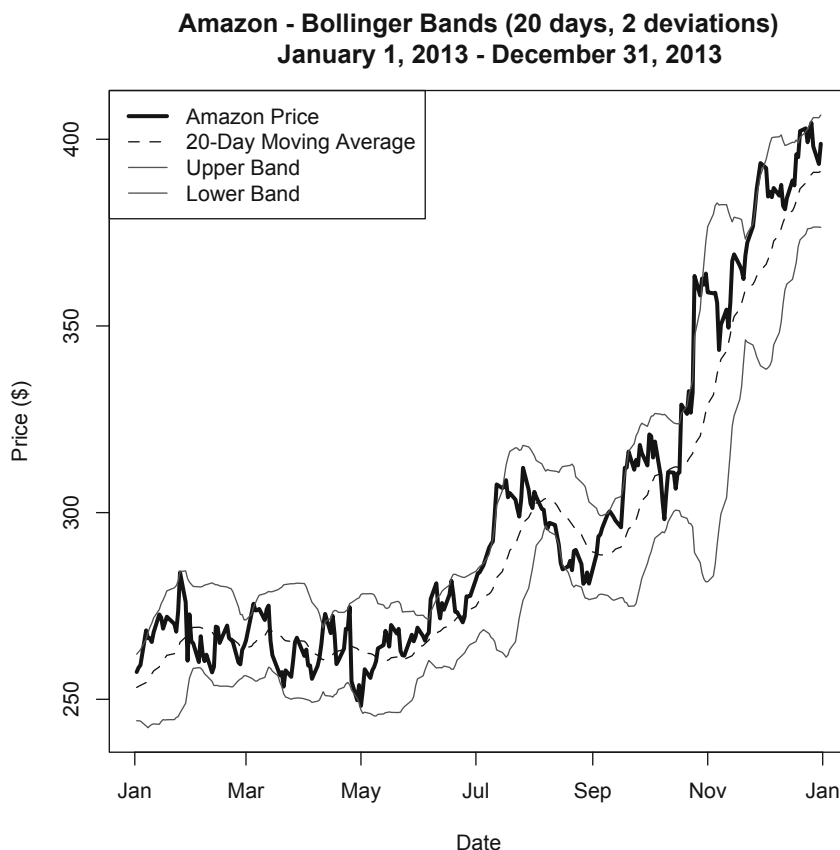
Fig. 1.9 AMZN stock price and Bollinger Bands, 2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

1.6.3   Momentum: Relative Strength Index

A common technical analysis momentum indicator is the Relative Strength Index (RSI). The typical calculation is to use a 14-day period. The RSI is calculated as

$$RSI = 100 - \frac{100}{1 + RS}, \tag{1.1}$$

where RS is equal to the up average divided by the down average with the averages calculated using the Wilder Exponential Moving Average described below.

The RSI is used in conjunction with an overbought line and an oversold line. The overbought line is typically set at a level of 70 and the oversold line is typically set at a level of 30. A buy signal is created when the RSI rises from below the oversold line and crosses the oversold line. Conversely, a sell signal is created when the RSI falls from above the overbought line and crosses the overbought line. Below we demonstrate how to calculate the RSI for Amazon.com for the 2012 to 2013 period.

Step 1: Obtain Closing Prices for Amazon.com Stock Since we have already imported the Amazon stock price data into `data.AMZN`, we only need to extract the closing price from that data object (i.e., the fourth column).

```
> AMZN.RSI<-data.AMZN[,4]
> AMZN.RSI$delta<-diff(AMZN.RSI$AMZN.Close)
> AMZN.RSI[c(1:3,nrow(AMZN.RSI)),]
           AMZN.Close delta
2010-12-31     180.00    NA
2011-01-03     184.22  4.22
2011-01-04     185.01  0.79
2013-12-31     398.79  5.42
```

In this step, we also calculate the difference in Amazon.com's price using the `diff` command. The difference between the closing price today and yesterday's closing price is reported in the column labeled `delta`.

Step 2: Create Dummy Variables to Indicate Whether Price Went Up or Price Went Down We construct the `up` variable, which takes on a value of 1 if the price of Amazon.com went up and zero otherwise. We also construct the `down` variable, which takes on a value of 1 if the price of Amazon.com went down and zero otherwise. To create these dummy variables we use the `ifelse` command.

```
> AMZN.RSI$up<-ifelse(AMZN.RSI$delta>0,1,0)
> AMZN.RSI$down<-ifelse(AMZN.RSI$delta<0,1,0)
> AMZN.RSI[c(1:3,nrow(AMZN.RSI)),]
           AMZN.Close delta up down
2010-12-31     180.00    NA NA   NA
2011-01-03     184.22  4.22  1    0
2011-01-04     185.01  0.79  1    0
2013-12-31     398.79  5.42  1    0
```

Step 3: Calculate Prices for Up Days and Prices for Down Days To construct a series of prices on up days, we multiply `AMZN.Close` with `up`. If it is an up day, `up` will equal one, so `up.val` will equal the Amazon.com closing price. On a down day, `up` will equal zero, so `up.val` will equal zero. The down day prices are calculated in a similar way.

```
> AMZN.RSI$up.val<-AMZN.RSI$delta*AMZN.RSI$up
> AMZN.RSI$down.val<--AMZN.RSI$delta*AMZN.RSI$down
> AMZN.RSI<-AMZN.RSI[-1,]
> AMZN.RSI[c(1:3,nrow(AMZN.RSI)),]
           AMZN.Close delta up down up.val down.val
2011-01-03     184.22  4.22  1    0   4.22        0
2011-01-04     185.01  0.79  1    0   0.79        0
2011-01-05     187.42  2.41  1    0   2.41        0
2013-12-31     398.79  5.42  1    0   5.42        0
```

Note that we deleted the December 31, 2010 observation in this step. We do not need the December 31, 2010 values for any subsequent calculation.

Step 4: Calculate Initial Up and Down 14-Day Averages Here, we use the command on the average function `FUN=mean`. This is similar to using the `rollmeanr` command above, but I just thought of switching it up for a little variety.

```
> AMZN.RSI$up.first.avg<-rollapply(AMZN.RSI$up.val,
+    width=14,FUN=mean,fill=NA,na.rm=TRUE)
> AMZN.RSI$down.first.avg<-rollapply(AMZN.RSI$down.val,
+    width=14,FUN=mean,fill=NA,na.rm=TRUE)
> AMZN.RSI[c(1:15,nrow(AMZN.RSI)),]
          AMZN.Close delta up down up.val down.val up.first.avg
2011-01-03    184.22  4.22  1    0   4.22     0.00           NA
2011-01-04    185.01  0.79  1    0   0.79     0.00           NA
2011-01-05    187.42  2.41  1    0   2.41     0.00           NA
2011-01-06    185.86 -1.56  0    1   0.00     1.56           NA
2011-01-07    185.49 -0.37  0    1   0.00     0.37           NA
2011-01-10    184.68 -0.81  0    1   0.00     0.81           NA
2011-01-11    184.34 -0.34  0    1   0.00     0.34           NA
2011-01-12    184.08 -0.26  0    1   0.00     0.26           NA
2011-01-13    185.53  1.45  1    0   1.45     0.00           NA
2011-01-14    188.75  3.22  1    0   3.22     0.00           NA
2011-01-18    191.25  2.50  1    0   2.50     0.00           NA
2011-01-19    186.87 -4.38  0    1   0.00     4.38           NA
2011-01-20    181.96 -4.91  0    1   0.00     4.91           NA
2011-01-21    177.42 -4.54  0    1   0.00     4.54    1.0421429
2011-01-24    176.85 -0.57  0    1   0.00     0.57    0.7407143
2013-12-31    398.79  5.42  1    0   5.42     0.00    2.4550000
          down.first.avg
2011-01-03             NA
2011-01-04             NA
2011-01-05             NA
2011-01-06             NA
2011-01-07             NA
2011-01-10             NA
2011-01-11             NA
2011-01-12             NA
2011-01-13             NA
2011-01-14             NA
2011-01-18             NA
2011-01-19             NA
2011-01-20             NA
2011-01-21       1.226429
2011-01-24       1.267143
2013-12-31       1.668571
```

Step 5: Calculate the Wilder Exponential Moving Average to Calculate Final Up and Down 14-Day Averages To make the calculation easier, we extract the `up.val` and `down.val` columns in `AMZN.RSI` into separate independent vectors. Then, we need to calculate the Wilder Exponential Moving Average for the up and down values. This average calculation assumes that the initial average the day before would have a weight of 13 out of 14 days and the current average will have a weight of one out of 14 days. We apply this same logic for the up average and down average. The calculated values under the Wilder Exponential Moving Average are reported under the `up.avg` and `down.avg` columns, respectively.

```
> up.val<-as.numeric(AMZN.RSI$up.val)
> down.val<-as.numeric(AMZN.RSI$down.val)
>
> AMZN.RSI$up.avg<-AMZN.RSI$up.first.avg
> for (i in 15:nrow(AMZN.RSI)){
+       AMZN.RSI$up.avg[i] <-
+         ((AMZN.RSI$up.avg[i-1]*13+up.val[i])/14)
+ }
> AMZN.RSI$down.avg<-AMZN.RSI$down.first.avg
> for (i in 15:nrow(AMZN.RSI)){
+       AMZN.RSI$down.avg[i]<-
+         ((AMZN.RSI$down.avg[i-1]*13+down.val[i])/14)
+ }
> AMZN.RSI[c(1:20,nrow(AMZN.RSI)),]
         AMZN.Close  delta up down up.val down.val up.first.avg
2011-01-03   184.22   4.22  1    0   4.22     0.00          NA
2011-01-04   185.01   0.79  1    0   0.79     0.00          NA
2011-01-05   187.42   2.41  1    0   2.41     0.00          NA
2011-01-06   185.86  -1.56  0    1   0.00     1.56          NA
2011-01-07   185.49  -0.37  0    1   0.00     0.37          NA
2011-01-10   184.68  -0.81  0    1   0.00     0.81          NA
2011-01-11   184.34  -0.34  0    1   0.00     0.34          NA
2011-01-12   184.08  -0.26  0    1   0.00     0.26          NA
2011-01-13   185.53   1.45  1    0   1.45     0.00          NA
2011-01-14   188.75   3.22  1    0   3.22     0.00          NA
2011-01-18   191.25   2.50  1    0   2.50     0.00          NA
2011-01-19   186.87  -4.38  0    1   0.00     4.38          NA
2011-01-20   181.96  -4.91  0    1   0.00     4.91          NA
2011-01-21   177.42  -4.54  0    1   0.00     4.54   1.0421429
2011-01-24   176.85  -0.57  0    1   0.00     0.57   0.7407143
2011-01-25   176.70  -0.15  0    1   0.00     0.15   0.6842857
2011-01-26   175.39  -1.31  0    1   0.00     1.31   0.5121429
2011-01-27   184.45   9.06  1    0   9.06     0.00   1.1592857
2011-01-28   171.14 -13.31  0    1   0.00    13.31   1.1592857
2011-01-31   169.64  -1.50  0    1   0.00     1.50   1.1592857
2013-12-31   398.79   5.42  1    0   5.42     0.00   2.4550000
         down.first.avg   up.avg down.avg
2011-01-03             NA     NA      NA
2011-01-04             NA     NA      NA
2011-01-05             NA     NA      NA
2011-01-06             NA     NA      NA
2011-01-07             NA     NA      NA
2011-01-10             NA     NA      NA
2011-01-11             NA     NA      NA
2011-01-12             NA     NA      NA
2011-01-13             NA     NA      NA
2011-01-14             NA     NA      NA
2011-01-18             NA     NA      NA
2011-01-19             NA     NA      NA
2011-01-20             NA     NA      NA
2011-01-21       1.226429 1.0421429 1.226429
2011-01-24       1.267143 0.9677041 1.179541
2011-01-25       1.277857 0.8985824 1.106002
2011-01-26       1.371429 0.8343979 1.120573
2011-01-27       1.260000 1.4219409 1.040532
2011-01-28       2.184286 1.3203737 1.916923
2011-01-31       2.233571 1.2260613 1.887143
2013-12-31       1.668571 2.5775619 1.654829
```

Step 6: Calculate the RSI We can now calculate the RSI using the formula above.

```
> AMZN.RSI$RS<-AMZN.RSI$up.avg/AMZN.RSI$down.avg
> AMZN.RSI$RSI<-100-(100/(1+AMZN.RSI$RS))
> AMZN.RSI[c(14:20,nrow(AMZN.RSI)),]
           AMZN.Close   delta up down up.val down.val up.first.avg
2011-01-21     177.42   -4.54  0    1   0.00     4.54    1.0421429
2011-01-24     176.85   -0.57  0    1   0.00     0.57    0.7407143
2011-01-25     176.70   -0.15  0    1   0.00     0.15    0.6842857
2011-01-26     175.39   -1.31  0    1   0.00     1.31    0.5121429
2011-01-27     184.45    9.06  1    0   9.06     0.00    1.1592857
2011-01-28     171.14  -13.31  0    1   0.00    13.31    1.1592857
2011-01-31     169.64   -1.50  0    1   0.00     1.50    1.1592857
2013-12-31     398.79    5.42  1    0   5.42     0.00    2.4550000
           down.first.avg    up.avg down.avg        RS      RSI
2011-01-21       1.226429 1.0421429 1.226429 0.8497379 45.93829
2011-01-24       1.267143 0.9677041 1.179541 0.8204075 45.06724
2011-01-25       1.277857 0.8985824 1.106002 0.8124598 44.82636
2011-01-26       1.371429 0.8343979 1.120573 0.7446169 42.68082
2011-01-27       1.260000 1.4219409 1.040532 1.3665512 57.74442
2011-01-28       2.184286 1.3203737 1.916923 0.6887985 40.78630
2011-01-31       2.233571 1.2260613 1.887143 0.6496918 39.38262
2013-12-31       1.668571 2.5775619 1.654829 1.5576001 60.90085
```

Step 7: Subset to Show Only 2012 and 2013 Data We now subset the data to only
show the data for 2012 and 2013. We also only keep the RSI variable (last column in
the data object), which we do by using the ncol command that returns the number
of columns in the AMZN.RSI data object.

```
> AMZN.RSI2012<-subset(AMZN.RSI[,ncol(AMZN.RSI)],
+     index(AMZN.RSI)>="2012-01-01")
> AMZN.RSI2012[c(1:3,nrow(AMZN.RSI2012)),]
                RSI
2012-01-03 42.61800
2012-01-04 41.17848
2012-01-05 41.31892
2013-12-31 60.90085
```

Step 8: Plot the RSI We now use the plot command to chart the RSI. We know
the range of the y-axis has to be between 0 and 100. As such, we can input these
values directly into the ylim argument of the plot command.

```
> title1<-"Amazon - Relative Strength Index"
> title2<-"January 2012 - December 2013"
> plot(x=index(AMZN.RSI2012),
+     xlab="Date",
+     y=AMZN.RSI2012$RSI,
+     ylab="RSI (14-Day Moving Average)",
+     ylim=c(0,100),
+     type="l",
+     main=paste(title1,"\n",title2))
> abline(h=c(30,70),lty=2)
```

Amazon - Relative Strength Index
January 2012 - December 2013
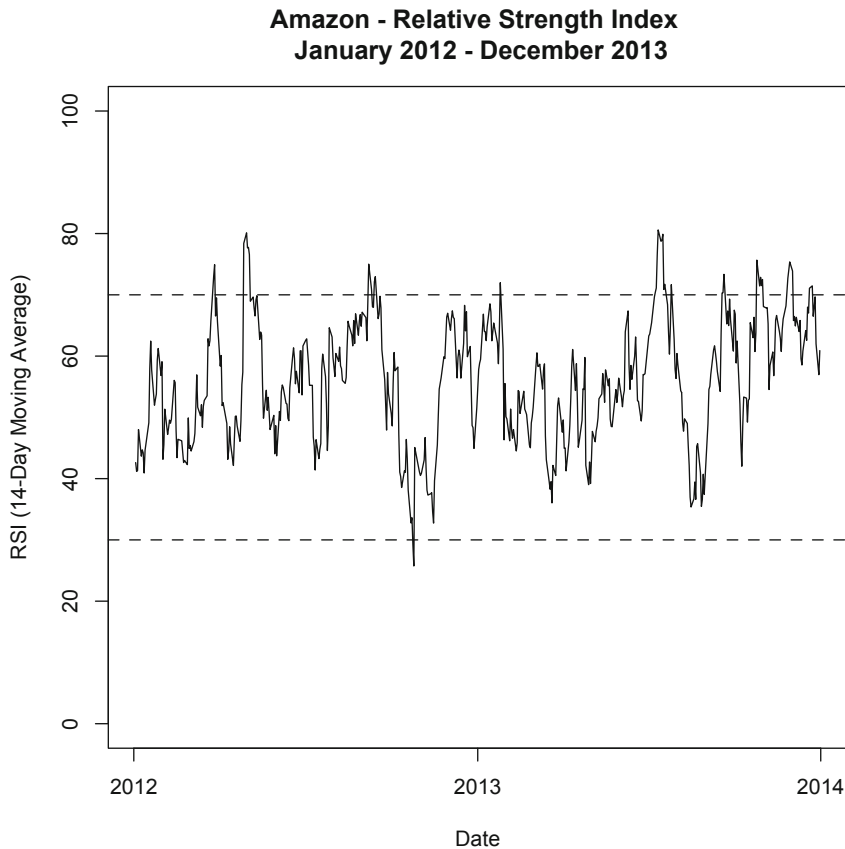
Fig. 1.10 AMZN stock price and relative strength index, 2012–2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

Figure 1.10 shows that around April, May, and September 2012 and July 2013, a buy indication for Amazon.com may have been made as the RSI was above 70 and crossed below 70 during those periods. In contrast, around November 2012, the RSI was below 30 and crossed above 30 during that period, which could be taken as a sell signal.

1.7 Further Reading

Additional references to learn more details about other R commands and functionality can be found in Zuur and Meesters [4] and Albert and Rizzo [1].

The technical analysis charts above can be recreated using chartSeries. For example, the 50-day and 200-day moving average example above can be replicated by:

```
> chartSeries(data.AMZN[,4],
+    theme="white.mono",
+    TA=c(addSMA(n=c(50,200))))
> zoomChart("2012::2013")
```

Other technical analysis indicators can be generated by `chartSeries`. For more details, visit the URL: http://www.quantmod.com/examples/charting/. To learn more about technical analysis, a good book is Kirkpatrick and Dahiquist [3].

References

1. Albert, J., & Rizzo, M. (2012). *R by example*. New York: Springer.
2. Chan, E. (2009). *Quantitative trading: How to build your own algorithmic trading business*. New Jersey: Wiley .
3. Kirkpatrick, C., & Dahlquist, J. (2010). *Technical analysis: The complete resource for financial market technicians* (2nd ed.). New Jersey: FT Press.
4. Zuur, A., Ieno, E., & Meesters, E. (2009). *A beginner's guide to R*. New York: Springer.

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,

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

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:

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

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 `Delt` 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

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

where ln is the natural logarithm operator and the rest of the variables are defined the same was 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

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

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,

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

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)),]
```

**IBM Stock Performance Based On
Total Returns and Price Returns
December 31, 2010 - December 31, 2013**
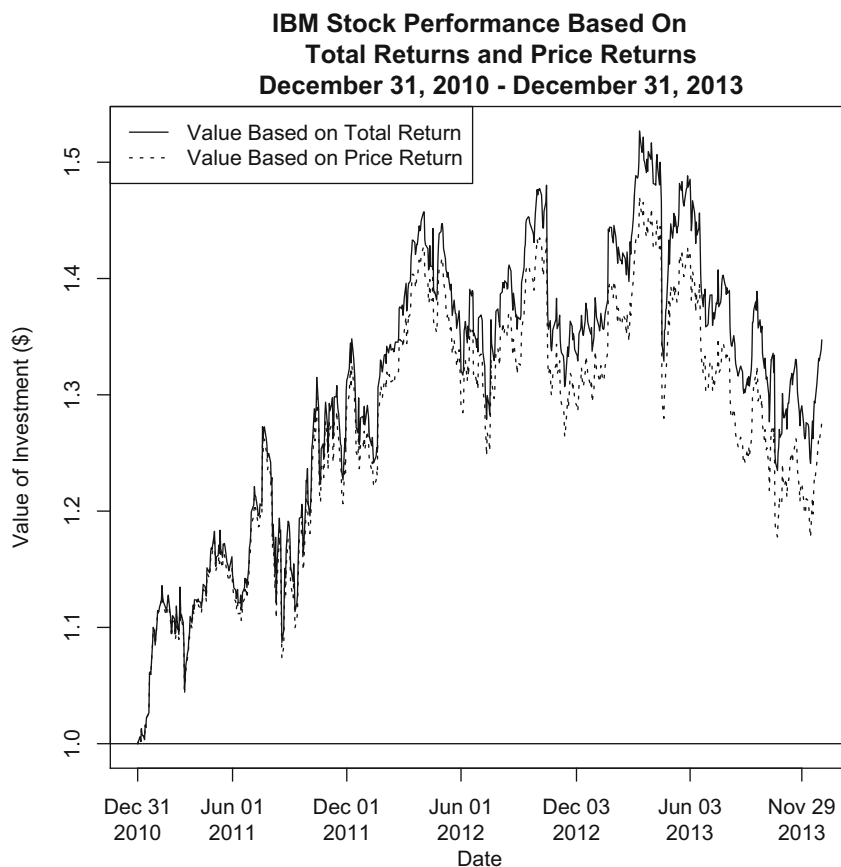
Fig. 2.1 Effect on investment performance of reinvesting dividends. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

```
            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 or 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. There-
fore, 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 index-ing 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 ad-justed 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 1.0000000 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 Decem-ber 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 De-cember 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.

**Value of $1 Invested in AMZN, IBM, YHOO,
And the S&P 500 Index Based on Total Returns
December 31, 2010 - December 31, 2013**
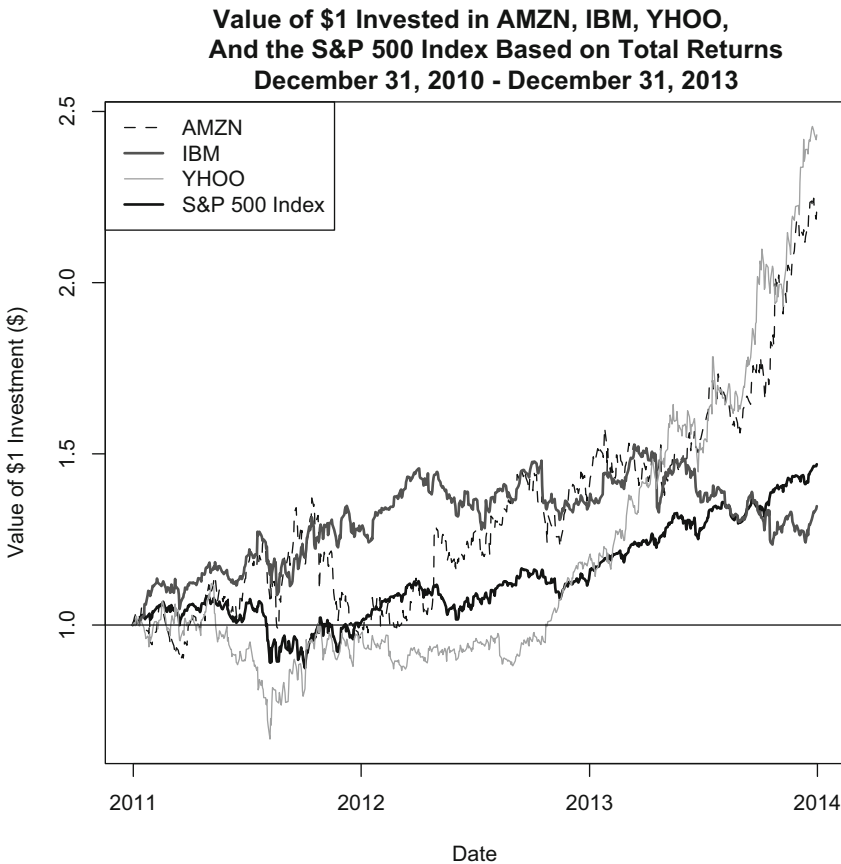
Fig. 2.2 Performance of AMZN, YHOO, IBM, and the S&P 500 index based on total returns, December 31, 2010 to December 31, 2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

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.

Chapter 3
Portfolio Returns

This chapter discusses the returns of a basket of securities or portfolio returns. A portfolio's return is simply the weighted-average return of the securities in the portfolio with the weights given by the amount of money you invest in each security. We demonstrate how this calculation is performed using two approaches: the long way in which all steps are laid out and the use of matrix algebra, which we will later use when performing mean-variance optimization across multiple securities.

We then show how to construct benchmark portfolio returns, which are returns of a hypothetical portfolio that we can use to compare the performance of our portfolio. Two common approaches in benchmark portfolio construction is the use of an equal-weighted portfolio and a value-weighted portfolio with quarterly rebalancing. An equal-weighted portfolio redistributes the total value of the portfolio equally among all the securities in the portfolios on each rebalancing date. In contrast, a value-weighted portfolio redistributes the total value of the portfolio based on the market capitalization of the securities (i.e., larger firms get a bigger share of the pie) on each rebalancing date. Between each rebalancing date, the weights in each security are allowed to move freely depending on their performance regardless of the weighting method used.

3.1 Constructing Portfolio Returns (Long Way)

In prior sections we discussed how to calculate returns of individual securities. Most investors likely have more than one security in their portfolio. In those cases, we may be interested in knowing the *portfolio return*. It turns out that the return on a portfolio of securities is equal to the weighted-average of the returns of the individual securities in the portfolio with the weight calculated as the percentage invested in that security relative to the total amount invested in the portfolio. That is, for a two-asset portfolio, we have

$$r_p = \frac{I_1}{I_1 + I_2} * r_1 + \frac{I_2}{I_1 + I_2} * r_2 = w_1 * r_1 + w_2 * r_2, \qquad (3.1)$$

where r_p is th portfolio return, I_1 is the dollar amount invested in security 1, I_2 is the dollar amount invested in security 2, r_1 is the return of security 1, r_2 is the return of security 2.

In Eq. (3.1), we can then simplify the percentage invested in security 1 as weight 1 and denote it by w_1 and we can simplify the percentage invested in security 2 as weight 2 and denote it by w_2.

With this simplification in mind, we can then extend this 2-asset portfolio return calculation into multiple assets. That is,

$$r_p = w_1 * r_1 + w_2 * r_2 + w_3 * r_3 + w_4 * r_4 + \cdots = \sum_{i=1}^{N} w_i * r_i, \qquad (3.2)$$

where the summation is done over the weighted-returns of each of the N assets in the portfolio.

Note that if we have cash in the portfolio, we can still assign a weight as well as a return to the cash (i.e., some short-term interest rate, such as a money market fund rate). Since all assets (including cash) are assigned weights, the sum of those weights must still equal one (i.e., equal to 100 % of the portfolio).

Continuing from our previous example, suppose we invested $ 50,000 in AMZN, $ 10,000 in GSPC, $ 30,000 in YHOO, and $ 10,000 in IBM. How much would our portfolio return be as of December 31, 2013 if we made the investment in those securities on December 31, 2010?

Step 1: Find First and Last Adjusted Closing Price for Each Security Over the Investment Period The data object `multi` calculated earlier contains the adjusted prices of AMZN, GSPC, YHOO, and IBM. Since we only need the first and last observation in the data object, we can create a new data object `period.ret` that contains the observation for December 31, 2010 and December 31, 2013.

```
> period.ret<-multi[c(1,nrow(multi)),]
> period.ret
           AMZN.Adjusted GSPC.Adjusted YHOO.Adjusted IBM.Adjusted
2010-12-31        180.00       1257.64         16.63       139.23
2013-12-31        398.79       1848.36         40.44       187.57
```

Step 2: Calculate Returns for Each Security Over the Investment Period The total return over the 3-year period December 31, 2010 to December 31, 2013 is equal to the percentage change in the adjusted closing price over those two dates. Earlier we used the `Delt` command to calculate the percentage change between consecutive observations in a time series. Since we only have two observations in `rets` (although they are temporally 3 years apart), we can use the `Delt` command. To apply the `Delt` command to all four securities, we can use the `lapply` command. The `lapply` command applies a function to each of the variables in the data object. In this case, we are applying `Delt` to each variable. The output below shows that we get the *net* total return for each security in a `list` object.

```
> rets<-lapply(period.ret,Delt)
> rets
$AMZN.Adjusted
           Delt.1.arithmetic
2010-12-31                NA
2013-12-31            1.2155

$GSPC.Adjusted
           Delt.1.arithmetic
2010-12-31                NA
2013-12-31         0.4697052

$YHOO.Adjusted
           Delt.1.arithmetic
2010-12-31                NA
2013-12-31           1.43175

$IBM.Adjusted
           Delt.1.arithmetic
2010-12-31                NA
2013-12-31         0.3471953
```

Step 3: Convert to a `data.frame` **and Cleanup Data** Since the returns data is in a `list` object, we have to convert it to a `data.frame` to continue with our calculations. We convert `rets` into a data frame using the `data.frame` command.

```
> rets<-data.frame(rets)
> rets
           Delt.1.arithmetic Delt.1.arithmetic.1 Delt.1.arithmetic.2
2010-12-31                NA                  NA                  NA
2013-12-31            1.2155           0.4697052             1.43175
           Delt.1.arithmetic.3
2010-12-31                  NA
2013-12-31           0.3471953
```

We then clean up the data by deleting the row labeled December 31, 2010 as it is just a row with "NA"s. We also convert the values into percentage points, so it is easier to read. In addition, we also rename the variables to make the names more meaningful.

```
> rets<-rets[2,]*100
> names(rets)<-paste(c("AMZN","GSPC","YHOO","IBM"))
> rets
              AMZN     GSPC    YHOO      IBM
2013-12-31 121.55 46.97052 143.175 34.71953
```

Step 4: Calculate Weight of Each Security in the Portfolio To calculate the weights, we create variables representing the dollar amount of investment in each security. We then calculate the weight of each security by calculating the percentage invested in that security divided by the total amount invested, which in this case is

equal to $ 100,000. As we can see, AMZN has a 50 % weight, GSPC has a 10 %
weight, YHOO has a 30 % weight, and IBM has a 10 % weight.

```
> i.AMZN<-50000
> i.GSPC<-10000
> i.YHOO<-30000
> i.IBM<-10000
> w.AMZN<-i.AMZN/(i.AMZN+i.GSPC+i.YHOO+i.IBM)
> w.AMZN
[1] 0.5
> w.GSPC<-i.GSPC/(i.AMZN+i.GSPC+i.YHOO+i.IBM)
> w.GSPC
[1] 0.1
> w.YHOO<-i.YHOO/(i.AMZN+i.GSPC+i.YHOO+i.IBM)
> w.YHOO
[1] 0.3
> w.IBM<-i.IBM/(i.AMZN+i.GSPC+i.YHOO+i.IBM)
> w.IBM
[1] 0.1
```

Step 5: Calculate Portfolio Return After calculating the weights, we can now
calculate the returns. The portfolio return is the weighted average return of each
security in the portfolio. As the output shows, the portfolio return over the period is
112 %.

```
> port.ret.4asset<-w.AMZN*rets$AMZN+w.GSPC*rets$GSPC+
+    w.YHOO*rets$YHOO+w.IBM*rets$IBM
> port.ret.4asset
[1] 111.8965
```

3.2 Constructing Portfolio Returns (Matrix Algebra)

As the number of securities in our portfolio grows, it will be too cumbersome to
implement Eq. (3.2). A convenient way to calculate portfolio returns when there are
many securities in the portfolio is to use matrix algebra. To implement this, we have
to calculate two vectors, where a vector can be thought of as a series of numbers
organized as either a row or a column. The first is a vector of weights, which in this
case is 50 % for AMZN, 10 % for GSPC, 30 % for YHOO, and 10 % for IBM. We
create this using the c(...) operator and we convert this to a matrix with one
row using the matrix command with the second argument equal to one (i.e., a row
vector).

```
> wgt<-c(0.5,0.1,0.3,0.1)
> mat.wgt<-matrix(wgt,1)
> mat.wgt
     [,1] [,2] [,3] [,4]
[1,]  0.5  0.1  0.3  0.1
```

The second is a column vector of returns. We again use the c(...) operator but this time type in a four in the second argument in the matrix command. This tells R that the elements inside the c(...) operator should be divided into four rows.

```
> ret<-c(rets$AMZN,rets$GSPC,rets$YHOO,rets$IBM)
> mat.ret<-matrix(ret,4)
> mat.ret
           [,1]
[1,] 121.55000
[2,]  46.97052
[3,] 143.17498
[4,]  34.71953
```

We can now calculate the portfolio return by multiplying the row vector of weights by the column vector of returns using the matrix multiplication operator %*%. Unlike normal multiplication in which $x * y = y * x$, this is not the case with matrix multiplication. In fact, the vectors need to be "conformable" for matrix multiplication to work. A detailed discussion of matrix multiplication is beyond the scope of this book. For our purposes, what is important is we multiply the weight matrix (left side) by the return matrix (right side) to get the identical answer as what we calculated using the manual approach earlier.

```
> port.ret<-mat.wgt %*% mat.ret
> port.ret
         [,1]
[1,] 111.8965
```

3.3 Constructing Benchmark Portfolio Returns

To determine whether our investment is performing well, we compare the returns of our investment to that of a benchmark. Typically, some index of comparable securities are used as the benchmark. Some benchmarks are readily-available (e.g., S&P 500 Index for large capitalization stocks). Alternatively, benchmarks can be constructed. In this section, we demonstrate how to construct an equal-weighted (EW) index and value-weighted (VW) index. We first setup the data we need to construct the indexes.

Step 1: Importing the Price Data In this section, we will construct a hypothetical EW and VW portfolio consisting of AMZ, YHOO, and IBM stock using returns in 2013. We now create a common data object that contains the data necessary to construct the EW and VW portfolios. Since we have called in the original Yahoo

Finance data for all three securities earlier, we only need to make sure that the data is still in the R workspace or memory.

```
> 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.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
>
> 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: Create Object with Only the Relevant Data   We then combine the adjusted and closing prices of the three securities.

```
> port<-data.AMZN[,c(4,6)]
> port<-merge(port,data.YHOO[,c(4,6)])
> port<-merge(port,data.IBM[,c(4,6)])
> port[c(1:3,nrow(port)),]
           AMZN.Close AMZN.Adjusted YHOO.Close YHOO.Adjusted IBM.Close
2010-12-31     180.00        180.00      16.63         16.63    146.76
2011-01-03     184.22        184.22      16.75         16.75    147.48
2011-01-04     185.01        185.01      16.59         16.59    147.64
2013-12-31     398.79        398.79      40.44         40.44    187.57
           IBM.Adjusted
2010-12-31       139.23
2011-01-03       139.91
2011-01-04       140.06
2013-12-31       187.57
```

Step 3: Calculate Returns of Each Security   Using the adjusted close prices, we calculate the total return of AMZN, YHOO, and IBM.

```
> port$AMZN.ret<-Delt(port$AMZN.Adjusted)
> port$YHOO.ret<-Delt(port$YHOO.Adjusted)
> port$IBM.ret<-Delt(port$IBM.Adjusted)
> port[c(1:3,nrow(port)),]
           AMZN.Close AMZN.Adjusted YHOO.Close YHOO.Adjusted IBM.Close
2010-12-31     180.00        180.00      16.63         16.63    146.76
2011-01-03     184.22        184.22      16.75         16.75    147.48
2011-01-04     185.01        185.01      16.59         16.59    147.64
2013-12-31     398.79        398.79      40.44         40.44    187.57
           IBM.Adjusted      AMZN.ret      YHOO.ret       IBM.ret
2010-12-31       139.23            NA            NA            NA
2011-01-03       139.91   0.023444444   0.007215875   0.004884005
2011-01-04       140.06   0.004288351  -0.009552239   0.001072118
2013-12-31       187.57   0.013778377   0.005970149   0.006222842
```

Step 4: Convert to `data.frame` **Object and Subset Data** We follow the same technique we discussed in subsetting data using dates in Chap. 1 to subset `port` to only include data from December 31, 2012 to December 31, 2013. We leave the December 31, 2010 data in because, looking ahead, we need that observation as a placeholder for later calculations.

```
> port<-cbind(data.frame(index(port)),
+       data.frame(port))
> names(port)[1]<-paste("date")
> port[c(1:3,nrow(port)),]
                 date AMZN.Close AMZN.Adjusted YHOO.Close YHOO.Adjusted
2010-12-31 2010-12-31     180.00        180.00      16.63         16.63
2011-01-03 2011-01-03     184.22        184.22      16.75         16.75
2011-01-04 2011-01-04     185.01        185.01      16.59         16.59
2013-12-31 2013-12-31     398.79        398.79      40.44         40.44
           IBM.Close IBM.Adjusted      AMZN.ret      YHOO.ret       IBM.ret
2010-12-31    146.76       139.23            NA            NA            NA
2011-01-03    147.48       139.91   0.023444444   0.007215875   0.004884005
2011-01-04    147.64       140.06   0.004288351  -0.009552239   0.001072118
2013-12-31    187.57       187.57   0.013778377   0.005970149   0.006222842
>
> port<-subset(port,
+       port$date >= "2012-12-31" &
+       port$date <= "2013-12-31")
> port[c(1:3,nrow(port)),]
                 date AMZN.Close AMZN.Adjusted YHOO.Close YHOO.Adjusted
2012-12-31 2012-12-31     250.87        250.87      19.90         19.90
2013-01-02 2013-01-02     257.31        257.31      20.08         20.08
2013-01-03 2013-01-03     258.48        258.48      19.78         19.78
2013-12-31 2013-12-31     398.79        398.79      40.44         40.44
           IBM.Close IBM.Adjusted      AMZN.ret      YHOO.ret       IBM.ret
2012-12-31    191.55       187.90   0.023207439   0.020512821   0.009021587
2013-01-02    196.35       192.61   0.025670666   0.009045226   0.025066525
2013-01-03    195.27       191.55   0.004547044  -0.014940239  -0.005503349
2013-12-31    187.57       187.57   0.013778377   0.005970149   0.006222842
```

As we can see by now, some of the techniques we are implementing in this text are variations from prior code we have written. Therefore, by using the R Editor when we write our code, we can save time by copying and pasting from previously written code. However, reporting the above output for purposes of this text is important because it displays the intermediate output of the data we are currently using. This serves as a guide to make sure we are following the code properly. We now turn to discussing the specific steps used to construct an EW portfolio and a VW portfolio.

3.3.1   Equal-Weighted Portfolio

An equal-weighted (EW) index gives equal weight to small firms and large firms. As such, an EW index is sometimes considered as an approach to capture the small capitalization stock premium (i.e., the belief that small capitalization stocks yield higher returns over large capitalization stocks).[1] An example of EW indexes is the S&P 500 Equal Weight Index, which Standard & Poor's created in response to needs of the market for an "official" EW index version of the S&P 500 Index, and those provided by MSCI for developed markets, emerging markets, and all countries.[2] These indexes are rebalanced quarterly, which means that in between quarters the constituent weights are allowed to fluctuate based on their performance. We follow this quarterly rebalancing approach.

Quarterly Rebalancing Explained

The quarterly rebalancing approach means that on each rebalancing date, the weight of each firm is set or reset to $1/N$ where N is the number of securities in the portfolio. For example, if we have 3 firms in our portfolio and $ 1000 to invest, each firm will be allocated $ 333.33 [= $ 1000/3] or 33.33 % of the portfolio value. Subsequent to this date, each $ 333.33 investment would be allowed to grow at the returns of that security. Then, on each rebalancing date, if we have the same number of firms, we will allocate 33.33 % of the portfolio's then-existing value to each firm. For example, if the aggregate portfolio value as of March 31, 2013 was $ 1500, then each of the three firms would then be

---

[1] There is no consensus in the literature as to whether a small cap premium still exists. There is literature that shows the small stock premium disappeared after it was discovered several decades ago, while there is literature that supports the use of a small stock premium even through the present.

[2] S&P 500 Equal Weight Index Methodology, April 2012, retrieved from http://us.spindices. com/documents/methodologies/methodology-sp-500-equal-weight-index.pdf on January 3, 2013 and MSCI Equal Weight Indices Methodology, May 2011 retrieved from http://www.msci. com/eqb/methodology/meth_docs/MSCI_Equal_Weighted_Indices_Methodology _May11.pdf on January 3, 2014.

allocated $ 500 [=$ 1500 / 3]. Suppose that Stock 1 had a value of $ 200, Stock 2 had a value of $ 500, and Stock 3 had a value of $ 800 at quarter end. Then, this means we would have to sell $ 300 of Stock 3 and purchase $ 300 of Stock 1. That way, each of the three stocks in the portfolio would have an investment of $ 500 after rebalancing.

We now turn to the mechanics of constructing an EW portfolio.

Step 1: Keep Only Variables We Need to Construct EW Portfolio We first take from `port` only the variables that we need, which are the returns of AMZN, YHOO, and IBM. To make calling in the variables easier, we also rename the return variables to only be the tickers. In addition, we renumber the index to denote the observation number.

```
> ewport<-port[c(1,8:10)]
> ewport[c(1:3,nrow(ewport)),]
                date    AMZN.ret     YHOO.ret      IBM.ret
2012-12-31 2012-12-31 0.023207439  0.020512821  0.009021587
2013-01-02 2013-01-02 0.025670666  0.009045226  0.025066525
2013-01-03 2013-01-03 0.004547044 -0.014940239 -0.005503349
2013-12-31 2013-12-31 0.013778377  0.005970149  0.006222842
>
> names(ewport)<-paste(c("date","AMZN","YHOO","IBM"))
> rownames(ewport)<-seq(1:nrow(ewport))
> ewport[c(1:3,nrow(ewport)),]
          date        AMZN         YHOO          IBM
1   2012-12-31 0.023207439  0.020512821  0.009021587
2   2013-01-02 0.025670666  0.009045226  0.025066525
3   2013-01-03 0.004547044 -0.014940239 -0.005503349
253 2013-12-31 0.013778377  0.005970149  0.006222842
```

Step 2: Converting Net Returns to Gross Returns We then convert the data to gross returns, by adding one to each security's returns. Note that to reduce the number of variables we use, we are overwriting the net return series. We need to be careful when overwriting a series so as not to lose any information. Since gross returns are simply one plus the net return, we can glean from the gross return series what we would need from the net return series.

```
> ewport$AMZN<-1+ewport$AMZN
> ewport$YHOO<-1+ewport$YHOO
> ewport$IBM<-1+ewport$IBM
> ewport[c(1:3,nrow(ewport)),]
          date     AMZN      YHOO       IBM
1   2012-12-31 1.023207 1.0205128 1.0090216
2   2013-01-02 1.025671 1.0090452 1.0250665
3   2013-01-03 1.004547 0.9850598 0.9944967
253 2013-12-31 1.013778 1.0059701 1.0062228
```

Step 3: Calculate EW Portfolio Values for 1Q 2013 Now, we shall start with the
EW portfolio values in the first quarter of 2013. As such, we subset ewport by
data from December 31, 2012 to March 31, 2013. The reason why we would want
to include December 31, 2012 is because, when we want to determine how much
we made based on the returns in 2013, we are implicitly making the assumption that
the investment was made at the closing price on the last trading day of 2012 (i.e.,
December 31, 2012).

```
> ew.q1<-subset(ewport,
+    ewport$date >= as.Date("2012-12-31") &
+    ewport$date <= as.Date("2013-03-31"))
> ew.q1[c(1:3,nrow(ew.q1)),]
        date     AMZN      YHOO        IBM
1  2012-12-31 1.023207 1.0205128 1.0090216
2  2013-01-02 1.025671 1.0090452 1.0250665
3  2013-01-03 1.004547 0.9850598 0.9944967
61 2013-03-28 1.004485 0.9974565 1.0114079
```

Although we need December 31, 2012, we are not interested in the return on that
date. Our main interest for December 31, 2012 is for a placeholder to begin our
indexing of each security's investment performance to $ 1. As such, we overwrite
the gross return values for each security on December 31, 2013 to 1.00. We can
then use the cumprod command to calculate the cumulative gross returns for each
security during the quarter.

```
> ew.q1[1,2:4]<-1
> ew.q1$AMZN<-cumprod(ew.q1$AMZN)
> ew.q1$YHOO<-cumprod(ew.q1$YHOO)
> ew.q1$IBM<-cumprod(ew.q1$IBM)
> ew.q1[c(1:3,nrow(ew.q1)),]
        date     AMZN      YHOO      IBM
1  2012-12-31 1.000000 1.0000000 1.000000
2  2013-01-02 1.025671 1.0090452 1.025067
3  2013-01-03 1.030334 0.9939698 1.019425
61 2013-03-28 1.062263 1.1824121 1.118254
```

We then setup a variable to denote the number of securities in the portfolio that we
can call in for each of the quarterly calculations.

```
> num.sec<-3
```

Note that we use the variable num.sec instead of the number "3" because in the
program it is easier to identify a variable instead of a number. For example, if we
decide to have four securities, we simply make a change in one spot of the program
instead of having to track down each part of the code that uses the number "3" to
identify the number of securities in the portfolio.

Since in our example each stock has 33 % of the portfolio value, we multiply the
cumulative gross returns of each security by 33 % or 1/num.sec to get the index

value for each security during the quarter. The variables that hold the index values have a suffix of `idx`.

```
> ew.q1$AMZN.idx<-(1/num.sec)*ew.q1$AMZN
> ew.q1$YHOO.idx<-(1/num.sec)*ew.q1$YHOO
> ew.q1$IBM.idx<-(1/num.sec)*ew.q1$IBM
> ew.q1[c(1:3,nrow(ew.q1)),]
         date     AMZN      YHOO      IBM AMZN.idx YHOO.idx  IBM.idx
1  2012-12-31 1.000000 1.0000000 1.000000 0.3333333 0.3333333 0.3333333
2  2013-01-02 1.025671 1.0090452 1.025067 0.3418902 0.3363484 0.3416888
3  2013-01-03 1.030334 0.9939698 1.019425 0.3434448 0.3313233 0.3398084
61 2013-03-28 1.062263 1.1824121 1.118254 0.3540878 0.3941374 0.3727515
```

The value of a portfolio is equal to the value of the components of the portfolio. Hence, to calculate the value of the EW portfolio on each day, we sum the values of the three index variables. We can see that on December 31, 2012, the value of the portfolio is $ 1.000, on January 2, 2013 the value of the portfolio is $ 1.020, on January 3, 2013 the value of the portfolio is $ 1.015, and so on until the end of the quarter on March 28, 2013, where the value of the portfolio is $ 1.121.

```
> q1.val<-data.frame(rowSums(ew.q1[,5:7]))
> q1.val[c(1:3,nrow(q1.val)),]
[1] 1.000000 1.019927 1.014577 1.120977
> names(q1.val)<-paste("port.val")
> q1.val$date<-ew.q1$date
> q1.val[c(1:3,nrow(q1.val)),]
    port.val       date
1   1.000000 2012-12-31
2   1.019927 2013-01-02
3   1.014577 2013-01-03
61  1.120977 2013-03-28
>
> q2.inv<-q1.val[nrow(q1.val),1]
> q2.inv
[1] 1.120977
```

In the above code, we created the `q1.val` and `q2.inv` variables. The purpose of `q1.val` is to hold the portfolio values for the first quarter. We will then combine all quarterly portfolio values at the end so that we have a complete series of EW portfolio values. `q2.inv` is created to hold the aggregate portfolio value at the end of the first quarter. This value will be redistributed equally among the three stocks at the beginning of the second quarter. As such, we will use `q2.inv` value of $ 1.121 in the second quarter calculations as the starting investment value instead of $ 1.

Step 4: Calculate EW Portfolio Values for 2Q 2013 The steps for the second quarter are pretty similar to that of the first quarter calculation except for this one change (and changing references from the first quarter to the second quarter).

```
> ew.q2<-subset(ewport,
+     ewport$date >= as.Date("2013-04-01") &
+     ewport$date <= as.Date("2013-06-30"))
> ew.q2[c(1:3,nrow(ew.q2)),]
          date      AMZN      YHOO       IBM
62  2013-04-01 0.9816879 0.9987250 0.9956691
63  2013-04-02 1.0065364 1.0119149 1.0093208
64  2013-04-03 0.9837080 0.9831791 0.9920913
125 2013-06-28 1.0005044 0.9866510 0.9767610
>
> ew.q2$AMZN<-cumprod(ew.q2$AMZN)
> ew.q2$YHOO<-cumprod(ew.q2$YHOO)
> ew.q2$IBM<-cumprod(ew.q2$IBM)
> ew.q2[c(1:3,nrow(ew.q2)),]
          date      AMZN      YHOO       IBM
62  2013-04-01 0.9816879 0.9987250 0.9956691
63  2013-04-02 0.9881046 1.0106247 1.0049496
64  2013-04-03 0.9720065 0.9936252 0.9970017
125 2013-06-28 1.0420278 1.0679983 0.9001523
>
> ew.q2$AMZN.idx<-(q2.inv/num.sec)*ew.q2$AMZN
> ew.q2$YHOO.idx<-(q2.inv/num.sec)*ew.q2$YHOO
> ew.q2$IBM.idx<-(q2.inv/num.sec)*ew.q2$IBM
> ew.q2[c(1:3,nrow(ew.q2)),]


          date      AMZN      YHOO       IBM AMZN.idx  YHOO.idx   IBM.idx
62  2013-04-01 0.9816879 0.9987250 0.9956691 0.3668164 0.3731825 0.3720406
63  2013-04-02 0.9881046 1.0106247 1.0049496 0.3692140 0.3776289 0.3755083
64  2013-04-03 0.9720065 0.9936252 0.9970017 0.3631988 0.3712768 0.3725385
125 2013-06-28 1.0420278 1.0679983 0.9001523 0.3893629 0.3990670 0.3363499
>
> q2.val<-data.frame(rowSums(ew.q2[,5:7]))
> q2.val[c(1:3,nrow(q2.val)),]
[1] 1.112039 1.122351 1.107014 1.124780
> names(q2.val)<-paste("port.val")
> q2.val$date<-ew.q2$date
> q2.val[c(1:3,nrow(q2.val)),]
     port.val       date
62   1.112039 2013-04-01
63   1.122351 2013-04-02
64   1.107014 2013-04-03
125  1.124780 2013-06-28
>
> q3.inv<-q2.val[nrow(q2.val),1]
> q3.inv
[1] 1.12478
```

The above calculations for the second quarter shows that the portfolio has grown
from $ 1.121 at the end of the first quarter to $ 1.125 at the end of the second quarter.
We then use $ 1.125 as the portfolio value that will be allocated evenly across the
three securities at the start of the third quarter.

Step 5: Calculate EW Portfolio Values for 3Q 2013 Again, the technique for constructing the third quarter EW portfolio values is similar to the code for the first and second quarters.

```
> ew.q3<-subset(ewport,
+    ewport$date >= as.Date("2013-07-01") &
+    ewport$date <= as.Date("2013-09-30"))
> ew.q3[c(1:3,nrow(ew.q3)),]
         date      AMZN      YHOO        IBM
126 2013-07-01 1.0158810 1.0043772 1.0008988
127 2013-07-02 1.0057781 0.9900951 1.0011621
128 2013-07-03 1.0010573 1.0240096 1.0091278
189 2013-09-30 0.9893358 0.9886736 0.9906949
>
> ew.q3$AMZN<-cumprod(ew.q3$AMZN)
> ew.q3$YHOO<-cumprod(ew.q3$YHOO)
> ew.q3$IBM<-cumprod(ew.q3$IBM)
> ew.q3[c(1:3,nrow(ew.q3)),]
         date     AMZN     YHOO       IBM
126 2013-07-01 1.015881 1.004377 1.0008988
127 2013-07-02 1.021751 0.994429 1.0020620
128 2013-07-03 1.022831 1.018305 1.0112086
189 2013-09-30 1.125860 1.319936 0.9738289
>
> ew.q3$AMZN.idx<-(q3.inv/num.sec)*ew.q3$AMZN
> ew.q3$YHOO.idx<-(q3.inv/num.sec)*ew.q3$YHOO
> ew.q3$IBM.idx<-(q3.inv/num.sec)*ew.q3$IBM
> ew.q3[c(1:3,nrow(ew.q3)),]
         date     AMZN     YHOO       IBM AMZN.idx YHOO.idx  IBM.idx
126 2013-07-01 1.015881 1.004377 1.0008988 0.3808808 0.3765678 0.3752636
127 2013-07-02 1.021751 0.994429 1.0020620 0.3830816 0.3728379 0.3756997
128 2013-07-03 1.022831 1.018305 1.0112086 0.3834866 0.3817896 0.3791290
189 2013-09-30 1.125860 1.319936 0.9738289 0.4221148 0.4948793 0.3651144
>
> q3.val<-data.frame(rowSums(ew.q3[,5:7]))
> q3.val[c(1:3,nrow(q3.val)),]
[1] 1.132712 1.131619 1.144405 1.282108
> names(q3.val)<-paste("port.val")
> q3.val$date<-ew.q3$date
> q3.val[c(1:3,nrow(q3.val)),]
     port.val       date
126 1.132712 2013-07-01
127 1.131619 2013-07-02
128 1.144405 2013-07-03
189 1.282108 2013-09-30
>
> q4.inv<-q3.val[nrow(q3.val),1]
> q4.inv
[1] 1.282108
```

As we can see, the EW Portfolio has now increased from $ 1.125 at the start of the second quarter to $ 1.282 by the end of the third quarter. We now take this value as the starting point for the fourth quarter.

Step 6: Calculate EW Portfolio Values for 4Q 2013 The technique to construct the fourth quarter EW portfolio values is similar to the prior quarters.

```
> ew.q4<-subset(ewport,
+    ewport$date >= as.Date("2013-10-01") &
+    ewport$date <= as.Date("2013-12-31"))
> ew.q4[c(1:3,nrow(ew.q4)),]
        date      AMZN      YHOO       IBM
190 2013-10-01 1.0265801 1.0343684 1.0064607
191 2013-10-02 0.9986291 0.9950452 0.9923940
192 2013-10-03 0.9820598 0.9923843 0.9940751
253 2013-12-31 1.0137784 1.0059701 1.0062228
>
> ew.q4$AMZN<-cumprod(ew.q4$AMZN)
> ew.q4$YHOO<-cumprod(ew.q4$YHOO)
> ew.q4$IBM<-cumprod(ew.q4$IBM)
> ew.q4[c(1:3,nrow(ew.q4)),]
        date     AMZN     YHOO       IBM
190 2013-10-01 1.026580 1.034368 1.0064607
191 2013-10-02 1.025173 1.029243 0.9988056
192 2013-10-03 1.006781 1.021405 0.9928878
253 2013-12-31 1.275557 1.219174 1.0183506
>
> ew.q4$AMZN.idx<-(q4.inv/num.sec)*ew.q4$AMZN
> ew.q4$YHOO.idx<-(q4.inv/num.sec)*ew.q4$YHOO
> ew.q4$IBM.idx<-(q4.inv/num.sec)*ew.q4$IBM
> ew.q4[c(1:3,nrow(ew.q4)),]
        date     AMZN     YHOO       IBM AMZN.idx YHOO.idx  IBM.idx
190 2013-10-01 1.026580 1.034368 1.0064607 0.4387290 0.4420575 0.4301306
191 2013-10-02 1.025173 1.029243 0.9988056 0.4381275 0.4398672 0.4268590
192 2013-10-03 1.006781 1.021405 0.9928878 0.4302675 0.4365173 0.4243299
253 2013-12-31 1.275557 1.219174 1.0183506 0.5451339 0.5210377 0.4352120
>
> q4.val<-data.frame(rowSums(ew.q4[,5:7]))
> q4.val[c(1:3,nrow(q4.val)),]
[1] 1.310917 1.304854 1.291115 1.501384
> names(q4.val)<-paste("port.val")
> q4.val$date<-ew.q4$date
> q4.val[c(1:3,nrow(q4.val)),]
    port.val       date
190 1.310917 2013-10-01
191 1.304854 2013-10-02
192 1.291115 2013-10-03
253 1.501384 2013-12-31
```

We now see that in the fourth quarter of 2013, the EW portfolio value jumps substantially from $ 1.282 to 1.501. What this means is that during the first three quarters of the year, the EW portfolio increased by 28.2 % but during the last quarter, the EW portfolio value increased another 17.1 % [= 1.501/1.282−1].

Programming Tip Although the code seems long, the code for all four quarters is very similar. Therefore, using the R editor and applying copying & pasting techniques can help save us time. We only need to be careful that we modify the portions of the code that pertain to the subsequent quarter, such as the appropriate quarterly variable names we are calling in.

Step 7: Combine Quarterly EW Portfolio Values into One Data Object Now, we can combine the four quarterly EW portfolio values into one data object `ew.port val` using the `rbind` command. The `rbind` command allows us to stack the four data objects one on top of the other.

```
> ew.portval<-rbind(q1.val,q2.val,q3.val,q4.val)
> ew.portval[c(1:3,nrow(ew.portval)),]
    port.val     date
1   1.000000 2012-12-31
2   1.019927 2013-01-02
3   1.014577 2013-01-03
253 1.501384 2013-12-31
```

The output here summarizes the performance of an EW portfolio in which money is invested at the closing prices on December 31, 2012 and was held through the end of December 31, 2013. As the output shows, the portfolio value grew by 50.1 % over this time period based on equal-weighted returns of the constituents of the portfolio and assuming quarterly rebalancing.

Much of the code is a result of needing to perform quarterly rebalancing. Therefore, we need to track the portfolio value through the end of each quarter and use that as the starting value in a subsequent quarter. This work is necessary to make the portfolio more realistic. Quarterly rebalancing makes more practical sense as we incur transactions costs only during the rebalancing dates. This makes the use of this EW portfolio more suitable as a benchmark because it is realistically implementable. The use of benchmarks that ignore the possible effects of transactions costs eating away at the returns should be avoided. Having said that, the return calculations above can still be made more realistic. However, depending on the purpose of the benchmarking exercise, the gains from including more assumptions have to be weighed against the additional complexity it brings.

3.3.2 Value-Weighted Portfolio

An alternative way to construct a portfolio of securities is to use *value-weighted returns* or *capitalization-weighted returns*. Some of the major indexes use some

form of value-weighting, such as the S&P 500 Index.[3] In a VW portfolio, the returns of larger firms are given more weight. In the context of say a VW sector index, one can think of capitalization-weighting as an approach to tracking the changes in the size of that sector.

A value-weighted return r_{VW} is constructed as follows:

$$r_{t,VW} = \sum_{i=1}^{N} w_{i,t} * r_{i,t},\tag{3.3}$$

where $w_{i,t}$ is the weight of security i at time t and $r_{i,t}$ is the return of asset i and time t. For our purposes, the weight of security i is equal to the market capitalization of security i divided by the market capitalization of all securities in the portfolio. In addition, we only rebalance the weights at the start of each quarter using the prior quarter end's market capitalization data. The reason we use past data is because we want to avoid any look-ahead bias. To see why, to calculate the day t market capitalization, we would need to know the return on day t for that security. So if we use the market capitalization on day t as the weights for day t return, we are assuming perfect foresight as to what the end-of day return would be. This is clearly not realistic. Therefore, the more appropriate approach is to take the prior day's market capitalization, as this information is known, and use those as the weights.

Step 1: Keep Only Variables We Need to Construct VW Portfolio To start constructing a VW portfolio, we first extract from `port` the close and adjusted closing price variables.

```
> vwport<-port[,c(1,2,4,6,8:10)]
> vwport[c(1:3,nrow(vwport)),]
                date AMZN.Close YHOO.Close IBM.Close    AMZN.ret
2012-12-31 2012-12-31     250.87      19.90    191.55 0.023207439
2013-01-02 2013-01-02     257.31      20.08    196.35 0.025670666
2013-01-03 2013-01-03     258.48      19.78    195.27 0.004547044
2013-12-31 2013-12-31     398.79      40.44    187.57 0.013778377
                YHOO.ret        IBM.ret
2012-12-31   0.020512821   0.009021587
2013-01-02   0.009045226   0.025066525
2013-01-03  -0.014940239  -0.005503349
2013-12-31   0.005970149   0.006222842
```

Since we already have a variable called `date`, the index of `vwport` can be changed to an indicator for the observation number (i.e., 1 to the last observation number of `vwport`.

---

[3] An alternative to using market capitalization weights is to use the free-float adjusted market capitalization as weights. For example, MSCI announced on December 10, 2000 that it was going to adjust all its equity indices for free float. MSCI defines free float as "total shares outstanding excluding shares held by strategic investors such as governments, corporations, controlling shareholders, and management, and shares subject to foreign ownership restrictions." Source: MSCI Press Release,

```
> rownames(vwport)<-seq(1:nrow(vwport))
> vwport[c(1:3,nrow(vwport)),]
          date AMZN.Close YHOO.Close IBM.Close    AMZN.ret      YHOO.ret
1   2012-12-31     250.87      19.90    191.55 0.023207439  0.020512821
2   2013-01-02     257.31      20.08    196.35 0.025670666  0.009045226
3   2013-01-03     258.48      19.78    195.27 0.004547044 -0.014940239
253 2013-12-31     398.79      40.44    187.57 0.013778377  0.005970149
        IBM.ret
1    0.009021587
2    0.025066525
3   -0.005503349
253  0.006222842
```

Step 2: Converting Net Returns to Gross Returns Similar to what we did when
we constructed the EW portfolio returns, we overwrite the net returns with gross
returns.

```
> vwport$AMZN.ret<-1+vwport$AMZN.ret
> vwport$YHOO.ret<-1+vwport$YHOO.ret
> vwport$IBM.ret<-1+vwport$IBM.ret
> vwport[c(1:3,nrow(vwport)),]
          date AMZN.Close YHOO.Close IBM.Close AMZN.ret  YHOO.ret
1   2012-12-31     250.87      19.90    191.55 1.023207 1.0205128
2   2013-01-02     257.31      20.08    196.35 1.025671 1.0090452
3   2013-01-03     258.48      19.78    195.27 1.004547 0.9850598
253 2013-12-31     398.79      40.44    187.57 1.013778 1.0059701
      IBM.ret
1   1.0090216
2   1.0250665
3   0.9944967
253 1.0062228
```

Step 3: Calculate the Market Capitalization of Each Security in the Portfolio
Now, we have to construct the market capitalization weights for each security at the
end of each quarter. A security's market cap is equal to its price multiplied by its
shares outstanding. As such, we need to get the data for those two components. Let
us first discuss how to find the price that is applicable at the end of each quarter.

Construct Series of Calendar Days Since the end of the quarter may not be a
trading day, we should create a date series that contains all calendar days between
December 31, 2012 to December 31, 2013. Note that the output shows we have dates
that include non-trading days such as January 1, 2013, which is New Year's Day and
is a stock market trading holiday.

"MSCI to Adjust for Free Float and to Increase Coverage to 85 %" December 10, 2000 retrieved
from http://www.msci.com/eqb/pressreleases/archive/20001210_pr01.pdf on January 3, 2014.

```
> date<-seq(as.Date("2012-12-31"),as.Date("2013-12-31"),by=1)
> date<-data.frame(date)
> date[c(1:3,nrow(date)),]
[1] "2012-12-31" "2013-01-01" "2013-01-02" "2013-12-31"
```

Create Data Object With Daily Prices, Filling in Last Available Price on Non-trading Days We then extract the prices from vwport.

```
> PRICE.qtr<-vwport[,c(1,2,3,4)]
> PRICE.qtr[c(1:3,nrow(PRICE.qtr)),]
          date AMZN.Close YHOO.Close IBM.Close
1   2012-12-31     250.87      19.90    191.55
2   2013-01-02     257.31      20.08    196.35
3   2013-01-03     258.48      19.78    195.27
253 2013-12-31     398.79      40.44    187.57
```

We then merge date and PRICE.qtr using a combination of the na.locf and merge commands. The merge command combines the two data objects. Using the all.x=TRUE option tells R that when merging we should keep all the data in the "by" variable that is available in x=date data object. The "by" variable in this case is the date. What the na.locf command does is that it copies over the last available data in y=PRICE.qtr when there is a date in x that is not available in y. For example, recall that there is no trading on New Year's Day, January 1, 2013, so the output below shows data on January 1, 2013 which is equal to the closing price on December 31, 2012. Going forward, any time there is a weekend or trading holiday, the last available value is used.

```
> PRICE.qtr<-na.locf(merge(x=date,y=PRICE.qtr,by="date",all.x=TRUE))
> PRICE.qtr[c(1:3,nrow(PRICE.qtr)),]
          date AMZN.Close YHOO.Close IBM.Close
1   2012-12-31     250.87      19.90    191.55
2   2013-01-01     250.87      19.90    191.55
3   2013-01-02     257.31      20.08    196.35
366 2013-12-31     398.79      40.44    187.57
```

Keep Only Prices at the End of Each Calendar Quarter Since we only need the data for the end of the quarter, we can now subset the data to only the four relevant quarter end dates. These are December 31, 2012, March 31, 2013, June 30, 2013, and September 30, 2013. We do not need data for December 31, 2013 because we are not calculating any returns in the first quarter of 2014.

```
> PRICE.qtr<-subset(PRICE.qtr,
+    PRICE.qtr$date==as.Date("2012-12-31") |
+    PRICE.qtr$date==as.Date("2013-03-31") |
+    PRICE.qtr$date==as.Date("2013-06-30") |
+    PRICE.qtr$date==as.Date("2013-09-30"))
> PRICE.qtr
          date AMZN.Close YHOO.Close IBM.Close
1   2012-12-31     250.87      19.90    191.55
91  2013-03-31     266.49      23.53    213.30
182 2013-06-30     277.69      25.13    191.11
274 2013-09-30     312.64      33.17    185.18
```

Note that the last trading day of the first quarter is March 28, 2013 and the last trading day of the second quarter is June 28, 2013. As such, had we not created the daily series of dates, we would not have been able to easily generate the relevant prices as of the quarter end dates that we can then match up with the quarter end shares outstanding for each security below.

Obtain Shares Outstanding Data from SEC Filings   The quarter-end shares outstanding are obtained from the SEC EDGAR database, which contains electronic filings of companies that are required by the SEC.[4] To obtain the shares outstanding data, we use each firm's annual filing (Form 10-K) and quarterly filing (Form 10-Q). There are typically at least two spots in these filings where we can find shares outstanding data. For example, the YHOO Form 10-K for the period ended December 31, 2012 was filed on March 1, 2013 and reports on the cover page of the Form 10-K shares outstanding of 1101 million as of February 15, 2013, while on its balance sheet YHOO reports shares outstanding as of the end of the period (i.e., December 31, 2012) of 1115 million. We choose the quarter end shares outstanding figures and enter it manually below.

```
> PRICE.qtr$AMZN.shout<-c(454000000,455000000,457000000,458000000)
> PRICE.qtr$YHOO.shout<-c(1115233000,1084766000,1065046000,1013059000)
> PRICE.qtr$IBM.shout<-c(1117367676,1108794396,1095425823,1085854383)
> PRICE.qtr
          date AMZN.Close YHOO.Close IBM.Close AMZN.shout YHOO.shout
1   2012-12-31     250.87      19.90    191.55   4.54e+08 1115233000
91  2013-03-31     266.49      23.53    213.30   4.55e+08 1084766000
182 2013-06-30     277.69      25.13    191.11   4.57e+08 1065046000
274 2013-09-30     312.64      33.17    185.18   4.58e+08 1013059000
     IBM.shout
1   1117367676
91  1108794396
182 1095425823
274 1085854383
```

---

[4] http://www.sec.gov/edgar/searchedgar/companysearch.html.

Calculate Market Capitalization of Each Security We now calculate the weights
for each security on each quarter end. We first check the structure of the data
in PRICE.qtr. We see that the date and closing price variables are read-in as
character variables.

```
> str(PRICE.qtr)
'data.frame':   4 obs. of  7 variables:
 $ date     : chr  "2012-12-31" "2013-03-31" "2013-06-30" "2013-09-30"
 $ AMZN.Close: chr  "250.87" "266.49" "277.69" "312.64"
 $ YHOO.Close: chr  "19.90" "23.53" "25.13" "33.17"
 $ IBM.Close : chr  "191.55" "213.30" "191.11" "185.18"
 $ AMZN.shout: num  4.54e+08 4.55e+08 4.57e+08 4.58e+08
 $ YHOO.shout: num  1.12e+09 1.08e+09 1.07e+09 1.01e+09
 $ IBM.shout : num  1.12e+09 1.11e+09 1.10e+09 1.09e+09
```

As such, we would have to convert the date into a date variable using the as.Date
command and convert the closing price variables using the as.numeric command.

```
> PRICE.qtr$date<-as.Date(PRICE.qtr$date)
> PRICE.qtr$AMZN.Close<-as.numeric(PRICE.qtr$AMZN.Close)
> PRICE.qtr$YHOO.Close<-as.numeric(PRICE.qtr$YHOO.Close)
> PRICE.qtr$IBM.Close<-as.numeric(PRICE.qtr$IBM.Close)
> str(PRICE.qtr)
'data.frame':   4 obs. of  7 variables:
 $ date     : Date, format: "2012-12-31" "2013-03-31" ...
 $ AMZN.Close: num  251 266 278 313
 $ YHOO.Close: num  19.9 23.5 25.1 33.2
 $ IBM.Close : num  192 213 191 185
 $ AMZN.shout: num  4.54e+08 4.55e+08 4.57e+08 4.58e+08
 $ YHOO.shout: num  1.12e+09 1.08e+09 1.07e+09 1.01e+09
 $ IBM.shout : num  1.12e+09 1.11e+09 1.10e+09 1.09e+09
```

To make the data object name more sensible to hold the weights of the securities, we
copy the data in PRICE.qtr into weights. We then calculate the market cap for
each security by multiplying that security's closing price with its shares outstanding.

```
> weights<-PRICE.qtr
> weights$AMZN.mcap<-weights$AMZN.Close*weights$AMZN.shout
> weights$YHOO.mcap<-weights$YHOO.Close*weights$YHOO.shout
> weights$IBM.mcap<-weights$IBM.Close*weights$IBM.shout
> weights
          date AMZN.Close YHOO.Close IBM.Close AMZN.shout YHOO.shout
1   2012-12-31     250.87      19.90    191.55   4.54e+08 1115233000
91  2013-03-31     266.49      23.53    213.30   4.55e+08 1084766000
182 2013-06-30     277.69      25.13    191.11   4.57e+08 1065046000
274 2013-09-30     312.64      33.17    185.18   4.58e+08 1013059000
      IBM.shout    AMZN.mcap    YHOO.mcap     IBM.mcap
1   1117367676 113894980000 22193136700 214031778338
91  1108794396 121252950000 25524543980 236505844667
182 1095425823 126904330000 26764605980 209346829034
274 1085854383 143189120000 33603167030 201078514644
```

Calculate Quarter-end Aggregate Market Capitalization Next, we calculate the total market cap on each quarter end date. To do this, we apply the `rowSums` command to the three market capitalization variables, which are in Columns 8–10.

```
> weights$tot.mcap<-rowSums(weights[8:10])
> weights
        date AMZN.Close YHOO.Close IBM.Close AMZN.shout YHOO.shout
1   2012-12-31     250.87      19.90    191.55   4.54e+08 1115233000
91  2013-03-31     266.49      23.53    213.30   4.55e+08 1084766000
182 2013-06-30     277.69      25.13    191.11   4.57e+08 1065046000
274 2013-09-30     312.64      33.17    185.18   4.58e+08 1013059000
      IBM.shout    AMZN.mcap   YHOO.mcap     IBM.mcap    tot.mcap
1   1117367676 113894980000 22193136700 214031778338 350119895038
91  1108794396 121252950000 25524543980 236505844667 383283338647
182 1095425823 126904330000 26764605980 209346829034 363015765014
274 1085854383 143189120000 33603167030 201078514644 377870801674
```

Step 4: Calculate Quarter-end Weights of Each Security in the Portfolio We can now calculate the quarter-end weights for the three securities by dividing each security's market cap by the combined market cap of the three securities.

```
> weights$AMZN.wgt<-weights$AMZN.mcap/weights$tot.mcap
> weights$YHOO.wgt<-weights$YHOO.mcap/weights$tot.mcap
> weights$IBM.wgt<-weights$IBM.mcap/weights$tot.mcap
> weights
        date AMZN.Close YHOO.Close IBM.Close AMZN.shout YHOO.shout
1   2012-12-31     250.87      19.90    191.55   4.54e+08 1115233000
91  2013-03-31     266.49      23.53    213.30   4.55e+08 1084766000
182 2013-06-30     277.69      25.13    191.11   4.57e+08 1065046000
274 2013-09-30     312.64      33.17    185.18   4.58e+08 1013059000
      IBM.shout    AMZN.mcap   YHOO.mcap     IBM.mcap    tot.mcap AMZN.wgt
1   1117367676 113894980000 22193136700 214031778338 350119895038 0.3253028
91  1108794396 121252950000 25524543980 236505844667 383283338647 0.3163533
182 1095425823 126904330000 26764605980 209346829034 363015765014 0.3495835
274 1085854383 143189120000 33603167030 201078514644 377870801674 0.3789367
      YHOO.wgt   IBM.wgt
1   0.06338725 0.6113100
91  0.06659445 0.6170522
182 0.07372849 0.5766880
274 0.08892766 0.5321356
```

We can create a clean data object `WEIGHT` by keeping only the date and the three weight variables in `weights`.

```
> WEIGHT<-weights[,c(1,12:14)]
> WEIGHT
        date  AMZN.wgt   YHOO.wgt   IBM.wgt
1   2012-12-31 0.3253028 0.06338725 0.6113100
91  2013-03-31 0.3163533 0.06659445 0.6170522
182 2013-06-30 0.3495835 0.07372849 0.5766880
274 2013-09-30 0.3789367 0.08892766 0.5321356
```

Since the weights are applicable to the start of the next quarter, we add one to all dates.

```
> WEIGHT$date<-WEIGHT$date+1
> WEIGHT
          date AMZN.wgt   YHOO.wgt    IBM.wgt
1   2013-01-01 0.3253028 0.06338725 0.6113100
91  2013-04-01 0.3163533 0.06659445 0.6170522
182 2013-07-01 0.3495835 0.07372849 0.5766880
274 2013-10-01 0.3789367 0.08892766 0.5321356
```

Comparing the weights above to an equal-weight of 33 %, we can see that AMZN's weight is pretty close to being 1/3 of the weights of the portfolio at the quarter ends. However, IBM is much larger and has weights ranging from 53 to 62 %. Because the sum of the weights have to equal unity, this means YHOO's weights are much smaller, capping out at 9 %.

Step 5: Calculating the Quarterly VW Portfolio Values This approach is similar to what we took in constructing the EW portfolio, in which we calculate the gross returns for each quarter and then apply to it the allocated value of each security based on its quarter-end weights. In the EW portfolio example, the weights were always 33 %. In the VW portfolio, as shown above, the weights used on the quarterly rebalancing dates are different.

We start with creating a series of dates from December 31, 2012 to December 31, 2013 so that we can find the applicable weights at the beginning of each quarter.

```
> date<-seq(as.Date("2012-12-31"),as.Date("2013-12-31"),by=1)
> date<-data.frame(date)
> date[c(1:3,nrow(date)),]
[1] "2012-12-31" "2013-01-01" "2013-01-02" "2013-12-31"
```

We then merge in the `WEIGHT` into `date` using a combination of the `na.locf` command and `merge` command. The `merge` command is implemented with the `all.x=TRUE` option, which means all dates in `date` are kept in the new data object, such that there will be missing values for the weights on all days except for the quarter end dates. However, recall that the `na.locf` command can be used to retain the last available value of weights, such that instead of NAs, we end up using the last available value.

```
> vwret<-na.locf(merge(date,WEIGHT,by="date",all.x=TRUE))
> vwret[c(1:3,nrow(vwret)),]
          date AMZN.wgt   YHOO.wgt    IBM.wgt
1   2012-12-31      <NA>       <NA>       <NA>
2   2013-01-01 0.3253028 0.06338725 0.6113100
3   2013-01-02 0.3253028 0.06338725 0.6113100
366 2013-12-31 0.3789367 0.08892766 0.5321356
```

After merging the data, the variables are all read-in by R as `character` variables. Therefore, to be able to perform arithmetic operations properly, we should convert

these to `numeric` variables. We use the `str` command to show us the structure of
the data. We then convert the date variable using the `as.Date` command and we
convert the other variables using the `as.numeric` command.

```
> str(vwret)
'data.frame':   366 obs. of  4 variables:
 $ date   : chr  "2012-12-31" "2013-01-01" "2013-01-02" "2013-01-03" ...
 $ AMZN.wgt: chr  NA "0.3253028" "0.3253028" "0.3253028" ...
 $ YHOO.wgt: chr  NA "0.06338725" "0.06338725" "0.06338725" ...
 $ IBM.wgt : chr  NA "0.6113100" "0.6113100" "0.6113100" ...
> vwret$date<-as.Date(vwret$date)
> vwret$AMZN.wgt<-as.numeric(vwret$AMZN.wgt)
> vwret$YHOO.wgt<-as.numeric(vwret$YHOO.wgt)
> vwret$IBM.wgt<-as.numeric(vwret$IBM.wgt)
> vwret[c(1:3,nrow(vwret)),]
          date AMZN.wgt   YHOO.wgt  IBM.wgt
1   2012-12-31       NA         NA       NA
2   2013-01-01 0.3253028 0.06338725 0.6113100
3   2013-01-02 0.3253028 0.06338725 0.6113100
366 2013-12-31 0.3789367 0.08892766 0.5321356
> str(vwret)
'data.frame':   366 obs. of  4 variables:
 $ date   : Date, format: "2012-12-31" "2013-01-01" ...
 $ AMZN.wgt: num  NA 0.325 0.325 0.325 0.325 ...
 $ YHOO.wgt: num  NA 0.0634 0.0634 0.0634 0.0634 ...
 $ IBM.wgt : num  NA 0.611 0.611 0.611 0.611 ...
```

Next, we extract the beginning of the quarter weights from `vwret`. To do this, we
use the `subset` command and the first day of the quarter.

```
> q1.vw.wgt<-subset(vwret,vwret$date==as.Date("2013-01-01"))
> q1.vw.wgt
        date AMZN.wgt   YHOO.wgt IBM.wgt
2 2013-01-01 0.3253028 0.06338725 0.61131
>
> q2.vw.wgt<-subset(vwret,vwret$date==as.Date("2013-04-01"))
> q2.vw.wgt
         date AMZN.wgt   YHOO.wgt   IBM.wgt
92 2013-04-01 0.3163533 0.06659445 0.6170522
>
> q3.vw.wgt<-subset(vwret,vwret$date==as.Date("2013-07-01"))
> q3.vw.wgt
          date AMZN.wgt   YHOO.wgt  IBM.wgt
183 2013-07-01 0.3495835 0.07372849 0.576688
>
> q4.vw.wgt<-subset(vwret,vwret$date==as.Date("2013-10-01"))
> q4.vw.wgt
          date AMZN.wgt   YHOO.wgt   IBM.wgt
275 2013-10-01 0.3789367 0.08892766 0.5321356
```

Step 6: Create Pie Charts of the Weights This step is not necessary, but we can
visually show the different weights for each of the four quarters in a pie chart. Note

that Q1 and Q2 are use the variable names as labels, while Q3 and Q4 uses re-labeled values and adds the actual percentage weight to the label. The comments in the code, which are denoted by the pound (#) sign, show where the amount and percent sign are added.

```
> par(mfrow=c(2,2))
>
> Q1.pie.values<-as.numeric(q1.vw.wgt[,-1])
> Q1.pie.labels<-names(q1.vw.wgt[,-1])
> pie(Q1.pie.values,
+     labels=Q1.pie.labels,
+     col=c("black","gray","gray40"),
+     main="Q1 Value Weighting")
>
> Q2.pie.values<-as.numeric(q2.vw.wgt[,-1])
> Q2.pie.labels<-names(q1.vw.wgt[,-1])
> pie(Q2.pie.values,
+     labels=Q2.pie.labels,
+     col=c("black","gray","gray40"),
+     main="Q2 Value Weighting")
>
> Q3.pie.values<-as.numeric(q3.vw.wgt[,-1])
> Q3.pie.labels<-c("Amazon","Yahoo","IBM")
> pct<-round(Q3.pie.values*100)
> Q3.pie.labels<-paste(Q3.pie.labels,pct) # Add Pct to Labels
> Q3.pie.labels<-paste(Q3.pie.labels,"%",sep="") # Add % Sign
> pie(Q3.pie.values,
+     labels=Q3.pie.labels,
+     col=c("black","gray","gray40"),
+     main="Q3 Value Weighting")
>
> Q4.pie.values<-as.numeric(q4.vw.wgt[,-1])
> Q4.pie.labels<-c("Amazon","Yahoo","IBM")
> pct<-round(Q4.pie.values*100)
> Q4.pie.labels<-paste(Q4.pie.labels,pct) # Add Pct to Labels
> Q4.pie.labels<-paste(Q4.pie.labels,"%",sep="") # Add % Sign
> pie(Q4.pie.values,
+     labels=Q4.pie.labels,
+     col=c("black","white","gray40"),
+     main="Q4 Value Weighting")
```

Note that if wanted to plot colored pie charts, we could modify the row that begins with `col` to say something like the following (see third row of the code):

```
> pie(Q4.pie.values,
+     labels=Q4.pie.labels,
+     col=rainbow(length(Q4.pie.labels)),
+     main="Q4 Value Weighting")
```

Figure 3.1 shows the chart generated by the above code. Visually, the weights for all four quarters seem somewhat stable with IBM being the largest of the three, then

Q1 Value Weighting

Q2 Value Weighting

Q3 Value Weighting

Q4 Value Weighting
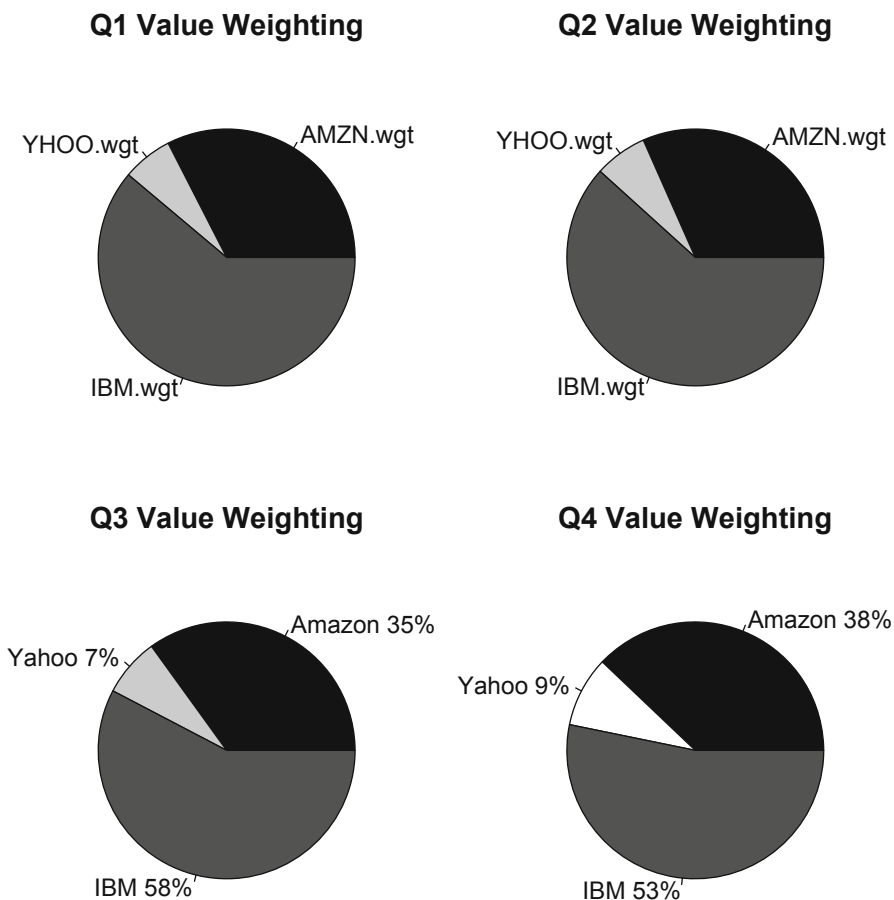
Fig. 3.1 Pie chart of portfolio weights

followed by Amazon.com. Note also that for the Q3 and Q4 pie charts, we modified the label to include the company name and percentage values.

Step 7: Calculating VW Portfolio Values for 1Q 2013 The programming technique we will use is similar to that used in the EW portfolio. We begin with the calculation for the first quarter, by subsetting vwport to only contain data from December 31, 2012 to March 31, 2013. We include December 31, 2012, not because we are interested in the return on that day, but because we are implicitly making the investment at the closing price as of December 31, 2012.

```
> vw.q1<-subset(vwport[,c(1,5:7)],
+    vwport$date >= as.Date("2012-12-31") &
+    vwport$date <= as.Date("2013-03-31"))
> vw.q1[c(1:3,nrow(vw.q1)),]
        date AMZN.ret  YHOO.ret   IBM.ret
```

```
1  2012-12-31 1.023207 1.0205128 1.0090216
2  2013-01-02 1.025671 1.0090452 1.0250665
3  2013-01-03 1.004547 0.9850598 0.9944967
61 2013-03-28 1.004485 0.9974565 1.0114079
```

We then shorten the variable names to make it easier to use.

```
> names(vw.q1)<-paste(c("date","AMZN","YHOO","IBM"))
> vw.q1[c(1:3,nrow(vw.q1)),]
        date     AMZN      YHOO       IBM
1  2012-12-31 1.023207 1.0205128 1.0090216
2  2013-01-02 1.025671 1.0090452 1.0250665
3  2013-01-03 1.004547 0.9850598 0.9944967
61 2013-03-28 1.004485 0.9974565 1.0114079
```

Next, we calculate the cumulative gross return for each security. To do this, we first set the first observation to 1.00, because we are not interested in any returns on December 31, 2012. We then write over the daily gross returns with the cumulative gross returns.

```
> vw.q1[1,2:4]<-1
> vw.q1$AMZN<-cumprod(vw.q1$AMZN)
> vw.q1$YHOO<-cumprod(vw.q1$YHOO)
> vw.q1$IBM<-cumprod(vw.q1$IBM)
> vw.q1[c(1:3,nrow(vw.q1)),]
        date     AMZN      YHOO      IBM
1  2012-12-31 1.000000 1.0000000 1.000000
2  2013-01-02 1.025671 1.0090452 1.025067
3  2013-01-03 1.030334 0.9939698 1.019425
61 2013-03-28 1.062263 1.1824121 1.118254
```

The next step is where we apply the quarter-end weights. In this particular case, we are applying the market capitalization weights for each security based on the price and shares outstanding data available as of December 31, 2012.

```
> vw.q1$AMZN.idx<-(1*q1.vw.wgt$AMZN.wgt)*vw.q1$AMZN
> vw.q1$YHOO.idx<-(1*q1.vw.wgt$YHOO.wgt)*vw.q1$YHOO
> vw.q1$IBM.idx<-(1*q1.vw.wgt$IBM.wgt)*vw.q1$IBM
> vw.q1[c(1:3,nrow(vw.q1)),]
        date     AMZN      YHOO      IBM  AMZN.idx   YHOO.idx   IBM.idx
1  2012-12-31 1.000000 1.0000000 1.000000 0.3253028 0.06338725 0.6113100
2  2013-01-02 1.025671 1.0090452 1.025067 0.3336535 0.06396060 0.6266334
3  2013-01-03 1.030334 0.9939698 1.019425 0.3351707 0.06300502 0.6231848
61 2013-03-28 1.062263 1.1824121 1.118254 0.3455572 0.07494985 0.6836001
```

The final step is to calculate the daily VW portfolio values by summing the daily index values of the three securities. We do this by applying the rowSums command to the fifth through seventh columns.

```
> q1.vw.val<-data.frame(rowSums(vw.q1[,5:7]))
> q1.vw.val[c(1:3,nrow(q1.vw.val)),]
[1] 1.000000 1.024248 1.021361 1.104107
> names(q1.vw.val)<-paste("port.val")
> q1.vw.val$date<-vw.q1$date
> q1.vw.val[c(1:3,nrow(q1.vw.val)),]
   port.val       date
1  1.000000 2012-12-31
2  1.024248 2013-01-02
3  1.021361 2013-01-03
61 1.104107 2013-03-28
>
> q2.vw.inv<-q1.vw.val[nrow(q1.vw.val),1]
> q2.vw.inv
[1] 1.104107
```

We also create `q2.vw.inv` that holds the end of the first quarter value for the VW portfolio of \$ 1.104. This value is then allocated using the end of first quarter weights to calculate the second quarter VW portfolio values.

Step 8: Calculating VW Portfolio Values for 2Q 2013 The calculation of the second quarter VW portfolio value follows closely the method used to calculate the first quarter VW portfolio value. The main difference is that we changed data object names to the next quarter (i.e, changing q1 to q2 and changing q2 to q3).

```
> vw.q2<-subset(vwport[,c(1,5:7)],
+    vwport$date >= as.Date("2013-04-01") &
+    vwport$date <= as.Date("2013-06-30"))
> vw.q2[c(1:3,nrow(vw.q2)),]
          date AMZN.ret YHOO.ret   IBM.ret
62  2013-04-01 0.9816879 0.9987250 0.9956691
63  2013-04-02 1.0065364 1.0119149 1.0093208
64  2013-04-03 0.9837080 0.9831791 0.9920913
125 2013-06-28 1.0005044 0.9866510 0.9767610
>
> names(vw.q2)<-paste(c("date","AMZN","YHOO","IBM"))
> vw.q2[c(1:3,nrow(vw.q2)),]
          date      AMZN      YHOO       IBM
62  2013-04-01 0.9816879 0.9987250 0.9956691
63  2013-04-02 1.0065364 1.0119149 1.0093208
64  2013-04-03 0.9837080 0.9831791 0.9920913
125 2013-06-28 1.0005044 0.9866510 0.9767610
>
> vw.q2$AMZN<-cumprod(vw.q2$AMZN)
> vw.q2$YHOO<-cumprod(vw.q2$YHOO)
> vw.q2$IBM<-cumprod(vw.q2$IBM)
> vw.q2[c(1:3,nrow(vw.q2)),]
          date      AMZN      YHOO       IBM
62  2013-04-01 0.9816879 0.9987250 0.9956691
63  2013-04-02 0.9881046 1.0106247 1.0049496
64  2013-04-03 0.9720065 0.9936252 0.9970017
125 2013-06-28 1.0420278 1.0679983 0.9001523
>
> vw.q2$AMZN.idx<-(q2.vw.inv*q2.vw.wgt$AMZN.wgt)*vw.q2$AMZN
> vw.q2$YHOO.idx<-(q2.vw.inv*q2.vw.wgt$YHOO.wgt)*vw.q2$YHOO
```

```
> vw.q2$IBM.idx<-(q2.vw.inv*q2.vw.wgt$IBM.wgt)*vw.q2$IBM
> vw.q2[c(1:3,nrow(vw.q2)),]
          date      AMZN      YHOO      IBM AMZN.idx   YHOO.idx
62  2013-04-01 0.9816879 0.9987250 0.9956691 0.3428917 0.07343366
63  2013-04-02 0.9881046 1.0106247 1.0049496 0.3451330 0.07430862
64  2013-04-03 0.9720065 0.9936252 0.9970017 0.3395101 0.07305868
125 2013-06-28 1.0420278 1.0679983 0.9001523 0.3639678 0.07852715
       IBM.idx
62  0.6783412
63  0.6846639
64  0.6792491
125 0.6132663
>
> q2.vw.val<-data.frame(rowSums(vw.q2[,5:7]))
> q2.vw.val[c(1:3,nrow(q2.vw.val)),]
[1] 1.094667 1.104106 1.091818 1.055761
> names(q2.vw.val)<-paste("port.val")
> q2.vw.val$date<-vw.q2$date
> q2.vw.val[c(1:3,nrow(q2.vw.val)),]
    port.val       date
62  1.094667 2013-04-01
63  1.104106 2013-04-02
64  1.091818 2013-04-03
125 1.055761 2013-06-28
>
> q3.vw.inv<-q2.vw.val[nrow(q2.vw.val),1]
> q3.vw.inv
[1] 1.055761
```

The results show that the VW portfolio's value decreased from $ 1.104 at the end
of the first quarter to $ 1.056 by the end of the second quarter. This means that at
the end of the first quarter, the VW portfolio returned 10.4 %, but by the end of the
second quarter the shares fell and the gain was reduced to 5.6 %.

Step 9: Calculating VW Portfolio Values for 3Q 2013 We then follow a similar
method to calculate the third quarter VW portfolio values.

```
> vw.q3<-subset(vwport[,c(1,5:7)],
+     vwport$date >= as.Date("2013-07-01") &
+     vwport$date <= as.Date("2013-09-30"))
> vw.q3[c(1:3,nrow(vw.q3)),]
          date AMZN.ret  YHOO.ret    IBM.ret
126 2013-07-01 1.0158810 1.0043772 1.0008988
127 2013-07-02 1.0057781 0.9900951 1.0011621
128 2013-07-03 1.0010573 1.0240096 1.0091278
189 2013-09-30 0.9893358 0.9886736 0.9906949
>
```

```
> names(vw.q3)<-paste(c("date","AMZN","YHOO","IBM"))
> vw.q3[c(1:3,nrow(vw.q3)),]
          date      AMZN      YHOO       IBM
126 2013-07-01 1.0158810 1.0043772 1.0008988
127 2013-07-02 1.0057781 0.9900951 1.0011621
128 2013-07-03 1.0010573 1.0240096 1.0091278
189 2013-09-30 0.9893358 0.9886736 0.9906949
>
> vw.q3$AMZN<-cumprod(vw.q3$AMZN)
> vw.q3$YHOO<-cumprod(vw.q3$YHOO)
> vw.q3$IBM<-cumprod(vw.q3$IBM)
> vw.q3[c(1:3,nrow(vw.q3)),]
          date     AMZN     YHOO       IBM
126 2013-07-01 1.015881 1.004377 1.0008988
127 2013-07-02 1.021751 0.994429 1.0020620
128 2013-07-03 1.022831 1.018305 1.0112086
189 2013-09-30 1.125860 1.319936 0.9738289
>
> vw.q3$AMZN.idx<-(q3.vw.inv*q3.vw.wgt$AMZN.wgt)*vw.q3$AMZN
> vw.q3$YHOO.idx<-(q3.vw.inv*q3.vw.wgt$YHOO.wgt)*vw.q3$YHOO
> vw.q3$IBM.idx<-(q3.vw.inv*q3.vw.wgt$IBM.wgt)*vw.q3$IBM
> vw.q3[c(1:3,nrow(vw.q3)),]
          date     AMZN     YHOO       IBM  AMZN.idx   YHOO.idx    IBM.idx
126 2013-07-01 1.015881 1.004377 1.0008988 0.3749380 0.07818041 0.6093921
127 2013-07-02 1.021751 0.994429 1.0020620 0.3771045 0.07740604 0.6101003
128 2013-07-03 1.022831 1.018305 1.0112086 0.3775032 0.07926452 0.6156692
189 2013-09-30 1.125860 1.319936 0.9738289 0.4155286 0.10274343 0.5929107
>
> q3.vw.val<-data.frame(rowSums(vw.q3[,5:7]))
> q3.vw.val[c(1:3,nrow(q3.vw.val)),]
[1] 1.062511 1.064611 1.072437 1.111183
> names(q3.vw.val)<-paste("port.val")
> q3.vw.val$date<-vw.q3$date
> q3.vw.val[c(1:3,nrow(q3.vw.val)),]
     port.val       date
126 1.062511 2013-07-01
127 1.064611 2013-07-02
128 1.072437 2013-07-03
189 1.111183 2013-09-30
>
> q4.vw.inv<-q3.vw.val[nrow(q3.vw.val),1]
> q4.vw.inv
[1] 1.111183
```

We can see that by the end of the third quarter, the VW portfolio has recovered past its first quarter value of \$ 1.104 to end the third quarter at \$ 1.111. This means that as of the end of the third quarter, the VW portfolio has return 11.1 %.

Step 10: Calculating VW Portfolio Values for 4Q 2013 We also use a similar method to calculate the fourth quarter VW portfolio value.

```
> vw.q4<-subset(vwport[,c(1,5:7)],
+     vwport$date >= as.Date("2013-10-01") &
+     vwport$date <= as.Date("2013-12-31"))
> vw.q4[c(1:3,nrow(vw.q4)),]
          date AMZN.ret YHOO.ret  IBM.ret
190 2013-10-01 1.0265801 1.0343684 1.0064607
191 2013-10-02 0.9986291 0.9950452 0.9923940
192 2013-10-03 0.9820598 0.9923843 0.9940751
253 2013-12-31 1.0137784 1.0059701 1.0062228
>
> names(vw.q4)<-paste(c("date","AMZN","YHOO","IBM"))
> vw.q4[c(1:3,nrow(vw.q4)),]
          date      AMZN      YHOO       IBM
190 2013-10-01 1.0265801 1.0343684 1.0064607
191 2013-10-02 0.9986291 0.9950452 0.9923940
192 2013-10-03 0.9820598 0.9923843 0.9940751
253 2013-12-31 1.0137784 1.0059701 1.0062228
>
> vw.q4$AMZN<-cumprod(vw.q4$AMZN)
> vw.q4$YHOO<-cumprod(vw.q4$YHOO)
> vw.q4$IBM<-cumprod(vw.q4$IBM)
> vw.q4[c(1:3,nrow(vw.q4)),]
          date     AMZN     YHOO       IBM
190 2013-10-01 1.026580 1.034368 1.0064607
191 2013-10-02 1.025173 1.029243 0.9988056
192 2013-10-03 1.006781 1.021405 0.9928878
253 2013-12-31 1.275557 1.219174 1.0183506
>
> vw.q4$AMZN.idx<-(q4.vw.inv*q4.vw.wgt$AMZN.wgt)*vw.q4$AMZN
> vw.q4$YHOO.idx<-(q4.vw.inv*q4.vw.wgt$YHOO.wgt)*vw.q4$YHOO
> vw.q4$IBM.idx<-(q4.vw.inv*q4.vw.wgt$IBM.wgt)*vw.q4$IBM
> vw.q4[c(1:3,nrow(vw.q4)),]
          date     AMZN     YHOO       IBM AMZN.idx  YHOO.idx   IBM.idx
190 2013-10-01 1.026580 1.034368 1.0064607 0.4322600 0.1022110 0.5951201
191 2013-10-02 1.025173 1.029243 0.9988056 0.4316674 0.1017046 0.5905936
192 2013-10-03 1.006781 1.021405 0.9928878 0.4239232 0.1009300 0.5870945
253 2013-12-31 1.275557 1.219174 1.0183506 0.5370960 0.1204725 0.6021506
>
> q4.vw.val<-data.frame(rowSums(vw.q4[,5:7]))
> q4.vw.val[c(1:3,nrow(q4.vw.val)),]
[1] 1.129591 1.123966 1.111948 1.259719
> names(q4.vw.val)<-paste("port.val")
> q4.vw.val$date<-vw.q4$date
> q4.vw.val[c(1:3,nrow(q4.vw.val)),]
    port.val       date
190 1.129591 2013-10-01
191 1.123966 2013-10-02
192 1.111948 2013-10-03
253 1.259719 2013-12-31
```

The results show that the VW portfolio cumulative return increased during this period from 11.1 % at the end of the third quarter to close the year at 26.0 % return.

Step 11: Combining Quarterly VW Portfolio Values into One Data Object we can then combine the VW portfolio values into one series. The results show that an investment made in the VW portfolio at closing prices on December 31, 2012 would have yielded a 26.0 % return.

```
> vw.portval<-rbind(q1.vw.val,q2.vw.val,q3.vw.val,q4.vw.val)
> vw.portval[c(1:3,nrow(vw.portval)),]
    port.val       date
1   1.000000 2012-12-31
2   1.024248 2013-01-02
3   1.021361 2013-01-03
253 1.259719 2013-12-31
```

3.3.3 Normalized EW and VW Portfolio Price Chart

We now compare the performance of an investment in the EW portfolio and an investment in the VW portfolio over the 1-year period from December 31, 2012 to December 31, 2013.

Step 1: Combine the Data We use the `merge` command to combine `vw.portval` and `ew.portval`.

```
> port.val<-merge(vw.portval,ew.portval,by="date")
> port.val[c(1:3,nrow(port.val)),]
          date port.val.x port.val.y
1   2012-12-31   1.000000   1.000000
2   2013-01-02   1.024248   1.019927
3   2013-01-03   1.021361   1.014577
253 2013-12-31   1.259719   1.501384
```

Step 2: Rename the Variables The current names of the variables for the portfolio values do not make too much sense. We will rename them `VW.cum` and `EW.cum` to denote that they are cumulative values for the VW portfolio and VW portfolio, respectively.

```
> names(port.val)<-paste(c("date","VW.cum","EW.cum"))
> port.val[c(1:3,nrow(port.val)),]
          date   VW.cum   EW.cum
1   2012-12-31 1.000000 1.000000
2   2013-01-02 1.024248 1.019927
3   2013-01-03 1.021361 1.014577
253 2013-12-31 1.259719 1.501384
```

Step 3: Plot the Data Since the data in `port.val` is already indexed to $ 1, all
we have to do now is chart the data using the `plot` command. Note that we have a
long title, which we split into three lines by using a hard **ENTER** at the end of each
line. This allows us to control when to break the title into separate lines.

```
> par(mfrow=c(1,1))
> y.range<-range(port.val[,2:3])
> y.range
[1] 1.000000 1.506593
> plot(port.val$EW.cum,
+    type="l",
+    xlab="Date",
+    ylab="Value of Investment",
+    ylim=y.range,
+    lty=1,
+    main="Value of $1 Investment in Equal-Weighted and
+        Value-Weighted Portfolios of AMZN, YHOO, and IBM
+        December 31, 2012 - December 31, 2013")
> lines(port.val$VW.cum,lty=2)
> abline(h=1,lty=1)
> legend("topleft",
+    c("Equal-Weighted Portfolio","Value-Weighted Portfolio"),
+    lty=c(1,2))
```

Note also that we adjusted the y-axis range using the `y.range` variable that is
constructed to hold the largest and smallest values in the data. This way we can be
sure that we can visualize the complete series of EW and VW portfolio values.

Figure 3.2 shows the performance of an investment in an EW and VW portfolio
of AMZN, YHOO, and IBM from December 31, 2012 to December 31, 2013.
As we can see from the chart, and confirm with the data in `port.val` above, the
EW portfolio returned over 50 % over this period while the VW portfolio returned
only 26 %. The reason is that, being the stock with the smallest market capitalization
of the three stocks in the portfolio, YHOO's substantial returns over the period were
muted in the VW portfolio. Therefore, as this example shows, returns of small stocks
are better captured by an EW portfolio.

3.3.4 Saving Benchmark Portfolio Returns into a CSV File

In this section, we show how to then save to a CSV file the EW and VW portfolio
returns into a CSV file that can be used later. Because the way the portfolio is
constructed, we can calculate the portfolio returns by basing them off the changes in
the daily portfolio values.

**Value of $1 Investment in Equal-Weighted and
Value-Weighted Portfolios of AMZN, YHOO, and IBM
December 31, 2012 - December 31, 2013**
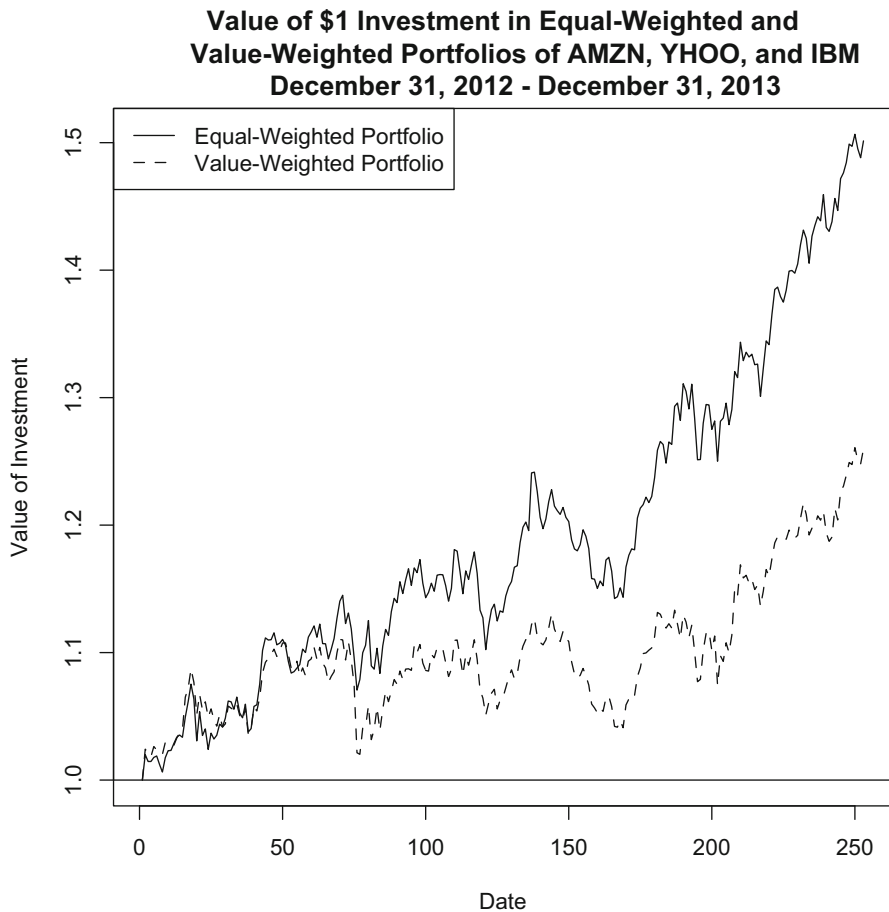
Fig. 3.2 Normalized plot of index of equal-weighted returns and value-weighted returns, January 2013–June 2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

Step 1: Convert the Data into an `xts` Object So that we can appropriately take the lag of the portfolio value, we need to convert the data into an `xts` object.

```
> port.xts<-xts(port.val[,2:3],order.by=port.val[,1])
> port.xts[c(1:3,nrow(port.xts)),]
             VW.cum   EW.cum
2012-12-31 1.000000 1.000000
2013-01-02 1.024248 1.019927
2013-01-03 1.021361 1.014577
2013-12-31 1.259719 1.501384
>
> port.xts$Lag.VW<-Lag(port.xts$VW.cum,k=1)
> port.xts$Lag.EW<-Lag(port.xts$EW.cum,k=1)
> port.xts[c(1:3,nrow(port.xts)),]
             VW.cum   EW.cum   Lag.VW   Lag.EW
2012-12-31 1.000000 1.000000       NA       NA
2013-01-02 1.024248 1.019927 1.000000 1.000000
2013-01-03 1.021361 1.014577 1.024248 1.019927
2013-12-31 1.259719 1.501384 1.247980 1.488191
```

Step 2: Create EW and VW Returns The lag values allows us to easily calculate the returns on each day.

```
> port.xts$VW.ret<-port.xts$VW.cum/port.xts$Lag.VW-1
> port.xts$EW.ret<-port.xts$EW.cum/port.xts$Lag.EW-1
> port.xts[c(1:3,nrow(port.xts)),]
             VW.cum   EW.cum   Lag.VW   Lag.EW        VW.ret        EW.ret
2012-12-31 1.000000 1.000000       NA       NA            NA            NA
2013-01-02 1.024248 1.019927 1.000000 1.000000  0.024247508  0.019927472
2013-01-03 1.021361 1.014577 1.024248 1.019927 -0.002818685 -0.005246421
2013-12-31 1.259719 1.501384 1.247980 1.488191  0.009406090  0.008864921
```

Step 3: Clean up the Data From `port.xts`, we only keep the portfolio values (`VW.cum` and `EW.cum`) and the portfolio returns (`VW.ret` and `EW.ret`).

```
> Port.Ret<-port.xts[,c(1,2,5,6)]
> Port.Ret[c(1:3,nrow(Port.Ret)),]
             VW.cum   EW.cum        VW.ret        EW.ret
2012-12-31 1.000000 1.000000            NA            NA
2013-01-02 1.024248 1.019927  0.024247508  0.019927472
2013-01-03 1.021361 1.014577 -0.002818685 -0.005246421
2013-12-31 1.259719 1.501384  0.009406090  0.008864921
```

Step 4: Create a Date Variable in the Data Object We now turn to the steps to save this data into a CSV file. If we directly save `Port.Ret` into a CSV file, we will not have a date variable. So, we first create a `DATE` object that holds the dates in `Port.Ret`. Then, we combine `DATE` with `Port.Ret`

```
> csv.port<-cbind(data.frame(index(Port.Ret)),data.frame(Port.Ret))
> names(csv.port)[1]<-paste("date")
> csv.port[c(1:3,nrow(csv.port)),]
                date VW.cum   EW.cum        VW.ret        EW.ret
2012-12-31 2012-12-31 1.000000 1.000000           NA            NA
2013-01-02 2013-01-02 1.024248 1.019927  0.024247508  0.019927472
2013-01-03 2013-01-03 1.021361 1.014577 -0.002818685 -0.005246421
2013-12-31 2013-12-31 1.259719 1.501384  0.009406090  0.008864921
```

Step 5: Replace Index with Observation Numbers If we export the data object as-is, we will end up with two columns of dates. The better thing to do is to rename the index to tell us the observation number of each day.

```
> rownames(csv.port)<-seq(1:nrow(csv.port))
> csv.port[c(1:3,nrow(csv.port)),]
          date VW.cum   EW.cum        VW.ret        EW.ret
1   2012-12-31 1.000000 1.000000           NA            NA
2   2013-01-02 1.024248 1.019927  0.024247508  0.019927472
3   2013-01-03 1.021361 1.014577 -0.002818685 -0.005246421
253 2013-12-31 1.259719 1.501384  0.009406090  0.008864921
```

Step 6: Save the Data to a CSV File To save `csv.port` into a CSV file, we use the `write.csv` command. For our purpose, we save the data with the filename **Hypothetical Portfolio (Daily).csv**.

```
> write.csv(csv.port,"Hypothetical Portfolio (Daily).csv")
```

3.4   Further Reading

Calculating returns is at the heart of most investment applications. We have only touched upon the basic return concepts in the previous chapter and in this chapter. There are more advanced return concepts that are used in practice, such as Time-Weighted Returns and Money-Weighted Returns. A good discussion of these and other return calculations can be found in Maginn et al. [1].

Reference

1. Maginn, J., Tuttle, D., Pinto, J., & McLeavey, D. (2007). *Managing investment portfolios: A dynamic process* (3rd ed.). New Jersey: Wiley.

Chapter 4
Risk

Investments inherently contain some level of risk. For most individual investors, we are unlikely to be able to take advantage of any arbitrage opportunities because large, institutional investors have the ability to profit from these opportunities and, by doing so, eliminate such opportunities within a short period of time. The closest instrument to a *truly* risk-free security is likely to be a short maturity US Treasury, as the likelihood of the US government defaulting on very short term obligations is likely extremely low and other risks, such as inflation risk and liquidity risk, are also likely negligible for very short maturities.

As a concept, risk is fairly easy to grasp. We know that larger swings in price are riskier than smaller swings in price. We know that more frequent price changes are riskier than less frequent price changes. We know that a bank account is less risky than investing in stocks. However, a perfect measure that quantifies risk still eludes us.

Investors have always yearned for the quantification of risk. Markowitz gave us the most common measure of risk we use today, which is the *variance* of a security's price. The variance or, its positive square root, *standard deviation* is a measure of how far a security's return deviates from its average during the period. Variance does not care whether the deviation from the average is a positive deviation or a negative deviation—both are treated as risk.

Because of the actual or perceived inefficiencies of variance as a measure of risk, many other measures of risk were developed and are frequently used in practice. Some focus on measuring loss or downside risk, such as Value-at-Risk (VaR) and Exected Shortfall (also known as conditional VaR or tail VaR). Other measures are modificiations of the close-to-close calculation that is typically applied when calculating variance. Examples of these are Parkinson, Garman-Klass, Rogers-Satchell-Yoon, and Yang and Zhang. These measures have been shown to be several times more efficient than close-to-close volatility (i.e., close-to-close volatility divided by alternative risk measure).

The rest of the chapter is broken up into the following parts. First, we begin with a discussion of the risk-return trade-off. At its core, this means that we should expect higher returns only if we are willing to take on more risk. Next, we discuss individual security risk, as measured typically by the variance (or standard deviation) of returns. Third, we discuss portfolio risk and emphasize that in the context of a portfolio we

are less concerned with individual security variance but are more concerned with the covariance of the securities in the portfolio (i.e., in a portfolio context, the way the assets in the portfolio move together is more important). Then, we discuss two measures of portfolio loss: Value-at-Risk (VaR) and Expected Shortfall (ES). Finally, we end with a discussion of several other measures of risk that are used in practice.

4.1 Risk-Return Trade-Off

The main trade-off we have to consider when making investments is between risk and return. The only way you can get a higher expected return from a security is if you are willing to take on more risk. To see why, let us first take a look at how stocks and bonds have performed over the last 50 years. To do this, we use Professor Kenneth French's data, which is available from the following URL: http://mba.tuck.dartmouth.edu/pages/ faculty/ken.french/data_library.html. For our current purpose, we will use his monthly data, which can be accessed by clicking on the **Fama/French Factors** link.

Step 1: Import Fama-French Data from Professor Kenneth French's Website
We retrieve Fama-French monthly data and save the text file (compressed initially in a ZIP file) as **F-F_Research_Data_Factors.txt**. Below is what the first ten lines of the file looks like. Note that the index from 1 to 10 on the left side of the output is added for ease of reference and is not part of the Fame-French raw data.

```
1   This file was created by CMPT_ME_BEME_RETS using the 201311 CRSP database.
2   The 1-month TBill return is from Ibbotson and Associates, Inc.
3
4            Mkt-RF      SMB      HML      RF
5    192607    2.65    -2.52    -2.67    0.22
6    192608    2.58    -1.20     4.52    0.25
7    192609    0.38    -1.34    -0.12    0.23
8    192610   -3.44    -0.15     0.59    0.32
9    192611    2.42    -0.24    -0.34    0.31
10   192612    2.75    -0.23    -0.04    0.28
```

As we can see, the data starts on the fourth line. We can also see that the data is of fixed-width. This means that each column of data can be separated by a fixed number of characters per column. In particular, the first column of dates has six characters to the last character, the second column labeled "Mkt-RF" is eight characters after that, the third column "SMB" is eight characters after, the fourth column "HML" is eight characters after, and the last column "RF" is eight characters after. To import this fixed-width file, we use the read.fwf command with widths of 6, 8, 8, 8, and 8 characters while skipping the first four lines of the data.

```
> FF.raw<-read.fwf(file="F-F_Research_Data_Factors.txt",
+        widths=c(6,8,8,8,8),skip=4)
> head(FF.raw)
       V1        V2        V3        V4        V5
1 192607     2.95     -2.50     -2.67      0.22
2 192608     2.63     -1.20      4.50      0.25
3 192609     0.38     -1.33     -0.30      0.23
4 192610    -3.24     -0.13      0.79      0.32
5 192611     2.54     -0.24     -0.39      0.31
6 192612     2.62     -0.22     -0.11      0.28
> tail(FF.raw)
           V1        V2        V3        V4        V5
1140     2010     17.39     13.58     -3.21      0.12
1141     2011      0.47     -6.00     -6.91      0.04
1142     2012     16.29      0.44      8.09      0.06
1143     2013     35.21      7.89      0.29      0.02
1144    <NA>      <NA>      <NA>      <NA>      <NA>
1145 Copyri ght 2014  Kenneth  R. Fren        ch
```

Step 2: Clean up Data   The bottom part of the file contains summary annual data, which we do not need. As such, we delete the data from Row 1051 to Row 1145. Note that since the data on Professor French's website gets updated periodically, a later version of the file will likely end up with different row numbers. So we have to manually look at the data to make sure we are deleting the correct rows. You can view the data in Excel, if it is easier, or in R. We also rename the variable names from V1 to V5 to something more meaningful.

```
> FF.raw<-FF.raw[-1051:-1145,]
> names(FF.raw)<-paste(c("text.date","RmxRf","SMB","HML","Rf"))
> head(FF.raw)
   text.date     RmxRf       SMB       HML        Rf
1     192607      2.95     -2.50     -2.67      0.22
2     192608      2.63     -1.20      4.50      0.25
3     192609      0.38     -1.33     -0.30      0.23
4     192610     -3.24     -0.13      0.79      0.32
5     192611      2.54     -0.24     -0.39      0.31
6     192612      2.62     -0.22     -0.11      0.28
> tail(FF.raw)
     text.date     RmxRf       SMB       HML        Rf
1045    201307      5.66      1.85      0.79      0.00
1046    201308     -2.69      0.28     -2.46      0.00
1047    201309      3.76      2.85     -1.52      0.00
1048    201310      4.17     -1.53      1.39      0.00
1049    201311      3.12      1.31     -0.38      0.00
1050    201312      2.81     -0.44     -0.17      0.00
```

Although the data looks like it is workable, the variables are all `Factor` variables. As such, we need to convert the data to `numeric` variables. For the date variable, it is easier for us to create a sequence of monthly dates rather than converting the `text.date` variable, so we take that simpler approach.

```
> str(FF.raw)
'data.frame': 1050 obs. of  5 variables:
 $ text.date: Factor w/ 1143 levels "      "," 1927",..: 90 91 92 93 94 95 96 97 98 99 ...
 $ RmxRf    : Factor w/ 872 levels "   0.00","   0.01",..: 191 168 24 601 161 167 440 265 442 33 ...
 $ SMB      : Factor w/ 720 levels "   SMB","   0.00",..: 544 449 461 358 368 366 362 35 482 24 ...
 $ HML      : Factor w/ 731 levels "   HML","   0.00",..: 550 277 385 68 393 367 292 226 547 57 ...
 $ Rf       : Factor w/ 172 levels "   RF","   0.00",..: 24 27 25 34 33 30 27 28 32 27 ...
> FF.raw<-FF.raw[,c(-1,-3,-4)]
> FF.raw$RmxRf<-as.numeric(as.character(FF.raw$RmxRf))/100
> FF.raw$Rf<-as.numeric(as.character(FF.raw$Rf))/100
> FF.raw$date<-seq(as.Date("1926-07-01"),as.Date("2013-12-31"),by="months")
> FF.raw$date<-as.yearmon(FF.raw$date,"%Y-%m-%d")
> FF.raw[c(1:3,nrow(FF.raw)),]
      RmxRf     Rf    date
1    0.0295 0.0022 Jul 1926
2    0.0263 0.0025 Aug 1926
3    0.0038 0.0023 Sep 1926
1050 0.0281 0.0000 Dec 2013
> str(FF.raw)
'data.frame': 1050 obs. of  3 variables:
 $ RmxRf: num  0.0295 0.0263 0.0038 -0.0324 0.0254 0.0262 -0.0011 0.0411 -0.0015 0.0052 ...
 $ Rf   : num  0.0022 0.0025 0.0023 0.0032 0.0031 0.0028 0.0025 0.0026 0.003 0.0025 ...
 $ date :Class 'yearmon'  num [1:1050] 1926 1927 1927 1927 1927 ...
```

Step 3: Calculate Raw Market Return Variable The Fama-French data gives us the excess market return, which is the return on the market less the risk-free rate. In our analysis, what we need is the raw market return, so we have to add the risk-free rate back.

```
> FF.raw$Rm<-FF.raw$RmxRf+FF.raw$Rf
> FF.raw[c(1:3,nrow(FF.raw)),]
      RmxRf     Rf    date      Rm
1    0.0295 0.0022 Jul 1926 0.0317
2    0.0263 0.0025 Aug 1926 0.0288
3    0.0038 0.0023 Sep 1926 0.0061
1050 0.0281 0.0000 Dec 2013 0.0281
```

Step 4: Subset Data from December 1963 to December 2013 Note that the returns are at the end of each month, so to get 50 years of returns we would assume an investment made at the end of December 1963, which is effectively the start of January 1964. As a placeholder for a starting value prior to January 1964, we subset the data to include December 1963.

```
> FF<-subset(FF.raw,
+     FF.raw$date>="1963-12-01" &
+     FF.raw$date<="2013-12-31")
> FF[c(1:3,nrow(FF)),]
      RmxRf     Rf    date      Rm
450  0.0183 0.0029 Dec 1963 0.0212
451  0.0224 0.0030 Jan 1964 0.0254
452  0.0154 0.0026 Feb 1964 0.0180
1050 0.0281 0.0000 Dec 2013 0.0281
```

Step 5: Calculate Gross Returns for the Market and Risk-free Rate We then calculate the gross returns by adding one to the market return and the risk-free rate. We also set the return for December 1963 to zero because we are assuming we are at

the end of the month, which means the gross market return and gross risk-free rate
of return for December 1963 equals one.

```
> FF$Gross.Rm<-1+FF$Rm
> FF$Gross.Rm[1]<-1
> FF$Gross.Rf<-1+FF$Rf
> FF$Gross.Rf[1]<-1
> FF[c(1:3,nrow(FF)),]
       RmxRf     Rf    date     Rm Gross.Rm Gross.Rf
450   0.0183 0.0029 Dec 1963 0.0212   1.0000   1.0000
451   0.0224 0.0030 Jan 1964 0.0254   1.0254   1.0030
452   0.0154 0.0026 Feb 1964 0.0180   1.0180   1.0026
1050  0.0281 0.0000 Dec 2013 0.0281   1.0281   1.0000
```

Step 6: Calculate Cumulative Returns for the Market and Risk-free Rate Us-
ing the cumprod command, we can cumulate the market return and risk-free rate
through the end of 2013. We can interpret the results to show that $ 1 investment in
stocks on the last day of 1963 would have resulted in $10\times$ more value at the end of
50 years than a $ 1 investment in bonds.

```
> FF$cum.Rm<-cumprod(FF$Gross.Rm)
> FF$cum.Rf<-cumprod(FF$Gross.Rf)
> FF[c(1:3,nrow(FF)),]
       RmxRf     Rf    date     Rm Gross.Rm Gross.Rf      cum.Rm     cum.Rf
450   0.0183 0.0029 Dec 1963 0.0212   1.0000   1.0000    1.000000   1.000000
451   0.0224 0.0030 Jan 1964 0.0254   1.0254   1.0030    1.025400   1.003000
452   0.0154 0.0026 Feb 1964 0.0180   1.0180   1.0026    1.043857   1.005608
1050  0.0281 0.0000 Dec 2013 0.0281   1.0281   1.0000  124.890232  12.099660
```

Step 7: Plot the Data We then use the plot command to chart the results.

```
> y.range<-range(FF$cum.Rm,FF$cum.Rf)
> y.range
[1]    1.0000 124.8902
> title1<-"Stock vs. Bond Returns"
> title2<-"1964 to 2013"
> plot(x=FF$date,
+    FF$cum.Rm,
+    type="l",
+    xlab="Date",
+    ylab="Value of $1 Investment ($)",
+    ylim=y.range,
+    main=paste(title1,"\n",title2))
> lines(x=FF$date,y=FF$cum.Rf,lty=2)
> legend("topleft",
+    c("Stocks (2013 Ending Value: $124.89)",
+    "Bonds (2013 Ending Value: $12.10)"),
+    lty=c(1,2))
```

Figure 4.1 clearly shows the larger returns of investing in stocks compared to investing
in bonds (technically, the data shows T-bills but we will be a little loose with the
terminology at this point). This $10\times$ difference in value at the end of 50 years is even

Stock vs. Bond Returns
1964 to 2013
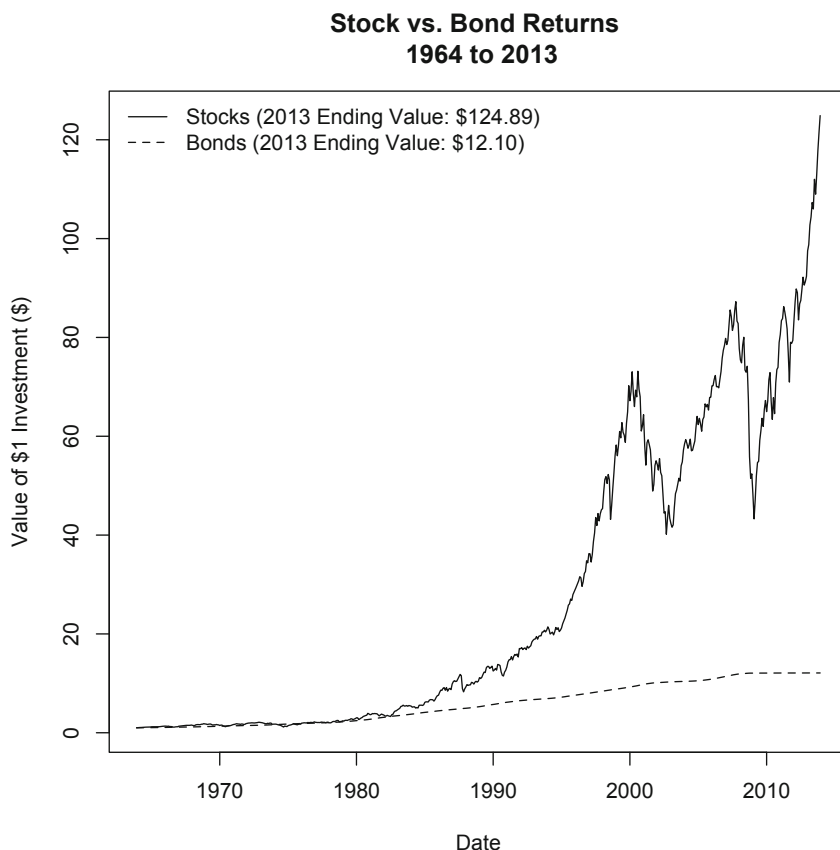
Fig. 4.1 Difference in returns from investing in stocks versus investing in bonds, December 1963 to December 2013. Data Source: http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

more impressive when we consider this difference includes the effects of the 1987 crash, the recession in the early 2000s, and the 2008/2009 crisis.

Given how much stocks outperformed bonds over the last 50 years, why then would investors bother putting money in bonds? The reason for this is that an investment in stocks is riskier than an investment in bonds.

Step 8: Plot Stock and Bond Returns To make this example more concrete, let us demonstrate this by comparing the returns of the S&P 500 Index and the returns of a 30-day Treasury Bill.

```
> y.range<-range(FF$Rm,FF$Rf)
> y.range
[1] -0.2264  0.1661
> title1<-"Volatility of Stock vs. Bond Returns"
> title2<-"1964 to 2013"
> plot(x=FF$date,
+      FF$Rm,
+      type="l",
+      xlab="Date",
+      ylab="Returns (%)",
+      ylim=y.range,
+      col="gray50",
+      main=paste(title1,"\n",title2))
> lines(x=FF$date,y=FF$Rf)
> abline(h=0)
> legend("topleft",
+      c("Stocks","Bonds"),
+      lty=c(1,2))
```

Figure 4.2 shows the volatility of stock and bond returns. The chart demonstrates how much more volatile stock returns are compared to bond returns. Volatility in this case is reflected in the large swings in the returns compared to the relatively flat, close to zero bond return line.

4.2   Individual Security Risk

Risk or *volatility* is really something that is difficult to measure accurately. We can qualitatively say that a stocks return is volatile, but it is sometimes better to be able to quantify such a statement. It is common for investors to use *variance* or, its positive square root, *standard deviation* as the measure of risk.

The variance of an asset's return, σ^2, is

$$\sigma^2 = \frac{1}{T-1} \sum_{t=1}^{T} (R_t - \bar{R})^2, \tag{4.1}$$

where R_t is the return of the asset on day t for $t = 1, \ldots, T$ and $\bar{R}$ is the average return of the asset over days 1 to T. The standard deviation is equal to the square root of the variance and is denoted by σ.[1]

What Eq. (4.1) tells us about variance as a measure of volatility is that variance captures deviations from the average. This deviation is squared because, from

---

[1] The calculation above technically calculates the *sample variance* and *sample standard deviation*. For ease of exposition, we drop the term *sample* when describing variance and standard deviation.
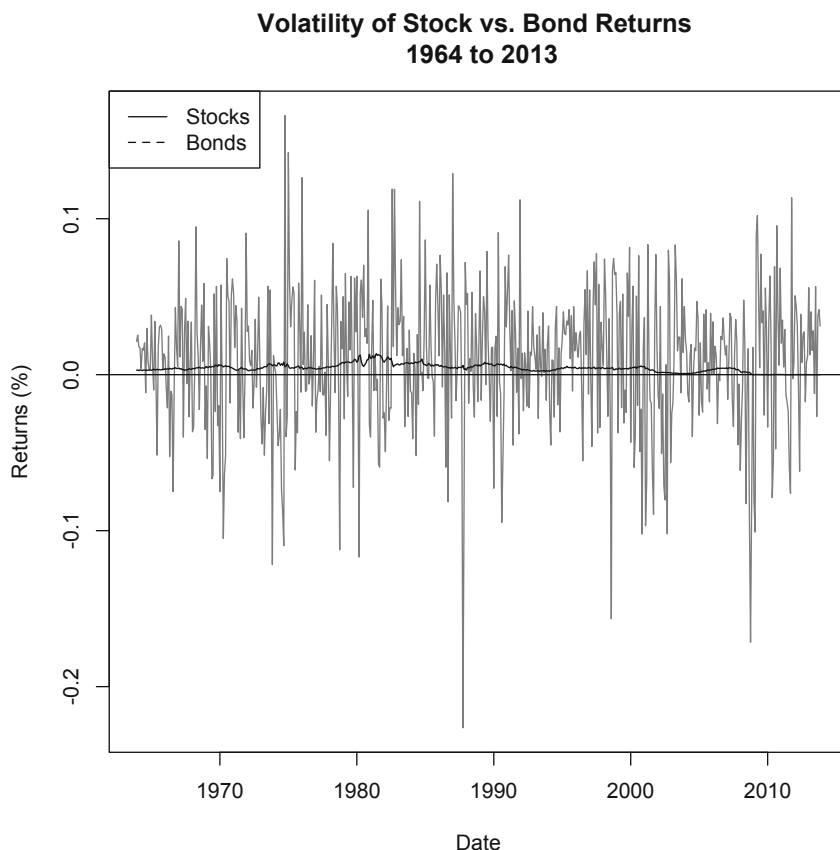
Fig. 4.2 Investing in stocks is riskier than investing in bonds. Data Source: http://mba.tuck.dart mouth.edu/pages/faculty/ken.french/data_library.html

the perspective of variance, both positive or negative deviations from the mean are considered "risk."[2]

We now calculate the variance and standard deviation of AMZN's returns. Note that the variance and standard deviation calculations change depending on the time period we choose.

Step 1: Import AMZN Data from Yahoo Finance We import AMZN data from December 31, 2010 to December 31, 2013 from Yahoo Finance. Note that we saved the file as **AMZN Yahoo.csv**.

---

[2] Had we not squared the deviations from the mean, positive and negative deviations could offset each other.

```
> 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),]
> library(xts)
> 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)),]
           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: Calculate Returns Using the `Delt` command in the `quantmod` package, we calculate the daily total return for AMZN stock.

```
> AMZN.ret<-data.AMZN[,6]
> library(quantmod)
> AMZN.ret$Return=Delt(AMZN.ret$AMZN.Adjusted)
> AMZN.ret<-AMZN.ret[-1,2]
> AMZN.ret[c(1:3,nrow(AMZN.ret)),]
                Return
2011-01-03 0.023444444
2011-01-04 0.004288351
2011-01-05 0.013026323
2013-12-31 0.013778377
```

Step 3: Calculate Full Period (2011–2013) Variance and Standard Deviation We use the `var` command to calculate variance and `sd` command to calculate standard deviation.

```
> AMZN.var.full<-var(AMZN.ret$Return)
> AMZN.var.full
             Return
Return 0.000425095
> AMZN.sd.full<-sd(AMZN.ret$Return)
> AMZN.sd.full
[1] 0.02061783
```

Step 4: Calculate Variance and Standard Deviation for 2011 We subset the data from January 1, 2011 to December 31, 2011 and calculate the variance and standard deviation of AMZN returns. To subset the data, we use the `subset` command. To identify the dates, we use the `index` command.

```
> AMZN.2011<-subset(AMZN.ret,
+     index(AMZN.ret) >= "2011-01-01" &
+     index(AMZN.ret) <= "2011-12-31")
> AMZN.2011[c(1:3,nrow(AMZN.2011)),]
                 Return
2011-01-03  0.023444444
2011-01-04  0.004288351
2011-01-05  0.013026323
2011-12-30 -0.004371333
>
> AMZN.var.2011<-var(AMZN.2011)
> AMZN.var.2011
            Return
Return 0.0005852692
> AMZN.sd.2011<-sd(AMZN.2011)
> AMZN.sd.2011
[1] 0.02419234
```

Step 5: Calculate Variance and Standard Deviation for 2012 and 2013 Using
the same technique in Step 4 to subset the data, we can calculate the variance and
standard deviation of AMZN returns for 2012 and 2013.

```
> AMZN.2012<-subset(AMZN.ret,
+     index(AMZN.ret) >= "2012-01-01" &
+     index(AMZN.ret) <= "2012-12-31")
> AMZN.2012[c(1:3,nrow(AMZN.2012)),]
                  Return
2012-01-03  0.0342576545
2012-01-04 -0.0084901972
2012-01-05  0.0005633485
2012-12-31  0.0232074394
>
> AMZN.var.2012<-var(AMZN.2012)
> AMZN.var.2012
            Return
Return 0.0004015231
> AMZN.sd.2012<-sd(AMZN.2012)
> AMZN.sd.2012
[1] 0.02003804
>
> AMZN.2013<-subset(AMZN.ret,
+     index(AMZN.ret) >= "2013-01-01" &
+     index(AMZN.ret) <= "2013-12-31")
> AMZN.2013[c(1:3,nrow(AMZN.2013)),]
                Return
2013-01-02 0.025670666
2013-01-03 0.004547044
2013-01-04 0.002592077
2013-12-31 0.013778377
>
> AMZN.var.2013<-var(AMZN.2013)
> AMZN.var.2013
            Return
Return 0.0002897266
> AMZN.sd.2013<-sd(AMZN.2013)
> AMZN.sd.2013
[1] 0.01702136
```

Step 6: Calculate Average Return for the Full Period and Each of the Subperiods Standard deviation is a measure of dispersion around the mean. To make the comparison meaningful, we have to calculate the mean for each of the four time periods above.

```
> mean.ret.full<-mean(AMZN.ret)
> mean.ret.full
[1] 0.001266679
>
> mean.ret.2011<-mean(AMZN.2011)
> mean.ret.2011
[1] 0.0001385631
>
> mean.ret.2012<-mean(AMZN.2012)
> mean.ret.2012
[1] 0.001680247
>
> mean.ret.2013<-mean(AMZN.2013)
> mean.ret.2013
[1] 0.00198451
```

Step 7: Combine All Data To combine all the data we previously calculated, we use a combination of the `rbind` and `cbind` commands. The first `cbind` command strings together all four variances into one row, the second `cbind` command strings together all four standard deviations into one row, and the third `cbind` command strings together all four means into one row. Then, we use the `rbind` command to stack the first row (variances) at the top, the second row (standard deviation) at the middle, and the third row (mean) at the bottom.

```
> AMZN.risk<-rbind(
+     cbind(AMZN.var.full,AMZN.var.2011,
+         AMZN.var.2012,AMZN.var.2013),
+     cbind(AMZN.sd.full,AMZN.sd.2011,
+         AMZN.sd.2012,AMZN.sd.2013),
+     cbind(mean.ret.full,mean.ret.2011,
+         mean.ret.2012,mean.ret.2013))
> AMZN.risk
             Return       Return       Return       Return
Return 0.000425095 0.0005852692 0.0004015231 0.0002897266
       0.020617832 0.0241923377 0.0200380423 0.0170213570
       0.001266679 0.0001385631 0.0016802471 0.0019845096
```

Step 8: Cleanup Data We're close to getting a decent table ready, but we still need one final tweak. The way we combined the data generates variable names that are misleading and row names that are unhelpful. As such, we should make modifications to these. We use the `rownames` command to change the index to say `Variance` for the first row, `Std Dev` for the second row, and `Mean` for the third row. We then use the `colnames` command to change the variable names to say "2011–2013," "2011," "2012," and "2013."

```
> options(digits=3)
> rownames(AMZN.risk)<-c("Variance","Std Dev","Mean")
> colnames(AMZN.risk)<-c("2011-2013","2011","2012","2013")
> AMZN.risk
          2011-2013     2011     2012     2013
Variance   0.000425 0.000585 0.000402 0.00029
Std Dev    0.020618 0.024192 0.020038 0.01702
Mean       0.001267 0.000139 0.001680 0.00198
> options(digits=7)
```

Based on daily returns, we can see that for the 2011–2013 period, AMZN's average return is 0.13 % but the standard deviation around this mean is 2.07 %. Assuming a normal distribution, 68 % of AMZN returns lie within one standard deviation from the mean and 95 % of the AMZN returns lie within two standard deviations from the mean. Using 2011 to 2013 data, this means that 68 % of AMZN's daily returns are between −1.9 and 2.2 % and 95 % of AMZN's daily returns are between −4.0 and 4.3 %.

Note that these are daily volatility estimates. As such, the volatility would be different for weekly and monthly returns. Standard deviations of returns based on different frequencies are not comparable. We have to convert such volatility measures to an annualized volatility number to make such a comparison. For variance, we multiply the daily variance by 252, which represents the approximate number of trading days in a year. For standard deviation, we multiply the daily standard deviation by the square root of 252. For the mean, we multiply the daily mean by 252.

```
> options(digits=3)
> annual.vol<-AMZN.risk
> annual.vol[1,]<-annual.vol[1,]*252
> annual.vol[2,]<-annual.vol[2,]*sqrt(252)
> annual.vol[3,]<-annual.vol[3,]*252
> annual.vol
          2011-2013   2011  2012  2013
Variance      0.107 0.1475 0.101 0.073
Std Dev       0.327 0.3840 0.318 0.270
Mean          0.319 0.0349 0.423 0.500
> options(digits=7)
```

As the output above shows, AMZN's annualized standard deviation for the full period was 32.7 %. Breaking down the data into three different years, we see that 2011 had the highest standard deviation at 38.4 % while 2013 had the lowest standard deviation at 27.0 %.

4.3   Portfolio Risk

The previous section discusses how to calculate risk for an individual security. However, most investments are done in the context of a portfolio of securities. In Markowitz [6], we learned that it is the covariance of the assets in the portfolio that

is important when assessing risk in the context of a portfolio. In particular, when we add assets that are not perfectly correlated with the securities in our portfolio, the overall risk of the portfolio decreases. In fact, Reilly and Brown [7] state that using 12–18 well-selected stocks can yield 90 % of the maximum benefits of diversification. Therefore, as a general rule, a portfolio's risk is not simply the weighted average of the standard deviations of each of the securities in the portfolio but likely something lower.

4.3.1   Two Assets (Manual Approach)

In the case of a two-asset portfolio, portfolio risk is calculated as

$$\sigma_p^2 = w_1^2 r_1^2 + w_2^2 r_2^2 + 2\sigma_{1,2} w_1 w_2 \tag{4.2}$$

where w_i is the weight of security i in the portfolio, r_i is the return of security i, and $\sigma_{1,2}$ is the covariance between the returns of securities 1 and 2. Note that $\sigma_{1,2} = \rho_{1,2}\sigma_1\sigma_2$, so in some instances Eq. (4.2) may be shown using the correlation coefficient term ($\rho_{1,2}$) instead but we have to add the product of the standard deviation of the two assets ($\sigma_1\sigma_2$).

Let us work through an example of calculating the portfolio risk for a $ 10,000 portfolio that invested $ 2500 in AMZN and $ 7500 in IBM.

Step 1: Calculate Weights of Securities in the Portfolio This means that the weights for our securities would be 25 % AMZN and 75 % IBM.

```
> wgt.AMZN=.25
> wgt.IBM=.75
```

Step 2: Import AMZN and IBM Data from Yahoo Finance and Calculate Total Returns Using the `read.csv` command, we import the data in **AMZN Yahoo.csv** and **IBM Yahoo.csv**. After which we calculate the returns for AMZN and IBM applying the `Delt` command to the adjusted closing price of each security.

```
> 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)),]
           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
> AMZN.Ret<-Delt(data.AMZN$AMZN.Adjusted)
> AMZN.Ret[c(1:3,nrow(AMZN.Ret)),]
           Delt.1.arithmetic
2010-12-31                NA
2011-01-03       0.023444444
2011-01-04       0.004288351
2013-12-31       0.013778377
>
> data.IBM<-read.csv("IBM Yahoo.csv",header=TRUE)
> date<-as.Date(data.IBM$Date,format="%Y-%m-%d")
> data.IBM<-cbind(date, data.IBM[,-1])
> data.IBM<-data.IBM[order(data.IBM$date),]
> data.IBM<-xts(data.IBM[,2:7],order.by=data.IBM[,1])
> names(data.IBM)<-
+    paste(c("IBM.Open","IBM.High","IBM.Low",
+    "IBM.Close","IBM.Volume","IBM.Adjusted"))
> data.IBM[c(1:3,nrow(data.IBM)),]
           IBM.Open IBM.High IBM.Low IBM.Close IBM.Volume IBM.Adjusted
2010-12-31   146.73   147.07  145.96    146.76    2969800       139.23
2011-01-03   147.21   148.20  147.14    147.48    4603800       139.91
2011-01-04   147.56   148.22  146.64    147.64    5060100       140.06
2013-12-31   186.49   187.79  186.30    187.57    3619700       187.57
> IBM.Ret<-Delt(data.IBM$IBM.Adjusted)
> IBM.Ret[c(1:3,nrow(IBM.Ret)),]
           Delt.1.arithmetic
2010-12-31                NA
2011-01-03       0.004884005
2011-01-04       0.001072118
2013-12-31       0.006222842
```

Step 3: Combine the Two Return Series   Notice that we still have variable names that do not make sense, so we also rename the variable names to `AMZN.Ret` and `IBM.Ret`.

```
> returns<-cbind(AMZN.Ret,IBM.Ret)
> returns[c(1:3,nrow(returns)),]
           Delt.1.arithmetic Delt.1.arithmetic.1
2010-12-31                NA                  NA
2011-01-03       0.023444444         0.004884005
2011-01-04       0.004288351         0.001072118
2013-12-31       0.013778377         0.006222842
> names(returns)<-paste(c("AMZN.Ret","IBM.Ret"))
> returns[c(1:3,nrow(returns)),]
              AMZN.Ret     IBM.Ret
2010-12-31          NA          NA
2011-01-03 0.023444444 0.004884005
2011-01-04 0.004288351 0.001072118
2013-12-31 0.013778377 0.006222842
> returns<-returns[-1,]
> returns[c(1:3,nrow(returns)),]
              AMZN.Ret      IBM.Ret
2011-01-03 0.023444444  0.004884005
2011-01-04 0.004288351  0.001072118
2011-01-05 0.013026323 -0.003998286
2013-12-31 0.013778377  0.006222842
```

Step 4: Calculate Standard Deviation and Covariance of the Securities   In Eq. (4.2), we need the standard deviation of AMZN returns, standard deviation of IBM returns, and the covariance of AMZN and IBM returns. As shown above, we calculate standard deviation using the `sd` command. To calculate covariance, we use the `cov` command.

```
> sd.AMZN<-sd(returns$AMZN.Ret)*sqrt(252)
> sd.AMZN
[1] 0.3272979
> sd.IBM<-sd(returns$IBM.Ret)*sqrt(252)
> sd.IBM
[1] 0.1923527
> ret.cov<-cov(returns$AMZN.Ret,returns$IBM.Ret)*252
> ret.cov
             IBM.Ret
AMZN.Ret 0.02326468
```

Note that all the values have been annualized. Since standard deviation scales up with the square root of time, we convert daily standard deviations to annual standard deviations by multiplying the daily standard deviation by the square root of 252. For covariance, the values scale up with time. As such, we multiply the daily covariance by 252 to get to the annualized number.

Verifying Relationship Between Covariance and Correlation
Recall that the covariance term is also equivalent to the product of the correlation between AMZN and IBM returns and the individual security standard deviations, i.e., $\sigma_{1,2} = \rho_{1,2}\sigma_1\sigma_2$. To verify this is the case, we can calculate the correlation between AMZN and IBM returns using the `cor` command. Based on this alternative approach, we calculate a covariance of 0.0233, which is identical to our previous result.

```
> ret.correl<-cor(returns$AMZN.Ret,returns$IBM.Ret)
> ret.correl
          IBM.Ret
AMZN.Ret 0.369535
>
> ret.correl*sd.AMZN*sd.IBM
            IBM.Ret
AMZN.Ret 0.02326468
```

Step 5: Calculate Portfolio Risk We can now calculate the portfolio risk for this two-asset portfolio. Using Eq. (4.2), we calculate the portfolio variance as 0.0362. We take the square root of that to get to a portfolio standard deviation of 19.0 %.

```
> port.var<-wgt.AMZN^2*sd.AMZN^2 + wgt.IBM^2*sd.IBM^2 +
+    2*ret.cov*wgt.AMZN*wgt.IBM
> port.var
          IBM.Ret
AMZN.Ret 0.03623176
> port.sd<-sqrt(port.var)
> port.sd
          IBM.Ret
AMZN.Ret 0.1903464
```

Benefit of Diversification
Note that this portfolio standard deviation is lower than the weighted-average standard deviation of AMZN and IBM returns of 22.6 %. Since AMZN and IBM returns only have a 0.37 correlation, portfolio risk is reduced by combining the two securities in a portfolio. In fact, the smaller the correlation, the more gains from diversification we can achieve.

```
> wgtd.sd<-wgt.AMZN*sd.AMZN+wgt.IBM*sd.IBM
> wgtd.sd
[1] 0.226089
```

4.3.2 Two Assets (Matrix Algebra)

We have three terms in Eq. (4.2), which is the formula to calculate the portfolio risk for two assets. The number of terms required to calculate portfolio risk grows with the number of securities you add in your portfolio because the covariance term (e.g., the third term in Eq. (4.2)) has to be calculated for each pair of securities in the portfolio. For N securities, we have N variance terms and $N(N-1)$ covariance terms. As such, if we had 10 securities, we would have 10 variance terms and 90 [= (10 * 9)] covariance terms.

Since most portfolios are constructed with more than two securities, manually laying out and calculating each set of covariance terms becomes extremely cumbersome and could get quite confusing to implement. However, in R, we can use matrix algebra to make this process simpler.

The formula for portfolio variance σ_P^2 in matrix form is

$$\sigma_P^2 = w\Sigma w^T, \tag{4.3}$$

where w is a vector of weights and Σ is the Variance–Covariance matrix of the securities in the portfolio. Although different texts may use different symbols, w and Σ are symbols typically used to refer to these variables in the finance literature. One can think of a *vector* as a column or row or numbers. A *matrix* is a set of vectors stacked either on top of each other or side-by-side. The T superscript denotes the transposition of a matrix, which in our cases flips a row vector of weights into a column vector of weights.

Let us now show how we calculate the portfolio risk for the same two-asset portfolio in our earlier example but using matrix algebra.

Step 1: Create Vector of Weights First, we know the weights of AMZN and IBM in the portfolio are 25 and 75 %, respectively. So we create a vector called WGT.2asset that holds the weights of the two assets. Then, we convert this vector into a 1 (row) $\times$ 2 (column) matrix using the matrix commands. We use the number 1 to signify that we want there to be one row. This 1×2 matrix WGT.2asset actually corresponds to the first term in Eq. (4.3) of w.

```
> WGT.2asset<-c(0.25,0.75)
> WGT.2asset
[1] 0.25 0.75
> WGT.2asset<-matrix(WGT.2asset,1)
> WGT.2asset
     [,1] [,2]
[1,] 0.25 0.75
```

Step 2: Create Transposed Vector of Weights We then need to transpose this vector from 1×2 to a 2×1 matrix. We do this by using the t(...) command. Thus, tWGT.2asset corresponds to w^T in Eq. (4.3).

```
> tWGT.2asset<-t(WGT.2asset)
> tWGT.2asset
    [,1]
[1,] 0.25
[2,] 0.75
```

Step 3: Construct Variance–Covariance Matrix To calculate the variance-covariance (VCOV) matrix in R, we have to convert `returns` into a matrix using the `as.matrix` command.

```
> mat.Ret<-as.matrix(returns)
> head(mat.Ret)
               AMZN.Ret       IBM.Ret
2011-01-03  0.023444444  0.004884005
2011-01-04  0.004288351  0.001072118
2011-01-05  0.013026323 -0.003998286
2011-01-06 -0.008323551  0.010967742
2011-01-07 -0.001990746 -0.004892576
2011-01-10 -0.004366812 -0.001995155
```

Then, we calculate the covariance using the `cov` command. Prior to using that command, we used the `scipen` argument of 100. This allows for a higher threshold before R converts the numbers into scientific notation, which are typically harder to read.

```
> options(scipen="100")
> cov(mat.Ret)
              AMZN.Ret        IBM.Ret
AMZN.Ret 0.00042509499 0.00009232017
IBM.Ret  0.00009232017 0.00014682368
```

Note that when the data is in matrix form, multiplying the matrix by a number would multiply every number in the matrix by that number (i.e., this is known as *scalar multiplication*). In our case, to annualize the variances and covariance we multiplied the matrix by 252 and all the elements in the matrix are multiplied by 252.

```
> VCOV.2asset<-cov(mat.Ret)*252
> VCOV.2asset
          AMZN.Ret    IBM.Ret
AMZN.Ret 0.10712394 0.02326468
IBM.Ret  0.02326468 0.03699957
```

Let us discuss for a moment what is in the annualized variance-covariance matrix. The variance-covariance matrix is a *square matrix*, which means that it has the same number of rows and columns. In the *main diagonal* of the matrix (i.e., going from the top-left to the bottom-right), the values represent the variances of the securities in the portfolio. In this case, the main diagonal contains the values 0.10712394, which is the variance of AMZN returns, and 0.03699957, which is the variance of IBM returns. As more securities are added, the elements in the main diagonal are the variances of each of those securities. The terms in the *off-diagonal* are the covariances of each

pair of securities. In our two-asset example, the 0.02326468 is the covariance of AMZN and IBM returns. The values of the covariance for each pair appear twice, because the covariance of AMZN/IBM returns is the same as the covariance of the IBM/AMZN returns. As discussed above, as the number of securities in the portfolio grows bigger, we will have two covariance terms for each pair of securities and the matrix just becomes larger and larger.

Step 4: Calculate Portfolio Risk We multiply all three terms to get the portfolio variance. Note that matrix multiplication requires a different notation than simple multiplication. To multiply matrices, we use `%*%` or the asterisk (*) symbol sandwiched in between two percent (%) symbols.

```
> mat.var2asset<-WGT.2asset %*% VCOV.2asset %*% tWGT.2asset
> mat.var2asset
           [,1]
[1,] 0.03623176
```

Note that to calculate the portfolio standard deviation, we can still use the `sqrt` command even on a matrix. As shown above, the portfolio standard deviation calculated using matrix algebra is equal to 19.03 %, which is identical to the portfolio standard deviation we calculated above using Eq. (4.2).

```
> mat.sd2asset<-sqrt(mat.var2asset)
> mat.sd2asset
          [,1]
[1,] 0.1903464
```

4.3.3 Multiple Assets

In this section, we show how to extend the technique to calculate portfolio risk using matrix algebra for multiple assets. The use of matrix algebra is necessary for large portfolios and we will see this technique again when we discuss mean-variance portfolio optimization in a later chapter. We continue from our previous example, and use returns data from 2011 to 2013.

Step 1: Import Data for AMZN, IBM, YHOO, and TSLA Continuing from our two asset example from the last section, we add two additional securities to our portfolio. These will be Yahoo (YHOO) and Tesla (TSLA). We assume that we invested $ 2000 in AMZN, $ 2000 in YHOO, $ 3000 in IBM, and $ 3000 in TSLA. Since we will still be using AMZN and IBM data, we only need to check that the data for these two securities are still available in the memory. Otherwise, we need to upload them again.

```
> 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
```

Next, we upload the data from YHOO and TSLA from the CSV files we download from Yahoo Finance. The YHOO data should be saved as **YHOO Yahoo.csv** and the TSLA data should be saved as **TSLA Yahoo.csv**.

```
> # 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)<-
+     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
>
> # TSLA Data
> data.TSLA<-read.csv("TSLA Yahoo.csv",header=TRUE)
> date<-as.Date(data.TSLA$Date,format="%Y-%m-%d")
> data.TSLA<-cbind(date, data.TSLA[,-1])
> data.TSLA<-data.TSLA[order(data.TSLA$date),]
> data.TSLA<-xts(data.TSLA[,2:7],order.by=data.TSLA[,1])
> names(data.TSLA)<-
+     paste(c("TSLA.Open","TSLA.High","TSLA.Low",
+     "TSLA.Close","TSLA.Volume","TSLA.Adjusted"))
> data.TSLA[c(1:3,nrow(data.TSLA)),]
           TSLA.Open TSLA.High TSLA.Low TSLA.Close TSLA.Volume TSLA.Adjusted
2010-12-31     26.57     27.25    26.50      26.63     1417900         26.63
2011-01-03     26.84     27.00    25.90      26.62     1283000         26.62
2011-01-04     26.66     26.95    26.02      26.67     1187400         26.67
2013-12-31    152.32    153.20   148.66     150.43     4262400        150.43
```

Step 2: Extract Adjusted Prices of Each Security Since we only need the adjusted closing prices for each of the four securities to calculate their daily returns, we create

the data object `multi` to hold only the adjusted closing prices. The adjusted closing price is the sixth column in the data.

```
> multi<-data.AMZN[,6]
> multi<-merge(multi,data.YHOO[,6])
> multi<-merge(multi,data.IBM[,6])
> multi<-merge(multi,data.TSLA[,6])
> multi[c(1:3,nrow(multi)),]
           AMZN.Adjusted YHOO.Adjusted IBM.Adjusted TSLA.Adjusted
2010-12-31        180.00         16.63       139.23         26.63
2011-01-03        184.22         16.75       139.91         26.62
2011-01-04        185.01         16.59       140.06         26.67
2013-12-31        398.79         40.44       187.57        150.43
```

Step 3: Calculate Returns for Each Security   By this point, we would have already gone through the return calculation several times that it is worth putting forth a new and more efficient set of code. In this particular case, we want to calculate the returns for all four securities. We will show how to do this by repeatedly applying a function to each of the series.

We convert `multi` to a matrix using the `matrix` command. Note that the `matrix` command include a second argument, which is the number of rows. This is easily identified by using the `nrow` command.

```
> mat.price<-matrix(multi,nrow(multi))
```

Next, we create a *function* so that we can apply the `Delt` command repeatedly.

```
> prc2ret<-function(x) Delt(x)
```

Then, we use the `apply` command to repeatedly apply to the columns of the matrix `mat.price` (rows use number 1, columns use number 2, and rows and columns use 1:2), the function `Delt`.

```
> mat.ret<-apply(mat.price,2,function(x){prc2ret(c(x))})
> mat.ret[1:4,]
            [,1]          [,2]          [,3]          [,4]
[1,]          NA            NA            NA            NA
[2,] 0.023444444  0.007215875  0.004884005 -0.0003755163
[3,] 0.004288351 -0.009552239  0.001072118  0.0018782870
[4,] 0.013026323  0.019288728 -0.003998286  0.0059992501
```

Note that when we show the output of matrices, we show the first four observations. The reason is that the numbering of the rows does not change regardless of whether we show different row numbers. The numbering will always be the number of rows that are reported. As such, this can result in confusion. To avoid any confusion, unless explicitly stated, matrices are going to be reported by showing the first few observations.

Step 4: Clean up Returns Data   The output shows that the first row of `mat.ret` are NAs. This is essentially equal to the December 31, 2010 observation when we

calculate returns. As such, we would want to delete that observation. The matrix
commands are similar to the normal R commands, so we type [−1,] to delete the
first row.

```
> mat.ret<-mat.ret[-1,]
> mat.ret[1:4,]
             [,1]         [,2]          [,3]           [,4]
[1,]  0.023444444  0.007215875  0.004884005 -0.0003755163
[2,]  0.004288351 -0.009552239  0.001072118  0.0018782870
[3,]  0.013026323  0.019288728 -0.003998286  0.0059992501
[4,] -0.008323551  0.008870491  0.010967742  0.0391352963
```

To make it clear what security is in which column, we will rename the column headers.

```
> colnames(mat.ret)<-c("AMZN","YHOO","IBM","TSLA")
> mat.ret[1:4,]
            AMZN         YHOO          IBM           TSLA
[1,]  0.023444444  0.007215875  0.004884005 -0.0003755163
[2,]  0.004288351 -0.009552239  0.001072118  0.0018782870
[3,]  0.013026323  0.019288728 -0.003998286  0.0059992501
[4,] -0.008323551  0.008870491  0.010967742  0.0391352963
```

Step 5: Calculate Annualized Variance–Covariance Matrix Now, we turn to
calculating the variance–covariance (VCOV) matrix. To perform this task, we
use the `cov` command. When applied to a matrix, the `cov` command will gen-
erate a VCOV matrix. If our output is in scientific notation, recall we used the
`options(scipen="100")` command earlier to increase the threshold before R
converts the values into scientific notation. That command should still be in effect
through this point. Otherwise, we can type the command again.

```
> VCOV<-cov(mat.ret)
> VCOV
            AMZN         YHOO          IBM           TSLA
AMZN 0.00042509499 0.00014525257 0.00009232017 0.00018498010
YHOO 0.00014525257 0.00036837396 0.00007541591 0.00016564269
IBM  0.00009232017 0.00007541591 0.00014682368 0.00009617837
TSLA 0.00018498010 0.00016564269 0.00009617837 0.00133374500
```

Next, we annualize the VCOV matrix by multiplying the daily variances and
covariances by 252.

```
> VCOV.annual<-252 * VCOV
> VCOV.annual
           AMZN       YHOO        IBM       TSLA
AMZN 0.10712394 0.03660365 0.02326468 0.04661499
YHOO 0.03660365 0.09283024 0.01900481 0.04174196
IBM  0.02326468 0.01900481 0.03699957 0.02423695
TSLA 0.04661499 0.04174196 0.02423695 0.33610374
```

The diagonal elements (top-left going to bottom-right) of the VCOV matrix are the variances of the securities. Note that the AMZN and IBM values are identical to the variances calculated in the previous section. The off-diagonal elements are the covariances between the pairs of securities. For example, AMZN/YHOO and YHOO/AMZN have a covariance of 0.3660 and IBM/TSLA and TSLA/IBM have a covariance of 0.02424.

Step 6: Create a Row vector of Weights The other element necessary to calculate the portfolio risk would be to create a matrix of weights. From the assumptions above, we have AMZN with weight of 20 %, YHOO with weight of 20 %, IBM with weight of 30 %, and TSLA with weight of 30 %. As a note, we have to make sure the weights in the portfolio sum to one. This means we have fully allocated the investment in the portfolio.

```
> wgt=c(.2,.2,.3,.3)
> mat.wgt<-matrix(wgt,1)
> mat.wgt
     [,1] [,2] [,3] [,4]
[1,]  0.2  0.2  0.3  0.3
```

We manually type in the weights in `wgt`. Then, we convert `wgt` into a matrix using the `matrix` command. Again, since we want this to be a row vector, we enter "1" to the right side of the comma.

Step 7: Create a Column Vector of Weights by Transposing the Row Vector of Weights We also need to create a transposed version of `mat.wgt`, which is done by applying the `t(...)` command.

```
> tmat.wgt<-t(mat.wgt)
> tmat.wgt
     [,1]
[1,]  0.2
[2,]  0.2
[3,]  0.3
[4,]  0.3
```

Step 8: Calculate the Portfolio Variance Using matrix multiplication, we multiply the weigths and VCOV matrix to calculate the portfolio variance.

```
> port.var<-mat.wgt %*% VCOV.annual %*% tmat.wgt
> port.var[1,1]
[1] 0.06454358
```

Step 9: Calculate the Portfolio Standard Deviation Then, taking the square root of `port.var` yields the portfolio standard deviation.

```
> port.sd<-sqrt(port.var)
> port.sd[1,1]
[1] 0.2540543
```

The output above shows that the portfolio standard deviation is 25.4 %. The portfolio standard deviation for this four asset portfolio is higher than the two-asset portfolio, which had a portfolio standard deviation of 19.0 %. The reason for this is that we added TSLA, which has a portfolio variance of 0.336 and that translates to a standard deviation of 58.0 %.

4.4 Value-at-Risk

A popular measure of the risk of loss in a portfolio is *Value-at-Risk* or *VaR*. VaR measures the loss in our portfolio over a pre-specified time horizon, assuming some level of probability. For example, we will calculate as an example the 1 and 5 % 1-Day VaR on December 31, 2013 of the Value-Weighted portfolio we constructed in Chapter 3, which was based on a portfolio of AMZN, IBM, and YHOO. In this section, we will use the market convention of representing VaR as a positive number and using the significance level (e.g., 1 or 5 %) instead of confidence level (e.g., 99 or 95 %). We also implement two types of VaR calculations: Gaussian VaR and Historical VaR. The former assumes that the data follow a normal or Gaussian distribution, while the latter uses the distributional properties of the actual data. As such, Historical VaR requires more data to implement and Gaussian VaR requires less data to implement.

VaR has been criticized for not being able to fully capture how large the loss we should expect if the loss exceeds VaR or what is known as *tail risk*. If the tail of the distribution is large, then there is a risk of a very large loss. Because of this limitation of VaR, the concept of *Expected Shortfall* (ES) was developed. ES tells us how much we should expect to lose if our losses exceed VaR. In fact, the Basel Committee on Banking Supervision is proposing to change the system banks use to calculate losses from VaR to ES.[3]

In this section, we calculate the VaR using both the Gaussian VaR/ES and Historical VaR/ES based on the Value-Weighted Portfolio we constructed in Chap. 3, which consists of AMZN, IBM, and YHOO. The value of our hypothetical portfolio on December 31, 2013 is $ 1,259,719. We will calculate the 1-Day VaR based on a 1 and 5 % significance level.

4.4.1 Gaussian VaR

One of the simplest approaches to estimate VaR is to assume that the portfolio returns follow a normal distribution. Hence, this approach is called *Gaussian VaR*. Because

---

[3] Bank of International Settlements Basel Committee on Banking Supervision (May 2012) Consultative documents: Fundamental review of the trading book, retrieved from http://www.bis.org/publ/bcbs219.pdf on January 5, 2014.

of the distributional assumption, we only need at least one year of daily returns data, which is approximately 252 observations.

Step 1: Import Daily Portfolio Returns for the Last Year To implement Gaussian VaR, we begin by importing the **Hypothetical Portfolio (Daily).csv** we constructed in Chap. 3. We will use Value-Weighted portfolio returns `VW.ret` in our calculation of Gaussian VaR. Assuming we placed $ 1 million at the start of the period in the portfolio, the portfolio value has grown to $ 1.26 million as of December 31, 2013.

```
> port.ret<-read.csv("Hypothetical Portfolio (Daily).csv")
> port.ret[c(1:3,nrow(port.ret)),]
      X       date   VW.cum   EW.cum        VW.ret        EW.ret
1     1 2012-12-31 1.000000 1.000000            NA            NA
2     2 2013-01-02 1.024248 1.019927  0.024247508  0.019927472
3     3 2013-01-03 1.021361 1.014577 -0.002818685 -0.005246421
253 253 2013-12-31 1.259719 1.501384  0.009406090  0.008864921
>
> port.ret<-port.ret$VW.ret[-1]
> port.ret[1:5]
[1]  0.024247508 -0.002818685 -0.002881561  0.007731793 -0.002645288
```

Step 2: Calculate Mean and Standard Deviation of Historical Daily Portfolio Returns We calculate the average or mean portfolio return using the `mean` command and we calculate the standard deviation of returns using the `sd` command. The portfolio mean is 0.10 %. In practice, some assume that because we are calculating VaR over a short horizon, the portfolio mean return is zero. In our example, we do not make this assumption but we can see from the output below that our calculated portfolio mean is small.

```
> port.mean<-mean(port.ret)
> port.mean
[1] 0.0009716944
>
> port.risk<-sd(port.ret)
> port.risk
[1] 0.01049939
```

Step 3: Calculate 1 and 5 % VaR The VaR is calculated as

$$VaR_\alpha = -(\mu - \sigma * Z_\alpha) * I, \tag{4.4}$$

where α is the significance level of the VaR, μ is the portfolio average return, σ is the standard deviation of the portfolio returns, Z_α is the z-score based on the VaR significance level, and I is the current portfolio value. The z-score is calculated using the `qnorm` command, which returns the inverse cumulative density function. The

qnorm function takes as an argument the desired significance level (e.g., 1 or 5 %).
As such, for the 1 % Gaussian VaR, we use qnorm(0.01).

```
> VaR01.Gaussian<- -(port.mean+port.risk*qnorm(0.01))*1259719
> VaR01.Gaussian<-format(VaR01.Gaussian,big.mark=',')
> VaR01.Gaussian
[1] "29,544.88"
>
> VaR05.Gaussian<- -(port.mean+port.risk*qnorm(0.05))*1259719
> VaR05.Gaussian<-format(VaR05.Gaussian,big.mark=',')
> VaR05.Gaussian
[1] "20,531.24"
```

This means that there is a 1 % (5 %) chance that our portfolio loses more than $ 29,545
($ 20,531) over the next day. Note that to make the VaR result easier to read, we used
the format command to format the VaR output to include a comma.

4.4.2   Historical VaR

Historical VaR uses a mix of current weights in the portfolio and a simulation of
historical returns of the securities in the portfolio to construct a simulated series of
portfolio Profits & Losses (P&L). Given its use of historical returns data, Historical
VaR works better when we have lots of data. Typically, three to five years of data is
recommended for this approach.

**Step 1: Import Three Years of Daily Returns Data for Each Security in the
Portfolio** Since our portfolio consists of AMZN, IBM, and YHOO, we have to
obtain daily data for the three securities. We then calculate the daily returns for each
of the securities.

```
> 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
> AMZN.Ret<-Delt(data.AMZN$AMZN.Adjusted)
> AMZN.Ret[c(1:3,nrow(AMZN.Ret)),]
           Delt.1.arithmetic
2010-12-31                NA
2011-01-03       0.023444444
2011-01-04       0.004288351
2013-12-31       0.013778377
>
> data.YHOO[c(1:3,nrow(data.YHOO)),]
           YHOO.Open YHOO.High YHOO.Low YHOO.Close YHOO.Volume YHOO.Adjusted
2010-12-31     16.74     16.76    16.47      16.63     7754500         16.63
2011-01-03     16.81     16.94    16.67      16.75    17684000         16.75
2011-01-04     16.71     16.83    16.57      16.59    11092800         16.59
2013-12-31     40.17     40.50    40.00      40.44     8291400         40.44
> YHOO.Ret<-Delt(data.YHOO$YHOO.Adjusted)
> YHOO.Ret[c(1:3,nrow(YHOO.Ret)),]
           Delt.1.arithmetic
2010-12-31                NA
2011-01-03       0.007215875
2011-01-04      -0.009552239
2013-12-31       0.005970149
>
> 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
> IBM.Ret<-Delt(data.IBM$IBM.Adjusted)
> IBM.Ret[c(1:3,nrow(IBM.Ret)),]
           Delt.1.arithmetic
2010-12-31                NA
2011-01-03       0.004884005
2011-01-04       0.001072118
2013-12-31       0.006222842
```

Step 2: Combine Returns Data into One Data Object Using the `cbind` com-
mand, we combine the returns for AMZN, YHOO, and IBM. The resulting data
has variable names that are unhelpful. As such, we rename the variable names to
something more meaningful.

```
> ret.data<-cbind(AMZN.Ret[-1,],YHOO.Ret[-1,],IBM.Ret[-1,])
> ret.data[c(1:3,nrow(ret.data)),]
            Delt.1.arithmetic Delt.1.arithmetic.1 Delt.1.arithmetic.2
2011-01-03        0.023444444         0.007215875          0.004884005
2011-01-04        0.004288351        -0.009552239          0.001072118
2011-01-05        0.013026323         0.019288728         -0.003998286
2013-12-31        0.013778377         0.005970149          0.006222842
> names(ret.data)<-paste(c("AMZN.Ret","YHOO.Ret","IBM.Ret"))
> ret.data[c(1:3,nrow(ret.data)),]
              AMZN.Ret     YHOO.Ret       IBM.Ret
2011-01-03 0.023444444  0.007215875  0.004884005
2011-01-04 0.004288351 -0.009552239  0.001072118
2011-01-05 0.013026323  0.019288728 -0.003998286
2013-12-31 0.013778377  0.005970149  0.006222842
```

Step 3: Identify the Value of Each Security in the Portfolio as of December 31, 2013 Using data from when we constructed the portfolio in Chap. 3, we identify the ending values for each security in the portfolio. As of December 31, 2013, the amount of the portfolio in AMZN was $ 537,000, YHOO was $ 120,500, and IBM was $ 602,200. This means the total portfolio value as of December 31, 2013 was $ 1,259,700.

```
> last.idx<-c(0.5370,0.1205,0.6022)*1000000
> last.idx
[1] 537000 120500 602200
>
> port.val<-sum(last.idx)
> port.val
[1] 1259700
```

Step 4: Calculate Simulated Portfolio Returns Applying Current Portfolio Weights to Historical Security Returns We simulate portfolio returns using the three years of historical returns data of each security in the portfolio. We assume that the current value of each security in the portfolio remains frozen over our VaR time horizon (e.g., one day). Therefore, each return observation is applied to the current value of the security. That is,

$$P\&L = V_{AMZN} R_{AMZN}^t + V_{YHOO} * R_{YHOO}^t + V_{IBM} * R_{IBM}^t, \qquad (4.5)$$

where V denotes the value of the security in the portfolio and R is the return of the security on day t. The variable name generated by this step is AMZN.Ret, which is misleading, so we rename this variable to Port.PnL, which is more fitting.

```
> sim.portPnL<-last.idx[1]*ret.data$AMZN.Ret+
+    last.idx[2]*ret.data$YHOO.Ret+
+    last.idx[3]*ret.data$IBM.Ret
> sim.portPnL[c(1:3,nrow(sim.portPnL)),]
             AMZN.Ret
2011-01-03 16400.327
2011-01-04  1797.429
2011-01-05  6911.659
2013-12-31 11865.787
>
> names(sim.portPnL)<-paste("Port.PnL")
> sim.portPnL[c(1:3,nrow(sim.portPnL)),]
             Port.PnL
2011-01-0316400.327
2011-01-041797.429
2011-01-056911.659
2013-12-3111865.787
```

Step 5: Calculate Appropriate Quantile for the 1 and 5 % VaR To find the VaR
of significance level α, we use the `quantile` command. The first argument is
the *negative* of the simulated portfolio P&L and the second argument is the $1 - \alpha$
confidence level. We then format the output so that the result has a comma, which
makes the results a little easier to read.

```
> VaR01.Historical=quantile(-sim.portPnL$Port.PnL,0.99)
> VaR01.Historical<-format(VaR01.Historical,big.mark=',')
> VaR01.Historical
       99%
"39,743.93"
>
> VaR05.Historical=quantile(-sim.portPnL$Port.PnL,0.95)
> VaR05.Historical<-format(VaR05.Historical,big.mark=',')
> VaR05.Historical
       95%
"23,556.31"
```

Note that the higher the significance level (i.e., 5 vs. 1 %) or the lower the confidence
level (i.e., 95 vs. 99 %), the lower the VaR amount. To see why, suppose we were
asked to give our best estimate of a number that we are sure we would not exceed.
The higher the estimate of this number we come up with, the higher the likelihood
that we will not exceed it. As such, we should expect to have a higher number at the
99 % confidence level than at the 95 % confidence level.

Step 6: Plot the VaR in Relation to P&L Density Sometimes it may be easier for
us to visualize the data by plotting (i) the density of the simulated portfolio P&L, (ii)
the normal distribution of P&L's based on the mean and standard deviation of the
simulated portfolio P&Ls, and (iii) our estimates of the 1 and 5 % 1-Day Historical
VaR. We first plot (i) and (iii), then we add (ii).

To get the density of the simulated portfolio P&L, we use the `density` command. Typing in `ret.d` generates the output shown below.

```
> ret.d=density(sim.portPnL$Port.PnL)
> ret.d

Call:
        density.default(x = sim.portPnL$Port.PnL)

Data: sim.portPnL$Port.PnL (754 obs.);  Bandwidth 'bw' = 3336

      x                 y
 Min.   :-78186   Min.   :0.000000001825
 1st Qu.:-34022   1st Qu.:0.000000155868
 Median : 10143   Median :0.000000666051
 Mean   : 10143   Mean   :0.000005655116
 3rd Qu.: 54307   3rd Qu.:0.000008209102
 Max.   : 98471   Max.   :0.000028428691
```

We can now plot (i) and (iii). Note that we add two vertical lines using the `abline` command. The two lines denote our estimates of VaR, so we can visually see from the density of the simulated portfolio P&L, where the Historical VaR estimates are.

```
> plot(ret.d,
+     xlab="Profit & Loss",
+     ylab="",
+     yaxt="n",
+     main="Density of Simulated Portfolio P&L Over Three Years
+         And 1% and 5% 1-Day Historical Value-at-Risk (VaR)")
> abline(v=-quantile(-sim.portPnL$Port.PnL,0.99),col="gray",lty=1)
> abline(v=-quantile(-sim.portPnL$Port.PnL,0.95),col="black",lty=2)
```

Next, we now turn to number (ii) on the list, which is the plot of the normal distribution based on the mean and standard deviation of the simulated portfolio P&L. First, we create a sequence of 1000 numbers between the smallest and largest simulated P&L. This creates bounds for the normal density plot that follows.

```
> x<-seq(min(sim.portPnL$Port.PnL),max(sim.portPnL$Port.PnL),length=1000)
> head(x)
[1] -68178.87 -68022.07 -67865.27 -67708.47 -67551.67 -67394.87
> tail(x)
[1] 87680.07 87836.87 87993.67 88150.47 88307.27 88464.07
```

Then, we use the `dnorm` command to provide the values of the probability density function for the normal distribution. The `dnorm` command takes on three arguments. The first argument uses the `x` vector we created above. The second argument is the mean of the simulated portfolio P&L, which we calculate using the `mean` command. The third argument is the standard deviation of the simulated portfolio P&L, which

we calculate using the `sd` command. This creates a normal density based on the mean and standard deviation of the simulated portfolio P&L.

```
> y<-dnorm(x,mean=mean(sim.portPnL$Port.PnL),sd=sd(sim.portPnL$Port.PnL))
> head(y)
[1] 0.000000003393869 0.000000003532761 0.000000003677004 0.000000003826788
[5] 0.000000003982313 0.000000004143781
> tail(y)
[1] 0.00000000002364541 0.00000000002248787 0.00000000002138504
[4] 0.00000000002033446 0.00000000001933373 0.00000000001838057
```

We then use the `lines` command to add the normal density plot as a dashed bell curve onto the plot.

```
> lines(x,y,type="l",col="black",lwd=1,lty=3)
```

Finally, we add a legend on the top-right portion of the chart, so the legend does not obstruct the other elements of the chart.

```
> legend("topright",
+    c("Simulated P&L Distribution",
+    "Normal Distribution",
+    "1% 1-Day VaR","5% 1-Day VaR"),
+    col=c("black","black","gray","black"),
+    lty=c(1,3,1,2))
```

Figure 4.3 shows the results of our plot. The chart shows that the simulated P&L distribution is more peaked than what a normal distribution with the same mean and standard deviation would show. This means that there are more observations that are around the mean or average P&L of $ 1127 in the simulated P&L.

Using the Square Root of T rule to Scale 1-Day VaR To a T-Day VaR
Assuming the returns are independent and identically distributed (i.i.d), we can calculate a T-Day VaR by multiplying the 1-Day VaR by $\sqrt{T}$. For example, our 1 % 1-Day Historical VaR was $ 39,744. Assuming returns are i.i.d., the 1 % 10-Day Historical VaR is equal to $ 125,682.

```
> VaR01.10day<-quantile(-sim.portPnL$Port.PnL,0.99)*sqrt(10)
> VaR01.10day
     99%
125681.3
```

Density of Simulated Portfolio P&L Over Three Years
And 1% and 5% 1-Day Historical VaR
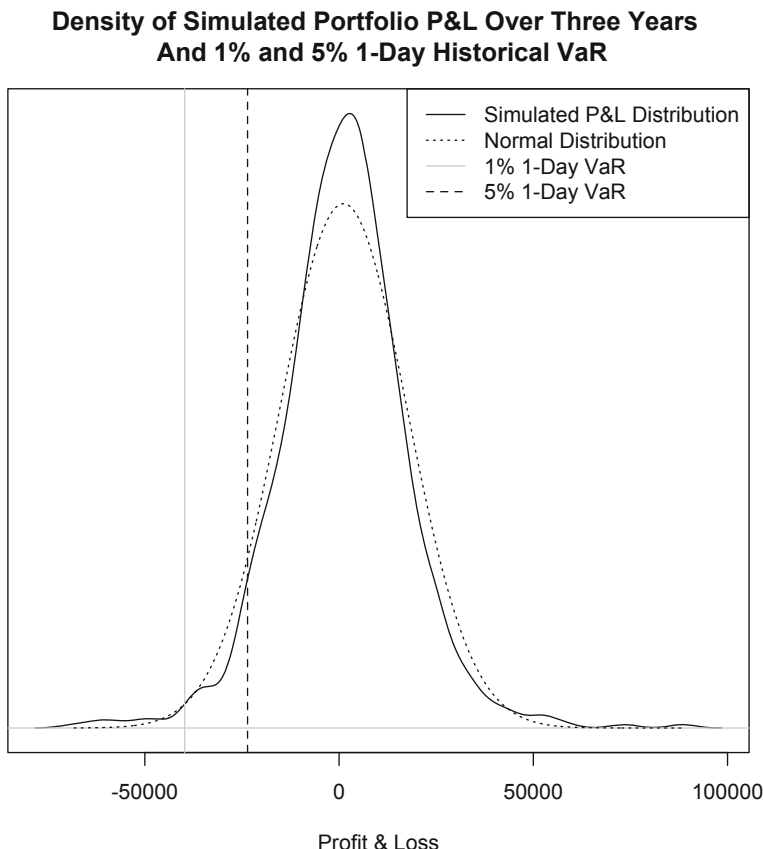
Fig. 4.3 Historical 1 and 5 % 1-day VaR based on simulated portfolio profits & losses. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

4.5   Expected Shortfall

One of the issues with VaR is that it does not capture the shape of the tail of the distribution. That is, VaR does not allow us to answer the question, "if losses exceed VaR, how bad should we expect that loss to be?" The answer to this is called *expected shortfall* (ES). ES is known by many other names, such as *tail VaR* and *tail loss*.[4] To parallel our VaR discussion, we show two methods of calculating ES: Gaussian ES

---

[4] In fact, some texts may call our ES measure *expected tail loss* and use the term *expected shortfall* to denote the average loss that exceeds a benchmark VaR. We stay with our definition of ES, but bear in mind some texts may not have the same definition of ES. The general idea is the same except that the VaR limit from which the ES is calculated using either the portfolio itself or some benchmark portfolio.

and Historical ES. Similar to the difference between Historical VaR and Gaussian VaR, the difference between Historical ES and Gaussian ES is the assumption of a specific distribution or a lack thereof.

4.5.1 Gaussian ES

Similar to VaR, we can also calculate ES by assuming a normal distribution. This is known as a *Gaussian ES*. The calculation of Gaussian ES is

$$\mu + \sigma * \mathtt{dnorm}(\mathtt{qnorm}(\alpha))/\alpha, \tag{4.6}$$

where ϕ is the $\mathtt{dnorm}$ function (i.e., height of the normal probability density function) in R and Φ is the $\mathtt{qnorm}$ function (i.e., inverse normal cumulative density function) in R. μ is the mean portfolio return, σ is the standard deviation of the portfolio returns, and α is the significance level.

```
> ES01.Gaussian<-1259719*(port.mean+port.risk*(dnorm(qnorm(.01))/.01))
> ES01.Gaussian<-format(ES01.Gaussian,big.mark=',')
> ES01.Gaussian
[1] "36,474.94"
>
> ES05.Gaussian<-1259719*(port.mean+port.risk*(dnorm(qnorm(.05))/.05))
> ES05.Gaussian<-format(ES05.Gaussian,big.mark=',')
> ES05.Gaussian
[1] "28,506.09"
```

The result means that there is a 1 % (5 %) chance our losses exceed VaR, but when it does, we expect that, on average, we will lose $ 36,475 ($ 28,506).

4.5.2 Historical ES

The Historical ES is calculated by taking the average portfolio loss that falls short of the Historical VaR estimate.

Step 1: Identify Historical VaR Limit for Portfolio Let us first create variables that hold the historical 1 and 5 % VaR estimates.

```
> VaR01.hist=-quantile(-sim.portPnL$Port.PnL,0.99)
> VaR01.hist
      99%
-39743.93
>
> VaR05.hist=-quantile(-sim.portPnL$Port.PnL,0.95)
> VaR05.hist
      95%
-23556.31
```

Step 2: Identify Simulated Portfolio Losses in Excess of VaR   To calculate ES, we have to find the average of the losses exceeding a loss of \$ 39,744 for the 1 % 1-Day Historical ES and \$ 23,556 for the 5 % 1-Day Historical ES. As such, the next step would be for us to use the P&L of our simulated portfolio to identify those losses. To do this, we use dummy variables to indicate what returns fall short of the threshold for 1 and 5 % Historical ES.

```
> ES.PnL<-sim.portPnL$Port.PnL
> ES.PnL[c(1:3,nrow(ES.PnL)),]
           Port.PnL
2011-01-03 16400.327
2011-01-04  1797.429
2011-01-05  6911.659
2013-12-31 11865.787
>
> ES.PnL$dummy01<-ifelse(ES.PnL$Port.PnL<VaR01.hist,1,0)
> ES.PnL$dummy05<-ifelse(ES.PnL$Port.PnL<VaR05.hist,1,0)
> ES.PnL[c(1:3,nrow(ES.PnL)),]
           Port.PnL dummy01 dummy05
2011-01-03 16400.327       0       0
2011-01-04  1797.429       0       0
2011-01-05  6911.659       0       0
2013-12-31 11865.787       0       0
```

Step 3: Extract Portfolio Losses in Excess of VaR   Now that we have dummy variables that identify the days on which the P&L falls short of the Historical VaR estimate, we then extract those observations into a data object for the 1 % Historical ES `shortfall01` and 5 % Historical ES `shortfall05`.

```
> shortfall01<-subset(ES.PnL,ES.PnL$dummy01==1)
> head(shortfall01)
           Port.PnL dummy01 dummy05
2011-01-28 -48438.67       1       1
2011-08-04 -55908.08       1       1
2011-08-08 -51547.54       1       1
2011-08-10 -60356.44       1       1
2011-08-18 -68178.87       1       1
2011-10-26 -62135.91       1       1
>
> shortfall05<-subset(ES.PnL,ES.PnL$dummy05==1)
> head(shortfall05)
           Port.PnL dummy01 dummy05
2011-01-28 -48438.67       1       1
2011-02-22 -33172.07       0       1
2011-03-10 -27224.22       0       1
2011-03-16 -27157.38       0       1
2011-05-16 -35772.66       0       1
2011-06-01 -25230.50       0       1
```

Step 4: Compute Average of Losses in Excess of VaR   To get the ES, we take the average of the returns is each of these data objects. We find that the 1 % 1-Day Historical ES is equal to \$ 54,216. This result means that we have a 1 % probability

that the loss in our portfolio would exceed the VaR, but, when it does, we expect that, on average, we would lose $ 54,216. Similarly, the 5 % 1-Day Historical ES is equal to $ 35,150. This result means that we have a 5 % probability that the loss in our portfolio would exceed the VaR, but, when it does, we expect that, on average, we would lose $ 35,150.

```
> avg.ES01<--mean(shortfall01$Port.PnL)
> avg.ES01
[1] 54215.51
> ES01.Historical<-format(avg.ES01,big.mark=',')
> ES01.Historical
[1] "54,215.51"
>
> avg.ES05<--mean(shortfall05$Port.PnL)
> avg.ES05
[1] 35149.55
> ES05.Historical<-format(avg.ES05,big.mark=',')
> ES05.Historical
[1] "35,149.55"
```

4.5.3 Comparing VaR and ES

We expect that the ES values to be bigger losses than the VaR because, by definition, the ES is the mean or average loss when losses exceed VaR. This can easily be seen when we combine the results into one table.

We use a combination of the `rbind` and `cbind` commands to bring together all the resulting calculations of Historical and Gaussian VaR and ES. The intermediate results looks quite messy with variable names and row labels that are not too meaningful.

```
> VaR.ES.Combined<-data.frame(rbind(
+    cbind(VaR01.Historical,ES01.Historical[1],
+       VaR01.Gaussian,ES01.Gaussian[1]),
+    cbind(VaR05.Historical,ES05.Historical[1],
+       VaR05.Gaussian,ES05.Gaussian[1])))
> VaR.ES.Combined
    VaR01.Historical      V2 VaR01.Gaussian       V4
99%       39,743.93 54,215.51     29,544.88 36,474.94
95%       23,556.31 35,149.55     20,531.24 28,506.09
```

As such, we rename the variable names to make them easy to understand and consistent. We also rename the row names to conform to the convention of using significance levels and, to avoid confusion, include the time horizonin our VaR and ES calculations.

```
> names(VaR.ES.Combined)<-paste(
+    c("VaR Historical","ES Historical","VaR Gaussian","ES Gaussian"))
> rownames(VaR.ES.Combined)<-paste(c("1% 1-Day","5% 1-Day"))
> VaR.ES.Combined
         VaR Historical ES Historical VaR Gaussian ES Gaussian
1% 1-Day       39,743.93      54,215.51     29,544.88    36,474.94
5% 1-Day       23,556.31      35,149.55     20,531.24    28,506.09
```

Looking at the output above, we can easily confirm that the ES is larger than the VaR, which is almost always the case.

4.6 Alternative Risk Measures

In this section, we will discuss alternative risk measures beyond close-to-close volatility. The advantage of using close-to-close volatility is that it only requires you to look at closing prices, but you have to use many observations to get a good estimate of volatility. Using a large number of observations entails getting a long historical series. The earlier part of such long historical data may be less relevant to measure volatility today. Therefore, we may want to consider alternative measures of volatility that are more efficient than close-to-close volatility by utilizing the open, high, and low prices of the day in addition to the closing price. The measures we will discuss are the Parkinson, Garmann-Klass, Rogers-Satchell-Yoon, and Yang-Zhou. Parkinson uses high and low prices, while the remaining alternative volatility measures all use open, high, low, and close data.

4.6.1 Parkinson

The Parkinson volatility measure uses the stock's high and low price of the day. Specifically,

$$\sigma_{\text{Parkinson}} = \sqrt{\frac{1}{4T \ln 2} \sum_{t=1}^{T} \ln\left(\frac{h_t}{l_t}\right)^2}, \tag{4.7}$$

where T is the number of days in the sample period, h_t is the high price on day t, and l_T is the low price on day t.

We now demonstrate how to implement this in R. We apply this to the Amazon.com data we have been using as our example in this book.

Step 1: Import Amazon Data from Yahoo Finance   We use the same methodology we applied in earlier examples to obtain the Amazon data.

```
> 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),]
> library(xts)
> 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)),]
           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: Keep the High and Low Price   We keep Column 2 and 3 in `data.AMZN`. Because we are not calculating returns, we also delete the December 31, 2010 observation.

```
> parkinson<-data.AMZN[-1,2:3]
> parkinson[c(1:3,nrow(parkinson)),]
           AMZN.High AMZN.Low
2011-01-03    186.00   181.21
2011-01-04    187.70   183.78
2011-01-05    187.45   184.07
2013-12-31    398.83   393.80
```

Step 3: Calculate the Terms in the Parkinson Formula   We first calculate the term $\ln(h_t/l_t)$ and then take the square of that value.

```
> parkinson$log.hi.low<-log(parkinson$AMZN.High/parkinson$AMZN.Low)
> parkinson$log.square<-(parkinson$log.hi.low)**2
> parkinson[c(1:3,nrow(parkinson)),]
           AMZN.High AMZN.Low log.hi.low   log.square
2011-01-03    186.00   181.21 0.02609009 0.0006806930
2011-01-04    187.70   183.78 0.02110555 0.0004454444
2011-01-05    187.45   184.07 0.01819602 0.0003310953
2013-12-31    398.83   393.80 0.01269209 0.0001610893
```

Step 4: Calculate the Sum of the Values Under the `log.square` Column   Using the `sum` command, we calculate the sum of the `log.square` values.

```
> parkinson.sum<-sum(parkinson$log.square)
> parkinson.sum
[1] 0.5612609
```

Step 5: Calculate the Daily Parkinson Volatility Measure We can now combine
the various components and calculate the daily Parkinson volatility measure.

```
> parkinson.vol<-sqrt(1/(4*nrow(parkinson)*log(2))*parkinson.sum)
> parkinson.vol
[1] 0.01638528
```

Step 6: Calculate the Annualized Parkinson Volatility Using the square root of
time rule, we convert daily volatility into annual volatility by multiplying the daily
volatility by the square root of 252. The output below shows that the annualized
Parkinson volatility is 26.0 %.

```
> annual.parkinson.vol<-parkinson.vol*sqrt(252)
> annual.parkinson.vol
[1] 0.2601083
```

4.6.2   Garman-Klass

The Garman-Klass volatility measure can be viewed as an extension of the Parkin-
son volatility measure that includes opening and closing prices. The Garman-Klass
volatility is calculated as follows:

$$\sigma_{\text{Garman-Klass}} = \sqrt{\frac{1}{2T} \sum_{t=1}^{T} \ln\left(\frac{h_t}{l_t}\right)^2 - \frac{2\ln 2 - 1}{T} \ln\left(\frac{c_t}{o_t}\right)^2}, \qquad (4.8)$$

where o_t is the open price on day t and all other terms are defined in the same manner
as in Eq. (4.7). We now demonstrate how to calculate the Garman-Klass volatility
measure using our Amazon data.

Step 1: Import Amazon Open, High, Low, and Close Price Data Since we
already imported the data when we calculated the Parkinson risk measure, we can
simply extract the appropriate columns from `data.AMZN`. In addition, since we are
not calculating returns, we can delete the December 31, 2010 observation.

```
> garman.klass<-data.AMZN[-1,1:4]
> garman.klass[c(1:3,nrow(garman.klass)),]
           AMZN.Open AMZN.High AMZN.Low AMZN.Close
2011-01-03    181.37    186.00   181.21     184.22
2011-01-04    186.15    187.70   183.78     185.01
2011-01-05    184.10    187.45   184.07     187.42
2013-12-31    394.58    398.83   393.80     398.79
```

Step 2: Calculate the First Term We calculate the first term inside the square root
in Eq. (4.8). Notice also that the summation term is equal to the `parkinson.sum`
we calcualted above. Using that same variable can help us save some time coding.

```
> garman.klass.one<-(1/(2*nrow(garman.klass)))*parkinson.sum
> garman.klass.one
[1] 0.0003721889
```

Step 3: Calculate the Second Term We now calculate the second term inside the square root in Eq. (4.8).

```
> garman.klass.two<-((2*log(2)-1)/nrow(garman.klass))*
+    sum(log(garman.klass$AMZN.Close/garman.klass$AMZN.Open)**2)
> garman.klass.two
[1] 0.0001009363
```

Step 4: Calculate the Daily Garman-Klass Volatility Subtracting the second term from the first and taking the square root of the difference gives us the daily Garman-Klass risk measure.

```
> garman.klass.vol<-sqrt(garman.klass.one-garman.klass.two)
> garman.klass.vol
[1] 0.01646975
```

Step 5: Annualize the Volatility To get the annualized volatility, we multiply the daily volatility by the square root of 252. The output shows that the Garman-Klass volatility is equal to 26.1 %.

```
> annual.garman.klass.vol<-garman.klass.vol*sqrt(252)
> annual.garman.klass.vol
[1] 0.2614491
```

4.6.3   Rogers, Satchell, and Yoon

The prior risk measures we have discussed all assume that the mean return is zero. In contrast, the Rogers, Satchell, and Yoon (RSY) volatility properly measures the volatility of securities with an average return that is not zero. The equation for the RSY volatility is:

$$\sigma_{\text{RSY}} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(\ln\left(\frac{h_t}{c_t}\right)\ln\left(\frac{h_t}{o_t}\right) + \ln\left(\frac{l_t}{c_t}\right)\ln\left(\frac{l_t}{o_t}\right)\right)},\qquad(4.9)$$

where all the variables are defined in the same way as they were in the Parkinson and Garman-Klass formulas. We now demonstrate how to implement the RSY volatility measure on our Amazon.com data.

Step 1: Obtain Open, High, Low, and Close Data This is similar to Step 1 of the Garman-Klass procedure.

```
> rsy.vol<-data.AMZN[-1,1:4]
> rsy.vol[c(1:3,nrow(rsy.vol)),]
           AMZN.Open AMZN.High AMZN.Low AMZN.Close
2011-01-03    181.37    186.00   181.21     184.22
2011-01-04    186.15    187.70   183.78     185.01
2011-01-05    184.10    187.45   184.07     187.42
2013-12-31    394.58    398.83   393.80     398.79
```

Step 2: Calculate the Product of First Two Log Terms We first calculate each of the log terms (`rsy.one` and `rsy.two`) and then take their product (`rsy.one.two`).

```
> rsy.one<-log(rsy.vol$AMZN.High/rsy.vol$AMZN.Close)
> rsy.one[c(1:3,nrow(rsy.one)),]
             AMZN.High
2011-01-03 0.0096159782
2011-01-04 0.0144350659
2011-01-05 0.0001600555
2013-12-31 0.0001002984
>
> rsy.two<-log(rsy.vol$AMZN.High/rsy.vol$AMZN.Open)
> rsy.two[c(1:3,nrow(rsy.two)),]
             AMZN.High
2011-01-03 0.025207530
2011-01-04 0.008292143
2011-01-05 0.018033055
2013-12-31 0.010713353
>
> rsy.one.two<-rsy.one*rsy.two
> rsy.one.two[c(1:3,nrow(rsy.one.two)),]
                AMZN.High
2011-01-03 0.000242395059
2011-01-04 0.000119697635
2011-01-05 0.000002886289
2013-12-31 0.000001074532
```

Step 3: Calculate the Product of Last Two Log Terms Similar to Step 2, we first calculate each of the log terms (`rsy.three` and `rsy.four`) and then take their product (`rsy.three.four`).

```
> rsy.three<-log(rsy.vol$AMZN.Low/rsy.vol$AMZN.Close)
> rsy.three[c(1:3,nrow(rsy.three)),]
               AMZN.Low
2011-01-03 -0.016474116
2011-01-04 -0.006670488
2011-01-05 -0.018035968
2013-12-31 -0.012591796
>
> rsy.four<-log(rsy.vol$AMZN.Low/rsy.vol$AMZN.Open)
> rsy.four[c(1:3,nrow(rsy.four)),]
                AMZN.Low
2011-01-03 -0.0008825639
2011-01-04 -0.0128134102
2011-01-05 -0.0001629682
2013-12-31 -0.0019787419
>
> rsy.three.four<-rsy.three*rsy.four
> rsy.three.four[c(1:3,nrow(rsy.three.four)),]
                 AMZN.Low
2011-01-03 0.000014539460
2011-01-04 0.000085471694
2011-01-05 0.000002939289
2013-12-31 0.000024915915
```

Step 4: Calculate the RSY Volatility Measure We bring together all the terms and calculate the annualized risk measure.

```
> rsy.vol<-sqrt((1/nrow(rsy.vol))*sum((rsy.one.two+rsy.three.four)))
> rsy.vol
[1] 0.01642503
```

Step 5: Annualize the RSY Volatility Measure We multiply the daily RSY volatility by the square root of 252. As the output shows, the RSY Volatility Measure also shows an annualized volatility of 26.1 %.

```
> annual.rsy.vol<-rsy.vol*sqrt(252)
> annual.rsy.vol
[1] 0.2607392
```

4.6.4 Yang and Zhang

Yang and Zhang developed a volatility measure that handles both opening jumps and drift. We can think of the Yang-Zhang volatility as the sum of the overnight (i.e., volatility from the prior day's close to today's open) and a weighted average of the RSY Volatility and the day's open-to-close volatility. The following Yang-Zhang volatility formula looks slightly more involved than the others, but becomes clearer when we break the calculations dows into pieces.

$$\sigma_{\text{Yang}-\text{Zhang}} = \sqrt{\sigma_{\text{overnightvol}}^2 + k\sigma_{\text{open}-\text{to}-\text{closevol}}^2 + (1-k)\sigma_{\text{RSY}}^2}, \qquad (4.10)$$

where

$$\sigma_{\text{overnight vol}}^2 = \frac{1}{T-1} \sum_{t=1}^{T} \left(\ln\left(\frac{o_t}{c_{t-1}}\right) - \text{Avg}\ln\left(\frac{o_t}{c_{t-1}}\right) \right)^2,$$

$$\sigma_{\text{open}-\text{to}-\text{close vol}}^2 = \frac{1}{T-1} \sum_{t=1}^{T} \left(\ln\left(\frac{c_t}{o_t}\right) - \text{Avg}\ln\left(\frac{c_t}{o_t}\right) \right)^2, \text{ and}$$

$$k = \frac{\alpha - 1}{\alpha + \frac{T+1}{T-1}}.$$

Below we demonstrate how to calculate the Yang-Zhang volatility measure on Amazon's stock data.

Step 1: Import Amazon Open, High, Low, and Close Data and Create Variable for Yesterday's Closing Price We import Columns 1 to 4 from `data.AMZN`. The high and low price is used to compute the RSY volatility measure (i.e., part of the third term in Eq. (4.10). Since we have already calculated the RSY volatility measure previously, we can just call in `rsy.vol` instead of recalculating the number. Also,

we need the lag closing price, so we use the `Lag` command to construct a variable `Lag.Close` that takes on the prior day's `AMZN.Close` value.

```
> yz.vol<-data.AMZN[,1:4]
> yz.vol$Lag.Close<-Lag(yz.vol$AMZN.Close,k=1)
> yz.vol[c(1:3,nrow(yz.vol)),]
           AMZN.Open AMZN.High AMZN.Low AMZN.Close Lag.Close
2010-12-31    181.96    182.30   179.51     180.00        NA
2011-01-03    181.37    186.00   181.21     184.22    180.00
2011-01-04    186.15    187.70   183.78     185.01    184.22
2013-12-31    394.58    398.83   393.80     398.79    393.37
```

Step 2: Delete December 31, 2010 Data   Since we needed to use the December 31, 2010 closing price as the lag closing price on January 3, 2011, we could not delete that observation in Step 1. So, we delete that extraneous observation in this step and limit the data to only show data for the period 2011 to 2013.

```
> yz.vol<-yz.vol[-1,]
> yz.vol[c(1:3,nrow(yz.vol)),]
           AMZN.Open AMZN.High AMZN.Low AMZN.Close Lag.Close
2011-01-03    181.37    186.00   181.21     184.22    180.00
2011-01-04    186.15    187.70   183.78     185.01    184.22
2011-01-05    184.10    187.45   184.07     187.42    185.01
2013-12-31    394.58    398.83   393.80     398.79    393.37
```

Step 3: Calculate the First Term in the Yang-Zhang Equation   We first calculate the mean of the variable because we need to subtract this number from each observation Then, we calculate the entire first term.

```
> yz.one.mean<-mean(log(yz.vol$AMZN.Open/yz.vol$Lag.Close))
> yz.one.mean
[1] 0.0003841034
>
> yz.one<-1/(nrow(yz.vol)-1)*sum((
+    log(yz.vol$AMZN.Open/yz.vol$Lag.Close)-yz.one.mean)**2)
> yz.one
[1] 0.0001484233
```

Step 4: Calculate the Second Term in the Yang-Zhang Equation   We first calculate the mean of the variable because we need to subtract this number from each observation Then, we calculate the entire second term.

```
> yz.two.mean<-mean(log(yz.vol$AMZN.Close/yz.vol$AMZN.Open))
> yz.two.mean
[1] 0.0006709074
>
> yz.two<-1/(nrow(yz.vol)-1)*sum((
+   log(yz.vol$AMZN.Close/yz.vol$AMZN.Open)-yz.two.mean)**2)
> yz.two
[1] 0.00026119
```

Step 5: Calculate k In their paper, Yang and Zhang [9] suggest that the value of α in practice should be 1.34. We follow that assumption here.

```
> k=0.34/(1.34+(nrow(yz.vol)+1)/(nrow(yz.vol)-1))
> k
[1] 0.1451344
```

Step 6: Calculate the Annualized Yang-Zhang Volatility We combine the terms and calculate the annualized Yang-Zhang volatility measure.

```
> annual.yz.vol<-sqrt(yz.one+k*yz.two+(1-k)*rsy.vol^2)*sqrt(252)
> annual.yz.vol
[1] 0.3241503
```

4.6.5 Comparing the Risk Measures

We now compare the alternative risk measures to the close-to-close volatility of Amazon over the same period. Before we can compare these numbers, we have to first calculate the close-to-close volatility for this same time period.

Step 1: Calculate Amazon Returns Data We have to calculate the Amazon returns using data in `data.AMZN`.

```
> AMZN.ret<-data.AMZN[,6]
> library(quantmod)
> AMZN.ret$Return=Delt(AMZN.ret$AMZN.Adjusted)
> AMZN.ret<-AMZN.ret[-1,2]
> AMZN.ret[c(1:3,nrow(AMZN.ret)),]
                 Return
2011-01-03 0.023444444
2011-01-04 0.004288351
2011-01-05 0.013026323
2013-12-31 0.013778377
```

Step 2: Calculate Log Returns We copy the Amazon returns data into `cl2cl.ret` data object so we can add log returns to it. Since these are arithmetic returns, we need to convert them to logarithmic returns for comparability with the alternative risk measures we estimated above.

```
> cl2cl.ret<-AMZN.ret
> cl2cl.ret$logret<-log(1+cl2cl.ret$Return)
> cl2cl.ret[c(1:3,nrow(cl2cl.ret)),]
                 Return      logret
2011-01-03 0.023444444 0.023173845
2011-01-04 0.004288351 0.004279182
2011-01-05 0.013026323 0.012942210
2013-12-31 0.013778377 0.013684318
```

Step 3: Calculate Standard Deviation of the Returns Since these are close-to-close log returns, we can now calculate the standard deviation of `logret`.

```
> cl2cl.vol<-sd(cl2cl.ret$logret)
> cl2cl.vol
[1] 0.02053515
```

Step 4: Annualize the Close-to-Close Volatility We multiply the daily volatility by the square root of 252.

```
> annual.cl2cl.vol<-cl2cl.vol*sqrt(252)
> annual.cl2cl.vol
[1] 0.3259855
```

Step 5: Create Table of the Different Volatility Measures Now we can turn to comparing the different volatility measures. As the output shows, the Close-to-Close and Yang-Zhang volatility are fairly close at 32.5 % volatility for Amazon over this period, while the Parkinson, Garman-Klass, and RSY are pretty close together at approximately 26.1 % volatility over the same period.

```
> vol.measures<-rbind(annual.cl2cl.vol,annual.parkinson.vol,
+    annual.garman.klass.vol,annual.rsy.vol,annual.yz.vol)
> rownames(vol.measures)<-c("Close-to-Close","Parkinson","Garman-Klass",
+    "Rogers et al","Yang-Zhang")
> colnames(vol.measures)<-c("Volatility")
> vol.measures
               Volatility
Close-to-Close  0.3259855
Parkinson       0.2601083
Garman-Klass    0.2614491
Rogers et al    0.2607392
Yang-Zhang      0.3241503
```

4.7 Further Reading

Many investment textbooks (e.g., Bodie et al. [3] and Reilly and Brown [7]) have more detailed discussions of standard deviation/variance as well as portfolio risk.

Jorion [5] provides an excellent and readable treatment of value-at-risk. A comprehensive discussion of value-at-risk and expected shortfall, as well as the use of a banchmark VaR, can be found in Alexander [1].

Bennett and Gil [2] and Sinclair [8] have a very good discussion of various risk measures.

References

1. Alexander, C. (2009). *Value at risk models, market risk analysis* (Vol. 4). London: Wiley.
2. Bennet, C., & Gil, M. (2012). Measuring historical volatility. Santander Equity Derivatives Report.
3. Bodie, Z., Kane, A., & Marcus, A. (2012). *Essentials of investments* (9th ed.). New York: McGraw-Hill/Irwin.

4. Hodges, S., & Tompkins, R. (2002). Volatility cones and their sampling properties. *The Journal of Derivatives, 10,* 27–42.
5. Jorion, P. (2006). *Value-at-risk: The new benchmark for managing financial risk* (3rd ed.). New York: McGraw-Hill.
6. Markowitz, H. (1952). Portfolio selection. *Journal of Finance, 7,* 77–91.
7. Reilly, F., & Brown, K. (2002). *Investment analysis & portfolio management* (10th ed.). Ohio: South-Western Cengage Learning.
8. Sinclair, E. (2008). *Volatility trading*. New Jersey: Wiley.
9. Yang, D., & Zhang, Q. (2000). Drift-independent volatility estimation based on high, low, open, and close prices. *Journal of Business, 73,* 477–491.

Chapter 5
Factor Models

Factor models are used in many financial applications, such as identifying the determinants of a security's return as well as in cost of capital calculations. *Factors* help explain the variation in a security's return. The simplest factor model is one based on a single factor. The most popular of these models is the *Capital Asset Pricing Model* (CAPM). In the CAPM, only the sensitivity of the security's return to the market portfolio's return matters. The CAPM is based on a series of assumptions, which are detailed in any decent corporate finance or investments textbook. However, the CAPM does not perform well in empirical testing, but, to its defense, it is unclear whether such poor performance is really a failure of the CAPM or the failure to use the "true" market portfolio of all available assets in these tests (i.e., this is known as *Roll's Critique*).

In many empirical tests, especially for practical applications, a simpler version of the CAPM is often used. This is known as the *market model*. Like the CAPM, the market model is also a single factor model but, this time, there is no imposition as to what kind of market proxy should be used. Hence, any broad-based stock market index is often used, such as the S&P 500 Index, MSCI World Index, etc.

An alternative to the CAPM, which is a single factor model, are *multi-factor models*. These models including additional factors that help explain more of the variation in expected stock returns. The most common multi-factor model in finance is the Three Factor Model developed by Eugene Fama and Kenneth French (FF Model). In addition to the market's return, the FF Model also includes as a factor the difference in returns between small and large capitalization stocks and the difference in returns of high and low book-to-market (i.e., value and growth stocks) stocks. The FF Model is based on statistical results, but is now becoming the prevalent factor model used in academia.

5.1 CAPM

Due to its ease of implementation, the most commonly-applied factor model is the Capital Asset Pricing Model (CAPM) by Sharpe [12]. At least the starting point of many cost of capital calculations is the CAPM, although adjustments may be made

to account for the size premium, country risk premium, and other premiums some analysts believe may not be captured by the CAPM. Therefore, regardless of the empirical findings against the CAPM, it is a worthwhile exercise to know how to implement the CAPM.

The value of a company's stock can be characterized as the present value of the stream of dividend payments shareholders expect to receive from holding the stock. Since these dividend payments are expected to arrive at different points in time in the future, we have to discount those dividend payments by an appropriate risk-adjusted discount rate. The CAPM gives us the appropriate risk-adjusted discount rate to use. From a mathematical perspective, the formula for the CAPM is:

$$r_i = r_f + \beta_i(r_m - r_f), \tag{5.1}$$

where r_i is the return on asset i, r_f is the return on the risk-free asset, r_m is the return on the market proxy, and β_i is the sensitivity of asset i to the overall market.

CAPM Regression Although the CAPM equation can often be expressed as Eq. (5.1), empirical tests of the CAPM typically convert the formula into its *excess return form*. The *excess* pertains to the return over the risk-free rate of both the subject security's return and the market return. That is, the CAPM in excess return form is as follows:

$$r_i - r_f = \alpha + \beta_i(r_m - r_f), \tag{5.2}$$

where all the variables are defined in the same way as in Eq. (5.1). The α and β_i in Eq. (5.2) are estimated using an OLS regression.

To perform the CAPM regression, we need to choose (i) the length of the estimation period, (ii) the frequency of the returns data, and (iii) the risk-free rate used in the calculation. Technically, this gives us a large number of possibilities and could potentially lead to a large number of variations in the results. In our implementation, for (i) and (ii), we follow the methodology described on the Morningstar and Yahoo Finance websites, which is to use 3 years of monthly returns or 36 monthly return observations for both the subject security's return and market return.[1] For (iii), we use the 3-Month Treasury Bill rate following Morningstar's description.

Step 1: Import Portfolio Returns and Convert to a `data.frame` **Object** In practice, we will likely have kept track of our portfolio's return and for implementing the CAPM we would then import that data. To mimic this for our example, we constructed a hypothetical portfolio and saved the monthly returns of that portfolio as **Hypothetical Portfolio (monthly).csv**. The details of how this portfolio is constructed is described in Appendix B. We import the CSV file containing our portfolio returns using the `read.csv` command.

---

[1] Morningstar Report: Mutual Fund Data Definitions, Ratings and Risk, retrieved from http://quicktake.morningstar.com/DataDefs/FundRatingsAndRisk.html on January 5, 2014 and Yahoo Finance, Key statistics definitions (Trading information/Stock Price History).

```
> port<-read.csv("Hypothetical Portfolio (Monthly).csv")
> port[c(1:3,nrow(port)),]
    X    date     EW.ret
1   1 Jan 2011 0.005211301
2   2 Feb 2011 0.014024705
3   3 Mar 2011 0.021291224
36 36 Dec 2013 0.050203327
```

When we import the data, the `date` variable is read-in by R as a `Factor`. To be able to properly work with the `date` variable, it is better to convert the values to a class that is compatible to be read-in as a date. Since we are only using month and year, we use the `yearmon` class, which we can apply by using the `as.yearmon` command. Note the `%b  %Y` format. `%b` tells R the format being read has a three letter month and the `%Y` tells R the year is a four-digit year. Details on the different date formats are found in Appendix A.

```
> class(port$date)
[1] "factor"
>
> port$date<-as.yearmon(as.character(port$date),"%b %Y")
> port[c(1:3,nrow(port)),]
    X    date     EW.ret
1   1 Jan 2011 0.005211301
2   2 Feb 2011 0.014024705
3   3 Mar 2011 0.021291224
36 36 Dec 2013 0.050203327
>
> class(port$date)
[1] "yearmon"
```

We then convert the data to a `data.frame` object using the `data.frame` command. The `data.frame` class is simply more flexible for performing a number of calculations we want to implement.

```
> port.df<-data.frame(port)
> port.df[c(1:3,nrow(port.df)),]
    X    date     EW.ret
1   1 Jan 2011 0.005211301
2   2 Feb 2011 0.014024705
3   3 Mar 2011 0.021291224
36 36 Dec 2013 0.050203327
```

Step 2: Import S&P 500 Index Data from Yahoo Finance and Calculate Monthly Market Returns Now that we have our portfolio returns, we need to calculate the monthly return for our market proxy. A typical choice for the market proxy is S&P 500 Index with ticker ^GSPC in Yahoo Finance. We upload the **GSPC Yahoo.csv** file and use a similar technique as described in Chap. 2 to calculate the monthly market return.

```
> 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.mkt<-xts(data.GSPC[,2:7],order.by=data.GSPC[,1])
> names(data.mkt)[1:6]<-
+    paste(c("GSPC.Open","GSPC.High","GSPC.Low",
+    "GSPC.Close","GSPC.Volume","GSPC.Adjusted"))
> data.mkt[c(1:3,nrow(data.mkt)),]
          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
> mkt.monthly<-to.monthly(data.mkt)
> mkt.monthly[c(1:3,nrow(mkt.monthly)),]
         data.mkt.Open data.mkt.High data.mkt.Low data.mkt.Close
Dec 2010       1256.76       1259.34      1254.19        1257.64
Jan 2011       1257.62       1302.67      1257.62        1286.12
Feb 2011       1289.14       1344.07      1289.14        1327.22
Dec 2013       1806.55       1849.44      1767.99        1848.36
         data.mkt.Volume data.mkt.Adjusted
Dec 2010      1799770000           1257.64
Jan 2011     92164940000           1286.12
Feb 2011     60027800000           1327.22
Dec 2013     62664960000           1848.36
Warning message:
timezone of object (UTC) is different than current timezone ().
> mkt.monthly<-mkt.monthly[,6]
> mkt.ret<-Delt(mkt.monthly$data.mkt.Adjusted)
> names(mkt.ret)<-paste("mkt.ret")
> mkt.ret[c(1:3,nrow(mkt.ret)),]
            mkt.ret
Dec 2010         NA
Jan 2011 0.02264559
Feb 2011 0.03195658
Dec 2013 0.02356283
Warning message:
timezone of object (UTC) is different than current timezone ().
>
> mkt.ret<-mkt.ret[-1,]
> mkt.ret[c(1:3,nrow(mkt.ret)),]
              mkt.ret
Jan 2011  0.022645590
Feb 2011  0.031956583
Mar 2011 -0.001047302
Dec 2013  0.023562833
Warning message:
timezone of object (UTC) is different than current timezone ().
```

We then convert `mkt.ret` into a `data.frame` to make it compatible with the portfolio return data we calculated in Step 1.

```
> market.df<-data.frame(mkt.ret)
> head(market.df)
               mkt.ret
Jan 2011  0.022645590
Feb 2011  0.031956583
Mar 2011 -0.001047302
Apr 2011  0.028495358
May 2011 -0.013500928
Jun 2011 -0.018257508
```

Step 3: Import Risk-Free Rate Data from FRED and Setup Data to Contain Monthly Risk-Free Returns The last component we need to implement the CAPM is the risk-free rate. We use the 3-Month Constant Maturity Treasury rate. We obtain this directly from the Federal Reserve Electronic Database (FRED). FRED allows us to download an Excel file with one worksheet that contains the data for the security we want. Since it is easier to import a CSV file, we open the file in Excel and save the lone worksheet to the R working directory as a CSV file labeled **DGS3MO FRED.csv**.

```
> rf<-read.csv("DGS3MO FRED.csv",skip=10)
> rf[1:3,]
  observation_date DGS3MO
1       1982-01-04  11.87
2       1982-01-05  12.20
3       1982-01-06  12.16
```

The observation_date variable is a Factor. So, we create a new date variable that is read in by R as a Date. The values for DGS3MO are read-in by R as characters, so we use a combination of the as.numeric and as.character commands to convert that data into numeric. Notice that a warning message appears that says "NAs introduced by coercion" after we convert the 3-Month Treasury yields to numeric. That is because there are "#N/A" values in the data, which are considered characters, and R had to convert those to numeric "NA" values when we converted to numeric variables.

```
> rf$date<-as.Date(rf$observation_date,"%Y-%m-%d")
> rf$DGS3MO<-as.numeric(as.character(rf$DGS3MO))
Warning message:
NAs introduced by coercion
> rf[c(1:3,nrow(rf)),]
     observation_date DGS3MO       date
1          1982-01-04  11.87 1982-01-04
2          1982-01-05  12.20 1982-01-05
3          1982-01-06  12.16 1982-01-06
8350       2014-01-03   0.07 2014-01-03
> str(rf)
'data.frame':   8350 obs. of  3 variables:
 $ observation_date: Factor w/ 8350 levels "1982-01-04","1982-01-05",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ DGS3MO          : num  11.9 12.2 12.2 12.2 12 ...
 $ date            : Date, format: "1982-01-04" "1982-01-05" ...
```

Let us take a moment to think about what we want the final data object to look like. What we want is to have a data object that contains monthly risk-free rates of return. We now think about what data we currently have. So, we now have annualized 3-Month Treasury yields on a daily basis. We can then take the annualized yield at the start of the month and convert that to a monthly yield and assume that is the monthly risk-free rate. To implement this, we use the following approach:

Step 3a: Convert to `xts` Object   When the data is an `xts` object, we can use the `to.monthly` command to convert the daily data into monthly data.

```
> rf<-xts(rf$DGS3MO,order.by=rf$date)
> rf[1:3,]
            [,1]
1982-01-04 11.87
1982-01-05 12.20
1982-01-06 12.16
>
> names(rf)<-paste("DGS3MO")
> rf[1:3,]
           DGS3MO
1982-01-04  11.87
1982-01-05  12.20
1982-01-06  12.16
```

Step 3b: Apply `to.monthly` Command to Identify First Yield for Each Month
Now that the data is an `xts` object, we can apply the `to.monthly` command which gives us the opening yield for each month, among other things.

```
> rf.monthly<-to.monthly(rf)
Warning message:
In to.period(x, "months", indexAt = indexAt, name = name, ...) :
  missing values removed from data
> rf.monthly[1:3,]
         rf.Open rf.High rf.Low rf.Close
Jan 1982   11.87   14.06  11.87    13.08
Feb 1982   14.77   15.49  12.79    13.00
Mar 1982   12.81   14.16  12.68    13.99
Warning message:
timezone of object (UTC) is different than current timezone ().
```

Step 3c: Convert Opening Annualized Yield for Each Month Into a Monthly Yield   The yields reported above are annualized yields. Since we are interested in only a monthly yield, we take the geometric average of the annualized yield to arrive at the monthly yield. Since the monthly yields are going to be small, we use the `options(scipen="100")` option to force R not to convert the values into scientific notation.

```
> options(scipen="100")
> rf.monthly<-(1+rf.monthly[,1]/100)^(1/12)-1
> rf.monthly[c(1:3,nrow(rf.monthly)),]
               rf.Open
Jan 1982 0.00939109694
Feb 1982 0.01154614300
Mar 1982 0.01009518279
Jan 2014 0.00005831463

Warning message:
timezone of object (UTC) is different than current timezone ().
```

Step 3d: Subset Data to January 2011 Through December 2013 The last step for the risk-free rate is to create data for the sub-period we are interested in. Since these yields can be thought in a similar light as returns, we only need to keep the monthly values of the yields during the relevant period. As such, we only need data from January 2011 through December 2013.

```
> rf.sub<-subset(rf.monthly,
+    index(rf.monthly) >= as.yearmon("Jan 2011") &
+    index(rf.monthly) <= as.yearmon("Dec 2013"))
> rf.sub[c(1:3,nrow(rf.sub)),]
               rf.Open
Jan 2011 0.00012491414
Feb 2011 0.00012491414
Mar 2011 0.00011659187
Dec 2013 0.00004165712
Warning message:
timezone of object (UTC) is different than current timezone ().
```

Step 4: Combine Firm, Market, and Risk-Free Data Into One Data Object We then combine the three series using the cbind command.

```
> combo<-cbind(market.df,data.frame(rf.sub),port.df$port.ret)
> combo[c(1:3,nrow(combo)),]
              mkt.ret       rf.Open port.df$port.ret
Jan 2011   0.022645590 0.00012491414      0.005211301
Feb 2011   0.031956583 0.00012491414      0.014024705
Mar 2011 -0.001047302 0.00011659187      0.021291224
Dec 2013   0.023562833 0.00004165712      0.050203327
```

We then rename the variables to make them more meaningful.

```
> names(combo)<-paste(c("mkt.ret","rf","port.ret"))
> combo[c(1:3,nrow(combo)),]
              mkt.ret            rf    port.ret
Jan 2011   0.022645590 0.00012491414 0.005211301
Feb 2011   0.031956583 0.00012491414 0.014024705
Mar 2011 -0.001047302 0.00011659187 0.021291224
Dec 2013   0.023562833 0.00004165712 0.050203327
```

Step 5: Calculate Excess Firm Return and Excess Market Return From the data above, we need to calculate two variables that we will use in the CAPM regression. The first is the excess firm return or $r_i - r_f$ (exret) and the excess market return or $r_i - r_m$ (exmkt).

```
> combo$exret<-combo$port.ret-combo$rf
> combo$exmkt<-combo$mkt.ret-combo$rf
> combo[c(1:3,nrow(combo)),]
               mkt.ret           rf     port.ret        exret         exmkt
Jan 2011  0.022645590 0.00012491414 0.005211301 0.005086387  0.022520676
Feb 2011  0.031956583 0.00012491414 0.014024705 0.013899791  0.031831668
Mar 2011 -0.001047302 0.00011659187 0.021291224 0.021174632 -0.001163894
Dec 2013  0.023562833 0.00004165712 0.050203327 0.050161670  0.023521176
```

Step 6: Run Regression of Excess Firm Return on Excess Market Return We now use the lm command to perform an OLS regression of the form in Eq. (5.2).

```
> options(digits=3)
> CAPM<-lm(combo$exret~combo$exmkt)
> summary(CAPM)

Call:
lm(formula = combo$exret ~ combo$exmkt)

Residuals:
     Min       1Q   Median       3Q      Max
-0.07352 -0.02361 -0.00116  0.02435  0.08262

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.01187    0.00583    2.04     0.05 *
combo$exmkt  0.76437    0.16041    4.77 0.000035 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0333 on 34 degrees of freedom
Multiple R-squared:   0.4,     Adjusted R-squared:  0.383
F-statistic: 22.7 on 1 and 34 DF,  p-value: 0.0000346
```

CAPM Alpha When making investments into a fund, investors often consider the contribution of the fund manager to the performance of the fund. As such, we need a way to measure whether a fund manager provided value that goes beyond simply investing in the index (i.e., is there a benefit to active management by the manager instead of simply using a passive investment strategy). An alternative way to look at this problem is we want to know how well did the fund manager do compared to her benchmark?

To answer this question, we look at the *alpha* of the manager as a measure this outperformance. The alpha of a portfolio can be calculated by using the excess return

form of the CAPM. The regression controls for the sensitivity of the portfolio's return to its benchmark. Therefore, any return that is not captured by the market can be attributed to the manager. This attribution can be observed through the *alpha* or intercept term of the regression. If this *alpha* is positive and statistically significant, the manager is taken to have provided positive value. Conversely, a negative and statistically significant *alpha* is taken to mean that the manager has provided negative value (i.e., we would have been better off investing in the benchmark).

The *alpha* of the portfolio is the intercept term from the regression output above. The intercept is equal to 0.01187, which translates to a monthly return of 1.2 %. This alpha is statistically significant at the 5 % level, which is a conventional level of significance used in practice. As such, assuming our hypothetical portfolio returns, the manager provided an incremental return of 1.2 % per month beyond that of what is expected from its sensitivity to the benchmark.

CAPM Beta Another important measure we can derive from the results of the CAPM regression is the portfolio's *beta*. The beta of a portfolio measures how sensitive the portfolio's return is to the movement of the overall market. Therefore, the beta of the portfolio measures what is called *systematic risk* or *market risk*. Systematic risk is the portion of a security's risk that cannot be diversified away and, as such, it is commonly thought of as the level of risk that investors are compensated from taking on.

From the regression results above, we can read-off what the beta for the portfolio is. This is the parameter estimate of the `combo$exmkt` variable of 0.76 with a *p*-value of 0.00. The *p*-value tells us the minimum significance level for which the beta is statistically significant. A *p*-value of 0.00 means that at any conventional level of significance (e.g., 10, 5, or 1 %), the beta of our portfolio is statistically different from 0.

An alternative way to view the beta and the *p*-value is to call them directly from the regression summary results. Instead of simply stopping at `summary(CAPM)`, we can include `$coefficient[...]` where inside the square brackets we could place one or a vector of numbers using the `c(...)` operator to output. Looking at the regression summary output above, we see the coefficients have eight values. The numbering works like this. Number 1 is the `Estimate` of the intercept, number 2 is the estimate of the excess market return (or the beta), number 3 goes back up as the `Std. Err.` of the intercept, then number 4 goes down as the `Std. Err.` of the beta, and so on. So, if we need to know what the beta and *p*-value of the beta are, we choose numbers 2 and 8.

```
> beta<-summary(CAPM)$coefficients[2]
> beta
[1] 0.764
> beta.pval<-summary(CAPM)$coefficients[8]
> beta.pval
[1] 0.0000346
```

We could have simplified the output by using `[c(2,8)]`, but calling the output separately allows to set the beta and *p*-value into separate variables in case we need them at a later time.

The results of the CAPM regression show that the CAPM beta is 0.7643. This beta of 0.7643 can then be used in the CAPM to calculate, say, the cost of equity for the company. This means that if the market goes up by 1 %, we expect our portfolio to go up by only 0.76 %. However, if the market goes down by 1 %, we expect our portfolio to only go down by 0.76 %. A beta less than one is consistent with betas of *defensive stocks* as these stocks are less affected by adverse market movements.

Using Calculated Beta to Estimate Cost of Equity Using CAPM To calculate the cost of equity using the CAPM, we need to find an Equity Risk Premium (ERP) that is compatible. For our example, we should use an ERP that is calculated using a 3-Month Treasury security. One such source for the historical ERP is Professor Damodaran's website.[2]

Professor Damodaran's calculations show an ERP of 5.38 % based on an arithmetic average of returns from 1928 to 2012 and 7.19 % based on an arithmetic average of returns from 1962 to 2012. This range of ERPs is consistent with the general range of ERPs of around 5–8 %. Given that the 3-Month Treasury as of December 31, 2013 was approximately 0.07 %, we estimate the cost of equity of our portfolio is 4.16 % [= 0.07 % + 0.76 * 5.38 %] based on the ERP from 1926 to 2012 and 5.53 % [= 0.07 % + 0.76 * 7.19 %] based on the ERP from 1962 to 2012. The low cost of equity is due to the low beta of 0.76 we estimated for our portfolio.

Calculate Adjusted Beta There have been studies that show betas that are above the market beta of one tend to go down in the long-term, while betas that are below the market beta of one tend to go up in the long-term. Since valuations take a long-term view, some believe betas used in calculating the cost of equity need to be adjusted to reflect this reversion to the market beta. One common adjustment is to apply 2/3 weight to the raw beta, which is the beta we calculated above, and 1/3 weight to the market beta of one.

```
> adj.beta<-(2/3)*beta+(1/3)*1
> adj.beta
[1] 0.843
> options(digits=7)
```

The weighting we used for the adjusted beta calculation is the standard assumption used by Bloomberg when reporting adjusted beta. Using this adjusted beta, the cost of equity equals 4.60 % [= 0.07 % + 0.89 * 5.38 %] based on the ERP from 1926 to 2012 and 6.13 % [= 0.07 % + 0.89 * 7.19 %] based on the ERP from 1962 to 2012.

The cost of equity calculated using the adjusted beta is higher than the cost of equity calculated using the raw beta. This is consistent with what we would have

---

[2] Annual Returns on Stocks, T. Bonds and T. Bills: 1928—Current, retrieved from http://pages.stern.nyu.edu/ adamodar/New_Home_Page/datafile/histretSP.html on January 5, 2014.

expected. Since our raw beta is less than one, we would expect that the adjusted beta to be higher as the adjusted beta "pulls" the raw beta upwards towards one.

5.2   Market Model

The CAPM requires that we use expected returns and the "true" market portfolio. A more common way to calculate beta in practice is to use the *market model*, because the market model uses a market proxy without the requirement that this market proxy be the "true" market portfolio. In addition, the market model does not require the use of a risk-free rate and, therefore, there is no need to calculate the excess returns of the firm and the market. That is,

$$r_i = \alpha + \beta r_m, \tag{5.3}$$

where all variables are defined the same way as in the CAPM regression.

   Since we already have all the inputs when we calculated the CAPM above, we only need to run the regression of the form in Eq. 5.3 to see our results.

```
> options(digits=3)
> reg<-lm(combo$port.ret~combo$mkt.ret)
> summary(reg)

Call:
lm(formula = combo$port.ret ~ combo$mkt.ret)

Residuals:
     Min       1Q   Median       3Q      Max
-0.07352 -0.02361 -0.00116  0.02435  0.08261

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)     0.01189    0.00584    2.04     0.05 *
combo$mkt.ret   0.76433    0.16041    4.76 0.000035 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0333 on 34 degrees of freedom
Multiple R-squared:   0.4,     Adjusted R-squared:  0.383
F-statistic: 22.7 on 1 and 34 DF,  p-value: 0.0000346
```

The beta calculated using the market model is 0.7643, which is very close to the beta calculated using the CAPM of 0.7644. The small difference is primarily driven by the low, stable risk-free rate during the estimation period.

Any inferences we made with the CAPM beta can also be applied to the market model beta. For example, the interpretation of a beta of 0.7643 is that for a 1 % increase (decrease) in the S&P 500 Index return, we expect that our portfolio would go up (down) by 0.76 %. We can also implement the adjusted beta calculation using the beta calculated from the market model. Given that the two beta calculations are so close, there is virtually no difference in the resulting adjusted beta from these two approaches to calculating beta.

```
> beta.mktmod<-summary(reg)$coefficients[2]
> beta.mktmod
[1] 0.764
>
> adj.beta.mktmod<-(2/3)*beta.mktmod+(1/3)*1
> adj.beta.mktmod
[1] 0.843
> options(digits=7)
```

Given the raw beta and adjusted beta calculated using the market model are virtually identical to the estimates using the CAPM, the cost of equity estimated from these two betas are also going to be virtually identical.

5.3   Rolling Window Regressions

The estimates of alpha and beta are sensitive to the method we use to calculate them. One such assumption is the time period over which we estimate these parameters. For example, if you are using 1 year of data, did you use data from January to December 2011, July 2011 to June 2012, January 2012 to December 2012, and so. We can see that there are many possible variations of the assumption we can choose from.

In this section, we demonstrate how to run regressions over a rolling window, so we can calculate the alphas and betas for Amazon.com over multiple periods to analyze the variation of the alpha and beta through time. In our example, we will calculate alphas and betas using regressions on 252 trading day periods from 2012 to 2013. Since our first return starts in 2011, the 252 trading day window, which is approximately one calendar year in length, will use virtually all of the returns in 2011 to generate the first estimate of alpha and beta on December 31, 2011.

Step 1: Import Amazon.com and S&P 500 Index Data   We first read in the **AMZN Yahoo.csv** file into a data object labeled `data.AMZN` and then read in the **GSPC Yahoo.csv** file into a data object labeled `data.mkt`. The S&P 500 Index is a conventional market proxy used in calculating betas in a market model.

```
> # Import AMZN Data
> 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)),]
           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
>
> # Import GSPC Data
> data.mkt<-read.csv("GSPC Yahoo.csv",header=TRUE)
> date<-as.Date(data.mkt$Date,format="%Y-%m-%d")
> data.mkt<-cbind(date, data.mkt[,-1])
> data.mkt<-data.mkt[order(data.mkt$date),]
> data.mkt<-xts(data.GSPC[,2:7],order.by=data.GSPC[,1])
> names(data.mkt)[1:6]<-
+    paste(c("GSPC.Open","GSPC.High","GSPC.Low",
+     "GSPC.Close","GSPC.Volume","GSPC.Adjusted"))
> data.mkt[c(1:3,nrow(data.mkt))]
           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
```

Step 2: Calculate the Amazon.com and Market Returns We calculate the logarithmic returns of Amazon.com and the market. We follow the same method we used in earlier chapers by first taking the log of the adjusted prices (AMZN.Adjusted and GSPC.Adjusted) and then taking the difference between successive observations.

```
> rets<-diff(log(data.AMZN$AMZN.Adjusted))
> rets$GSPC<-diff(log(data.mkt$GSPC.Adjusted))
> names(rets)[1]<-"AMZN"
> rets[c(1:3,nrow(rets)),]
                    AMZN         GSPC
2010-12-31            NA           NA
2011-01-03 0.023173845  0.011251310
2011-01-04 0.004279182 -0.001313890
2013-12-31 0.013684318  0.003951835
>
> rets<-rets[-1,]
> rets[c(1:3,nrow(rets)),]
                    AMZN         GSPC
2011-01-03 0.023173845  0.011251310
2011-01-04 0.004279182 -0.001313890
2011-01-05 0.012942210  0.004994592
2013-12-31 0.013684318  0.003951835
```

Since the first return we can calculate is the return for January 3, 2011, the values for December 31, 2010 are labeled NA. As such, we clean-up the returns data by deleting the December 31, 2010 observation.

Step 3: Create the Rolling Window Regression Function Using the `zoo` package and the `rollapply` command, we run a regression over rolling 252 trading day windows.

```
> require(zoo)
> coeffs<-rollapply(rets,
+    width=252,
+    FUN=function(X)
+    {
+    roll.reg=lm(AMZN~GSPC,
+        data=as.data.frame(X))
+        return(roll.reg$coef)
+    },
+    by.column=FALSE)
>
> coeffs[c(1,251:253,nrow(coeffs)),]
           X.Intercept.      GSPC
2011-01-03           NA        NA
2011-12-29           NA        NA
2011-12-30 -0.0001549878 0.9577844
2012-01-03 -0.0001289250 0.9605848
2013-12-31  0.0005404550 1.2623034
```

Note that the term `return(roll.reg$coeff)` is used to output the parameter estimates (i.e., the intercept term and the coefficient for the market return) of the regression.

Step 4: Remove NAs From the Data As the output above shows, the first 251 observations of `coeffs` are NAs. This is because we need 252 observations to generate the first regression, which would be on December 30, 2011. As such, we would want to delete all those NAs.

```
> coeffs<-na.omit(coeffs)
> coeffs[c(1:3,nrow(coeffs)),]
           X.Intercept.      GSPC
2011-12-30 -0.0001549878 0.9577844
2012-01-03 -0.0001289250 0.9605848
2012-01-04 -0.0001854681 0.9606978
2013-12-31  0.0005404550 1.2623034
```

Step 5: Clean-Up Data For presentation purposes, the `coeffs` data has a single observation in 2011, contains labels that may not make sense to other people, and we may feel that there are too many decimals that are being reported. As such, we may want to delete that first observation (as it looks out of place and we do not need it), rename the variables to `Alpha` and `Beta` and use the `options(digits=3)` command to reduce the number of decimals being displayed.

```
> coeffs<-coeffs[-1,]
> names(coeffs)<-c("Alpha","Beta")
> options(digits=3)
> coeffs[c(1:3,nrow(coeffs)),]
                Alpha  Beta
2012-01-03 -0.000129 0.961
2012-01-04 -0.000185 0.961
2012-01-05 -0.000227 0.960
2013-12-31  0.000540 1.262
```

Step 6: Plot the Data Although we now have the data in tabular form, it is easier to see the changes in alpha and beta through time in a plot. Here, we stack 2 line charts one on top of the other so we can see how the alpha and beta changes over the same time period more easily.

```
> par(oma=c(0,0,4,0))
> par(mfrow=c(2,1))
> plot(x=index(coeffs),
+     xlab="Date",
+     y=coeffs$Alpha,
+     ylab="alpha",
+     type="l")
> plot(x=index(coeffs),
+     xlab="Date",
+     y=coeffs$Beta,
+     ylab="beta",
+     type="l")
> title(main="Amazon.com Inc. Alpha and Beta
+     Using Rolling 252-Day Windowns and
+     Daily Returns From 2012 to 2013",
+     outer=TRUE)
> par(mfrow=c(1,1))
```

Figure 5.1 shows the output of the above code. As we can see, Amazon.com's alpha has been fluctuating but in a pretty tight band. The beta, however, has increased from less than one in the beginning of 2012 to approximately 1.3 by the end of 2013.

5.4 Fama-French Three Factor Model

The popularity of the CAPM is due to its simplicity and it is grounded in finance theory. However, the CAPM has not performed well in empirical testing. This suggests other factors may need to be added to help explain the remaining variation in asset returns unexplained by the market. One such model that has gained popularity, especially in the academic community, is the Fama-French Three Factor (FF) Model (see Fama and French [6]). For example, Cochrane [3] points out that the FF Model has taken the place of the CAPM for routine risk adjustment in empirical work.
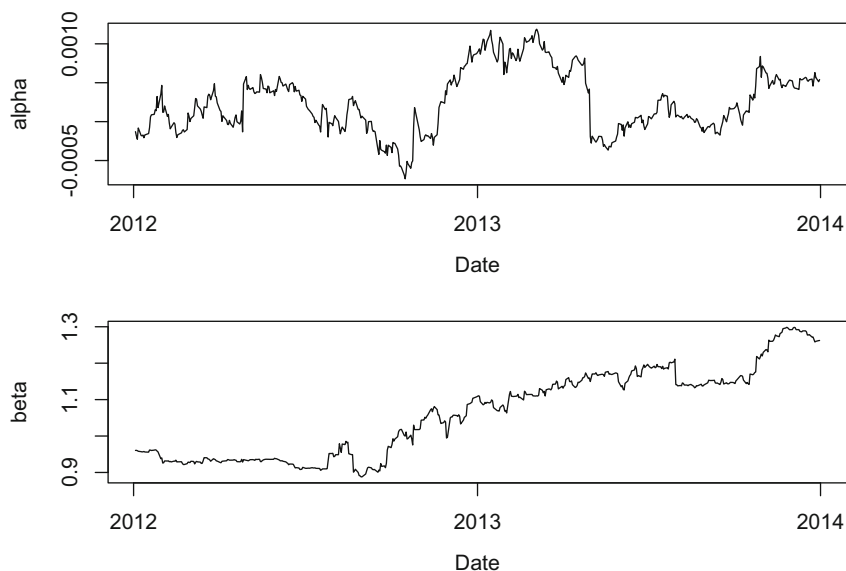
Fig. 5.1 Amazon.com alpha and beta through time, 2012–2013. Data Source: CSI www.csidata. com. Reproduced with permission of CSI ©2013

In the FF Model, we have

$$r_i = r_f + \beta_i(r_m - r_F) + h\,HML + s\,SMB, \tag{5.4}$$

where HML is the difference in the returns of portfolios with high B/M ratios and low B/M ratios and SMB is the difference in returns of portfolios of small company stocks and big company stocks. The rest of the variables are defined the same way as in Eq. (5.1).

The data needed to implement the FF Model can be downloaded from Professor Kenneth French's Data Library (U.S. Research Returns Data).[3] Note that the files on Professor French's data library are regularly updated, so downloading data at a later date results in retrieving more data albeit the start of the data should remain the same as what we report below.

The FF factors data can be download in separate ZIP files on a monthly, weekly, or daily frequency. After unpacking the ZIP file, be sure to put the text file in the R working directory so R can access it.

Step 1: Import Portfolio Returns Data We will continue to use the portfolio we constructed at the beginning of this chapter. As such, we should check to make sure the `port` data is still in the R memory.

---

[3] http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html#Research.

```
> port[c(1:3,nrow(port)),]
    X    date    port.ret
1    1 Jan 2011 0.005211301
2    2 Feb 2011 0.014024705
3    3 Mar 2011 0.021291224
36  36 Dec 2013 0.050203327
```

Step 2: Import Fama-French Data Retrieved From Ken French's Website Since
we are using our portfolio of monthly returns above, we use the FF monthly data
series. Since we follow the same technique of importing FF data in Chap. 3, we will
not repeat the details of those steps here.

```
> FF.raw<-read.fwf(file="F-F_Research_Data_Factors.txt",
+       widths=c(6,8,8,8,8),skip=4)
> head(FF.raw)
       V1      V2       V3      V4       V5
1 192607    2.95   -2.50   -2.67    0.22
2 192608    2.63   -1.20    4.50    0.25
3 192609    0.38   -1.33   -0.30    0.23
4 192610   -3.24   -0.13    0.79    0.32
5 192611    2.54   -0.24   -0.39    0.31
6 192612    2.62   -0.22   -0.11    0.28
> tail(FF.raw)
           V1      V2       V3      V4       V5
1140     2010   17.39    13.58   -3.21    0.12
1141     2011    0.47    -6.00   -6.91    0.04
1142     2012   16.29     0.44    8.09    0.06
1143     2013   35.21     7.89    0.29    0.02
1144     <NA>    <NA>     <NA>    <NA>    <NA>
1145 Copyri ght 2014  Kenneth  R. Fren       ch
>
> FF.raw<-FF.raw[-1051:-1145,]
> names(FF.raw)<-paste(c("text.date","RmxRf","SMB","HML","Rf"))
> head(FF.raw)
  text.date   RmxRf     SMB     HML      Rf
1    192607    2.95   -2.50   -2.67    0.22
2    192608    2.63   -1.20    4.50    0.25
3    192609    0.38   -1.33   -0.30    0.23
4    192610   -3.24   -0.13    0.79    0.32
5    192611    2.54   -0.24   -0.39    0.31
6    192612    2.62   -0.22   -0.11    0.28
> tail(FF.raw)
      text.date   RmxRf     SMB     HML      Rf
1045     201307    5.66    1.85    0.79    0.00
1046     201308   -2.69    0.28   -2.46    0.00
1047     201309    3.76    2.85   -1.52    0.00
1048     201310    4.17   -1.53    1.39    0.00
1049     201311    3.12    1.31   -0.38    0.00
1050     201312    2.81   -0.44   -0.17    0.00
```

When the data is read-in, all variables are considered `Factors` by R. We need to
convert all these variables into either a `Date` or `numeric`. The date variable is
harder to convert, so the easier alternative is to generate a sequence of monthly date

observations and use that new series. Once a new workable date variable is created, we can then drop the `text.date` variable.

```
> FF.raw<-FF.raw[,-1]
> FF.raw$RmxRf<-as.numeric(as.character(FF.raw$RmxRf))/100
> FF.raw$Rf<-as.numeric(as.character(FF.raw$Rf))/100
> FF.raw$SMB<-as.numeric(as.character(FF.raw$SMB))/100
> FF.raw$HML<-as.numeric(as.character(FF.raw$HML))/100
> FF.raw$FF.date<-seq(as.Date("1926-07-01"),
+    as.Date("2013-12-31"),by="months")
> FF.raw$FF.date<-as.yearmon(FF.raw$FF.date,"%Y-%m-%d")
> FF.raw[c(1:3,nrow(FF.raw)),]
      RmxRf     SMB     HML     Rf FF.date
1    0.0295 -0.0250 -0.0267 0.0022 Jul 1926
2    0.0263 -0.0120  0.0450 0.0025 Aug 1926
3    0.0038 -0.0133 -0.0030 0.0023 Sep 1926
1050 0.0281 -0.0044 -0.0017 0.0000 Dec 2013
```

Note that the `FF.date` variable says the first of the month, but that observation is really for the month of July. The sole purpose of using the first of the month is to make the creation of the date sequence simpler.

Step 3: Subset Fama-French Data to Relevant Time Period As the output above shows, the FF data begins in July 1926, but we only need monthly return data from January 2011. Therefore, we should subset the FF data to only include data from January 2011 to December 2013, which is the latest available monthly observation of FF data.

```
> FF.data<-subset(FF.raw,
+    FF.raw$FF.date>="2011-01-01" &
+    FF.raw$FF.date<="2013-12-31")
> FF.data[c(1:3,nrow(FF.data)),]
      RmxRf     SMB     HML     Rf FF.date
1015 0.0201 -0.0246  0.0086 0.0001 Jan 2011
1016 0.0349  0.0163  0.0167 0.0001 Feb 2011
1017 0.0047  0.0262 -0.0122 0.0001 Mar 2011
1050 0.0281 -0.0044 -0.0017 0.0000 Dec 2013
```

Step 4: Combine Portfolio Returns Data and Fama-French Data Next, we combine `FF.data` and `port`. Note that we use the `options` command with `digits=3` option here.

```
> options(digits=3)
> FF.data<-cbind(FF.data,data.frame(port))
> FF.data[c(1:3,nrow(FF.data)),]
      RmxRf     SMB     HML     Rf FF.date  X     date port.ret
1015 0.0201 -0.0246  0.0086 0.0001 Jan 2011  1 Jan 2011  0.00521
1016 0.0349  0.0163  0.0167 0.0001 Feb 2011  2 Feb 2011  0.01402
1017 0.0047  0.0262 -0.0122 0.0001 Mar 2011  3 Mar 2011  0.02129
1050 0.0281 -0.0044 -0.0017 0.0000 Dec 2013 36 Dec 2013  0.05020
```

Notice that the current index values begin at index number 1015 and end at index number 1050.[4] This index number is not meaningful. So, we should modify the index to signify the observation number using the `rownames` command. The new index shows that we have 36 observations, which reflects one observation each month over a 3-year period. To avoid confusion with the `date` variable, we format the `date` to only show the year and month. Lastly, we need to create an excess return variable for the portfolio return (`exret`).

```
> rownames(FF.data)<-seq(1,nrow(FF.data))
> FF.data$date<-format(FF.data$date,"%Y-%m")
> FF.data$exret<-FF.data$port.ret-FF.data$Rf
> FF.data[c(1:3,nrow(FF.data)),]
     RmxRf    SMB     HML    Rf FF.date  X    date port.ret   exret
1   0.0201 -0.0246  0.0086 0.0001 Jan 2011  1 2011-01  0.00521 0.00511
2   0.0349  0.0163  0.0167 0.0001 Feb 2011  2 2011-02  0.01402 0.01392
3   0.0047  0.0262 -0.0122 0.0001 Mar 2011  3 2011-03  0.02129 0.02119
36  0.0281 -0.0044 -0.0017 0.0000 Dec 2013 36 2013-12  0.05020 0.05020
```

Step 5: Run Regression Using Fama-French Factors Now, we can run a regression based on Eq. (5.4) to find the parameter estimates. The FF model shows that only the excess market return (`RmxRf`) is the only coefficient that is statistically significant. However, the model has an F-statistic that is highly significant.

```
> FF.reg<-lm(FF.data$exret~RmxRf+SMB+HML,data=FF.data)
> summary(FF.reg)

Call:
lm(formula = FF.data$exret ~ RmxRf + SMB + HML, data = FF.data)

Residuals:
    Min      1Q  Median      3Q     Max
-0.0650 -0.0253  0.0024  0.0212  0.0631

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.01147    0.00596    1.93   0.0630 .
RmxRf        0.66347    0.18779    3.53   0.0013 **
SMB          0.38006    0.39182    0.97   0.3393
HML         -0.45130    0.38449   -1.17   0.2492
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0327 on 32 degrees of freedom
Multiple R-squared:  0.455,    Adjusted R-squared:  0.404
F-statistic: 8.92 on 3 and 32 DF,  p-value: 0.000193
```

---

[4] We use FF data retrieved in April 2014 in this text. As discussed previously, retrieving FF data at a later date will result in different index numbers. We have to manually inspect the data either in R or, if more convenient, in Excel, so we can identify the correct rows to delete.

Compare Fama-French Results with CAPM Results We then compare the beta from the FF model with that of the CAPM. Using the data in `FF.data`, we run the CAPM regression.

```
> CAPM.reg<-lm(exret~RmxRf,data=FF.data)
> summary(CAPM.reg)

Call:
lm(formula = exret ~ RmxRf, data = FF.data)

Residuals:
     Min       1Q   Median       3Q      Max
-0.07396 -0.02264  0.00099  0.02095  0.07883

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.01061    0.00587    1.81     0.08 .
RmxRf        0.73920    0.15206    4.86 0.000026 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.033 on 34 degrees of freedom
Multiple R-squared:  0.41,     Adjusted R-squared:  0.393
F-statistic: 23.6 on 1 and 34 DF,  p-value: 0.000026
```

We can create a small summary table comparing the FF and CAPM beta, *p*-value of the beta, and adjusted R-squared. We can do this by combining the `rbind` and `cbind` commands.

```
> betas<-rbind(
+     cbind(summary(FF.reg)$coefficient[2],
+        summary(FF.reg)$coefficient[14],
+        summary(FF.reg)$adj.r.squared),
+     cbind(summary(CAPM.reg)$coefficient[2],
+        summary(CAPM.reg)$coefficient[8],
+        summary(CAPM.reg)$adj.r.squared))
> betas
      [,1]     [,2]  [,3]
[1,] 0.663 0.001274 0.404
[2,] 0.739 0.000026 0.393
>
> colnames(betas)<-paste(c("Beta","p-Value","Adj. R-Squared"))
> rownames(betas)<-paste(c("Fama-French","CAPM"))
> betas
             Beta  p-Value Adj. R-Squared
Fama-French 0.663 0.001274          0.404
CAPM        0.739 0.000026          0.393
> options(digits=7)
```

The betas and *p*-values suggest that the returns of our portfolio is sensitive to the changes in the market. The CAPM beta was low at 0.739, but the FF beta is even lower at 0.663. Since FF is a three-factor model, the calculation of the cost of equity has to be with all three factors. We cannot simply use the 0.663 beta with the ERP to calculate a cost of equity. In addition, the output shows that FF regression is a slightly better model than the CAPM in explaining the variation in our portfolio's returns based on having a higher Adjusted R-Squared.

5.5   Event Studies

Most of what we know in corporate finance has been attributed to event studies. An *event study* tests the impact of new information on a firm's stock price. As such, event studies presume that markets are semi-strong form efficient. The popularity of event studies also extends to other areas, such as the field of securities litigation in which experts are almost always expected to support their opinions on materiality of disclosed information using an event study.

The foundation of the event study is based on the Capital Asset Pricing Model (CAPM). An event study estimates the impact of a specific event on a company or a group of companies by looking at the firm's stock price performance. The event study uses the historical relationship between the firm's stock return and one or more independent variables. The independent variables are often chosen to be a proxy for the market and/or the firm's industry. Based on this historical relationship, an event study predicts a return for the company on the event date. An abnormal return is then calculated as the difference between the firm's actual return on the event date and the predicted return on the event date.

As the name suggests, an event study requires an "event" to be identified. This "event" could be something that affects a particular firm (e.g., corporate earnings announcements) or something that affects an entire industry (e.g., passing of new legislation). Moreover, some events are not reported in isolation (i.e., events are announced together with other information that could also affect the firm's stock price). Therefore, the hypothesis we test in an event study depends on the specifics of the event and a more involved analysis may be necessary to determine the effect of the particular "event" we intend to study. Put differently, we cannot simply apply a cookie-cutter approach to performing event studies, as the underlying economics may not match a purely mechanical application of event studies. This is more true for single-firm event studies, because we cannot benefit from averaging our results across multiple firms.

To conduct an event study, we need to identify two time periods. The first time period is the day or days around which the effects of an event has occurred. We call this the *event window*. The length of the event window depends on the type of event being studied. In the US, strong empirical evidence exists to show that stock prices react very quickly, usually within minutes, of material news announcements. Therefore, a one-day event window is likely sufficient when the timing of the event

can be clearly identified. In some instances, news may be leaked days or weeks prior to the official announcement. This can be the case for some mergers. In this case, the event window can be extended to longer periods to fully capture the effect of the event.

Once the event window is defined, we can then select the appropriate *estimation period* from which we estimate the parameters under "normal" conditions unaffected by the event. The estimation procedure uses a market model of the form

$$r_{i,t} = \alpha + \beta r_{m,t} + \epsilon_{i,t}, \tag{5.5}$$

where $r_{i,t}$ is the return on day t of the subject firm i and $r_{m,t}$ is the return on day t of the market proxy m. It is typical to calculate the returns in an event study using log returns, so $r_{i,t} = \ln(P_{i,t}/P_{1,t-1})$ and $r_{m,t} = \ln(P_{m,t}/P_{m,t-1})$. However, using arithmetic returns is also used in practice. Although the capital asset pricing model (CAPM) can also be used instead of Eq. (5.5), the market model has the advantage of not needing to use the "true" market portfolio and allows us to use historical returns rather than expected returns. A common choice for the market proxy is the S&P 500 Index or another broad-based index.

Note that what constitutes "normal" depends on the event being studied. For example, if we are studying earnings announcements, then placing the estimation period prior to the event window may be appropriate. However, if we are studying an event immediately after a large merger that changes the underlying economics of the firm, then placing the estimation period after the event window may be better. Further, one must also be cognizant of whether the volatility of the abnormal return is significantly different due to non-event factors. For example, we likely will find many significant days if we analyze an event during the 2008/2009 financial crisis when we choose an estimation period before or after the crisis.

Aside from the location of the estimation period, we must also determine the length of the estimation period. Typically, when daily data is used, a 6-month or 1-year period is sufficient. We use this specification. Alternatives could be weekly or monthly data over a longer date range. Note that the choice of the length of the estimation period is a trade-off between including more data with the risk of including less relevant data to predict the performance of the stock during the event window.

Since we are using statistical methods to estimate the abnormal return, not all non-zero abnormal returns should be interpreted as a reaction to the disclosed information. The reason is that we have to account for the normal volatility of the firm's stock return. For example, a highly volatile stock may move up or down 5–10 % in any given day, which means that a 10 abnormal return for that stock cannot be discerned from the normal volatility of the stock. Hence, we have to test whether the abnormal return is sufficiently large that it can be considered statistically significant. We use a t-test to perform this task, which is

$$t_{i,t} = \frac{\varepsilon_{i,t}}{\sigma}, \tag{5.6}$$

where σ is the Root Mean Square Error (RMSE) or Residual Standard Error of the market model regression in Eq. (5.5). An approach to determining statistical significance of the abnormal return is to find the corresponding p-value of the t-statistic, which is based on the number of degrees of freedom. The p-value will tells us the smallest possible significance level for which the t-statistic is considered statistically significant. In other words, for a sufficiently large number of observations, a t-statistic of 1.65 will have a 2-tail p-value of 0.10. This means that the abnormal return is statistically significant at the 10 % level. Other conventional levels of significance are equal to 1 and 5 %.

5.5.1 Example: Netflix July 2013 Earnings Announcement

After markets closed on July 22, 2013, Netflix announced earnings that more than quadrupled from a year ago but its new subscriber numbers for the second quarter of 630,000 failed to meet investors' expectations of 800,000 new subscribers.[5] Netflix stock dropped 4.5 % on July 23, 2013, the first full trading day on which investors could have reacted to Netflix's information. News reports attributed the drop to the miss in new subscriber numbers. A 4.5 % drop seems huge, but we must put this in the context of NFLX stock volatility, which may be high as its stock tripled from its price 1 year before the announcement. Therefore, we can use an event study to determine whether the price reaction on July 23, 2013 can be distinguished from normal volatility in Netflix stock.

Step 1: Identify the Event and the Event Window The event is the July 22, 2013 Netflix earnings announcement that was made after markets closed. As such, the first full trading day investors could react to this news would be on July 23, 2013. We therefore use a one-day Event Window of July 23, 2013.

Step 2: Identify the Estimation Period We use a one calendar year period ending the day before the Event Window as our Estimation Period. That is, our Estimation Period uses returns from July 23, 2012 to July 22, 2013.

Step 3: Import Netflix Data From Yahoo Finance From July 20, 2012 to July 23, 2013 To calculate a return on July 23, 2013, we would need data the trading day before. Since July 21 and 22, 2012 are non-trading days, we need the price from July 20, 2012. We save this file as **NFLX Yahoo (event study).csv**.

---

[5] MarketWatch, Netflix, Cisco fall; Apple jumps after results, July 23, 2013, retrieved from http://www.marketwatch.com/story/netflix-cisco-fall-apple-slips-ahead-of-results-2013-07-23 on January 13, 2014.

```
> data.NFLX<-read.csv("NFLX Yahoo (event study).csv",header=TRUE)
> date<-as.Date(data.NFLX$Date,format="%Y-%m-%d")
> data.NFLX<-cbind(date, data.NFLX[,-1])
> data.NFLX<-data.NFLX[order(data.NFLX$date),]
> library(xts)
> firm<-xts(data.NFLX[,2:7],order.by=data.NFLX[,1])
> firm[c(1:3,nrow(firm)),]
            Open   High    Low  Close   Volume Adj.Close
2012-07-20  82.66  83.52  81.50  81.82  3481400    81.82
2012-07-23  80.73  81.22  78.50  79.94  4423600    79.94
2012-07-24  81.07  82.47  77.81  80.39  9242900    80.39
2013-07-23 251.40 262.23 246.20 250.26 10975600   250.26
> names(firm)<-paste(c("Firm.Open","Firm.High","Firm.Low",
+   "Firm.Close","Firm.Volume","Firm.Adjusted"))
> firm[c(1:3,nrow(firm)),]
           Firm.Open Firm.High Firm.Low Firm.Close Firm.Volume Firm.Adjusted
2012-07-20     82.66     83.52    81.50      81.82     3481400         81.82
2012-07-23     80.73     81.22    78.50      79.94     4423600         79.94
2012-07-24     81.07     82.47    77.81      80.39     9242900         80.39
2013-07-23    251.40    262.23   246.20     250.26    10975600        250.26
```

Step 4: Import SPDR S&P 500 Index ETF Data From Yahoo Finance From July 22, 2012 to July 23, 2013 We use the SPDR S&P 500 Index ETF (SPY) as the market proxy.[6] Similar to the Netflix's data, we also import Yahoo Finance data from July 20, 2012 to July 23, 2013 for SPY. We save this file as **SPY Yahoo (event study).csv**.

```
> data.SPY<-read.csv("SPY Yahoo (event study).csv",header=TRUE)
> date<-as.Date(data.SPY$Date,format="%Y-%m-%d")
> data.SPY<-cbind(date, data.SPY[,-1])
> data.SPY<-data.SPY[order(data.SPY$date),]
> market<-xts(data.SPY[,2:7],order.by=data.SPY[,1])
> market[c(1:3,nrow(market)),]
            Open   High    Low  Close    Volume Adj.Close
2012-07-20 136.95 137.16 136.32 136.47 142904500    132.12
2012-07-23 134.47 136.38 133.84 135.09 145210900    130.79
2012-07-24 135.19 135.25 133.03 133.93 173301200    129.66
2013-07-23 169.80 169.83 169.05 169.14  80829700    167.41
> names(market)<-paste(c("Mkt.Open","Mkt.High","Mkt.Low",
+   "Mkt.Close","Mkt.Volume","Mkt.Adjusted"))
> market[c(1:3,nrow(market)),]
           Mkt.Open Mkt.High Mkt.Low Mkt.Close  Mkt.Volume Mkt.Adjusted
2012-07-20   136.95   137.16  136.32    136.47   142904500       132.12
2012-07-23   134.47   136.38  133.84    135.09   145210900       130.79
2012-07-24   135.19   135.25  133.03    133.93   173301200       129.66
2013-07-23   169.80   169.83  169.05    169.14    80829700       167.41
```

---

[6] It is also common to use the S&P 500 Index or other broad-based index as the market proxy.

Step 5: Combine the Two Data Objects Using the `merge` command, we combine
the adjusted close prices for NFLX and SPY.

```
> data.all<-merge(firm[,6],market[,6])
> data.all[c(1:3,nrow(data.all)),]
           Firm.Adjusted Mkt.Adjusted
2012-07-20         81.82       132.12
2012-07-23         79.94       130.79
2012-07-24         80.39       129.66
2013-07-23        250.26       167.41
```

Step 6: Calculate NFLX and SPY Returns Using the `diff` and `log` commands,
we calculate the logarithmic returns of NFLX and SPY. When we calculate the
returns, we convert them into percentage points. For example, on July 23, 2012,
Netflix's return of -2.32 is equal to $-2.32\,\%$.

```
> library(quantmod)
> data.all$Firm.Ret<-diff(log(data.all$Firm.Adjusted))*100
> data.all$Mkt.Ret<-diff(log(data.all$Mkt.Adjusted))*100
> data.all[c(1:3,nrow(data.all)),]
           Firm.Adjusted Mkt.Adjusted   Firm.Ret    Mkt.Ret
2012-07-20         81.82       132.12         NA         NA
2012-07-23         79.94       130.79 -2.3245359 -1.0117617
2012-07-24         80.39       129.66  0.5613437 -0.8677344
2013-07-23        250.26       167.41 -4.5691443 -0.2148100
```

Step 7: Perform Market Model Regression During the Estimation Period We
create `est.per` to contain return data during the Estimation Period of July 23, 2012
to July 22, 2013. We then run a market model regression on the data in `est.per`
using the `lm` command.

```
> est.per<-data.all[c(-1,-nrow(data.all)),3:4]
> est.per[c(1:3,nrow(est.per)),]
              Firm.Ret     Mkt.Ret
2012-07-23  -2.3245359 -1.01176170
2012-07-24   0.5613437 -0.86773438
2012-07-25 -28.7889417  0.02313476
2013-07-22  -0.9951843  0.19689156
>
> mkt.model<-lm(est.per$Firm.Ret~est.per$Mkt.Ret)
> summary(mkt.model)

Call:
lm(formula = est.per$Firm.Ret ~ est.per$Mkt.Ret)

Residuals:
    Min     1Q  Median     3Q     Max
-29.182 -1.861  -0.499  1.483  34.825

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       0.3705     0.2913   1.272    0.205
est.per$Mkt.Ret   0.9938     0.3831   2.594    0.010 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1     1

Residual standard error: 4.57 on 248 degrees of freedom
Multiple R-squared:  0.02642,   Adjusted R-squared:  0.0225
F-statistic: 6.731 on 1 and 248 DF,  p-value: 0.01004
```

Step 8: Calculate Abnormal Return, t-Statistic, and *p*-Value for Dates in Event Window We first calculate the predicted return Pred.Ret for Netflix on July 23, 2013 of 0.157 %. This means that based on the historical relationship between the market's return and Netflix's return, we would expect Netflix to return a positive 0.157 % today. However, Netflix had a -4.57 %, which means the abnormal return Ab.Ret is equal to $-4.73\%[= -4.57\% - 0.157\%]$. As noted above in the regression results, the residual standard error is 4.8 %, which is higher than the Ab.Ret. This results in a tStat of $-1.03[= -4.73\%/4.57\%]$. A tStat of that magnitude with $248°$ of freedom equals a *p*-value of 0.302. This implies that Netflix's stock return on July 23, 2013 is not statistically significant at conventional levels of significance.

```
> event.window<-data.all[nrow(data.all),3:4]
> event.window
            Firm.Ret  Mkt.Ret
2013-07-23 -4.569144 -0.21481
>
> event.window$Pred.Ret<-summary(mkt.model)$coefficients[1]+
+     summary(mkt.model)$coefficients[2]*event.window$Mkt.Ret
> event.window$Ab.Ret<-event.window$Firm.Ret-event.window$Pred.Ret
> event.window$tStat<-event.window$Ab.Ret/summary(mkt.model)$sigma
> event.window$pval<-2*(1-pt(abs(event.window$tStat),df=nrow(est.per)-2))
> options(digits=3)
> event.window
           Firm.Ret Mkt.Ret Pred.Ret Ab.Ret tStat  pval
2013-07-23    -4.57  -0.215    0.157  -4.73 -1.03 0.302
> options(digits=7)
```

As we can see from Fig. 5.2, Netflix's stock price tripled during the one calendar year period prior to the July 23, 2013 announcement, which caused the significant volatility in the stock relative to the market.

```
> title1<-"Netflix Stock Price"
> title2<-"July 20, 2012 to July 23, 2013"
> plot(data.all$Firm.Adjusted,
+     auto.grid=FALSE,
+     xlab="Date",
+     ylab="Price ($)",
+     minor.ticks="FALSE",
+     main=paste(title1,"\n",title2))
```

Figure 5.2 also shows that Netflix stock was more volatile after the jump in January 2013. Although an estimation period of one calendar year for daily returns data is typical, we may want to estimate the market model over a shorter period given that the period closer to the event window but after the jump may be more relevant in measuring the price impact of the event. We now continue from our previous calculations and create a second set of calculations.

Step 9: Identify the Date of the Jump in January 2013 From the chart, we only know that the jump occurs some time in January 2013. However, we need a more precise date in order to determine when we should start our estimation period.

```
> subset(est.per,index(est.per)>="2013-01-01" &
+       index(est.per)<="2013-01-31")
             Firm.Ret      Mkt.Ret
2013-01-02 -0.6283878   2.532318085
2013-01-03  4.8577949  -0.223760670
2013-01-04 -0.6335380   0.433082543
2013-01-07  3.2998178  -0.272203972
2013-01-08 -2.0778910  -0.286964326
2013-01-09 -1.2948852   0.252012734
2013-01-10  2.1557227   0.793876036
2013-01-11  3.3020211  -0.006936496
2013-01-14  2.1100714  -0.069391440
2013-01-15 -1.7159434   0.069391440
2013-01-16 -4.2281741  -0.013874436
2013-01-17  0.2254330   0.643133505
2013-01-18  1.4933990   0.220340239
2013-01-22 -1.3808728   0.541895211
2013-01-23  5.4223258   0.157216615
2013-01-24 35.2229673   0.027316807
2013-01-25 14.3727094   0.565146365
2013-01-28 -4.4931729  -0.122290932
2013-01-29  4.2333405   0.393514315
2013-01-30 -0.8431853  -0.393514315
2013-01-31 -1.4777706  -0.245031432
```
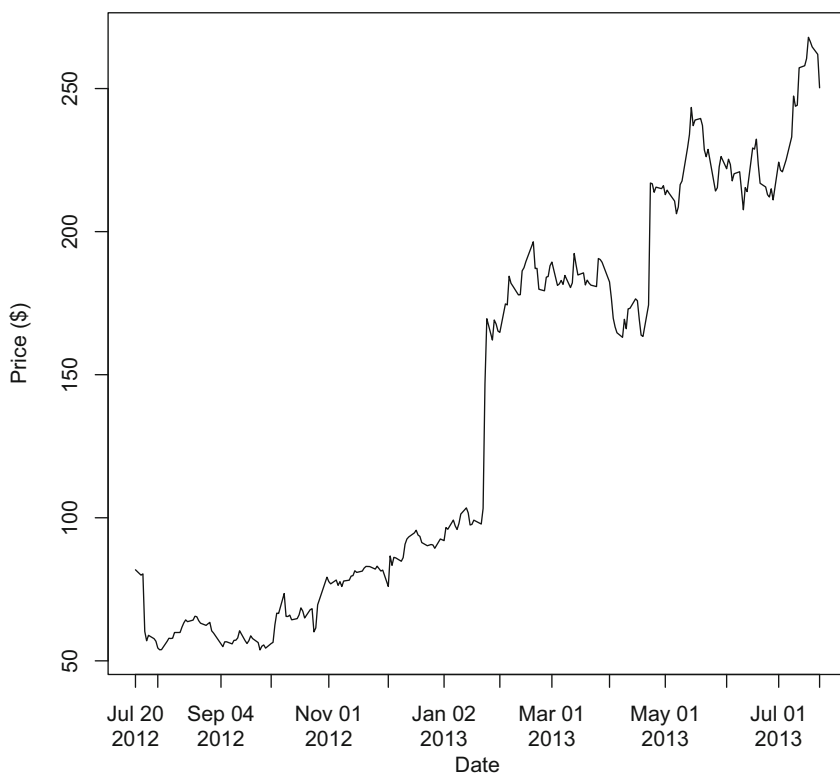

Fig. 5.2 Netflix stock price, July 20, 2012 to July 23, 2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

The output above shows that Netflix's stock price increased by 42 and 15 % on January 24, 2013 and January 25, 2013, respectively. For purpose of this analysis, we will exclude all returns on or before January 25, 2013. Therefore, we start our alternative estimation period on January 28, 2013.

Step 10: Construct Series for Estimation Period Beginning January 28, 2013
Since we already have our old estimation period that goes through July 22, 2013, we only need to subset that data object to begin January 28, 2013.

```
> est.per2<-subset(est.per,index(est.per)>="2013-01-28")
> est.per2[c(1,3,nrow(est.per2)),]
              Firm.Ret    Mkt.Ret
2013-01-28 -4.4931729 -0.1222909
2013-01-30 -0.8431853 -0.3935143
2013-07-22 -0.9951843  0.1968916
> nrow(est.per2)
[1] 122
```

As the output shows, there are 122 observations in our alternative estimation period. Although there are no strict rules as to what the minimum number of observations that should be used, 120 observations for daily data is reasonably sufficient.

Step 11: Calculate the Market Model Parameters of this Alternative Estimation Period Using the `lm` command, we run an OLS regression of the Netflix returns on the SPY returns.

```
> mkt.model2<-lm(est.per2$Firm.Ret~est.per2$Mkt.Ret)
> summary(mkt.model2)

Call:
lm(formula = est.per2$Firm.Ret ~ est.per2$Mkt.Ret)

Residuals:
   Min    1Q Median    3Q    Max
-7.413 -1.741 -0.432  1.032 20.601

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)        0.2505     0.2961   0.846   0.3994
est.per2$Mkt.Ret   0.9936     0.3837   2.589   0.0108 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1     1

Residual standard error: 3.24 on 120 degrees of freedom
Multiple R-squared:  0.05291,   Adjusted R-squared:  0.04502
F-statistic: 6.704 on 1 and 120 DF,  p-value: 0.01081
```

Step 12: Calculate Abnormal Return and Statistical Significance of the Event Date Based on the parameters from the regression, we estimate the abnormal return on July 23, 2010 of −4.61 %. The alternative estimation period yields a t-statistic

of -1.42, which corresponds to a p-value of 0.158. This implies that the abnormal return on July 23, 2010 is still not statistically significant even after using the period after the jump in January 2013.

```
> event.window2<-data.all[nrow(data.all),3:4]
> event.window2
            Firm.Ret  Mkt.Ret
2013-07-23 -4.569144 -0.21481
>
> event.window2$Pred.Ret<-summary(mkt.model2)$coefficients[1]+
+    summary(mkt.model2)$coefficients[2]*event.window2$Mkt.Ret
> event.window2$Ab.Ret<-event.window2$Firm.Ret-event.window2$Pred.Ret
> event.window2$tStat<-event.window2$Ab.Ret/summary(mkt.model2)$sigma
> event.window2$pval<-2*(1-pt(abs(event.window2$tStat),df=nrow(est.per2)-2))
> options(digits=3)
> event.window2
           Firm.Ret Mkt.Ret Pred.Ret Ab.Ret tStat  pval
2013-07-23    -4.57  -0.215    0.037  -4.61 -1.42 0.158
> options(digits=7)
```

The precision of the inputs and assumptions depends on the ultimate application of the event study. In many academic studies, the final event study result is often an average across a large sample. This averaging has the benefit of reducing the individual firm-specific effects on the abnormal returns and it may be less important to have very precise inputs and assumptions. However, when dealing with individual firm event studies, we have to be more careful in the inputs and assumptions that we make.

5.6 Further Reading

The original paper on the CAPM is Sharpe [12]. In the above example, we calculated the beta using 36 months of returns data. Another common method is to use 2 weeks of weekly returns data. Alternative methods are also used but we have to be aware of the following two trade-offs. The first trade-off is that higher frequency (i.e., daily returns data) is more susceptible to asynchronous trading issues than lower frequency (i.e., monthly data). The second trade-off is that shorter estimation periods may be more relevant to the current beta but may be more affected by extreme events than longer estimation periods, but longer estimation periods may include older and less relevant data.

The original paper on the FF Model is Fama and French [6]. Much more data compiled for the FF Model can be found in Ken French's website. An excellent treatment of factor models can be found in Damodaran [5] and Koller et al. [9].

The seminal paper on event studies is Fama et al. [7]. A good background on event studies can be found in Kothari and Warner [10]. The authors report over 575 published articles in five top finance journals made use of event studies from 1974 to

2000. A good discussion of the approach we use is in MacKinlay [11]. An alternative approach is to implement the event study in one-pass, by using a dummy variables to represent the abnormal return on event dates. This is discussed in Binder [2] and Karafiath [8]. Another variation of the event study is to augment the market model to include an industry proxy, such that the abnormal return can be interpreted as a return after accounting for market and industry factors. This model is commonly used in the context of securities litigation and is discussed in Crew et al. [4].

References

1. Arnott, R., & Lovell, R. (1993). Rebalancing why? when? how often? *The Journal of Investing, 2,* 5–10.
2. Binder, J. (1985). On the use of the multivariate regression model in event studies. *Journal of Accounting Research, 23,* 370–383.
3. Cochrane, J. (2011). Presidential address: Discount rates. *Journal of Finance, 66,* 1047–1108.
4. Crew, N., Gold, K., & Moore, M. (2007). Federal securities acts and areas of expert analysis. In R. Weil, et al. (Eds.), *Litigation services handbook* (4th ed.). New Jersey: Wiley.
5. Damodaran, A. (2012). *Investment valuation: Tools and techniques for determining the value of any asset* (3rd ed.). New Jersey: Wiley.
6. Fama, E., & French, K. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics, 33,* 3–56.
7. Fama, E., Fischer, L., Jensen, M., & Roll, R. (1969). The speed of adjustment of stock prices to new information. *International Economic Review, 10,* 1–21.
8. Karafiath, I. (1988). Using dummy variables in the event methodology. *Financial Review, 23,* 351–357.
9. Koller, T., Goedhart, M., & Wessels, D. (2010). *Valuation: Measuring and managing the value of companies* (5th ed.). New Jersey: Wiley.
10. Kothari, S. P., & Warner, J. B. (2005). Econometrics of event studies. In: B. E. Eckbo (Ed.), *Handbook of corporate finance: Empirical corporate finance*. Elsevier/North-Holland.
11. MacKinlay, A. C. (1997). *Event studies in economics and finance. Journal of Economic Literature, 35,* 13–39.
12. Sharpe, W. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance, 3,* 425–442.

Chapter 6
Risk-Adjusted Portfolio Performance Measures

Our discussion on factor models in Chap. 5 showed that we should only expect higher returns if we take on more risk. Therefore, the performance of an investment should be taken in the context of the level of risk taken. For example, the return of a manager that invests in stocks should not be compared with the return of a manager that invests in bonds, as we would expect the latter to return less because of the lower level of risk undertaken. A more appropriate comparison would be to somehow normalize the returns from the different investments by their respectively risk levels. The complicating issue is that there is no single definition of returns as well as no single definition of risk.

In this chapter, we implement several risk-adjusted performance measures that differ in their definition of returns and/or risks. In particular, we will look at the Sharpe Ratio, Roy's Safety First Ratio, Treynor Ratio, Sortino Ratio, and Information Ratio. We can use these different risk-adjusted performance measures to help us rank portfolios. However, because of the differences in the definition of return and risk, these metric may rank investments differently.

6.1 Portfolio and Benchmark Data

Prior to implementing these different risk-adjusted performance measures, we first need to have portfolios whose performance we wish to analyze. In practice, we would have portfolios of our own and we would be analyzing those portfolios. However, for our simplified analysis, we will compare the performance of two portfolios: an Actual Portfolio, which is a proxy for our investment portfolio, and an Alternative Portfolio, which is a proxy for an investment alternative. In addition, we need to set a benchmark portfolio for use with some metrics. The portfolios could either be our actual portfolios or existing securities we view as potential investment candidates. For some metrics, we need a benchmark portfolio.

Step 1: Import Actual Portfolio Data In practice, if we were measuring the risk-adjusted performance of our actual portfolio, we would be importing returns data of that portfolio. However, for illustrative purposes, we will import monthly returns data from a hypothetical portfolio constructed using equal-weighted returns of AMZN, IBM, and YHOO with monthly rebalancing from January 2011 to December 2013. The file we import is labeled **Hypothetical Portfolio (Monthly).csv**. The details of how this portfolio is calculated is described in Appendix B.

```
> port<-read.csv("Hypothetical Portfolio (Monthly).csv"
+       ,header=TRUE)
> port[c(1:3,nrow(port)),]
    X    date    port.ret
1   1 Jan 2011 0.005211301
2   2 Feb 2011 0.014024705
3   3 Mar 2011 0.021291224
36 36 Dec 2013 0.050203327
```

Step 2: Import Alternative Portfolio Data For our alternative portfolio, we consider an investment in the S&P 500 Index ETF with ticker symbol SPY. Since our Actual Portfolio is comprised of AMZN, IBM, and YHOO, which are all large capitalization stocks, we consider another portfolio of large capitalization stocks as an alternative for our illustration. The SPY data is retrieved from Yahoo Finance and is saved in a file labeled **SPY Yahoo.csv**. We then construct monthly returns for the SPY from January 2011 to December 2013 using techniques we discuss in earlier chapters.

```
> SPY<-read.csv("SPY Yahoo.csv",header=TRUE)
> date<-as.Date(SPY$Date,format="%Y-%m-%d")
> SPY<-cbind(date, SPY[,-1])
> SPY<-SPY[order(SPY$date),]
> library(xts)
> SPY<-xts(SPY[,2:7],order.by=SPY[,1])
> names(SPY)<-
+    paste(c("SPY.Open","SPY.High","SPY.Low",
+    "SPY.Close","SPY.Volume","SPY.Adjusted"))
> SPY[c(1:3,nrow(SPY)),]
           SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted
2010-12-31   125.53   125.87  125.33    125.75   91218900       118.11
2011-01-03   126.71   127.60  125.70    127.05  138725200       119.33
2011-01-04   127.33   127.37  126.19    126.98  137409700       119.26
2013-12-31   184.07   184.69  183.93    184.69   86119900       184.69
> SPY<-to.monthly(SPY)
> SPY[c(1:3,nrow(SPY)),]
         SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted
Dec 2010   125.53   125.87  125.33    125.75   91218900       118.11
Jan 2011   126.71   130.35  125.70    128.68 2860314700       120.86
Feb 2011   129.46   134.69  129.38    133.15 2819841400       125.06
Dec 2013   181.09   184.69  177.32    184.69 2232587400       184.69
Warning message:
timezone of object (UTC) is different than current timezone ().
> SPY<-SPY[,6]
> library(quantmod)
> altport.ret<-Delt(SPY$SPY.Adjusted)
> names(altport.ret)<-paste("altport.ret")
> altport.ret[c(1:3,nrow(altport.ret)),]
         altport.ret
Dec 2010          NA
Jan 2011  0.02328338
Feb 2011  0.03475095
Dec 2013  0.02594156
Warning message:
timezone of object (UTC) is different than current timezone ().
```

Step 3: Import Benchmark Bortfolio Data   Since the Actual Portfolio and Alternative Portfolio are comprised of large capitalization stocks, we use the S&P 500 Index as our benchmark portfolio. The S&P 500 Index data is obtained from Yahoo Finance and is saved in a file labeled **GSPC Yahoo.csv**. From the daily Yahoo Finance data we calculate monthly benchmark returns from January 2011 to December 2013.

```
> GSPC<-read.csv("GSPC Yahoo.csv",header=TRUE)
> date<-as.Date(GSPC$Date,format="%Y-%m-%d")
> GSPC<-cbind(date, GSPC[,-1])
> GSPC<-GSPC[order(GSPC$date),]
> GSPC<-xts(GSPC[,2:7],order.by=GSPC[,1])
> names(GSPC)<-
+    paste(c("GSPC.Open","GSPC.High","GSPC.Low",
+    "GSPC.Close","GSPC.Volume","GSPC.Adjusted"))
> GSPC[c(1:3,nrow(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
> GSPC<-to.monthly(GSPC)
> GSPC[c(1:3,nrow(GSPC)),]
         GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
Dec 2010   1256.76   1259.34  1254.19    1257.64  1799770000       1257.64
Jan 2011   1257.62   1302.67  1257.62    1286.12 92164940000       1286.12
Feb 2011   1289.14   1344.07  1289.14    1327.22 60027800000       1327.22
Dec 2013   1806.55   1849.44  1767.99    1848.36 62664960000       1848.36
Warning message:
timezone of object (UTC) is different than current timezone ().
> benchmark.ret<-Delt(GSPC$GSPC.Adjusted)
> names(benchmark.ret)<-paste("bench.ret")
> benchmark.ret[c(1:3,nrow(benchmark.ret)),]
         bench.ret
Dec 2010        NA
Jan 2011 0.02264559
Feb 2011 0.03195658
Dec 2013 0.02356283
Warning message:
timezone of object (UTC) is different than current timezone ().
```

Step 4: Combine Portfolios and Benchmark Data To make the subsequent calcu-
lations easier, we will combine the returns of the three securities into one data object.
We use the `digits=3` option to reduce the number of decimals that are reported as
having too many decimals does not add much information and only makes reading
the values more difficult. Note that using `digits=3` here changes how we display
any number for the rest of our R session.

```
> options(digits=3)
> port<-cbind(port[,-1],
+    data.frame(altport.ret[-1,],
+    data.frame(benchmark.ret[-1,])))
> port[c(1:3,nrow(port)),]
              date port.ret altport.ret bench.ret
Jan 2011 Jan 2011  0.00521     0.02328   0.02265
Feb 2011 Feb 2011  0.01402     0.03475   0.03196
Mar 2011 Mar 2011  0.02129     0.00008  -0.00105
Dec 2013 Dec 2013  0.05020     0.02594   0.02356
```

Step 5: Rename Index Values to Observation Number The output above shows that the index (i.e., column on the left with no header) and the first column contain identical date values. Since no additional information is given by the index, it is better to replace the index with the observation number as this gives us more useful information. We change the index values by using the rownames command. As we can see, we have 36 monthly return observations for the two portfolios and the benchmark.

```
> rownames(port)<-seq(1,nrow(port),1)
> port[c(1:3,nrow(port)),]
       date port.ret altport.ret bench.ret
1  Jan 2011  0.00521     0.02328   0.02265
2  Feb 2011  0.01402     0.03475   0.03196
3  Mar 2011  0.02129     0.00008  -0.00105
36 Dec 2013  0.05020     0.02594   0.02356
```

We will use the data object port in our subsequent analysis.

6.2 Sharpe Ratio

The Sharpe Ratio compares the excess return of a portfolio relative to the risk-free rate with the portfolio's standard deviation. That is,

$$Sharpe_p = \frac{E(r_p) - r_f}{\sigma_p}, \tag{6.1}$$

where $E(r_p)$ is the annualized average return of portfolio p, r_f is the annual risk-free rate, and σ_p is the annualized standard deviation of portfolio p. The higher the Sharpe Ratio, the higher the risk-adjusted performance of the portfolio. Below, we show the steps in calculating the Sharpe Ratio for the Actual Portfolio and Alternative Portfolio.

Step 1: Identify the Frequency of the Returns and the Length of the Estimation Period The Sharpe Ratio is dependent on both the frequency of the data used (e.g., daily versus monthly) as well as the time period selected for the analysis of the returns. According to the Morningstar Investment Glossary, we can calculate the Sharpe Ratio using the past 36 months of returns.[1] We follow this definition for our example.

Step 2: Identify the Risk-Free Rate In Eq. (6.1), we see that we need a risk-free rate. Recall we used the 3-Month Treasury in our CAPM calculation in Chap. 5. For consistency, we will also use the 3-Month Constant Maturity Treasury to calculate the Sharpe Ratio. On December 31, 2013, the 3-Month T-Bill had a yield of 0.07 %.

---

[1] http://www.morningstar.com/InvGlossary/sharpe_ratio.aspx.

```
> Rf=0.0007
```

Step 3: Annualize Monthly Portfolio Returns and Standard Deviation It is typical to present the Sharpe Ratio as an annual number. So we have to annualize all components of the Sharpe Ratio when we perform the calculation. We begin by looking at the expected or average return of the portfolio. Since we are using monthly returns over a 3 years period, we have 36 return observations. The average monthly return is calculated using the `mean` command, which we then multiply by 12 to get to an annualized return number.

```
> annual.port.ret<-mean(port$port.ret)*12
> annual.port.ret
[1] 0.247
```

We then calculate the annualized portfolio standard deviation, by calculating the monthly portfolio standard deviation and scaling the resulting value up by the square root of 12.

```
> annual.port.sd<-sd(port$port.ret)*sqrt(12)
> annual.port.sd
[1] 0.147
```

Step 4: Calculate Sharpe Ratio We now have all the inputs we need to calculate the Sharpe Ratio. The Sharpe Ratio for the Actual Portfolio is 1.68.

```
> Sharpe.port<-(annual.port.ret-Rf)/annual.port.sd
> Sharpe.port
[1] 1.68
```

Step 5: Repeat Steps 1–4 for the Alternative Portfolio Following the same steps above when we calculated the Sharpe Ratio for the Actual Portfolio, we can also compute the Sharpe Ratio for the Alternative Portfolio. We show the Sharpe Ratio for the Alternative Portfolio is 1.29.

```
> annual.altport.ret<-mean(port$altport.ret)*12
> annual.altport.ret
[1] 0.157
> annual.altport.sd<-sd(port$altport.ret)*sqrt(12)
> annual.altport.sd
[1] 0.121
> Sharpe.altport<-(annual.altport.ret-Rf)/annual.altport.sd
> Sharpe.altport
[1] 1.29
```

The Actual Portfolio has a Sharpe Ratio of 1.68, which is higher than the Sharpe Ratio of the Alternative Portfolio of 1.29. As such, based on the Sharpe Ratio, the Actual Portfolio yields a higher risk-adjusted return.

6.3   Roy's Safety First Ratio

Roy's Safety First (SF) Ratio makes a slight modification to the Sharpe Ratio. Specifically, instead of using the risk-free rate, Roy's SF Ratio instead uses a target or minimum acceptable return to calculate the excess return. That is,

$$RoySF_p = \frac{E(r_p) - MAR}{\sigma_p}, \tag{6.2}$$

where *MAR* is the minimum acceptable return and all other variables are defined similarly to Eq. (6.1). Again, the higher the Roy's Safety First Ratio, the better the risk-adjusted performance of the portfolio.

Step 1: Determine the MAR To implement Roy's SF Ratio, we need to come up with a minimum acceptable return (MAR). This MAR can potentially depend on many factors, such as investors return objectives, risk preferences, etc. For example, a simple minimum acceptable return could be to have at least a real return of zero. That is, the MAR should match inflation expectations. Based on the December 2013 Livingston Survey, the 2012–2013 inflation rate is 1.5 %. We use this as our proxy for MAR and calculate Roy's SF Ratio for our portfolio and alternative portfolio using Eq. (6.2).

```
> mar<-0.015
```

Step 2: Calculate the Annual Portfolio Return and Standard Deviation Since we already calculated the annual portfolio return and standard deviation above, we only need to make sure the correct values are still held in R's memory.

```
> annual.port.ret
[1] 0.247
> annual.port.sd
[1] 0.147
```

Advantage of Calling in Previously Created Data

Some may wonder why we are calling-in the values we previously created for `annual.port.ret` instead of re-importing and calculating the data. The reason for this is that when calculations are interrelated, it is best to make sure we are using the same data. One way to ensure that is the case is to call-in the same variable names. This way we avoid making any errors when re-pulling and recalculating the data.

In addition, calling-in previously created data also makes our code much simpler to understand. We are not using excess space performing calculations we have already done. However, if we are performing new, independent calculations on the data or the previous data has already been modified, we may not have a choice but to re-pull and recalculate the data.

Step 3: Calculate the Roy's Safety First Ratio for the Actual Portfolio We now have all the inputs to calculate Roy's SF Ratio. We find that for the Actual Portfolio, the Roy's SF Ratio is 1.58.

```
> Roy.SF.port<-(annual.port.ret-mar)/annual.port.sd
> Roy.SF.port
[1] 1.58
```

Step 4: Repeat Steps 1–3 for the Alternative Portfolio Following the same steps above when we calculated the Roy's SF Ratio for the Actual Portfolio, we can also compute the Roy's SF Ratio for the Alternative Portfolio. We show the Roy's SF Ratio for the Alternative Portfolio is 1.17.

```
> annual.altport.ret
[1] 0.157
> annual.altport.sd
[1] 0.121
> Roy.SF.altport<-(annual.altport.ret-mar)/annual.altport.sd
> Roy.SF.altport
[1] 1.17
```

The Actual Portfolio has a Roy's SF Ratio of 1.58, which is higher than the Roy's SF Ratio of the Alternative Portfolio of 1.17. As such, based on the Roy's SF Ratio, the Actual Portfolio yields a higher risk-adjusted return.

6.4 Treynor Ratio

The Treynor Ratio modifies the denominator in the Sharpe Ratio to use beta instead of standard deviation in the denominator. That is,

$$Treynor_p = \frac{E(r_p) - r_f}{\beta_p},\tag{6.3}$$

where β_p is the beta of the portfolio and the rest of the terms are defined similarly to Eq. (6.1). The higher the Treynor Ratio, the better the risk-adjusted performance of the portfolio.

Step 1: Identify the Frequency of Returns and Length of the Estimation Period
Similar to the Sharpe Ratio, the frequency of the returns and the period over which the returns are analyzed would impact the resulting Treynor Ratio. Using beta instead of standard deviation implies that the Treynor Ratio only considers systematic risk while the Sharpe Ratio considers total risk (i.e., systematic and firm-specific risk). Therefore, the ratios could differ if the ratio of systematic risk to total risk is different in the two funds we are comparing.

Step 2: Import Actual Portfolio Returns Since we already calculated the annual portfolio return earlier, we only need to make sure that the data is still in R's memory.

```
> annual.port.ret
[1] 0.247
```

Step 3: Obtain the Relevant Risk-Free Rate We already called in the risk-free rate earlier, so we only need to make sure the data is still in R's memory. Note that we use the scipen=100 option, which makes increases the threshold for R before it converts the numbers into scientific notation. This makes reading the values a lot easier.

```
> options(scipen=100)
> Rf
[1] 0.0007
```

Step 4: Estimate the Beta of the Actual Portfolio To get the beta, we run a market model regression using the lm command (see Chap. 5 for details). We then extract the beta from the regression summary by typing coefficients[2].

```
> port[c(1:3,nrow(port)),]
        date port.ret altport.ret bench.ret
1  Jan 2011  0.00521     0.02328   0.02265
2  Feb 2011  0.01402     0.03475   0.03196
3  Mar 2011  0.02129     0.00008  -0.00105
36 Dec 2013  0.05020     0.02594   0.02356
> reg<-lm(port$port.ret~port$bench.ret)
> port.beta<-summary(reg)$coefficients[2]
> port.beta
[1] 0.764
```

Note that we did not print out the regression summary like we did when we implemented the CAPM and market model regressions in Chap. 5. Since we already know where the beta comes from and how to extract the beta, showing the full output is unnecessary at this point. However, we can always print the regression output using the `summary` command.

Step 5: Calculate the Treynor Ratio for the Actual Portfolio Using the beta calculated in Step 2 as the denominator in Eq. (6.3) to calculate the Treynor Ratio.

```
> Treynor.port<-(annual.port.ret-Rf)/port.beta
> Treynor.port
[1] 0.322
```

Step 6: Repeat Steps 1–5 for the Alternative Portfolio Similarly, we apply the same technique to calculate the Treynor Ratio for the Alternative Portfolio.

```
> annual.altport.ret
[1] 0.157
> Rf
[1] 0.0007
> reg.altport<-lm(port$altport.ret~port$bench.ret)
> beta.altport<-summary(reg.altport)$coefficients[2]
> beta.altport
[1] 0.999
> Treynor.altport<-(annual.altport.ret-Rf)/beta.altport
> Treynor.altport
[1] 0.157
```

The Actual Portfolio has a Treynor Ratio of 0.322, which is higher than the Treynor Ratio of the Alternative Portfolio of 0.157. As such, based on the Treynor Ratio, the Actual Portfolio yields a higher risk-adjusted return.

6.5 Sortino Ratio

Given our implementation of Roy's SF Ratio using a minimum acceptable return (MAR), the Sortino Ratio has the same numerator Roy's SF Ratio in Eq. (6.2). However, the denominator measures only the deviation of values lower than the MAR, which is referred to as Downside Deviation (DD). That is,

$$Sortino_p = \frac{E(r_p) - MAR}{DD_p},$$
(6.4)

where $E(r_p)$ is the average return of portfolio p, MAR is the minimum acceptable return, and DD_p is the downside deviation. The higher the Sortino Ratio, the better the risk-adjusted performance of the portfolio.

Step 1: Determine the Period MAR From the Roy's SF Ratio calculation, we used an annual MAR of 1.5 %. However, since the Sortino Ratio requires identifying monthly returns below the MAR, we have to convert our annual MAR to a monthly MAR by dividing the annual MAR by 12.

```
> mar
[1] 0.015
> period.mar<-mar/12
> period.mar
[1] 0.00125
```

Step 2: Identify Monthly Returns in Actual Portfolio That are Less Than the MAR We create a variable called dummy that takes on the value of 1 if the return is less than the period MAR and 0 otherwise.

```
> port[c(1:3,nrow(port)),]
      date port.ret altport.ret bench.ret
1  Jan 2011  0.00521     0.02328   0.02265
2  Feb 2011  0.01402     0.03475   0.03196
3  Mar 2011  0.02129     0.00008  -0.00105
36 Dec 2013  0.05020     0.02594   0.02356
> downside.port<-port
> downside.port$dummy<-ifelse(port$port.ret<period.mar,1,0)
> downside.port[c(1:3,nrow(downside.port)),]
      date port.ret altport.ret bench.ret dummy
1  Jan 2011  0.00521     0.02328   0.02265     0
2  Feb 2011  0.01402     0.03475   0.03196     0
3  Mar 2011  0.02129     0.00008  -0.00105     0
36 Dec 2013  0.05020     0.02594   0.02356     0
```

Step 3: Extract Monthly Returns That are Less Than the MAR We then subset the data to only contain observations with dummy=1.

```
> downside.port<-subset(downside.port,downside.port$dummy==1)
> downside.port[c(1:3,nrow(downside.port)),]
      date port.ret altport.ret bench.ret dummy
5  May 2011 -0.0219     -0.0113   -0.0135      1
6  Jun 2011 -0.0120     -0.0168   -0.0183      1
8  Aug 2011 -0.0148     -0.0550   -0.0568      1
32 Aug 2013 -0.0542     -0.0300   -0.0313      1
```

Step 4: Calculate Downside Deviation downside.port only contains returns that are less than the period MAR. The standard deviation of these returns is called the

downside deviation. We then annualize the downside deviation to make it compatible
with the other inputs to the Sortino Ratio.

```
> dd.port<-sd(downside.port$port.ret)*sqrt(12)
> dd.port
[1] 0.0564
```

Step 5: Calculate Sortino Ratio We now have all the inputs needed to calculate
the Sortino Ratio. The output shows that the Sortino Ratio for the Actual Portfolio
is 4.11.

```
> Sortino.port=(annual.port.ret-mar)/dd.port
> Sortino.port
[1] 4.11
```

Step 6: Repeat Steps 1–5 for the Alternative Portfolio Similarly, we apply the
same technique to calculate the Sortino Ratio for the Alternative Portfolio. Our
calculations show the Sortino Ratio for the Alternative Portfolio is 1.76.

```
> downside.altport<-port
> downside.altport$dummy<-ifelse(port$altport.ret<period.mar,1,0)
> downside.altport[c(1:3,nrow(downside.altport)),]
        date port.ret altport.ret bench.ret dummy
1  Jan 2011  0.00521     0.02328    0.02265     0
2  Feb 2011  0.01402     0.03475    0.03196     0
3  Mar 2011  0.02129     0.00008   -0.00105     1
36 Dec 2013  0.05020     0.02594    0.02356     0
> downside.altport<-subset(downside.altport,downside.altport$dummy==1)
> downside.altport[c(1:3,nrow(downside.altport)),]
        date port.ret altport.ret bench.ret dummy
3  Mar 2011   0.0213     0.00008   -0.00105     1
5  May 2011  -0.0219    -0.01127   -0.01350     1
6  Jun 2011  -0.0120    -0.01682   -0.01826     1
32 Aug 2013  -0.0542    -0.03000   -0.03130     1
> dd.altport<-sd(downside.altport$altport.ret)*sqrt(12)
> dd.altport
[1] 0.0808
> Sortino.altport<-(annual.altport.ret-mar)/dd.altport
> Sortino.altport
[1] 1.76
```

The Actual Portfolio has a Sortino Ratio of 4.11, which is higher than the Sortino
Ratio of the Alternative Portfolio of 1.76. As such, based on the Sortino Ratio, the
Actual Portfolio yields a higher risk-adjusted return.

6.6 Information Ratio

While the four ratios above are close variants of one another, the *Information Ratio* (IR) is a risk-adjusted return measure that requires more modification and its interpretation is not as simple. The IR is the ratio of the portfolio alpha over the tracking error. The *alpha* of the portfolio is the annualized *active return*, which is the difference between the portfolio's return and the benchmark return (e.g., return of the S&P 500 Index). The *tracking error* is the annualized standard deviation of the active return. As such, if we use monthly returns, the alpha is equal to the monthly active return multiplied by 12 and the tracking error is equal to the standard deviation of the monthly returns multiplied by the square root of 12.

Step 1: Calculate the Active Return of the Actual Portfolio To calculate the IR, we first calculate the Active Return for our portfolio and the alternative portfolio. The calculation of the Active Return is the portfolio return less the benchmark return.

```
> port[c(1:3,nrow(port)),]
      date port.ret altport.ret bench.ret
1  Jan 2011  0.00521     0.02328   0.02265
2  Feb 2011  0.01402     0.03475   0.03196
3  Mar 2011  0.02129     0.00008  -0.00105
36 Dec 2013  0.05020     0.02594   0.02356
> Act.Ret.port<-port$port.ret-port$bench.ret
> head(Act.Ret.port)
[1] -0.01743 -0.01793  0.02234  0.03627 -0.00842  0.00624
```

Step 2: Calculate Portfolio Alpha The alpha of the Actual portfolio is the annualized average monthly active return. We calculate the average monthly active return using the `mean` command. We then multiply this monthly average by 12 to annualize it.

```
> alpha.port<-mean(Act.Ret.port)*12
> alpha.port
[1] 0.111
```

Step 3: Calculate Tracking Error The tracking error is the annualized standard deviation of the monthly active returns. We calculate the standard deviation of the monthly active returns using the `sd` command. We then multiply this monthly standard deviation by the square root of 12 to annualize it.

```
> tracking.error.port<-sd(Act.Ret.port)*sqrt(12)
> tracking.error.port
[1] 0.117
```

Step 4: Calculate the Information Ratio We then divide the alpha by the tracking error of the portfolio to get the Information Ratio. We find the Information Ratio of the Actual Portfolio equals 0.944.

```
> IR.port<-alpha.port/tracking.error.port
> IR.port
[1] 0.944
```

Step 5: Repeat Steps 1–4 for the Alternative Portfolio Similarly, we apply the same technique to calculate the Information Ratio for the Alternative Portfolio. Our calculations show the Information Ratio for the Alternative Portfolio is 6.09.

```
> Act.Ret.altport<-port$altport.ret-port$bench.ret
> head(Act.Ret.altport)
[1] 0.000638 0.002794 0.001127 0.000528 0.002234 0.001440
> alpha.altport<-mean(Act.Ret.altport)*12
> alpha.altport
[1] 0.0209
> tracking.error.altport<-sd(Act.Ret.altport)*sqrt(12)
> tracking.error.altport
[1] 0.00343
> IR.altport<-alpha.altport/tracking.error.altport
> IR.altport
[1] 6.09
```

The Actual Portfolio has an Information Ratio of 0.944, which is lower than the Information Ratio of the Alternative Portfolio of 6.09. As such, based on the Information Ratio, the Alternative Portfolio yields a higher risk-adjusted return.

6.7   Combining Results

Although we have done the calculations above, we would want to summarize the results of the analysis for ease of comparison. To visualize better the results of the individual risk-adjusted performance measures, we can combine all five results into a tabular form. We do this using a combination of the `rbind` command and `cbind` command. The `cbind` strings together the risk-adjusted return measures for the portfolio into one row and the measures for the alternative portfolio in another row. The `rbind` command then stacks both rows together.

```
> RARet<-rbind(cbind(Sharpe.port,
+                     Roy.SF.port,
+                     Treynor.port,
+                     Sortino.port,
+                     IR.port),
+              cbind(Sharpe.altport,
+                     Roy.SF.altport,
+                     Treynor.altport,
+                     Sortino.altport,
+                     IR.altport))
> RARet
      Sharpe.port Roy.SF.port Treynor.port Sortino.port IR.port
[1,]         1.68        1.58        0.322         4.11   0.944
[2,]         1.29        1.17        0.157         1.76   6.088
```

We then fix the variable names using the `colnames` command and the row labels using the `rownames` command. Note that since this is the end of our discussion of risk-adjusted performance measures, we use the `digits=7` option to convert the number of decimal places on display back to the `R` default value.s

```
> colnames(RARet)<-paste(c("Sharpe","Roy SF","Treynor",
+      "Sortino","Info Ratio"))
> rownames(RARet)<-paste(c("Portfolio","Alt Portfolio"))
> RARet
               Sharpe Roy SF Treynor Sortino Info Ratio
Portfolio        1.68   1.58   0.322    4.11      0.944
Alt Portfolio    1.29   1.17   0.157    1.76      6.088
> options(digits=7)
```

Now that we have all the different calculations in one table, it is now easier to see which portfolio performs better under which metrics. Recall that the rule for all these ratios is that the higher the ratio the better. As we can see from the output above, our Actual Portfolio has a higher Sharpe Ratio, Roy SF Ratio, Treynor Ratio, and Sortino Ratio than the Alternative Portfolio. Looking specifically at the Sharpe and Treynor Ratios, we can see that our portfolio provides better risk-adjusted returns regardless of whether we measure risk as total risk (i.e., standard deviation) or only systematic risk (i.e., beta).

Looking at the Sharpe Ratio and Roy SF Ratio, even considering a minimum acceptable return slightly higher than the risk-free rate does not change our results. Comparing the Roy SF Ratio and Sortino Ratio, we can see the difference in the two ratios is that the Roy SF Ratio uses standard deviation in the denominator while the Sortino Ratio uses the downside deviation (i.e., standard deviation of returns that fail to meet the MAR). The results are consistent with the Actual Portfolio having many more returns that exceed the MAR that caused a lot more of the deviation, as we have quite a small downside deviation. The distribution of returns in the alternative portfolio seems to be more symmetric, although still skewed a little bit by returns above the MAR.

The results above for the IR suggests a different story. The IR implies that the manager of the Actual Portfolio performed worse relative to the risk assumed compared to the manager of the Alternative Portfolio. The active choice by the manager of our Actual Portfolio to select securities resulted in a higher alpha but also a correspondingly higher tracking error. The Alternative Portfolio had a lower alpha but also tracked our benchmark much closer.

6.8 Further Reading

Maginn et al. [2] contains a detailed discussion of the portfolio performance measures we discussed in this chapter. Applications of risk-adjusted performance measures for hedge funds can be found in Tran [6] and Gregoriou [1].

References

1. Gregoriou, G., Hubner, G., Papageorgiou, N., & Rouah, F. (2005). *Hedge funds: Insights in performance measurement, risk analysis, and portfolio allocation*. New Jersey: Wiley.
2. Maginn, J., Tuttle, D., Pinto, J., & McLeavey, D. (2007). *Managing investment portfolios: A dynamic process* (3rd ed). New Jersey: Wiley.
3. Markowitz, H. (1952). Portfolio selection. *The Journal of Finance, 7,* 77–91.
4. Sharpe, W. (1975). Adjusting for risk in portfolio performance. *Journal of Portfolio Management, 1,* 29–34.
5. Sharpe, W. (1994). The Sharpe ratio. *Journal of Portfolio Management, 21,* 49–58.
6. Tran, V. (2006). *Evaluating hedge fund performance*. New Jersey: Wiley.

Chapter 7
Markowitz Mean-Variance Optimization

In Markowitz [3], Harry Markowitz put forth a theory of how to use mean and variance to determine portfolios that offer the highest return for a given level of risk. This approach showed that by properly diversifying your portfolio, we can reduce the risk of the entire portfolio from that of the weighted-average risk of the assets that make up the portfolio. Subsequent research has shown that most of the benefits of diversification can be attained by holding 12–18 securities in our portfolio.

In this chapter, we show how to construct the Mean-Variance (MV) Efficient Frontier, which is the set of all portfolio generated by various combinations of the securities in the portfolio that yield the highest return for a given level of risk. We begin by manually constructing the MV Efficient Frontier using two assets, so we can gain the intuition of what we are trying to accomplish. Then, we convert this manual approach for two assets to an approach that uses *quadratic programming*. Since many investors (e.g., pension funds) do not or cannot short sell securities for various reasons, we implement the two-asset case not allowing short selling.

After showing we get the same results in the two-asset case either through the manual approach or quadratic programming, we extend the quadratic programming approach to show how we find the MV efficient portfolios when we have multiple securities. In the multiple asset scenario, we first continue with the no short sale case and then demonstrate the effect on the MV Efficient Frontier of allowing short selling. We will see that allowing short-selling extends outward the MV Efficient Frontier such that we are able to achieve higher returns for a given level of risk.

7.1 Two Assets the "Long Way"

We first demonstrate the intuition of how to construct the MV Efficient Frontier using the case of a 2-asset portfolio and manually laying out each step of the calculation. This approach is termed the "Long Way" because we lay out each step without the benefit of any mathematical optimization techniques.

Let us consider the scenario in which we can either put money in US large-capitalization stocks (using the S&P 500 Index ETF with symbol SPY) or US

investment grade bonds (using the SPDR Barclays Aggregate Bond ETF with symbol LAG). In determining how much money to invest in each of the two securities, we may want to know what combination of these two securities yields the smallest portfolio risk (i.e., the minimum variance portfolio) and we may want to know what combination of these two securities yields the highest Sharpe Ratio (i.e., the tangency portfolio). We demonstrate below how to identify these two portfolios. In addition, not all portfolios generated by a combination of SPY and LAG yield the highest return for a given level of risk. We identify the set of portfolios that meet that criteria and these portfolios are called *mean-variance efficient portfolios*.

Step 1: Import SPY and LAG Data from Yahoo Finance The first step is for us to use the data for SPY and LAG from Yahoo Finance from December 31, 2010 to December 31, 2013. These are saved in files labeled **SPY Yahoo.csv** and **LAG Yahoo.csv**. We then calculate monthly returns for these two securities. We limit the intermediate output below as we have performed the same calculations several times earlier (see, for example, Chap. 5).

```
> # Large-Cap Equity ETF Data
> data.SPY<-read.csv("SPY Yahoo.csv",header=TRUE)
> date<-as.Date(data.SPY$Date,format="%Y-%m-%d")
> data.SPY<-cbind(date, data.SPY[,-1])
> data.SPY<-data.SPY[order(data.SPY$date),]
> library(xts)
> data.SPY<-xts(data.SPY[,2:7],order.by=data.SPY[,1])
> names(data.SPY)<-
+    paste(c("SPY.Open","SPY.High","SPY.Low",
+    "SPY.Close","SPY.Volume","SPY.Adjusted"))
> library(quantmod)
> SPY.monthly<-to.monthly(data.SPY)
> SPY.monthly<-SPY.monthly[,6]
> SPY.ret<-Delt(SPY.monthly$data.SPY.Adjusted)
> names(SPY.ret)<-paste("SPY.Ret")
> SPY.ret[c(1:3,nrow(SPY.ret)),]
             SPY.Ret
Dec 2010          NA
Jan 2011 0.02328338
Feb 2011 0.03475095
Dec 2013 0.02594156
Warning message:
timezone of object (UTC) is different than current timezone ().
>
> # Aggregate Bond Market ETF Data
> data.LAG<-read.csv("LAG Yahoo.csv",header=TRUE)
> date<-as.Date(data.LAG$Date,format="%Y-%m-%d")
> data.LAG<-cbind(date, data.LAG[,-1])
> data.LAG<-data.LAG[order(data.LAG$date),]
> data.LAG<-xts(data.LAG[,2:7],order.by=data.LAG[,1])
> names(data.LAG)<-
+    paste(c("LAG.Open","LAG.High","LAG.Low",
+    "LAG.Close","LAG.Volume","LAG.Adjusted"))
> LAG.monthly<-to.monthly(data.LAG)
```

```
> LAG.monthly<-LAG.monthly[,6]
> LAG.ret<-Delt(LAG.monthly$data.LAG.Adjusted)
> names(LAG.ret)<-paste("LAG.Ret")
> LAG.ret[c(1:3,nrow(LAG.ret)),]
              LAG.Ret
Dec 2010          NA
Jan 2011  0.000000000
Feb 2011  0.002720560
Dec 2013 -0.004413063
Warning message:
timezone of object (UTC) is different than current timezone ().
>
```

Step 2: Combine Monthly Return Data for SPY and LAG We then combine the
two returns data into one object named `Ret.monthly`. We use the `digits=3`
option to reduce the number of decimals on display. Note that this setting will persist
for the remainder of the R session or until we override the settings.

```
> options(digits=3)
> Ret.monthly<-cbind(SPY.ret[-1,],LAG.ret[-1,])
> Ret.monthly[c(1:3,nrow(Ret.monthly)),]
         SPY.Ret   LAG.Ret
Jan 2011 0.02328  0.000000
Feb 2011 0.03475  0.002721
Mar 2011 0.00008 -0.000969
Dec 2013 0.02594 -0.004413
```

Step 3: Calculate the Mean and Standard Deviation of SPY and LAG Returns
As the name suggests, the Mean-Variance Efficient Frontier requires us to find the
mean and variance (technically, standard deviation) of the portfolio returns. As such,
we would need to find the mean and standard deviation of the returns of the individual
securities. The mean return is calculated using the `mean` command and the standard
deviation is calculated using the `sd` command.

```
> SPY.Avg<-mean(Ret.monthly$SPY.Ret)
> SPY.Avg
[1] 0.0131
> SPY.sd<-sd(Ret.monthly$SPY.Ret)
> SPY.sd
[1] 0.035
> LAG.Avg<-mean(Ret.monthly$LAG.Ret)
> LAG.Avg
[1] 0.00258
> LAG.sd<-sd(Ret.monthly$LAG.Ret)
> LAG.sd
[1] 0.00846
```

Step 4: Find the Covariance of SPY and LAG Returns Recall from Chap. 4 that,
in the context of portfolios, what is important is how the assets in the portfolio move
together. As such, we find the covariance of the returns of SPY and LAG using the
`cov` command.

```
> covar<-cov(Ret.monthly$SPY.Ret,Ret.monthly$LAG.Ret)
> covar
         LAG.Ret
SPY.Ret -5.27e-05
```

Note that the output of the `cov` command has labels that may end up to be more misleading than helpful. As such, to avoid confusion, we clean up the `covar` result to make it a single number with no labels. In addition, we use the `scipen=100` option so that the value is not reported in scientific notation.

```
> options(scipen=100)
> covar<-covar[1,1]
> covar
[1] -0.0000527
```

Step 5: Create a Series of Weights for SPY Stock   When we calculated portfolio risk in Chap. 4, we assumed the weights of the two securities. To find out all the risk/return combination of SPY and LAG stock, we need to create a series that all possible weight combinations. Since individual investors and some institutional investors do not or are not allowed to short sell stock, we can restrict our weights from 100 % SPY/0 % LAG to 0 % SPY/100 % LAG. Since the weights must sum to one, we can create a series of weights for SPY between 0 and 100 % by 1 % increments. Note that the number of increments is a trade-off between speed and accuracy. For our purpose, using 1 % increments would be sufficiently accurate and would not affect the speed of running the program substantially.

```
> Portfolio<-data.frame(seq(0,1,by=.01))
> names(Portfolio)<-paste("SPY.wgt")
> Portfolio[1:5,]
[1] 0.00 0.01 0.02 0.03 0.04
> Portfolio[(nrow(Portfolio)-5):nrow(Portfolio),]
[1] 0.95 0.96 0.97 0.98 0.99 1.00
```

Step 6: Create a Series of Weights for LAG Stock   Since we only have two assets in the portfolio and the portfolio weights must sum to one, we can use the above series of weights for SPY to find LAG's weights. We accomplish this by subtracting the SPY weight from 1 for all 101 observations of the data.

```
> Portfolio$LAG.wgt<-1-Portfolio$SPY.wgt
> Portfolio[c(1:3,nrow(Portfolio)),]
    SPY.wgt LAG.wgt
1      0.00    1.00
2      0.01    0.99
3      0.02    0.98
101    1.00    0.00
```

Step 7: Calculate Portfolio Return for Each Weight Combination   Since we have combinations of SPY and LAG weights, we can calculate portfolio returns given each of those combinations. The portfolio return is the weighted average of the returns of the securities in the portfolio.

```
> Portfolio$PortRet<-Portfolio$SPY.wgt*SPY.Avg+
+    Portfolio$LAG.wgt*LAG.Avg
> Portfolio[c(1:3,nrow(Portfolio)),]
    SPY.wgt LAG.wgt PortRet
1      0.00    1.00 0.00258
2      0.01    0.99 0.00269
3      0.02    0.98 0.00279
101    1.00    0.00 0.01309
```

Step 8: Calculate Portfolio Risk for Each Weight Combination Similarly, we can calculate portfolio risks for each combination of SPY and LAG weights. Chapter 4 describes the calculation in more detail.

```
> Portfolio$PortRisk<-sqrt((Portfolio$SPY.wgt^2*SPY.sd^2)+
+    (Portfolio$LAG.wgt^2*LAG.sd^2)+
+    (2*covar*Portfolio$SPY.wgt*Portfolio$LAG.wgt))
> Portfolio[c(1:3,nrow(Portfolio)),]
    SPY.wgt LAG.wgt PortRet PortRisk
1      0.00    1.00 0.00258  0.00846
2      0.01    0.99 0.00269  0.00832
3      0.02    0.98 0.00279  0.00820
101    1.00    0.00 0.01309  0.03503
```

Step 9: Identify the Minimum-Variance Portfolio The *minimum variance portfolio* is the portfolio that has the smallest portfolio risk. To identify this portfolio, we use a combination of the subset command and min command. The output below shows that the minimum variance portfolio (minvar.port) is attained if we invest 9 % of our portfolio in SPY and 91 % in LAG. This portfolio gives us a monthly portfolio return of 0.35 % and monthly portfolio standard deviation of 0.78 %.

```
> minvar.port<-subset(Portfolio,Portfolio$PortRisk==min(Portfolio$PortRisk))
> minvar.port
    SPY.wgt LAG.wgt PortRet PortRisk
10     0.09    0.91 0.00353  0.00778
```

Step 10: Identify the Tangency Portfolio The *tangency portfolio* is the portfolio that yields the highest Sharpe Ratio. The Sharpe Ratio requires a risk-free rate and we use the 3-Month Constant Maturity Treasury as a proxy for that rate. Since the yield as of December 31, 2013 of 0.07 % is an annual yield, we convert this to a monthly yield by dividing the rate by 12.

```
> riskfree=.0007/12
> Portfolio$Sharpe<-(Portfolio$PortRet-riskfree)/Portfolio$PortRisk
> Portfolio[c(1:3,nrow(Portfolio)),]
    SPY.wgt LAG.wgt PortRet PortRisk Sharpe
1      0.00    1.00 0.00258  0.00846  0.299
2      0.01    0.99 0.00269  0.00832  0.316
3      0.02    0.98 0.00279  0.00820  0.334
101    1.00    0.00 0.01309  0.03503  0.372
```

To identify the portfolio with the highest Sharpe Ratio, we use a combination of the `subset` command and the `max` command. The output below shows that the tangency portfolio is attained if we invest 22 % in SPY and 78 % in LAG. This portfolio gives us a monthly return of 0.49 % and monthly standard deviation 0.92 % with a Sharpe Ratio of 0.525.

```
> tangency.port<-subset(Portfolio,Portfolio$Sharpe==max(Portfolio$Sharpe))
> tangency.port
   SPY.wgt LAG.wgt PortRet PortRisk Sharpe
23    0.22    0.78  0.0049  0.00921  0.525
```

Step 11: Identify Efficient Portfolios Not all combinations of SPY and LAG can be classified as *efficient portfolios*. Efficient portfolios are only those portfolios that yield the highest return for a given level of risk. We extract these efficient portfolios by using the `subset` command and only keep those portfolios with returns greater than the return of the minimum variance portfolio.

```
> eff.frontier<-subset(Portfolio,Portfolio$PortRet>=minvar.port$PortRet)
> eff.frontier[c(1:3,nrow(eff.frontier)),]
    SPY.wgt LAG.wgt PortRet PortRisk Sharpe
10     0.09    0.91 0.00353  0.00778  0.446
11     0.10    0.90 0.00363  0.00779  0.459
12     0.11    0.89 0.00374  0.00782  0.471
101    1.00    0.00 0.01309  0.03503  0.372
```

Step 12: Plot the MV Efficient Frontier We can visualize the MV Efficient Frontier by using the `plot` command. We first plot in gray circles all the portfolios we formed from all combinations of SPY and LAG. We then add a solid triangular point using the `points` command and `pch=17` option to denote the minimum variance portfolio. The `cex=3` option increases the size of the triangle, so we can easily see the minimum variance portfolio on the frontier. Then, we add a solid circle using the `points` command and `pch=19` option to denote the tangency portfolio. We also use the `cex=3` option to increase the size of the solid circle so we can easily see where the tangency portfolio is on the frontier. Then, we plot the efficient frontier using the `points` command and color the circles black using the `col="black"` option.

```
> plot(x=Portfolio$PortRisk,
+     xlab="Portfolio Risk",
+     y=Portfolio$PortRet,
+     ylab="Portfolio Return",
+     col="gray40",
+     main='Mean-Variance Efficient Frontier of Two Assets
+        Based on the "Long Way"')
> abline(h=0,lty=1)
> points(x=minvar.port$PortRisk,y=minvar.port$PortRet,pch=17,cex=3)
> points(x=tangency.port$PortRisk,y=tangency.port$PortRet,pch=19,cex=3)
> points(x=eff.frontier$PortRisk,y=eff.frontier$PortRet)
```

As shown in Fig. 7.1, the gray circles below the efficient frontier have the same level of risk as some of the portfolios on the frontier. This means that these portfolios

**Mean-Variance Efficient Frontier of Two Assets
Based on the "Long Way"**
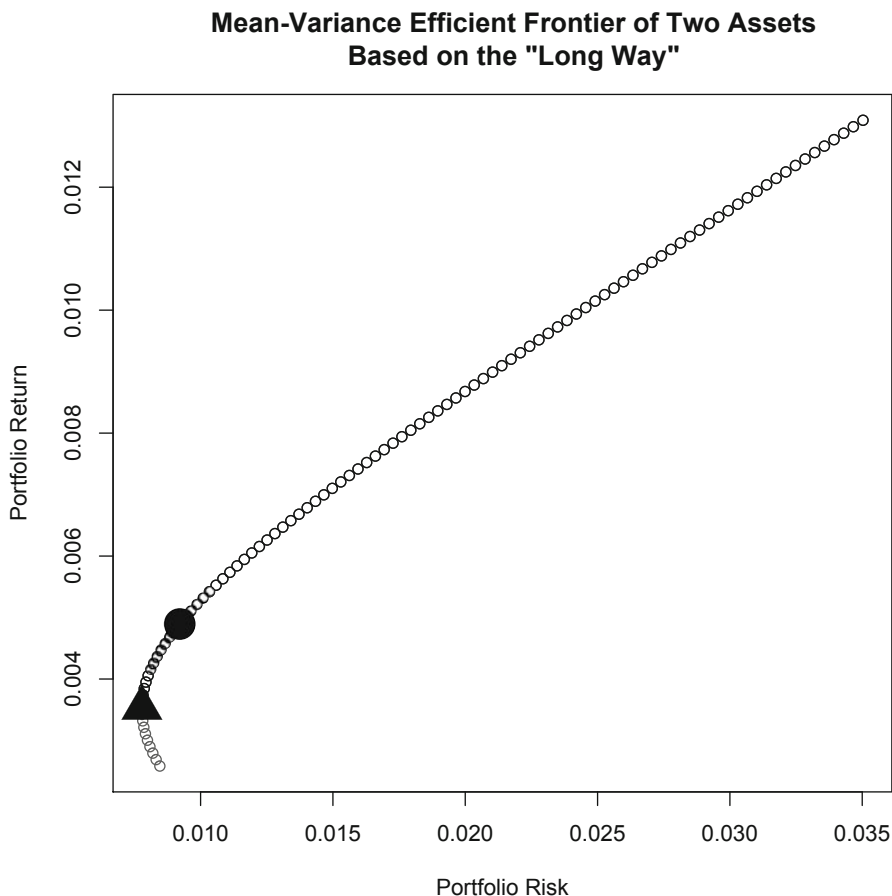
Fig. 7.1 Mean variance efficient frontier of portfolios consisting of SPY and LAG stock calculated using the "Long Way"

below the frontier are not efficient because they provide lower returns for the same level of risk. This is why when identifying points on the MV efficient frontier, we only looked at returns of portfolios above the minimum variance point.

7.2 Two-Assets Using Quadratic Programming

The "Long Way" is tractable in the two-asset case. However, most portfolios have more than two assets, and using the "Long Way" becomes extremely challenging very quickly. To see why, note that we will have a growing number of variance-covariance terms the more securities we put in our portfolio. In fact, given N securities in our

portfolio, the variance-covariance matrix would have $N(N+1)/2$ elements, which consists of N variance terms and $N(N-1)/2$ covariance terms. Given two assets, we would have three $[= 2(2+1)/2]$ terms in total, consisting of two variance terms and one $[= 2(2-1)/2]$ covariance term. When we have four assets, as in our later example, we would have a total of $10 [= 4(4+1)/2]$ terms, consisting of four variance terms and six $[= 4(4-1)/2]$ covariance terms. If we had 15 securities, we would end up with a total of $120 [= 15(15+1)/2]$ total terms, consisting of 15 variance terms and $105 [= 15(15-1)/2]$ covariance terms. As such, using the "Long Way" is going to be difficult in practice.

The "Long Way" is useful to gain an intuition of what is going on when we apply *quadratic programming*. The technique is called quadratic programming because we are performing an optimization on a *quadratic function*. A quadratic function is a function that has a variable (in this case weight) that is squared. In the two asset case, we are minimizing Eq. (4.2). The weights in that formula are squared. For our purpose, quadratic programming finds the combination of securities that yield the minimum risk for a specific level of return.

Step 1: Import Returns Data and Convert it Into a Matrix Recall in Chap. 4 we had to use matrix algebra, we need to use more of that when we calculate the MV efficient frontier using quadratic programming. Note that we have already imported the returns data in the prior section, so we simply check to make sure the Ret.monthly data object is still in the R memory. We then convert the returns data into a matrix using the matrix command. The matrix command takes on two arguments. The first argument is the data object that will be converted and the second argument is how many rows should the matrix have. We then rename the column headers of the matrix so that we know the first column are SPY returns and the second column are LAG returns.

```
> Ret.monthly[c(1:3,nrow(Ret.monthly)),]
         SPY.Ret    LAG.Ret
Jan 2011 0.02328  0.000000
Feb 2011 0.03475  0.002721
Mar 2011 0.00008 -0.000969
Dec 2013 0.02594 -0.004413
> mat.ret<-matrix(Ret.monthly,nrow(Ret.monthly))
> mat.ret[1:3,]
        [,1]       [,2]
[1,] 0.02328  0.000000
[2,] 0.03475  0.002721
[3,] 0.00008 -0.000969
> colnames(mat.ret)<-c("SPY","LAG")
> head(mat.ret)
         SPY        LAG
[1,]  0.02328  0.000000
[2,]  0.03475  0.002721
[3,]  0.00008 -0.000969
[4,]  0.02902  0.018429
[5,] -0.01127  0.012190
[6,] -0.01682 -0.006022
```

```
> tail(mat.ret)
          SPY       LAG
[31,]  0.0517 -0.000356
[32,] -0.0300 -0.007839
[33,]  0.0316  0.015083
[34,]  0.0463  0.005661
[35,]  0.0296 -0.003518
[36,]  0.0259 -0.004413
```

Step 2: Calculate Variance-Covariance (VCOV) Matrix of Returns Applying the
`cov` command to a matrix creates a VCOV matrix. The terms on the main diagonal
(i.e., top-left to bottom-right) are the variance terms. In particular, the variance of
SPY returns is 0.00123 and the variance of LAG returns is 0.00007. The numbers
off the main diagonal are the covariance terms. The two numbers in the off-diagonal
are both equal to -0.0000527 because the covariance of SPY/LAG and LAG/SPY
are the same number.

```
> VCOV<-cov(mat.ret)
> VCOV
          SPY       LAG
SPY  0.0012271 -0.0000527
LAG -0.0000527  0.0000716
```

Step 3: Construct the Target Portfolio Return Vector Instead of creating a series
of weights for SPY and LAG, in the quadratic programming approach we construct a
series of target portfolio returns. The term *vector* means a row or column of numbers,
so a row of target returns is called a vector. To create this series of dummy portfolio
returns, we have to come up with upper and lower bounds. When short sales are not
allowed, as is the case here, the natural bounds for the returns would be the SPY
return on one end and the LAG return on the other end. To see why, the two extreme
combination of returns would be 100 % SPY/0 % LAG, which yields a portfolio
return equal to the SPY return, and 0 % SPY/100 % LAG, which yields a portfolio
return equal to the LAG return. So the first step is to find the average return for SPY
and LAG over the relevant time period.

```
> avg.ret<-matrix(apply(mat.ret,2,mean))
> colnames(avg.ret)<-paste("Avg.Ret")
> rownames(avg.ret)<-paste(c("SPY","LAG"))
> avg.ret
    Avg.Ret
SPY 0.01309
LAG 0.00258
```

Then, we set the smaller of the average returns as the minimum return `min.ret`
and the larger of the average returns as the maximum return `max.ret`.

```
> min.ret<-min(avg.ret)
> min.ret
[1] 0.00258
> max.ret<-max(avg.ret)
> max.ret
[1] 0.0131
```

Next, we create a sequence that begins with `min.ret` and ends with `max.ret` with 100 increments in between. Again, the number of increments is a trade-off between accuracy and speed. Having 100 increments is a reasonable choice for our purposes.

```
> increments=100
> tgt.ret<-seq(min.ret,max.ret,length=increments)
> head(tgt.ret)
[1] 0.00258 0.00269 0.00280 0.00290 0.00301 0.00311
> tail(tgt.ret)
[1] 0.0126 0.0127 0.0128 0.0129 0.0130 0.0131
```

Step 4: Construct Dummy Portfolio Standard Deviation Vector An output of the quadratic program are portfolio standard deviations. However, the output needs to overwrite the values of an existing vector of standard deviations after each iteration. Since the number of iterations equal the number of increments, we create a series of 100 zeroes for our dummy portfolio standard deviation vector. To do this, we use the `rep` command, which repeats the value of zero, with the `length` option equal to the number of increments.

```
> tgt.sd<-rep(0,length=increments)
> tgt.sd
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[37] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[73] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Step 5: Construct Dummy Portfolio Weights Vector Similarly, an output of the quadratic programming are portfolio weights. However, the output needs to overwrite the values of an existing vector of weights after each iteration. Since the number of iterations equals the number of increments, we create a series of 100 zeroes for our dummy portfolio weights vector. To do this, we use the `rep` command, which repeats the value of zero, with the `length` option equal to the number of increments.

```
> wgt<-matrix(0,nrow=increments,ncol=length(avg.ret))
> head(wgt)
     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
[4,]    0    0
[5,]    0    0
[6,]    0    0
> tail(wgt)
      [,1] [,2]
[95,]    0    0
[96,]    0    0
[97,]    0    0
[98,]    0    0
[99,]    0    0
[100,]   0    0
```

Step 6: Run the `quadprog` **Optimizer** For our purpose, we will not go through all the details of how the optimizer works. We will do a high-level pass on what the optimizer is attempting to do and the output it generates.

To run the optimizer, we use the `solve.QP` function in the `quadprog` package.[1] The optimizer loops through the calculations 100 times (i.e., the number of `increments` we set). During each iteration, we want the optimizer to minimize portfolio risk as in Eq. (4.3) subject to three constraints: (i) the weights in the portfolio have to equal one, (ii) the portfolio return has to equal a target return, which we have specified in `tgt.ret`, and (iii) the weights for each security has to be greater than or equal to zero (i.e., short selling is not allowed). What makes the code below appear to be complicated is because much of the code is required to modify the built-in setup of `solveQP` to conform to the above objective function and constraints.

```
> library(quadprog)
> for (i in 1:increments){
+     Dmat<-2*VCOV
+     dvec<-c(rep(0,length(avg.ret)))
+     Amat<-cbind(rep(1,length(avg.ret)),avg.ret,
+         diag(1,nrow=ncol(Ret.monthly)))
+     bvec<-c(1,tgt.ret[i],rep(0,ncol(Ret.monthly)))
+     soln<-solve.QP(Dmat,dvec,Amat,bvec=bvec,meq=2)
+     tgt.sd[i]<-sqrt(soln$value)
+     wgt[i,]<-soln$solution}
```

Below, we report the output of the optimizer for the portfolio standard deviation, which corresponds to each target portfolio return level we constructed.

```
> head(tgt.sd)
[1] 0.00846 0.00832 0.00819 0.00808 0.00799 0.00791
> tail(tgt.sd)
[1] 0.0332 0.0336 0.0339 0.0343 0.0347 0.0350
```

The output of `wgt` below shows the weights of SPY and LAG for each target return level.

```
> options(scipen=100)
> head(wgt)
                    [,1]  [,2]
[1,] 0.0000000000000000416 1.000
[2,] 0.0101010101010101244 0.990
[3,] 0.0202020202020202211 0.980
[4,] 0.0303030303030303108 0.970
[5,] 0.0404040404040404075 0.960
[6,] 0.0505050505050505597 0.949
```

[1] S original by Berwin A. Turlach R port by Andreas Weingessel <Andreas.Weingessel@ci.tuwien.ac.at> (2013). quadprog: Functions to solve Quadratic Programming Problems.. R package version 1.5-5. http://CRAN.R-project.org/package=quadprog.

```
> tail(wgt)
        [,1]                    [,2]
[95,]   0.949 0.050505050505050830
[96,]   0.960 0.040404040404040664
[97,]   0.970 0.030303030303030720
[98,]   0.980 0.020202020202020776
[99,]   0.990 0.010101010101010388
[100,]  1.000 0.000000000000000444
```

Note that the first and last observations above seem to sum to a number slightly greater than one. This is because the optimizer stops after some threshold error level is reached and doesn't continue the loop until the values equal exactly zero. The latter would only slow down the calculations with no real gains. For our purposes, these very small numbers for SPY in the first observation and LAG in the last observation can be considered equal to zero. Hence, we can overwrite those entries to make them zero. We also rename the column headers so we have more informative labels that directly tell us these are weights of SPY and LAG.

```
> colnames(wgt)<-paste(c("wgt.SPY","wgt.LAG"))
> wgt[1,1]<-0
> wgt[nrow(wgt),2]<-0
> head(wgt)
     wgt.SPY wgt.LAG
[1,]  0.0000   1.000
[2,]  0.0101   0.990
[3,]  0.0202   0.980
[4,]  0.0303   0.970
[5,]  0.0404   0.960
[6,]  0.0505   0.949
> tail(wgt)
      wgt.SPY wgt.LAG
[95,]   0.949  0.0505
[96,]   0.960  0.0404
[97,]   0.970  0.0303
[98,]   0.980  0.0202
[99,]   0.990  0.0101
[100,]  1.000  0.0000
```

Step 7: Combine Portfolio Returns, Portfolio Standard Deviations, and Portfolio Weights We then combine the `tgt.ret` and `tgt.sd` vectors into a `tgt.port` matrix.

```
> tgt.port<-data.frame(cbind(tgt.ret,tgt.sd,wgt))
> head(tgt.port)
  tgt.ret  tgt.sd wgt.SPY wgt.LAG
1 0.00258 0.00846  0.0000   1.000
2 0.00269 0.00832  0.0101   0.990
3 0.00280 0.00819  0.0202   0.980
4 0.00290 0.00808  0.0303   0.970
5 0.00301 0.00799  0.0404   0.960
6 0.00311 0.00791  0.0505   0.949
```

```
> tail(tgt.port)
     tgt.ret tgt.sd wgt.SPY wgt.LAG
95   0.0126 0.0332   0.949  0.0505
96   0.0127 0.0336   0.960  0.0404
97   0.0128 0.0339   0.970  0.0303
98   0.0129 0.0343   0.980  0.0202
99   0.0130 0.0347   0.990  0.0101
100  0.0131 0.0350   1.000  0.0000
```

Step 8: Identify the Minimum Variance Portfolio The *minimum variance portfolio* is the portfolio that has the smallest portfolio risk. To identify this portfolio, we use a combination of the `subset` command and `min` command. The output below shows that the minimum variance portfolio (`minvar.port`) is attained if we invest 9 % of our portfolio in SPY and 91 % in LAG. This portfolio gives us a monthly portfolio return of 0.35 % and monthly portfolio standard deviation of 0.78 %.

```
> minvar.port<-subset(tgt.port,tgt.port$tgt.sd==min(tgt.port$tgt.sd))
> minvar.port
   tgt.ret  tgt.sd wgt.SPY wgt.LAG
10 0.00354 0.00778  0.0909   0.909
```

Notice that the weight allocation, portfolio return, and portfolio standard deviation of the minimum variance portfolio are slightly different when calculated using the quadratic programming approach compared to the "Long Way." This is a result of creating target weights as the starting point in the "Long Way" and creating target returns as the starting point in the quadratic programming approach.

Step 9: Identify the Tangency Portfolio The *tangency portfolio* is the portfolio that yields the highest Sharpe Ratio. The Sharpe Ratio requires a risk-free rate and we use the 3-Month Constant Maturity Treasury as a proxy for that rate. Since the yield as of December 31, 2013 of 0.07 % is an annual yield, we convert this to a monthly yield by dividing the rate by 12. Since we are using the same risk-free rate as we have calculated earlier, we simply need to check that `riskfree` remains in R's memory prior to calcualting the Sharpe Ratio of each portfolio.

```
> riskfree
[1] 0.0000583
> tgt.port$Sharpe<-(tgt.port$tgt.ret-riskfree)/tgt.port$tgt.sd
> head(tgt.port)
  tgt.ret  tgt.sd wgt.SPY wgt.LAG Sharpe
1 0.00258 0.00846  0.0000   1.000  0.299
2 0.00269 0.00832  0.0101   0.990  0.316
3 0.00280 0.00819  0.0202   0.980  0.334
4 0.00290 0.00808  0.0303   0.970  0.352
5 0.00301 0.00799  0.0404   0.960  0.369
6 0.00311 0.00791  0.0505   0.949  0.386
> tail(tgt.port)
     tgt.ret tgt.sd wgt.SPY wgt.LAG Sharpe
95   0.0126 0.0332   0.949  0.0505  0.377
96   0.0127 0.0336   0.960  0.0404  0.376
97   0.0128 0.0339   0.970  0.0303  0.375
98   0.0129 0.0343   0.980  0.0202  0.374
99   0.0130 0.0347   0.990  0.0101  0.373
100  0.0131 0.0350   1.000  0.0000  0.372
```

To identify the portfolio with the highest Sharpe Ratio, we use a combination of the `subset` command and the `max` command. The output below shows that the tangency portfolio is attained if we invest 22 % in SPY and 78 % in LAG. This portfolio gives us a monthly return of 0.49 % and monthly standard deviation 0.93 % with a Sharpe Ratio of 0.525.

```
> tangency.port<-subset(tgt.port,tgt.port$Sharpe==max(tgt.port$Sharpe))
> tangency.port
   tgt.ret  tgt.sd wgt.SPY wgt.LAG Sharpe
23 0.00492 0.00926   0.222   0.778  0.525
```

For reasons explained above, the values generated by the quadratic programming approach are slightly different than the results of the "Long Way."

Step 10: Identify Efficient Portfolios Not all combinations of SPY and LAG can be classified as *efficient portfolios*. Efficient portfolios are only those portfolios that yield the highest return for a given level of risk. We extract these efficient portfolios by using the `subset` command and only keep those portfolios with returns greater than the return of the minimum variance portfolio.

```
> eff.frontier<-subset(tgt.port,tgt.port$tgt.ret>=minvar.port$tgt.ret)
> eff.frontier[c(1:3,nrow(eff.frontier)),]
     tgt.ret  tgt.sd wgt.SPY wgt.LAG Sharpe
10  0.00354 0.00778  0.0909   0.909  0.447
11  0.00365 0.00780  0.1010   0.899  0.460
12  0.00375 0.00783  0.1111   0.889  0.472
100 0.01309 0.03503  1.0000   0.000  0.372
```

Step 11: Plot the MV Efficient Frontier We can visualize the MV Efficient Frontier by using the `plot` command. We first plot in gray circles all the portfolios we formed from all combinations of SPY and LAG. We then add a solid triangular point using the `points` command and `pch=17` option to denote the minimum variance portfolio. Then, we add a solid circle using the `points` comand and `pch=19` option to denote the tangency portfolio. Note that we use the `cex=3` option for both the triangle and circle, so we can easily identify where on the efficient frontier the minimum variance portfolio and tangency portfolio are, respectively. Then, we plot the efficient frontier using the `points` command and color the circles black using the `col="black"` option.

```
> plot(x=tgt.sd,
+     xlab="Portfolio Risk",
+     y=tgt.ret,
+     ylab="Portfolio Return",
+     col="gray40",
+     main="Mean-Variance Efficient Frontier of Two Assets
+     Based on the Quadratic Programming Approach")
> abline(h=0,lty=1)
> points(x=minvar.port$tgt.sd,y=minvar.port$tgt.ret,pch=17,cex=3)
> points(x=tangency.port$tgt.sd,y=tangency.port$tgt.ret,pch=19,cex=3)
> points(x=eff.frontier$tgt.sd,y=eff.frontier$tgt.ret)
```
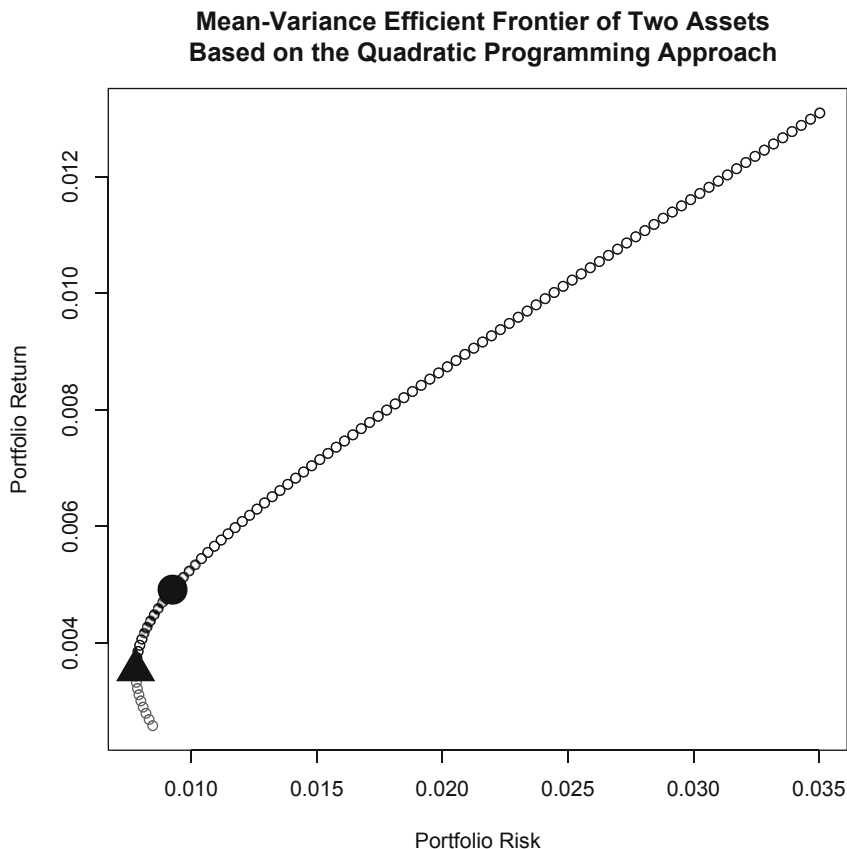
Mean-Variance Efficient Frontier of Two Assets
Based on the Quadratic Programming Approach

Fig. 7.2 Mean variance efficient frontier of portfolios consisting of SPY and LAG stock calculated using quadratic programming

As shown in Fig. 7.2, the gray circles below the efficient frontier have the same level of risk as some of the portfolios on the frontier. This means that these portfolios below the frontier are not efficient because they provide lower returns for the same level of risk as some other portfolio that lies on the MV Efficient Frontier. This is why, when identifying points on the MV efficient frontier, we only looked at returns of portfolios above the minimum variance point.

Compare "Long Way" and Quadratic Programming

We noted that the results of the quadratic programming approach seem to be slightly different from the results of the "Long Way" for the minimum variance portfolio and tangency portfolio. We explained that the difference is due to the

different method with which the calculations are undertaken. Specifically, the
"Long Way" fits the calculations to a set combination of portfolio weights
while the quadratic programming approach fits the calculations to a set port-
folio return level. Regardless of this issue, we show below that a plot of the
efficient frontiers calculated using the "Long Way" and quadratic programming
approaches are identical for all practical purposes.

```
> plot(x=Portfolio$PortRisk,
+     xlab="Portfolio Risk",
+     y=Portfolio$PortRet,
+     ylab="Portfolio Return",
+     type="l",
+     lwd=6,
+     lty=3,
+     main="Mean-Variance Efficient Frontier of Two Assets
+     Comparing Both Approaches")
> abline(h=0,lty=1)
> lines(x=tgt.sd,y=tgt.ret,col="gray40",type="l",lwd=2)
> legend("bottomright",
+       c('The "Long Way"',"Quadratic Programming"),
+       col=c("black","gray40"),
+       lty=c(3,1),
+       lwd=c(6,2))
```

Figure 7.3 shows the results of this plot. As the chart demonstrates, there is
virtually no difference in the results of the two approaches.

7.3   Multiple Assets Using Quadratic Programming

Our portfolio above is comprised of only two ETFs. Although these ETFs provide
us with exposure to many securities, our exposure was limited to only large-
capitalization US stocks and US investment grade bonds. In this section, we show
how to extend the quadratic programming approach to include two additional assets
that give us exposure to small capitalization US stocks (as represented by the SPDR
S&P 600 Small Cap ETF with symbol SLY) and global equities (as represented by the
SPDR MSCI All Country World Index ex USA ETF with symbol CWI). We will see
that the method for adding new securities when applying the quadratic programming
approach is relatively straightforward.

**Step 1: Import SPY, LAG, SLY, and CWI Data From Yahoo Finance and Cal-
culate Monthly Returns for Each Security** We have previously retrieved SPY and
LAG data, so we check to make sure that data is still in the R's memory.
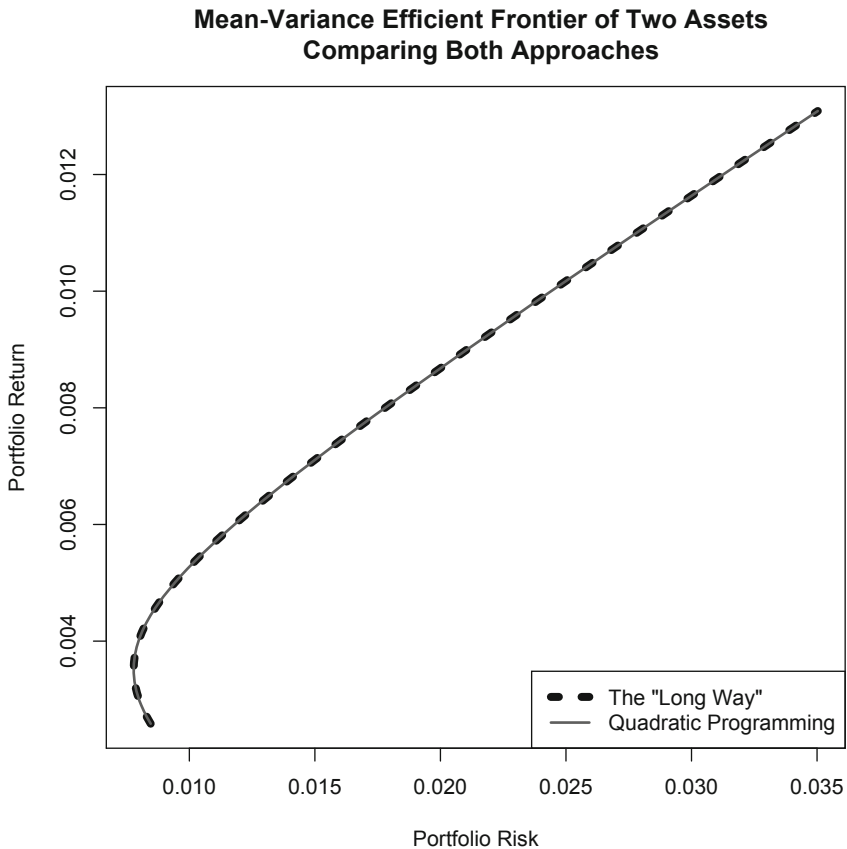
**Mean-Variance Efficient Frontier of Two Assets
Comparing Both Approaches**

Fig. 7.3 Comparison of mean variance efficient frontiers calculated using the "Long Way" and quadratic programming

```
> # Large-Cap ETF Data
> SPY.ret[c(1:3,nrow(SPY.ret)),]
         SPY.Ret
Dec 2010      NA
Jan 2011  0.0233
Feb 2011  0.0348
Dec 2013  0.0259
Warning message:
timezone of object (UTC) is different than current timezone ().
>
> # Aggregate Bond Market Data
> LAG.ret[c(1:3,nrow(LAG.ret)),]
         LAG.Ret
Dec 2010      NA
Jan 2011  0.00000
Feb 2011  0.00272
Dec 2013 -0.00441
Warning message:
timezone of object (UTC) is different than current timezone ().
```

Then, we import data for SLY and CWI from Yahoo Finance data for the period December 31, 2010 to December 31, 2013. The CSV files containing the SLY and CWI data are saved in the R working directory as **SLY Yahoo.csv** and **CWI Yahoo.csv**, respectively. We import this data and calculate monthly returns using similar techniques we have discussed previously.

```
> # Small Cap Data Data
> data.SLY<-read.csv("SLY Yahoo.csv",header=TRUE)
> date<-as.Date(data.SLY$Date,format="%Y-%m-%d")
> data.SLY<-cbind(date, data.SLY[,-1])
> data.SLY<-data.SLY[order(data.SLY$date),]
> data.SLY<-xts(data.SLY[,2:7],order.by=data.SLY[,1])
> names(data.SLY)<-
+    paste(c("SLY.Open","SLY.High","SLY.Low",
+    "SLY.Close","SLY.Volume","SLY.Adjusted"))
> SLY.monthly<-to.monthly(data.SLY)
> SLY.monthly<-SLY.monthly[,6]
> SLY.ret<-Delt(SLY.monthly$data.SLY.Adjusted)
> names(SLY.ret)<-paste("SLY.Ret")
> SLY.ret[c(1:3,nrow(SLY.ret)),]
         SLY.Ret
Dec 2010      NA
Jan 2011 0.00145
Feb 2011 0.03806
Dec 2013 0.01242
Warning message:
timezone of object (UTC) is different than current timezone ().
>
> # Global Equities Data
> data.CWI<-read.csv("CWI Yahoo.csv",header=TRUE)
> date<-as.Date(data.CWI$Date,format="%Y-%m-%d")
> data.CWI<-cbind(date, data.CWI[,-1])
> data.CWI<-data.CWI[order(data.CWI$date),]
> data.CWI<-xts(data.CWI[,2:7],order.by=data.CWI[,1])
> names(data.CWI)<-
+    paste(c("CWI.Open","CWI.High","CWI.Low",
+    "CWI.Close","CWI.Volume","CWI.Adjusted"))
> CWI.monthly<-to.monthly(data.CWI)
> CWI.monthly<-CWI.monthly[,6]
> CWI.ret<-Delt(CWI.monthly$data.CWI.Adjusted)
> names(CWI.ret)<-paste("CWI.Ret")
> CWI.ret[c(1:3,nrow(CWI.ret)),]
         CWI.Ret
Dec 2010      NA
Jan 2011  0.0113
Feb 2011  0.0291
Dec 2013  0.0151
Warning message:
timezone of object (UTC) is different than current timezone ().
```

Step 2: Combine Monthly Returns Data for the Four Securities and Convert Combined Data Into a Returns Matrix Using the cbind command, we combine the monthly returns for SPY, LAG, SLY, and CWI from January 2011 to December 2013.

```
> Ret.monthly<-cbind(SPY.ret[-1,],LAG.ret[-1,],SLY.ret[-1,],CWI.ret[-1,])
> Ret.monthly[c(1:3,nrow(Ret.monthly)),]
         SPY.Ret   LAG.Ret SLY.Ret  CWI.Ret
Jan 2011 0.02328  0.000000 0.00145  0.01131
Feb 2011 0.03475  0.002721 0.03806  0.02907
Mar 2011 0.00008 -0.000969 0.03342 -0.00497
Dec 2013 0.02594 -0.004413 0.01242  0.01510
```

We then use the `matrix` command to convert the combined data into a matrix. We rename the column headers using the `colnames` command with the relevant ticker to better identify the return series.

```
> mat.ret<-matrix(Ret.monthly,nrow(Ret.monthly))
> mat.ret[1:3,]
        [,1]       [,2]    [,3]     [,4]
[1,] 0.02328  0.000000 0.00145  0.01131
[2,] 0.03475  0.002721 0.03806  0.02907
[3,] 0.00008 -0.000969 0.03342 -0.00497
>
> colnames(mat.ret)<-c("SPY","LAG","SLY","CWI")
> head(mat.ret)
         SPY       LAG      SLY      CWI
[1,]  0.02328  0.000000  0.00145  0.01131
[2,]  0.03475  0.002721  0.03806  0.02907
[3,]  0.00008 -0.000969  0.03342 -0.00497
[4,]  0.02902  0.018429  0.02754  0.05148
[5,] -0.01127  0.012190 -0.01588 -0.03353
[6,] -0.01682 -0.006022 -0.01229 -0.01290
> tail(mat.ret)
          SPY       LAG     SLY      CWI
[31,]  0.0517 -0.000356  0.0691  0.04342
[32,] -0.0300 -0.007839 -0.0253 -0.01491
[33,]  0.0316  0.015083  0.0620  0.06683
[34,]  0.0463  0.005661  0.0412  0.03398
[35,]  0.0296 -0.003518  0.0414  0.00314
[36,]  0.0259 -0.004413  0.0124  0.01510
```

Step 3: Calculate Variance-Covariance (VCOV) Matrix of Returns Applying the `cov` command to a matrix creates a VCOV matrix. The terms on the main diagonal (i.e., top-left to bottom-right) are the variance terms. Specifically, the variance of returns for SPY is 0.00123, LAG is 0.00007, SLY is 0.00209, and CWI is 0.00232. The numbers that are off the main diagonal are the covariance terms. Notice that the covariance terms come in pairs because, say, the covariance of SPY/LAG and covariance of LAG/SPY are the same.

```
> VCOV<-cov(mat.ret)
> VCOV
          SPY        LAG       SLY       CWI
SPY  0.0012271 -0.0000527  0.001516  0.0014881
LAG -0.0000527  0.0000716 -0.000102 -0.0000129
SLY  0.0015163 -0.0001023  0.002089  0.0019055
CWI  0.0014881 -0.0000129  0.001906  0.0023205
```

Step 4: Construct the Target Portfolio Return Vector We now create a target portfolio return vector that is a constraint that needs to be met when run the optimizer. The first step here is to calculate the average return of each security in the portfolio. We use the `apply` command with a second argument of two to tell R that we want to apply the function `mean` to each column. This returns a column vector, where each element is the average monthly return for the security. We rename the row names and column names to make describing the output easier.

```
> avg.ret<-matrix(apply(mat.ret,2,mean))
> rownames(avg.ret)<-c("SPY","LAG","SLY","CWI")
> colnames(avg.ret)<-c("Avg.Ret")
> avg.ret
    Avg.Ret
SPY 0.01309
LAG 0.00258
SLY 0.01504
CWI 0.00507
```

We now need to set the bounds of the target return vector. Since this scenario still does not allow short sale constraints, the smallest return we can earn is the minimum average return among the four securities in the portfolio (i.e., LAG return of 0.258 %) and the largest return we can earn is the maximum average return among the four securities in the portfolio (i.e., SLY return of 1.504 %). To see why, since the portfolio return is the weighted-average return of the securities in the portfolio and the sum of the weights must equal one (at least in this scenario), we can do no worse than putting 100 % of our money in the security with the smallest return and we can do no better than putting all of our money in the security with the largest return.

```
> min.ret<-min(avg.ret)
> min.ret
[1] 0.00258
> max.ret<-max(avg.ret)
> max.ret
[1] 0.015
```

We then use 100 increments between the minimum and maximum returns to generate our target returns. The optimizer is run 100 times, which is the same number of increments that we set here. For each run of the optimizer, the optimizer finds the combination of weights of the four securities that generates the target return with the lowest risk. For example, there will be some combination of SPY, LAG, SLY, and CWI that generates a return of 0.26 %. Since this is the LAG return, we know that the portfolio will be comprised of 100 % LAG and no weight is placed on the other three securities. Conversely, the 1.5 % portfolio target return on the high end will be comprised of 100 % SLY and no weight is placed on the other three securities.

```
> increments=100
> tgt.ret<-seq(min.ret,max.ret,length=increments)
> head(tgt.ret)
[1] 0.00258 0.00271 0.00284 0.00296 0.00309 0.00321
> tail(tgt.ret)
[1] 0.0144 0.0145 0.0147 0.0148 0.0149 0.0150
```

Step 5: Construct Dummy Portfolio Standard Deviation Vector For each of
the 100 loops the optimizer will go through, the optimizer will output a portfolio
standard deviation. We need a placeholder for all 100 of those observations so that
the optimizer can overwrite those entries with the appropriate portfolio standard
deviation after each loop.

```
> tgt.sd<-rep(0,length=increments)
> tgt.sd
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [37] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [73] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Step 6: Construct Dummy Portfolio Weights Vector Similarly, the optimizer also
outputs portfolio weights and needs to overwrite a placeholder for those weights.

```
> wgt<-matrix(0,nrow=increments,ncol=length(avg.ret))
> head(wgt)
      [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
[4,]    0    0    0    0
[5,]    0    0    0    0
[6,]    0    0    0    0
> tail(wgt)
       [,1] [,2] [,3] [,4]
[95,]     0    0    0    0
[96,]     0    0    0    0
[97,]     0    0    0    0
[98,]     0    0    0    0
[99,]     0    0    0    0
[100,]    0    0    0    0
```

Step 7: Run the `quadprog` **Optimizer** The subsequent code runs the `solve.QP`
function in the `quadprog` package. This fills in the portfolio standard deviations in
`tgt.sd` and portfolio weights in `wgt`.

```
> library(quadprog)
> for (i in 1:increments){
+     Dmat<-2*VCOV
+     dvec<-c(rep(0,length(avg.ret)))
+     Amat<-cbind(rep(1,length(avg.ret)),avg.ret,
+         diag(1,nrow=ncol(Ret.monthly)))
+     bvec<-c(1,tgt.ret[i],rep(0,ncol(Ret.monthly)))
+     soln<-solve.QP(Dmat,dvec,Amat,bvec=bvec,meq=2)
+     tgt.sd[i]<-sqrt(soln$value)
+     wgt[i,]<-soln$solution}
```

```
> head(tgt.sd)
[1] 0.00846 0.00822 0.00808 0.00795 0.00783 0.00775
> tail(tgt.sd)
[1] 0.0418 0.0426 0.0433 0.0441 0.0449 0.0457
> head(wgt)
                        [,1]   [,2]   [,3]                 [,4]
[1,] -0.0000000000000000244 1.000 0.00000  0.0000000000000004961
[2,]  0.0000000000000000000 0.974 0.00613  0.0199339540675214991
[3,] -0.0000000000000000906 0.972 0.01817  0.0101757214723419264
[4,] -0.0000000000000000839 0.969 0.03022  0.0004174888771624164
[5,] -0.0000000000000000647 0.960 0.04040 -0.0000000000000000163
[6,] -0.0000000000000000450 0.949 0.05051 -0.0000000000000000112
> tail(wgt)
                        [,1]                   [,2] [,3] [,4]
[95,]   0.32248458976308358  0.0000000000000000000 0.678    0
[96,]   0.25798767181046622  0.0000000000000000000 0.742    0
[97,]   0.19349075385784631  0.0000000000000000000 0.807    0
[98,]   0.12899383590523161  0.0000000000000000000 0.871    0
[99,]   0.06449691795261669  0.0000000000000000000 0.936    0
[100,]  0.00000000000000178 -0.0000000000000000278 1.000    0
```

Note that there are some observations that look like we have assigned negative
weights to certain securities although we have not allowed short sales. Those values
are so small they are virtually zero, so we should set those to zero. We will see that
when we allow short sales, the weights we calculate are going to be nontrivial.

```
> colnames(wgt)<-c("wgt.SPY","wgt.LAG","wgt.SLY","wgt.CWI")
> wgt[,1]<-ifelse(abs(wgt[,1])<=0.0000001,0,wgt[,1])
> wgt[,2]<-ifelse(abs(wgt[,2])<=0.0000001,0,wgt[,2])
> wgt[,3]<-ifelse(abs(wgt[,3])<=0.0000001,0,wgt[,3])
> wgt[,4]<-ifelse(abs(wgt[,4])<=0.0000001,0,wgt[,4])
> head(wgt)
     wgt.SPY wgt.LAG wgt.SLY  wgt.CWI
[1,]       0   1.000 0.00000 0.000000
[2,]       0   0.974 0.00613 0.019934
[3,]       0   0.972 0.01817 0.010176
[4,]       0   0.969 0.03022 0.000417
[5,]       0   0.960 0.04040 0.000000
[6,]       0   0.949 0.05051 0.000000
> tail(wgt)
      wgt.SPY wgt.LAG wgt.SLY wgt.CWI
[95,]   0.3225       0   0.678       0
[96,]   0.2580       0   0.742       0
[97,]   0.1935       0   0.807       0
[98,]   0.1290       0   0.871       0
[99,]   0.0645       0   0.936       0
[100,]  0.0000       0   1.000       0
```

Check to See Weights Still Sum to One

When hard coding entries to override results, it is good practice to check that the results still make sense. In the above example, we assert that the values should be 0 because they are so small and we have already assigned full weight of 100 % to one security. We can easily check by summing all the weights in each row using the `rowSums` command and output the results. As we can see, our check results in 100 observations with 1 as the sum of the weights for each observation.

```
> CHECK<-rowSums(wgt)
> CHECK
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [35] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [69] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Step 8: Combine Portfolio Returns, Portfolio Standard Deviations, and Portfolio Weights We now combine the different data objects we created, so we can see what combination of the four securities generates a specific portfolio return/portfolio standard deviation mix.

```
> tgt.port<-data.frame(cbind(tgt.ret,tgt.sd,wgt))
> head(tgt.port)
  tgt.ret  tgt.sd wgt.SPY wgt.LAG wgt.SLY  wgt.CWI
1 0.00258 0.00846       0   1.000 0.00000 0.000000
2 0.00271 0.00822       0   0.974 0.00613 0.019934
3 0.00284 0.00808       0   0.972 0.01817 0.010176
4 0.00296 0.00795       0   0.969 0.03022 0.000417
5 0.00309 0.00783       0   0.960 0.04040 0.000000
6 0.00321 0.00775       0   0.949 0.05051 0.000000
> tail(tgt.port)
    tgt.ret tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI
95   0.0144 0.0418  0.3225       0   0.678       0
96   0.0145 0.0426  0.2580       0   0.742       0
97   0.0147 0.0433  0.1935       0   0.807       0
98   0.0148 0.0441  0.1290       0   0.871       0
99   0.0149 0.0449  0.0645       0   0.936       0
100  0.0150 0.0457  0.0000       0   1.000       0
> no.short.tgt.port<-tgt.port
```

Note that we added the last line of code above so that we can save into a data object for later plotting the portfolio return and risk generated by the optimizer when short selling is not allowed. We will use this to compare with the results when short selling is allowed.

Step 9: Identify the Minimum Variance Portfolio We can now find the portfolio with the lowest standard deviation. This is known as the *minimum variance portfolio.* The output shows that the portfolio that gives us the smallest portfolio standard

deviation is the one 93 % invested in bonds (LAG) and 7 % invested in small capi-
talization stocks (SLY). This yields a portfolio standard deviation of 0.77 % with a
portfolio return of 0.35 %.

```
> minvar.port<-subset(tgt.port,tgt.port$tgt.sd==min(tgt.port$tgt.sd))
> minvar.port
  tgt.ret  tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI
8 0.00346 0.00767       0   0.929  0.0707       0
```

Step 10: Identify the Tangency Portfolio We first check that the `riskfree` vari-
able is still in R's memory. Then, we calculate the Sharpe Ratio for each of the
portfolios we constructed.

```
> riskfree
[1] 0.0000583
> tgt.port$Sharpe<-(tgt.port$tgt.ret-riskfree)/tgt.port$tgt.sd
> head(tgt.port)
  tgt.ret  tgt.sd wgt.SPY wgt.LAG  wgt.SLY  wgt.CWI Sharpe
1 0.00258 0.00846       0   1.000 0.00000 0.000000  0.299
2 0.00271 0.00822       0   0.974 0.00613 0.019934  0.322
3 0.00284 0.00808       0   0.972 0.01817 0.010176  0.344
4 0.00296 0.00795       0   0.969 0.03022 0.000417  0.365
5 0.00309 0.00783       0   0.960 0.04040 0.000000  0.387
6 0.00321 0.00775       0   0.949 0.05051 0.000000  0.407
> tail(tgt.port)
    tgt.ret tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI Sharpe
95   0.0144 0.0418  0.3225       0   0.678       0  0.343
96   0.0145 0.0426  0.2580       0   0.742       0  0.340
97   0.0147 0.0433  0.1935       0   0.807       0  0.337
98   0.0148 0.0441  0.1290       0   0.871       0  0.334
99   0.0149 0.0449  0.0645       0   0.936       0  0.331
100  0.0150 0.0457  0.0000       0   1.000       0  0.328
```

We then identify the portfolio with the highest Sharpe Ratio and that is the *tangency
portfolio.* This portfolio is the one with a 16 % investment in US large cap stocks
(SPY), 79 % investment in US bonds (LAG), and 5 % investment in small cap stocks
(SLY). The Sharpe Ratio of this portfolio is 0.527.

```
> tangency.port<-subset(tgt.port,tgt.port$Sharpe==max(tgt.port$Sharpe))
> tangency.port
   tgt.ret tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI Sharpe
19 0.00485 0.0091    0.16   0.793  0.0468       0  0.527
```

Step 11: Identify Efficient Portfolios For plotting purposes, we want to identify
the portfolios that offer the highest return for a given level of risk. These portfolios
are called *efficient portfolios*. The easiest way to identify these portfolios is that they
are all the portfolios that have returns higher than the return of the minimum variance
portfolio.

```
> eff.frontier<-subset(tgt.port,tgt.port$tgt.ret>=minvar.port$tgt.ret)
> eff.frontier[c(1:3,nrow(eff.frontier)),]
    tgt.ret  tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI Sharpe
8   0.00346 0.00767  0.0000   0.929  0.0707       0  0.444
9   0.00359 0.00768  0.0000   0.919  0.0808       0  0.460
10  0.00372 0.00771  0.0039   0.908  0.0876       0  0.474
100 0.01504 0.04571  0.0000   0.000  1.0000       0  0.328
```

Step 12: Plot the MV Efficient Frontier We now plot the efficient frontier of this four asset portfolio. Note that these portfolios are generated by the optimizer with a restriction on short selling.

```
> plot(x=tgt.sd,
+     y=tgt.ret,
+     col="gray40",
+     xlab="Portfolio Risk",
+     ylab="Portfolio Return",
+     main="Mean-Variance Efficient Frontier of Four Assets
+     Based on the Quadratic Programming Approach
+      (Not Allowing Short Selling)")
> abline(h=0,lty=1)
> points(x=minvar.port$tgt.sd,y=minvar.port$tgt.ret,pch=17,cex=3)
> points(x=tangency.port$tgt.sd,y=tangency.port$tgt.ret,pch=19,cex=3)
> points(x=eff.frontier$tgt.sd,y=eff.frontier$tgt.ret)
```

Figure 7.4 shows the MV Efficient Frontier of this Four-Asset Portfolio.

7.4   Effect of Allowing Short Selling

Before we proceed with running some code, let us take some time to think about what we should expect to see as the effect of allowing short selling. Note that by *not* allowing short selling we are adding a constraint that restricts the optimizer from putting negative weights on securities that could enhance return if we were allowed to take on short positions. As such, we should expect two things. First, we cannot be worse off when we do not impose a short selling restriction. Second, short selling should enhance returns in situations where taking a short position in a security can benefit us. Indeed, these results hold as we will see at the end of this section.

Now, let us turn to the portions of the code that need to be modified when short selling is allowed. No modification is necessary for Steps 1–3. We will begin the modifications with Step 4 of the last section.

Step 4ss: Construct the Target Portfolio Return Vector (Modifying to Allow for Short Selling) The portion of Step 4 that is different has to do with how we construct the bounds of the portfolio return vector. Recall that when short selling was not allowed, we can set limits on the portfolio returns by the maximum and minimum average return from the four securities. However, when short selling is allowed, we can in theory sell the other securities and invest the proceeds from the
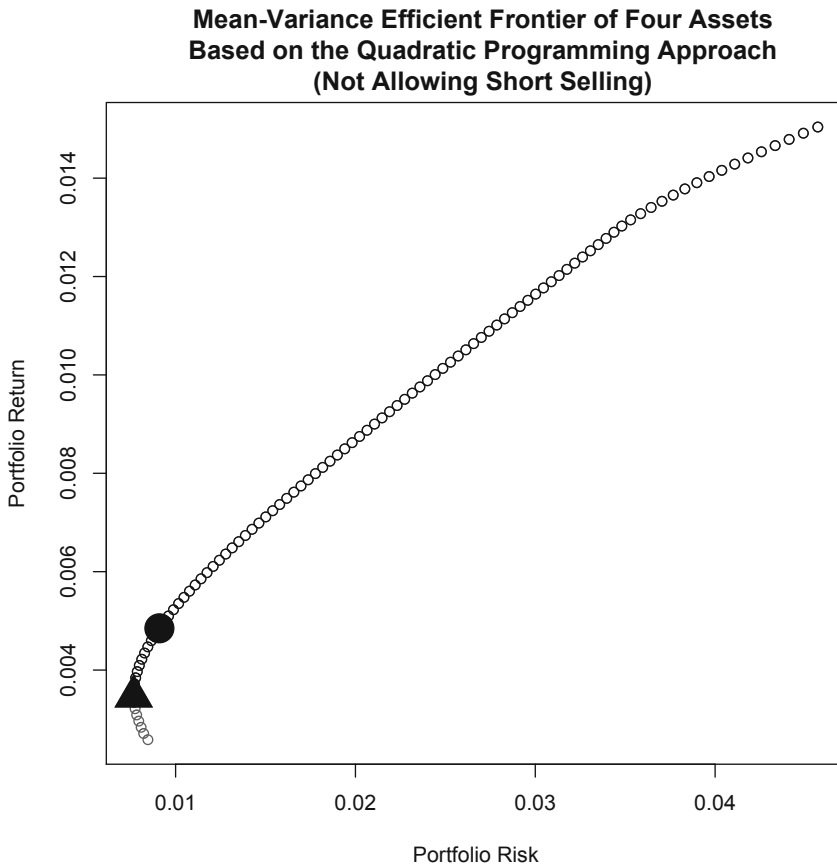
Mean-Variance Efficient Frontier of Four Assets
Based on the Quadratic Programming Approach
(Not Allowing Short Selling)

Fig. 7.4 Mean variance efficient frontier of portfolios consisting of SPY, LAG, CWI, and SLY stock without allowing short selling

sale of those securities in the security with the largest return. For the upper bound, we will construct it to be twice the maximum return of the four securities in our portfolio. This number is arbitrary but gives us a good enough range of portfolio returns for our purpose.

```
> tgt.ret<-seq(min.ret,max.ret*2,length=increments)
> head(tgt.ret)
[1] 0.00258 0.00286 0.00314 0.00342 0.00370 0.00397
> tail(tgt.ret)
[1] 0.0287 0.0290 0.0292 0.0295 0.0298 0.0301
```

Step 5ss: Construct Dummy Portfolio Standard Deviation Vector There is no change here.

Step 6ss: Construct Dummy Portfolio Weights Vector There is no change here.

Step 7ss: Run the `quadprog` **Optimizer (Modifying to Allow for Short Selling)**
The two changes are in the `Amat` and `bvec` lines. What we took out are terms related
to the third constraint placed on the optimizer, which is that the weights have to be
greater or equal to 0.

```
> library(quadprog)
> for (i in 1:length(tgt.ret)){
+     Dmat<-2*VCOV
+     dvec<-c(rep(0,length(avg.ret)))
+     Amat<-cbind(rep(1,length(avg.ret)),avg.ret)
+     bvec<-c(1,tgt.ret[i])
+     soln<-solve.QP(Dmat,dvec,Amat,bvec=bvec,meq=2)
+     tgt.sd[i]<-sqrt(soln$value)
+     wgt[i,]<-soln$solution}
> head(tgt.sd)
[1] 0.00821 0.00794 0.00771 0.00751 0.00734 0.00721
> tail(tgt.sd)
[1] 0.0479 0.0485 0.0490 0.0496 0.0501 0.0506
> head(wgt)
      wgt.SPY wgt.LAG wgt.SLY  wgt.CWI
[1,] -0.13872   1.004   0.113  0.02233
[2,] -0.11265   0.994   0.117  0.00217
[3,] -0.08657   0.983   0.121 -0.01799
[4,] -0.06049   0.973   0.126 -0.03814
[5,] -0.03442   0.963   0.130 -0.05830
[6,] -0.00834   0.953   0.134 -0.07846
> tail(wgt)
       wgt.SPY  wgt.LAG wgt.SLY wgt.CWI
[95,]     2.31  0.04094   0.519   -1.87
[96,]     2.34  0.03069   0.523   -1.89
[97,]     2.36  0.02045   0.528   -1.91
[98,]     2.39  0.01020   0.532   -1.93
[99,]     2.42 -0.00004   0.536   -1.95
[100,]    2.44 -0.01028   0.541   -1.97
```

As we can see, the negative weights placed on certain assets are very far from zero.
For example, portfolio number 100 has 244 % of the portfolio weight in SPY, which
is financed by short selling CWI by an amount equal to 197 % of the value of the
portfolio. To convince ourselves, we may want to check that all the weights still sum
to one.

```
> CHECK.wgt<-rowSums(wgt)
> CHECK.wgt
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [37] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [73] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

**Step 8ss: Combine Portfolio Returns, Portfolio Standard Deviations, and
Portfolio Weights** There is no change in the code here.

```
> tgt.port<-data.frame(cbind(tgt.ret,tgt.sd,wgt))
> head(tgt.port)
   tgt.ret  tgt.sd wgt.SPY wgt.LAG wgt.SLY  wgt.CWI
1 0.00258 0.00821 -0.13872   1.004   0.113  0.02233
2 0.00286 0.00794 -0.11265   0.994   0.117  0.00217
3 0.00314 0.00771 -0.08657   0.983   0.121 -0.01799
4 0.00342 0.00751 -0.06049   0.973   0.126 -0.03814
5 0.00370 0.00734 -0.03442   0.963   0.130 -0.05830
6 0.00397 0.00721 -0.00834   0.953   0.134 -0.07846
> tail(tgt.port)
    tgt.ret tgt.sd wgt.SPY  wgt.LAG wgt.SLY wgt.CWI
95   0.0287 0.0479    2.31  0.04094   0.519   -1.87
96   0.0290 0.0485    2.34  0.03069   0.523   -1.89
97   0.0292 0.0490    2.36  0.02045   0.528   -1.91
98   0.0295 0.0496    2.39  0.01020   0.532   -1.93
99   0.0298 0.0501    2.42 -0.00004   0.536   -1.95
100  0.0301 0.0506    2.44 -0.01028   0.541   -1.97
> with.short.tgt.port<-tgt.port
```

We added the last line so that we can plot the portfolios constructed when the short
sale restrictions were lifted and compare the results to the case when short sales were
not allowed.

Step 9ss: Identify the Minimum Variance Portfolio The minimum variance port-
folio when short selling is allowed has a portfolio standard deviation of 0.71 % and
a portfolio return of 0.48 %. This has lower risk and higher return than the minimum
variance portfolio when short selling was not allowed. In addition, the mix of the
securities is also very different. When short selling was allowed, we short sold in-
ternational stocks (CWI) and invested 7 % of the portfolio in US large cap stocks
(SPY), 92 % in US bonds (LAG), and 15 % in small cap stocks (SLY). In contrast,
when short selling was not allowed, the minimum variance portfolio comprised of
93 % bonds (LAG) and 7 % large cap stocks (SLY).

```
> minvar.port<-subset(tgt.port,tgt.port$tgt.sd==min(tgt.port$tgt.sd))
> minvar.port
  tgt.ret  tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI
9 0.00481 0.00707  0.0699   0.922   0.147  -0.139
```

Step 10ss: Identify the Tangency Portfolio We now turn to calculating the tangency
portfolio, which is the portfolio with the highest Sharpe Ratio. As such, we first
calculate the Sharpe Ratios of all the portfolios we constructed.

```
> riskfree
[1] 0.0000583
> tgt.port$Sharpe<-(tgt.port$tgt.ret-riskfree)/tgt.port$tgt.sd
> head(tgt.port)
   tgt.ret  tgt.sd  wgt.SPY wgt.LAG wgt.SLY  wgt.CWI Sharpe
1 0.00258 0.00821 -0.13872   1.004   0.113  0.02233  0.308
2 0.00286 0.00794 -0.11265   0.994   0.117  0.00217  0.353
3 0.00314 0.00771 -0.08657   0.983   0.121 -0.01799  0.400
4 0.00342 0.00751 -0.06049   0.973   0.126 -0.03814  0.448
5 0.00370 0.00734 -0.03442   0.963   0.130 -0.05830  0.496
6 0.00397 0.00721 -0.00834   0.953   0.134 -0.07846  0.543
```

Now, we identify which of these portfolios yields the highest Sharpe Ratio. The tangency portfolio when short sales are allowed has a portfolio return of 0.76 % and portfolio standard deviation of 0.91 %. In contrast, the tangency portfolio when short sales were restricted had a portfolio return of 0.49 % and portfolio standard deviation of 0.91 %. In other words, for the same level of risk, we expect a higher return when the short sale restrictions is lifted. Moreover, the weighting scheme is different as CWI can now be short sold instead of being bounded by a 0 % weight. This results in a Sharpe Ratio of 0.829, which is higher than the 0.527 Sharpe Ratio of the tangency portfolio when short sales were not allowed.

```
> tangency.port<-subset(tgt.port,tgt.port$Sharpe==max(tgt.port$Sharpe))
> tangency.port
   tgt.ret  tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI Sharpe
19 0.00758 0.00908   0.331   0.819    0.19   -0.34  0.829
```

Step 11ss: Identify Efficient Portfolios For plotting purposes, we identify the efficient portfolios. These are the portfolios that offer the highest return for a given level of risk. We identify these portfolios as those portfolios with returns that are higher than the return of the minimum variance portfolio.

```
> eff.frontier<-subset(tgt.port,tgt.port$tgt.ret>=minvar.port$tgt.ret)
> eff.frontier[c(1:3,nrow(eff.frontier)),]
     tgt.ret  tgt.sd wgt.SPY wgt.LAG wgt.SLY wgt.CWI Sharpe
9    0.00481 0.00707  0.0699  0.9219   0.147  -0.139  0.672
10   0.00508 0.00711  0.0960  0.9117   0.151  -0.159  0.707
11   0.00536 0.00719  0.1220  0.9014   0.156  -0.179  0.738
100  0.03008 0.05065  2.4428 -0.0103   0.541  -1.973  0.593
```

Step 12ss: Plot the MV Efficient Frontier We now plot the efficient frontier of this four-asset portfolio when short selling is not restricted.

```
> plot(x=tgt.sd,
+     y=tgt.ret,
+     col="gray40",
+     xlab="Portfolio Risk",
+     ylab="Portfolio Return",
+     main="Mean-Variance Efficient Frontier of Four Assets
+     Based on the Quadratic Programming Approach
+     (Allowing Short Selling)")
> abline(h=0,lty=1)
> points(x=minvar.port$tgt.sd,y=minvar.port$tgt.ret,pch=17,cex=3)
> points(x=tangency.port$tgt.sd,y=tangency.port$tgt.ret,pch=19,cex=3)
> points(x=eff.frontier$tgt.sd,y=eff.frontier$tgt.ret)
```

Figure 7.5 shows the MV Efficient Frontier of the Four-Asset Portfolio when short selling restrictions are removed.
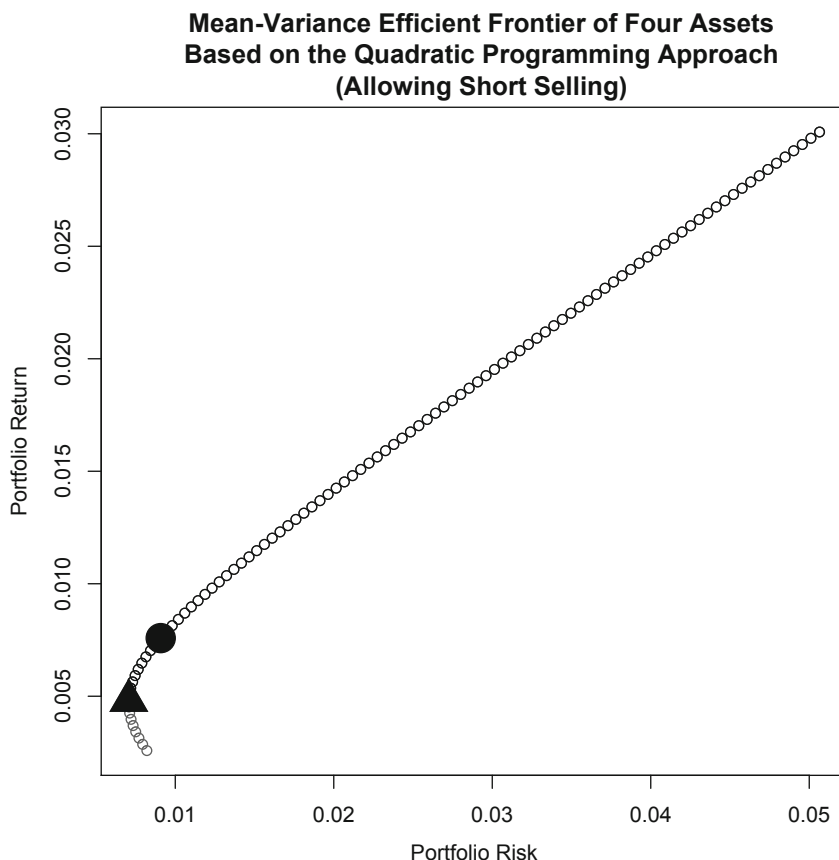
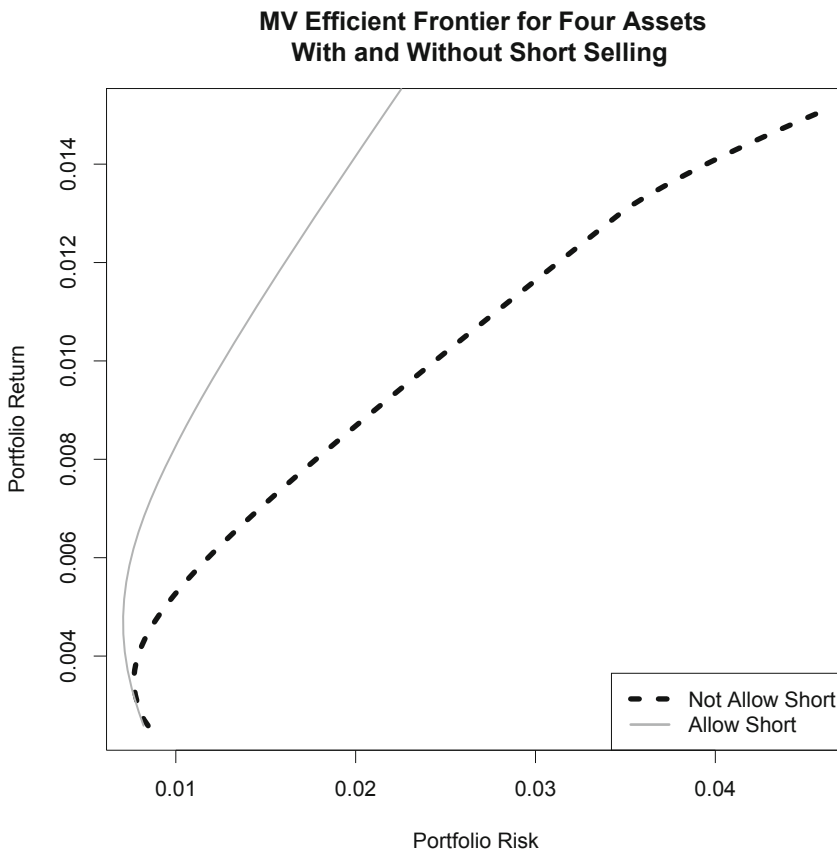**Fig. 7.5** Mean variance efficient frontier of portfolios consisting of SPY, LAG, CWI, and SLY stock allowing short selling. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com and Federal Reserve Electronic Database

Comparing the Efficient Frontier With and Without Short Sales

With two separate charts, it is hard to visualize the difference between the MV Efficient Frontier with an without short sales. So, we plot the two MV Efficient Frontiers together, which we show in Fig. 7.6. The black line represent the MV Efficient Frontier when short selling is not allowed. The gray line represents the MV Efficient Frontier when short selling is allowed. The results are consistent with our initial expectations. That is, the gray line (efficient frontier when short sale restrictions are lifted) attains higher levels of portfolio returns for the same level of risk as the black line (efficient frontier when short sales are not allowed) once we start moving to higher levels of portfolio risk. For lower levels of portfolio risk, relaxing the short sale restrictions does not enhance returns.

```
> plot(x=no.short.tgt.port$tgt.sd,
+     y=no.short.tgt.port$tgt.ret,
+     xlab="Portfolio Risk",
+     ylab="Portfolio Return",
+     type="l",
+     lwd=6,
+     lty=3,
+     main="MV Efficient Frontier for Four Assets
+     With and Without Short Selling")
> lines(x=with.short.tgt.port$tgt.sd,
+     y=with.short.tgt.port$tgt.ret,
+     col="gray60",
+     type="l",
+     lwd=2)
> legend("bottomright",
+     c("Not Allow Short","Allow Short"),
+     col=c("black","gray60"),
+     lty=c(3,1),
+     lwd=c(6,2))
```


Fig. 7.6 Comparison of mean variance efficient frontiers of portfolios consisting of SPY, LAG, CWI, and SLY stock allowing and not allowing short selling

7.5   Further Reading

More details on the Sharpe Ratio are found in Sharpe [4] and [5]. Markowitz [3] is the original paper on portfolio selection. A more comprehensive and modern treatment of asset allocation and portfolio management can be found in Maginn et al. [2]. Additional techniques used to calculate the efficient frontier are found in Elton et al. [1].

References

1. Elton, E., Gruber, M., Brown, S., & Goetzmann, W. (2010). *Modern portfolio theory and investment analysis* (8th ed.). New Jersey: Wiley.
2. Maginn, J., Tuttle, D., Pinto, J., & McLeavey, D. (2007). *Managing investment portfolios: A dynamic process* (3rd ed.). New Jersey: Wiley.
3. Markowitz, H. (1952). Portfolio selection. *The Journal of Finance, 7,* 77–91.
4. Sharpe, W. (1975). Adjusting for risk in portfolio performance. *Journal of Portfolio Management, 1,* 29–34.
5. Sharpe, W. (1994). The Sharpe ratio. *Journal of Portfolio Management, 21,* 49–58.

Chapter 8
Fixed Income

In prior chapters, our analysis focused primarily on equity securities as examples. However, based on size, the US stock market is actually smaller than the US fixed income market. In 2012, the US bond market stood at $ 37 trillion, while the US equity market stood at $ 21 trillion.[1] In this chapter, our analysis focuses on fixed income securities as examples. Many of the techniques used in this chapter are also equally applicable to both equity and fixed income securities.

We begin our analysis of fixed income securities by showing how to obtain and analyze economic data. Many investments have a relationship with macroeconomic variables, which makes analyzing economic data important to understanding how our investments may perform during our holing period. We then demonstrate an analysis of Treasury yields. Since other fixed income instruments rely on the rates of Treasury securities, understanding the shape of the yield curve and the slope of the yield curve is essential to making sound fixed income investment decisions. We also show how to look at the real yield of Treasury securities and observe the decline in real yields in recent years. Then, we demonstrate how to identify mean reverting patterns in Treasury securities. Next, we analyze the time series of spreads between corporates of different investment grade ratings. We show how such an analysis can reveal the widening or tightening of credit spreads.

The second part of this chapter demonstrates discounted cash flow bond valuation. First, we look at the simple case of valuing a bond on a coupon payment date. We implement the calculation in two ways, with the first laying out every step and the second by showing how to create a function so we can repeatedly calculate bond prices with relative ease. Second, we implement a bond valuation calculation on non-coupon payment dates. The main difference in this latter approach is we need to add accrued interest. We also demonstrate how to calculate duration and convexity of bonds, which are tools used to manage interest rate risk.

---

[1] http://finance.zacks.com/bond-market-size-vs-stock-market-size-5863.html.

	A	B	C	F	G	AQ	AR	AS
1	Country	Subject Descriptor	Units	1980	1981	2017	2018	Estimates Start After
2	United States	Gross domestic product, constant prices	Percent change	-0.245	2.595	3.364	3.066	2012
3								
4	International Monetary Fund, World Economic Outlook Database, October 2013							

Fig. 8.1 Screenshot of IMF WEO data on US Real GDP. IMF data reproduced with permission. Data source: International Monetary Fund World Economic Outlook October 2013, http://www.imf.org/external/pubs/ft/weo/2013/02/weodata/index.aspx, retrieved on January 17, 2014

8.1   Economic Analysis

When making investments, understanding how the economy is doing is essential to the success of our investment strategy. For example, when interest rates rise, demand for fixed income securities may increase. However, there are too many economic indicators to discuss in this text and different indicators are more important for some sectors compared to others. As such, for illustrative purposes, we only analyze three economic indicators: Real GDP, unemployment, and inflation. The sources and techniques used in this section can be applied to identify and analyze other economic data.

8.1.1   Real GDP

We first discuss Real GDP, which is an indicator of the strength of the economy. Most securities tend to move with the overall economy. When the economy does well, securities prices tend to increase. Conversely, when the economy does poorly, securities prices tend to decrease. As such, it is important to know where we are in the economic cycle to determine the best investments that fit our strategy.

An analysis we can perform is to analyze the growth in Real GDP. Below, we will construct a *bar chart* of historical and projected Real GDP growth using data retrieved from the IMF website. In particular, we obtain the data from the October 2013 World Economic Outlook.[2] This database contains many more information that what we use here and is a very useful source of economic data. As of early 2014, the US Real GDP growth data contains actual historical results from 1980 through 2012 and projected results from 2013 through 2018. From a presentation perspective, a key element for this analysis is to distinguish between historical results and projected results. For that purpose, we will construct a chart that uses different colored bars for the historical data and for the projected data.

---

[2] http://www.imf.org/external/pubs/ft/weo /2013/02/weodata/index.aspx.

Step 1: Import Historical and Projected Real GDP Growth Data from the IMF Website The historical real GDP is from 1980 to 2012 and the data contains projections from 2013 to 2018. The Excel file retrievable from the IMF website has one worksheet. As such, we open the file in Excel and save that single worksheet into a CSV file labeled **USRGDP IMF WEO.csv**. Figure 8.1 displays the data as you would observe in Excel but hides many of the intermediate columns. As we can see, the data we are interested in is the first two rows and from columns six (Column F) through 44 (Column AR). Hence, we subset the data to only those rows and columns. Then, we can transpose the data using the `t(...)` operator. Transposing converts data in rows to data in columns or vice versa. In our example, instead of having a data object 2 rows and 39 columns we now have a data object with 39 rows and 2 columns after we transpose the data.

```
> us.rgdp<-read.csv("USRGDP IMF WEO.csv",header=FALSE)
> us.rgdp<-us.rgdp[1:2,6:44]
> us.rgdp<-t(us.rgdp)
> us.rgdp[c(1:3,nrow(us.rgdp)),]
       1      2
V6  1980 -0.245
V7  1981  2.595
V8  1982 -1.911
V44 2018  3.066
```

The index of us.rgdp reported above indicate the original column names. That is, V6 is the 6th column and V44 is the 44th column.

> **Updates to the IMF World Economic Outlook** The IMF World Economic Outlook data is updated several times a year. We retrieved the above data in January 2014, which uses the September 2013 WEO data. On a later date, the IMF website may have updated data from a later WEO publication. This means that the data in the projection period above (i.e., 2013–2018) could change as more actual data is recorded and projections are extended out to later years. The historical data should likely stay the same.

Step 2: Rename Variables and Fix Index of Data Object The variable names above are "1" and "2," which is not informative. Similarly, the index of V6 to V44 are also not informative. Hence, we change the variable names to "Year" and "Value" and change the index labels to denote the observation number in the data object.

```
> colnames(us.rgdp)<-paste(c("Year","Value"))
> rownames(us.rgdp)<-seq(1,nrow(us.rgdp),1)
> us.rgdp<-data.frame(us.rgdp)
> us.rgdp[c(1:3,nrow(us.rgdp)),]
   Year  Value
1  1980 -0.245
2  1981  2.595
3  1982 -1.911
39 2018  3.066
```

Step 3: Create Separate Variable for Historical Real GDP Growth Data To distinguish between historical and projected US Real GDP growth data, we will use a *stacked bar chart*. The historical data will have non-zero (i.e., positive or negative values) data through 2012 and zero from 2013–2018, while the projected data will have zero through 2012 and non-zero data from 2013–2018. As such, we need to create two variables that implement our approach above.

```
> us.rgdp$historical<-ifelse(us.rgdp$Year<=2012,us.rgdp$Value,0)
> us.rgdp[c(1:3,nrow(us.rgdp)),]
   Year  Value historical
1  1980 -0.245     -0.245
2  1981  2.595      2.595
3  1982 -1.911     -1.911
39 2018  3.066      0.000
>
> us.rgdp$projected<-ifelse(us.rgdp$Year>2012,us.rgdp$Value,0)
> us.rgdp[c(1:3,nrow(us.rgdp)),]
   Year  Value historical projected
1  1980 -0.245     -0.245     0.000
2  1981  2.595      2.595     0.000
3  1982 -1.911     -1.911     0.000
39 2018  3.066      0.000     3.066
```

Step 4: Setup Data for Transposition and Transpose the Data The simplest way to make sure we transpose data the correct way for the variables we intend to use is to limit the data only to those variables. In this case, we really only care about the `historical` and `projected` variables. As such, we limit the data to only Columns 3 and 4. We then convert the data by first converting `us.rgdp` into a matrix using the `as.matrix` command and then using the `t(...)` operator to transpose the data from 39 rows by 2 columns to 2 rows by 39 columns.

```
> us.rgdp<-us.rgdp[,3:4]
> us.rgdp[c(1:3,nrow(us.rgdp)),]
   historical projected
1      -0.245     0.000
2       2.595     0.000
3      -1.911     0.000
39      0.000     3.066
>
> us.mat<-as.matrix(us.rgdp)
> t.us<-t(us.mat)
> head(t.us)
                1     2      3     4     5     6     7     8     9   10    11
historical -0.245 2.595 -1.911 4.633 7.259 4.239 3.512 3.462 4.204 3.68 1.919
projected   0.000 0.000  0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00 0.000
               12    13    14    15    16    17    18   19    20    21    22
historical -0.073 3.555 2.745 4.037 2.719 3.796 4.487 4.45 4.846 4.091 0.949
projected   0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.00 0.000 0.000 0.000
               23    24    25    26    27   28     29     30    31    32    33
historical 1.776 2.791 3.798 3.351 2.667 1.79 -0.291 -2.802 2.507 1.847 2.779
projected  0.000 0.000 0.000 0.000 0.000 0.00  0.000  0.000 0.000 0.000 0.000
              34    35   36    37    38    39
historical 0.00 0.000 0.00 0.000 0.000 0.000
projected  1.56 2.588 3.35 3.476 3.364 3.066
```

Step 6: Setup Sequence of Even Years to Customize x-axis With close to 40 years of data, the x-axis gets too crowded if we have a label for each year. Since the data starts in 1980, we keep only even numbered years. To implement this, we first create `xlabel`, which is a sequence of years from 1980 to 2018. Then, we use the `%% 2==0` to identify which elements are even numbers. We then change every odd year value to a blank character.

```
> xlabel=seq(1980,2018,by=1)
> even.years<-xlabel %% 2==0
> even.years
 [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
[13]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
[25]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
[37]  TRUE FALSE  TRUE
> years.even<-cbind(data.frame(xlabel),data.frame(even.years))
> head(years.even)
  xlabel even.years
1   1980       TRUE
2   1981      FALSE
3   1982       TRUE
4   1983      FALSE
5   1984       TRUE
6   1985      FALSE
> years.even$Year<-ifelse(years.even$even.years=="TRUE",xlabel," ")
> xlabel.even<-years.even[,3]
> head(xlabel.even)
[1] "1980" " "    "1982" " "    "1984" " "
```

Step 7: Plot a Bar Chart of Annual Real GDP Growth Rate To generate a bar chart in R we use the `barplot` command. The last two arguments relate to the x-axis. The `las=2` argument flips the text of the x-axis 90°. The `names.arg` argument tells R to use the `xlabel.even` sequence for the x-axis.

```
> range(us.rgdp)
[1] -2.802  7.259
> y.range<-c(-4,8)
> y.range
[1] -4  8
> barplot(t.us,col=c("black","gray60"),
+    ylab="Real GDP Growth (%)",
+    ylim=y.range,
+    names.arg=xlabel.even,
+    las=2,
+    main="United States Real GDP Growth
+    Historical (1980-212) and Forecasted (2013-2018)")
> legend("topright",
+    c("Historical","Forecasted"),
+    col=c("black","gray60"),
+    pch=c(15,15))
```
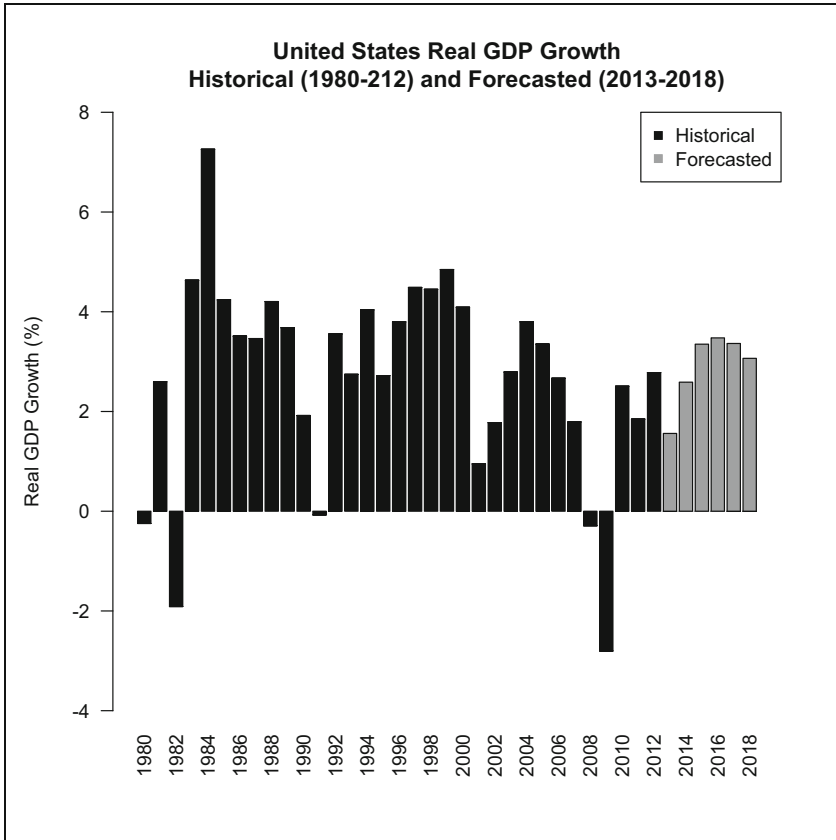
Fig. 8.2 Historical and projected US Real GDP growth rate, 1980–2013. IMF data reproduced with permission. Data Source: International Monetary Fund World Economic Outlook October 2013, http://www.imf.org/external/pubs/ft/weo/2013/02/weodata/index.aspx, retrieved on January 17, 2014

Figure 8.2 shows the plot of the US Real GDP growth rate from 1980 to 2018. As we can see, we start off with black bars from 1980 to 2012. The last five bars, which denote the forecasts from 2013 to 2018, are in gray. This presentation of the data makes it easy to distinguish which data are historical data and which data are projections.

8.1.2 Unemployment Rate

For the GDP data, we used historical and projected data from the International Monetary Fund. There is also a wealth of economic and financial data available on

the Federal Reserve Electronic Database (FRED). One of the many available data series on the FRED database is the US unemployment rate data.

For our example, we analyze the unemployment rate over the last 50 years and identify its peaks and long-term average. To emphasize the peaks and long-term average, we need to *annotate* the chart we create. Below we show how to implement this analysis.

Step 1: Import Unemployment Rate Data from FRED We are interested in the Civilian Unemployment Rate with symbol UNRATE from the United States Department of Labor. This is a monthly and seasonally adjusted series. We save the data as **UNRATE FRED.csv**. We create the data object `US.unempl`. We see that the start of the data is January 1948. The end of the data would depend on when we retrieved the data from FRED.

```
> US.unempl<-read.csv("UNRATE FRED.csv",skip=10)
> US.unempl$date<-as.Date(US.unempl$observation_date,"%Y-%m-%d")
> US.unempl$UNRATE<-as.numeric(as.character(US.unempl$UNRATE))
> US.unempl<-xts(US.unempl$UNRATE,order.by=US.unempl$date)
> names(US.unempl)<-paste("UNRATE")
> US.unempl[1:3,]
           UNRATE
1948-01-01   3.4
1948-02-01   3.8
1948-03-01   4.0
```

Step 2: Subset Data from 1964 to 2013 Since we are only concerned with the unemployment rate over the last 50 years, we should limit the data to years 1964–2013. To perform this filter, we use the `subset` command. We also use the `index` command to call out the date in the index and use that to subset the data.

```
> US.unempl<-subset(US.unempl,
+     index(US.unempl)>="1964-01-01" &
+     index(US.unempl)<="2013-12-31")
> US.unempl[1:3,]
           UNRATE
1964-01-01   5.6
1964-02-01   5.4
1964-03-01   5.4
```

Step 3: Calculate Long-Term Average Unemployment Rate We then calculate the average unemployment rate over the last 50 years, which is 6.1 %.

```
> lt.avg<-mean(US.unempl$UNRATE)
> lt.avg
[1] 6.1255
```

Step 4: Plot the Unemployment Rate Data We first plot the time series of historical data using the `plot` command.

```
> plot(x=index(US.unempl),
+     xlab="Date (Quarters)",
+     y=US.unempl$UNRATE,
+     ylab="Unemployment Rate (%)",
+     ylim=c(2,12),
+     type="l",
+     main="US Unemployment Rate
+     From 1964 to 2013")
```

We then overlay the annotations onto the chart. There are three points that we want to mark. First, we want to identify the long-term average. The abline command with an h option generates a horizontal line, which in this case is equal to the long-term average yield.

```
> abline(h=lt.avg,lty=2)
> text(as.Date("2001-01-01"),7.4,"Long-Term")
> text(as.Date("2001-01-01"),7,"Avg. = 6.1%")
> arrows(x0=as.Date("2001-01-01"),
+     y0=6.9,
+     x1=as.Date("2001-01-01"),
+     y1=6.2,
+     code=2,
+     length=0.10)
```

Second, we mark the point with the highest unemployment rate, which was in November 1982. The unemployment rate stood at 10.8 % during that month.

```
> points(as.Date("1982-11-01"),10.8,pch=16)
> text(as.Date("1992-01-01"),11.5,"November 1982")
> arrows(x0=as.Date("1992-01-01"),
+     y0=11.3,
+     x1=as.Date("1983-07-01"),
+     y1=10.9,
+     code=2,
+     length=0.10)
```

Third, we mark the point with the second highest unemployment rate, which was in October 2009. The unemployment rate stood at 10.0 % during that month.

```
> points(as.Date("2009-10-01"),10,pch=16)
> text(as.Date("2010-01-01"),11,"October 2009")
> arrows(x0=as.Date("2009-10-01"),
+     y0=10.8,
+     x1=as.Date("2009-10-01"),
+     y1=10.1,
+     code=2,
+     length=0.10)
```

Figure 8.3 shows our annotated unemployment rate chart. Note that the annotation itself has to be performed by trial-and-error, until we get a really good feel for how to create the arrows and text boxes.
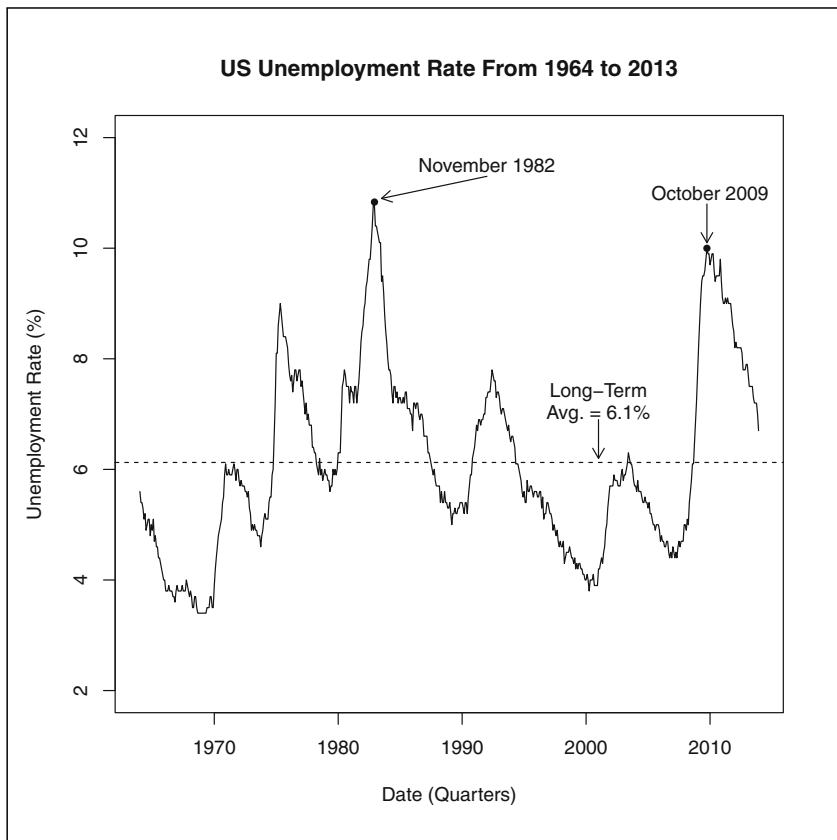
Fig. 8.3 US inflation rate, 1964–2013. Data Source: Federal Reserve Electronic Database

Retrieving FRED Data Directly Using `getSymbols`

Again, although not recommended, there is another way in which you can get FRED data. I have written Step 1 above to make the resulting data object identical to this alternative method because some users would prefer to use this more automated approach. To get the result above, we need to use the `getSymbols` command in the `quantmod` package. Unlike Yahoo Finance data, we cannot choose the date range over which we download the FRED data. We also have to explicitly tell R to look at the FRED database by including the `src=FRED` argument. To get the data for unemployment we will use the following code:

```
> library(quantmod)
>
> alt.unrate<-getSymbols("UNRATE",src="FRED",auto.assign=FALSE)
>
```

```
> class(alt.unrate)
[1] "xts" "zoo"
> class(US.unempl)
[1] "xts" "zoo"
```

If you use the `head` and `tail` commands on `alt.unrate`, you will get the same result as above. Note that you have to remember to add the `auto.assign=FALSE` string because you will not be able to see any output when you use the `head` or `tail` commands if you do not. Moreover, the `class` command verifies that the class of the data object is also `xts` and `zoo`.

8.1.3 Inflation Rate

Another example of key economic data available on FRED is the US Consumer Price Index (CPI) data, which is commonly-used to calculate the inflation rate. The inflation rate is one of the most important factors investors consider in determining their expected rates of return. A higher inflation rate would lead to a higher required expected rate of return, as a dollar tomorrow is worth less than a dollar is worth today.

The Bureau of Labor Statistics of the US Department of Labor reports the Consumer Price Index (CPI) data every month. The year-over-year changes in the CPI is typically used as a common measure of the inflation rate. The year-over-year change means that the inflation rate for a particular month is equal to the percentage change in the CPI for that same month last year.

For our example, we look at the time series of inflation rates based on year-over-year changes in CPI over the last 50 years. We then overlay the recession periods as identified by the NBER, so we can analyze whether any patterns exist between the inflation rate and recessions.

Step 1: Import CPI Data from FRED The measure of CPI we use is the CPI for all urban consumers, which has the symbol CPIAUCNS from FRED. This is monthly data that is not seasonally adjusted. Note that the data begins in 1913. We save the FRED data as **CPIAUCNS FRED.csv**, which as of January 2014 contains data through December 2013. Since the data on FRED gets continuously updated, pulling the data at a later date will result in retrieving data that includes data for more recent periods. We convert the CSV file containing the raw FRED data to an `xts` object using techniques previously discussed.

```
> US.CPI<-read.csv("CPIAUCNS FRED.csv",skip=10)
> US.CPI$date<-as.Date(US.CPI$observation_date,"%Y-%m-%d")
> US.CPI$CPIAUCNS<-as.numeric(as.character(US.CPI$CPIAUCNS))
> US.CPI<-xts(US.CPI$CPIAUCNS,order.by=US.CPI$date)
> names(US.CPI)<-paste("CPIAUCNS")
> US.CPI[1:3,]
            CPIAUCNS
1913-01-01      9.8
1913-02-01      9.8
1913-03-01      9.8
```

Step 2: Calculate a 12-Month Lag Variable Since the inflation rate is calculated as year-over-year changes in CPI, we create a variable that takes on the value of the CPI during the same month the year before. As we have monthly data, this requires us to take a 12-month lag of the data. As such, we use the `Lag` command with `k=12` to denote the 12-period lag.

```
> US.Lag12<-Lag(US.CPI$CPIAUCNS,k=12)
> US.Lag12[1:20,]
            Lag.12
1913-01-01      NA
1913-02-01      NA
1913-03-01      NA
1913-04-01      NA
1913-05-01      NA
1913-06-01      NA
1913-07-01      NA
1913-08-01      NA
1913-09-01      NA
1913-10-01      NA
1913-11-01      NA
1913-12-01      NA
1914-01-01     9.8
1914-02-01     9.8
1914-03-01     9.8
1914-04-01     9.8
1914-05-01     9.7
1914-06-01     9.8
1914-07-01     9.9
1914-08-01     9.9
```

Visually inspecting the data is important at this point to ensure we are properly matching the lag data to the appropriate period. From the output in Step 1, we know January 1913 CPI is 9.8 %, which matches the January 1914 12-Month Lag CPI value (`Lag.12`) reported above.

Step 3: Combine CPI and Lag CPI Data Using the `merge` command, we combine the `US.CPI` and tt US.Lag12 data objects. To make the variables easier to remember, we rename them the CPI to `us.cpi` and lag CPI to `lag.cpi`. We show only the 10th through 15th observation below, which displays the crossover period from where we do not have lag CPI data and when we have lag CPI data. Since the raw FRED data

begins in January 1913, we should expect the first observation which has a 12-period lag CPI data would be January 1914. The output below confirms this expectation.

```
> US.CPI<-merge(US.CPI,US.Lag12)
> names(US.CPI)<-paste(c("us.cpi","lag.cpi"))
> US.CPI[10:15,]
           us.cpi lag.cpi
1913-10-01   10.0      NA
1913-11-01   10.1      NA
1913-12-01   10.0      NA
1914-01-01   10.0     9.8
1914-02-01    9.9     9.8
1914-03-01    9.9     9.8
```

Step 4: Calculate Inflation Rate   We can then calculate the inflation rate as the percentage change between `us.cpi` and `lag.cpi` for each period in which we have both values populated. This means that we can calculate an inflation rate beginning January 1914. The output below calculates the inflation rate in *percentage point* terms by multiplying the values by 100, which means that the January value of 2.04 is interpreted as 2.04 %. The use of percentage points makes reading graph labels easier.

```
> US.CPI$inflation<-(US.CPI$us.cpi/US.CPI$lag.cpi-1)*100
> US.CPI[10:15,]
           us.cpi lag.cpi inflation
1913-10-01   10.0      NA        NA
1913-11-01   10.1      NA        NA
1913-12-01   10.0      NA        NA
1914-01-01   10.0     9.8  2.040816
1914-02-01    9.9     9.8  1.020408
1914-03-01    9.9     9.8  1.020408
```

Step 5: Subset Inflation Data Over the Last 50 Years   Since we are interested in the last 50 years of data, we use the `subset` command to limit the data to values from January 1964 to December 2013. In this step, we take the opportunity to clean up the data and only keep the `inflation` variable.

```
> US.CPI<-subset(US.CPI[,3],
+     index(US.CPI)>="1964-01-01" &
+     index(US.CPI)<="2013-12-01")
> US.CPI[c(1:3,nrow(US.CPI)),]
             inflation
1964-01-01  1.644737
1964-02-01  1.644737
1964-03-01  1.311475
2013-12-01  1.501736
```

Step 6: Plot the Inflation Rate Data   We then plot the inflation rate using the `plot` command. The first set of codes creates a simple plot of the inflation rate data.

```
> plot(x=index(US.CPI),
+     y=US.CPI$inflation,
+     xlab="Date",
+     ylab="Inflation Rate (%)",
+     type="l",
+     main="US Inflation Rates From 1964 to 2013
+     Based on Year Over Year Changes in CPI")
```

The next set of codes creates the *shaded areas* to identify the recession periods within our date range. We use recession period dates identified by the NBER Dating Commmittee. We then add shading to the chart in two steps. The first step is to use the `par("usr")` command to report the coordinates of the chart. Then, we use the `rect` command to draw a rectangle, where the coordinates are given by (i) our first date, (ii) the coordinates for the top of the chart (`shade[2]`), (iii) our last date, and (iv) the coordinates for the bottom of the chart (`shade[3]`). The `box` argument then redraws the borders of the chart that are partially overwritten when we shaded the recession periods. Note that the recession period shading overlaps with the inflation rate plot. However, we needed to plot the inflation rate first to get the boundaries of the plot so that we can create the rectangular boxes needed by the recession period shading.

```
> shade<-par("usr")
> shade
[1] -2921.280000 16769.280000    -2.771306     15.430591
> rect(as.Date("2007-12-01"),shade[2],
+     as.Date("2009-06-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("2001-03-01"),shade[2],
+     as.Date("2001-11-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("1990-07-01"),shade[2],
+     as.Date("1991-03-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("1981-07-01"),shade[2],
+     as.Date("1982-11-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("1980-01-01"),shade[2],
+     as.Date("1980-07-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("1973-11-01"),shade[2],
+     as.Date("1975-03-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("1969-12-01"),shade[2],
+   as.Date("1970-11-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("1960-04-01"),shade[2],
+   as.Date("1961-02-01"),shade[3],col="gray60",lty=0)
> box(which="plot",lty=1)
```

As a solution to the recession period shading overlapping with the inflation rate line, we can fix this by redrawing the CPI change line using the `lines` command. We then add a horizontal line to denote a 0 % inflation rate as a frame of reference when interpreting the chart.

```
> lines(x=index(US.CPI),y=US.CPI$inflation)
> abline(h=0)
```

US Inflation Rates From 1964 to 2013
Based on Year Over Year Changes in CPI
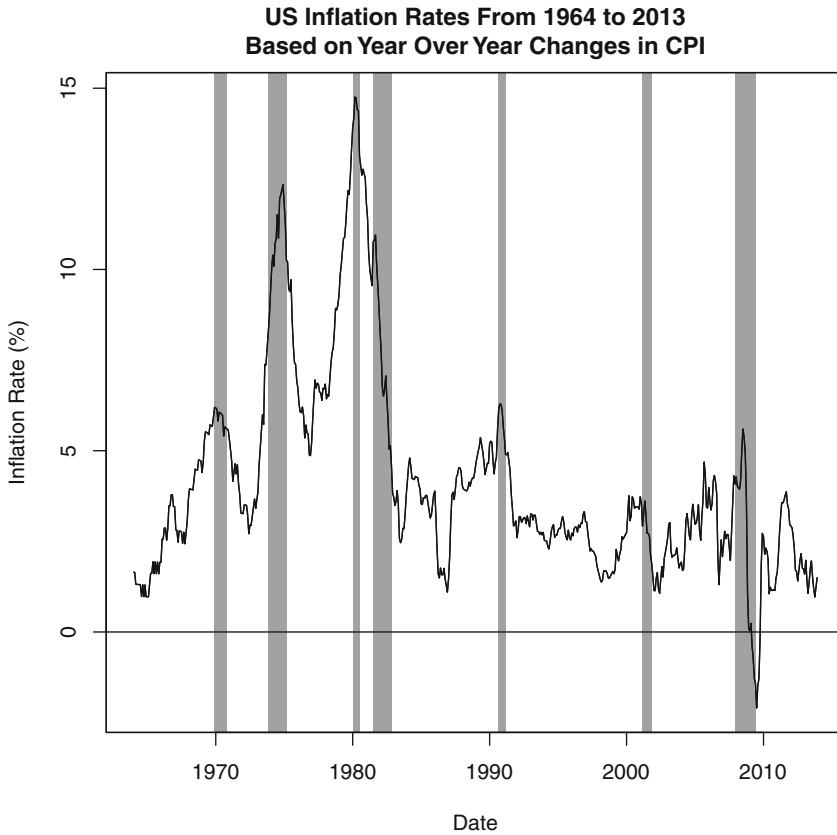
Fig. 8.4 US inflation rate and recessions, 1964–2013. Data Source: Federal Reserve Electronic Database and NBER Business Cycle Dates

Figure 8.4 shows the inflation chart from 1964 to 2013. We can see that inflation generally increases during recession periods and typically decreases after. Over the last 50 years, there were two instances during which inflation was higher than 10 %. These were in the mid-1970s and early 1980s. In the latest recession, we can see that the decline in inflation rates after the recession was severe enough to cause the inflation rate to become negative from March to October 2009.

8.2   US Treasuries

8.2.1   Shape of the US Treasury Yield Curve

The rates of US Treasury securities form the benchmark for the rates of other fixed income instruments. As such, knowledge of Treasury yields is required to determine whether the yields of other fixed income securities are large enough for the incremental risk we are taking on. In addition, trading strategies can be constructed around the differential between short rates and long-term rates. If T-bill rates for 3-months is substantially lower than 6-month rates, an investor who expects to hold a T-Bill for 3 months may instead purchase a 6-month T-Bill and sell it in 3 months to take advantage of the higher yields on the longer dated instrument. This strategy is known as *riding the yield curve*.

The Treasury Yield Curve is a curve comprised of rates on key maturities of US Treasury securities. The yield curve can take on many shapes. It can be upward sloping, which means that short-term rates are lower than long-term rates. An upward sloping yield curve is also called "normal," which implies this is the most typical shape of the yield curve we observe. The yield curve can be inverted, which means that the short-term rates are higher than long-term rates. The yield curve can also be flat, which means that the short-term rates are almost the same as the long-term rates. In this section, we demonstrate how to find examples of these different shapes of the yield curve and plot these examples.

Step 1: Obtaining US Treasury Yields for Key Maturities To construct the Treasury yield curve, we have to first obtain the yields of Treasury securities of different maturities. For our example, we use the following key maturities: 3 months (DGS3MO), 6 months (DGS6MO), 1 year (DGS1), 2 years (DGS2), 3 years (DGS3), 5 years (DGS5), 7 years (DGS7), 10 years (DGS10), 20 years (DGS20), and 30 years (DGS30). The text in the parentheses are the FRED symbols for these Treasury securities. We download CSV files from the Federal Reserve Electronic Database (FRED) website for each of the yields for each key rate. When on the FRED website, we can type the symbols for each security, which brings us to the relevant page. The FRED website provides access through an Excel file with one worksheet that contains the relevant yields for all available data for that security. From a practical perspective, it is harder to deal with Excel files in R. As such, we save that single worksheet as a CSV file in our R working directory. We end up with the following files: **DGS3MO FRED.csv**, **DGS6MO FRED.csv**, **DGS1 FRED.csv**, **DGS3 FRED.csv**, **DGS5 FRED.csv**, **DGS7 FRED.csv**, **DGS10 FRED.csv**, **DGS20 FRED.csv**, and **DGS30 FRED.csv**.

```
> t3mo<-read.csv("DGS3MO FRED.csv",skip=10)
> t3mo$date<-as.Date(t3mo$observation_date,"%Y-%m-%d")
> t3mo$DGS3MO<-as.numeric(as.character(t3mo$DGS3MO))
Warning message:
NAs introduced by coercion
> t3mo<-xts(t3mo$DGS3MO,order.by=t3mo$date)
> names(t3mo)<-paste("DGS3MO")
> t3mo[1:3,]
           DGS3MO
1982-01-04  11.87
1982-01-05  12.20
1982-01-06  12.16
>
> t6mo<-read.csv("DGS6MO FRED.csv",skip=10)
> t6mo$date<-as.Date(t6mo$observation_date,"%Y-%m-%d")
> t6mo$DGS6MO<-as.numeric(as.character(t6mo$DGS6MO))
Warning message:
NAs introduced by coercion
> t6mo<-xts(t6mo$DGS6MO,order.by=t6mo$date)
> names(t6mo)<-paste("DGS6MO")
> t6mo[1:3,]
           DGS6MO
1982-01-04  13.16
1982-01-05  13.41
1982-01-06  13.46
>
> t1yr<-read.csv("DGS1 FRED.csv",skip=10)
> t1yr$date<-as.Date(t1yr$observation_date,"%Y-%m-%d")
> t1yr$DGS1<-as.numeric(as.character(t1yr$DGS1))
Warning message:
NAs introduced by coercion
> t1yr<-xts(t1yr$DGS1,order.by=t1yr$date)
> names(t1yr)<-paste("DGS1")
> t1yr[1:3,]
           DGS1
1962-01-02 3.22
1962-01-03 3.24
1962-01-04 3.24
>
> t2yr<-read.csv("DGS2 FRED.csv",skip=10)
> t2yr$date<-as.Date(t2yr$observation_date,"%Y-%m-%d")
> t2yr$DGS2<-as.numeric(as.character(t2yr$DGS2))
Warning message:
NAs introduced by coercion
> t2yr<-xts(t2yr$DGS2,order.by=t2yr$date)
> names(t2yr)<-paste("DGS2")
> t2yr[1:3,]
           DGS2
1976-06-01 7.26
1976-06-02 7.23
1976-06-03 7.22
```

```
>
> t3yr<-read.csv("DGS3 FRED.csv",skip=10)
> t3yr$date<-as.Date(t3yr$observation_date,"%Y-%m-%d")
> t3yr$DGS3<-as.numeric(as.character(t3yr$DGS3))
Warning message:
NAs introduced by coercion
> t3yr<-xts(t3yr$DGS3,order.by=t3yr$date)
> names(t3yr)<-paste("DGS3")
> t3yr[1:3,]
          DGS3
1962-01-02 3.70
1962-01-03 3.70
1962-01-04 3.69
>
> t5yr<-read.csv("DGS5 FRED.csv",skip=10)
> t5yr$date<-as.Date(t5yr$observation_date,"%Y-%m-%d")
> t5yr$DGS5<-as.numeric(as.character(t5yr$DGS5))
Warning message:
NAs introduced by coercion
> t5yr<-xts(t5yr$DGS5,order.by=t5yr$date)
> names(t5yr)<-paste("DGS5")
> t5yr[1:3,]
          DGS5
1962-01-02 3.88
1962-01-03 3.87
1962-01-04 3.86
>
> t7yr<-read.csv("DGS7 FRED.csv",skip=10)
> t7yr$date<-as.Date(t7yr$observation_date,"%Y-%m-%d")
> t7yr$DGS7<-as.numeric(as.character(t7yr$DGS7))
Warning message:
NAs introduced by coercion
> t7yr<-xts(t7yr$DGS7,order.by=t7yr$date)
> names(t7yr)<-paste("DGS7")
> t7yr[1:3,]
          DGS7
1969-07-01 6.88
1969-07-02 6.89
1969-07-03 6.85
>
> t10yr<-read.csv("DGS10 FRED.csv",skip=10)
> t10yr$date<-as.Date(t10yr$observation_date,"%Y-%m-%d")
> t10yr$DGS10<-as.numeric(as.character(t10yr$DGS10))
Warning message:
NAs introduced by coercion
> t10yr<-xts(t10yr$DGS10,order.by=t10yr$date)
> names(t10yr)<-paste("DGS10")
> t10yr[1:3,]
          DGS10
1962-01-02  4.06
1962-01-03  4.03
1962-01-04  3.99
>
```

```
> t20yr<-read.csv("DGS20 FRED.csv",skip=10)
> t20yr$date<-as.Date(t20yr$observation_date,"%Y-%m-%d")
> t20yr$DGS20<-as.numeric(as.character(t20yr$DGS20))
Warning message:
NAs introduced by coercion
> t20yr<-xts(t20yr$DGS20,order.by=t20yr$date)
> names(t20yr)<-paste("DGS20")
> t20yr[1:3,]
           DGS20
1993-10-01  6.12
 1993-10-04  6.10
 1993-10-05  6.12
>
> t30yr<-read.csv("DGS30 FRED.csv",skip=10)
> t30yr$date<-as.Date(t30yr$observation_date,"%Y-%m-%d")
> t30yr$DGS30<-as.numeric(as.character(t30yr$DGS30))
Warning message:
NAs introduced by coercion
> t30yr<-xts(t30yr$DGS30,order.by=t30yr$date)
> names(t30yr)<-paste("DGS30")
> t30yr[1:3,]
           DGS30
1977-02-15  7.70
1977-02-16  7.67
1977-02-17  7.67
```

Note that there is a warning message that is produced when we convert the yields into numeric values. The reason for the warning message is because there are character values in the original data that are labeled "N/A." When R attempts to convert the rest of the values, which are numbers, to a numeric object, it has to force the conversion of these N/As into a numeric "NA" value.

Step 2: Combine Yield Data into One Data Object We then combine the data for all key maturities into one data object labeled treasury. We iteratively merge two data objects at a time using the merge command. Note that the way we applied the merge command, we are creating NAs anytime one data object does not have any data for which another data object has data for. For example, the 1-Year Treasury has data beginning on January 2, 1962. However, the 6-month Treasury did not have data until January 4, 1982. As such, the 6-month Treasury would have an NA from January 2, 1962 until January 4, 1982.

```
> treasury<-t3mo
> treasury<-merge(treasury,t6mo)
> treasury<-merge(treasury,t1yr)
> treasury<-merge(treasury,t2yr)
> treasury<-merge(treasury,t3yr)
> treasury<-merge(treasury,t5yr)
> treasury<-merge(treasury,t7yr)
> treasury<-merge(treasury,t10yr)
> treasury<-merge(treasury,t20yr)
> treasury<-merge(treasury,t30yr)
> treasury[1:3,]
           DGS3MO DGS6MO DGS1 DGS2 DGS3 DGS5 DGS7 DGS10 DGS20 DGS30
1962-01-02    NA     NA 3.22   NA 3.70 3.88   NA  4.06    NA    NA
1962-01-03    NA     NA 3.24   NA 3.70 3.87   NA  4.03    NA    NA
1962-01-04    NA     NA 3.24   NA 3.69 3.86   NA  3.99    NA    NA
```

Step 3: Subset Data to Only Include Yields from 1990 to 2013 We then follow the technique we discussed in Chap. 1 to subset dates.

```
> extreme<-subset(treasury,
+   index(treasury) >= "1990-01-01" &
+   index(treasury) <= "2013-12-31")
```

We then use the na.omit command to delete observations with at least one NA. As such, we have excluded all data from February 18, 2002 through February 9, 2006 because the 30-year Treasury was not available during that period. The rest of the observations omitted are days on which no Treasury security traded.

```
> extreme<-na.omit(extreme[,c(1,8,10)])
> extreme[c(1:3,nrow(extreme)),]
           DGS3MO DGS10 DGS30
1990-01-02   7.83  7.94  8.00
1990-01-03   7.89  7.99  8.04
1990-01-04   7.84  7.98  8.04
2013-12-31   0.07  3.04  3.96
```

Step 4: Identify Examples of Different Shapes of the Yield Curve The next few steps show how to identify the period with the steepest inverted and upward sloping Treasury yield curve. To do this, we calculate the difference between the 30-Year and 3-Month Treasury yield. Using the ifelse command, we can find the smallest difference in yields and extract that observation as that will be our example of the steepest inverted yield curve. Similarly, we can find the largest difference in yields and extract that observation as that will be our example of the steepest upward sloping yield curve.

```
> extreme$sign.diff<-extreme$DGS30-extreme$DGS3MO
> extreme$inverted<-ifelse(extreme$sign.diff==min(extreme$sign.diff),1,0)
> inverted<-subset(extreme,extreme$inverted==1)
> inverted
           DGS3MO DGS10 DGS30 sign.diff inverted
2000-11-24   6.36  5.63  5.67     -0.69        1
>
> extreme$upward<-ifelse(extreme$sign.diff==max(extreme$sign.diff),1,0)
> upward<-subset(extreme,extreme$upward==1)
> upward
           DGS3MO DGS10 DGS30 sign.diff inverted upward
2010-01-11   0.04  3.85  4.74       4.7        0      1
```

To find the example of the flattest Treasury yield curve, we first find the absolute value of the difference between the 30-year and 3-month Treasuries. Using the `ifelse` command, we can find the minimum absolute difference and extract that observation as our example of the flattest Treasury yield curve.

```
> extreme$abs.diff<-abs(extreme$DGS30-extreme$DGS3MO)
> extreme$flat<-ifelse(extreme$abs.diff==min(extreme$abs.diff),1,0)
> flat<-subset(extreme,extreme$flat==1)
> flat$abs.diff2<-abs(flat$DGS30-flat$DGS10)
> flat$flat2<-ifelse(flat$abs.diff2==min(flat$abs.diff2),1,0)
> flat[,c(-4:-6)]
           DGS3MO DGS10 DGS30 abs.diff flat abs.diff2 flat2
2000-04-13   5.81  5.94  5.81        0    1      0.13     0
2006-03-02   4.62  4.64  4.62        0    1      0.02     1
2006-07-20   5.08  5.03  5.08        0    1      0.05     0
2006-07-21   5.10  5.05  5.10        0    1      0.05     0
2006-07-25   5.13  5.07  5.13        0    1      0.06     0
2006-07-28   5.07  5.00  5.07        0    1      0.07     0
2006-08-14   5.12  5.00  5.12        0    1      0.12     0
```

Note that the output to show the flatest yield curve suppresses the fourth through sixth columns. Those observations are related to the steepest upward and inverted yield curves. As such, suppressing those two columns in the output make it easier to see the columns that are relevant to the flat yield curve analysis.

In the above code, we also create a `flat2` variable, which is equal to the absolute value of the difference between the 30-year and 10-year Treasury yields. This second filter allows us to narrow the seven periods over which there was no difference between the 30-year and 3-month Treasury yields. Based on this filter, we find that the yield curve on March 2, 2006 has the smallest difference between the 30-year and 10-year Treasury yields.

Step 5: Extract the Yield Curve on Dates Selected and on December 31, 2013
We can now create variables that hold the dates with our examples of an inverted, flat, and upward sloping (i.e., normal) yield curve. Then, we can extract these dates from `treasury`. We also extract the yield curve on December 31, 2013, which as of the time of this writing gives us the "current" shape of the yield curve.

```
> invert.date<-as.Date("2000-11-24")
> normal.date<-as.Date("2010-01-11")
> flat.date<-as.Date("2006-03-02")
> current.date<-as.Date("2013-12-31")
>
> tyld.curve<-subset(treasury,
+     index(treasury) == invert.date |
+     index(treasury) == flat.date |
+     index(treasury) == normal.date |
+     index(treasury) == current.date)
> tyld.curve
           DGS3MO DGS6MO DGS1 DGS2 DGS3 DGS5 DGS7 DGS10 DGS20 DGS30
2000-11-24   6.36   6.34 6.12 5.86 5.74 5.63 5.70  5.63  5.86  5.67
2006-03-02   4.62   4.75 4.74 4.72 4.72 4.68 4.66  4.64  4.80  4.62
2010-01-11   0.04   0.13 0.35 0.95 1.55 2.58 3.32  3.85  4.64  4.74
2013-12-31   0.07   0.10 0.13 0.38 0.78 1.75 2.45  3.04  3.72  3.96
```

Step 6: Prepare Data for Plotting We have to set up the data so we can plot it properly. First, we transpose the data using the t(...) operator. Note that prior to the use of the transpose command, tyld.curve was a xts object and after the transpose it is now a matrix object.

```
> class(tyld.curve)
[1] "xts" "zoo"
> tyld.curve<-t(tyld.curve)
> tyld.curve
           2000-11-24 2006-03-02 2010-01-11 2013-12-31
DGS3MO           6.36       4.62       0.04       0.07
DGS6MO           6.34       4.75       0.13       0.10
DGS1             6.12       4.74       0.35       0.13
DGS2             5.86       4.72       0.95       0.38
DGS3             5.74       4.72       1.55       0.78
DGS5             5.63       4.68       2.58       1.75

DGS7             5.70       4.66       3.32       2.45
DGS10            5.63       4.64       3.85       3.04
DGS20            5.86       4.80       4.64       3.72
DGS30            5.67       4.62       4.74       3.96
> class(tyld.curve)
[1] "matrix"
```

We then rename the index and variable names to make them more meaningful. The rows now have the time to maturity in years, while the variable names now represent the shape of the yield curve.

```
> rownames(tyld.curve)<-paste(c(0.25,0.5,1,2,3,5,7,10,20,30))
> colnames(tyld.curve)<-paste(c("inverted","flat","normal","current"))
> tyld.curve
     inverted flat normal current
0.25     6.36 4.62   0.04    0.07
0.5      6.34 4.75   0.13    0.10
1        6.12 4.74   0.35    0.13
2        5.86 4.72   0.95    0.38
3        5.74 4.72   1.55    0.78
5        5.63 4.68   2.58    1.75
7        5.70 4.66   3.32    2.45
10       5.63 4.64   3.85    3.04
20       5.86 4.80   4.64    3.72
30       5.67 4.62   4.74    3.96
```

Step 7: Plot the Yield Curve on the Four Dates Selected   Before we can plot the data, we have to generate the values for the x-axis, which correspond to the key maturity dates. We also have to setup the range for the y-axis using the `range` command. Then, when plotting the data, since `tyld.curve` is a matrix, we call the data by its column name. That is, the first column for the inverted yield curve, the second column for the flat yield curve, the third column for the normal yield curve, and the fourth column for the current yield curve.

```
> TTM<-c(0.25,0.5,1,2,3,5,7,10,20,30)
> TTM
 [1]  0.25  0.50  1.00  2.00  3.00  5.00  7.00 10.00 20.00 30.00
> y.range<-range(tyld.curve)
> y.range
[1] 0.04 6.36
> plot(x=TTM,y=tyld.curve[,1],type="o",ylim=y.range,
+     xlab="Time to Maturity",ylab="Yield (Percent)",
+     col="gray60",
+     lwd=2,
+     main="Shapes of the Treasury Yield Curve")
> lines(x=TTM,y=tyld.curve[,2],type="o",lty=3,col="black")
> lines(x=TTM,y=tyld.curve[,3],type="o",col="black",lwd=3)
> lines(x=TTM,y=tyld.curve[,4],type="o",lty=3,col="gray40",lwd=2)
> legend("bottomright",c("Inverted (11/24/2000)",
+     "Flat (08/26/2006)","Normal (01/11/2010)",
+     "December 31, 2013"),
+     col=c("gray60","black","black","gray40"),
+     lty=c(1,3,1,3),
+     lwd=c(2,1,3,2))
```

Figure 8.5 shows the different shapes of the yield curve. We can see that the steepest inverted yield curve is not that steep, but the steepest upward sloping or normal yield curve was very steep. In fact, the current yield curve is fairly steep with the short-end being close to zero and the long-end of the curve being around 4 %.

Shapes of the Treasury Yield Curve
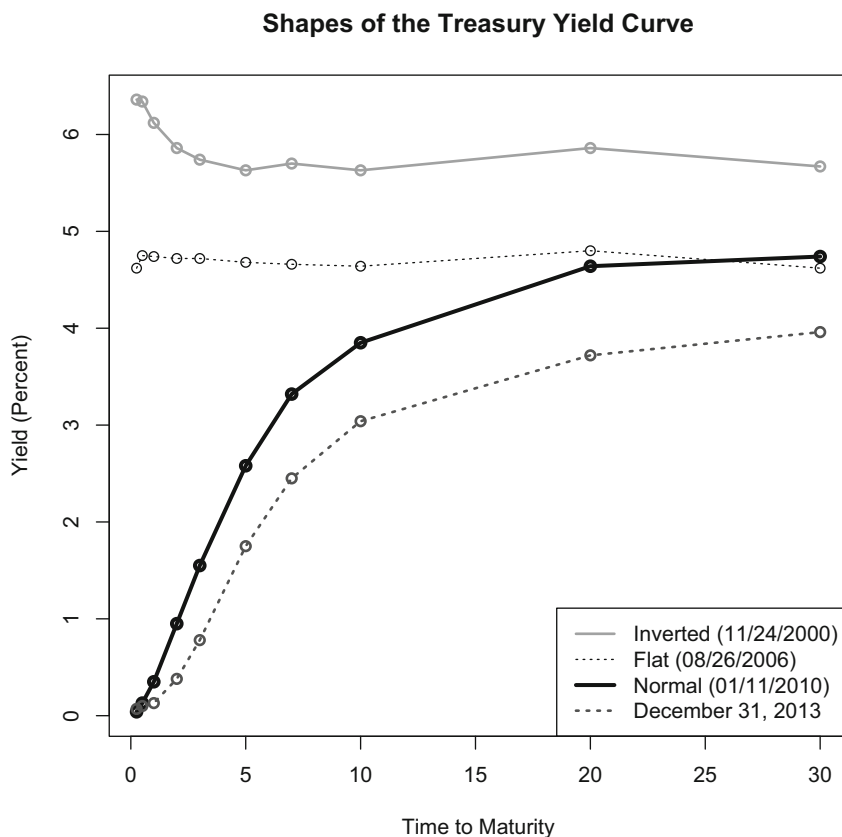
Fig. 8.5 Different shapes of the US Treasury yield curve. Data Source: Federal Reserve Electronic Database

8.2.2   Slope of the US Treasury Yield Curve

The slope of the Treasury yield curve has been known as a leading indicator of future economic activity and rising inflation expectations. The slope is calculated as the difference between the yields of long-term and short-term Treasury securities. In prior recessions, short-term interest rates rose above long-term rates, which is the opposite of the more conventional pattern of long-term interest rates being higher than short-term interest rates.[3] In this section, we show how to calculate a time series of the slope of the yield curve using the 3-Month Treasury Note and 30-Year Treasury Bond from 2007 to 2013.

---

[3] For more details, see Estrella and Trubin [2].

Step 1: Obtain Data for Three Month and 30-Year Treasury Since we already retrieved the 3-Month Treasury (`t3mo`) and 30-Year Treasury (`t30yr`) data above, we can combine the two into a new data object labeled `slope`.

```
> slope<-t3mo
> slope<-merge(t3mo,t30yr)
> slope[1:3,]
           DGS3MO DGS30
1977-02-15    NA  7.70
1977-02-16    NA  7.67
1977-02-17    NA  7.67
```

We then subset the data only to dates between 2007 and 2013. The reason is that the 30-Year Treasury Bond was discontinued from February 2002 to February 2006.

```
> slope<-subset(slope,
+     index(slope)>= "2007-01-01" &
+     index(slope)<= "2013-12-31")
> slope[c(1:3,nrow(slope)),]
           DGS3MO DGS30
2007-01-01    NA    NA
2007-01-02  5.07  4.79
2007-01-03  5.05  4.77
2013-12-31  0.07  3.96
```

Step 2: Remove the NAs on Non-trading Days There are days on which both the 2-Year and 30-Year Treasury did not trade and, consequently, the data contains NAs on those days. Because both series have NAs, the NA days are likely to be non-trading days and can, therefore, be safely deleted. We delete observations with NAs using the `na.omit` command.

```
> slope<-na.omit(slope)
> slope[c(1:3,nrow(slope)),]
           DGS3MO DGS30
2007-01-02  5.07  4.79
2007-01-03  5.05  4.77
2007-01-04  5.04  4.72
2013-12-31  0.07  3.96
```

Step 3: Plot the Raw Yield Data We first setup the a `y.range` variable to make sure the y-axis is wide enough to cover the smallest and largest yields from both series. To calculate the range, we use the `range` command. Since this is an `xts` object, we can use the `index` command to use the date in the index for the x-axis.

```
> y.range<-range(slope)
> y.range
[1] 0.00 5.35
> plot(x=index(slope),
+     xlab="Date",
+     y=slope$DGS30,
+     ylab="Yield (Percent)",
+     ylim=y.range,
+     type="l",
+     col="black",
+     lty=1,
+     lwd=1,
+     main="Yields on 3-Month and 30-Year Treasuries (2007-2013)")
```

Step 4: Shade the Most Recent Recession from December 1, 2007 to June 1, 2009 Next, to mark the date range of the most recent recession, we will add a shaded rectangular box to the chart to denote December 1, 2007 to June 1, 2009, which are the dates identified by the NBER Dating Committee as the official start and end dates of the last recession. The first step in doing this, is to use the par("usr") command to report the coordinates of the chart. Then, we use the rect command to draw a rectangle, where the coordinates are given by (i) our first date, (ii) the coordinates for the top of the chart (shade[2]), (iii) our last date, and (iv) the coordinates for the bottom of the chart (shade[3]). Finally, using the box command we redraw the original chart border because the upper border gets covered by the gray shaded rectangle.

```
> shade<-par("usr")
> shade
[1] 13412.800 16172.200    -0.214    5.564
> rect(as.Date("2007-12-01"),
+     shade[2],
+     as.Date("2009-06-01"),
+     shade[3],
+     col="gray60",
+     lty=0)
> box(which="plot",lty=1)
```

The shaded rectangle actually covers up part of the original line for the yield of the 30-year Trasury, so when we add lines to the chart, we need to add the line for the 30-year Treasury again. We then add a line for the 3-month Treasury. The last step would be to create a legend.

```
> lines(x=index(slope),y=slope$DGS30)
> lines(x=index(slope),y=slope$DGS3MO,lty=2,col="black")
> legend("topright",
+     c("3-Month Treasury","30-Year Treasury"),
+     lty=c(2,1))
```

Yields on 3-Month and 30-Year Treasuries (2007-2013)
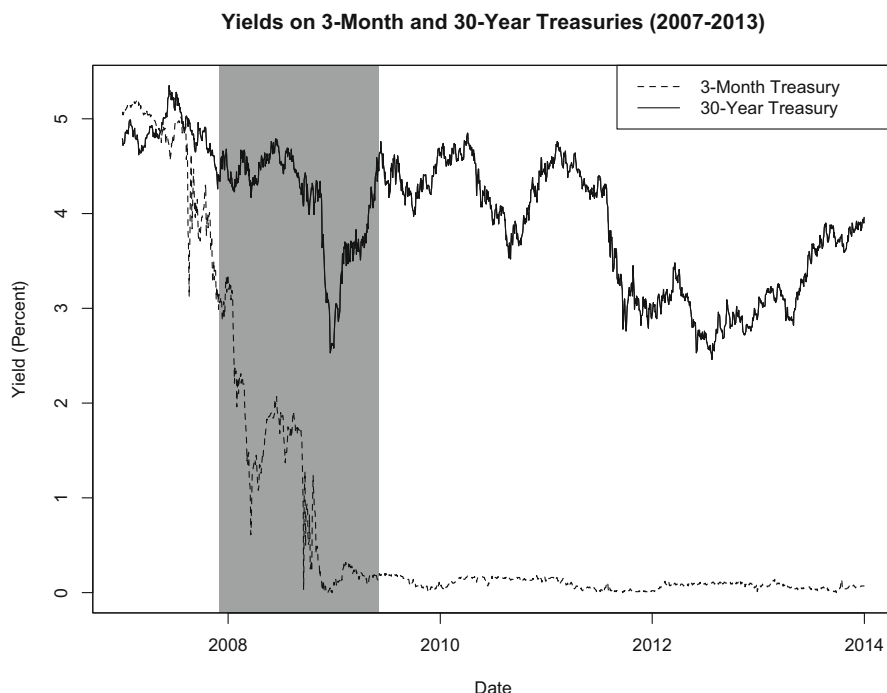
Fig. 8.6 Yields on short-term and long-term Treasury securities, 2007–2012. Data Source: Federal Reserve Electronic Database

Figure 8.6 shows the result of our plot. As the chart shows, the last recession was when we see short-term rates drop significantly from over 5 % to close to zero. These rates have not recovered through the end of 2013. The long-term rates were also volatile during this period, dropping from a high of approximately 5 % down to approximately 2.5 % twice: once during the recession and another during 2012–2013.

Step 5: Calculate the Slope of the Yield Curve   The chart above indicates that the slope of the yield curve has steepened since the last recession. This means that gap between the long-term yields and short-term yields on treasuries have widened. We can verify this by looking at the difference between the two yields. We multiply the yields by 100 in order to represent the spread in basis points (bps). Note that 100 bps is equal to 1 %.

```
> slope$slope<-(slope$DGS30-slope$DGS3MO)*100
> slope[c(1:3,nrow(slope)),]
           DGS3MO DGS30 slope
2007-01-02   5.07  4.79   -28
2007-01-03   5.05  4.77   -28
2007-01-04   5.04  4.72   -32
2013-12-31   0.07  3.96   389
```

Step 6: Plot the Slope of the Yield Curve   Next, we plot the slope of the yield curve using the same technique we plotted the long-term and short-term rates above.

```
> plot(x=index(slope),
+     xlab="Date",
+     y=slope$slope,
+     ylab="Spread (bps)",
+     type="l",
+     lty=1,
+     lwd=1,
+     main="Slope of the US Treasury Yield Curve from 2007 to 2013
+     Based on the Spread Between the
+     3-Month and 30-Year Treasury Yields")
> shade<-par("usr")
> shade
[1] 13412.80 16172.20  -72.88   490.88
> rect(as.Date("2007-12-01"),
+     shade[2],
+     as.Date("2009-06-01"),
+     shade[3],
+     col="gray60",
+     lty=0)
> box(which="plot",lty=1)
> abline(h=0,lty=1)
> lines(x=index(slope),y=slope$slope,lty=1,lwd=1)
```

Figure 8.7 shows the slope of the Treasury yield curve from 2007 to 2013. The figure shows that the long-term rates are pretty high relative to short-term rates. The magnitude of the difference may be attributed to two reasons. First, we can partly attribute this gap to the Fed maintaining the short-term rates at low levels since 2008. Second, this gap is partly caused by the stimulus that was largely financed by federal debt, which put pressure on long-term Treasuries.

8.2.3   Real Yields on US Treasuries

The real yield on Treasuries is an indicator of economic growth. The real yield falls when economic growth falls, and the real yield rises when economic growth rises. Real yields can can be observed by looking at the yields on Treasury Inflation Protected Securities (TIPS). The inflation protection comes from the fact that the principal and coupon of the TIPS are tied to the consumer price index, the change in which is commonly-used as the rate of inflation.

Another important reason to look at real yields is to observe how the real interest rate is moving. Recall that the real yield is equal to the nominal yield minus the inflation rate. Therefore, when we see the nominal yield decline, it could be a result of the real yield declining or the inflation rate declining or both. Consider the case in

Slope of the US Treasury Yield Curve from 2007 to 2013
Based on the Spread Between the
3–Month and 30–Year Treasury Yields
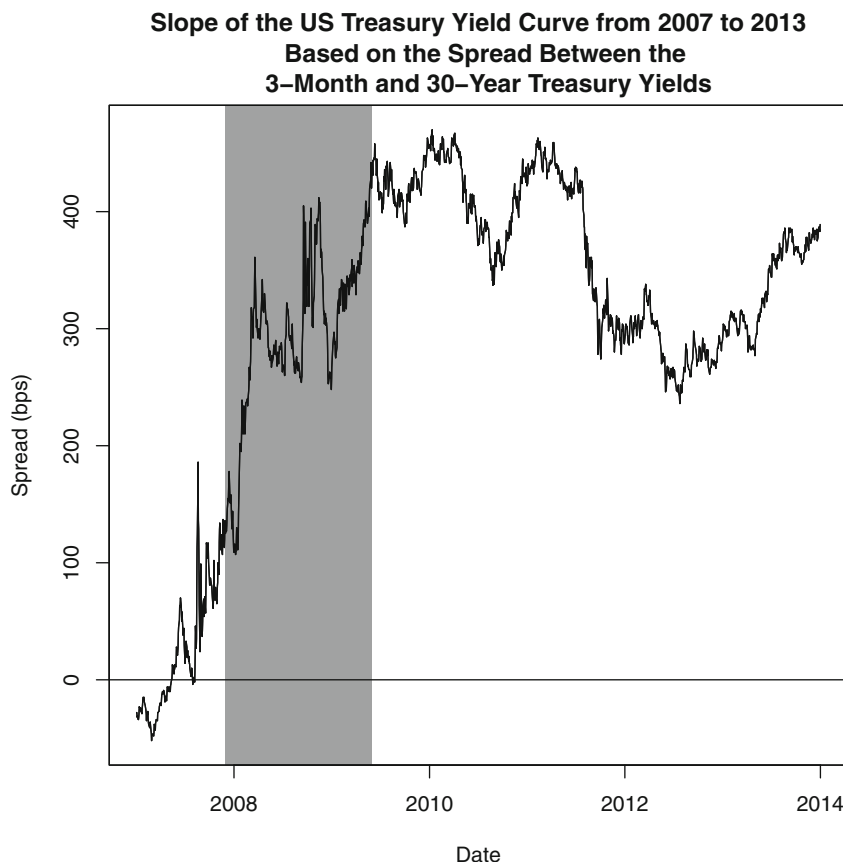
Fig. 8.7 Slope of Treasury yield curve based on spread of 3-Month and 30-Year Treasuries, 2007–2012. Data Source: Federal Reserve Electronic Database and author's calculations

which we observe the nominal yield decline from 5 to 3 %, which is a 2 % decline. However, if the inflation rate declined from 3 to 1 %, the real yield in both instances would be the same at 2 % (i.e., $5 - 3\% = 2\%$ and $3 - 1\% = 2\%$). This can be revealed by analyzing the real yields.

Step 1: Obtain 10-Year TIPS Data from FRED We use the 10-Year TIPS in our example to show how the real yields have performed in from January 2003 to December 2013. We begin by obtaining the data from FRED and save the data as **DFII10 FRED.csv**.

```
> TIPS<-read.csv("DFII10 FRED.csv",skip=10)
> TIPS$date<-as.Date(TIPS$observation_date,"%Y-%m-%d")
> TIPS$DFII10<-as.numeric(as.character(TIPS$DFII10))
Warning message:
NAs introduced by coercion
> TIPS<-xts(TIPS$DFII10,order.by=TIPS$date)
> names(TIPS)<-paste("DFII10")
> TIPS[1:3,]
           DFII10
2003-01-02   2.43
2003-01-03   2.43
2003-01-06   2.46
```

Step 2: Subset Data from 2003 to 2013   To subset the data, we use the subset command.

```
> TIPS<-subset(TIPS,
+     index(TIPS)>="2003-01-01" &
+     index(TIPS)<="2013-12-31")
> TIPS[c(1:3,nrow(TIPS)),]
           DFII10
2003-01-02   2.43
2003-01-03   2.43
2003-01-06   2.46
2013-12-31   0.80
```

Step 3: Plot the Real Yield Data   We plot the data using the techniques we used previously, including shading the period covered by the most recent recession.

```
> plot(x=index(TIPS),
+     xlab="Date",
+     y=TIPS$DFII10,
+     ylab="Yield (Percent)",
+     type="l",
+     main="Yield on 10-Year Treasury Inflation Protected Securities (TIPS)
+     January 2003 - December 2013")
> shade<-par("usr")
> shade
[1] 11893.3600 16230.6400    -1.0308     3.3108
> rect(as.Date("2007-12-01"),
+     shade[2],
+     as.Date("2009-06-01"),
+     shade[3],
+     col="gray60",
+     lty=0)
> box(which="plot",lty=1)
> lines(x=index(TIPS),y=TIPS$DFII10)
> abline(h=0,lty=1)
```

Figure 8.8 shows the output of our code. As we can see, real yields began a sharp decline after the 2008/2009 crisis. In fact, by the end of 2010, real yields on Treasuries

Yield on 10–Year Treasury Inflation Protected Securities (TIPS)
January 2003 – December 2013
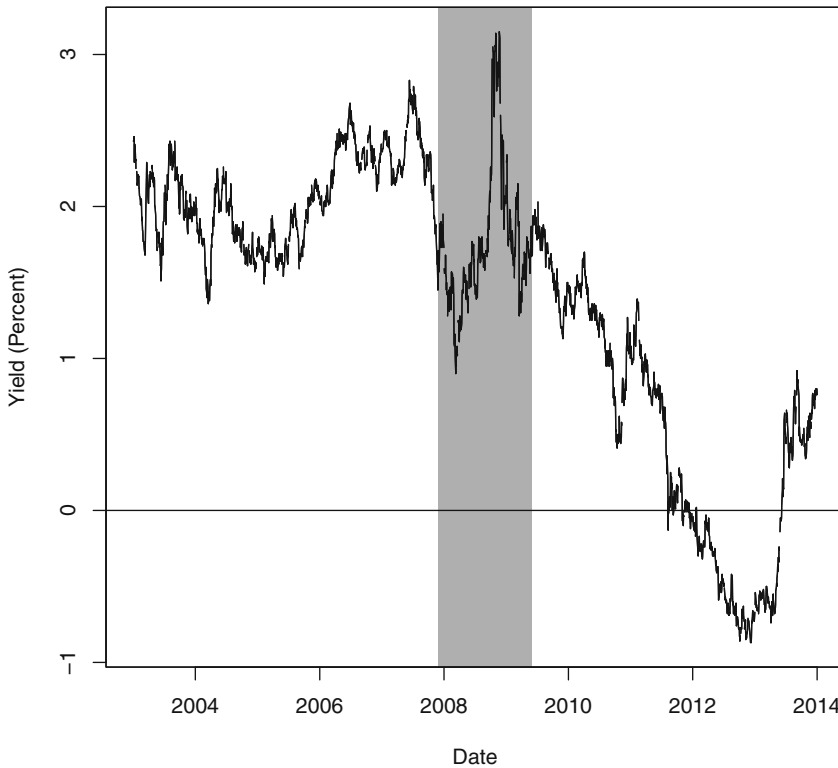
Fig. 8.8 Declining Real Treasury yields, 2003–2012. Data Source: Federal Reserve Electronic Database and author's calculations

were negative, which can be an indication of the severe risk aversion and flight to safety that was observed after the 2008/2009 crisis. Only in 2013 did real yields break the declining trend and end the year at a positive level.

8.2.4   Expected Inflation Rates

The *breakeven rate* of TIPS is equal to the yield on conventional Treasury securities less the yield on TIPS. This difference can be interpreted as the market's inflation expectation over the term of the TIPS/Treasury security.

Step 1: Import 10-Year TIPS Data from FRED We retrieve 10-Year Treasury Inflation Protection Securities (TIPS) with symbol DFII10 from FRED. The output below shows the data begins in January 2003. This will limit the length of the data

we use in our analysis. However, since we still have over 10 years of daily data, the limitation on the data does not impact our analysis.

```
> TIPS[1:3,]
           DFII10
2003-01-02   2.43
2003-01-03   2.43
2003-01-06   2.46
```

Step 2: Import 10-Year Treasury Data from FRED We retrieve 10-Year Constant Maturity Treasury data with symbol DGS10 from FRED. We already read-in the data in an earlier section, so we only need to call-in the data object `t10yr` in this section.

```
> t10yr[1:3,]
           DGS10
1962-01-02   4.06
1962-01-03   4.03
1962-01-04   3.99
```

Step 3: Combine and Clean the Data We combine the data using the `merge` command. This step alone would show a series of NAs for the 10-Year TIPS data from 1962 to 2002, as the 10-Year TIPS did not have data for that period. However, from 2003 through 2013, both the 10-Year TIPS and 10-Year Treasury have NAs on the same dates, which suggests these dates are non-trading days. As such, we can safely remove all observations with at least one variable with an NA. This means we delete data prior to when we have TIPS data *and* from the beginning of the TIPS data we delete non-trading days from the data. This is implemented by using the `na.omit` command. This results in data beginning in January 2003.

```
> exp.infl<-na.omit(merge(TIPS,t10yr))
> exp.infl[1:3,]
           DFII10 DGS10
2003-01-02   2.43  4.07
2003-01-03   2.43  4.05
2003-01-06   2.46  4.09
```

Step 4: Limit Data from January 2003 to December 2013 We use the `subset` command to limit the data to our desired date range.

```
> exp.infl<-subset(exp.infl,
+    index(exp.infl)<="2013-12-31")
> exp.infl[c(1:3,nrow(exp.infl)),]
           DFII10 DGS10
2003-01-02   2.43  4.07
2003-01-03   2.43  4.05
2003-01-06   2.46  4.09
2013-12-31   0.80  3.04
```

Step 5: Calculate the Inflation Expectation We then calculate the market's infla-
tion expectation by subtracting the 10-Year TIPS yield from the 10-Year Treasury
yield. This difference can be interpreted as the market's average annual inflation
expectation over the next 10 years.

```
> exp.infl$infl.10yr<-exp.infl$DGS10-exp.infl$DFII10
> exp.infl[c(1:3,nrow(exp.infl)),]
           DFII10 DGS10 infl.10yr
2003-01-02   2.43  4.07      1.64
2003-01-03   2.43  4.05      1.62
2003-01-06   2.46  4.09      1.63
2013-12-31   0.80  3.04      2.24
```

We can compare our result to other sources of inflation projections. A common source
of such inflation expectations is the Livingston Survey by the Federal Reserve Bank of
Philadelphia.[4] The Livingston Survey is the oldest continuous survey of economists'
expectations. In the December 2013 Livingston Survey, the long-term outlook for
average annual inflation for the next 10 years is 2.3 %, which is consistent with what
the market was expecting inflation over the next 10 years is going to be.

Step 6: Plot Expected Inflation We use the `plot` command to chart our data. For
context, we also shade the latest recession period.

```
> plot(x=index(exp.infl),
+     y=exp.infl$infl.10yr,
+     xlab="Date",
+     ylab="Inflation Rate (%)",
+     type="l",
+     main="Expected Inflation Implied From
+     10-Year TIPS Breakeven Rate
+     January 2003 - December 2013")
> shade<-par("usr")
> shade
[1] 11893.3600 16230.6400   -0.0688    2.8688
> rect(as.Date("2007-12-01"),
+     shade[2],
+     as.Date("2009-06-01"),
+     shade[3],
+     col="gray60",
+     lty=0)
> box(which="plot",lty=1)
> lines(x=index(exp.infl),y=exp.infl$infl.10yr)
```

---

[4] See   http://www.philadelphiafed.org/research-and-data/real-time-center/livingston-survey/   for
more details.

**Expected Inflation Implied From
10−Year TIPS Breakeven Rate
January 2003 − December 2013**
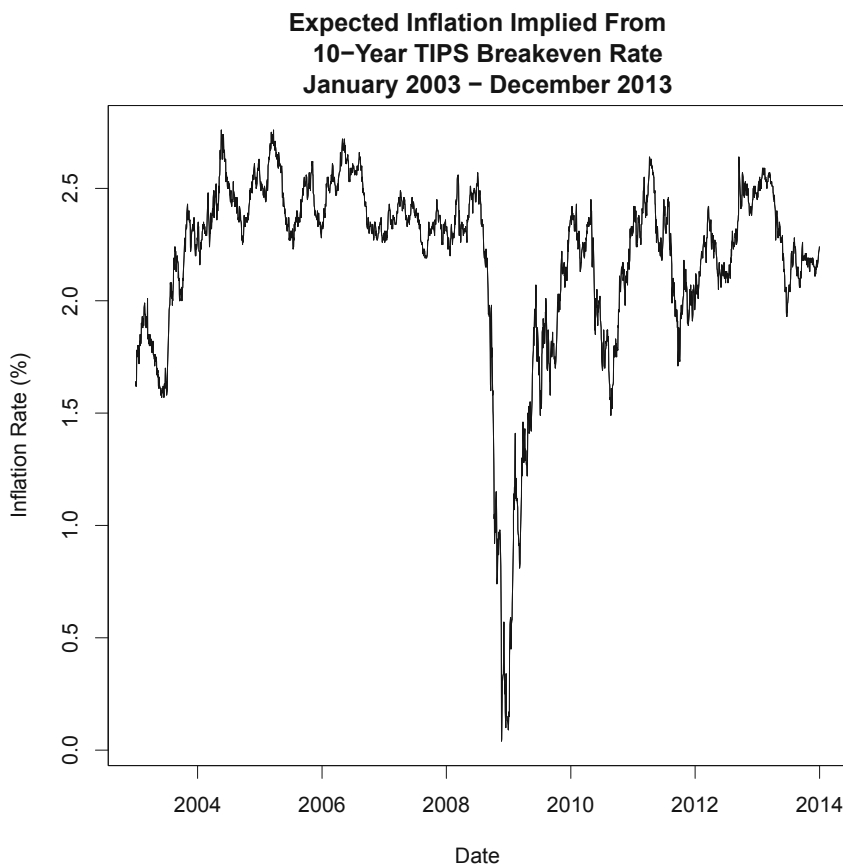
Fig. 8.9 Expected inflation rates based on breakeven rate of TIPS, 2003–2012. Data Source: Federal Reserve Electronic Database and author's calculations

Figure 8.9 shows the market's inflation expectation from 2003 to 2013. As we can see, during the last recession, the inflation expectation fell sharply to almost zero. We can compare that to the average expected inflation rate of 2.2 % for the entire period. As the recession period expected inflation was so low compared to the historical average, we may expect the inflation rate to move higher towards the long-term mean. The data shows that this is what happened after the crisis.

```
> mean(exp.infl$infl.10yr)
[1] 2.186466
```

Step 7: Setup Actual Inflation Rates for 2003–2013 We can compare the expected inflation to historical inflation. To do this, we first create a subset of the actual US historical inflation rates for the 2003–2013 period.

```
> CPI.2003<-subset(US.CPI,
+     index(US.CPI)>="2003-01-01" &
+     index(US.CPI)<="2013-12-01")
> CPI.2003[c(1:3,nrow(CPI.2003)),]
            inflation
2003-01-01  2.597403
2003-02-01  2.980877
2003-03-01  3.020134
2013-12-01  1.501736
```

Step 8: Plot the Inflation Data Here, we opt to show a stack the two line charts with the expected inflation at the top and the actual inflation at the bottom.

```
> par(oma=c(0,0,4,0))
> par(mfrow=c(2,1))
> plot(x=index(exp.infl),
+     y=exp.infl$infl.10yr,
+     xlab="Date",
+     ylab="Expected Inflation Rate (%)",
+     type="l")
> plot(x=index(CPI.2003),
+     y=CPI.2003$inflation,
+     xlab="Date",
+     ylab="Actual Inflation Rate (%)",
+     type="l")
> title(main="Comparing Expected Inflation Implied from TIPS
+     With Actual Inflation Based on Changes in CPI From 2003 to 2013",
+     outer=TRUE)
> par(mfrow=c(1,1))
```

The results of this code is shown in Fig. 8.10. We can see that the expected inflation ranges from a little over 0 % around 2009 to a high of less than 3 %. In contrast, historical inflation fell to −2 % in mid-2009 and was around 5 % in 2008. Therefore, historical inflation had a much wider range of values during the crisis compared to the the values implied by looking at the TIPS. Similarly, outside the crisis periods, the expected inflation rates implied from TIPS fluctuate around 1.5–3.0 %, while the actual inflation rate fluctuated around 1.5 to over 4.0 %. Note, however, we have to be a little careful with a temporal comparison between these two series because the expected inflation calculated from the 10-Year TIPS is the inflation expected over the 10-year period of the security.

8.2.5 Mean Reversion

When yields are predictable, there is opportunity for investors to make money. One indicator of return predictability is when yields are mean reverting. This means that if yields today are higher than some historical average, yields would have a tendency to decline towards the mean in the future. Conversely, if yields today are lower than some historical average, yields would have a tendency to rise towards the mean in

**Comparing Expected Inflation Implied from TIPS
With Actual Inflation Based on Changes in CPI From 2003 to 2013**
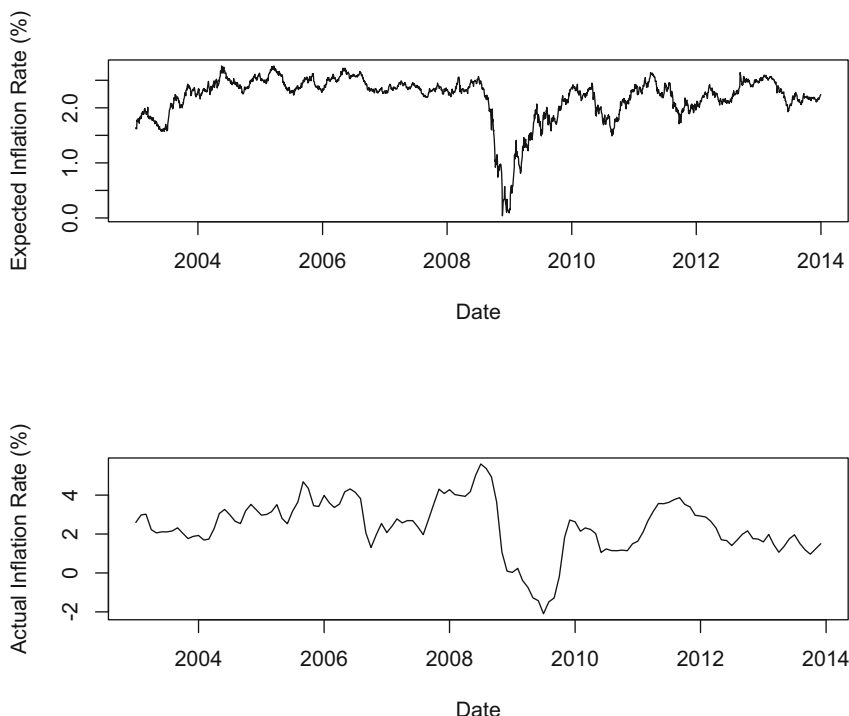
Fig. 8.10 Expected and historical US inflation rates, 2003–2012. Data Source: Federal Reserve Electronic Database

the future. We implement an analysis to determine whether yields for the 10-Year Constant Maturity Treasury exhibit mean reversion. In this analysis, we will use a 200-week moving average of yields.

Step 1: Obtain 10-Year Treasury Data from FRED   To perform this analysis, we obtain weekly 10-Year Treasury data from FRED (WGS10YR). We save this data as **WGS10YR FRED.csv**.

```
> cmt<-read.csv("WGS10YRFRED.csv",skip=10)
> cmt$date<-as.Date(cmt$observation_date,"%Y-%m-%d")
> cmt$WGS10YR<-as.numeric(as.character(cmt$WGS10YR))
> cmt<-xts(cmt$WGS10YR,order.by=cmt$date)
> names(cmt)<-paste("WGS10YR")
> cmt[1:3,]
          WGS10YR
1962-01-05   4.03
1962-01-12   4.06
1962-01-19   4.11
```

Step 2: Calculate a 200-Week Moving Average of the Yields We then use a 200-week Moving Average as an indicator of the trend in yields. A moving average can be calculated in R using the `rollmeanr` command. We use the `k=200` argument to calculate the 200-week moving average for each week. The final argument of `fill=NA` is to fill in with NAs the first 199 weekly observations in the data as we cannot calculate a 200-week moving average for those dates. The output below shows observation numbers 195–205, so we can see when the NA ends and the rolling average begins.

```
> cmt$roll<-rollmeanr(cmt$WGS10YR,k=200,fill=NA)
> cmt[195:205,]
           WGS10YR    roll
1965-09-24    4.29      NA
1965-10-01    4.34      NA
1965-10-08    4.33      NA
1965-10-15    4.32      NA

1965-10-22    4.36      NA
1965-10-29    4.39 4.08610
1965-11-05    4.43 4.08810
1965-11-12    4.46 4.09010
1965-11-19    4.45 4.09180
1965-11-26    4.45 4.09355
1965-12-03    4.50 4.09555
> class(cmt)
[1] "xts" "zoo"
```

Step 3: Subset the Data to Include Yields from 1970 to 2013 We subset the data to only include data from January 1970 to December 2013.

```
> cmt<-subset(cmt,
+     index(cmt)>="1970-01-01" &
+     index(cmt)<="2013-12-31")
> cmt[c(1:3,nrow(cmt)),]
          WGS10YR    roll
1970-01-02    7.94 5.62455
1970-01-09    7.93 5.63945
1970-01-16    7.82 5.65410
2013-12-27    2.99 2.48625
```

Step 4: Plot the Yield and Moving Average Data Note that we split the title up into three lines. The reason for this is that for presentation purposes we don't need to squeeze the title into two lines if it aesthetically looks better in three lines. Deciding whether two or three lines or however many lines is necessary may have to be done using trial-and-error.
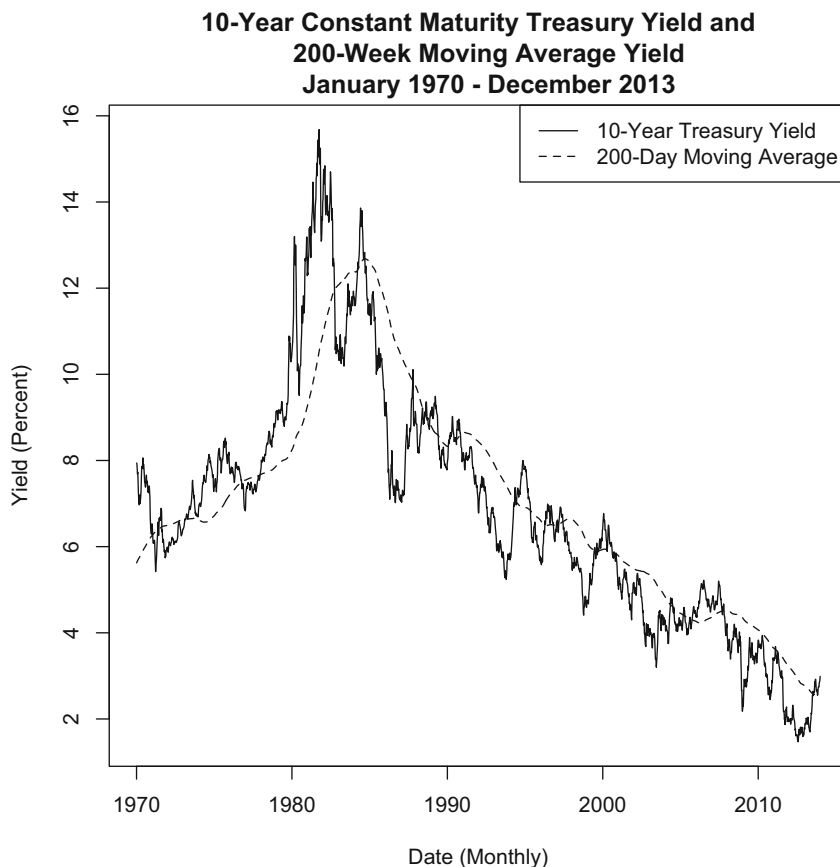
Fig. 8.11 Mean reversion in 10-year Treasury yields, 1970–2013. Data Source: Federal Reserve Electronic Database and author's calculations

```
> plot(x=index(cmt),
+    xlab="Date (Monthly)",
+    y=cmt$WGS10YR,
+    ylab="Yield (Percent)",
+    type="l",
+    col="black",
+    main="10-Year Constant Maturity Treasury Yield and
+    200-Week Moving Average Yield
+    January 1970 - December 2013")
> abline(h=0)
> lines(x=index(cmt),y=cmt$roll,lty=2)
> legend("topright",
+    c("10-Year Treasury Yield","200-Day Moving Average"),
+    lty=c(1,2))
```

The output of our plot is shown in Fig. 8.11. The chart shows that the 10-Year Treasury yields follow closely its 200-week Moving Average.

8.3   Investment Grade Bond Spreads

Treasury securities, like those we looked at in the previous section, are often referred to as default risk-free instruments or, simply, risk-free instruments. However, fixed income securities issued by companies, such as corporate bonds, have varying levels of default risk. Rating agencies, such as Standard & Poor's, Fitch, and Moody's, provide credit ratings to corporate fixed income instruments. Using the Moody's definition, the highest credit rating is Aaa and the lowest investment grade credit rating is Baa. This means that firms or bond issues with less than a Baa rating from Moody's is considered a high-yield bond or a junk bond.

8.3.1   Time Series of Spreads

A common analysis is to evaluate the size of the spread between investment grade bonds. As an example, we analyze the spread of Aaa and Baa bonds from 2006 to 2013. A below-investment grade firm or bond may have limited access to investors' funds, as some types of investors cannot invest in non-investment grade debt as a matter of investment policy. As such, Aaa and Baa rated issues (i.e., investment grade bonds) are fighting for access to a similar pool of capital and investors may decide to invest in one over the other depending on the then-existing risk-return trade off. Therefore, evaluating the spread between the yield of Aaa and Baa bonds is a good indicator of the size of credit risk premium and, based on the investors' own analysis, the investor can determine whether the spread is sufficiently large or sufficiently small to entice her into making an investment.

Step 1: Obtain Moody's Aaa and Baa Corporate Index Yield Data from FRED
Suppose we want to get an understanding of the level of credit spread premium from 2007 to 2012. We obtain data for the Moody's Aaa (AAA) and Baa (BAA) Rated Bonds from FRED. We save the data as **AAA FRED.csv** and **BAA FRED.csv**, respectively.

```
> aaa<-read.csv("AAA FRED.csv",skip=10)
> aaa$date<-as.Date(aaa$observation_date,"%Y-%m-%d")
> aaa$AAA<-as.numeric(as.character(aaa$AAA))
> aaa<-xts(aaa$AAA,order.by=aaa$date)
> names(aaa)<-paste("AAA")
> aaa[1:3,]
           AAA
1919-01-01 5.35
1919-02-01 5.35
1919-03-01 5.39
>
> baa<-read.csv("BAA FRED.csv",skip=10)
> baa$date<-as.Date(baa$observation_date,"%Y-%m-%d")
> baa$BAA<-as.numeric(as.character(baa$BAA))
```

```
> baa<-xts(baa$BAA,order.by=baa$date)
> names(baa)<-paste("BAA")
> baa[1:3,]
           BAA
1919-01-01 7.12
1919-02-01 7.20
1919-03-01 7.15
```

Step 2: Combine Data and Subset to Only Yields from 2006 to 2013 We first merge the two data series into one data object called moodys.

```
> moodys<-merge(aaa,baa)
> moodys[1:3,]
            AAA  BAA
1919-01-01 5.35 7.12
1919-02-01 5.35 7.20
1919-03-01 5.39 7.15
```

We then subset the data to only include yields from June 2006 to December 2013.

```
> moodys<-subset(moodys,
+     index(moodys)>="2006-01-01" &
+     index(moodys)<="2013-12-31")
> moodys[c(1:3,nrow(moodys)),]
            AAA  BAA
2006-01-01 5.29 6.24
2006-02-01 5.35 6.27
2006-03-01 5.53 6.41
2013-12-01 4.62 5.38
```

Step 3: Setup Additional Variables for Plotting For plotting purposes, we have to construct a data.frame object with a date variable. We use the technique we used in Chap. 1 when we subset the data by date.

```
> moodys<-cbind(data.frame(index(moodys)),data.frame(moodys))
> names(moodys)[1]<-paste("date")
> moodys[c(1:3,nrow(moodys)),]
                 date AAA  BAA
2006-01-01 2006-01-01 5.29 6.24
2006-02-01 2006-02-01 5.35 6.27
2006-03-01 2006-03-01 5.53 6.41
2013-12-01 2013-12-01 4.62 5.38
```

Step 4: Plot the Yields and Shade the Spread We now plot the Aaa and Baa series in one chart and shade the area in between to highlight the size of the spread. This shading is performed using the polygon command.

```
> plot(x=moodys$date,
+     xlab="Date",
+     y=moodys$AAA,
+     ylab="Yield (Percent)",
+     ylim=y.range,
+     type="l",
+     lwd=1,
+     lty=1,
+     main="Moody's Aaa and Baa Index Yields From 2006 to 2013")
> lines(x=moodys$date,y=moodys$AAA,lty=1,lwd=3)
> lines(x=moodys$date,y=moodys$BAA,lty=3,lwd=3)
> polygon(c(moodys$date,
+     rev(moodys$date)),
+     c(moodys$BAA,rev(moodys$AAA)),
+     col="gray50",
+     density=20,
+     border=NA)
> abline(v=c(as.Date("2007-12-01"),as.Date("2009-06-01")),lty=1)
> legend("topright",
+     c("Aaa","Baa"),
+     lwd=c(3,3),
+     lty=c(1,3))
```

One difference in the approach we show the last recession period was to use vertical lines instead of shading. The vertical lines are created using the `abline` command. The reason for this is that using the rectangular shading technique above will not go well in conjunction with shading the spread.

Figure 8.12 shows the widening of the yields during the crisis period. This is because the yield reflects the cost of borrowing of companies, which increased for firms with lower credit ratings during the crisis.

8.3.2   Spreads and Real GDP Growth

We again look at the investment grade bond market, but this time in relation to the economic cycle. During times of stress, we expect that higher-rated bonds would perform better than lower-rated bonds. To analyze this relationship, we look at the spreads between investment grade bonds and the real GDP growth rate. In our example, we look at this relationship over the last 20 years from 1994 to 2013.

Step 1: Import Moody's Aaa and Baa Yield Data from FRED   We have retrieved the Moody's Aaa and Baa yield data earlier, so we only have to merge the two data objects into `spread` then subset it to only contain yields between 1994 and 2013.

Moody's Aaa and Baa Index Yields From 2006 to 2013
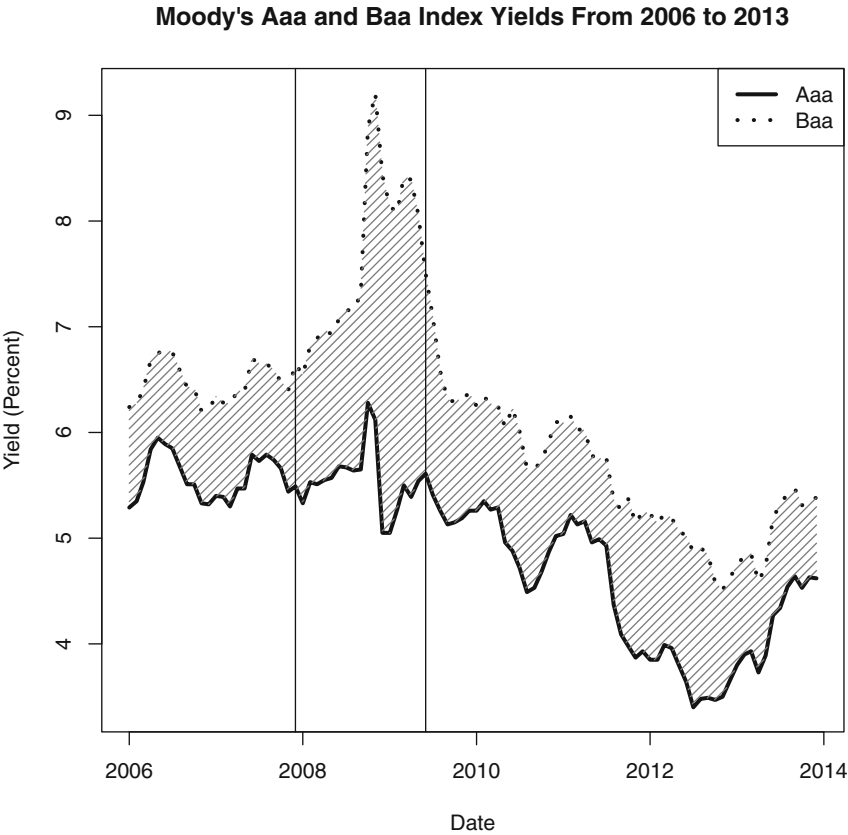
Fig. 8.12 Spread between investment grade bonds, 2006–2013. Data Source: Federal Reserve Electronic Database

```
> spread<-merge(aaa,baa)
> spread[1:3,]
            AAA  BAA
1919-01-01 5.35 7.12
1919-02-01 5.35 7.20
1919-03-01 5.39 7.15
>
> spread<-subset(spread,
+    index(spread)>="1994-01-01" &
+    index(spread)<="2013-12-31")
> spread[c(1:3,nrow(spread)),]
            AAA  BAA
1994-01-01 6.92 7.65
1994-02-01 7.08 7.76
1994-03-01 7.48 8.13
2013-12-01 4.62 5.38
```

Step 2: Calculate Spread Between Baa and Aaa Bonds We calculate the spread between investment grade bonds as the difference between the Baa yield and the Aaa yield converted into basis points. Note that the Baa yield is the higher of the two yields due to the increased premium of investing in lower-rated bonds, so the spread has to be positive. We also present the spread in basis points by multiplying the difference in Aaa and Baa yields by 100.

```
> spread$spread<-(spread$BAA-spread$AAA)*100
> spread[c(1:3,nrow(spread)),]
            AAA  BAA spread
1994-01-01 6.92 7.65     73
1994-02-01 7.08 7.76     68
1994-03-01 7.48 8.13     65
2013-12-01 4.62 5.38     76
```

Step 3: Import Real GDP Data from FRED We then retrieve real GDP data with symbol (GDPC1) from FRED. We download the series and save it to a file named **GDPC1 FRED.csv**.

```
> RGDP<-read.csv("GDPC1 FRED.csv",skip=10)
> RGDP$date<-as.Date(RGDP$observation_date,"%Y-%m-%d")
> RGDP$GDPC1<-as.numeric(as.character(RGDP$GDPC1))
> RGDP<-xts(RGDP$GDPC1,order.by=RGDP$date)
> names(RGDP)<-paste("GDPC1")
> RGDP[1:3,]
            GDPC1
1947-01-01 1932.6
1947-04-01 1930.4
1947-07-01 1928.4
```

Step 4: Calculate Year-Over-Year GDP Growth The real GDP data we retrieved is quarterly, so to calculate a year-over-year GDP growth rate we have to create a 4-period lag variable using the `Lag` command with `k=4`. We then combine the lag variable into `RGDP`, which as we can see from the output shows that we have correctly done the correct number of lags.

```
> Lag.GDP<-Lag(RGDP$GDPC1,k=4)
> Lag.GDP[1:6,]
            Lag.4
1947-01-01    NA
1947-04-01    NA
1947-07-01    NA
1947-10-01    NA
1948-01-01 1932.6
1948-04-01 1930.4
>
> RGDP$Lag.GDP<-Lag.GDP[,1]
> RGDP[1:6,]
            GDPC1 Lag.GDP
1947-01-01 1932.6      NA
```

```
1947-04-01 1930.4      NA
1947-07-01 1928.4      NA
1947-10-01 1958.8      NA
1948-01-01 1987.6  1932.6
1948-04-01 2019.9  1930.4
```

We then calculate the year-over-year change (in percentage points) as the percentage change between the value under GDPC1 and the value under Lag.GDP multiplied by 100. Using percentage points makes it easier for charting purposes.

```
> RGDP$YOYChg<-((RGDP$GDPC1-RGDP$Lag.GDP)/RGDP$Lag.GDP)*100
> RGDP[1:6,]
           GDPC1 Lag.GDP   YOYChg
1947-01-01 1932.6      NA       NA
1947-04-01 1930.4      NA       NA
1947-07-01 1928.4      NA       NA
1947-10-01 1958.8      NA       NA
1948-01-01 1987.6  1932.6 2.845907
1948-04-01 2019.9  1930.4 4.636345
```

We then subset the data to only include values from 1994 to 2013.

```
> RGDP<-subset(RGDP,
+     index(RGDP)>="1994-01-01" &
+     index(RGDP)<="2013-12-31")
> RGDP[c(1:3,nrow(RGDP)),]
              GDPC1 Lag.GDP   YOYChg
1994-01-01   9737.6  9414.0 3.437434
1994-04-01   9870.7  9469.9 4.232357
1994-07-01   9928.9  9516.1 4.337912
2013-07-01 15839.3 15534.0 1.965366
```

Step 5: Plot 2-axis Chart   We first set up two ranges for the two y-axes. Using the range command, we find that the values in spread range from 3.4bp to 338.0bp. As such, we set the y1.range equal to 0 to 350bp. For the range of real GDP growth rate, we use the ceiling and floor commands to get the next biggest integer and the next smallest integer, respectively. Note also the order with which we apply the range. We start with the positive number $(+6\%)$ and end with the negative number (-5%). This allows us to *invert* the axis when charting.

```
> range(spread)
[1]   3.4 338.0
> y1.range<-c(0,350)
> y1.range
[1]    0 350
> y2.range<-c(ceiling(max(range(RGDP$YOYChg))),
+     floor(min(range(RGDP$YOYChg))))
> y2.range
[1]  6 -5
```

Let us break the charting itself into several steps. First, we create the plot of the spread as though it was a normal single chart plot, but we add the line `par(mar= c(5,5,5,5))`. This creates enough space for us to add a second axis later. The rest of the code, including the shading for the two recession periods (March 2001–November 2001 and December 2007–June 2009), follows the same technique as our prior plotting codes.

```
> par(mar=c(5,5,5,5))
> plot(x=index(spread),
+     xlab="Date",
+     y=spread$spread,
+     ylab="Baa - Aaa Spread (bps)",
+     ylim=y1.range,
+     type="l",
+     main="Moody's Baa Minus Aaa Spread and Real GDP Growth
+     1994 - 2013")
> shade<-par("usr")
> shade
[1]  8475.04 16330.96   -14.00   364.00
> rect(as.Date("2007-12-01"),shade[2],
+     as.Date("2009-06-01"),shade[3],col="gray60",lty=0)
> rect(as.Date("2001-03-01"),shade[2],
+     as.Date("2001-11-01"),shade[3],col="gray60",lty=0)
> lines(x=index(spread),y=spread$spread,type="l")
```

Second, we now plot the data for the second axis. The `par(new=TRUE)` tells R that we are adding to an existing plot. The `xaxt="n"` and `yaxt="n"` suppresses the axes labels generated by this second plot command. We manually add the 2nd axis label using the `axis(4)` and `mtext` commands. The `side=4` denotes the right side of the chart and `line=3` starts the text on the 3rd line from the margin.

```
> par(new=TRUE)
> plot(x=index(RGDP),
+     xlab=" ",
+     xaxt="n",
+     y=RGDP$YOYChg,
+     ylab=" ",
+     yaxt="n",
+     ylim=y2.range,
+     type="l",
+     lty=2)
> axis(4)
> mtext("Year-over-Year Real GDP Growth (%)",side=4,line=3)
```

Finally, we add a legend to the plot. The legend is quite long, so it overlaps with the top of the recession period shading. The legend default is to have a transparent fill, so the recession shading actually penetrates the legend making it hard to read the text in the legend. To prevent this, we add the `bg="white"` argument to fill-in the legend background with the color white.
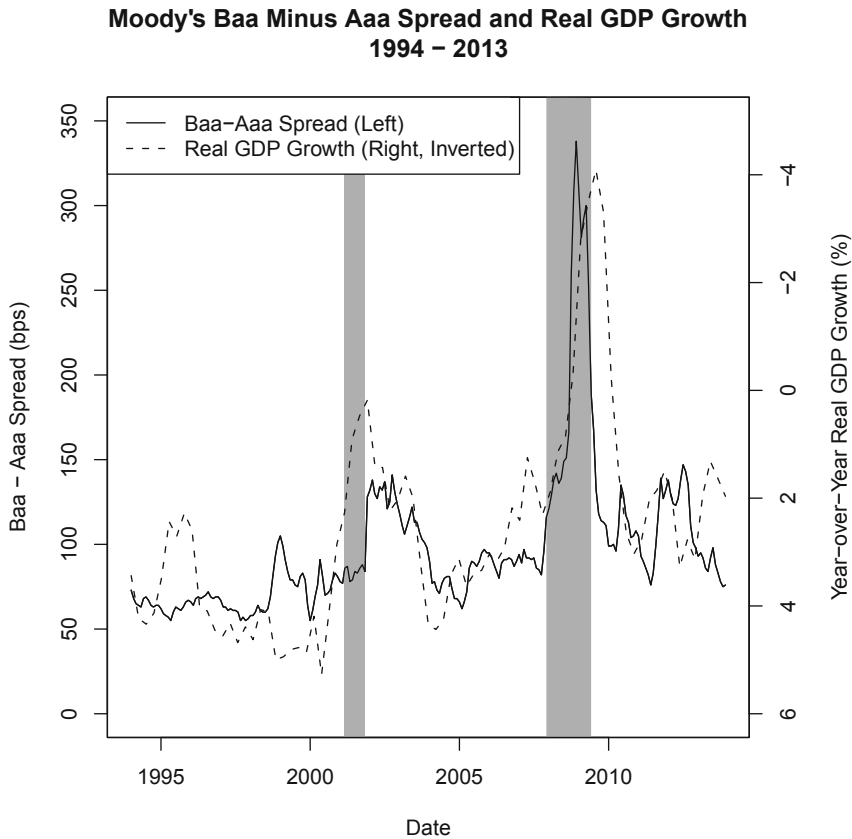
Fig. 8.13 Spread between investment grade bonds and the economic cycle, 1994–2013. Data Source: Federal Reserve Electronic Database

```
> legend("topleft",
+     c("Baa-Aaa Spread (Left)","Real GDP Growth (Right, Inverted)"),
+     lty=c(1,2),
+     bg="white")
```

Figure 8.13 shows that bond spreads spike shortly after (i.e., with a lag) the recession began. The biggest spike in spreads occurred during the 2008/2009 recession, where GDP growth was the most negative. Recall that the right axis was intentionally constructed to be inverted, so that a "spike" in real GDP growth is actually a huge negative year-over-year GDP growth.

8.4   Bond ETFs

Individual investors used to have limited choices when investing in bonds given the large minimum investment requirements of most fixed income securities. In recent years, the development of bond Exchange Traded Funds (ETFs), which are securities that trade like stock, individual investors can now gain exposure from investing in a portfolio of bonds. There are many types of bond ETFs available, such as those that represent the aggregate bond market, bonds of different credit ratings, or bonds in different sectors. In this section, we show how to compare the performance of investment grade (IG) and high-yield (HY) bond ETFs using Yahoo Finance data from December 31, 2010 to December 31, 2013.

Step 1: Import Data for Bond ETFs from Yahoo Finance We import Yahoo Finance data for the SPDR Barclays Aggregate Bond ETF (LAG) and SPDR Barclays High Yield Bond ETF (JNK). We should save the data for these two ETFs as **LAG Yahoo.csv** and **JNK Yahoo.csv**, respectively.

```
> data.LAG<-read.csv("LAG Yahoo.csv",header=TRUE)
> date<-as.Date(data.LAG$Date,format="%Y-%m-%d")
> data.LAG<-cbind(date, data.LAG[,-1])
> data.LAG<-data.LAG[order(data.LAG$date),]
> data.LAG<-xts(data.LAG[,2:7],order.by=data.LAG[,1])
> names(data.LAG)<-
+     paste(c("LAG.Open","LAG.High","LAG.Low",
+     "LAG.Close","LAG.Volume","LAG.Adjusted"))
> data.LAG[c(1:3,nrow(data.LAG)),]
           LAG.Open LAG.High LAG.Low LAG.Close LAG.Volume LAG.Adjusted
2010-12-31    55.49    55.56   55.38     55.56       9700        51.46
2011-01-03    55.41    55.56   55.38     55.54      10500        51.44
2011-01-04    55.58    55.60   55.46     55.46      11000        51.37
2013-12-31    56.45    56.46   56.33     56.40      18900        56.40
>
> data.JNK<-read.csv("JNK Yahoo.csv",header=TRUE)
> date<-as.Date(data.JNK$Date,format="%Y-%m-%d")
> data.JNK<-cbind(date, data.JNK[,-1])
> data.JNK<-data.JNK[order(data.JNK$date),]
> data.JNK<-xts(data.JNK[,2:7],order.by=data.JNK[,1])
> names(data.JNK)<-
+     paste(c("JNK.Open","JNK.High","JNK.Low",
+     "JNK.Close","JNK.Volume","JNK.Adjusted"))
> data.JNK[c(1:3,nrow(data.JNK)),]
           JNK.Open JNK.High JNK.Low JNK.Close JNK.Volume JNK.Adjusted
2010-12-31    39.68    39.75   39.60     39.71     986600        32.31
2011-01-03    39.87    39.95   39.70     39.87    4061900        32.44
2011-01-04    39.96    40.02   39.85     39.99    3602200        32.53
2013-12-31    40.53    40.58   40.51     40.56    2814300        40.56
```

Step 2: Combine Adjusted Close Price Data for LAG and JNK We now combine the two data objects using the cbind command. We then rename the variable names to make it easier to work with the data.

```
> bonds<-cbind(
+     data.frame(index(data.LAG)),
+     data.frame(data.LAG$LAG.Adjusted),
+     data.frame(data.JNK$JNK.Adjusted))
> bonds[c(1:3,nrow(bonds)),]
           index.data.LAG. LAG.Adjusted JNK.Adjusted
2010-12-31      2010-12-31        51.46        32.31
2011-01-03      2011-01-03        51.44        32.44
2011-01-04      2011-01-04        51.37        32.53
2013-12-31      2013-12-31        56.40        40.56
>
> colnames(bonds)<-paste(c("date","LAG","JNK"))
> bonds[c(1:3,nrow(bonds)),]
                  date   LAG   JNK
2010-12-31 2010-12-31 51.46 32.31
2011-01-03 2011-01-03 51.44 32.44
2011-01-04 2011-01-04 51.37 32.53
2013-12-31 2013-12-31 56.40 40.56
```

Step 3: Create Normalized Value for LAG and JNK We then create indexed values of LAG and JNK by dividing the price each day by the security's price on December 31, 2010. This technique normalizes the price to $ 1 as of December 31, 2010 and makes it easier for us to evaluate the performance of the ETFs going forward.

```
> bonds$LAG.idx<-bonds$LAG/bonds$LAG[1]
> bonds$JNK.idx<-bonds$JNK/bonds$JNK[1]
> bonds[c(1:3,nrow(bonds)),]
                  date   LAG   JNK   LAG.idx  JNK.idx
2010-12-31 2010-12-31 51.46 32.31 1.0000000 1.000000
2011-01-03 2011-01-03 51.44 32.44 0.9996113 1.004024
2011-01-04 2011-01-04 51.37 32.53 0.9982511 1.006809
2013-12-31 2013-12-31 56.40 40.56 1.0959969 1.255339
```

Step 4: Plot the Bond ETF Data We now chart the ETF data so we can visually compare the performance of investment grade and high yield bonds. We create a `y.range` variable to make sure the y-axis is large enough to encompass the values of both ETFs.

```
> y.range<-range(bonds[,4:5])
> y.range
[1] 0.9418137 1.2553389
> plot(x=bonds$date,
+     xlab="Date",
+     y=bonds$LAG.idx,
+     ylab="Value of Investment ($)",
+     ylim=y.range,
+     type="l",
+     col="black",
+     main="Value of $1 Invested in Bond ETFs
```

Value of $1 Invested in Bond ETFs
December 31, 2010 - December 31, 2013
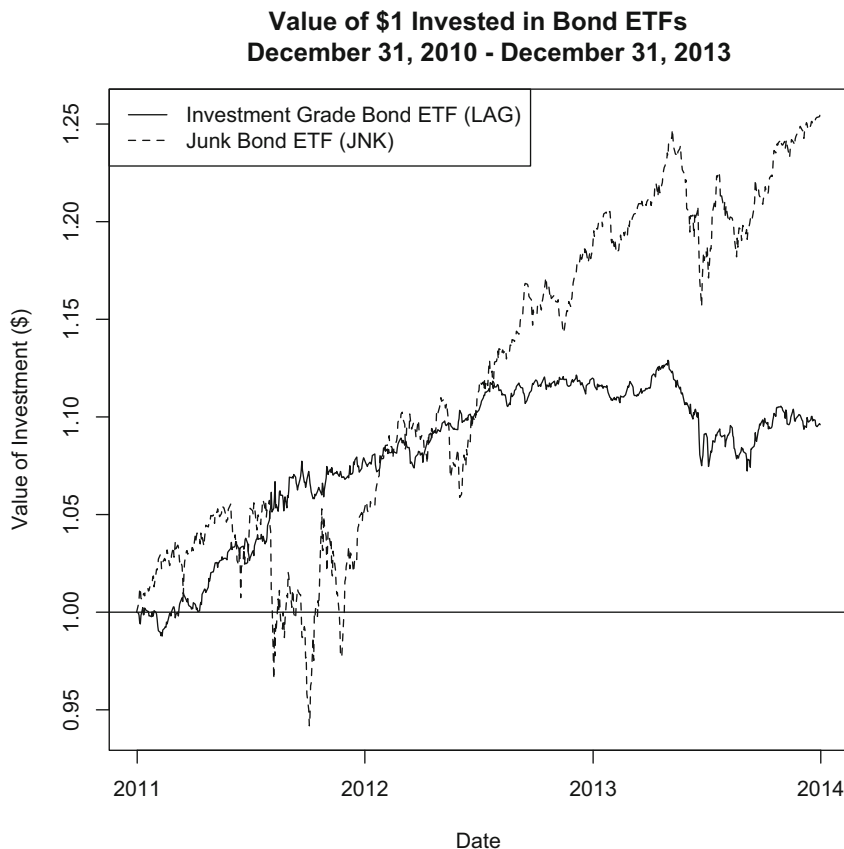
Fig. 8.14 Performance of investment grade and high-yield bond exchange traded funds, December 31, 2010–December 31, 2013. Reproduced with permission of CSI ©2013. Data Source: CSI www.csidata.com

```
+      December 31, 2010 - December 31, 2013")
> lines(x=bonds$date,y=bonds$JNK.idx,lty=2)
> abline(h=1,lty=1)
> legend("topleft",
+      c("Investment Grade Bond ETF (LAG)","Junk Bond ETF (JNK)"),
+      lty=c(1,2))
```

Figure 8.14 shows the chart of LAG and JNK. The chart shows that high-yield (HY) or junk bonds did better than investment grade (IG) bonds from 2011 to 2013. The chart shows that IG bonds are less volatile than HY bonds, which is what we would expect given that IG bonds are less risky.

8.5   Bond Valuation on Coupon Payment Dates

The intrinsic value of a bond is the present value of the coupon and principal payments bondholders expect to receive from owning the bond discounted at the appropriate risk-adjusted discount rate. Therefore, to value a bond, we would need to know the bond's promised cash flows and its yield to maturity (YTM). On the other hand, if we know the price of the bond and the bond's cash flows, we can then find the bond's YTM. In this section, we demonstrate how to implement these two calculations.

8.5.1   Pricing Vanilla Bonds with Known Yield-to-Maturity

We can calculate the value of a bond as the present value of its promised cash flows discounted using the yield to maturity.[5] That is,

$$V_{Bond} = \sum_{t=1}^{Tf} \frac{C/f}{(1 + y/f)^t} + \frac{FV}{(1 + y/f)^{Tf}}, \tag{8.1}$$

where C is the coupon payment, f is the frequency of the coupon payment, y is the yield to maturity of the bond, FV is the face value or par value of the bond, and T is the time to maturity of the bond (in years). This formula works best in the context of a plain vanilla bond (i.e., a bond with a fixed rate, fixed maturity, and no embedded options).

Equation (8.1) works well when we value the bond immediately after a coupon payment is made. The reason is that the promised future cash flows are in equally-spaced intervals and there is no accrued interest. This means that the investor who purchases a bond that settles on that day will get the full coupon on the next coupon payment date.[6]

Let us go through an example of how to use Eq. (8.1) to value a bond. Consider the Total Capital SA Medium Term Notes (TOTAL MTNs) due on December 8, 2017 that pays annual coupons with an annual rate of 4.25 %. As of December 9, 2013 (December 8, 2013 falls on a Sunday), the price for this TOTAL MTN was GBP1097 and its yield was 1.7 %.[7]

Step 1: Setup Variables for Bond Characteristics In this section, we assume we have formed an expectation as to what the yield on this bond should be and, based

---

[5] The methodology we show in this section to calculate the present value of a bond can be applied to a *Discounted Cash Flow* (DCF) analysis of other securities.

[6] The price that an investor pays for the bond, including the calculation of the accrued interest, is from the *settlement date* and not the trade date. The settlement date is usually the trade date plus three trading days (T + 3) for US corporates. However, the number of days after the trade date can vary substantially. For our purposes, we will assume the trade date and settlement date are the same, but we should keep in mind that the trade date and settlement date can be different.

[7] Data is obtained from Fixed Income Investor (UK), http://www.fixedincomeinvestor.co.uk.

on that belief, we calculate the bond price. We will assume the yield is 1.7 %, which consequently is the yield reported by Fixed Income Investor.

```
> rm(list=ls())
> coupon=0.0425
> maturity=4
> yield=0.017
> par=1000
> coupon.freq=1
```

Note that for purposes of this section, we will refer to the price or value of a bond in relation to the $ 1000 par value. This is not the typical way bond prices are quoted. In general, bond prices are quoted as a percentage of the par value. So a price of 99 means that we would have to pay $ 990 to purchase a bond with a par value of $ 1000. For simplicity, we will use the term price to refer to the $ 990 in the above example.

Step 2: Create Variables Accounting for the Periodicity of the Coupon Payments
First, we create a `period` variable that takes the value of the time to maturity multiplied by the coupon frequency. In our example, the 4-year time to maturity of the bond means that there will be four payment dates (i.e., 4 years multiplied by 1 payment per year). Second, `coupon.period` refers to the coupon payment made per period as a percentage of par. In our example, the TOTAL MTN pays GBP42.50 per year based on a 4.25 % coupon rate and GBP1000 par value. Third, the `yield.period` denotes the yield to maturity per period or 1.7 % YTM per year.

```
> periods=maturity*coupon.freq
> periods
[1] 4
```

```
> coupon.period=coupon/coupon.freq
> coupon.period
[1] 0.0425
> yield.period=yield/coupon.freq
> yield.period
[1] 0.017
```

Step 3: Construct a Vector of Applicable Per Period Coupon Rates We then setup a vector called `bond.coupon` that contains the coupon payments for each period. We do this by using the `rep` command, which repeats the `coupon.period` of 4.25 % for a certain number of `times` (i.e., four).

```
> bond.coupon<-rep(coupon.period,times=periods,
+         length.out=NA,each=1)
> bond.df<-as.data.frame(bond.coupon)
> bond.df
  bond.coupon
1      0.0425
2      0.0425
3      0.0425
4      0.0425
```

Step 4: Calculate Periodic Cash Flows, Yield to Maturity, and Present Value of Each Period's Cash Flows We now setup four additional variables: First, we calculate the `cf` variable that is equal to the periodic dollar coupon payments of GBP42.50 for all but the last payment period. GBP42.50 is calculated by multiplying the periodic coupon rate of 4.25 % by the par value of GBP1000. On the last payment period, the `cf` is equal to the periodic coupon payment of GBP4.25 plus the par value of GBP1000 (i.e., you get the principal back at maturity). Second, we create the `ytm` variable, which is equal to the periodic yield of 1.7 %. Third, we construct a `period` date, which is the number of payment periods. We use this when calculating the number of periods to discount the cash flows. Fourth, the `pv.cf` variable is equal to $cf/(1 + ytm)^{period}$.

```
> bond.df$cf<-coupon.period*par
> bond.df$cf[periods]=bond.df$cf[periods]+par
> bond.df$ytm<-yield.period
> bond.df$period<-c(1:periods)
> bond.df$pv.cf<-bond.df$cf/
+ ((1+bond.df$ytm)^bond.df$period)
> bond.df
  bond.coupon      cf ytm period   pv.cf
1      0.0425   42.5 0.017      1  41.790
2      0.0425   42.5 0.017      2  41.091
3      0.0425   42.5 0.017      3  40.404
4      0.0425 1042.5 0.017      4 974.523
```

Step 5: Calculate Value of Bond by Summing the Present Value of Each Period's Cash Flows The value of the bond is then calculated using the `sum` command and taking the summation of all observations under the `pv.cf` variable. This equals GBP1098, which is reasonably close to the GBP1097 price reported by Fixed Income Investor.

```
> value=sum(bond.df$pv.cf)
> value
[1] 1097.808
```

In this section, we discussed the use of yield-to-maturity to price a bond. In many instances, yield-to-worst is often typically quoted in the market. Yield-to-worst is basically the same calculation as yield-to-maturity except that it is performed across all dates in which the bond can be called and the maturity date. As such, the yield-to-worst for a non-callable bond will equal its yield-to-maturity. It may also be possible that the highest yield is the one that is at the maturity date, so in those situations the yield-to-worst will also equal the yield-to-maturity.

8.5.2 Vanilla Bond Pricing Function

In the previous section, we went through how to calculate the value of a bond in R by laying out all the bond's cash flows. In this section, we will create a bond valuation function `bondprc` that simplifies the way we calculate bond values so that we can

value many bonds easily. The structure of the code and the variable names are tied closely to the analysis in the prior section to assist in understanding how this function works.

```
> bondprc<-function(coupon,maturity,yield,par,coupon.freq){
+
+     periods=maturity*coupon.freq
+     coupon.period=coupon/coupon.freq
+     yield.period=yield/coupon.freq
+
+     bond.coupon<-rep(coupon.period,times=periods,length.out=NA,each=1)
+     bond.df<-as.data.frame(bond.coupon)
+     for (i in 1:periods) {
+         bond.df$cf[i]=par*coupon.period
+         bond.df$period[i]=i
+         bond.df$yield[i]=yield.period
+         }
+     bond.df$cf[periods]=bond.df$cf[periods]+par
+     bond.df$PV=bond.df$cf/((1+bond.df$yield)^bond.df$period)
+     bond.df
+     value=sum(bond.df$PV)
+     value
+     }
```

The above function uses a loop to calculate the bond's cash flows, period, and yield. This generates the same values as in the previous section but hides the actual calculations instead of reporting it.

To calculate the value of the TOTAL MTN from the example in the previous section, we can enter the inputs of the TOTAL MTN into the `bondprc` function we created. In the following order, we enter the annual coupon rate, time to maturity in years, the yield to maturity, the par value, and the coupon frequency. As the output shows, the value calculated using the function equals GBP1098, which is identical to the bond value we calculated in the prior section.

```
> bondprc(coupon,maturity,yield,par,coupon.freq)
[1] 1097.808
```

As we can see from the above calculations using the `bondprc` function, using a function reduces inputs to only one line. In addition, a function that has been verified does not need to be touched, which reduces any human error in making changes to the code. As we have verified the `bondprc` function returns the same value as the detailed bond cash flow calculation, we can be confident about repeatedly using this function without having to constantly re-check the mechanics of the calculation. We only need to make sure the inputs we apply to the function are accurate.

8.5.3   Finding Bond Yield-to-Maturity with Known Price

In the prior sections, we calculated the value of the bond when we know the yield to maturity. In this section, we will estimate the yield to maturity of the bond when we know the market price of the bond. The yield of the bond is important because it is a proxy for the cost of debt. Different bonds can have different yields to maturity, as the yield to maturity is a function of all the bond's features such as time to maturity and coupon rate.

Unlike calculating the value of the bond using a formula, we cannot do the same when finding the yield to maturity. Therefore, the only way to find the YTM is through trial-and-error. Fortunately, there are algorithms that we can use to make this trial-and-error process automated. To find the YTM, we use the `uniroot` function. We also create a `bondval` function that is a modification of the `bondprc` function to make it easier to calculate the YTM.[8]

Step 1: Create a Vector of the Bond's Cash Flows, with the Purchase Price as the First Observation and as a Negative Number Continuing from our TOTAL MTN example, we first construct a vector of the bond's cash flows. Make sure that there is at least one sign change in the cash flows. Typically, we make the price of the bond a cash outflow to denote that we are purchasing the bond and we make the bond's coupon payments and principal payment as cash inflows. Without the sign change, we will not be able to compute the YTM and the calculation returns an error.

```
> bond.cf<-c(-value,
+     rep(coupon.period*par,periods-1),
+     par+coupon.period*par)
> bond.cf
[1] -1097.808    42.500    42.500    42.500  1042.500
```

Step 2: Create Functions to Iteratively Find the Yield to Maturity We then create the first of two functions. This function calculates the value of the bond and we use the character `i` as the variable that represents the per period yield.

```
> bondval<-function(i,bond.cf,
+     t=seq(along=bond.cf)) sum(bond.cf/(1+i)^t)
```

The second function uses the `uniroot` function to find the root of the cash flows.

```
> bond.ytm <- function(bond.cf) {
+     uniroot(bondval,
+     c(0,1), bond.cf=bond.cf)$root}
```

---

[8] The YTM uses the same calculation as the *Internal Rate of Return* (IRR). As such, we can use the techniques we discuss in this section to calculate the IRR of other types of investments.

Step 3: Annualize the Periodic Yield to Maturity Calculated Above We then calculate the YTM of the TOTAL MTN by multiplying the calculated yield by the coupon frequency. The result of this calculation is a YTM of 1.7 %, which is identical to the YTM of 1.7 % reported by Fixed Income Investor.

```
> YTM<-bond.ytm(bond.cf)*coupon.freq
> YTM
[1] 0.01699932
> options(digits=3)
> YTM
[1] 0.017
> options(digits=7)
```

Note that the estimation procedure to calculate the YTM may cause minor differences from the actual YTM (especially differences due to rounding). We used the `digits=3` option to reduce the number of decimal places shown to demonstrate that the YTM is effectively 1.7 %, which is equal to the yield we used to calculate the bond value used to estimate the YTM in this calculation.

8.6 Duration and Convexity

Fixed income instruments are sensitive to changes in interest rates. As such, managing interest rate risk is an important consideration when dealing with fixed income portfolios. *Duration* and *convexity* are two techniques that help investors immunize against small changes in interest rates. The idea behind duration is that it estimates the impact of small changes in interest rates on the value of the fixed income instrument, and the idea behind convexity is to enhance this estimate by also considering the impact of the rate of the change in interest rates.

Macaulay duration, which can loosely be interpreted as the weighted-average time to maturity of the bond,[9] is calculated as:

$$Macaulay\,D = \frac{\sum_{t=1}^{T} t_t PV(CF_t)}{Price},$$ (8.2)

where t_t is the cash flow period and $PV(CF_t)$ is the present value of the period t cash flows, and $Price$ is the price of the bond.

Modified duration adjusts Macaulay duration by considering the yield of the bond, and gives the price sensitivity of the bond measured as the percentage change in price for a given change in the yield. Modified duration is calculated as:

$$Modified\,D = \frac{Macaulay\,D}{1 + YTM/freq},$$ (8.3)

---

[9] Technically, this definition only holds when coupons are reinvested at the YTM of the bond.

where YTM is the yield to maturity and $freq$ is the coupon frequency.

Below, we calculate the duration and convexity of the 4.25 % TOTAL MTN Due December 8, 2017. Recall that our estimated price for the TOTAL MTN on December 8, 2013 was GBP1098. We use our calculated price to show how well duration and convexity adjustments estimate the change in the price of a bond by comparing the resulting estimate to the full valuation of the bond by changing the yield by the same basis point change.

Step 1: Create Vector of Periods and Cash Flows for the Bond We create a `period` variable, which equals the number of the bond's cash flow payment periods. Then, we construct a vector `cf` of the bond's cash flows. This equals the GBP42.50 per coupon payment period prior to the last coupon payment date. On the last coupon payment date, the bond pays GBP42.50 of coupons plus the GBP1000 principal.

```
> price=value
> period<-seq(1,maturity*coupon.freq,by=1)
> cf<-c(rep(par*coupon/coupon.freq,
+    maturity*coupon.freq-1),
+    par*(1+coupon/coupon.freq))
> duration<-data.frame(period)
> duration$cf<-cf
> duration
  period     cf
1      1   42.5
2      2   42.5
3      3   42.5
4      4 1042.5
```

Step 2: Calculate Duration of Each Period's Cash Flows We need to create two variables before we can calculate the duration. First, we create the `tcf` variable, which represents the period multiplied by the cash flow (i.e., `period * cf`). Second, we calculate a column for the present value factor `PV.factor`. The `PV.factor` is equal to $1/(1+y)^t$, where y is the yield to maturity and t is the period. The duration `dur` for each period is equal to `tcf * PV.factor / price`.

```
> duration$tcf<-duration$period*duration$cf
> duration$PV.factor<-(1+yield/coupon.freq)^(-period)
> duration$dur<-(duration$tcf*duration$PV.factor)/price
> duration
  period     cf    tcf PV.factor        dur
1      1   42.5   42.5 0.9832842 0.03806638
2      2   42.5   85.0 0.9668478 0.07486013
3      3   42.5  127.5 0.9506861 0.11041317
4      4 1042.5 4170.0 0.9347946 3.55079667
```

Step 3: Calculate Macaulay Duration We can now calculate the Macaulay Duration `Mac.duration`, which equals the sum of the values in the `dur` column divided by the coupon frequency. We find the Macaulay Duration is equal to 3.77. This can be interpreted as the weighted-average time to maturity of the TOTAL MTN is 3.77 years. Note that, for a zero-coupon bond, the duration is equal to the time to maturity (TTM) because there are no intermediate cash flows. As such, for a coupon-paying bond, the duration is less than the TTM because the intermediate coupon payments shorten the weighted-average time of cash flows from the bond.

```
> Mac.duration<-sum(duration$dur)/coupon.freq
> Mac.duration
[1] 3.774136
```

Step 4: Calculate Modified Duration Using Eq. (8.3), we find the Modified Duration as 3.71.

```
> Mod.duration=Mac.duration/(1+yield/coupon.freq)
> Mod.duration
[1] 3.711049
```

Step 5: Calculate Convexity Adjustment We now turn to estimating the convexity of this bond. To calculate the convexity, we construct the variables `c1` and `c2`, where `c1` is equal to

$$(t^2 + t) * cf \tag{8.4}$$

and `c2` is equal to

$$price * (1 + YTM/2)^2. \tag{8.5}$$

Then, we calculate the per period convexity adjustment `conv` as

$$c1 \times PV.factor \times c2. \tag{8.6}$$

```
> convexity<-duration
> convexity$c1<-(period^2+period)*cf
> convexity$c2<-price*(1+yield/coupon.freq)^2
> convexity$conv<-(convexity$c1*convexity$PV.factor)/convexity$c2
> convexity
  period    cf    tcf PV.factor        dur    c1       c2        conv
1      1  42.5   42.5 0.9832842 0.03806638    85 1135.451  0.07360878
2      2  42.5   85.0 0.9668478 0.07486013   255 1135.451  0.21713505
3      3  42.5  127.5 0.9506861 0.11041317   510 1135.451  0.42701091
4      4 1042.5 4170.0 0.9347946 3.55079667 20850 1135.451 17.16539899
```

We can then find the convexity by taking the sum of the values in `conv` variable, which yields a convexity adjustment of 17.88.

```
> Convexity<-sum(convexity$conv)
> Convexity
[1] 17.88315
```

Step 6: Estimate Price Change Based on Duration Duration is typically the measure for a 100 bp change in yields. Suppose that yields go up by 100 bp, what is the estimated price change due to duration? The answer is given by the formula:

$$\Delta P = -D * \Delta y * P, \tag{8.7}$$

where ΔP is the change in price, D is the Modified Duration, Δy is the change in yield (in decimals), and P is the contemporaneous price of the bond. Note that in front of the Modified Duration is a minus sign, because there is a negative or inverse relationship between the change in yield and the change in the bond price. Put differently, an increase in yield will result in a lower bond price and, conversely, a decrease in yield will result in a higher bond price.

   To estimate the price change due to duration alone, we create a variable called `delta.yld`, which we assume is equal to an increase of 1 % or 100 basis points. Using Eq. (8.7), we find the estimated price *decline* based on duration alone of GBP40.74. As such, given a price of GBP1098 and an increase in yield of 100 bp, the estimated new price based on the duration adjustment is GBP1057.

```
> delta.yld<-0.0100
> delta.Dur<--Mod.duration*delta.yld*price
> delta.Dur
[1] -40.74019
> px.Dur<-price+delta.Dur
> px.Dur
[1] 1057.068
```

Step 7: Estimate Price Change Using Both Duration and Convexity Adjustment
We now turn to estimate the change in price based on duration and convexity. This is given by the formula:

$$\Delta P = P * [(-D * \Delta y) + (0.5 * C * (\Delta y)^2))], \tag{8.8}$$

where C is the Convexity and the other terms are the same as in Eq. (8.7). The estimate of the price *decline* with the duration and convexity adjustment is equal to $-$ GBP39.76. We find that the price of the TOTAL MTN after making the adjustments for Duration and Convexity is equal to GBP1058.

```
> delta.DurConv<-((-Mod.duration*delta.yld)+
+    (0.5*Convexity*(delta.yld^2)))*price
> delta.DurConv
[1] -39.75858
> px.DurConv<-price+delta.DurConv
> px.DurConv
[1] 1058.05
```

Step 8: Summarize Estimated Bond Price Based on Duration and Convexity Adjustment We can summarize the results of the duration and convexity calculations into one output.

```
> est.change<-cbind(price,delta.Dur,px.Dur,
+    delta.DurConv,px.DurConv)
> est.change
        price delta.Dur   px.Dur delta.DurConv px.DurConv
[1,] 1097.808 -40.74019 1057.068     -39.75858    1058.05
```

Step 9: Compare Estimates with Full Price Valuation at New Yield We can then compare the estimated price with the bond price using full valuation. To get the full valuation price of the bond, we use the `bondprc` function we constructed in the prior section. The value of the bond given a 100 bp increase in yield based on a full valuation is GBP1058, which is pretty close to the price of the bond as estimated using the Duration and Convexity adjustments.

```
> bondprc(coupon,maturity,yield+delta.yld,par,coupon.freq)
[1] 1058.031
```

8.7   Bond Valuation on Non-Coupon Payment Dates

Valuing bonds on non-coupon payment dates requires us to take into account accrued interest. The *clean price* is the price of the bond without any accrued interest. The accrued interest is the portion of the coupon the seller of the bond is entitled to when trying to value the bond in between coupon payment dates. For example, suppose we have a 4 % bond that pays semi-annual coupons. This means the bond pays 2 % of the par value every 6 months. If the bondholder sells the bond in 3 months, she is entitled to interest payments equal to 1 % of the par value, which will be tacked on to the clean price. Hence, on non-coupon payment dates, the value of the bond must include the remaining portion of the next coupon in addition to the discounted value of all future coupon payments and the principal payment. Mathematically, for a bond that pays f coupons per year, this is expressed as

$$V_{Bond} = \frac{FV}{(1+y/f)^{fT-1+n}} + \sum_{t=1}^{Tf} \frac{C}{(1+y/f)^{(t-1+n)}} + \frac{d}{360/f}C, \qquad (8.9)$$

where C is the periodic coupon payment, which is equal to $(r/f)FV$, r is the annual coupon rate, FV is the face value of the bond, T is the number of coupons to be paid from settlement date until maturity, n is the number of days to the next coupon divided by the days during each coupon period, and d is the number of days since the last coupon payment was made. d is divided by $360/f$ because the day-count convention for US/UK bonds is 30/360. This means that every calendar month is treated as though there were 30 days.

In this section, we value the same TOTAL MTN we did above, but this time we moved forward in time to January 8, 2014. On January 8, 2014, the TOTAL MTN had a yield of 1.8 % according to Fixed Income Investor. Using that yield, we can value the TOTAL MTN as of January 8, 2014.

Step 1: Move Relevant Dates Forward We will continue the calculations from our earlier example, so all variables are defined in the same manner unless explicitly stated. The first step is to move the settlement date forward to January 8, 2014 and the next coupon date as December 8, 2014. The maturity date remains at December 8, 2017. As such, we have four annual coupon payments remaining.

```
> settle.date<-as.Date("2014-01-08")
> next.coupon<-as.Date("2014-12-08")
> mat.date<-as.Date("2017-12-08")
> cpn.pmts<-4
```

Step 2: Identify Relevant Number of Days for the Accrued Interest Calculation
First, we have to find the number of days until the next coupon payment date. This is used to determine what portion of the interest belongs to the investor that purchased the bond on the settlement date.

```
> days.next.cpn<-as.numeric((next.coupon-settle.date))
> days.next.cpn
[1] 334
```

Second, we count the number of coupon periods per year based on a 360-day calendar year.

```
> days.cpn.per<-360/coupon.freq
> days.cpn.per
[1] 360
```

Finally, we calculate the number of days since the last coupon payment. This ratio is used to calculate the accrued interest owed to the seller of the bond.

```
> days.last.cpn<-days.cpn.per-days.next.cpn
> days.last.cpn
[1] 26
```

Step 3: Create Variable for Contemporaneous Yield   Suppose we believe the yield on the bond is now 1.8 %. We create a variable for the per period yield to use in our calculations.

```
> yield=0.018
> yield.period=yield
> yield.period
[1] 0.018
```

Step 4: Calculate the Present Value of the Principal Payment of the Bond   This is simply discounting the principal payment we expect to receive on December 8, 2017 to January 8, 2014. The only difference here and the valuation on a coupon payment date is that the discounting is not done on a full year but, instead, this calculation includes a stub period.

```
> pv.principal<-par/(1+(yield.period))^(cpn.pmts-
+     1+(days.next.cpn/days.cpn.per))
> pv.principal
[1] 932.3274
```

Step 5: Calculate the Present Value of the Coupon Payments   Again, this is an exercise in discounting from each coupon payment date back to January 8, 2014. The only difference in this case is that the discounting is not done in full years but includes a stub period.

```
> bond.cf<-rep(coupon.period*par,
+     times=cpn.pmts,
+     length.out=NA,
+     each=1)
> bond.cf
[1] 42.5 42.5 42.5 42.5
>
> bond.cf<-data.frame(bond.cf)
> bond.cf
  bond.cf
1    42.5
2    42.5
3    42.5
4    42.5
>
> bond.cf$period<-c(1:cpn.pmts)
> bond.cf
  bond.cf period
1    42.5      1
2    42.5      2
3    42.5      3
4    42.5      4
>
```

```
> bond.cf$disc<-(1+yield.period)^(bond.cf$period-
+    1+(days.next.cpn/days.cpn.per))
> bond.cf
  bond.cf period     disc
1    42.5      1 1.016689
2    42.5      2 1.034990
3    42.5      3 1.053619
4    42.5      4 1.072585

> bond.cf$value<-bond.cf$bond.cf/bond.cf$disc
> bond.cf
  bond.cf period    disc    value
1    42.5      1 1.016689 41.80235
2    42.5      2 1.034990 41.06321
3    42.5      3 1.053619 40.33715
4    42.5      4 1.072585 39.62391
```

Using the `sum` command, we can add all the numbers under the `value` column to arrive at the PV of the coupon payments `pv.coupons`.

```
> pv.coupons<-sum(bond.cf$value)
> pv.coupons
[1] 162.8266
```

Step 6: Calculate Accrued Interest The *accrued interest* is based directly on the percentage of days within the period in which the seller held the bond through the settlement date. The accrued interest is not discounted.

```
> interest<- -(par*(coupon.period)*(days.last.cpn/days.cpn.per))
> interest
[1] -3.069444
```

Step 7: Estimate the Value of the Bond by Adding the Results of Steps 4 Through 6 The calculations show that the PV of the principal we will receive is GBP 932, the PV of the coupons we will receive is GBP 163, and the accrued interest we owe to the seller is GBP 3. As such, combining these together yields a bond value of GBP 1092.

```
> bond.value<-pv.principal+pv.coupons+interest
> bond.value
[1] 1092.085
```

Note that this bond value is pretty close to the January 8, 2014 price for the TOTAL MTN of GBP 1092.64. The `interest` variable above is *added* to the `pv.principal` and `pv.coupons` because the value of the accrued interest calculated in Step 6 is negative (i.e., we are taking the perspective that this is a cash outflow).

8.8   Further Reading

An excellent overview of the bond markets can be found in Thau [5]. More details on bond calculations can be found in Fabozzi [3] and Tuckman and Serrat [6]. Two excellent books on bond strategies are Crescenzi [1] and Huggins and Schaller [4].

There are additional resources on the web from which we can obtain up-to-date fixed income information. US bond prices can be obtained from Yahoo Finance Bonds Center (http://bonds.yahoo.com). A wealth of information about bonds can be found on various regulatory websites. For example, the website of the Financial Industry Regulatory Authority (FINRA) (http://www.finra.org) contains, among other things, information on US corporate and agency data. Another regulator, the Municipal Securities Rulemaking Board (MSRB) provides information and market data through its Electronic Municipal Market Access (EMMA) website (http://emma.msrb.org/). Sterling (UK) corporate bond and bond index prices, as well as other information on Sterling bond markets can be obtained from Fixed Income Investor (http://www.fixed incomeinvestor.co.uk).

References

1. Crescenzi, A. (2010). *The strategic bond investor: Strategies and tools to unlock the power of the bond market* (2nd ed.). New York McGraw-Hill.
2. Estrella, A., & Trubin, M. (2006). The yield curve as a leading indicator: Some practical issues. *Current Issues in Economics and Finance, 12,* 1–7.
3. Fabozzi, F. (2009). *Bond markets, analysis and strategies* (7th ed.). New Jersey: Prentice-Hall.
4. Huggins, D., & Schaller, C. (2013). *Fixed income relative value analysis: A practitioner's guide to the theory, tools, and trades*. United Kingdom: Bloomberg Press.
5. Thau, A. (2011). *The bond book* (3rd ed.). New York McGraw-Hill.
6. Tuckman, B., & Serrat, A. (2012). *Fixed income securities: Tools for today's markets* (3rd ed.). New Jersey: Wiley.

Chapter 9
Options

One of the most important formulas in all of finance is the options pricing model (OPM) developed by Black and Scholes [2] and Merton [6] (BSM). Many traders use the intuition behind the BSM OPM in their trading strategies. A complete discussion and derivation of the BSM OPM is beyond the scope of this book. However, we can go through a simplified example to show the underlying principle of how options are priced under the BSM framework.

Consider a stock that sells for $ 70 per share today and this stock could be worth either $ 100 per share or $ 50 per share tomorrow. How would we value an option to buy 100 shares of the stock for $ 80 per share tomorrow? This value is derived from the fact that we make the choice to purchase the stock tomorrow *after* we know what the stock price is. Put differently, we have to pay something for the right to buy the stock tomorrow only when its price is higher than $ 80 tomorrow. To do this, we can look at the value of this choice at the end of the day tomorrow. If the stock price is $ 100, we would make $ 2000 [= max($ 100 − $ 80 or 0) * 100 shares]. If the stock price is $ 50, we would not do anything because the *intrinsic value* of the option is zero (i.e., max($ 50 − $ 80, 0) = 0).

The BSM OPM is based on the *principle of no arbitrage*. Loosely speaking, if we have two assets that have the same cash flows, these two assets must have the same price. In the context of the BSM OPM, if we can find a portfolio of stocks and bonds that replicate the payoff of the option (i.e., $ 2000 when the stock price is $ 100 and 0 if the stock price is $ 50), the price of the portfolio of stocks and bonds must equal the price of the option.

Let us assume that we purchased 40 shares of the stock today. You would pay a price of $ 2800 [= 40 shares * $ 70 per share]. If the stock price goes up to $ 100 tomorrow, the 40 shares of stock we hold is now worth $ 4000 [= 40 shares * $ 100 per share]. On the other hand, if the stock price goes down to $ 50 tomorrow, the 40 shares of stock we hold is now worth $ 2000 [= 40 shares * $ 50 per share]. This means that the payoff from holding 40 shares of stock is always $ 2000 more than the payoff from holding an option to buy 100 shares of the stock, regardless of the state of the world tomorrow. Based on this, if no arbitrage opportunities exist, we know that the price of the option today must be $ 2000 less than the price of the 40 shares

of stock today. That is, the option price is equal to $ 800 for 100 shares [= $ 2800 price for 40 shares of stock − $ 2000] or $ 8 per share.

In this chapter, we first look at how to get options chain data from Yahoo Finance. We then show how to calculate call and put option values using the Black-Scholes-Merton (BSM) options pricing model (OPM). We then show various important concepts in options pricing that come out of the BSM OPM. We look at put-call parity, calculating the Greeks, and estimating implied volatility. Then, we show how we can use implied volatility to gauge how much risk investors believe exist in the market. Finally, we end the chapter by showing how to calculate option prices using a Binomial Model.

9.1   Obtaining Options Chain Data

The *options chain* is a list of traded options for a particular company's stock at a given point in time. A publicly-available source for options chain data is Yahoo Finance. The data on Yahoo Finance for options is provided by Interactive Data Real-Time Services. On the web, we can see what options data are available for each stock by clicking on the **Options** link on the left panel when we are viewing a security's data on Yahoo Finance. The different option expiration dates going forward for which data are available can be found on the middle of the screen. The available option expiration dates will change depending on when we access Yahoo Finance.

There is no choice to download options chain data on Yahoo Finance in a CSV file, but we can download options chain data from Yahoo Finance using the `quantmod` package and the `getOptionChain` command. Unfortunately, given that there is no access to historical options data, the output below can only be reproduced by importing CSV files of the call and put options that can be constructed at the end of Step 2 below. Steps 1 and 2 are still shown to demonstrate how to obtain the raw options data from Yahoo Finance, so we can extend this analysis to other option contracts.

In our example, we stand on January 1, 2014 and want to get the options chain for Amazon.com (AMZN) for options that expire approximately 3 months out. For our purpose, we will use the options contracts that expire March 2014 expiration.[1] We then want to test whether there are inconsistencies with the option price relative to the intrinsic value of the option.

A call (put) option gives the holder the right but not the obligation to buy (sell) the stock for the *strike price* through the *time to maturity*. Therefore, if it is not profitable for the option holder to exercise the option, he will let the option expire. However, if the intrinsic value of the option is positive (i.e., the stock price is greater than the strike price for a call and the strike price is greater than the stock price for a put), the option holder will exercise the option. In such a case, the option holder will receive

---

[1] Amazon.com options chain data as retrieved from Yahoo Finance are reproduced with permission from Interactive Data Real Time Services.

the stock and pay the strike price in a call option and will sell the stock and receive the strike price in a put option.

Step 1: Import Options Chain Data for AMZN March 2014 Options from Yahoo Finance The `getOptionChain` command takes two arguments: the ticker of the underlying stock (e.g., AMZN for Amazon) and the expiration year/month of the contracts (i.e., `Exp="2014-03"`).

```
> options<-getOptionChain("AMZN",Exp="2014-03")
> head(options)
$calls
                    Strike   Last   Chg   Bid    Ask Vol  OI
AMZN140322C00300000    300  99.80 -1.64 99.80 102.05   5   4
AMZN140322C00305000    305 101.10  0.00 94.95  97.35  10   2
AMZN140322C00320000    320  80.71  0.00 81.25  83.35  20  20
AMZN140322C00340000    340  63.00 -0.85 63.80  64.90   1   3
AMZN140322C00350000    350  56.00  0.00 55.70  56.70   2   4
AMZN140322C00370000    370  39.10  1.40 40.70  41.45   1  12
AMZN140322C00375000    375  34.62  0.00 37.35  37.85   1   1
AMZN140322C00380000    380  34.00  2.65 34.10  34.80   2   9
AMZN140322C00385000    385  34.91  0.00 31.15  31.60   1   6
AMZN140322C00390000    390  27.65  2.05 28.20  28.60  12  32
AMZN140322C00395000    395  25.30  1.75 25.50  25.85  25  32
AMZN140322C00400000    400  22.85  1.95 22.95  23.30 102 159
AMZN140322C00405000    405  20.68  1.58 20.55  20.95  25  96
AMZN140322C00410000    410  18.35  1.15 18.40  18.70  71  55
AMZN140322C00415000    415  16.05  1.25 16.45  16.85   9  57
AMZN140322C00420000    420  14.60  0.70 14.60  14.90  24  18
AMZN140322C00425000    425  12.95  1.05 12.95  13.30   4  13
AMZN140322C00430000    430  11.05  0.55 11.40  11.70   1  22
AMZN140322C00435000    435  10.05  0.00 10.05  10.35   3   3
AMZN140322C00440000    440   8.85  0.50  8.85   9.10   6  58
AMZN140322C00445000    445   7.78  0.45  7.75   8.00   8   3
AMZN140322C00450000    450   6.85  0.60  6.80   7.15  40  31
AMZN140322C00455000    455   7.15  0.00  5.95   6.20   5   5
AMZN140322C00460000    460   5.10 -0.20  5.20   5.45  10   9
AMZN140322C00465000    465   4.60 -0.90  4.50   4.75   4  10
AMZN140322C00470000    470   3.77  0.00  3.95   4.15   1   2
AMZN140322C00480000    480   3.00 -1.21  2.92   3.25   5   1
AMZN140322C00520000    520   1.55  0.00  1.02   1.23   4   4
AMZN140322C00525000    525   1.44  0.00  0.91   1.09   1   1

$puts
                    Strike  Last  Chg   Bid   Ask Vol  OI
AMZN140322P00200000    200  0.25 0.00  0.08  0.29  45  45
AMZN140322P00240000    240  0.55 0.00  0.43  0.65  30  30
AMZN140322P00245000    245  0.67 0.00  0.50  0.76  21  12
AMZN140322P00250000    250  0.79 0.00  0.56  0.86 409 405
AMZN140322P00255000    255  0.90 0.00  0.64  0.93   3  12
AMZN140322P00260000    260  1.02 0.00  0.72  1.02   3 403
AMZN140322P00265000    265  1.01 0.00  0.82  1.06   2   8
AMZN140322P00270000    270  1.18 0.00  0.93  1.16 208 208
AMZN140322P00275000    275  1.31 0.00  1.07  1.29  12  12
AMZN140322P00280000    280  1.45 0.00  1.22  1.43  10 400
```

```
AMZN140322P00285000     285  1.64  0.00  1.38  1.59  12  12
AMZN140322P00290000     290  1.77  0.00  1.53  1.76   2   2
AMZN140322P00295000     295  1.93  0.00  1.74  1.96   5   5
AMZN140322P00300000     300  1.99 -0.09  1.97  2.20  10  18
AMZN140322P00310000     310  2.63  0.00  2.59  2.79   1   1
AMZN140322P00320000     320  3.89  0.00  3.35  3.55 150  61
AMZN140322P00325000     325  4.25  0.00  3.80  4.10   1   1
AMZN140322P00330000     330  5.00  0.00  4.30  4.50 274 181
AMZN140322P00335000     335  5.15 -0.55  4.95  5.20   1   5
AMZN140322P00340000     340  6.10  0.00  5.65  5.90 100 142
AMZN140322P00350000     350  7.52 -0.93  7.30  7.55   1 177
AMZN140322P00355000     355  8.70 -0.60  8.30  8.55   2  56
AMZN140322P00360000     360  9.75 -0.85  9.40  9.70  72 116
AMZN140322P00365000     365 11.39 -0.61 10.70 10.95   3   8
AMZN140322P00370000     370 12.45 -1.03 12.10 12.40   9  30
AMZN140322P00375000     375 14.20 -0.07 13.60 13.90  42   3
AMZN140322P00380000     380 16.50 -1.08 15.40 15.70  24  17
AMZN140322P00385000     385 18.42 -1.22 17.25 17.60   3  52
AMZN140322P00390000     390 20.00 -1.15 19.35 19.65   8  60
AMZN140322P00395000     395 22.30 -2.25 21.55 21.95  15  20
AMZN140322P00400000     400 24.50 -2.42 24.05 24.40  87  34
AMZN140322P00405000     405 27.60 -2.60 26.75 27.00   5 608
AMZN140322P00440000     440 52.00 -2.00 49.85 50.45   8   7
AMZN140322P00450000     450 58.70 -3.50 56.85 58.70   1   5

$symbol
[1] "AMZN"
```

Step 2: Separate Calls and Puts and Keep Only Strike Price, Last Price, and Volume of each Option Contract For our analysis of intrinsic value below, we only need to use the strike price, last price, and volume for each option contract. In addition, we will analyze puts and calls separately, so we split the data into those two option types

```
> calls<-options$calls[,c(1:2,6)]
> calls
                    Strike   Last Vol
AMZN140322C00300000    300  99.80   5
AMZN140322C00305000    305 101.10  10
AMZN140322C00320000    320  80.71  20
AMZN140322C00340000    340  63.00   1
AMZN140322C00350000    350  56.00   2
AMZN140322C00370000    370  39.10   1
AMZN140322C00375000    375  34.62   1
AMZN140322C00380000    380  34.00   2
AMZN140322C00385000    385  34.91   1
AMZN140322C00390000    390  27.65  12
AMZN140322C00395000    395  25.30  25
AMZN140322C00400000    400  22.85 102
AMZN140322C00405000    405  20.68  25
AMZN140322C00410000    410  18.35  71
AMZN140322C00415000    415  16.05   9
AMZN140322C00420000    420  14.60  24
```

```
AMZN140322C00425000     425   12.95    4
AMZN140322C00430000     430   11.05    1
AMZN140322C00435000     435   10.05    3
AMZN140322C00440000     440    8.85    6
AMZN140322C00445000     445    7.78    8
AMZN140322C00450000     450    6.85   40
AMZN140322C00455000     455    7.15    5
AMZN140322C00460000     460    5.10   10
AMZN140322C00465000     465    4.60    4
AMZN140322C00470000     470    3.77    1
AMZN140322C00480000     480    3.00    5
AMZN140322C00520000     520    1.55    4
AMZN140322C00525000     525    1.44    1
> puts<-options$puts[,c(1:2,6)]
> puts
                     Strike  Last Vol
AMZN140322P00200000     200  0.25   45
AMZN140322P00240000     240  0.55   30
AMZN140322P00245000     245  0.67   21
AMZN140322P00250000     250  0.79  409
AMZN140322P00255000     255  0.90    3
AMZN140322P00260000     260  1.02    3
AMZN140322P00265000     265  1.01    2
AMZN140322P00270000     270  1.18  208
AMZN140322P00275000     275  1.31   12
AMZN140322P00280000     280  1.45   10
AMZN140322P00285000     285  1.64   12
AMZN140322P00290000     290  1.77    2
AMZN140322P00295000     295  1.93    5
AMZN140322P00300000     300  1.99   10
AMZN140322P00310000     310  2.63    1
AMZN140322P00320000     320  3.89  150
AMZN140322P00325000     325  4.25    1
AMZN140322P00330000     330  5.00  274
AMZN140322P00335000     335  5.15    1
AMZN140322P00340000     340  6.10  100
AMZN140322P00350000     350  7.52    1
AMZN140322P00355000     355  8.70    2
AMZN140322P00360000     360  9.75   72
AMZN140322P00365000     365 11.39    3
AMZN140322P00370000     370 12.45    9
AMZN140322P00375000     375 14.20   42
AMZN140322P00380000     380 16.50   24
AMZN140322P00385000     385 18.42    3
AMZN140322P00390000     390 20.00    8
AMZN140322P00395000     395 22.30   15
AMZN140322P00400000     400 24.50   87
AMZN140322P00405000     405 27.60    5
AMZN140322P00440000     440 52.00    8
AMZN140322P00450000     450 58.70    1
```

We save `calls` into **AMZN Mar 2014 Calls (Interactive Data).csv** and `puts` into **AMZN Mar 2014 Puts (Interactive Data).csv**.

```
> write.csv(AMZN.options$calls,"AMZN Mar 2014 Calls (Interactive Data).csv")
```

```
> write.csv(AMZN.options$puts,"AMZN Mar 2014 Puts (Interactive Data).csv")
```

Step 2 (CSV Import) To reproduce the results at the end of Step 2, we import the file **AMZN March 2014 Calls (Interactive Data).csv** and **AMZN March 2014 Puts (Interactive Data).csv**.[2]

```
> calls.csv<-read.csv("AMZN Mar 2014 Calls (Interactive Data).csv")
> calls.csv[c(1:3,nrow(calls.csv)),]
                    X Strike   Last   Chg   Bid    Ask Vol OI
1  AMZN140322C00300000    300  99.80 -1.64 99.80 102.05   5  4
2  AMZN140322C00305000    305 101.10  0.00 94.95  97.35  10  2
3  AMZN140322C00320000    320  80.71  0.00 81.25  83.35  20 20
29 AMZN140322C00525000    525   1.44  0.00  0.91   1.09   1  1
> puts.csv<-read.csv("AMZN Mar 2014 Puts (Interactive Data).csv")
> puts.csv[c(1:3,nrow(puts.csv)),]
                    X Strike   Last  Chg   Bid   Ask Vol OI
1  AMZN140322P00200000    200   0.25  0.0  0.08  0.29  45 45
2  AMZN140322P00240000    240   0.55  0.0  0.43  0.65  30 30
3  AMZN140322P00245000    245   0.67  0.0  0.50  0.76  21 12
34 AMZN140322P00450000    450  58.70 -3.5 56.85 58.70   1  5
```

We then clean up the call and put options data by only keeping the `Strike` (Column 2), `Last` (Column 3), and `Vol` (Column 7) columns. We ignore the labeling in the first column as this does not seem to add additional information beyond what is contained in the `Strike` and the knowledge that these are AMZN call or put options due March 22, 2014. As such, using observation numbers simplifies the data output.

```
> calls<-calls.csv[,c(2,3,7)]
> calls
   Strike   Last Vol
1     300  99.80   5
2     305 101.10  10
3     320  80.71  20
4     340  63.00   1
5     350  56.00   2
6     370  39.10   1
7     375  34.62   1
8     380  34.00   2
9     385  34.91   1
10    390  27.65  12
11    395  25.30  25
12    400  22.85 102
13    405  20.68  25
14    410  18.35  71
15    415  16.05   9
16    420  14.60  24
17    425  12.95   4
18    430  11.05   1
19    435  10.05   3
20    440   8.85   6
```

---

[2] The data in this file is reproduced with permission from Interactive Data Real-Time Services.

```
21    445   7.78   8
22    450   6.85  40
23    455   7.15   5
24    460   5.10  10
25    465   4.60   4
26    470   3.77   1
27    480   3.00   5
28    520   1.55   4
29    525   1.44   1
> puts<-puts.csv[,c(2,3,7)]
> puts
   Strike  Last Vol
1     200  0.25  45
2     240  0.55  30
3     245  0.67  21
4     250  0.79 409
5     255  0.90   3
6     260  1.02   3
7     265  1.01   2
8     270  1.18 208
9     275  1.31  12
10    280  1.45  10
11    285  1.64  12
12    290  1.77   2
13    295  1.93   5
14    300  1.99  10
15    310  2.63   1
16    320  3.89 150
17    325  4.25   1
18    330  5.00 274
19    335  5.15   1
20    340  6.10 100
21    350  7.52   1
22    355  8.70   2
23    360  9.75  72
24    365 11.39   3
25    370 12.45   9
26    375 14.20  42
27    380 16.50  24
28    385 18.42   3
29    390 20.00   8
30    395 22.30  15
31    400 24.50  87
32    405 27.60   5
33    440 52.00   8
34    450 58.70   1
```

Step 3: Determine Consistency of Market Price and Intrinsic Value We know that the intrinsic value of a call option is equal to $\max[S_t - K, 0]$ and the intrinsic value of a put option is equal to $\max[K - S_t, 0]$, where S_t is the stock price today and K is the strike price of the option. However, on dates prior to its expiration, the price of an option should be higher than its intrinsic value because there is a chance that the option value will end up higher (i.e., the option contains time value). As such, if

the price of the option is less than the option's intrinsic value, we have to consider the cause of such a result as well as whether we should discard such data. Below we create a variable called `diff`, which is the difference between the AMZN closing stock price on December 31, 2013 of \$ 398.79 and the strike price of the option. We then create a dummy variable that flags us with a value of zero if the difference is greater than the last price of the option.

```
> stock.price<-398.79
>
> calls$diff<-stock.price-calls$Strike
> calls$dummy<-ifelse(calls$diff<calls$Last,1,0)
> calls
   Strike   Last Vol     diff dummy
1     300  99.80   5    98.79     1
2     305 101.10  10    93.79     1
3     320  80.71  20    78.79     1
4     340  63.00   1    58.79     1
5     350  56.00   2    48.79     1
6     370  39.10   1    28.79     1
7     375  34.62   1    23.79     1
8     380  34.00   2    18.79     1
9     385  34.91   1    13.79     1
10    390  27.65  12     8.79     1
11    395  25.30  25     3.79     1
12    400  22.85 102    -1.21     1
13    405  20.68  25    -6.21     1
14    410  18.35  71   -11.21     1
15    415  16.05   9   -16.21     1
16    420  14.60  24   -21.21     1
17    425  12.95   4   -26.21     1
18    430  11.05   1   -31.21     1
19    435  10.05   3   -36.21     1
20    440   8.85   6   -41.21     1
21    445   7.78   8   -46.21     1
22    450   6.85  40   -51.21     1
23    455   7.15   5   -56.21     1
24    460   5.10  10   -61.21     1
25    465   4.60   4   -66.21     1
26    470   3.77   1   -71.21     1
27    480   3.00   5   -81.21     1
28    520   1.55   4  -121.21     1
29    525   1.44   1  -126.21     1
>
> puts$diff<-puts$Strike-stock.price
> puts$dummy<-ifelse(puts$diff<puts$Last,1,0)
> puts
   Strike Last Vol    diff dummy
1     200 0.25  45 -198.79     1
2     240 0.55  30 -158.79     1
3     245 0.67  21 -153.79     1
4     250 0.79 409 -148.79     1
5     255 0.90   3 -143.79     1
6     260 1.02   3 -138.79     1
```

```
7     265  1.01    2 –133.79    1
8     270  1.18 208 –128.79    1
9     275  1.31  12 –123.79    1
10    280  1.45  10 –118.79    1
11    285  1.64  12 –113.79    1
12    290  1.77    2 –108.79    1
13    295  1.93    5 –103.79    1
14    300  1.99  10  –98.79    1
15    310  2.63    1  –88.79    1
16    320  3.89 150  –78.79    1
17    325  4.25    1  –73.79    1
18    330  5.00 274  –68.79    1
19    335  5.15    1  –63.79    1
20    340  6.10 100  –58.79    1
21    350  7.52    1  –48.79    1
22    355  8.70    2  –43.79    1
23    360  9.75  72  –38.79    1
24    365 11.39    3  –33.79    1
25    370 12.45    9  –28.79    1
26    375 14.20  42  –23.79    1
27    380 16.50  24  –18.79    1
28    385 18.42    3  –13.79    1
29    390 20.00    8   –8.79    1
30    395 22.30  15   –3.79    1
31    400 24.50  87    1.21    1
32    405 27.60    5    6.21    1
33    440 52.00    8   41.21    1
34    450 58.70    1   51.21    1
```

The data we use appears clean as our output for both the call and put options show only 1s under the `dummy` column. Similar to checking the price data in Chap. 1, what we did in this section is an approach to test whether the options data retrieved is free from errors or if we need to perform further investigation of the data.

9.2   Black-Scholes-Merton Options Pricing Model

In the Black-Scholes-Merton (BSM) Options Pricing Model (OPM), options are valued based on the value of a portfolio of stocks and bonds that replicate the cash flows of the option. The beauty of the BSM OPM is that we do not need to know stochastic calculus to implement the model, as the BSM OPM is a closed-form model. That is,

$$c = SN(d1) + K \exp(-rf * T)N(d2) \tag{9.1}$$

and

$$p = K \exp(-rf * T)N(-d2) - SN(-d1), \tag{9.2}$$

where

$$d1 = \frac{ln(S/K) + (rf + 0.5 * \sigma^2) * T}{\sigma\sqrt{T}}$$

and

$$d2 = d1 - \sigma\sqrt{T}.$$

In the above equations, S is the underlying stock price, K is the strike price of the option, T is the time to maturity of the option, rf is the risk-free rate, σ is the volatility of the underlying asset, and $N()$ is the standard normal density.

In this section, we will estimate the value of March 2014 AMZN call and put options using the BSM OPM. Specifically, we will value the two call and put options that are closest to being at the money as of December 31, 2013. Given that the price of AMZN stock as of December 31, 2013 was $ 398.79, we will choose options with strike prices of $ 395 and 400.

Step 1: Setup Observable Inputs to the BSM OPM Aside form the strike price of the options, there are three other observable inputs to the BSM OPM. These are the stock price, risk-free rate, and time to maturity of the option. We first create variables for AMZN's close price as of December 31, 2013 and the risk-free rate as of December 31, 2013 as represented by the 3-Month Constant Maturity Treasury yield.

```
> price=stock.price
> price
[1] 398.79
>
> rf=0.0007
```

Next, we calculate the time to maturity of the option. The March 2014 options expire on March 22, 2014 and our valuation date is December 31, 2013. We then convert this time to maturity as a fraction of a year.

```
> expiry.date<-as.Date("2014-03-22")
> value.date<-as.Date("2013-12-31")
> TTM<-as.numeric((expiry.date-value.date)/365)
> TTM
[1] 0.2219178
```

Step 2: Estimate Historical Volatility as Proxy for Volatility Used in BSM OPM
Of the inputs to the BSM OPM, the volatility is the only unobservable input. As such, to calculate the option value, we need to have an input for the stock's volatility. A starting point could be the historical volatility of the stock. Below, we calculate the annualized daily volatility of AMZN returns over the last 3 years. Note that to annualize the daily volatility, we multiply the standard deviation of daily returns by the square root of 252.

```
> stock<-read.csv("AMZN Yahoo.csv",header=TRUE)
> date<-as.Date(stock$Date,format="%Y-%m-%d")
> stock<-cbind(date, stock[,-1])
> stock<-stock[order(stock$date),]
> library(xts)
> stock<-xts(stock[,2:7],order.by=stock[,1])
> names(stock)<-
+   paste(c("Open","High","Low","Close","Vol","Adj"))
> stock[c(1:3,nrow(stock)),]
              Open   High    Low  Close      Vol     Adj
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
>
> volatility<-stock[,6]
> volatility[c(1:3,nrow(volatility)),]
              Adj
2010-12-31 180.00
2011-01-03 184.22
2011-01-04 185.01
2013-12-31 398.79
>
> volatility$Ret<-diff(log(volatility$Adj))
> volatility[c(1:3,nrow(volatility)),]
              Adj         Ret
2010-12-31 180.00          NA
2011-01-03 184.22 0.023173845
2011-01-04 185.01 0.004279182
2013-12-31 398.79 0.013684318
>
> volatility<-volatility[-1,]
> volatility[c(1:3,nrow(volatility)),]
              Adj         Ret
2011-01-03 184.22 0.023173845
2011-01-04 185.01 0.004279182
2011-01-05 187.42 0.012942210
2013-12-31 398.79 0.013684318
>
> hist.vol<-sd(volatility$Ret)*sqrt(252)
> hist.vol
[1] 0.3259855
>
> vol=hist.vol
```

Step 3: Subset the Call Options to the Two Closest Strike Prices to Being at the Money Based on AMZN's stock price of $ 398.79, we will keep the options with a strike price of $ 395 and 400. For our current application, we only need the Strike and Last columns.

```
> bs.call<-subset(calls[,1:2],
+    calls$Strike == 395 |
+    calls$Strike == 400)
> rownames(bs.call)<-c(1,2)
> bs.call
  Strike  Last
1    395 25.30
2    400 22.85
```

Step 4: Calculate the Black-Scholes-Merton Value of these Two Call Options
We first calculate the value of d1 and d2, then we apply Eq. (9.1) to find the option
value of $ 26.26 for the call with a strike of $ 395 and 23.87 for the call with a strike
of $ 400. Note that both BSM OPM estimates are higher than the last price of the
call options reported on Yahoo Finance.

```
> d1<-(log(price/bs.call$Strike)+(rf+0.5*(vol^2))*TTM)/(vol*sqrt(TTM))
> d1
[1] 0.13997756 0.05806618
> d2<-d1-vol*sqrt(TTM)
> d2
[1] -0.01358817 -0.09549955
> bs.call$optval<-price*pnorm(d1,mean=0,sd=1)-
+    bs.call$Strike*exp(-rf*TTM)*pnorm(d2,mean=0,sd=1)
> bs.call
  Strike  Last    optval
1    395 25.30 26.26364
2    400 22.85 23.87290
```

**Step 5: Subset the Put Options to the Two Closest Strike Prices to Being at the
Money** We again will keep the options with strike prices of $ 395 and 400.

```
> bs.put<-subset(puts[,1:2],
+    puts$Strike == 395 |
+    puts$Strike == 400)
> rownames(bs.put)<-c(1,2)
> bs.put
  Strike Last
1    395 22.3
2    400 24.5
```

Step 6: Calculate the Black-Scholes-Merton Value of these Two Put Options
Using Eq. (9.2) and the results of d1 and d2 calculated above, we can calculate the
value of the put options. The output shows that the put with a strike of $ 395 has a
value of $ 22.41 and the put with a strike of $ 400 has a price of $ 25.02. Note also
that the BSM OPM estimates are slightly higher than the last price of the put options
reported on Yahoo Finance.

```
> nd1=-d1
> nd1
[1] -0.13997756 -0.05806618
> nd2=-d2
> nd2
[1] 0.01358817 0.09549955
> bs.put$optval<-bs.put$Strike*
+     exp(-rf*TTM)*pnorm(nd2,mean=0,sd=1)-
+     price*pnorm(nd1,mean=0,sd=1)
> bs.put
  Strike Last   optval
1    395 22.3 22.41228
2    400 24.5 25.02076
```

9.3   Black-Scholes-Merton OPM Function

If we were to repeatedly calculate option values using Black-Scholes-Merton, we would benefit from converting the above calculation into a function. We allow the function to be flexible enough that identifying whether the option type is a Call or a Put will allow us to calculate the approrpiate option value. We accomplish this using an if statement in the code. Note that to denote equality we have to use *two* equal signs (i.e., ==) in the code.

```
> bs.opm <- function(S,K,T,riskfree,sigma,type){
+       d1<-(log(S/K)+(riskfree+0.5*sigma^2)*T)/(sigma*sqrt(T))
+       d2<-d1-sigma*sqrt(T)
+       if(type=="Call"){
+        opt.val<-S*pnorm(d1)-K*exp(-riskfree*T)*pnorm(d2)
+           }
+       if(type=="Put"){
+        opt.val<-K*exp(-riskfree*T)*pnorm(-d2)-S*pnorm(-d1)
+          }
+     opt.val
+     }
```

We can now calculate the value of the call and put options we calculated in the previous section. As we can see, our BSM function calculates identical values to the manual approach we used in the previous section.

```
> bs.opm(price,395,TTM,rf,vol,"Call")
[1] 26.26364
> bs.opm(price,400,TTM,rf,vol,"Call")
[1] 23.8729
> bs.opm(price,395,TTM,rf,vol,"Put")
[1] 22.41228
> bs.opm(price,400,TTM,rf,vol,"Put")
[1] 25.02076
```

9.4 Put-Call Parity

If we have the value of the call (put) option under BSM, we can find the value of
a put (call) option on the same underlying with the same strike price and maturity
using a relationship known as *put-call parity*. Put-call parity states that

$$p + S = c + K \exp(-rf * T), \tag{9.3}$$

where all the terms are defined in the same manner as Eqs. (9.1) and (9.2). To find
the value of a put option, we manipulate Eq. (9.3) to get

$$p = c + K \exp(-rf * T) - S. \tag{9.4}$$

Similarly, to get the value of the call option, we use the following formula:

$$c = p + S - K \exp(-rf * T). \tag{9.5}$$

We perform two implementations of put-call parity. First, we show that the put-call
parity holds for option values calculated using the BSM OPM.

```
> bs.call$optval.pcparity<-bs.put$optval-bs.put$Strike*exp(-rf*TTM)+price
> bs.call
  Strike  Last   optval optval.pcparity
1    395 25.30 26.26364        26.26364
2    400 22.85 23.87290        23.87290
> bs.put$optval.pcparity<-bs.call$optval+bs.put$Strike*exp(-rf*TTM)-price
> bs.put
  Strike Last   optval optval.pcparity
1    395 22.3 22.41228        22.41228
2    400 24.5 25.02076        25.02076
```

As the otuput shows, `optval` and `optval.pcparity` are identical.
We calculated `optval` by directly applying the BSM OPM. We calculated
`optval.pcparity` using put-call parity. This result demonstrates that put-call
parity holds for option values calculated using the BSM OPM.

Second, we show that put-call parity may not hold if we used the market price of
the options. As the output below shows, the values of `Last` and `Last.pcparity`
are not the same. This is because the assumptions of BSM OPM may not hold in
practice.

```
> bs.call$Last.pcparity<-bs.put$Last-bs.put$Strike*exp(-rf*TTM)+price
> bs.call
  Strike  Last   optval optval.pcparity Last.pcparity
1    395 25.30 26.26364        26.26364      26.15136
2    400 22.85 23.87290        23.87290      23.35213
> bs.put$Last.pcparity<-bs.call$Last+bs.put$Strike*exp(-rf*TTM)-price
> bs.put
  Strike Last   optval optval.pcparity Last.pcparity
1    395 22.3 22.41228        22.41228      21.44864
2    400 24.5 25.02076        25.02076      23.99787
```

9.5 The Greeks

When estimating the BSM value of the option, we may be interested in the sensitivity of the option price to changes in the inputs. These sensitivities are known as the Greeks. Specifically, the Greeks are Delta (change in option price given a change in the underlying price), Gamma (change in option price given the rate of change in the underlying price), Vega (change in option price given a change in volatility), Theta (change in option price given a change in the time to maturity), and Rho (change in option price given a change in the risk-free rate).

We continue using the two call options with strike prices that are closest to being at the money. We calculate the Greeks for each of those options. Since we only need the Strike and Last price for these options, we can subset the `bs.call` data object and only keep the first two columns.

```
> greeks.call<-bs.call[,1:2]
> greeks.call$delta<-pnorm(d1,mean=0,sd=1)
> greeks.call$gamma<-dnorm(d1,mean=0,sd=1)/(price*vol*sqrt(TTM))
> greeks.call$vega<-price*dnorm(d1,mean=0,sd=1)*sqrt(TTM)
> greeks.call$theta<--((price*vol*dnorm(d1,mean=0,sd=1))/
+   (2*sqrt(TTM))) - (rf*greeks.call$Strike*exp(-rf*TTM)*
+   pnorm(d2,mean=0,sd=1))
> greeks.call$rho<- greeks.call$Strike*TTM*exp(-rf*TTM)*
+   pnorm(d2,mean=0,sd=1)
> greeks.call$type<-c("call")
> greeks.call
  Strike Last    delta      gamma     vega    theta      rho type
1    395 25.30 0.5556611 0.006450848 74.21568 -54.64618 43.34687 call
2    400 22.85 0.5231520 0.006503383 74.82009 -55.08270 41.00040 call
```

Similarly, we calculate the Greeks for the two put options that are closest to being in the money. For the puts, we subset the `bs.put` data object and only keep the first two columns.

```
> greeks.put<-bs.put[,1:2]
> greeks.put$delta<-pnorm(d1,mean=0,sd=1)-1
> greeks.put$gamma<-dnorm(d1,mean=0,sd=1)/(price*vol*sqrt(TTM))
> greeks.put$vega<-price*dnorm(d1,mean=0,sd=1)*sqrt(TTM)
> greeks.put$theta<--((price*vol*dnorm(d1,mean=0,sd=1))/
+   (2*sqrt(TTM))) + (rf*greeks.put$Strike*exp(-rf*TTM)*
+   pnorm(nd2,mean=0,sd=1))
> greeks.put$rho<- -greeks.put$Strike*TTM*exp(-rf*TTM)*
+   pnorm(nd2,mean=0,sd=1)
> greeks.put$type<-c("put")
> rownames(greeks.put)<-c(1,2)
> greeks.put
  Strike Last    delta      gamma     vega    theta      rho type
1    395 22.3 -0.4443389 0.006450848 74.21568 -54.36972 -44.29705  put
2    400 24.5 -0.4768480 0.006503383 74.82009 -54.80274 -47.75293  put
```

9.6   Implied Volatility

Of the five inputs to the BSM OPM, the asset volatility is the one that is not observable. As such, it is often of interest to investors to know what the market thinks the volatility of the underlying asset is. The options' traded price can be thought of as the market's assessment of the value of the option and, if we assume that the only variable that is difficult to determine objectively is the volatility, we can use the BSM OPM to "solve" for the volatility implied by the market price.

The implied volatility is essentially found using trial and error. However, we do not need to do this manually (although we could). There are several numerical methods that can be used to perform this trial-and-error algorithm. We will use the Bisection Method.[3] The Bisection Method essentially first attempts some value for the volatility and compares that value to the market price of the option (`price`). If the difference between the two values is greater in absolute value terms than the acceptable error (`epsilon`), another volatility number is used which is half of the original volatility and either a volatility that is higher (`sigma.up`) or lower (`sigma.down`). That is where the name bisection comes from. The process repeats until we reach an acceptable level of difference between the estimated price and the market price of the option (i.e., the absolute value of the difference must be less than 0.00001 in our example) or if we get to 1000 iterations and no solution is found. The latter is needed because issues with the data may cause the iterations to go on an infinite loop. The use of 1000 iterations is generally sufficient to reach a reasonable answer, as those situations in which the program does not reach a solution after 1000 iterations typically result in wacky outputs.

```
> iv.opt<- function(S,K,T,riskfree,price,type){
+    sigma<- vol
+    sigma.up<- 1
+    sigma.down<- 0.001
+    count<- 0
+    epsilon<- bs.opm(S,K,TTM,riskfree,sigma,type)-price
+    while(abs(epsilon)>0.00001 && count<1000){
+        if(epsilon<0){
+           sigma.down < -sigma
+           sigma <- (sigma.up + sigma)/2
+           } else{
+              sigma.up<- sigma
+              sigma<- (sigma.down+sigma)/2
+              }
+           epsilon<- bs.opm(S,K,TTM,riskfree,sigma,type)-price
+           count<- count+1
+           }
+        if(count==1000){
+        return(NA)
+           } else{
+              return(sigma)
+              }
+        }
```

---

[3] This code below for the Bisection Method used in the implied volatility calculation is based on code available at http://www.r-bloggers.com/the-only-thing-smiling-today-is-volatility/.

To test that this works, we can reverse the calculations for the call option with a strike of $ 395. We calculated a BSM OPM value for that option of $ 26.26364. As the output shows, we get a volatility of 0.326, which is equal to the historical volatility we used when we calculated the BSM OPM value of this particular call option.

```
> iv.opt(stock.price,395,TTM,rf,26.26364,"Call")
[1] 0.3259855
> hist.vol
[1] 0.3259855
```

9.7 Gauging Market Risk

In this section, we plot the CBOE Volatility Index (VIX) over the last 10 years. The VIX was introduced in 1993 and is intended to measure the market's expectation of 30-day volatility as it calculates the implied volatility embedded in S&P 500 Index options with approximately 1-month to maturity. The VIX is often referred to as the "investor fear gauge," because it reflects the investors' view of expected future stock market volatility.

Step 1: Import VIX Data from FRED The CBOE Volatility Index with symbol VIXCLS available from FRED starts in 2004. We save this data to a CSV file labeled **VIXCLS FRED.csv** in the R working directory.

```
> vix<- read.csv("VIXCLS FRED.csv",skip=10)
> vix$date<- as.Date(vix$observation_date,"%Y-%m-%d")
> vix$VIXCLS<- as.numeric(as.character(vix$VIXCLS))
> vix<- xts(vix$VIXCLS,order.by=vix$date)
> names(vix)<- paste("VIXCLS")
> vix[c(1:3,nrow(vix)),]
           VIXCLS
2004-01-02  18.22
2004-01-05  17.49
2004-01-06  16.73
2014-01-16  12.53
```

Step 2: Subset the Data to End in 2013 The VIX data already begins in 2004, so we only need to subset the data to end in December 2013.

```
> vix<- subset(vix,index(vix)<=as.Date("2013-12-31"))
> vix[c(1:3,nrow(vix)),]
           VIXCLS
2004-01-02  18.22
2004-01-05  17.49
2004-01-06  16.73
2013-12-31  13.72
```

Step 3: Import S&P 500 Index Data from FRED The S&P 500 Index uses the symbol SP500. The data on FRED begins in 1957. We save this data to a CSV file labeled **SP500 FRED.csv** in the R working directory.

```
> spx<- read.csv("SP500 FRED.csv",skip=10)
> spx$date<- as.Date(spx$observation_date,"%Y-%m-%d")
> spx$SP500<- as.numeric(as.character(spx$SP500))
Warning message:
NAs introduced by coercion
> spx<- xts(spx$SP500,order.by=spx$date)
> names(spx)<- paste("SP500")
> spx[c(1:3,nrow(spx)),]
             SP500
1957-01-02   46.20
1957-01-03   46.60
1957-01-04   46.66
2014-01-03 1831.37
```

Step 4: Remove NAs from S&P 500 Data The data retrieved from FRED has NAs for holidays, such as Thanksgiving and Christmas. For our analysis, we would like to have 30 trading days of data to calculate the standard deviation. In anticipation of the function we will use below, we have to make sure that we calculate returns properly and that no NAs are left in the data. We do this by using the `na.omit` command. As the output below shows, we had 14,873 observations prior to deleting the NAs. After deleting the NAs, we end up with 14,352 observations.

```
> nrow(spx)
[1] 14873
>
> spx<- na.omit(spx)
> nrow(spx)
[1] 14352
```

Step 5: Calculate Log Returns for the S&P 500 Index We use the `diff` and `log` commands to calculate the daily logarithmic returns.

```
> spx$ret<- diff(log(spx$SP500))
> spx[c(1:3,nrow(spx)),]
             SP500             ret
1957-01-02   46.20              NA
1957-01-03   46.60  0.0086207430
1957-01-04   46.66  0.0012867255
2014-01-03 1831.37 -0.0003330285
```

Step 6: Calculate the Rolling 30-Day Standard Deviation of S&P 500 Index Returns We use a combination of the `rollapply` and `sd` functions to make the rolling standard deviation calculation. The `na.rm=TRUE` option tells R to ignore any NAs in the data. We then annualize the daily standard deviation by multiplying it by the square root of 252. For compariability with how the VIX data is presented, we multiply the result by 100 so that the final result will be in percentage points.

```
> spx.sd<- rollapply(spx$ret,30,sd,na.rm=TRUE)*sqrt(252)*100
> spx[c(1:3,nrow(spx)),]
            SP500           ret
1957-01-02   46.20            NA
1957-01-03   46.60  0.0086207430
1957-01-04   46.66  0.0012867255
2014-01-03 1831.37 -0.0003330285
```

Step 7: Subset the Data to 2004–2013 Note that the S&P 500 Index data begins in 1957. However, for comparability with the time period available for the VIX, we limit the S&P 500 Index data from 2004 to 2013. I also renamed the column to `Hist.Vol` so that the column name is more informative.

```
> spx.sd<- subset(spx.sd,index(spx.sd)>=as.Date("2004-01-02") &
+    index(spx.sd)<=as.Date("2013-12-31"))

> colnames(spx.sd)<- "Hist.Vol"
> spx.sd[c(1:3,nrow(spx.sd)),]
            Hist.Vol
2004-01-02 10.020015
2004-01-05 10.307824
2004-01-06  9.783665
2013-12-31  8.568639
```

Step 8: Combine VIX and S&P 500 Index Data and Check for NAs in the Data
We then merge the two data series together. Using the `all=TRUE` option allows us to make sure all the dates in both series appear.

```
> vol.idx<- merge(vix,spx.sd,all=TRUE)
> vol.idx[c(1:20,nrow(vol.idx)),]
           VIXCLS  Hist.Vol
2004-01-02  18.22 10.020015
2004-01-05  17.49 10.307824
2004-01-06  16.73  9.783665
2004-01-07  15.50  9.777873
2004-01-08  15.61  8.961196
2004-01-09  16.75  9.547491
2004-01-12  16.82  9.557716
2004-01-13  18.04  9.778368
2004-01-14  16.75  9.570953
2004-01-15  15.56  9.449614
2004-01-16  15.00  9.482028
2004-01-20  15.21  9.497729
2004-01-21  14.34  9.162500
2004-01-22  14.71  9.199384
2004-01-23  14.84  8.716726
2004-01-26  14.55  9.075067
2004-01-27  15.35  9.426974
2004-01-28  16.78 10.488907
2004-01-29  17.14 10.294170
2004-01-30  16.63 10.282766
2013-12-31  13.72  8.568639
>
> vol.idx[! complete.cases(vol.idx),]
     VIXCLS Hist.Vol
```

Using the `all=TRUE` option creates the potential for creating NAs in the data. As such, we use the last command to check whether there are NAs in the data. As the result shows, there are no NAs in the data as only the headers "VIXCLS" and "Hist.Vol" show up with no corresponding observations. Had there been NAs in the data, each observation would have been displayed using the command above.

Step 9: Plot the Data We use the `plot` command to plot the data. To make sure we have the correct range of volatility values, we use the `range` command to get the minimum and maximum values of `vol.idx`.

```
> y.range<- range(vol.idx)
> y.range
[1]  5.848424 80.860000
> plot(x=index(vol.idx),
+     y=vol.idx$VIXCLS,
+     ylab="Volatility",
+     xlab=" ",
+     ylim=y.range,
+     type="l",
+     col="gray40",
+     main="Implied and Historical Market Volatility From 2004 to 2013")
> lines(x=index(vol.idx),y=vol.idx$Hist.Vol,lty=2)
> legend("topright",
+     c("VIX","Historical Volatility"),
+     lty=c(1,2),
+     col=c("gray40","black"))
```

Figure 9.1 shows the output of our plot. The figure shows that the VIX is generally higher than the historical volatility from 2004 to 2013. However, the shape of the two volatility estimates are generally the same.

9.8 Binomial OPM

Most options problems in practice cannot be solved using a closed-form solution (e.g., Black-Scholes-Merton). Numerical methods have to be implemented to solve such issues. These models can become extremely complex. In this section, we will discuss the binomial options pricing model, from which we can learn the basic intuition of how these types of models work. Moreover, when certain criteria are met, the option value calculated from the binomial model with converge to the result of the Black-Scholes-Merton model.

The *binomial model* is a lattice-based or tree-based model that allows us to model the path of the underlying asset's price in discrete time steps. To describe how the binomial model works, consider a stock with value V today. In a binomial model, the value in 6 months can either go up by u or go down by d. That is, the value of the stock can either be Vu or Vd at the end of 6 months. At the end of the year, the value of the stock depends on whether you start at node Vu or node Vd. Starting in node

Implied and Historical Market Volatility From 2004 to 2013
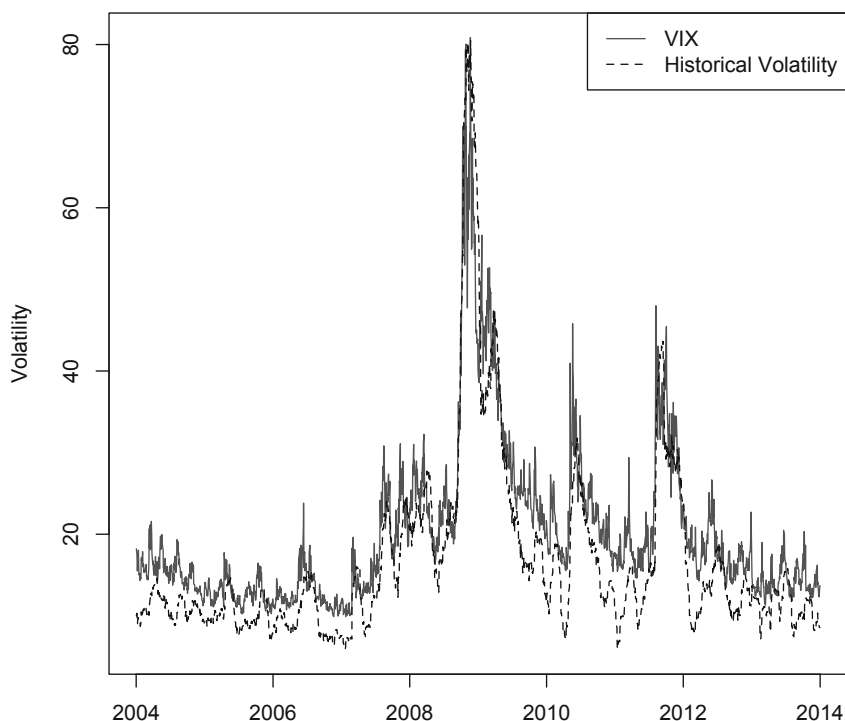
Fig. 9.1 CBOE volatility index and 30-day historical volatility of the S&P 500 index. Data Source: CBOE data is provided by Chicago Board Options Exchange, Incorporated (CBOE), and CBOE makes no warranties of any kind with respect to this data. S&P 500 Index data is provided by S&P Dow Jones Indices. Data obtained from Federal Reserve Electronic Database

Vu, we can either go up to node Vuu or go down to node Vud. Starting in node Vd, we can either go up to node Vdu or go down to node Vdd. Note that one feature of the binomial tree is that since u and d are fixed increments, the tree recombines. That is $Vud = Vdu$. Below is a graphical representation of how the underlying stock price can move as time passes.
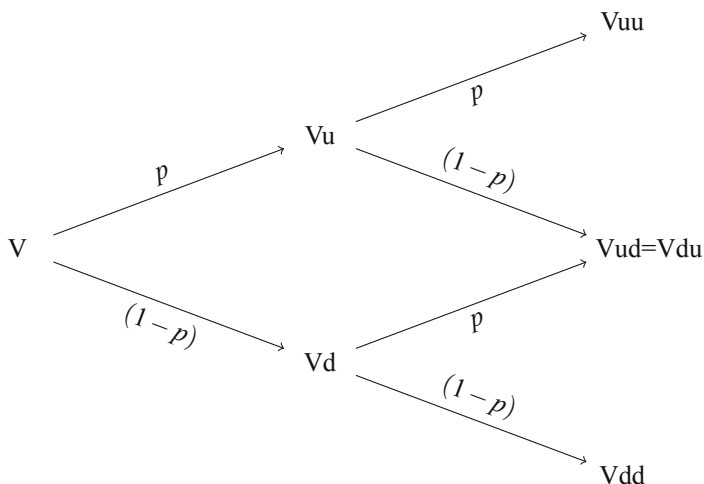
The above shows that there is a *risk-neutral probability p* that denotes the likelihood that the stock would go up.[4] Therefore, the likelihood that the stock would go down is $(1 - p)$. This will be useful in determining the value of the option.

We can now put some numbers to show how to calculate the option value using the Binomial Model. Consider a call option on Amazon stock with a strike price of $ 395 expiring in March 2014 (TTM = 0.222). Amazon's stock price on December 31, 2013 is $ 398.79 and the volatility of the underlying stock is 32.6 %. Using the risk-free rate of 0.07 %, what is the value of the call option using the Binomial Model?

First, we have to calculate the up move factor, u, which is calculated as

$$u = \exp(\sigma * \sqrt{dt}) = 1.1147, \tag{9.6}$$

where σ is the volatility of the underlying asset and dt is the time increment (i.e., TTM/number of periods in tree $= 0.222/2 = 0.111$). We can then calculate the down move factor, d, as

$$d = 1/u = 1/1.1147 = 0.8971. \tag{9.7}$$

Below, we show how the stock price evolves based on the u and d we calculated above. The stock price today is $ 398.79. At the end of the first period, our stock price can go from $ 398.79 to either $ 444.53 [= $ 398.79 * 1.1147] or $ 357.75 [= $ 398.79 * 0.8971]. If the stock price was at $ 444.53 at the end of the first period, it

---

[4] Note that the risk-neutral probability is not the same as real-world probabilities or subjective probabilities that we are more commonly exposed to. However, there is a specific set of risk-neutral probabilities that correspond to a set of real-world probabilities. In pricing derivatives, it is almost always easier to use risk-neutral probabilities because we can use the risk-free rate to discount cash flows instead of having to deal with a risk premium in the discount rate.

can either end up at $ 495.52 [= $ 444.53 * 1.1147] or $ 398.79 [= $ 444.53 * 0.8971] by the end of the second period. If the stock price was at $ 357.75 at the end of the second period, it can end up at $ 398.79 [= $ 357.75 * 1.1147] or $ 320.94 [= $ 357.75 * 0.8971] by the end of the second period.
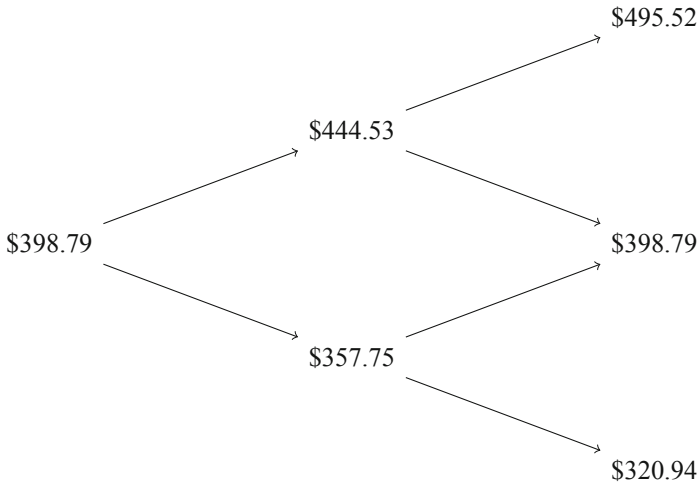
Then, we calculate the risk-neutral probability of an up move p as

$$
\begin{aligned}
p &= ((1 + r * dt) - d)/(u - d) \\
&= ((1 + 0.0007 * 0.111) - 0.8971)/(1.1147 - 0.8971) \\
&= 0.4732,
\end{aligned}
\tag{9.8}
$$

where r is the risk-free rate. Therefore, we have $1 - p = 0.5268$. We then add the risk-neutral probabilities to our binomial tree.
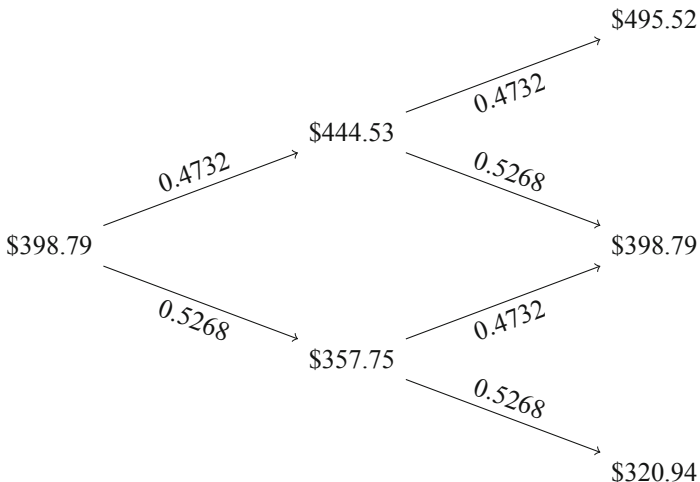
To calculate the value of the call option with a strike price of $ 395, we have to start at the time to maturity of the option. That is, we start at the end of the second period and work our way to the present using backward substitution. Taking the $Vuu = \$ 495.52$ node, the intrinsic value of the call option at that node is $ 100.52 [$= \max(\$ 495.52 - \$ 395.00, 0)$]. Similarly, at the $Vud = Vdu = \$ 398.79$ node, the intrinsic value of the call option at that node is $ 3.79 [$= \max(\$ 398.79 - \$ 395.00, 0)$]. Lastly, at the $Vdd = \$ 320.94$ node, the intrinsic value of the call option at that node is zero because the strike price exceeds the stock price at that node.
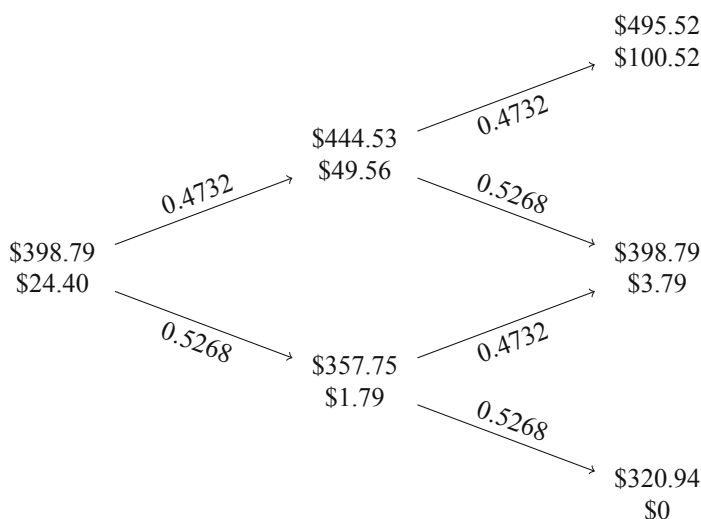
Continuing to work backwards through time, we now calculate the first pe-
riod option values for the up-node and down-node. Recall that the time increment we have is 0.111 years (i.e., half the time from valuation date of December 31, 2013 to the option expiration date on March 22, 2014). Therefore, the call option value at node Vu is equal to $ 49.56 [$= (0.4732 * \$ 100.52 + 0.5268 * \$ 3.79)/(1 + (0.0007 * 0.111))$] and at node Vd is equal to $ 1.79 [$= (0.4732 * \$ 3.79 + 0.5268 * \$ 0)/(1 + (0.0007 * 0.111))$]. We then work backwards to get to the price today for the call option using backward substitu-
tion, which is $ 24.40 [$= (0.4732 * \$ 49.56 + 0.5268 * \$ 1.79)/(1 + (0.0007 * 0.111))$].

9.8.1   The Long Way

Now that we understand the mechanics of the Binomial Model, we can then program this specific calculation.

Step 1: Setup Variables for Characteristics of the Option An option would have a strike price and time to maturity. The option would be based on the underlying

asset's price, volatility, and risk-free rate. The last choice we have to make is to determine how many time steps we would want in our calculation.

```
> S=398.79
> K=395
> TTM=0.2219178
> r=0.0007
> sigma=0.3259855
> n=2
```

Step 2: Calculate Time Increment We divide the `TTM` by the number of time increments `n`.

```
> dt=TTM/n
> dt
[1] 0.1109589
```

Step 3: Calculate the Periodic Discount Factor Since there are dt periods, we have to scale down the risk-free rate, which is in annual terms, into the appropriate periodic risk-free rate.

```
> disc=(1+r*dt)
> disc
[1] 1.000078
```

Step 4: Calculate Up Move and Down Move The up move and down move are calculated based on the volatility of the underlying asset.

```
> u=exp(sigma*sqrt(dt))
> u
[1] 1.114702

> d=1/u
> d
[1] 0.8971005
```

Step 5: Calculate Risk-neutral Probability of an Up Move Using Eq. (9.8), we calculate the risk-neutral probability.

```
> p=((1+r*dt)-d)/(u-d)
> p
[1] 0.4732367
```

Step 6: Generate Values at All Nodes at Maturity The shortcut to this procedure is to calculate an upstate multiplicative factor and downstate multiplicative factor and multiply the two together.

```
> UP<- u^(0:n)
> UP
[1] 1.000000 1.114702 1.242561
>
> DOWN<- d^(n:0)
> DOWN
[1] 0.8047893 0.8971005 1.0000000
>
> terminal<- S*UP*DOWN
> terminal
[1] 320.9419 398.7900 495.5210
```

Step 7: Calculate the Intrinsic Value of the Call Option at Maturity Similar to
our example above, we only do this at the terminal nodes.

```
> terminal.optval<- ifelse(terminal-K<0,0,terminal-K)
> terminal.optval
[1]    0.000   3.790 100.521
```

Step 8: Calculate the Option Value via Backward Substitution The most efficient
way to use this, we need to use two loops. We should go through this step-by-step as
the four lines of code may look pretty daunting. The first line of code is the loop that
we want to start from one time increment prior to maturity (from=n-1) and walk
backwards in time to the present (to=0) one time increment at a time (by=-1). The
second line of code is the loop that calcualtes the value at each node. The third line
of code calculates the value of the option at each node.

```
> for (j in seq(from=n-1,to=0,by=-1))
+      for (i in 0:j)
+      terminal.optval[i+1]=
+          (p*terminal.optval[i+2]+(1-p)*terminal.optval[i+1])/disc
>
> terminal.optval
[1]   24.39776  49.56281 100.52099
```

Step 9: Extract Call Option Value The call option value is the first element of the
list in `terminal.optval`, which is $ 24.40. This equals the value we calculated
previously.

```
> call.optval<- terminal.optval[1]
> call.optval
[1] 24.39776
```

9.8.2   Binomial Model Function

We can also create a function to calculate the call and put option value using a
Binomial Model. Note that we add a variable called `type`, which takes the value of
`call` or `put` so we can calculate either option type with one function. We create a

variable x=1 for a call option and x=-1 for a put option. We then setup a warning
if the option type is not equal to `call` or `put`. This warning will be in the form
of an error message. The rest of the calculations should look very similar to the
calculations above.

```
> EuroCRR<- function(S,K,T,r,sigma,n,type){
+        x=NA
+        if (type=="call") x=1
+        if (type=="put") x=-1
+        if (is.na(x)) stop("Option Type can only be call or put")
+        dt=T/n
+        u=exp(sigma*sqrt(dt))
+        d=1/u
+        p=((1+r*dt)-d)/(u-d)
+        disc<- (1+r*dt)
+        OptVal<- x*(S*u^(0:n)*d^(n:0)-K)
+        OptVal=ifelse(OptVal<0,0,OptVal)
+        for (j in seq(from=n-1,to=0,by=-1))
+                for (i in 0:j)
+                OptVal[i+1]=(p*OptVal[i+2]+(1-p)*OptVal[i+1])/disc
+     value=OptVal[1]
+     results<- rbind(u,d,p,value)
+        results
+ }
```

We now calculate the value of a European call using two time steps. We see that
the value generated by the Binomial Model Function is equal to $ 24.40, which is
identical to the two calculations we performed above.

```
> EuroCRR(398.79,395,0.2219178,0.0007,0.3259855,2,"call")
            [,1]
u       1.1147023
d       0.8971005
p       0.4732367
value 24.3977599
```

The Binomial Model is said to converge to the Black-Scholes-Merton OPM when
the time increments approaches infinity. Using the Binomial Model Function, we
can see the effect of increasing the time increments. We begin by increasing the time
increment from two to 100 and then to 1000. We see that the value at 1000 time
increments is equal to $ 26.258, which is very close to the value we calculate above
using the BSM OPM of $ 26.264.

```
> EuroCRR(398.79,395,0.2219178,0.0007,0.3259855,100,"call")
            [,1]
u     1.0154751
d     0.9847607
p     0.4962115
value 26.3065046
> EuroCRR(398.79,395,0.2219178,0.0007,0.3259855,1000,"call")
            [,1]
u     1.0048680
d     0.9951556
p     0.4988020
value 26.2583744
```

We can do a similar analysis with put options. At 1000 time increments, the value of
the put option is equal to $ 22.407. This is very similar to the value of the put option
under BSM OPM we calculated above of $ 22.412.

```
> EuroCRR(398.79,395,0.2219178,0.0007,0.3259855,2,"put")
            [,1]
u     1.1147023
d     0.8971005
p     0.4732367
value 20.5464068
> EuroCRR(398.79,395,0.2219178,0.0007,0.3259855,100,"put")
            [,1]
u     1.0154751
d     0.9847607
p     0.4962115
value 22.4551491
> EuroCRR(398.79,395,0.2219178,0.0007,0.3259855,1000,"put")
            [,1]
u     1.0048680
d     0.9951556
p     0.4988020
value 22.4070189
```

9.9 Further Reading

Discussion of options and derivatives can get complicated pretty quickly. As such,
depending on your level of training, different texts may be appropriate. Excellent
introductory texts include Hull [3] and Wilmott [8]. A slightly more advanced dis-
cussion can be found in Joshi [4] and Bjork [1]. For the more technically inclined,
an excellent formal treatment of options pricing can be found in Oksendal [7] and
Karatzas and Shreve [5].

References

1. Bjork, T. (2009). *Arbitrage theory in continuous time* (3rd ed.). New York: Oxford University Press.
2. Black, F., & Scholes, M. (1973). Pricing of options and corporate liabilities. *The Journal of Political Economy*, *81,* 637–654.
3. Hull, J. (2011). *Options, futures, and other derivatives* (8th ed.). Prentice Hall.
4. Joshi, M. (2008). *The concepts and practice of mathematical finance* (2nd ed.). United Kingdom: Cambridge University Press.
5. Karatzas, I., & Shreve, S. (1998). *Methods of mathematical finance*. New York: Springer.
6. Merton, R. (1973). *Theory of rational option pricing*. *Bell Journal of Economics and Management Science, 4,* 141–183.
7. Oksendal, B. (2003). *Stochastic differential equations: An introduction with applications* (6th ed.). Germany: Springer.
8. Wilmott, P. (2007). *Paul Wilmott introduces quantitative finance* (2nd ed.). New Jersey: Wiley.

Appendix A
Getting Started with R

A.1 Installing R

To get started with R, we first have to download and install R on our local system. To get R, go to the Comprehensive R Archive Network (CRAN) website (http://cran.r-project.org) and download the latest version. There are versions of R for Windows, OS X, and Linux. The code in this book has been tested to run on the Windows and OS X versions of R. As of the writing of this book, the latest version of R available was version 3.0.2 dated September 25, 2013 and named "Frisbee Sailing."

A.2 The R Working Directory

In this book, the programs and any external CSV files we will use should be placed in the R working directory. To determine which directory on your hard drive is used by R as the working directory, type in the `getwd()` command in the command prompt. The *command prompt* is the line that begins with ">" sign when you start R in what is called the R console.

```
> getwd()
[1] "/Users/cang"
```

Note that the line that does not begin with the > sign is the output of the command. The output tells us that the working directory in my system is `/Users/cang`. We can then create and save all relevant programs and files into the R working directory.

Although we can change the working directory, this is not essential for this book and we can simply use whatever default working directory R outputs.

A.3   R **Console Output**

Every line of output in this book that begins with the command prompt (>) or a plus sign (+) is a line of code. The plus sign (+) signifies that the particular line of code and the preceding line(s) of code is being treated by R as one continuous string of code that for whatever reason was separated into multiple lines. For example, when we subset data, we can type the following:

```
> subset(data,index(data)>="2010-12-31")

   or

> subset(data,
+   index(data)>="2010-12-31")
```

The second type of code makes it easier to read, but it uses several lines instead of the first type of code that puts all the arguments in one line. Both lines will generate the same output, which is to subset `data` to only contain values from 12/31/2010. Experienced programmers may prefer to use the second form of the code. In this book, we will use both forms.

A.4   **R Editor**

When writing code in R, we can directly type in the code in the R console and when we hit **ENTER**, the code we just typed will execute. However, a better way for us to program is to use the R editor. The R editor looks like an ordinary text editor (e.g., Notepad in Windows or TextEdit in OS X). We can call up the R Editor by clicking on the **File-New Script** or **CTRL+N** in R for Windows. In OS X, a new script can be created by clicking on **File-New Document** or **Command+N**.

We would then type in the R editor the text in the output that follows the command prompt (>) or plus sign (+), excluding these symbols. In other words, for the subsetting example above, we would type the following in the R Editor.

```
subset(data,
   index(data)>="2010-12-31")
```

We then highlight the portion of the code we want to execute and type **CTRL+R** in Windows or **Command+ENTER** in OS X. Doing so in the code above will generate the output above in which we break the code into two lines with a plus (+) sign preceding the second line.

Note also that the second line is indented. Indenting or not indenting code does not affect how the program is executed. Indenting only makes it easier to identify which lines belong together. In our example above, the indented second line can easily be identified as being one set of code together with the first line.

A.5 Packages

Packages are like toolboxes that contain a lot of functions that help us simplify our tasks. The base installation of R comes with a set of built-in packages to help us do many things. However, for some of the calculations in this book, we will need to install additional packages. We will use primarily the `quantmod` and `xts` packages.

To load a package, we use the `library` command. For example, to load the `quantmod` package, we type in the following:

```
> library(quantmod)
Loading required package: Defaults
Loading required package: xts
Loading required package: zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric

Loading required package: TTR
Version 0.4-0 included new data defaults. See ?getSymbols.
```

During each R session, the first time we load a package, we may see a series of messages like the above for `quantmod`. We only need to load packages once during each R session, so to make sure the code gets executed smoothly we should always load the `quantmod` and `xts` packages every time we start R.

Reloading a package that has already been previously loaded does not do any harm. However, no messages appear for the subsequent loading of the package. As such, we should not be surprised if a command prompt appears right after, such as what we observe from the output below of a second execution of the `library(quantmod)` code.

```
> library(quantmod)
>
```

If the package has not been installed in our local system, R will generate an error message. For example, if we try to load the package `tseries` and it has not been installed in our local machine, we would observe the following output.

```
> library(tseries)
Error in library(tseries) : there is no package called 'tseries'
```

We can then install the package using the `install.packages` command. Note that we should put the package name inside quotes or else R will generate an error stating it could not find `tseries`.

```
> install.packages(tseries)
Error in install.packages(tseries) : object 'tseries' not found

> install.packages("tseries")
trying URL 'http://cran.wustl.edu/bin/macosx/contrib/3.0/tseries_0.10-32.tgz'
Content type 'application/x-gzip' length 308854 bytes (301 Kb)
opened URL
==================================================
downloaded 301 Kb


The downloaded binary packages are in
        /var/folders/7_/36vllrlx7d99kys673pg9c340000gn/T//RtmpJIEAZS/downloaded_packages
```

Unless we have previously selected a CRAN mirror site from where we download
the package, R will open a window that would require us to select a CRAN mirror
site. We can technically choose any site and the package will install, but a good rule
to follow when selecting a CRAN mirror site is to choose a CRAN mirror that is in
close proximity to our current location. For example, in Chicago, Illinois, one of the
closest CRAN mirrors is in Missouri. Each time we open an R session and have to
use a CRAN mirror, this window will always pop up. As such, we are not tied to the
choice we make as to which CRAN mirror to use.

One nice feature of R is that if you install a package that depends on other packages,
and those other packages are not on our system, R will download those other packages
automatically. In addition, we can use the help command to pull-up the help file for
that package. A new window will open up in which a webpage-styled help document
specific to that package will load.

```
> help(package="tseries")
starting httpd help server ... done
```

A.6   Basic Commands

In this book, we will use the teletype font to denote R-related commands,
functions, or objects. Hopefully, this will aid in separating R-related inputs from
normal text.

R is interactive, so we can directly type in expressions in the command prompt
and see the results just like we would in a calculator. For example, suppose we want
to add 2 and 3 together.

```
> 2+3
[1] 5
```

Alternatively, we can also multiply 2 and 3.

```
> 2*3
[1] 6
```

Using R as a calculator may not be the best use of R. We can create variables by using the assignment operator "←" which is comprised of a less than sign (<) and a minus sign (−). Suppose we let $x = 2$ and $y = 3$ and add and multiply the two variables. Note that to see what the variable holds, we have to type in the variable name in the command prompt. That is, the first line merely generates an empty command prompt and typing x again shows us the output of 2.

```
> x<-2
> x
[1] 2
> y<-3
> y
[1] 3
> x+y
[1] 5
> x*y
[1] 6
```

Although we primarily will use the assignment operator above, we sometimes will also assign values using the equal sign (=). A third alternative way to assign values is to use the assignment operator with the arrow pointing right (→). However, this may be too confusing to use, so we will systematically make assignments only having the arrow pointing to the left or using the equal sign.

```
> 2->x
> x
[1] 2
```

Note that R is case-sensitive in terms of commands, functions, variables, or objects. For example, the variable x above is in lower case. An error would be generated if we typed in an upper case X instead.

```
> X
Error: object 'X' not found
```

The above is clearly not an exhaustive list of issues, but merely the most basic of commands. We will go through the necessary operations, functions, and commands in the text of the book, which is used for specific applications. This will likely aid in recalling how to use these terms in the future.

A.7   The R Workspace

The R workspace contains a list of the objects that are in the R memory. To view the R workspace, we use the ls() command.

```
> ls()
[1] "x" "y"
```

From the codes we typed in this Appendix, only the x and y variables are in the workspace. Within the chapters of the book, we may use certain objects created in earlier sections. For example, we may retrieve stock price data in the first section and use that same stock price data in the fifth section. As such, having objects in the R memory is helpful as it helps avoid re-pulling data.

However, when we move to a different chapter or perform a completely independent analysis, we may want to clear up the memory of any old objects. The reason for this is that we may not want to inadvertently call an object and generate incorrect calculations. As such, to avoid this, we use the rm(list=ls()) command. This will delete all the objects in memory, as the output below shows.

```
> rm(list=ls())
> ls()
character(0)
```

Note that we should be careful when we use the rm(list=ls()) command as this will delete any objects in the workspace. If we were to share this code with others, we should make a conscious choice whether we should include the code with this command or not. The person running the code may inadvertently erase important objects from their workspace if we send the code with the rm(list=ls()) command.

A.8   Vectors

We can create vectors in R, which can contain numbers or text. The vector a contains a vector of numbers, vector b contains a vector of text, and vector c contains a vector of numbers and text.

```
> a<-c(1,2,3,5,6,7)
> a
[1] 1 2 3 5 6 7
> b<-c("Sun","Mon","Tue","Thu","Fri","Sat")
> b
[1] "Sun" "Mon" "Tue" "Thu" "Fri" "Sat"
> c<-c(1,2,3,"Sun","Mon","Tue")
> c
[1] "1"   "2"   "3"   "Sun" "Mon" "Tue"
```

Note that the fully numeric vector a lists out the numbers without quotes, where as any vector that partially has text or characters lists out the values with quotes.

In the above example, we created a vector using the the c(...) operator. Note that many things that need to be strung together in R make use of the c(...) operator.

We can select elements in a vector by using square brackets and identifying the element number. Suppose we want to choose the fourth element in vector a and vector b. We thus put 4 inside square brackets after calling the vector name.

```
> a[4]
[1] 5
> b[4]
[1] "Thu"
```

A.9   Combining Vectors

We can combine vectors by rows or by columns. To combine vectors as columns, we use the `cbind` command.

```
> col.vectors<-cbind(a,b)
> col.vectors
      a    b
[1,] "1" "Sun"
[2,] "2" "Mon"
[3,] "3" "Tue"
[4,] "5" "Thu"
[5,] "6" "Fri"
[6,] "7" "Sat"
```

To combine vectors as rows, we use the `rbind` command.

```
> row.vectors<-rbind(a,b)
> row.vectors
  [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
a "1"   "2"   "3"   "5"   "6"   "7"
b "Sun" "Mon" "Tue" "Thu" "Fri" "Sat"
```

The `col.vectors` object looks like a 6 (row) × 2 (column) matrix. We can add column vector `c` to the left side of this object by applying another `cbind` command and putting the `c` vector as the first argument, which now creates a 6 (row) × 3 (column) object.

```
> leftcol.vectors<-cbind(c,col.vectors)
> leftcol.vectors
      c     a    b
[1,] "1"   "1" "Sun"
[2,] "2"   "2" "Mon"
[3,] "3"   "3" "Tue"
[4,] "Sun" "5" "Thu"
[5,] "Mon" "6" "Fri"
[6,] "Tue" "7" "Sat"
```

Alternatively, we could have added the column vector `c` to the right side of the object by putting the `c` vector as the second object.

```
> rightcol.vectors<-cbind(col.vectors,c)
> rightcol.vectors
      a    b      c
[1,] "1"  "Sun"  "1"
[2,] "2"  "Mon"  "2"
[3,] "3"  "Tue"  "3"
[4,] "5"  "Thu"  "Sun"
[5,] "6"  "Fri"  "Mon"
[6,] "7"  "Sat"  "Tue"
```

The above examples show that the order in which the columns appear depends on
the order that they appear inside the `cbind` command. This adds flexibility in terms
of how we want to show the data in the object. In the main text, we will go through
in the context of applications other ways of combining data.

A.10 Matrices

When we bound the vectors above together, we converted its data structure into a
`matrix`. We show the class of the object using the `class` command.

```
> class(rightcol.vectors)
[1] "matrix"
> dim(rightcol.vectors)
[1] 6 3
```

A `matrix` is a `vector` with dimensions. In our example, `righcol.vectors`
has dimensions of six rows $\times$ three columns and is often described as a 6×3 matrix.
We will deal with matrices when we deal with portfolio calculations as this greatly
simplifies the implementation.

A.11 data.frame

The `data.frame` object is a flexible data structure that we will use when performing
more involved calculations. A `data.frame` object has rows and columns.

   To convert the `rightcol.vectors` matrix above into a `data.frame` object,
we use the `data.frame` command.

```
> df<-data.frame(rightcol.vectors)
> class(df)
[1] "data.frame"
> df
  a   b   c
1 1 Sun   1
2 2 Mon   2
3 3 Tue   3
4 5 Thu Sun
5 6 Fri Mon
6 7 Sat Tue
```

Now that the object is a data.frame, we can pick out certain columns by using the dollar sign ($) followed by the variable name. For example, if we want to pick out the second column of df, we type df$b.

```
> df$b
[1] Sun Mon Tue Thu Fri Sat
Levels: Fri Mon Sat Sun Thu Tue
```

We can also accomplish the same thing by placing square brackets after df and typing 2 after the comma, as in df[,2].

```
> df[,2]
[1] Sun Mon Tue Thu Fri Sat
Levels: Fri Mon Sat Sun Thu Tue
```

In a data.frame, we can also identify rows by their row number. For example, we want to view only the fourth row. We use the square brackets again, but this time putting a number 4 before the comma, such as df[4,].

```
> df[4,]
  a   b   c
4 5 Thu Sun
```

If we want to know the value that is in a specific row/column, we can do that by entering values on both sides of the comma. Suppose we want to know the fourth value in the second column of df, we would type in the command prompt df[4,2].

```
df[4,2].
```

```
> df[4,2]
[1] Thu
Levels: Fri Mon Sat Sun Thu Tue
```

These techniques will come in handy when we have to subset data and extract elements from a data.frame object throughout the text.

Table A.1 Symbols and
description of R date formats
for December 31, 2013

Symbol	Description	Examples
%Y	4-digit year	2013
%y	2-digit year	13
%m	2-digit month	12
%b	Short month name	Dec
%B	Long month name	December
%d	Day of the week (0 to 31)	31
%a	Short weekday	Tue
%A	Long weekday	Tuesday

A.12   Date Formats

R has specific symbols it uses for its date formats. The table above describes these
symbols and provides examples.

Appendix B
Constructing a Hypothetical Portfolio

In this Appendix, we construct a hypothetical portfolio assuming portfolio returns are equal to that of an equal-weighted portfolio of Amazon.com (AMZN), Yahoo (YHOO), and IBM stocks from January 2011 to December 2013 with monthly rebalancing. To implement this, we need to make sure we have data from December 2010 in order to calculate a monthly return for the monthly of January 2011. Note that this is merely a hypothetical portfolio with constructed returns. This is meant as a proxy for our actual portfolio returns used in this book.

To construct the portfolio returns, we calculate the monthly returns for AMZN, YHOO, and IBM using the techniques we discussed in Chap. 2.

```
> # Amazon Data
> 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)),]
           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
> AMZN.monthly<-to.monthly(data.AMZN)
> AMZN.monthly[c(1:3,nrow(AMZN.monthly)),]
         data.AMZN.Open data.AMZN.High data.AMZN.Low data.AMZN.Close
Dec 2010         181.96         182.30        179.51          180.00
Jan 2011         181.37         191.60        166.90          169.64
Feb 2011         170.52         191.40        169.51          173.29
Dec 2013         399.00         405.63        379.50          398.79
         data.AMZN.Volume data.AMZN.Adjusted
Dec 2010          3451900             180.00
Jan 2011        113611300             169.64
Feb 2011         95776400             173.29
Dec 2013         55638100             398.79
Warning message:
timezone of object (UTC) is different than current timezone ().
> AMZN.monthly<-AMZN.monthly[,6]
> AMZN.ret<-Delt(AMZN.monthly$data.AMZN.Adjusted)
> names(AMZN.ret)<-paste("AMZN.ret")
> AMZN.ret[c(1:3,nrow(AMZN.ret)),]
            AMZN.ret
Dec 2010          NA
Jan 2011 -0.05755556
Feb 2011  0.02151615
Dec 2013  0.01313450
Warning message:
timezone of object (UTC) is different than current timezone ().
>
> # 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)<-
+       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
> YHOO.monthly<-to.monthly(data.YHOO)
> YHOO.monthly[c(1:3,nrow(YHOO.monthly)),]
         data.YHOO.Open data.YHOO.High data.YHOO.Low data.YHOO.Close
Dec 2010          16.74          16.76         16.47           16.63
Jan 2011          16.81          17.34         15.41           16.12
Feb 2011          16.33          17.84         16.04           16.40
Dec 2013          37.04          41.05         36.25           40.44
         data.YHOO.Volume data.YHOO.Adjusted
Dec 2010          7754500              16.63
Jan 2011        441126900              16.12
Feb 2011        446785900              16.40
Dec 2013        307732000              40.44
Warning message:
timezone of object (UTC) is different than current timezone ().
> YHOO.monthly<-YHOO.monthly[,6]
> YHOO.ret<-Delt(YHOO.monthly$data.YHOO.Adjusted)
> names(YHOO.ret)<-paste("YHOO.ret")
> YHOO.ret[c(1:3,nrow(YHOO.ret)),]
            YHOO.ret
Dec 2010          NA
Jan 2011 -0.03066747
Feb 2011  0.01736973
Dec 2013  0.09356409
Warning message:
timezone of object (UTC) is different than current timezone ().
>
```

```
> # IBM Data
> 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
> IBM.monthly<-to.monthly(data.IBM)
> IBM.monthly[c(1:3,nrow(IBM.monthly)),]
         data.IBM.Open data.IBM.High data.IBM.Low data.IBM.Close
Dec 2010        146.73        147.07       145.96         146.76
Jan 2011        147.21        164.35       146.64         162.00
Feb 2011        162.11        166.25       159.03         161.88
Dec 2013        179.46        187.79       172.73         187.57
         data.IBM.Volume data.IBM.Adjusted
Dec 2010         2969800            139.23
Jan 2011       118831600            153.69
Feb 2011        89067500            154.18
Dec 2013        97954300            187.57
Warning message:
timezone of object (UTC) is different than current timezone ().
> IBM.monthly<-IBM.monthly[,6]
> IBM.ret<-Delt(IBM.monthly$data.IBM.Adjusted)
> names(IBM.ret)<-paste("IBM.ret")
> IBM.ret[c(1:3,nrow(IBM.ret)),]
             IBM.ret
Dec 2010          NA
Jan 2011 0.103856927
Feb 2011 0.003188236
Dec 2013 0.043911398
Warning message:
timezone of object (UTC) is different than current timezone ().
```

We then combine the data together and calculate the EW portfolio returns. The EW portfolio returns are calculated using the `rowMeans` command. We then clean up the data to only hold the `port.ret` data.

```
> port<-cbind(AMZN.ret,YHOO.ret,IBM.ret)
> port[c(1:3,nrow(port)),]
           AMZN.ret    YHOO.ret     IBM.ret
Dec 2010         NA          NA          NA
Jan 2011 -0.05755556 -0.03066747 0.103856927
Feb 2011  0.02151615  0.01736973 0.003188236
Dec 2013  0.01313450  0.09356409 0.043911398
> port$port.ret<-rowMeans(port)
> port[c(1:3,nrow(port)),]
           AMZN.ret    YHOO.ret     IBM.ret    port.ret
Dec 2010         NA          NA          NA          NA
Jan 2011 -0.05755556 -0.03066747 0.103856927 0.005211301
Feb 2011  0.02151615  0.01736973 0.003188236 0.014024705
Dec 2013  0.01313450  0.09356409 0.043911398 0.050203327
> port<-port[-1,4]
> port[c(1:3,nrow(port)),]
            port.ret
Jan 2011 0.005211301
Feb 2011 0.014024705
Mar 2011 0.021291224
Dec 2013 0.050203327
```

We then save `port` into a CSV file labeled **Hypothetical Portfolio (monthly).csv**.

```
> csv.port<-cbind(data.frame(index(port)),data.frame(port))
> names(csv.port)[1]<-paste("date")
> csv.port[c(1:3,nrow(csv.port)),]
             date    port.ret
Jan 2011 Jan 2011 0.005211301
Feb 2011 Feb 2011 0.014024705
Mar 2011 Mar 2011 0.021291224
Dec 2013 Dec 2013 0.050203327
> rownames(csv.port)<-seq(1,nrow(csv.port),by=1)
> csv.port[c(1:3,nrow(csv.port)),]
       date    port.ret
1  Jan 2011 0.005211301
2  Feb 2011 0.014024705
3  Mar 2011 0.021291224
36 Dec 2013 0.050203327
> write.csv(csv.port,"Hypothetical Portfolio (Monthly).csv")
```

Index