PREDICTIVE ANALYTICS:

Microsoft® Excel

que®

Conrad Carlberg

Predictive Analytics: Microsoft® Excel

Conrad Carlberg

QUE®

800 East 96th Street,
Indianapolis, Indiana 46240
USA

Contents at a Glance

Predictive Analytics: Microsoft® Excel

ISBN-13: 978-0-7897-4941-3
ISBN-10: 0-7897-4941-6

Library of Congress Cataloging-in-Publication data is on file.

Printed in the United States of America

Second Printing: March 2013

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Microsoft is a registered trademark of Microsoft Corporation.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Que Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact

International Sales
international@pearsoned.com

Editor-in-Chief
Greg Wiegand

Acquisitions Editor
Loretta Yates

Development Editor
Charlotte Kughen

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Water Crest Publishing

Indexer
Tim Wright

Proofreader
Debbie Williams

Technical Editor
Bob Umlas

Publishing Coordinator
Cindy Teeters

Book Designer
Anne Jones

Compositor
Nonie Ratcliff

Table of Contents

About the Author

Counting conservatively, this is Conrad Carlberg's eleventh book about quantitative analysis using Microsoft Excel, which he still regards with a mix of awe and exasperation. A look back at the "About the Author" paragraph in Carlberg's first book, published in 1995, shows that the only word that remains accurate is "He." Scary.

Dedication

For Sweet Sammy and Crazy Eddie. Welcome to the club, guys.

Acknowledgments

Once again I thank Loretta Yates of Que for backing her judgment. Charlotte Kughen for her work on guiding this book through development, and Sarah Kearns for her skillful copy edit. Bob Umlas, of course, a.k.a. The Excel Trickster, for his technical edit, which kept me from veering too far off course. And Que in general, for not being Wiley.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an editor-in-chief for Que Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@quepublishing.com

Mail: Greg Wiegand
Editor-in-Chief
Que Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at quepublishing.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

A few years ago, a new word started to show up on my personal reading lists: *analytics*. It threw me for a while because I couldn't quite figure out what it really meant.

In some contexts, it seemed to mean the sort of numeric analysis that for years my compatriots and I had referred to as *stats* or *quants*. Ours is a living language and neologisms are often welcome. McJob. Tebowing. Yadda yadda yadda.

Welcome or not, *analytics* has elbowed its way into our jargon. It does seem to connote quantitative analysis, including both descriptive and inferential statistics, with the implication that what is being analyzed is likely to be web traffic: hits, conversions, bounce rates, click paths, and so on. (That implication seems due to Google's Analytics software, which collects statistics on website traffic.)

Furthermore, there are at least two broad, identifiable branches to analytics: *decision* and *predictive*:

- *Decision analytics* has to do with classifying (mainly) people into segments of interest to the analyst. This branch of analytics depends heavily on multivariate statistical analyses, such as cluster analysis and multidimensional scaling. Decision analytics also uses a method called *logistic regression* to deal with the special problems created by dependent variables that are binary or nominal, such as buys versus doesn't buy and survives versus doesn't survive.
- *Predictive analytics* deals with forecasting, and often employs techniques that have been used for decades. Exponential smoothing (also termed exponentially weighted moving averages or EMWA) is one such technique, as is autoregression. Box-Jenkins analysis dates to

the middle of the twentieth century and comprises the moving average and regression approaches to forecasting.

Of course, these two broad branches aren't mutually exclusive. There's not a clear dividing line between situations in which you would use one and not the other, although that's often the case. But you can certainly find yourself asking questions such as these:

- I've classified my current database of prospects into likely buyers and likely non-buyers, according to demographics such as age, income, ZIP Code, and education level. Can I create a credible quarterly forecast of purchase volume if I apply the same classification criteria to a data set consisting of *past* prospects?
- I've extracted two principal components from a set of variables that measure the weekly performance of several product lines over the past two years. How do I forecast the performance of the products for the next quarter using the principal components as the outcome measures?

So, there can be overlap between decision analytics and predictive analytics. But not always—sometimes all you want to do is forecast, say, product revenue without first doing any classification or multivariate analysis. But at times you believe there's a need to forecast the behavior of segments or of components that aren't directly measurable. It's in that sort of situation that the two broad branches, decision and predictive analytics, nourish one another.

You, Analytics, and Excel

Can you do analytics—either kind—using Excel? Sure. Excel has a large array of tools that bear directly on analytics, including various mathematical and statistical functions that calculate logarithms, regression statistics, matrix multiplication and inversion, and many of the other tools needed for different kinds of analytics.

But not all the tools are native to Excel. For example, some situations call for you to use logistic regression: a technique that can work much better than ordinary least-squares regression when you have an outcome variable that takes on a very limited range of values, perhaps only two. Odds ratios are the workhorses of logistic regression, but although Excel offers a generous supply of least-squares functions, it doesn't offer a maximum likelihood odds ratio function.

Nevertheless, the tools are there. Using native Excel worksheet functions and formulas, you can build the basic model needed to do logistic regression. And if you apply Excel's Solver add-in to that model, you can turn out logistic regression analyses that match anything you can get from SAS, R, or any similar application designed specifically for statistical analysis. Furthermore, when you build the analysis yourself you can arrange for all the results that you might find of interest. There's no need to rely on someone else's sense of what matters. Most important, you maintain control over what's going on in the analysis.

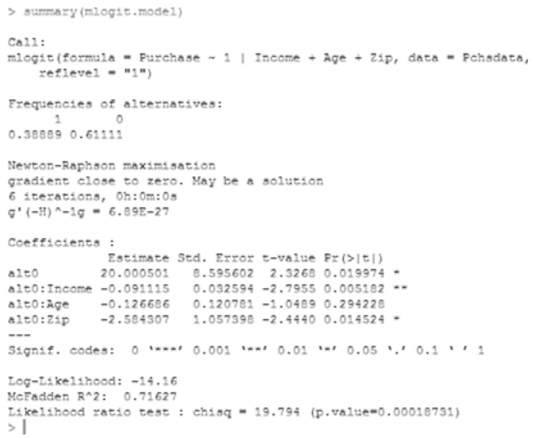
Similarly, if you're trying to make sense of the relationships between the individual variables in a 20-by-20 correlation matrix, principal components analysis is a good place to start and often represents the first step in more complex analyses, such as factor analysis with different kinds of axis rotation. Simple matrix algebra makes it a snap to get factor loadings and factor coefficients, and Excel has native worksheet functions that transpose, multiply, and invert matrices—and get their determinants with a simple formula.

This branch of analytics is often called *data reduction*, and it makes it feasible to forecast from an undifferentiated mass of individual variables. You do need some specialized software in the form of an Excel add-in to extract the components in the first place, and that software is supplied via download with this book.

Now, if you're already an analytics maven, you might have little need for a book like this one. You probably have access to specialized software that returns the results of logistic regression, that detects and accounts for seasonality in a time series, that determines how many principal components to retain by testing residual matrices, and so on.

But that specialized software sometimes tends to be singularly uninformative regarding the analysis. Figure I.1 shows a typical example.

Figure I.1
A logistic regression analysis prepared using the freeware statistical analysis application R.



```
> summary(mlogit.model)

Call:
mlogit(formula = Purchase ~ 1 | Income + Age + Zip, data = Pchsdata,
    reflevel = "1")

Frequencies of alternatives:
      1       0
0.38889 0.61111

Newton-Raphson maximisation
gradient close to zero. May be a solution
6 iterations, 0h:0m:0s
g'(-H)^-1g = 6.89E-27

Coefficients :
             Estimate Std. Error t-value Pr(>|t|)
alt0        20.000501   8.595602  2.3268 0.019974 *
alt0:Income -0.091115   0.032594 -2.7955 0.005182 **
alt0:Age    -0.126686   0.120781 -1.0489 0.294228
alt0:Zip    -2.584307   1.057398 -2.4440 0.014524 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Log-Likelihood: -14.16
McFadden R^2:  0.71627
Likelihood ratio test : chisq = 19.794 (p.value=0.00018731)
> |
```

R is a fairly popular statistics application that, like all applications, has its fair share of detractors. (I use it, but for confirmation and benchmarking purposes only, and that's how I use it in a couple of chapters in this book.) Its documentation is terse and dense. It can take much more work than it should to determine what a particular number in the output represents and how it was calculated. If you're an experienced R user, you've probably tracked down all that information and feel perfectly comfortable with it.

If you're like many of the rest of us, you want to see intermediate results. You want to see the individual odds ratios that come together in an overall likelihood ratio. You want to know if the Newton-Raphson algorithm used a "multistart" option. In short, you want to know more about the analysis than R (or SAS or Stata or SPSS, for that matter) provides.

Excel as a Platform

And that's one major reason I wrote this book. For years I have believed that so-called "advanced" statistical analysis does not require a greater degree of intelligence or imagination for understanding. That understanding tends to require more work, though, because there are more steps involved in getting from the raw data to the end product.

That makes Excel an ideal platform for working your way through a problem in analytics. Excel does not offer a tool that automatically determines the best method to forecast from a given baseline of data and then applies that method on your behalf. It does give you the tools to make that determination yourself and use its results to build your own forecasts.

On your way to the forecast, you can view the results of the intermediate steps. And more often than not, you can alter the inputs to see the effects of your edits: Is the outcome robust with respect to minor changes in the inputs? Or can a single small change make a major difference in the model that you're building? That's the sort of insight that you can create very easily in Excel but that comes only with much more effort with an application that is focused primarily on statistical analysis.

I would argue that if you're responsible for making a forecast, if you're directly involved in a predictive analytics project, you should also be directly involved at each step of the process. Because Excel gives you the tools but in general not the end result, it's an excellent way to familiarize yourself not only with how an analytic method works generally but also how it works with a particular data set.

What's in This Book

Because the term "analytics" is so wide-ranging, a single book on the topic necessarily has to do some picking and choosing. I wanted to include material that would enable you to acquire data from websites that engage in consumer commerce. But if you're going to deploy Google Analytics, or its more costly derivative Urchin, on Amazon.com, then you have to own Amazon.com.

But there are ways to use Excel and its data acquisition capabilities to get your hands on data from sites you don't own. I've been doing so for years to track the sales of my own books on sites such as Amazon. I have tens of thousands of data points to use as a predictive baseline and much more often than not I can forecast with great accuracy the number of copies of a given book that will be sold next month. I start this book showing you the way I use Excel to gather this information for me 24 × 7.

It seemed to me that the most valuable tools in the analytics arsenal are logistic regression, data reduction techniques, and Box-Jenkins forecasting. Logistic regression underlies everything from studies of predictive survival curves used by oncologists to the marketing programs designed to sell distressed resort properties. The singular aspect of that sort of work is that the outcome variable has two, or at most a few, nominal values. In this book, I discuss both the binomial analysis traditionally employed to assess that sort of data and the benefits—and drawbacks—of using logistic regression instead. You also see how to perform a logistic regression directly on an Excel worksheet, assisted by Excel's Solver (which *does* tell you whether or not you're using a multistart option).

As introduction to the more involved techniques of factor analysis, I discuss the rationale and methods for principal components analysis. Again, you can manage the analysis directly on the Excel worksheet, but you're assisted by VBA code that takes care of the initial extraction of the components from the raw data or, if you prefer, from a correlation matrix.

And this book addresses the techniques of forecasting in several chapters. Regression and autoregression get their own treatments, as do moving averages and the extension of that technique to exponential smoothing. Finally, I introduce ARIMA, which brings together autoregression and moving averages under one umbrella. The most exacting part of ARIMA analysis, the derivation of the autocorrelation and the partial autocorrelation coefficients from a baseline of data, is handled for you in open source VBA code that accompanies this book—so you can see for yourself exactly how it's done.

This page intentionally left blank

Building a Collector

1

IN THIS CHAPTER:

The word *analytics* connotes, among other notions, the idea that the raw data that gets analyzed includes quantitative measures of online browsing behavior. Data collection instruments such as Omniture and Google Analytics are useful in part because they track a variety of behavior—hits, views, downloads, buys, and so on—along with information about the source of the visit. However, there are times that you want to measure product performance but can't access the traffic data.

Suppose you supply a product that another company or companies resell on their own websites. Although those companies might share their web traffic information with you, it's more likely that they regard it as proprietary. In that case, if you want to analyze end users' purchasing behavior, you might be limited to any data that the resellers' websites make generally available.

That's the position that I'm in when it comes to sales of my books by web resellers such as Amazon. Although I hear rumors from time to time that Amazon shares data about actual sales with book authors, suppliers of music, and manufacturers of tangible products, it hasn't reached me in any particularly useful form. So I have to roll my own.

Fortunately, one of the sales and marketing devices that Amazon employs is product rankings. As to books, there are a couple of different rankings: overall sales, and sales within categories such as Books > Computers & Technology > Microsoft > Applications > Excel. Although as an author I hope that my books achieve a high sales ranking in a category—that gives them greater visibility—I really hope that they achieve a nice, high overall sales ranking, which I believe to bear a closer relationship to actual sales.

So what I want to do is create a means of accessing information about book ratings (including titles that compete with mine), save the data in a form that I can use for analysis, and make inferences about the results of the analysis.

In this chapter, I show you how to do just that: get analytics without the active cooperation of the website. The techniques don't require that you be interested in book sales. They can be used for anything from product sales to stock prices to yards gained per pass attempt.

Planning an Approach

Before you undertake something like the project I'm about to describe, there are a few issues you should keep in mind. If you can't think of a satisfactory way to deal with them, you might want to consider taking a completely different approach.

A Meaningful Variable

Most important is the availability of one or more variables on a website that bear on what you're interested in, even if only indirectly.

For example, Amazon does not publish on a publically accessible web page how many copies of a book it has sold, whether actual, physical books or downloaded electronic copies. But as I mentioned in the prior section, Amazon does publish sales rankings. For the project described here, I decided that I could live with the sales rankings as an indicator of sales figures.

I also learned that although Amazon usually updates the sales ranking every hour, sometimes the updates don't take place. Sometimes they're a little late. Sometimes several hours pass without any update occurring. And sometimes the rankings don't entirely make sense; particularly in the wee hours, a ranking can drift from, say, 20,000 at 4:00 a.m. to 19,995 at 5:00 a.m. to 19,990 at 6:00 a.m. and so on. That kind of movement can't reflect sales, and the deltas are much smaller and more regular than at other times of day. But I found that most of the time the updates take place hourly—give or take a couple of minutes—and correspond either to no sales (the rankings get larger) or to a presumed sale (the rankings get smaller, often by a generous amount).

Identifying Sales

I decided that I could live with changes in rankings as a stand-in for actual sales. I also decided that I could make an assumption about an increase in ranking, such as from 25,000 to 20,000. That's a big enough jump that I can assume a sale took place. I can't tell for sure how many units were sold. But these books don't sell like a Stieg Larsson novel. Amazon sells four or five copies of one of my books each day. So when I see an improvement in ranking of a few thousand ranks, it almost certainly indicates the sale of a single unit.

Given that information (or, maybe more accurately, educated guesses), it's possible to get a handle on what you need to include in a workbook to access, analyze, and synthesize the data sensibly.

Planning the Workbook Structure

Your workbook needs three types of worksheets: one type to collect the data from your web queries, one type to bring the query data together in a single place, and one type to run whatever analyses you decide are needed. (You also need a VBA module to hold your code, but that is covered in a later section of this chapter.)

Those three types of worksheet are discussed in the next three sections.

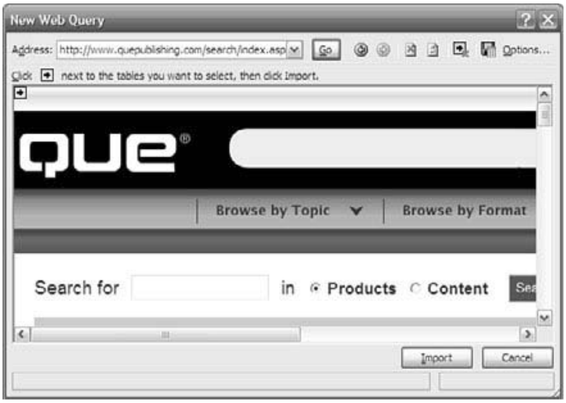
Query Sheets

If you haven't used Excel to retrieve data from the Web, you might be surprised at how easy it is. I'm not speaking here of using your browser to get to a web page, selecting and copying some or part of its contents, and then pasting back into the workbook. I'm speaking of queries that can execute automatically and on a predetermined schedule, thus enabling you to walk away and let the computer gather data without you micromanaging it.

Suppose that you want to retrieve data from Amazon about a book entitled *Statistical Analysis with Excel*. You open a new workbook and rename an unused worksheet to something such as "Stats."

Next, start your browser if necessary and navigate to Amazon's page for that book. When you're there, copy the page's full address from the browser's address box—drag across the address to highlight it and either press Ctrl + C or use the browser's Edit menu to choose the Copy command.

Switch back to Excel and click the Ribbon's Data tab. Click the From Web button in the Get External Data group. The New Web Query window displays as shown in Figure 1.1.

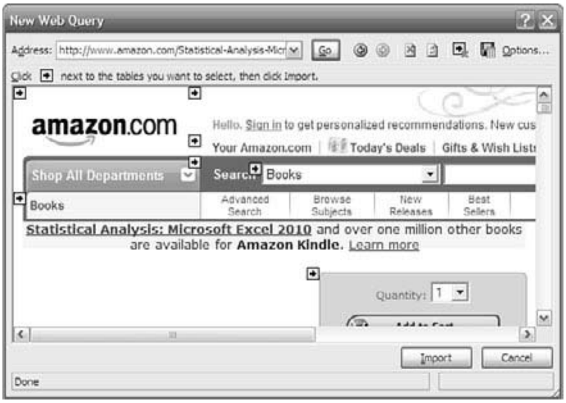


Address: http://www.quepublishing.com/search/index.asp

Click ➡ next to the tables you want to select, then click Import.

Figure 1.1
What you see at first depends on your browser's home page.

Drag through the address that appears in the Address box and press Ctrl + V to paste the address you copied. When you click the Go button, the query window opens the Amazon page (see Figure 1.2).



New Web Query

Address: http://www.amazon.com/Statistical-Analysis-Micr

Click ➔ next to the tables you want to select, then click Import.

Hello. Sign in to get personalized recommendations. New cus

Statistical Analysis: Microsoft Excel 2010 and over one million other books are available for Amazon Kindle. Learn more

Quantity: 1

Done

Figure 1.2
Web pages normally consist of several *tables*, rectangular areas that divide the page into different segments.

Typically you see one or more square icons (a black arrow on a yellow background) that identify the locations of different parts of the web page, called *tables*. You can select them by clicking them. When you click one of these icons to select it, the icon turns green to indicate that you want to download the data in that table.

When you position your mouse pointer over a yellow icon, a heavy border appears around the table that's associated with the icon. This helps you select one or more tables to download.

Nevertheless, I recommend that you select the entire page by clicking the icon in the upper-left corner of the window. If you select only a table or tables that contain the data you're interested in, it could easily happen that a day, week, or month from now the page might be changed, so that the data you want appears in a different table. Then your query will miss the data.

But if you select the entire web page instead of just a table or tables, the page's owner would have to remove the data you're interested in completely for you to miss it, and in that case it wouldn't matter how much of the page you selected to download.

The individual table icons are useful mainly when you want to do a one-time-only download from a web page. Then you might want to avoid downloading a lot of extraneous stuff that would just make it harder to find the data you're after. In the type of case I'm describing here, though, you'll let Excel do the finding for you.

Furthermore, you don't save much time or bandwidth by selecting just a subset of the web page. In most cases you're picking up a few thousand bytes of text at most in an entire page.

Speed of Execution

Nevertheless, you should be aware of a speed versus version tradeoff. I have learned that using Excel 2007 and 2010, web queries can take significantly more time to complete than in earlier versions of Excel. Among the changes made to Excel 2007 was the addition of much more thorough checks of the data returned from web pages for malicious content.

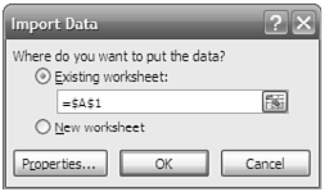
I've found that when using Excel 2002, it takes about 30 seconds to execute eight web queries in the way I'm describing here. Using Excel 2010, it takes nearly three times as long.

The basic intent of the project I'm describing here is to automatically and regularly update your downloaded data, so it probably seems unimportant to worry about whether the process takes half a minute or a minute and a half. Occasionally, though, for one reason or another I want to get an immediate update of the information and so I force the process to run manually. On those occasions I'd rather not wait.

Perhaps I shouldn't, but I trust the results of my Amazon queries to be free of malicious content, so I run my app on the more quick-footed Excel 2002. It's safer, though, to give yourself the added protections afforded by Excel 2007 or 2010, and if you can stand the extra minute or so of query execution time then by all means you should use the slower, safer way.

Bringing the Data Back

After you have clicked a yellow icon to turn it green (using, I hope, the one in the upper-left corner of the New Web Query window so that you get the entire page), click the Import button at the bottom of the New Web Query window. After a few seconds, the Import Data window appears as shown in Figure 1.3.



Import Data

Where do you want to put the data?

● Existing worksheet:

=$A$1

○ New worksheet

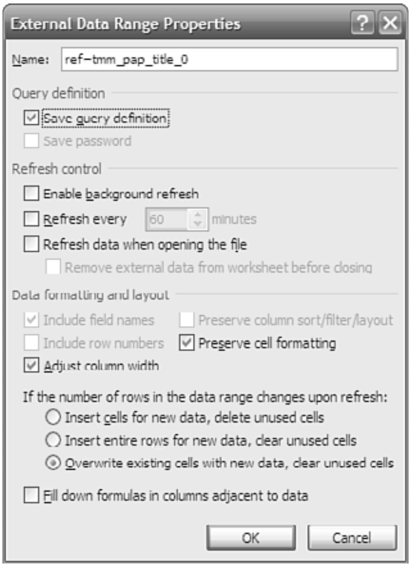
Figure 1.3
You can import immediately, but it's a good idea to check the property settings first.

Accept the default destination of cell A1 and click OK. (The reason to use cell A1 becomes apparent when it's time to refresh the query using VBA, later in this chapter.) There are some useful properties to set, so I recommend that you click the Properties button before you complete the import. The Properties window that displays is shown in Figure 1.4.

Be sure that the Save Query Definition box is checked. That way you can repeatedly run the query without having to define it all over again.

The Enable Background Refresh checkbox requires a little explanation. If it is filled, any VBA procedure that is running continues running as the query executes, and other direct actions by the user can proceed normally. Sometimes that can cause problems if a procedure depends on the results of the query: If the procedure expects to find data that isn't available yet, you might get a run-time error or a meaningless result. Therefore, I usually clear the Enable Background Refresh checkbox.

Figure 1.4
By default the Save Query Definition checkbox is filled, but you should verify the setting when you create the query.



External Data Range Properties

Name: ref-tmm_pap_title_0

Query definition

☑ Save query definition
□ Save password

Refresh control

□ Enable background refresh
□ Refresh every 60 minutes
□ Refresh data when opening the file
□ Remove external data from worksheet before closing

Data formatting and layout

☑ Include field names
□ Preserve column sort/filter/layout
□ Include row numbers
☑ Preserve cell formatting
☑ Adjust column width

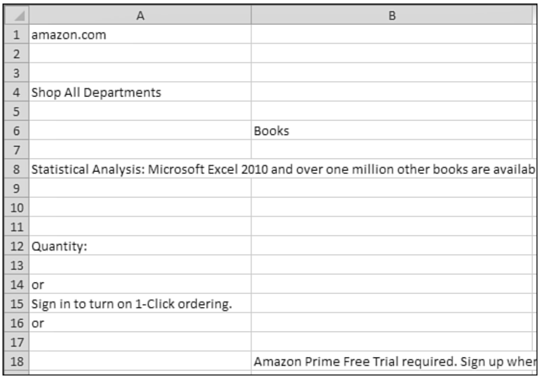
If the number of rows in the data range changes upon refresh:
○ Insert cells for new data, delete unused cells
○ Insert entire rows for new data, clear unused cells
● Overwrite existing cells with new data, clear unused cells

□ Fill down formulas in columns adjacent to data

The various options in the Properties dialog box listed under Data Formatting and Layout are subtly different but can be important. I spend three pages detailing the differences in another Que book, *Managing Data with Excel*, and I don't propose to do it again here. For present purposes, you might just as well accept the default values.

Click OK when you have made any changes you want and then click OK in the Import Data window. Excel completes the query, usually within a few seconds, and writes the results to the worksheet (see Figure 1.5).

Figure 1.5
When the query has finished executing, you wind up with a haystack of text and numbers. The next step is to find the needle.

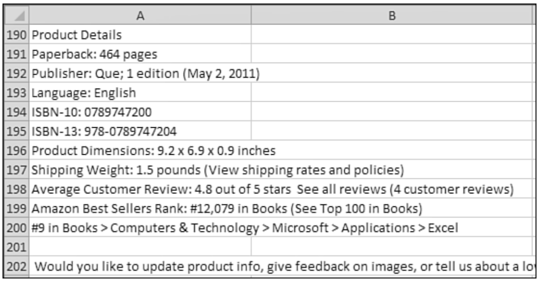


	A	B
1	amazon.com	
2		
3		
4	Shop All Departments	
5		
6		Books
7		
8	Statistical Analysis: Microsoft Excel 2010 and over one million other books are availab	
9		
10		
11		
12	Quantity:	
13		
14	or	
15	Sign in to turn on 1-Click ordering.	
16	or	
17		
18		Amazon Prime Free Trial required. Sign up wher

Finding the Data

After the data has been retrieved from the web page, the next task is to locate the piece or pieces of information you're looking for. I want to stress, though, that you need do this once only for each product you're tracking—and quite possibly just once for all the products. It depends on how the web page administrator is managing the data.

What you need to look for is a string of text that's normally a constant: one that doesn't change from hour to hour or day to day. Figure 1.6 shows a portion of the results of a query.



	A	B
190	Product Details	
191	Paperback: 464 pages	
192	Publisher: Que; 1 edition (May 2, 2011)	
193	Language: English	
194	ISBN-10: 0789747200	
195	ISBN-13: 978-0789747204	
196	Product Dimensions: 9.2 x 6.9 x 0.9 inches	
197	Shipping Weight: 1.5 pounds (View shipping rates and policies)	
198	Average Customer Review: 4.8 out of 5 stars See all reviews (4 customer reviews)	
199	Amazon Best Sellers Rank: #12,079 in Books (See Top 100 in Books)	
200	#9 in Books > Computers & Technology > Microsoft > Applications > Excel	
201		
202	Would you like to update product info, give feedback on images, or tell us about a lo	

Figure 1.6
In this case, the string `Sellers Rank` in cell A199 uniquely locates the product ranking.

If you use Excel's Find feature to scan a worksheet for the string `Sellers Rank`, you can locate the worksheet cell that also contains the ranking for the product. With just a little more work, which you can easily automate and which I describe in the next section about VBA code, you can isolate the actual ranking from the surrounding text; it's that ranking that you're after.

Why not just note the cell address where the ranking is found after the query is finished? That would work fine if you could depend on the web page's layout remaining static. But the website administrator has only to add an extra line, or remove one, above the data's current location, and that will throw off the location of the cell with the data you're after. No, you have to look for it each time, and the Find operation occurs very fast anyway.

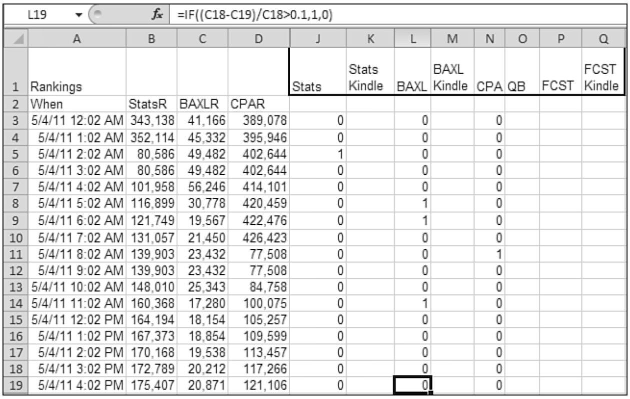
Summary Sheets

After you've acquired the data from a web page and isolated the figure you're looking for, you need a place to put that figure plus other relevant information such as date and time. That place is normally a separate worksheet. You normally expect to be querying the same web page repeatedly, as hours and days elapse. Therefore, you'll want to store information that you've already retrieved somewhere that won't get overwritten the next time the query runs.

So, establish an unused worksheet and name it something appropriate such as *Summary* or *Synthesis* or *All Products*. There are a few structural rules covered in the next section that you'll find helpful to follow. But you can include some other useful analyses on the summary sheet, as long as they don't interfere with the basic structure.

Structuring the Summary Sheet

Figure 1.7 shows the structures that I put on my summary sheet.



L19 =IF((C18-C19)/C18>0.1,1,0)

	A	B	C	D	J	K	L	M	N	O	P	Q
1	Rankings				Stats	Stats Kindle	BAXL	BAXL Kindle	CPA	QB	FCST	FCST Kindle
2	When	StatsR	BAXLR	CPAR								
3	5/4/11 12:02 AM	343,138	41,166	389,078	0		0		0			
4	5/4/11 1:02 AM	352,114	45,332	395,946	0		0		0			
5	5/4/11 2:02 AM	80,586	49,482	402,644	1		0		0			
6	5/4/11 3:02 AM	80,586	49,482	402,644	0		0		0			
7	5/4/11 4:02 AM	101,958	56,246	414,101	0		0		0			
8	5/4/11 5:02 AM	116,899	30,778	420,459	0		1		0			
9	5/4/11 6:02 AM	121,749	19,567	422,476	0		1		0			
10	5/4/11 7:02 AM	131,057	21,450	426,423	0		0		0			
11	5/4/11 8:02 AM	139,903	23,432	77,508	0		0		1			
12	5/4/11 9:02 AM	139,903	23,432	77,508	0		0		0			
13	5/4/11 10:02 AM	148,010	25,343	84,758	0		0		0			
14	5/4/11 11:02 AM	160,368	17,280	100,075	0		1		0			
15	5/4/11 12:02 PM	164,194	18,154	105,257	0		0		0			
16	5/4/11 1:02 PM	167,373	18,854	109,599	0		0		0			
17	5/4/11 2:02 PM	170,168	19,538	113,457	0		0		0			
18	5/4/11 3:02 PM	172,789	20,212	117,266	0		0		0			
19	5/4/11 4:02 PM	175,407	20,871	121,106	0		0		0			

Figure 1.7
You can put snapshot analyses supported by worksheet functions on the summary sheet.

In Figure 1.7, the first few columns are reserved for the rankings that I have obtained via web queries from the appropriate Amazon pages. I also store the date and time the queries finished executing in column A. That time data provides my basis for longitudinal summaries: a baseline for the forecasting analyses that I discuss in Chapters 3, 4, 5, and 9.

It's at this point that you have a decision to make. It's nice to be able to retrieve data about sales rankings for products such as books. If you've written a good one, it's gratifying to see the rankings drop as time goes by. (Remember, high rankings are better: A rank of 1 is a best seller.) But you likely never got paid a royalty or a commission, or had to fill a re-order, strictly on the basis of a sales ranking. It's the sales themselves that you're ultimately seeking: Granted that intermediate objectives such as clicks and conversions and rankings are important indicators, they don't directly represent revenue.

Identifying Sales

So how do you translate sales rankings into a count of sales? I started by tracking sales rankings on Amazon for about a week and noticed some points of interest.

Telling a Sale from No Sale A jump from a lower ranking to a higher ranking probably means the sale of at least one item. If the item has no sales during a given period, its ranking declines as other items do sell and move up.

Ranking Sales How do you rank sales? You can't do it strictly on the number sold. A book, for example, might have sold 200 copies over a two-year period. Another book might have sold 100 copies since it was published last week. The second book is clearly performing better than the first, so you have to combine elapsed time somehow with number sold. Amazon doesn't say, but my guess would be that the rankings are based in part on the ratio of sales to days elapsed since publication—in other words, sales per day.

Improved Rankings Without Sales There are periods when an item's ranking improves very gradually over a period of hours. There's no reason to believe that an improvement from a ranking of, say, 20,000 to 19,999 indicates a sale. More likely it is a result of another day passing and the rankings recalculating accordingly. That means that before you conclude a sale took place, you need a minimum criterion.

Deciding on a Criterion The criterion should be a rate, not a constant number. If a book jumps from a ranking of 200,101 to 200,001, that 100-place increase is considerably different from a book that jumps from a ranking of 101 to 1. I decided to conclude that a sale had taken place if an increase in rankings equaled or exceeded ten percent of the prior ranking. So, if a book ranked 15,000 at 3:00 p.m. and 13,000 at 4:00 p.m.:

(15000 – 13000)/15000 = 0.13 or 13%

I conclude that a sale took place.

Structuring the Formula Suppose that I have two rankings for a given product, one taken at 3:00 p.m. in cell C18 and one taken at 4:00 p.m. in cell C19. If I want to test whether a sale took place between 3:00 p.m. and 4:00 p.m., I can enter this formula in, say, L19:

```
=IF((C18-C19)/C18>0.1,1,0)
```

The formula returns a 1 if the difference between the two rankings is positive (for example, an increase from a ranking of 1,000 to 900 is positive) and exceeds 10% of the earlier ranking. The formula returns a zero otherwise. After the formula is established on the worksheet, I use the same VBA code that re-executes the queries to copy the formula to the next available row.

Snapshot Formulas

I also like to watch two other statistics that don’t depend on an ordered baseline of data the way that sales estimates do. These are the total sales per book and the minimum (that is, the highest) sales ranking attained by each book since I started tracking the rankings.

I use Excel’s MIN() and SUM() functions to get those analyses. I put them at the top of the columns so that they won’t interfere with the results that come back from the web pages as the queries execute over time.

Figure 1.8 shows what those analyses look like.

So, for example, cell J2 might contain this formula:

```
=SUM(J5:J1000000)
```

It sums the values from the fifth to the millionth row in column J, which contains a 1 for every assumed sale, and a 0 otherwise. The result tells me the number of copies of this book that I assume have been sold by Amazon, judging by the changes in sales rankings.

To get the minimum, best sales ranking for the same book, I use this formula in cell T2:

```
=MIN(B4:B1000000)
```

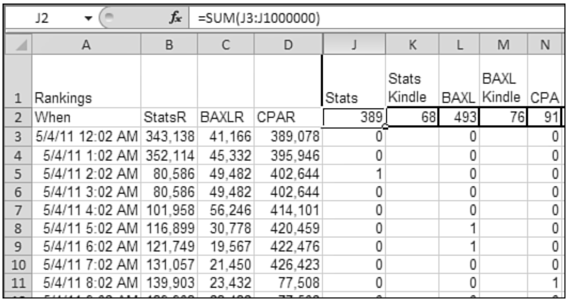


Figure 1.8
You can put snapshot analyses supported by worksheet functions on the summary sheet.

J2 =SUM(J3:J1000000)

	A	B	C	D	J	K	L	M	N
1	Rankings				Stats	Stats Kindle	BAXL	BAXL Kindle	CPA
2	When	StatsR	BAXLR	CPAR	389	68	493	76	91
3	5/4/11 12:02 AM	343,138	41,166	389,078	0		0		0
4	5/4/11 1:02 AM	352,114	45,332	395,946	0		0		0
5	5/4/11 2:02 AM	80,586	49,482	402,644	1		0		0
6	5/4/11 3:02 AM	80,586	49,482	402,644	0		0		0
7	5/4/11 4:02 AM	101,958	56,246	414,101	0		0		0
8	5/4/11 5:02 AM	116,899	30,778	420,459	0		1		0
9	5/4/11 6:02 AM	121,749	19,567	422,476	0		1		0
10	5/4/11 7:02 AM	131,057	21,450	426,423	0		0		0
11	5/4/11 8:02 AM	139,903	23,432	77,508	0		0		1

The formula returns the smallest numeric value for the fourth to the millionth row in column B.

Notice that the range address in these formulas uses a constant, 1,000,000. There are more elegant ways of making sure that you capture all relevant cells (such as dynamic range names and tables), but this one is simple, pragmatic, and doesn't slow down processing.

More Complicated Breakdowns

Figure 1.9 shows a table I use for a quick monthly tabulation of sales of certain books. I get similar tables for daily breakdowns, but they are pivot tables and can get a little cumbersome when you have as many as a couple of hundred days to summarize. The table in Figure 1.9 is driven by worksheet functions and is a quick monthly overview instead of a more detailed daily analysis.

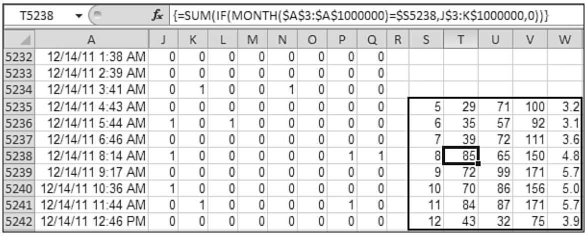


Figure 1.9
This table counts sales per month and projects sales for a full month.

T5238 {=SUM(IF(MONTH($A$3:$A$1000000)=$S5238,J$3:K$1000000,0))}

	A	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
5232	12/14/11 1:38 AM	0	0	0	0	0	0	0	0						
5233	12/14/11 2:39 AM	0	0	0	0	0	0	0	0						
5234	12/14/11 3:41 AM	0	1	0	0	1	0	0	0						
5235	12/14/11 4:43 AM	0	0	0	0	0	0	0	0		5	29	71	100	3.2
5236	12/14/11 5:44 AM	1	0	1	0	0	0	0	0		6	35	57	92	3.1
5237	12/14/11 6:46 AM	0	0	0	0	0	0	0	0		7	39	72	111	3.6
5238	12/14/11 8:14 AM	1	0	0	0	0	0	1	1		8	85	65	150	4.8
5239	12/14/11 9:17 AM	0	0	0	0	0	0	0	0		9	72	99	171	5.7
5240	12/14/11 10:36 AM	1	0	0	0	0	0	0	0		10	70	86	156	5.0
5241	12/14/11 11:44 AM	0	1	0	0	0	0	1	0		11	84	87	171	5.7
5242	12/14/11 12:46 PM	0	0	0	0	0	0	0	0		12	43	32	75	3.9

Column S in Figure 1.9 simply contains the numbers of the months that I'm interested in: May through December. When I get to May 2012, it will be necessary to add a column with the year, to distinguish May 2011 from May 2012.

Columns T and U contain array formulas. I describe the formulas in column T here; the formulas in column U work the same way but use different columns as the data sources.

The array formulas in column T check the month number in column S against the month implied by the date in column A. If the two month indicators are equal, the formula sums the values in columns J and K. I go into the topic of array formulas, what they require, and why they're sometimes needed, in Chapter 5, "Forecasting a Time Series: Regression." For now, though, it's enough to be aware that you need to enter these formulas with a special keyboard sequence. Instead of simply typing the formula and pressing Enter, you type it and then press Ctrl + Shift + Enter. This procedure is called *array-entering* a formula. Excel notices that you have array-entered a formula, and in that case it surrounds the formula as shown in the formula box with curly braces.

It's useful to take the formula apart to see what's going on inside it. Here's the full formula for the August figures:

```
=SUM(IF(MONTH($A$3:$A$1000000)=$S5238,J$3:K$1000000,0))
```

Working from the inside out, consider this fragment:

```
MONTH($A$3:$A$1000000)
```

The MONTH() function returns the month of a date, and column A consists of dates following the second row. When you apply the MONTH function to an array of dates, as is done here, you get an array of month numbers. (That's one reason the formula must be entered as an array formula using Ctrl + Shift + Enter: The MONTH() function normally expects to evaluate a single date value, not a whole array of them. Array-entering the formula alerts Excel to the possibility that a function that normally takes a single value will be taking multiple values.)

That MONTH() fragment is surrounded by an IF() function. The IF() function tests whether the month numbers returned by MONTH() equal the number in cell S5238. In Figure 1.9, cell S5238 contains the number 8—the eighth month, or August. So the IF() function begins by converting all those month numbers from MONTH() to TRUE and FALSE values, depending on whether MONTH() returns an 8.

The next argument to the IF() function in this case is the range J3:K1000000. That range contains the 1's and 0's calculated from the changes in the sales rankings. Those 1's and 0's are retained by the IF() given that the logical test, month equals 8, is true. The IF() function retains any row where the month of the date in column A is 8, and in that case it holds on to the values found in columns J and K.

What if a date in column A does not fall in the month of August? Then the logical test posed by the IF() function fails, and the IF()'s third argument, 0, is used.

Here's the formula as I've discussed it so far:

```
IF(MONTH($A$3:$A$1000000)=$S5238,J$3:K$1000000,0)
```

More briefly: If the dates in column A are in the month shown in S5238 (which is 8), use the associated values in columns J and K. Otherwise, use the number 0.

Finally, surround the fragment with the SUM() function:

```
=SUM(IF(MONTH($A$3:$A$1000000)=$S5238,J$3:K$1000000,0))
```

That is, take the sum of the values in columns J and K if they were observed during August, using zero instead if they were observed during some other month.

The same formula, adjusted to total two different columns, is in cell U5238:

```
=SUM(IF(MONTH($A$3:$A$1000000)=$S5238,L$3:M$1000000,0))
```

Here, we total the values found in columns L and M if the associated date is from August. Columns L and M contain sales estimates for a different book than columns J and K. (You probably have noticed that a book gets two columns because there is a paperback edition as well as a Kindle edition of each.)

Column V simply totals the results from columns T and U:

```
=T5238+U5238
```

And column W calculates the average number of books sold during the month:

```
=V5238/31
```

That calculation is a little tricky during the current month. Cell W5242 uses this formula while the current date is still within December:

```
=V5242/(TODAY()-DATEVALUE("11/30/2011"))
```

I don't want to divide by 31 before we reach the end of the month, so I need to calculate the number of days that have elapsed during the current month. That number is the difference between the value returned by the TODAY() function (which gives today's date) minus the value of the last day in the prior month. So if today is December 20, the result of the subtraction is 20.

The VBA Code

This section walks you through the VBA code that I use to update my data collector hourly. It runs itself, so I can get it going, walk away, and not come back for hours or days. You can start by using Alt+F11 to open the Visual Basic Editor. Then arrange for a fresh module by choosing Insert, Module. The code that's shown in this chapter, or your own VBA code, goes into the new module.

Although the figures in this chapter indicate that I'm obtaining data on several books, I'm limiting the code used here to just two. You can extend it to more products than two if you want.

I should also note that I've revised the code somewhat to make clearer what's going on. A number of modifications would structure the code a bit more tightly and run a little faster. But they tend to obscure the general flow of the task.

```
Option Explicit
```

I always specify Option Explicit at the top of my VBA modules. Doing so forces me to declare variables explicitly, often with `Dim` statements. For example:

```
Dim NextRow as Integer
```

`Dim`, short for *dimension*, informs VBA that a variable named `NextRow` is intended to exist in the module's code: The variable is *declared* to exist. Later on, if I mistype the name of the variable in the code

```
NextRoe = NextRow + 1
```

for example, VBA complains that I'm trying to use a variable that I haven't declared. If I don't have `Option Explicit` at the top of the module, VBA assumes that I meant to *implicitly* declare a new variable, `NextRoe`, and assign to it the current value of `NextRow + 1`. The implicit, on-the-fly declaration of variables was common programming practice 40 years ago. VBA still tolerates it unless you protect yourself with `Option Explicit`.

The DoItAgain Subroutine

This very short, three-statement subroutine causes the main `GetNewData` subroutine to run an hour from now:

```
Private Sub DoItAgain()
Application.OnTime Now + TimeValue("01:01:00"), "GetNewData"
End Sub
```

The `DoItAgain` subroutine uses VBA's `OnTime` method to schedule the time when a procedure is to run next. The time is specified first, using the fragment `Now + TimeValue("01:01:00")`. VBA has a function named `Now` that returns the current system date and time. VBA has another function named `TimeValue` that converts a string to a time value.

So, `Now + TimeValue("01:01:00")` gets the time as of right now and adds to it one hour, one minute, and zero seconds.

The second argument to `OnTime` as used here is `GetNewData`, which is the code's main procedure. Its name must be enclosed in quotes when used as an argument.

The full statement instructs VBA to run the procedure named `GetNewData`, 61 minutes after the `OnTime` method is invoked.

I settled on an hour and a minute after some experimenting. Amazon states that it updates the rankings hourly, and that seems to be nearly correct. Sometimes the update doesn't happen exactly when it should, and sometimes several hours pass before the next update takes place. These events are mildly atypical, though, and something close to an hour is normal.

However, it takes a bit of time for my code to run: to acquire new data by way of the queries and to test the changes in rankings for evidence of a sale. So I add a minute to the one hour in order to catch back up with Amazon's schedule.

The GetNewData Subroutine

Here’s the main procedure: `GetNewData`. It executes web queries, obtains sales rankings, and copies and pastes formulas that convert rankings to assumed sales figures.

```
Sub GetNewData()
Dim NextRow As Integer, NextRank As Long
Dim UnitsLeft As Integer, NextLeft As Integer
```

I begin by declaring the variables I need in this procedure:

- `NextRow` is the worksheet row where I write the next set of query data.
- `NextRank` contains the next sales ranking for a given book.
- `NextLeft` contains the number of units that Amazon has left in stock, if that figure is reported on the book’s Amazon page.

NOTE

A variable such as `NextRow`, if declared as type Integer, can take on only integer values with a maximum positive value of 32767. I don’t expect to use more than that number of rows in my application. A variable such as `NextRank` (which stores the book ranking) can easily have a value greater than 32767, and so I declare it as Long. *Long* is a “long integer,” takes integer values only, and has a maximum positive value of more than two billion. That’s plenty large enough to store an Amazon book ranking.

```
Application.ScreenUpdating = False
Application.Calculation = xlCalculationManual
```

I don’t like to be distracted by the screen if I happen to be looking at it when the code changes the active worksheet. So I turn off screen updating and the Excel window stays put until the code has finished running, or until a statement turning screen updating back on is executed (you set its value to `True` instead of `False`).

Also, I don’t want formulas recalculating as the queries write new data to the worksheets; I let them recalculate after the queries have finished running. So, I set Excel’s calculation property to manual—later on I reset it to automatic.

```
RefreshSheets "Stats"
RefreshSheets "BAXL Kindle"
```

Now I call the `RefreshSheets` subroutine with the name of the query sheet as the argument. I call it once for each query sheet. As noted earlier, I actually call this subroutine eight times but I don’t want to subject you to all eight calls here—what happens is largely the same in each call. The `RefreshSheets` subroutine is shown and described following this main `GetNewData` subroutine at the end of the current section on the VBA code.

```
NextRow = ThisWorkbook.Sheets("Summary").Cells(Rows.Count, 1).End(xlUp).Row
➥+ 1
```

The code needs to find the next available row on the Summary sheet so that it can write the new data from the queries to that sheet without overwriting existing records. Here’s how it does that.

If you're using an Excel worksheet directly, one of the ways to move around is with Ctrl + *arrow*, where *arrow* means up or down arrow, right or left arrow. If the active cell is in a column of data, for example, you can find the last contiguous non-empty cell with Ctrl + Down Arrow. If you're at the bottom of a column, something such as A65536, and that cell is empty, you can find the lowermost non-blank cell with Ctrl + Up Arrow.

Using VBA, the way to emulate that keyboard action is with the `End` property, which belongs to a cell object (really a range object because a cell is a special instance of a range). So, `Cells(Rows.Count,1).End(xlUp)` tells VBA to go to the final row in Column A and act as though you had pressed Ctrl + Up Arrow. That takes VBA to the lowermost non-empty cell, assuming as I do that the cell defined by the final row of Column A is itself empty.

Then all I have to do is get the row where the non-empty cell is found, add 1 to the row number, and I have the row number that I need to write into next. The result is assigned to the variable `NextRow`.

I do have to tell VBA which cell to regard as the active cell, and I do that by citing the Summary worksheet, and directing VBA's attention to that worksheet's final row, first column. I start things out with `ThisWorkbook` just in case I have more than one workbook open and some other workbook is active at the time that the code starts to run. The `ThisWorkbook` object specifies the workbook that contains the code that is running.

```
ThisWorkbook.Sheets("Summary").Cells(NextRow, 1) = Now
```

Having found the next row to write in, I put the current date and time into the Summary worksheet, in the first column of that row.

```
With ThisWorkbook.Sheets("Summary")
```

The `With` statement initiates a block of code in which objects that are preceded by a dot, such as `.Cells(NextRow,2)`, are deemed by VBA to belong to the object named in the `With` block—here, that's the worksheet named Summary.

```
NextRank = GetRank("Stats", " in")
```

The `GetRank` procedure is called with the name of the query sheet named `Stats`, and the string `" in"`, which helps `GetRank` locate the sales ranking that has been retrieved by the query for the statistics book.

The `GetRank` procedure, which is actually a function written in VBA, is discussed later in this section. The value returned by that function, which is the sales rank of the book in question, is assigned to the variable `NextRank`.

```
.Cells(NextRow, 2) = NextRank
```

Then the value just assigned to `NextRank` is written to the cell at the intersection of the `NextRow`-th row and column 2. (Column 2 is specific to and reserved for sales rankings of the statistics book. When queries on other books are run later in the code, the value assigned at that point to `NextRank` is written to a different column.) Because of the earlier `With` statement, the cell in the `NextRow`-th row and the second column is taken to belong to the worksheet named Summary in the workbook that contains the running code.

```
NextLeft = GetUnitsLeft("Stats")
```

Now the code gets into some functionality that I have abandoned because it's a field that isn't updated with any frequency by Amazon—not, at least, for the books I'm interested in tracking. When Amazon's inventory of a book gets down to a certain level, often called the reorder point, the book's web page gets a text string along the lines of "Only 5 left in stock—order soon (more on the way)."

It occurred to me that by quantifying the number of sales between the dates when this message disappeared, and then reappeared, I might be able to calculate the number of copies Amazon ordered from the publisher. So, for a while I tracked that string, but then it became clear that for some reason Amazon was not putting that message up for the books I was most interested in. Furthermore, the message probably has no relevance to a book's Kindle edition. So I dropped the code, but I have restored it here in case you are interested in seeing how to deal with a message or other web data that might or might not be on a web page at the time you query it.

I get the figure by invoking another function in my code, `GetUnitsLeft`, which, like `GetRank`, takes the name of the book's query sheet as an argument. The `GetUnitsLeft` function is also discussed below. It returns a value to the `NextLeft` variable.

```
If NextLeft <> -1 Then
    .Cells(NextRow, 20) = NextLeft
End If
```

If information about the number of units left in stock is missing, `NextLeft` has been set equal to -1 and the code goes on to the next task. But if `NextLeft` has some value other than -1, that value is written to the cell in `NextRow`-th row and the twentieth column. Again, column 20 is reserved for units-on-hand information about the statistics book.

The next two statements get the sales rank for the book whose query sheet is named "BAXL Kindle" and write the sales rank to the `NextRow`-th row, third column on the Summary sheet:

```
NextRank = GetRank("BAXL Kindle", " Paid in")
.Cells(NextRow, 3) = NextRank
```

Notice that the second argument to the `GetRank` function is now `" Paid in"` instead of `" in"`. The reason is that this is a Kindle book, and Amazon follows the sales rank with `" Paid in"` rather than simply `" in"`. The string passed to the `GetRank` function is used to help strip out the actual ranking from its surrounding verbiage; you find out how when we get to that function.

In the prior section titled "Identifying Sales," I discussed how some worksheet formulas can be used to decide if a sale of a book occurred by comparing two consecutive rankings of the same book. Now the VBA code selects, copies, and pastes those formulas into the next row down. The newly pasted formulas then recalculate based on the differences between the most recent and the now-current rankings. Begin by activating the Summary worksheet:

```
.Activate
```

Select the cells that contain the formulas to copy. They are found in the prior row—that is, in the row that precedes NextRow, where we are now writing the current query data. Select the cells in that row from the tenth column to the seventeenth:

```
.Range(.Cells(NextRow - 1, 10), .Cells(NextRow - 1, 17)).Select
```

Autofill the selection one row down. The addresses used by the formulas automatically adjust so that they point to the newly current row:

```
    Selection.AutoFill Destination:=.Range(.Cells(NextRow - 1, 10), _
      .Cells(NextRow, 17)), Type:=xlFillDefault
```

Notice that the destination of the autofill includes the row NextRow -1 as well as NextRow. It's a minor peculiarity of the Autofill method that the destination range must include the source range. In this case, the source range is the row NextRow -1 (columns 10 through 17) and that range is to be autofilled into the row NextRow (columns 10 through 17). But that destination range as specified in the VBA statement must include the source range.

```
.Rows(NextRow + 1).Select
Selection.Insert Shift:=xlDown
.Cells(NextRow, 1).Select
```

The prior three statements insert a blank row following the NextRow row—that is, the row that the code has just populated with sales rankings and sales formulas. The reason is that there is a table of analyses (discussed later in this chapter) that are found a few rows below the rankings. It's helpful to push that table down a row when a new row has been populated with rankings.

```
End With
```

Terminate the With block that deems any object beginning with a dot (such as .Cells) to belong to the Summary worksheet in ThisWorkbook.

```
Application.Calculation = xlCalculationAutomatic
```

Turn automatic calculation back on.

```
ThisWorkbook.Save
```

Save the workbook.

```
Application.ScreenUpdating = True
```

Turn screen updates back on.

```
DoItAgain
```

Run the three-statement DoItAgain subroutine, discussed earlier, that causes the GetNewData subroutine to run again an hour from now.

```
End Sub
```

End the GetNewData subroutine.

The GetRank Function

Here's the function named GetRank. Notice that it has two arguments, SheetName and TrailingString. The values are passed to the function by the statement that calls it. For example:

```
NextRank = GetRank("Stats", " in")
```

where "Stats" is the worksheet name and " in" is a string of characters from the web page that follows—that is, trails—the actual ranking.

```
Function GetRank(SheetName As String, TrailingString As String) As Long
```

The function returns a value to the statement that called it. That value is often of a certain type: integer, a text string, a decimal number, and so on. To accommodate that, the function itself is declared to be of a certain type. Here, that type is Long. In VBA, Long means an integer value that might be much larger than the maximum value for a regular integer, which tops out at 32,767. There are many more books than that in Amazon's inventory, and this function is intended to return a book's ranking. Therefore, to accommodate the possibility that the rank is greater than 32,767, I declare the function as type Long, which can return a value greater than 2 billion—more than large enough.

```
Dim FoundString As String
Dim StartPos As Integer, StopPos As Integer
```

Three variables are declared. FoundString holds the value of the sales rank. It's declared as a string because when the code strips it out of the surrounding text, it's a string. Later it's converted to a Long integer.

StartPos and StopPos determine the starting and stopping positions of the sales rank within the string. For example, this string:

```
Amazon Best Sellers Rank: #64,788 in Books
```

has its first numeric representation in character number 28 of the string, so StartPos is assigned the value 28.

```
On Error GoTo EarlyOut
```

This code is designed to run 24/7, so I can't expect myself to nurse it along if it runs into an error that it can't recover from. The On Error statement tells VBA that if it encounters an error such as a letter right in the middle of what is supposed to be a numeric sales ranking, it shouldn't terminate processing. Instead, transfer control to a point named EarlyOut. That way, processing can continue. Even if it happens at 2:00 a.m., I can check it out at 9:00 a.m. and fix whatever happened, and I won't necessarily have lost data from the intervening seven hours.

The next five statements are responsible for finding the cell in the worksheet that contains the sales ranking, stripping the numeric ranking out of that cell, and assigning the result to the GetRank function.

First, find the cell that contains the string "sellers Rank:". The rank we're looking for comes directly after that string.

```
    Cells.Find(What:="sellers Rank:", LookIn:= _
         xlFormulas, LookAt:=xlPart, SearchOrder:=xlByRows, _
         SearchDirection:= xlNext, MatchCase:=False).Activate
```

The `Find` method is used to locate the string. Most of the arguments, such as `LookIn` and `LookAt`, are there to keep the settings in place for the Find dialog box in Excel's user interface. I don't much care about forcing a `Case` match, so long as I get the string I'm looking for. Finally, I use the `Activate` method at the end to make the found cell active.

```
    FoundString = ActiveCell
```

For convenience in subsequent handling, I set the string variable `FoundString` to the contents of the cell where `"sellers Rank:"` is found.

```
    StartPos = InStr(FoundString, " #") + 2
```

I use VBA's `InStr` function to locate the space and the pound sign (#) in the contents of the active cell. I want to start the string that contains the sales rank numbers immediately after the space and pound sign, so I add 2 to the position where that substring is found. The result is stored in `StartPos`.

```
    StopPos = InStr(FoundString, TrailingString)
```

Then I look inside `FoundString` again, this time to find the trailing string—the characters that Amazon supplies right after the sales ranking. In the case of tangible books, the ranking is followed by `" in"`. In the case of Kindle books, the ranking is followed by `" Paid in"`—or at least that's the case in December 2011. So the value of `TrailingString` is passed to the `GetRank` function, along with the name of the query sheet. If I'm querying for a tangible book, I pass `" in"` and if I'm querying for a Kindle book, I pass `" Paid in"`.

```
    GetRank = 1 * Mid(FoundString, StartPos, StopPos - StartPos)
```

Finally, the code gets the value of the sales ranking using VBA's `Mid` function, which returns a string that is inside another string. In this case, `Mid` is instructed to look in `FoundString`, beginning at `StartPos`, and to return `(StopPos - StartPos)` characters. Those characters are the ranking, but because they came from a string they are still a string. Therefore I multiply the result by 1 to coerce the string to a numeric value, and assign it to `GetRank`.

NOTE One thing you should always do in a function that you write yourself is to assign a value to the function before the function terminates. This is not an absolute requirement—you won't get a compile error if you fail to do so—but it's important nevertheless and particularly so if you're writing a function that the user can enter on the worksheet.

The prior statement assigns the value of the sales rank to the function `GetRank` itself. When the function terminates and control returns to the statement that called the function, `GetRank` itself equals the value that it calculated. And in the statement that called the function, which is `NextRank = GetRank("BAXL", " in")`, the variable `NextRank` is assigned the value of the `GetRank` function.

```
    On Error GoTo 0
```

This second `On Error` statement returns the default status. Telling VBA to "go to 0" in the event of an error causes VBA to terminate with a run-time error if it hits another error. When this function cedes control back to the calling statement, the prior `On Error` statement is no longer in effect and so this `On Error GoTo 0` statement could be omitted, but I wanted you to know about it.

```
Exit Function
```

In the normal course of events, the function terminates at the `Exit Function` statement and control returns to the statement that called the function. But if something goes wrong, the `Exit Function` statement is bypassed and some minor housekeeping takes place first.

Here's the label I mentioned earlier. Control transfers here if something goes wrong in the portion of the function that locates the sales ranking and converts it to a `Long` integer:

```
EarlyOut:
GetRank = 0
```

The value returned by the function is set to zero. That's the value that is written to the Summary sheet as the sales rank. When I see that, I know that something went wrong. Perhaps Amazon's code had removed the old ranking from its page and was getting ready to put the new ranking in place at the moment that this code executed its query.

```
End Function
```

After a value, even an illogical one such as a sales ranking of zero, is assigned to the function, its task is completed and control returns to the calling statement.

The GetUnitsLeft Function

Here's the function mentioned earlier that determines how many units of a particular book are left.

```
Function GetUnitsLeft(SheetName As String) As Long
```

The name of the query sheet is passed as `SheetName` to the function. The function returns a value of the Long integer type.

```
Dim FoundString As String
```

As in the `GetRank` function, `FoundString` will contain the value in the cell we're looking for.

```
Dim StartPos As Integer, StopPos As Integer
```

`StartPos` and `StopPos` have the same purposes that they have in `GetRank`: to bracket the value you want to return to the main procedure.

```
GetUnitsLeft = -1
```

Start out by assigning a default value to the function.

```
Sheets(SheetName).Activate
```

Activate the query sheet.

```
On Error GoTo Nodata
```

Establish an error handler: go to the Nodata label in the event of a runtime error.

```
Cells.Find(What:="left in stock", LookIn:= xlFormulas, _
     LookAt:=xlPart, SearchOrder:=xlByRows, SearchDirection:= _
     xlNext, MatchCase:=False).Activate
```

Execute the Find method. In this case, where you are looking for the number of units in stock, there is a particular, invariant string value that locates the cell we're after: "left in stock". Therefore, you don't need to pass a different string to the procedure as an argument, depending on which book we're querying.

```
FoundString = ActiveCell
```

The contents of the cell that Find found are assigned to the FoundString variable.

```
StartPos = InStr(FoundString, "Only ") + 5
```

The InStr function returns the position in the searched string where "Only " *starts*. Therefore, add the five characters in "Only " to the result of InStr to locate the position where the number left in stock begins.

```
StopPos = InStr(FoundString, " left in")
```

StopPos locates the position where " left in" begins.

```
GetUnitsLeft = 1 * Mid(FoundString, StartPos, StopPos - StartPos)
```

Finally, look in FoundString starting at StartPos and use the Mid function to strip out as many characters as the difference between StopPos and StartPos. Multiply by 1 to convert the result from a text string to a number, and assign the result to the function.

```
Exit Function
```

The Exit statement returns control to the calling statement. In case an error occurred, though, control has already passed to the Nodata label.

```
Nodata:
```

More often than not the reorder point for a product has *not* been reached, and so an error usually occurs. When it does, control comes here and you simply end the function. Recall that you began by assigning the function the value of -1, so if the string you're looking for could not be found then the function retains the value of -1 and passes that value back to the calling statement.

```
End Function
```

The RefreshSheets Subroutine

This subroutine runs once for each query sheet. It's called by the main GetNewData subroutine.

```
Sub RefreshSheets(SheetName As String)
```

The name of the query sheet is passed as an argument.

```
With ThisWorkbook.Sheets(SheetName)
```

A `With` block is initiated so that subsequent objects, properties, and methods that begin with a dot are taken to belong to the sheet named `SheetName` in `ThisWorkbook`.

```
    .Activate
```

The current query sheet is activated so that its query can be run.

```
    .Cells(1, 1).Select
```

Select cell A1. Earlier I recommended that you cause the queries to return their results beginning in cell A1. If you adopt that recommendation, you know that the query results include that cell A1 and that if you select it you're able to refresh the query. (It's necessary that a cell in the query results range be active for the refresh to take place.)

```
    Selection.QueryTable.Refresh BackgroundQuery:=False
```

Refresh the query. Set `BackgroundQuery` to `False` so that the code does not continue while the refresh is taking place.

```
    End With
    End Sub
```

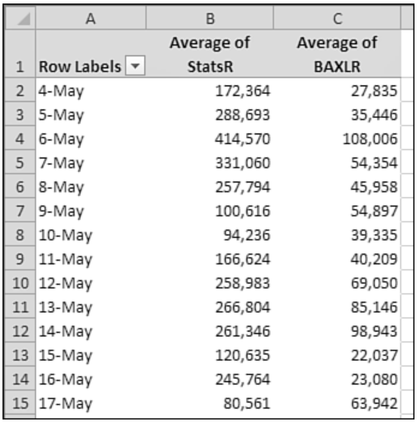
Terminate the `With` block and end the subroutine.

The Analysis Sheets

I use several worksheets and chart sheets to analyze what's going on with the sales of my books. For an analysis that looks at the data over time, a pivot table (and pivot chart) is a nearly perfect solution. It collapses the hourly figures that I get from my web queries into more manageable time slices—I find that summarizing by day strikes a good balance between the overwhelming amount of data in an hourly analysis and the lack of detail in a monthly or even weekly analysis. Of course, when your baseline extends over several hundred days, it's time to start thinking about weeks or months as your unit of analysis.

Figure 1.10 shows one of the pivot tables that I rely on to synthesize the data on sales rankings.

Figure 1.10
When you connect this pivot table to a pivot chart, you can start making real sense of the data.



	A	B	C
1	Row Labels	Average of StatsR	Average of BAXLR
2	4-May	172,364	27,835
3	5-May	288,693	35,446
4	6-May	414,570	108,006
5	7-May	331,060	54,354
6	8-May	257,794	45,958
7	9-May	100,616	54,897
8	10-May	94,236	39,335
9	11-May	166,624	40,209
10	12-May	258,983	69,050
11	13-May	266,804	85,146
12	14-May	261,346	98,943
13	15-May	120,635	22,037
14	16-May	245,764	23,080
15	17-May	80,561	63,942

As powerful as pivot tables are—and I believe that pivot tables are the most powerful and flexible tool for data synthesis and analysis available in Excel—they can't tell when you have changed the underlying data, and (without help) they can't tell that their underlying data range has added another row.

In contrast, something as simple as the SUM() function can update itself when the underlying values change. If you have entered this formula

=SUM(A1:A5)

in some cell, the value it returns changes immediately if you change any of the values in A1:A5. That's not true of a pivot table that's based on those cells or any other cell. You have to do something special to refresh the table when the underlying data changes.

But even SUM() won't change its own argument. If you now put a new value in A6, SUM() doesn't change itself from SUM(A1:A5) to SUM(A1:A6).

The way I prefer to handle that is by means of *dynamic range names*. You might already know that you can assign a name to a worksheet range and use that name instead of a range address. If you have given A1:A5 the name *Addends*, you can use this instead of SUM(A1:A5):

=SUM(Addends)

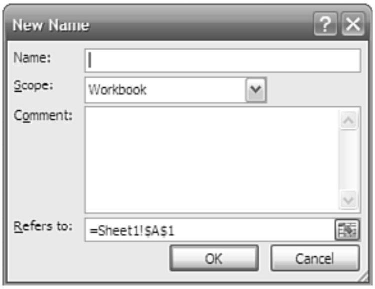
I show you how to name a range shortly. First, you should know that if you define a range properly, you can get it to change its own dimensions when you add a new row or column to it. (Tables, new in Excel 2007, do that automatically, and you should probably investigate their capabilities. I still prefer dynamic range names because of a possibly irrational belief that I can control them better.)

Defining a Dynamic Range Name

Begin by selecting the worksheet that contains the range that you want to name. This step is not strictly necessary, but I usually find it helpful. Then follow these steps:

1. Click the Ribbon's Formulas tab.
2. Click the Define Name button in the Defined Names group. The New Name dialog box shown in Figure 1.11 appears.

Figure 1.11
You can use a formula instead of an address in the Refers To box.


New Name

Name:

Scope: Workbook

Comment:

Refers to: =Sheet1!$A$1

3. Enter the name you want to establish in the Name box.
4. In this case, leave the Scope with the default value of Workbook.
5. I base the pivot table on the data in my Summary sheet, shown in Figure 1.7. With the Summary sheet's layout as shown there, type the following formula in the Refers To box:

   ```
   =OFFSET(Summary!$A$2,0,0,COUNTA(Summary!$B:$B),3)
   ```
6. Click OK.

Here's a quick explanation of the formula. The OFFSET() function returns a range of cells that are offset from an anchor cell. Here, the anchor cell is defined as A2 on the Summary sheet. Notice in Figure 1.7 that cell A2 is where the label "When" is entered, and more labels follow it in row 2, columns B through I. I want to include those labels in the defined range because they're needed for the pivot table.

The two zeros that follow $A$2 in the OFFSET() function tell Excel how many rows and how many columns away from A2 the resulting range should begin. In this case, because both values are zero, the resulting range is offset by zero rows and zero columns: that is, the range begins at cell A2.

The fourth argument to the OFFSET() function, COUNTA(Summary!$B:$B), tells Excel how many rows to include in the offset range. The COUNTA() function tells Excel to count the number of values in (here) column B on the Summary worksheet. So when the VBA code runs and adds a new value in column B, the COUNTA function counts an additional value and redefines the number of rows in the range. That's how adding a new row at the bottom of the range causes the definition of the range to increase by one row.

NOTE

I use COUNTA() instead of COUNT() because I want to include the text value in cell B2. Using COUNT() would ignore that text value; COUNTA() pays attention to both numeric and alphanumeric values.

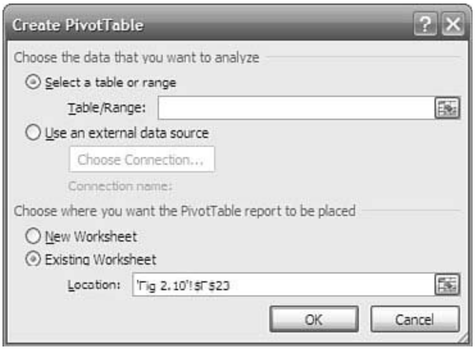
The final argument to the OFFSET() function tells Excel how many columns to include in the result range. Here I want to include three columns: one for the date, one for the Stats book rankings, and one for the BAXL book rankings.

It's called a *dynamic* range name because the dimensions of the range can change depending on the number of records or fields, or both, that are added to or removed from the range.

Using the Dynamic Range Name

With the dynamic range name defined, you can use it when you create a pivot table. Here's how to do that:

1. Begin by activating a blank worksheet and then click the Ribbon's Insert tab.
2. Click Pivot Table in the Tables group. The dialog box shown in Figure 1.12 appears.



Create PivotTable

Choose the data that you want to analyze

● Select a table or range

Table/Range:

○ Use an external data source

Choose Connection...

Connection name:

Choose where you want the PivotTable report to be placed

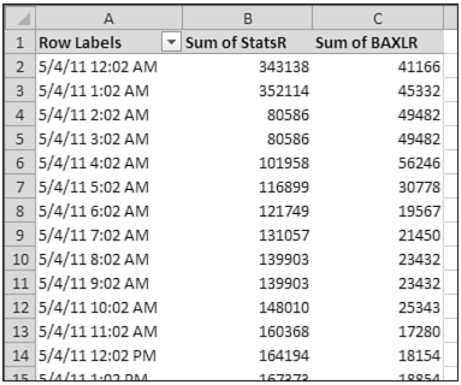
○ New Worksheet

● Existing Worksheet

Location: 'Fig 2.10'!$F$23

Figure 1.12
Use the dynamic range name you defined in the Table/Range box.

3. Type the name you supplied in Step 3 of the prior numbered list into the Table/Range box. This causes Excel to use that range as the data source for the pivot table.
4. Click OK. The PivotTable Field List appears.
5. Drag the When field into the Row Labels box.
6. Drag the StatsR field into the Σ Values box.
7. Drag the BAXLR field into the Σ Values box.
8. Dismiss the PivotTable Field List by clicking its Close box. The pivot table now appears as shown in Figure 1.13.



	A	B	C
1	Row Labels	Sum of StatsR	Sum of BAXLR
2	5/4/11 12:02 AM	343138	41166
3	5/4/11 1:02 AM	352114	45332
4	5/4/11 2:02 AM	80586	49482
5	5/4/11 3:02 AM	80586	49482
6	5/4/11 4:02 AM	101958	56246
7	5/4/11 5:02 AM	116899	30778
8	5/4/11 6:02 AM	121749	19567
9	5/4/11 7:02 AM	131057	21450
10	5/4/11 8:02 AM	139903	23432
11	5/4/11 9:02 AM	139903	23432
12	5/4/11 10:02 AM	148010	25343
13	5/4/11 11:02 AM	160368	17280
14	5/4/11 12:02 PM	164194	18154
15	5/4/11 1:02 PM	167373	18854

Figure 1.13
You still need to group on date and to show the average instead of the sum.

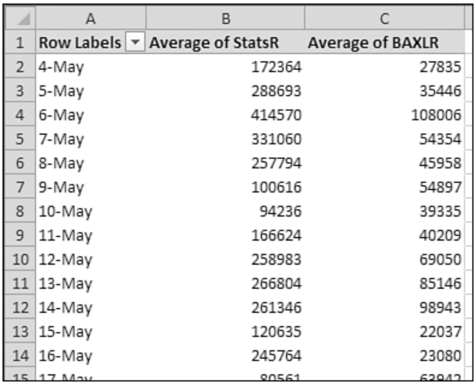
9. Right-click in the date column of the pivot table and choose Group from the shortcut menu.
10. Grouping by Month is the default selection. Click it to deselect it, and click Day to select it instead. Click OK.
11. The pivot table might seem to disappear from the visible worksheet. That can happen when the grouping operation causes the pivot table to occupy many fewer rows. Just

scroll up to it and then right-click in the column for the first book's rankings. Choose Value Field Settings from the shortcut menu.

12. Change the summary type from Sum to Average and click OK.
13. Right-click in the column for the second book's rankings, choose Value Field Settings, and change the summary type from Sum to Average. Click OK.

The pivot table should now appear as shown in Figure 1.14.

Figure 1.14
You can also use the Value Field Settings to select a display format for the summary values in the pivot table.

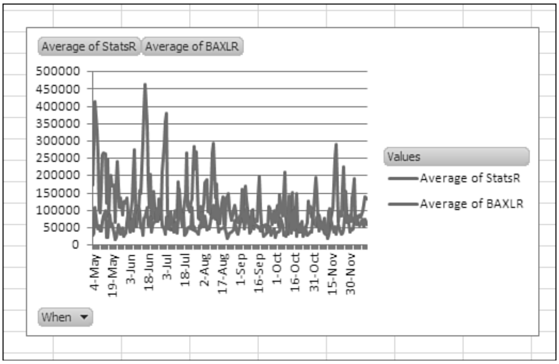


	A	B	C
1	Row Labels	Average of StatsR	Average of BAXLR
2	4-May	172364	27835
3	5-May	288693	35446
4	6-May	414570	108006
5	7-May	331060	54354
6	8-May	257794	45958
7	9-May	100616	54897
8	10-May	94236	39335
9	11-May	166624	40209
10	12-May	258983	69050
11	13-May	266804	85146
12	14-May	261346	98943
13	15-May	120635	22037
14	16-May	245764	23080
15	17-May	80561	63942

A pivot table such as the one in Figure 1.14 isn't very informative by itself. There are too many data points to make any useful inferences from it. Fortunately, it's easy to create a pivot chart from a pivot table. Just select any cell in the pivot table, click the Ribbon's Insert tab, and choose (say) a Line chart from the Charts group.

You get a chart embedded in the active worksheet, and you can usually tell more about your data, at least on a macro level, from the chart than directly from the pivot table (see Figure 1.15).

Figure 1.15
You usually need to do some tinkering with the pivot chart before it starts to show you what you need to know.



One of the useful aspects of pivot charts is that they don't need any special handling to keep up with the underlying pivot table. When the pivot table's data changes, or gets more rows, the pivot chart updates automatically.

You do need a way to force a refresh of the pivot table. There are many ways to handle that. Some are mildly complicated, such as writing a special Worksheet Activate event handler using VBA. That event handler would force a refresh any time you activate the worksheet that contains the pivot table. Probably the simplest way is to right-click any cell in the pivot table and to choose Refresh from the shortcut menu.

Among the tweaks I like to apply to a newly created pivot chart are the following:

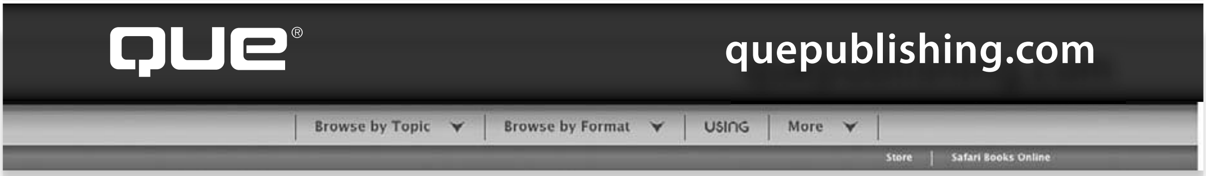
- Right-click an embedded chart, click Move in the shortcut menu, and choose to move the chart to a new sheet.
- Right-click a data series in the chart and choose to format it. Reduce the width of the lines and change any obnoxious color to a more acceptable one—black, for example.
- Add a trendline. Right-click a data series and choose Add Trendline from the shortcut menu. In line charts such as the one in Figure 1.15, linear trendlines and moving average trendlines are often very informative. Tinker with the length of a moving average trendline until you get something that smooths out the rough in the series but preserves its basic movement. Chapter 3, "Forecasting with Moving Averages," and Chapter 4, "Forecasting a Time Series: Smoothing," provide guidance on selecting the length of a moving average.

This page intentionally left blank

INDEX

How can we make this index more useful? Email us at indexes@quepublishing.com

This page intentionally left blank


que®
quepublishing.com
Browse by Topic
Browse by Format
USING
More
Store
Safari Books Online

Business Management
Finance and Investing
Graphics, Pictures & Video
Gadgets & Hardware
General Computing
Entertainment & Gaming
Internet & Web Apps
Computer Software
Operating Systems
Web Design & Development