

# EXCEL BASICS TO BLACKBELT

THIRD EDITION

An  
Accelerated  
Guide to  
Decision  
Support  
Designs

Elliot Bendoly

CAMBRIDGE

# EXCEL BASICS TO BLACKBELT

*Third Edition*

This third edition capitalizes on the success of the previous editions and leverages the important advancements in visualization, data analysis, and sharing capabilities that have emerged in recent years. It serves as an accelerated guide to decision support designs for consultants, service professionals, and students. This “fast track” enables a ramping up of skills in Excel for those who may have never used it to reach a level of mastery that will allow them to integrate Excel with widely available associated applications, make use of intelligent data visualization and analysis techniques, automate activity through basic VBA designs, and develop easy-to-use interfaces for customizing use. The content of this edition has been completely restructured and revised, with updates that correspond with the latest versions of software and references to contemporary add-in development across platforms. It also features best practices in design and analytical consideration, including methodical discussions of problem structuring and evaluation, as well as numerous case examples from practice.

Elliot Bendoly is Distinguished Professor of Management Sciences at the Fisher College of Business, Ohio State University. He serves as an associate editor for the *Journal of Operations Management*, a senior editor for *Production and Operations Management*, and an associate editor for *Decision Sciences*. His previously published books include (as co-editor) *Visual Analytics for Management* (2016) and *The Handbook of Behavioral Operations Management* (2015).



# EXCEL BASICS TO BLACKBELT

*An Accelerated Guide to Decision Support Designs,  
Third Edition*

ELLIOT BENDOLY

*Ohio State University*



CAMBRIDGE  
UNIVERSITY PRESS

**CAMBRIDGE**  
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre,  
New Delhi – 110025, India

79 Anson Road, #06–04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of  
education, learning, and research at the highest international levels of excellence.

[www.cambridge.org](http://www.cambridge.org)

Information on this title: [www.cambridge.org/9781108738361](http://www.cambridge.org/9781108738361)

DOI: 10.1017/9781108768641

© Elliot Bendoly 2008, 2013, 2020

This publication is in copyright. Subject to statutory exception  
and to the provisions of relevant collective licensing agreements,  
no reproduction of any part may take place without the written  
permission of Cambridge University Press.

First published 2008

Second edition published 2013

Third edition published 2020

Printed in the United Kingdom by TJ International Ltd. Padstow Cornwall

*A catalogue record for this publication is available from the British Library.*

*Library of Congress Cataloging-in-Publication Data*

Names: Bendoly, Elliot, author.

Title: Excel basics to blackbelt : an accelerated guide to decision support designs, third edition /  
Elliot Bendoly, Ohio State University.

Description: Third edition. | New York, NY : Cambridge University Press, 2020. | Includes index.  
Identifiers: LCCN 2019059887 | ISBN 9781108738361 (paperback) | ISBN 9781108768641 (ebook)

Subjects: LCSH: Decision support systems. | Microsoft Excel (Computer file)

Classification: LCC HD30.213 .B46 2020 | DDC 005.54--dc23

LC record available at <https://lccn.loc.gov/2019059887>

ISBN 978-1-108-73836-1 Paperback

Cambridge University Press has no responsibility for the persistence or accuracy of  
URLs for external or third-party internet websites referred to in this publication  
and does not guarantee that any content on such websites is, or will remain,  
accurate or appropriate.

# Contents

<i>Associated Links</i>	<i>page ix</i>
<i>Preface</i>	xi
<b>SECTION 1 GETTING ORIENTED</b>	1
<b>1 Necessary Foundations for Decision Support</b>	3
1.1 Intentions and Approaches	3
1.2 Stages in Development	5
1.3 Overview and Resources	8
<b>2 The Common Development Environment</b>	12
2.1 Entering Data and Simple Extrapolation	13
2.2 Format and Attributes of Cells	17
2.3 Functions (An Introduction)	25
2.4 Data Management (An Introduction)	33
2.5 Copying Content	34
Supplement: The Nature of Conditional Statements	35
Practice Problems	39
<b>3 Acquisition, Cleaning, and Consolidation</b>	41
3.1 Text File Imports and Basic Table Transfers	41
3.2 Online Data Acquisition	44
3.3 Living Data Records: The Basics	53
3.4 Selective Filtering and Aggregation	58
3.5 Guidance in Attribute Grouping	60
3.6 Guidance in Record Grouping	67
3.7 Caution with Data	82
Supplement: Unique Data Generation Hacks	82
Practice Problems	85

<b>SECTION 2 STRUCTURING INTELLIGENCE</b>	87
<b>4 Estimating and Appreciating Uncertainty</b>	89
4.1 Prediction and Estimating Risk	90
4.2 Simulating Data: The Basics	97
4.3 The Use of Simulation in Analysis	105
4.4 Examples in Simulation Modeling	107
Supplement: Control Made Friendly	135
Practice Problems	140
<b>5 Visualizing Data, Dynamics and Risk</b>	142
5.1 Approaching the Task of Visualization	143
5.2 Working with Standard Charts	145
5.3 Complex Area Visualizations	168
5.4 Visualizing System Structures	180
5.5 Options in Other Platforms	191
Supplement: Dynamics along Visually Derived Paths	192
Practice Problems	194
<b>SECTION 3 PRESCRIPTION DEVELOPMENT</b>	197
<b>6 The Analytics of Solution Search</b>	199
6.1 Encapsulating Decision-Making Contexts	200
6.2 Developing Solutions from Structures	210
6.3 Optimality Searches under Moderate Complexity	220
6.4 Deeper Insights into Optimization Solutions	239
Supplement: Decision Logic with Interdependent Utilities	249
Practice Problems	251
<b>7 Searches in Highly Complex and Uncertain Landscapes</b>	255
7.1 Limitations of Simple Hill Climbing	255
7.2 Genetic Algorithms for Evolutionary Search	264
7.3 Evolutionary Search Using @RISK	267
7.4 Simulation Optimization Basics	284
7.5 Optimization for System Simulations	289
Supplement: Leveraging Solver's Evolutionary Search Engine	300
Practice Problems	302
<b>SECTION 4 ADVANCED AUTOMATION AND INTERFACING</b>	305
<b>8 VBA Editing and Code Development</b>	307
8.1 The Visual Basic for Applications Editor	307
8.2 Previewing Unique VBA Capabilities	314
8.3 Syntax and Storage	322

<i>Contents</i>	vii
8.4 Common Operations in VBA	333
8.5 User Defined Functions (UDFs)	345
8.6 Error Handling	353
8.7 Increasing Access to Code: Creating Add-ins	356
Supplement: Coding Parallels in iOS Swift	358
Practice Problems	360
<b>9 Application and User Interfacing</b>	362
9.1 Application Automation and Integration	364
9.2 Guided Design and Protection in Workbooks	382
9.3 GUIs beyond the Worksheet: Userforms	386
9.4 Customizing Excel Ribbon and Menu Interfaces	393
9.5 Final Thoughts on Packaging	399
Supplement: Simple Interface Parallels in iOS Swift	400
Practice Problems	402
<i>Glossary of Key Terms</i>	405
<i>Appendix A Workbook Shortcut (Hot Key) Reference</i>	419
<i>Appendix B Blackbelt Ribbon Add-in Tools and Functions</i>	421
<i>Index</i>	425



## Associated Links

**Main resource/course portal:** [www.excel-blackbelt.com](http://www.excel-blackbelt.com)

**Blackbelt Ribbon add-in:** *Most current versions always updated on above site.*

**Example workbooks from the text:**

Available either through the Blackbelt course portal (left sidebar), or using [www.bbexamples.com/](http://www.bbexamples.com/) followed by the chapter number and the filename (provided in the text). For example, to access Chp2\_IdentitiesList.xlsx, use [www.bbexamples.com/Chp2/Chp2\\_IdentitiesList.xlsx](http://www.bbexamples.com/Chp2/Chp2_IdentitiesList.xlsx)

**Demonstration Blackbelt projects:**

<https://sites.google.com/site/basic2blackbelt/home/Gallery>

**LinkedIn group** (which contains Non-English language subgroups):

[www.blackbeltsgroup.com](http://www.blackbeltsgroup.com)

**Data Visualization resources:** [www.ma-vis.com](http://www.ma-vis.com)



## Preface

Change can be daunting. People tend to resist it, and often for good reason. We can't control everything that life and work throws at us, but we can control a good deal of it. And it's that control that allows us to be effective in dealing with that which is unavoidably dynamic and out of our control. Without that stability, without a little bit of caution regarding change, we'd spend most of our time fighting fires and chasing shiny objects, and not actually getting things done.

On the other hand, when there are clear opportunities for doing things better through change, we need to capitalize on them. A little well-planned chasing can go a long way. It's how we learn. Again, this doesn't mean that we should drop everything the moment a new technology or framework pops up, with the hope that it might pay off. Instead it just means that we need to be deliberate in our *choices* of which changes to embrace and which, at least for the moment, to disregard.

The backdrop for this third edition is certainly one that is full of new opportunities in complementary and alternative technologies for robust and highly accessible decision support tool development. This growth has taken place both within the sphere of ubiquitous Microsoft applications as well as across myriad other development environments (G-Suite, R, Tableau, Swift builds for iOS, etc.). However, the backdrop also involves something far more personal. It is painted by my own experiences in tool development and, more generally, problem-solving processes in educational settings and in practice; experience studying the behavior of other individuals as they face these challenges; and exposure to best practices in leveraging intelligent structures in processes and intermediate products, including the effective utilization of visualization in the field.

In the time between the original edition and this publication, I've assembled some of these experiences into a handbook on human behavior in complex management decision-making settings, and others into a handbook on the psychology and practice around visualization for

management. Along with these texts, I've authored an additional 50-plus articles, largely on how individuals in the field and in controlled settings interact with technology. I've taught decision support tool development, business analytics, and management visualization courses to nearly one thousand individuals in master's, MBA, and undergraduate business programs, involving approximately 200 proof-of-concept builds for industry and individual users. Through all of this, I've seen some technical and process approaches overtaken (appropriately) by the march of technical progress, while others show continued strength and adaptability. It is high time that some of these learnings are folded into the discussions of decision support tool approaches in Excel/VBA.

Frameworks such as the OUtCoMES Roadmap, and its more contemporary descendant the OUtCoMES Cycle, have evolved. They now provide guidance beyond model development and into broader systems and design thinking applications, demonstrating alignment with and enrichment of other established frameworks (PDCA, A3, etc.). Specialized visualizations have been developed to support this as part of the Blackbelt Ribbon suite, whose functionality and structure provide examples we will discuss throughout the text. Conversations distinguishing and integrating descriptive, predictive, and prescriptive analytical intentions, and statistical versus computational approaches, have also proven effective in training individuals for (and placing them in) a wide variety of professional management, consultant, and analyst roles. These distinction and integration opportunities are similarly reflected in the current edition.

In the case of technical approaches that have been overtaken, there are also lessons in pragmatism to be learned. These include the replacement of MapPoint by PowerMap and the MS Power BI Suite; the supplanting of earlier interactivity between Excel and G-Suite, XLStat, and Palisade products by advancements in these respective applications; Microsoft's divestment of MSNBC, impacting MSN stock data connections; and the ever-shifting landscape of RExcel. Fortunately, we are not at a loss for technical guidance on how to adapt to many of these changes. There are countless blogs and publications that maintain this chase, and we are all indebted to the many expert developers who continue to inform us on these fronts.

With this in mind, and a shift in tactics from the second edition, the present text focuses less on examples that are likely to be overtaken in the near term, and more on examples of process and on specific technical guidance largely resilient to those changes, either because they are both sufficiently robust and extensively established in practice, or because I have direct influence on their maintenance (e.g., the revised Blackbelt Ribbon suite of tools). The biggest exception to this is the additional time afforded to technical discussions relevant to the mobile application space (use of the Power BI

desktop to share dynamic visuals, communication with G-Suite tools, Swift code examples, etc.). This is an area in which we need greater management analytic discourse, so I'm willing to do a little chasing here.

In all of this, it almost goes without saying that the course of our discussions will continue to focus on what today remains the most flexible, accessible, and relevant platform for management tool inception and testing in the world, namely the Excel and the VBA developer combined environment. With the scale and velocity of data available today, we don't approach these discussions with the rose-tinted view that this is the destination of our build-journeys. As a clearer picture of our needs in practice evolves, we seek out appropriate alternative tactics and technologies, gaining efficiency as we trade off flexibility for customized functional consistency. But before we get there – while we speculate on what we might need, explore the fuzzy front end of our development process, and seek convenient testing grounds for building out complex solutions – the Excel/VBA sandboxing environment will be regarded as terra firma for our work, a stable foundation for exploring the unknown. To this end, it is critical that we develop for ourselves a rich understanding of what is possible here – an understanding that surpasses that of the 95 percent of users who still think it's just a nice place to calculate sums. When individuals uncover the full range of control they can have here, it changes the way they confront all other challenges. I can ask no more but to help them along the way.

Best wishes in all that you do.

Dr. Elliot Bendoly, PhD



# **Section 1**

## Getting Oriented



# 1

## Necessary Foundations for Decision Support

**Objective:** *Appreciate the wide range of intentions, focal methods, and general characteristics of effective decision support tools.*

We make decisions all the time. Some decisions may appear routine: What shirt to wear? What to have for lunch? Should I purchase this online or go to the store? Should I respond to a post that I feel knowledgeable about? Others are more complex: Should I recommend that my client invest in a particular firm? Should I offer to take on additional work? Should my firm build or buy a new technology to help with its processes? Should I recommend a settlement in a lawsuit?

With complex questions comes the need for complex consideration. Sometimes we need help with these considerations. The sources of help can vary, but increasingly these sources tend to have two things in common: facilitated analytical strength and facilitated access – access to both underlying data and processes as well as to means by which to convey results to others. These sources of assistance often take the form of prepackaged off-the-shelf software tools. However, they can also be uniquely customized, and more and more frequently, this customization is being developed by individual users.

### 1.1 Intentions and Approaches

We can begin to understand the universe of tools designed to assist with decision making by listing their potential benefits. A few of the more common appear in Figure 1.1. For many managers, developers, and analysts, only a few of these sample benefits will prove essential. For others, the full complement of potential benefits must be considered along with others. However, the omission of one of these benefits should not be viewed as a failure. Similarly, not all efforts to try to squeeze each of these into a single tool are justifiable. Like all things, there are pros and cons to design

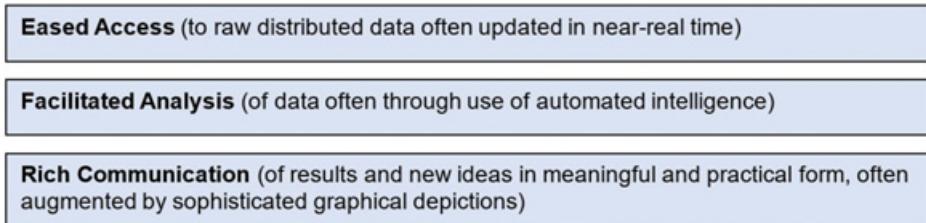
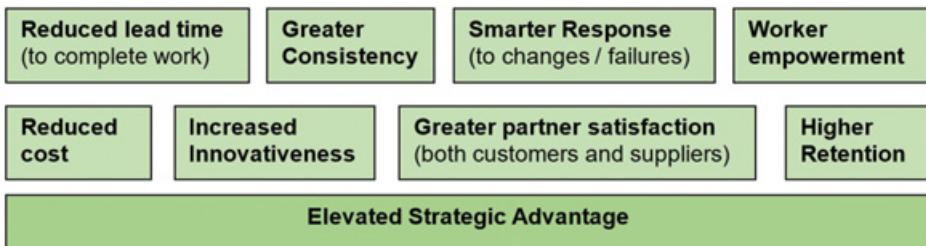
*Typical Attributes**Common Benefits*

Figure 1.1 Common elements of DSS

complexity. Tools that are not sufficiently straightforward won't get used. Tools that include too many features may not see timely completion; and since an excess of features can detract from users' ability to find key features or to leverage principal functionality, again such tools might not get used. This defeats the core purpose of Decision Support Systems (DSS): to support decision making.

So, let's back up a second and think somewhat differently about this. The design of the fundamental attributes that are typical of all DSS, facilitated analytics and access, must be deliberately aligned to desired benefits in order for those benefits to arise. And, if we need to be somewhat cautious, pragmatically, regarding the variety of benefits we are shooting for in a DSS design, then we can and should be specific about these attributes as well.

That shouldn't be a surprise to anyone. If you have a specific objective to accomplish, you do well to use the most appropriate approaches. The same general rule applies to all the forms of analysis that are facilitated by a DSS. Broadly, for example, we tend to delineate between descriptive, predictive, and prescriptive analytics, each and/or all of which might support a particular objective benefit in decision support. Descriptive work can provide details of the rich landscape in which decisions are made, even if the possible interconnections within that landscape have not yet been verified. Statistically we can describe measures of centrality such as means and medians, measures of variation and its possible skew. Computationally we can describe the multi-dimensional landscape visually, we can filter, or we can aggregate and

explore it interactively. Descriptive approaches built into tools can go a very long way in supporting decision making, though they may not go far enough.

Taking this one level further, our needs may require us to develop tools that are predictive, attempting to justify interconnections we may have begun to suspect through our descriptive statistical and computational efforts. Once again, we might approach this task using established statistical analyses to verify significant relationships between variables in our data, or we could leverage more computationally intensive and fluid methods such as stepwise estimation, variations on network analysis, or machine learning. A level yet further would involve prescriptive analytics to attempt to determine closed-form statistical expected value maximizing decision sets, or computationally searching for optimal or theoretically near-optimal ones.

How we pursue any one of these efforts, from data description through prescription, is dependent on the decision-making context in question and, once again, the objective benefits targeted. If we are able to begin with a fairly clear vision of what we need to achieve, or are at least willing to return to reevaluate this vision, we are going to be far more likely to make the right choices at each step of this journey.

## 1.2 Stages in Development

With this perspective in mind, we can also start to break down the various stages across which an integrated decision support system, potentially relying on all three levels of analysis, might be developed. Figure 1.2 provides a general map of these stages in a complete process that recognizes that forward progress can easily lead to a return to earlier stages for reconsideration. For a related decision tree depiction developed for the International Institute for Analytics see [www.excel-blackbelt.com](http://www.excel-blackbelt.com).

Across these stages we can envision a host of opportunities in which to use any number of smaller tools, functions, and subroutines to help us get the job done. Some of these tactics are executed once and seldom returned to. Others are functions that respond dynamically to new information and changes in other aspects of our approaches. Some are point and click, others require a bit more typing and organizing, while still others involve some serious research and not a small amount of trial-and-error to yield results. As we will see in the chapters of this book, there is a great deal that we learn just by trying; we gain a much deeper understanding of our options for future development, as well as pitfalls we might sidestep.

Among these stages we also find numerous opportunities for visualization, encompassing a range of analytical tactics, to contribute value. Indeed, the application of visualization in data analysis is critical to the development of decision support systems. It grants individual developers a view into what

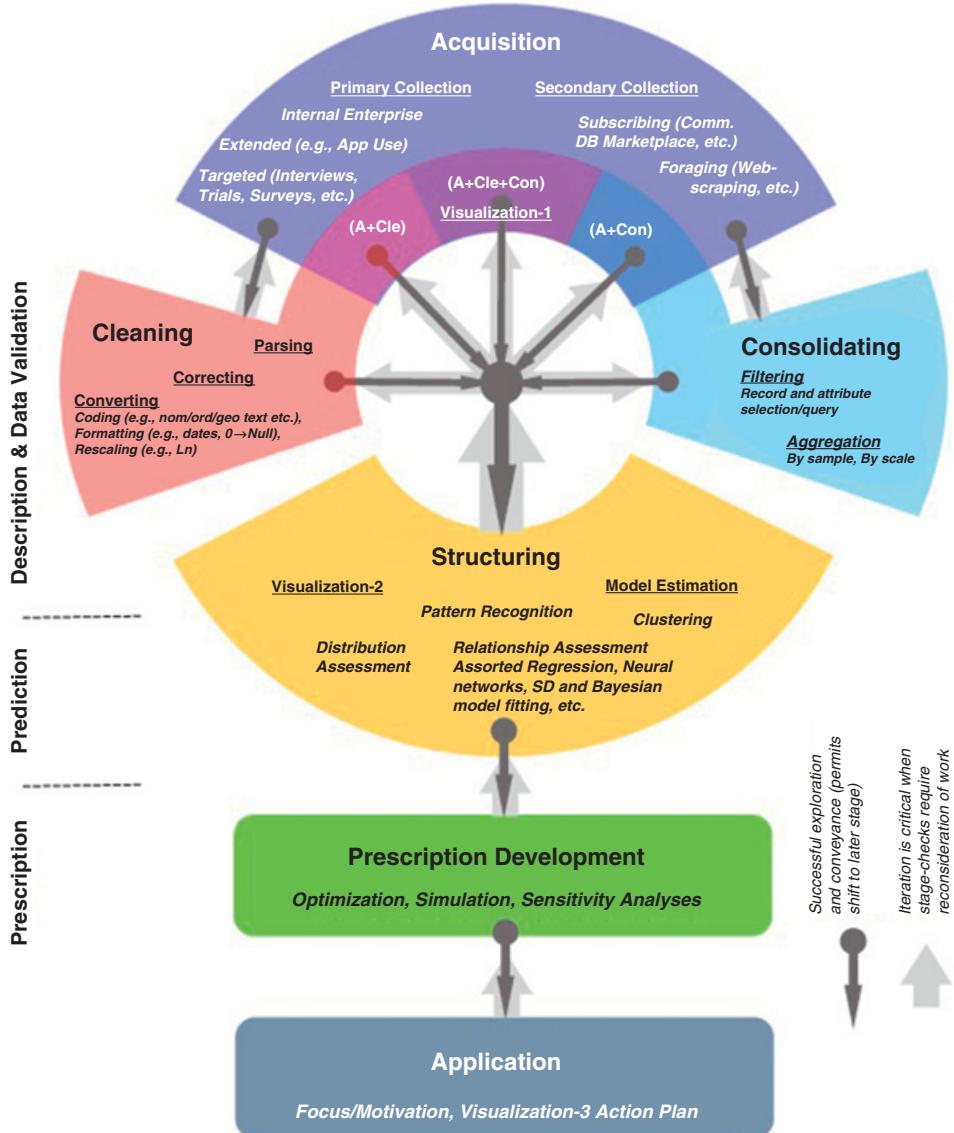


Figure 1.2 Stages in DSS development

opportunities and flaws exist in their work and assumptions, but it also provides the means to translate descriptions, predictions, and prescriptions into meaningful messages for various audiences served by these tools and their results. By doing so, it supports feedback that can further enhance the DSS itself.

Consider the following key principles described in *Beautiful Evidence* by Edward Tufte, a recognized scholar on data and relational visualization:

- 1) *Enforce Wise Visual Comparisons*: Comparison is a critical element in the development of understanding regarding the features of and anomalies within data.

Therefore, it is also critical in the practical application of findings from analysis. Comparison allows for the illustration of practical relevance of effects and decisions that may give rise to them. Wise visual comparisons are the mechanisms by which analytical and theoretical findings are vetted by real-world experience. They encourage faith in the analyst, and hence in most systems, as well as in the framework that the analyst is recommending.

- 2) *Show Cause:* Most experienced and practical researchers cringe when individuals confuse mere statistical correlation with causality. These assertions often lack evidence. However, in some cases certain reasoned explanations can be convincing. The task of the developer is to provide his or her reader with enough temporal and situational information for causal suggestions, either stated or implied, to be clear. The developer should also be able to facilitate scrutiny of data and analysis where appropriate. Tools should ultimately allow audiences to draw their own intelligently structured conclusions regarding causality rather than portray the perspectives of developers as unassailable.
- 3) *The World We Seek to Understand Is Multivariate, as Our Displays Should Be (AND as our analysis that supports them should be):* This is the classic story of the blind men and the elephant. Considering one dimension of data gives only limited and potentially flawed understanding of the big picture. However, not all pieces of information are useful. A little bit of rationality goes a long way in analysis. Hence one should push for multidimensional analysis and visualization to satisfy the needs of sufficiency, while being wary of distraction.
- 4) *Completely Integrate Words, Numbers, and Images:* Inconsistency between prose, data, and graphics weakens arguments. At the very least it confuses, at worst it misinforms. Analysts need to view graphical, numerical, and textual content as reinforcement mechanisms complementing each other. When alternate perspectives can be presented, don't confound these distinctions with differences in conveyance. Staying true here is another key to generating faith in the analyst and his or her tool.
- 5) *Most of What Happens in Design Depends upon the Quality, Relevance, and Integrity of Content:* Garbage fed into a graph results, at best, in beautiful garbage (but it still stinks). Know your audience. Understand the practice. Be fully aware of what needs to be analyzed and do it the right way. Mistakes and irrelevant detail can weaken your work.

Regardless of the technical nature of decision support tools, a serious consideration of these principles during development helps guarantee future use. While we will not devote a great deal of time to the psychological responses individuals have to certain visual forms, and hence best practices in visual designs, we will examine a range of technical approaches and continue to reiterate the importance of thought put into their selection and use. A more in-depth discussion of best practices in visual designs, backed by psychological theory and recent research, can be found in *Visual Analytics for Management* (2017).

Aside from these recommendations, it is essential to emphasize another point here, particularly for those new to DSS development: Decision support systems refer to applications that are designed to *support*, not *replace*, decision making. Unfortunately, DSS users (and developers) too often forget this concept, or the users simply equate the notion of intelligent support of human decision making with automated decision making. Not only does that miss the point of the application development, but it also sets up a sequence of potentially disruptive outcomes. These include excessive anthropomorphism, poor or impractical decisions, disastrous results, and the scapegoating of IT technicians. *It's easy for decision makers to view decision support systems as remedies for difficult work, particularly if they can blame others when things don't work.*

Although it is often difficult to codify, there is an implied contract between those who claim to deliver intelligent tools and those who accept their use: namely, an agreement that the analyst will not attempt to intentionally mislead the user. Ultimately, these issues contribute to the accountability and ethics of organizations, as well as the personal accountability of those developing the applications. If you want to develop a strong decision support tool, you have to identify your desk as where the buck stops. But if you want to be able to share the tool, you have to pay attention to how your applications might be used by others. It is critical here to guide built-in assistance in analysis, clear visualization of how characteristics of problems and solutions relate, and formally structured interfaces that deter, if not prohibit, misuse. It can be difficult to distinguish between those who intend to use their positions dutifully, but fail in execution, from those who will use their positions to deter others. If you set out to provide support for the right reasons, make sure you provide that support the right way.

### 1.3 Overview and Resources

Unlike many texts on related topics, this book is prepared for both the developer and the ultimate user. When reading these chapters, readers will learn what to expect from DSS and from those already using and building DSS tools. Fundamentally, it should become clear that the creation of decision support tools is something that even individuals with very little programming skill (or interest) can accomplish. That having been said, it will also be made clear that a little more programming acumen can help you accomplish a great deal more. This book works to drive home these points and to emphasize that individuals with just a bit of DSS development experience under their belts become popular very fast. They serve as bridges between career programmers who can build highly sophisticated resources, and the managers who face the reality in which only certain specific designs are going to prove useful.

How do we get there? We will need to acquire some experience in the various stages of development outlined in Figure 1.2, and we will need to do that in a way that is accessible (there's that word again) to essentially anyone. Ideally, we'll use a platform that is also still a backbone resource at most organizations, even if it's one that is often misperceived by those who know little about or overgeneralize its potential. We have that in the highly ubiquitous, deceptively familiar Microsoft Excel environment. Complementing this environment in our journey will be the use of associated tools, add-ins, code development, and affiliated applications such as the Microsoft BI suite. We will also discuss connections to other software environments such as those provided by Google, Palisade, and Apple.

What is the value of constructing tools using resources that career programmers might not champion in the development of commercialized applications? The question practically answers itself: Learning. If you have no intention of becoming a career programmer but would just like to learn how to get decision support tools built, and wouldn't mind becoming a bit more influential along the way, you need a ramp. The environments we will be discussing, and the approaches used within them, provide that ramp. You'll have the ability to get your hands dirty in each stage of development, from soup to nuts. You'll gain a systems perspective of how systems are built and how they work. Much like acquiring logic skills through programming experience, these skills are highly transferable. Further, the proof-of-concepts that you end up building can provide talking points and launch pads for further development – positioning you squarely to consult on the most effective paths that career programmers should take.

To help us through discussions and exercises in development, this book uses simplified contexts from real-world practice to enrich instruction on decision support. For example, we will introduce the case of a restaurant struggling to reconsider how to redesign itself in the face of changes in preferences. The Lobo's Cantina case gives us an opportunity to examine data on which predictions are built and the foundation of optimal policy and resource allocation decisions are made. How much space should be allocated to certain kinds of customer seating in the rebuild? How should we manage overbooking policies, given observed uncertainties in demand and human behavior? What kinds of risk profiles do our options in these decision-making contexts reveal, and how do we ultimately select the options to go with?

In another case to which we will return at multiple points in the book, we will consider the decisions of a firm trying to organize the shipping and transshipment of goods across national boundaries. Once again we start with discussions what we might do with data acquired through the firm's experiences. How can we describe visually the complexity of the tasks at hand? How can predictions in this context inform prescriptions? How do we

make choices in the presence of expected levels of risk, where are the safe bets, what are the next-best options, and how do we document and convey the anticipated returns and risks associated with the best options we settle on?

As for the structure of this book, we will generally follow the forward flow presented in Figure 1.2. This first section, titled “Getting Oriented,” focuses on easing people into the Excel workbook environment as a platform for tool development and visualization. Navigation and data acquisition are central themes, as are discussions of cleaning and consolidation tactics, along with critical caveats regarding some of these approaches. In this and other sections of the book we will consider some of the most valuable and often underused of Excel’s functions and tools. We will also introduce the Blackbelt Ribbon add-in and the additional capabilities it provides.

In the second section, “Structuring Intelligence,” we will begin to discuss what acquired data can tell us about the real-world contexts in which we must make our decisions. We will briefly consider how predictions of trends and relationships in data might be estimated but will pay special attention to the uncertainty that these predictions can imply. With this in mind we will describe the design and construction of simple simulations based on such real-world analysis. We will consider how to jointly leverage tools such as data tables in tandem with consolidation tactics such as Pivot Tables in the design of these assessments as well as the use of user-friendly controls to simplify the management of this work. The visualization of available data and the by-products of simulation will then be discussed. The focus will be on specific tools for the workbook and related environments, with reference to best practices in design outlined in texts such as *Visual Analytics for Management* (2017). Here we will also get some additional glimpses into the world of programming.

In the third section, “Prescription Development,” we forge ahead into discussions of the construction of mathematical and functional architectures for arriving at specific recommendations in decision making. We focus on available tools but also push beyond common experience with these tools. We will discuss how we might peek behind the optimization approaches used by tools such as Solver, as well as approaches made available by Palisade in their @RISK toolset. We will discuss limitations of various approaches, the integration of optimization and simulation tactics to seek out both high expected value and low-risk options, and critically consider the trade-offs between search time, search progress, and the practical value of benefits gained.

In the fourth section, “Advanced Automation and Interfacing,” we dive more deeply into development work that could not easily be conducted in the spreadsheet environment alone. Instead, we will turn our focus to the Visual

Basic for Applications (VBA) developer. I will give additional examples of macro editing, the creation of new user-defined functions, application calls, integrated automation, and advanced interface development with associated illustrations available at [www.excel-blackbelt.com](http://www.excel-blackbelt.com)). At the end of this section, you will have gained sufficient understanding of what is immediately possible to develop through simple programming tactics for your own work and, no less importantly, how to delegate that work to others.

On this last point, and as a reminder to the reader, this book is not designed for an advanced programming course. Nor is it a statistics text or a single-source dictionary defining all things Excel. This is a guide for professionals who want to utilize their own skills and need only the right coaching, inspiration, and reinforcement to demonstrate these skills. The content is designed not to inundate but rather to illuminate. Because readers come to this book at various skill levels, you should feel free to pick and choose among the chapters. Even those looking simply for novel tactics will find value in these pages. However, the real hope is that this book will open the world of DSS development to a broad community. Everyone deserves to know how accessible DSS design can be and the potential that it holds. It's time to shatter the wall between the programmer and the management professional, and the confluence of those skills can start here.

# 2

## The Common Development Environment

**Objective:** *Become acquainted with the most common and fundamental capabilities of the spreadsheet environment, the limitations of these capabilities, and opportunities to leverage them in combination.*

Before attempting to construct a home, architects and builders need to know what resources they have at their disposal and how they can access these resources. The same holds in decision support development. But as you'll learn in subsequent chapters, the tools available to Excel developers are more numerous than a typical user would imagine. In this chapter, we'll start with the low-hanging fruit by becoming acquainted with the general nature of Excel's front-end development environment. Figure 2.1 provides an annotated view of what people typically see when they open a new file in Excel. Only a few key elements of this interface are central to our initial discussion.

Excel files are called "workbooks." They contain any number of worksheets (spreadsheets) that can be used in tandem. Most users relegate the Excel application to the role of an information storage tool. This is largely because most users don't know what else Excel can do. Not that storing information in Excel is bad but there are often better alternatives available for storage, such as databases, in the case of very large sets of data. Functionally, the storage capability of Excel represents only the bare tip of this technological iceberg.

Regardless, knowing how the cell structure in a spreadsheet works is a good place to start and this knowledge is essential to further discussion.

Cells in spreadsheets can contain:

- **Fixed data:** Data you (or others) have entered, numerical or otherwise.
- **Formatting:** Background color, border thickness, font type, and so on.
- **Names:** References other than the standard column-row reference.
- **Comments:** Notes regarding the contents.
- **Formulae:** Mathematical or statistical, text-based, or a range of other types such as logic-based or search oriented.

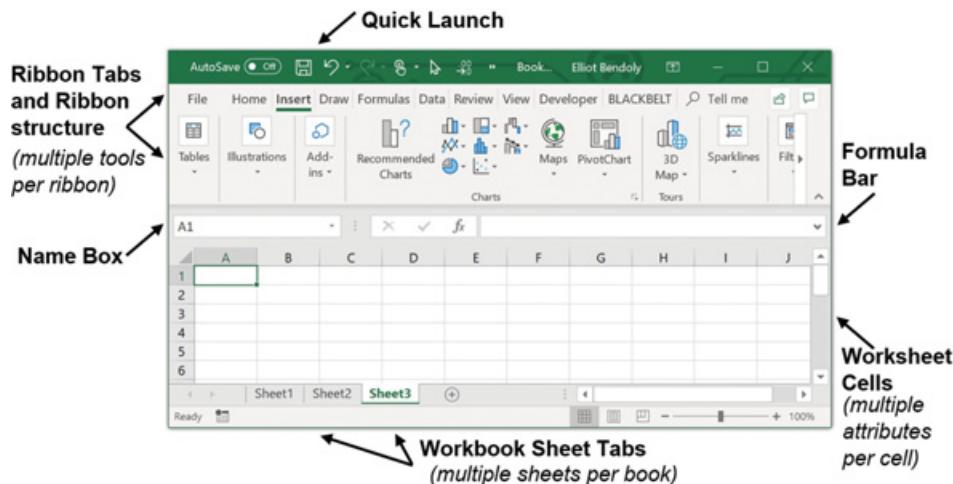


Figure 2.1 Basic front-end elements of the Excel environment

- **Live data links:** Data that are drawn from an externally linked source, such as a database or the Internet.

## 2.1 Entering Data and Simple Extrapolation

Data entry in Excel is about as basic as it gets – just click and type. And it's probably the only thing that people universally know how to do in Excel. But Excel makes some kinds of data entry easier in ways that many current users don't realize. For example, Excel has an automatic pattern-recognition element that, by default, attempts to aid you in filling in additional cells of data. Although it's not always successful, it's often convenient. Let's say I want to enter the number 1 into five cells in the first row. I start by typing the number 1 into cell A1. This is shown in Figure 2.2.

Note that when cell A1 is selected, its border is bold and a small square appears at the bottom right of the cell. That small square is the copy prompt. If I pull that square either to the right or down, Excel will attempt to fill in all other cells I highlight with the pattern – in this case, the number 1. By pulling the square to the right and highlighting the next four cells, I get the results shown in Figure 2.3.

At this point, with all five cells selected, I could pull the copy prompt down two rows and get the spreadsheet shown in Figure 2.4. Ascending values and alternating text values are also recognized. For example, take a look at Figure 2.5.

Or consider the copy shown in Figure 2.6.

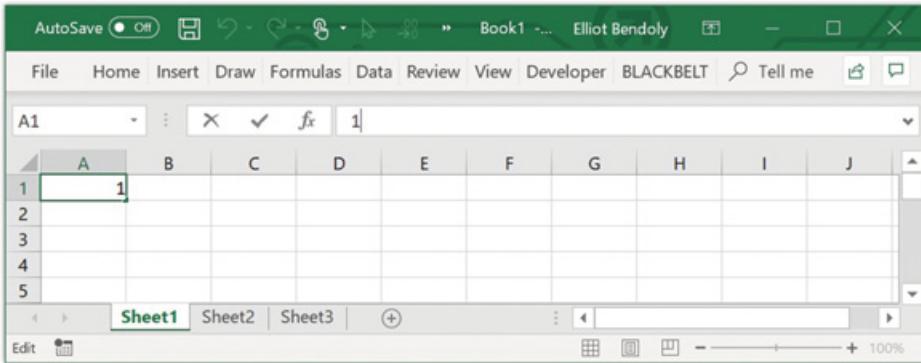


Figure 2.2 Initial entry

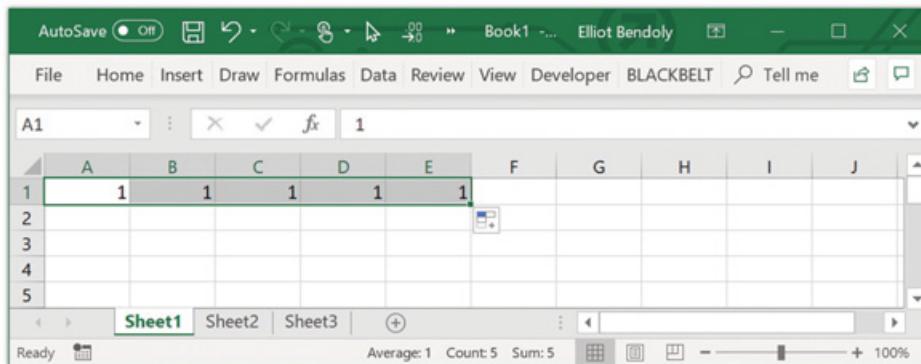


Figure 2.3 After copying across

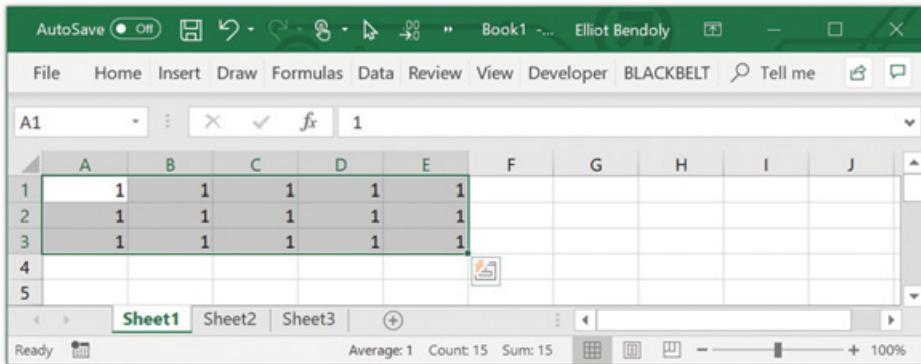


Figure 2.4 After copying the row down

Although this may not be exactly what we wanted (the alphabet isn't extrapolated here), we are provided with a potentially meaningful sequence based on Excel's existing pattern recognition. Some results of

pattern recognition by Excel are less intuitive, however. For example, take a look at Figure 2.7.

Here, as in the previous examples, Excel is trying to figure out what pattern the user is trying to specify. However, not every pattern that seems natural to us is encoded in its rules and other mathematical rules may have automatic priority in Excel. In this case, priority is given to extrapolation based on best-fit line estimation. Excel has considerable intelligence built into it, so this shouldn't be that much of a surprise.



Figure 2.5 Initial sequential entry, followed by copy



Figure 2.6 Initial text sequential entry, followed by copy

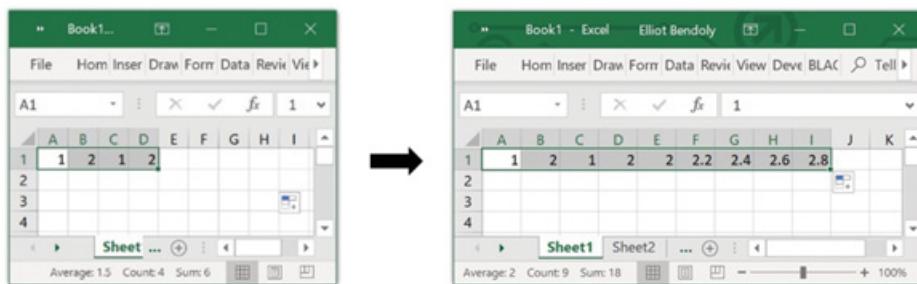


Figure 2.7 Initial switching sequence, followed by copy

When the application is given numbers, we should expect it to do math. If we want Excel to do something else because we need something else, we may need to inform Excel differently.

There are several simple ways to avoid ending up with an Excel extrapolated pattern that doesn't fit a user's need. One method is to completely avoid relying on pattern recognition and instead use a formula that generates the pattern you want. For example, type “=A1” into cell E1, press Enter, and use the copy prompt on E1 to pull that entry into all other cells in that row. This formula essentially duplicates the pattern in cells A1–D1 for as long as you want it repeated.

If you would prefer to rely on Excel's pattern-recognition mechanisms, you could also try entering your numbers as text. Excel's options for intelligently identifying and extending text patterns are more limited and this might generate unexpected results. To enter numbers as text in any given cell, precede the number with an apostrophe, such as ‘1 or ‘2. This will ensure that Excel interprets the entry as text – at least as far as pattern recognition is concerned. Within the spreadsheet, Excel will still let you perform mathematical functions using the numbers following that apostrophe but this is an added step that may create other difficulties in formatting and contribute to more advanced use of data down the line, so it isn't an approach that's often used.

Still another mechanism to augment existing pattern-recognition capabilities is available through the Edit Custom Lists feature. A Custom List is an ordered sequence of text entries recognized by Excel and available for series extension in worksheets. If you enter a value from an existing Custom List (e.g., “June”), extension with the copy prompt will extract subsequent entries (“July,” “August,” etc.). In Excel 2007, this would have been found under “Options” accessed through the Office button. In more contemporary versions of Excel, these options are found under the File tab in the upper left-hand corner. An overview of Options access is presented in Figure 2.8.

Regardless of the version of Excel, clicking on the Edit Custom Lists button (under Advanced, General) will open the Custom Lists dialog box, as seen in Figure 2.9. This functionality allows you to view the existing custom lists, generate new lists, and import other lists that aren't currently recognized by the workbook. Any custom list that you create will be stored on a file on your computer, so that each time you return to Excel, that list will be recognized. You will have taught Excel your first of many lessons. You can also edit or delete that list through this same interface.

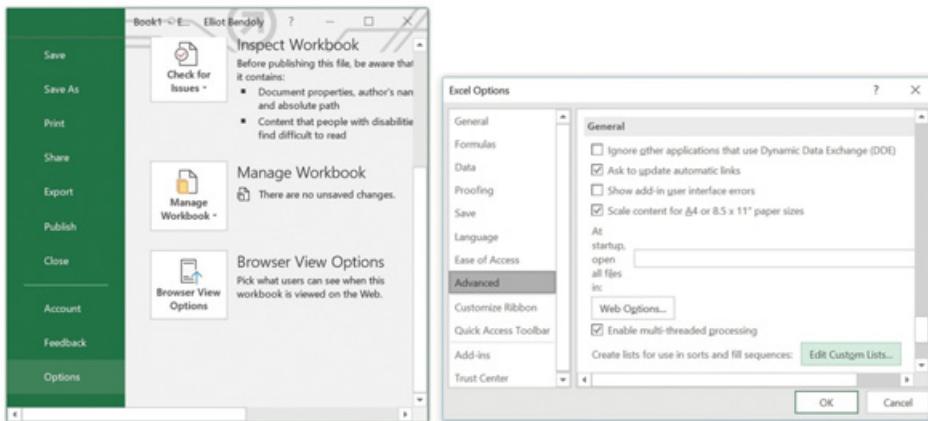


Figure 2.8 Excel Options access and access to Edit Custom Lists

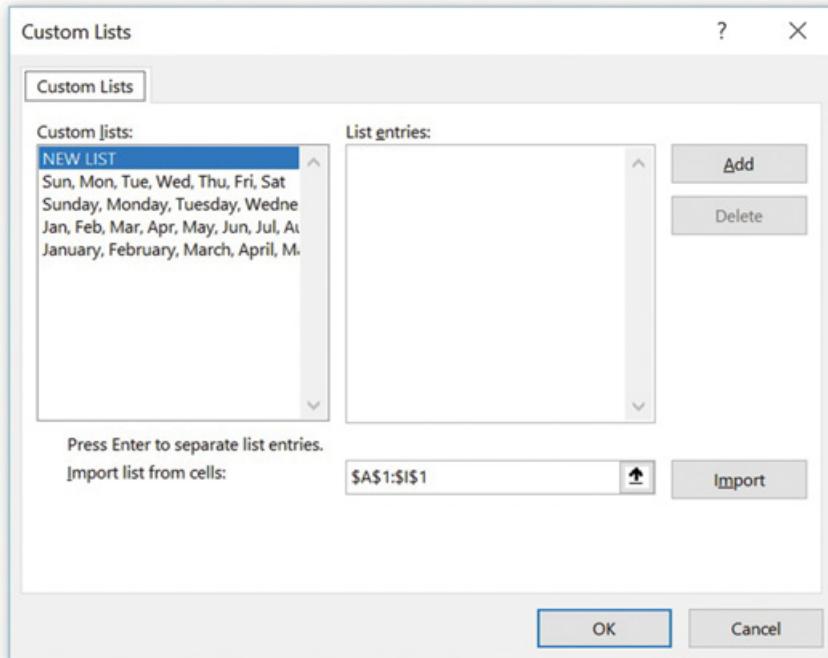


Figure 2.9 Edit Custom Lists interface

## 2.2 Format and Attributes of Cells

While cells possess a wide range of features of their own, apart from their contents, being familiar with even a handful will get you pretty far. We'll begin with a focus on formatting, naming, comment annotation, and work with hyperlinks before moving on to the use of functions to drive dynamic content.

### **2.2.1 Static Formatting for Cells**

Static formats for individual cells, as well as ranges of multiple cells, can be modified in a variety of ways. Access to formatting options is gained by either right-clicking on the desired cell (range) and then selecting Format Cells in the shortcut window that appears (as shown in Figure 2.10), or by selecting Home>Format>Format Cells from the standard toolbar. The Format Cells dialog box becomes available by either approach. The formatting options in this dialog box include the following:

- **Number:** Allows modification of the type of numeric or text presentation for that cell. For example, this allows numbers that represent percentages to be presented as such (i.e., with a % sign), numbers that represent dates to be presented in month-day format, and very large or very small numbers to be presented in scientific notation (e.g., 3.45E8 instead of 345,000,000, or 3.45E-8 instead of 0.000000345), among other options.

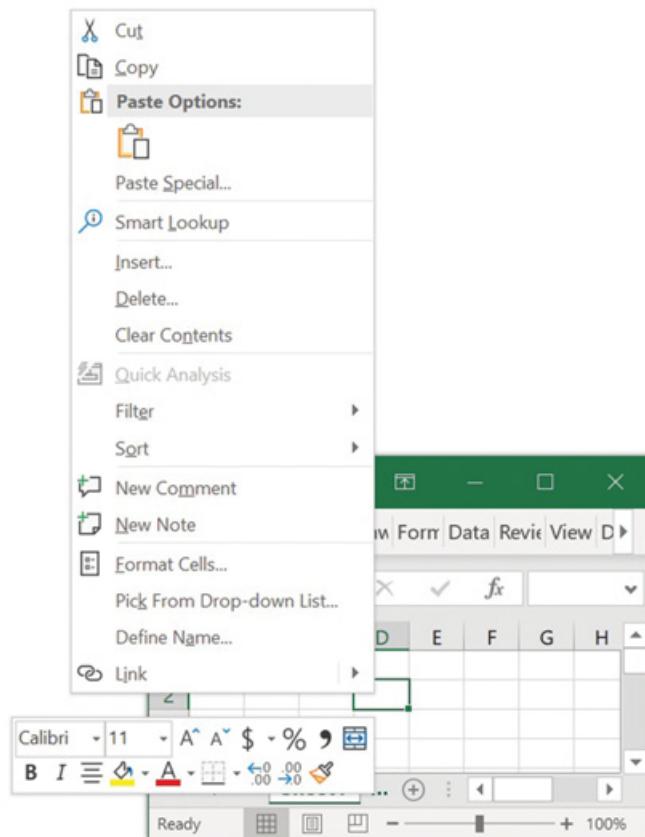


Figure 2.10 Accessing cell-formatting options

- **Alignment:** Allows changes to the horizontal and vertical alignment of contents within cells as well as whether cells will merge with neighboring cells and whether text within a cell will run outside the boundary of the cell or will wrap (as it would in a paragraph in Microsoft Word) based on the boundary of the cell.
- **Font:** Self-explanatory. Pick a font, any font, not to mention color, bold, italics, underline, and so on.
- **Border:** Allows changes in the appearance of the cell boundary. Line thickness, color, and line type are options.
- **Fill:** Allows changes in the appearance of the interior background of the cell. Pattern selection (e.g., striped or hatched) and color gradients are options, though not necessarily encouraged.
- **Protections:** When accompanied by sheet security options, prevents unauthorized users from modifying the contents or other attributes of the cell (more on this in Chapter 9).

A variety of automatic table formatting options can also make your static formatting tasks faster, provided that you are happy with the choices available. These “Format as Table” options are also found in the Home ribbon and can offer some very helpful shortcuts for setting up headers and row delineations.

### 2.2.2 Conditional Formatting

Unlike static formatting, conditional formatting offers a more dynamic approach to highlighting the contents of cells. Any cell, or set of cells, subject to conditional formatting will take on a special appearance only under specified circumstances. In the current, standard versions of Excel, each cell can have multiple conditional formatting rules associated with it in addition to the default cell format.

Click the Conditional Formatting button in the Home tab. A drop-down menu appears with a list of conditional formatting options (shown in Figure 2.11). From here you can select which type of conditional formatting you want to apply.

If you select the Manage Rules option, the Conditional Formatting Rules Manager opens and you can add, edit, and delete rules from the same window. Basically, this dialog box allows you to spell out rules or conditions and to specify fonts, borders, and patterns to apply when those rules/conditions are met. In the example shown in Figure 2.12, cells subjected to the conditional formatting rule will take on one colored background pattern when their values are less than 0.4, whereas those with values greater than 0.6 will take on an alternate pattern.

Occasionally, the focus of formatting is not the content of the cell you want to format but rather of some other cell in which data exist and potentially change. Figure 2.13 provides an example of a cell (say B3) that has been formatted conditionally on the values contained in cells A3 and A4. If the

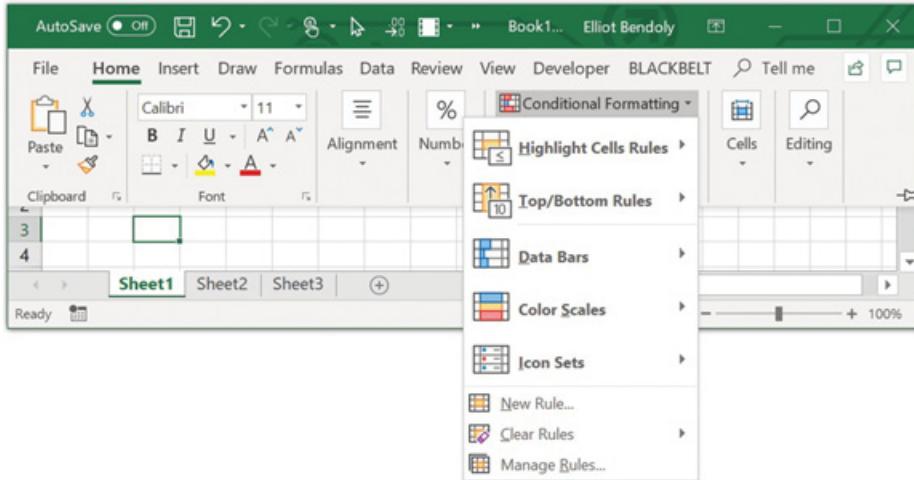


Figure 2.11 Accessing conditional formatting

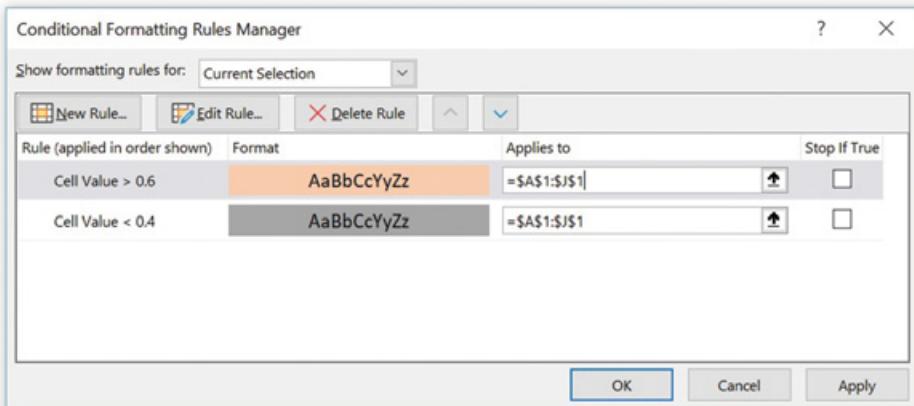


Figure 2.12 Developing conditional formatting rules

equality in the parentheses depicted in the first rule's formula ( $\$A\$4=1$ ) is TRUE, then B3 will take on the first rule's formatting (regardless of what value it contains at this point). If the equality is FALSE, then the second rule's formula is checked and the formatting is applied, if needed.

Any other default, such as static formatting, will apply to cells that don't meet either criterion. In this case, the static format of these cells involves no background shading of any kind, so cells with values between 0.4 and 0.6 are unshaded. Furthermore, the rules that have been applied obey a strict hierarchy. If the top rule is met, any later rules that would override that formatting will not play out. If the order of the hierarchy needs changing (e.g., the "red" rule should have precedence over the "blue" rule), the

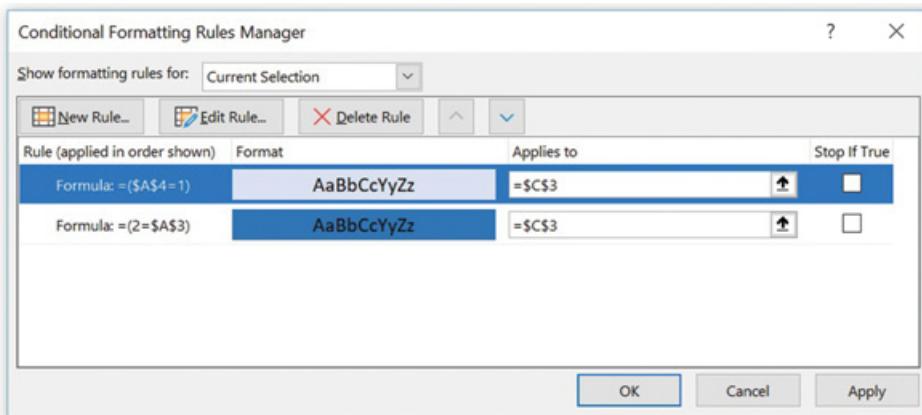


Figure 2.13 Conditioning on the value of OTHER cells

order can be changed by selecting a rule and then clicking on the up or down arrow buttons available in the form.

There are a few points to remember about formatting multiple cells:

- All static formatting and conditional formatting actions can be applied to multiple cells simultaneously by selecting a group of cells before beginning any of the previously mentioned procedures. A group of adjacent cells can be selected by selecting a cell at a corner of the range desired, holding down the mouse button, and highlighting all other cells in the adjacent range. A group of nonadjacent cells can be selected by holding down the Control (Ctrl) key and then clicking on each of the cells in the desired group. When the cells are selected, the same access is available to both static and conditional formatting windows.
- A number of special multicell conditional formatting options exist that format cells according to their relative magnitude. These include data bars, color scales, and icon sets. Upon selecting a range of cells for formatting, any of these options can be selected to portray a rich visual representation of these cell values. Just be cautious about your choices here. Some of these options, particularly among the icon sets, can prove misleading to audiences.
- If you've already formatted a single cell in a particular way and would like to replicate that format in other cells, the Format Painter is a handy tool for copying that format to new cells. You can access the Format Painter by clicking the paintbrush icon in the Home toolbar. Select the cell with the appropriate format you want to copy, click the Format Painter icon, and select the set of additional cells to which you would like to apply your format.

### 2.2.3 Naming Cells and Multicell Ranges

Customized, unambiguous naming provides a meaningful supplemental way of referencing cells and other objects in our common development

environment. For example, a cell that is consistently used to contain estimated shipping cost data might more meaningfully be named `ShippingCost` as opposed to `G4`. If cell `H13` contains the rate of return on an investment, a more meaningful name for this cell might be `RateofReturn`.

Unambiguously named cells can be helpful for at least a couple of reasons. First, other users can more easily understand what the cell contains. For example, when calculating a formula, it's beneficial for others to know what terms you are referring to in the calculation. Excel makes it easy to visually associate terms with the data they represent by highlighting cells on a spreadsheet when cell formulae are being edited. A second reason for clear naming is that it gives you a better shot at recalling what you were thinking after you built a tool (when you may want to modify that tool).

Names can be assigned to cells by selecting the desired cell and then entering a new name for that cell in the name box, which is usually located near the upper-left corner of the spreadsheet. When unnamed cells are initially selected, their column-row reference will appear in that name box. Click the name box, enter the new name for the cell, and then press Enter. The new name for the cell, for example, `Marketplace`, will display in the name box (Figure 2.14).

The cell may now be referenced by either its column-row designation or its new name. More conveniently, however, if at any point the developer wants to move the location of that cell and its contents, the new name will come along with it. This can be very helpful in avoiding confusion when other cells or applications depend on being able to locate the cell's information after such a move.

If a typographical error is made in naming, or if the developer later wants to change the name of the cell, the most secure route to correct this issue is to select Name Manager from the Formulas menu to open the Name Manager dialog box. This dialog box enables the user to delete the undesired names

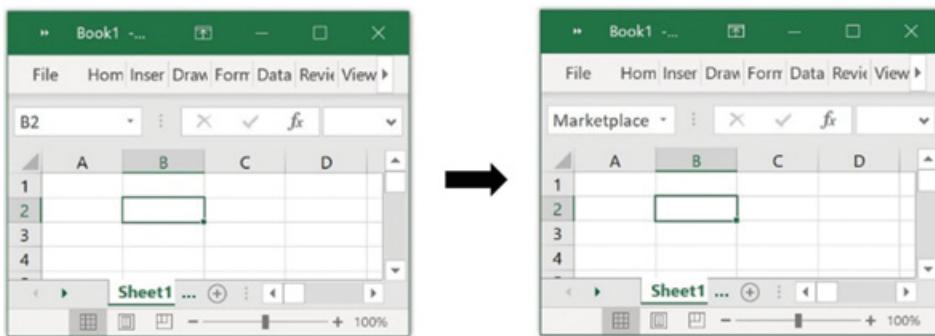


Figure 2.14 Making use of names to reference cells

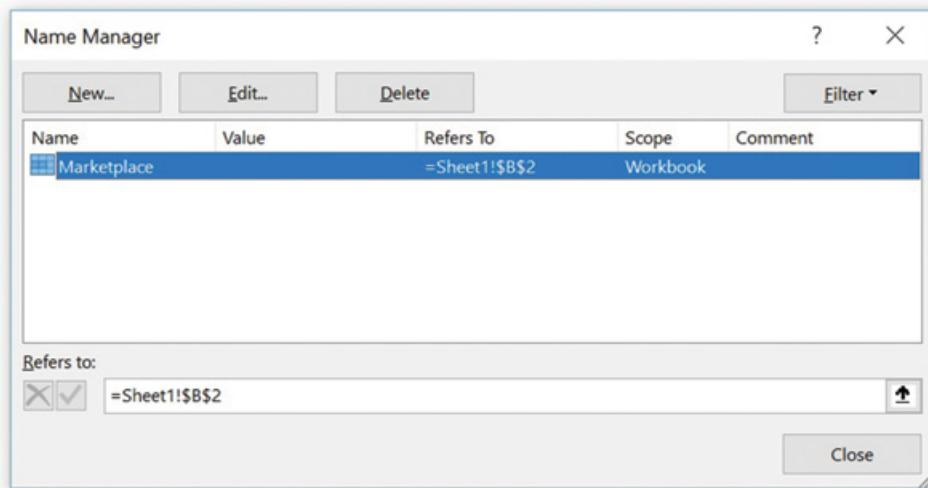


Figure 2.15 Managing names assigned to elements in a workbook

and add other names. All existing names in a workbook can be modified from the Name Manager (Figure 2.15).

As with formatting, multiple cells can be selected simultaneously and assigned a specific name. For example, if cells A1:A30 reference information on the profitability of ten leading firms, the cells can be named Top10Companies. The use of range names becomes more meaningful in advanced applications, but they remain extremely useful in helping others understand the design of a developer's tool.

#### 2.2.4 Worksheet and Other Object Names

Referring back to the idea that Excel documents are really workbooks that contain multiple worksheets, each worksheet has a name that can be renamed as well. Changing the name of a worksheet is extremely straightforward. Just double-click the current tab name corresponding to the worksheet of choice (e.g., Sheet1 as shown in Figure 2.16) and type in a new name for that sheet (e.g., Raw Data).

Objects include items such as drawn shapes (e.g., circles), controls (e.g., option buttons), charts (e.g., a bar graph), inserted media (e.g., .wav files), and other embedded tools. Each of these items can be assigned names for reference (Figure 2.17). Names for cells and objects are universal across all worksheets in a specific workbook. In other words, if on Sheet1 you name a cell A1 PRICE, you can still refer to that cell as PRICE for anything done on Sheet2 in that workbook.

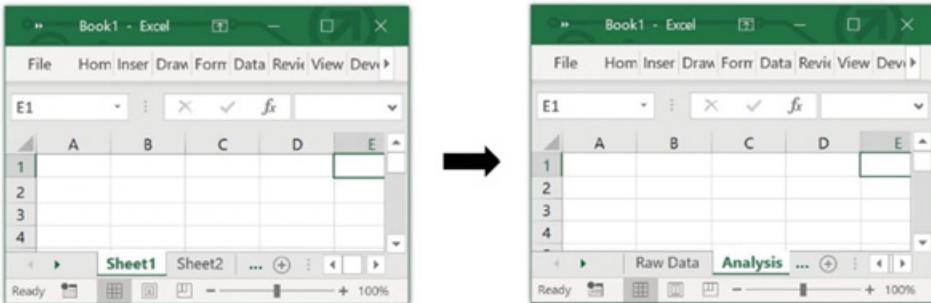


Figure 2.16 Modifying sheet names

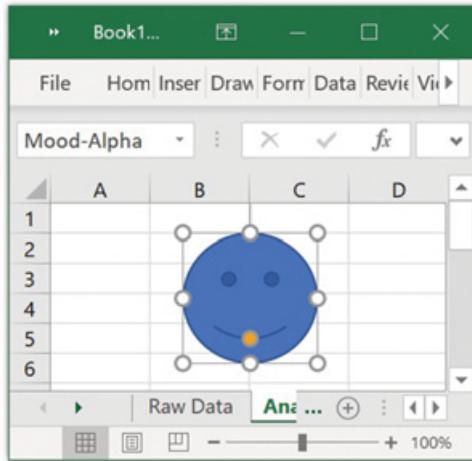


Figure 2.17 Assigning names to objects

### 2.2.5 Attaching Notes and Comments

Along with the naming of cells to provide better reference mechanisms, notes and comments, depending on which version of Excel you have access to, can be added to specific cells to add greater clarity when needed. For example, aside from naming a cell “Cost,” a developer might add the text shown in Figure 2.18 to appear in comment form when the cursor passes over the cell. Comments and/or notes are added to selected cells by right-clicking on the cell and then selecting Insert Comment from the shortcut menu. The developer will then be able to modify the text within the new comment bubble as well as manipulate the height and width of that bubble. The more contemporary comment capability offers a nice mechanism for tracking discussion between multiple users in shared workbooks.

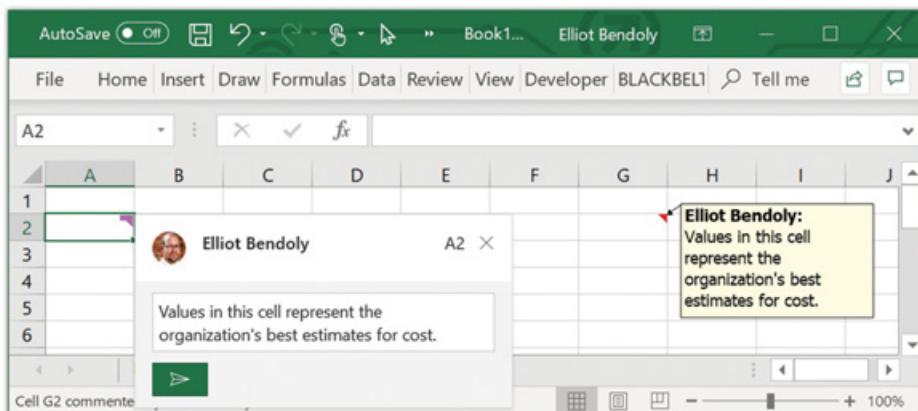


Figure 2.18 Application of a comment and a note

### 2.2.6 Working with Hyperlinks

Hyperlinks (links to Web pages and pages on local drives) can be embedded within cells. In their simplest form, they involve cell content used to house the text of the URL in question, for example. The option is available by either right-clicking and selecting Hyperlink from the shortcut menu, or by selecting a cell and then selecting Hyperlink from the Insert tab on the main menu. One might ask: Why waste a cell this way when hyperlinks can be assigned to noncell objects, such as drawn circles? Honestly, I used to ask the same question. However, there are at least a couple of reasons why. The first is the ability for other users to immediately view and, if need be, edit links presented in a tabular structure of a worksheet. The other has to do with content drawn in from alternate sources, which may have such links associated with content by default. In other words, as long as other sources retain hyperlinks in a manner that is most readily drawn into the cells of a spreadsheet, it is worth understanding how to make use of these cell-embedded hyperlinks (even if we wouldn't have designed things this way ourselves). We will talk more about the extraction of hyperlink content from cells later on in the text.

## 2.3 Functions (An Introduction)

Some formulae are basic, such as adding the contents of two cells, or dividing the content of one cell by that of another and then subtracting that of a third. The syntax of other functions can be less immediately obvious. Excel, with standard add-ins installed, has more than 600 functions already built in for use within cells. This is probably more than any one person will ever use and will not include some functions you might want to have (which you might end up developing yourself), but it's nice to have this set available as a starting point.

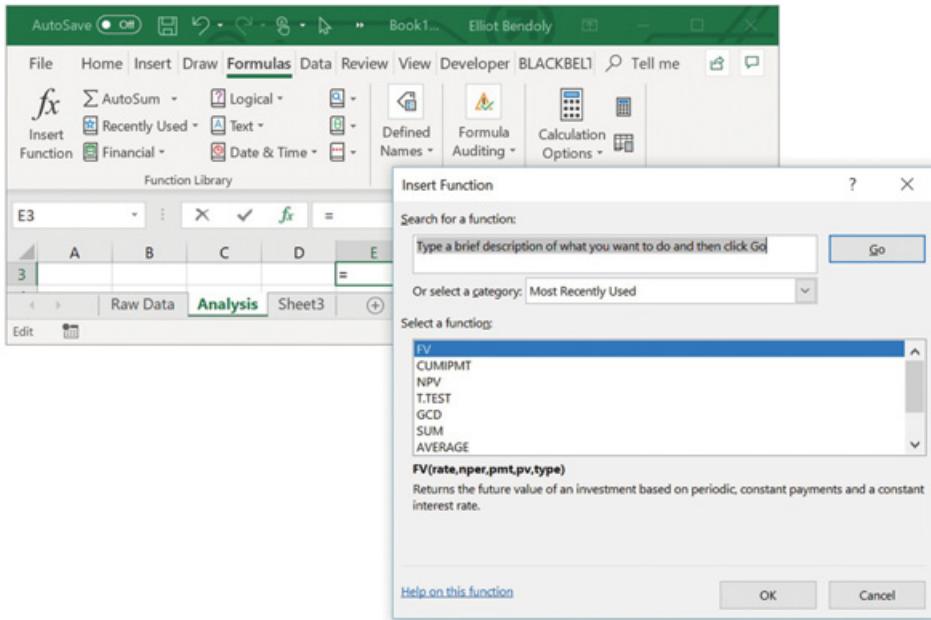


Figure 2.19 Selection of functions for use by category

After you select a cell in which to embed a built-in function, you can insert a function by following the Formulas>Insert Function menu path. The Insert Function dialog box (shown in Figure 2.19) will display.

The great thing about this dialog box is that the functions are organized into a set of approximately ten fairly intuitive categories, plus the all-encompassing All category and User Defined items. Even better, this dialog box gives you instructions on the types of inputs each function takes and what each function does with these inputs. The following is a list of some of the options available in the Insert Function dialog box:

- **Financial:** Anything from interest-accrual calculations to net present value (NPV) to yields on T-bills. Useful stuff (makes you wonder why you were ever forced to memorize any formulae in finance classes).
- **Date & Time:** Getting and working with current date and time representations.
- **Math & Trig:** Common calculations that come up in business models such as sums ( $\Sigma X$ ), products ( $\prod X$ ), factorials ( $X!$ ), exponentials ( $e^X$ ), and rounding. Other, more sophisticated calculations that greatly assist in large-scale data consolidations are also available. These include  $\text{SUMPRODUCT}(X,Y)$ , which allows an array of X values (i.e., vertical or horizontal range of cells, named or otherwise) to be multiplied by an array of Y values of the same dimension. Each component in X is multiplied by its corresponding value in Y and the sum of these products serves as the output.

- **Statistical:** Averages, counts, normal distribution (such as z-scores) calculations, quartiles, F-tests, and even things like the Poisson distribution (sort of, as we'll see in Chapter 4).
- **Database:** Not an impressive list but handy when interfacing Excel with databases. Provides averages, sums, and other summaries based on the contents of fields in databases.
- **Text:** Functions that allow you to merge text into a single string [such as `CONCATENATE("alien", "ate") = "alienate"`], determine the length of a string [such as `LEN("alienate") = 8`], extract a portion of text in a cell [such as `MID("alienate",2,4) = "lien"`], or simply find text within a large text [such as `FIND("nate", "alienate") = 5`, or `FIND("nation", "alienate") = #VALUE!`, which essentially represents an error because "nation" cannot be found anywhere in the text "alienate"]].
- **Information:** Provides information on the contents of cells, such as whether the number contained is odd, whether it's a number at all, or whether the cell contains an error as an output of the function within it (taking the square root of a negative number would provide the error term `#NUM`).
- **Engineering:** If you are a manager, this set is probably not that useful to you. At least not immediately, though as you broaden your developer skills you might be surprised. This collection includes options such as the translation of binary to hexadecimal notation (for use in computer science) and calculations with imaginary complex numbers (for use in physics).
- **Logical:** A short list but a critical one in decision support – especially the IF statement. IF can be applied to both numerical and text inquiries. It allows you to test whether the value in another cell is equal to, not equal to, or in some way related to (e.g., greater than) other values and it allows you to specify which calculation you want to be active in this cell under either condition (e.g., calculate something if TRUE or calculate something else if FALSE).
- **Lookup & Reference:** Some of the most useful (though underutilized) functions are in this category. Sifting through data can be frustrating for a manager but these functions make the task a lot easier, quicker, and potentially more accurate. INDIRECT is an example of a simple reference tool that allows for an alternative means of constructing cell range references and accessing information elsewhere in a workbook. Providing a cell reference or cell name in quotation marks as the argument of this function will return the value in the corresponding cell. For example, if cell A1 is named FirstCell and contains the value 12, the `INDIRECT("A"&"1")` or `INDIRECT("First"&"Cell")` will provide the value of 12 when called.

In your exploration of the functions available to Excel, you may find other sets available as well (Cube, Web, etc.). As noted, function availability can also be a result of application additions and your own efforts to construct new ones. You might be wondering, with all of these functions available – many of which I don't see myself using – why on Earth would someone want to make more?

Since I've just posed this question, it would be pretty frustrating if I didn't provide an answer. Fortunately, the answer is a simple one:

*Real-world analysis is complicated and all tools have their limitations.*

Now, let's back this up with some examples. While this chapter is designed as an overview of the workbook environment and not a discussion on working with data (which we will get to soon), it's useful to peek under the hood of these functions a bit. In this case I'm going to focus on two kinds of functions that are not only relevant to the spreadsheet work but also generally critical to work we will confront in the back-end coding environment. These functions have the all-important capability of helping us identify information and information cases on which much of our subsequent analysis and reporting is ultimately based. Identification tasks can also inform further data acquisition efforts (Figure 1.2), so this is not a bad place for a little critical examination.

### **2.3.1 *Lookup and Reference Function Examples***

Lookup and Reference functions are often used in conjunction with other functions in a workbook. For illustration, let's consider an example using VLOOKUP, MATCH, and OFFSET. In the workbook Chp2\_IdentitiesList we have a scenario involving a sampling of human resources data including employee names, job roles, IDs, and salaries. Headers to the main data are in row 6, with the main data in rows 7 and below. As in Figure 2.20, rows 2 and 4 contain formulaic queries to this data, with the formula content of the selected cell (B2) shown in the formula bar. In this cell we see that VLOOKUP is being used. It is also clear that certain cell ranges needed as inputs have been "named" and are referenced that way within the function. Conveniently, clicking on the function will outline the cell ranges used as inputs and coordinate color coding of those highlights with the formula bar text (Figure 2.20).

A critical point needs to be emphasized here. VLOOKUP requires three inputs: What to look for, What the full table range is (to look in), and Which column of that table data should be returned from. The assumption is always that the thing looked for is in the first column of that table. A fourth parameter is optional but it is a crucial one. It is presented with square brackets around it in the function bar. Here the value of TRUE or FALSE is expected (or 1 and 0, equivalently). FALSE indicates that you will only accept information if the thing you are looking for is found. If it isn't found, an error will be returned. TRUE indicates an inexact search, where the search stops as soon as the thing looked for appears to be exceeded. There are only very specific occasions when TRUE proves useful; your leftmost column will need to be sorted and you are going to have to be OK with the search ending

The screenshot shows a Microsoft Excel spreadsheet titled 'Chp2\_1...'. The formula bar displays '=VLOOKUP(name,studentinfo,2,FALSE)'. The main table has columns 'Selected Name', 'Role', 'Emp ID', and 'Salary'. Row 2 contains the formula '=VLOOKUP(name,studentinfo,2,FALSE)'. Row 3 is a header row with 'Next Name in List' and 'Role'. Row 4 contains 'Barnes, Melanie' and 'Group Manager'. Row 5 is blank. Row 6 is a header row with 'Name', 'Role', 'Emp ID', and 'Salary'. Rows 7 through 14 contain data: Flores, Jessica (Communications, 4451, \$ 85,360); Long, Ryan (Quality Manager, 3124, \$ 87,010); Hughes, Jackson (Customer Lead, 4303, \$ 95,810); Coleman, James (Audit, 1312, \$ 99,060); Ross, Logan (Lead Designer, 5043, \$ 102,870); Barnes, Melanie (Group Manager, 4123, \$ 113,550); Perry, Gabriel (Process Intern, 2342, \$ 117,740); Patterson, Kaitlyn (Supplier Relations, 3452, \$ 130,150). The bottom of the screen shows the ribbon tabs 'Raw Data' and 'Edit'.

	A	VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])		
1	Selected Name	Role	Emp ID	Salary
2	Ross, Logan	=VLOOKUP(name,studentinfo,2,FALSE)	5043	\$ 102,870
3	Next Name in List	Role	Emp ID	Salary
4	Barnes, Melanie	Group Manager	4123	\$ 113,550
5				
6	Name	Role	Emp ID	Salary
7	Flores, Jessica	Communications	4451	\$ 85,360
8	Long, Ryan	Quality Manager	3124	\$ 87,010
9	Hughes, Jackson	Customer Lead	4303	\$ 95,810
10	Coleman, James	Audit	1312	\$ 99,060
11	Ross, Logan	Lead Designer	5043	\$ 102,870
12	Barnes, Melanie	Group Manager	4123	\$ 113,550
13	Perry, Gabriel	Process Intern	2342	\$ 117,740
14	Patterson, Kaitlyn	Supplier Relations	3452	\$ 130,150
15				

Figure 2.20 Example of VLOOKUP function in use

without the exact thing you are looking for being found. We'll see an example of this in Chapter 4. Oddly, while these cases of use exist on occasion, TRUE is the default for VLOOKUP, which leads to countless errors in practice when FALSE is really needed. My recommendation: Always specify this fourth parameter. You'll find yourself using FALSE most of the time.

Now, more generally, although we may already have an understanding of how to use functions that we've read about, Excel also won't leave us completely in the dark if we don't. If I select Insert Function, I'm given a full list of all the parameters associated with the function, some of which may be optional. In the case of OFFSET, for example, the Functions Argument dialog box opens (shown in Figure 2.21). For an example of integrated use, see Figure 2.22.

In Figure 2.22 I used the MATCH function to determine which row a worker's name is in (within just the Names column, NamesList) and I selected the worker's name that appears just after it using the OFFSET function (my starting base is the Nameheader cell in this table). Other functions such as INDEX can provide similar functionality, with some limitations. Illustrations of the differences between how INDEX and combination of MATCH and OFFSET might be applied are provided in the Chp2\_IndexingMatches workbook (see Figure 2.23).

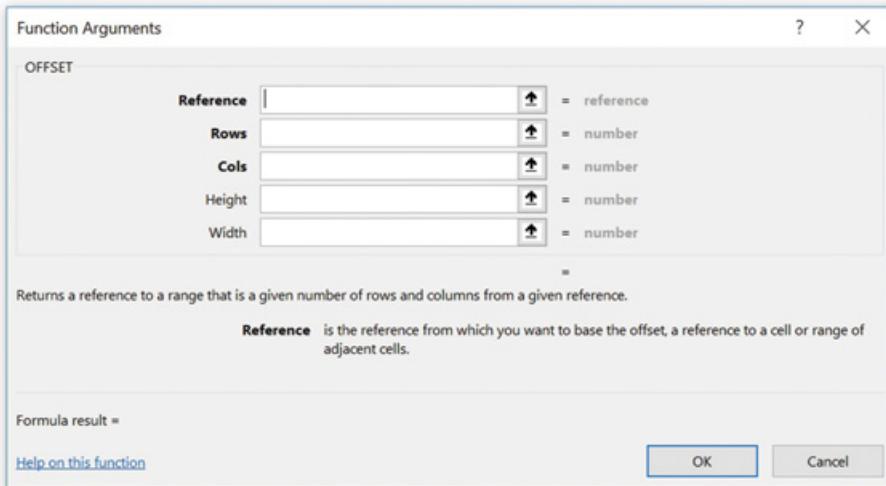


Figure 2.21 Example of assisted field interface for the OFFSET function

The screenshot shows an Excel spreadsheet with data in columns A, B, C, and D. Row 1 contains headers: 'Selected Name', 'Role', 'Emp ID', and 'Salary'. Row 2 contains data: 'Ross, Logan', 'Lead Designer', '5043', and '\$ 102,870'. Row 3 contains: 'Next Name in List', 'Role', 'Emp ID', and 'Salary'. Row 4 contains: 'Barnes, Melanie', 'Group Manager', '4123', and '\$ 113,550'. Row 5 is blank. Row 6 is a header row for a table with columns: 'Name', 'Role', 'Emp ID', and 'Salary'. Rows 7 through 12 contain data for the table. The formula in cell A4 is '=OFFSET(Nameheader,MATCH(name,NamesList,0)+1,0)'. The 'Raw Data' tab is selected at the bottom of the table.

A	B	C	D
Selected Name	Role	Emp ID	Salary
Ross, Logan	Lead Designer	5043	\$ 102,870
Next Name in List	Role	Emp ID	Salary
Barnes, Melanie	Group Manager	4123	\$ 113,550
Name	Role	Emp ID	Salary
Flores, Jessica	Communications	4451	\$ 85,360
Long, Ryan	Quality Manager	3124	\$ 87,010
Hughes, Jackson	Customer Lead	4303	\$ 95,810
Coleman, James	Audit	1312	\$ 99,060
Ross, Logan	Lead Designer	5043	\$ 102,870
Barnes, Melanie	Group Manager	4123	\$ 113,550

Figure 2.22 Example of combined use of MATCH and OFFSET

Now let's discuss the limitations implied in these examples. Why was VLOOKUP not used to accomplish all of these tasks? VLOOKUP, as powerful and convenient as it is, has clear limitations. In order to use VLOOKUP, the content you want returned must exist to the right of the

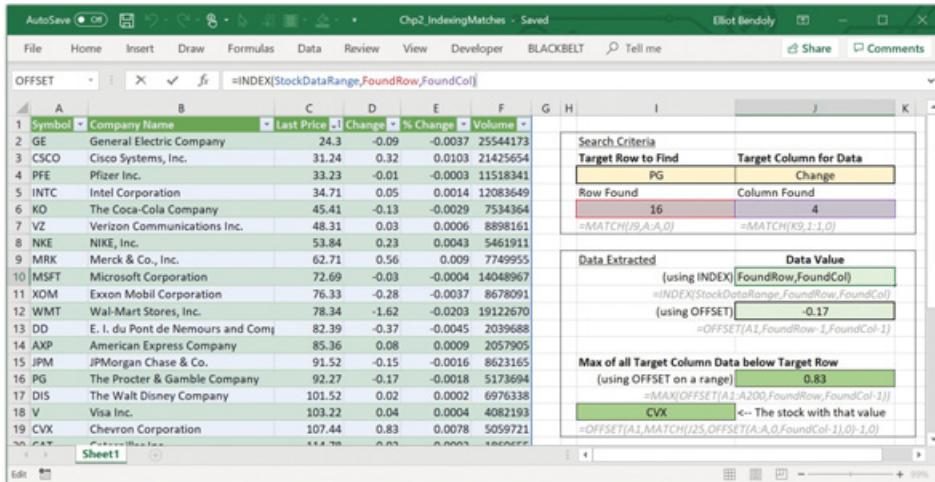


Figure 2.23 Example of MATCH in combination with INDEX

content you are looking for, since VLOOKUP always looks up content in the leftmost column of a referenced range. VLOOKUP is also designed to return only information related to an item sought (if the final parameter is set to FALSE, for an exact search). We will never be guaranteed access to content above or below that entry, even in inexact searches of sorted data. Short of duplicating column data, the combination of MATCH with either OFFSET or INDEX provides some workarounds, though even these have their limitations. While OFFSET can be applied to a range of cells (so that functions such as SUM can work off shifting ranges of data), INDEX doesn't share that capability. This is why it is so useful to have OFFSET as an option.

There are other, more nuanced limitations to families of functions. For example, in cases where content looked up must correspond to multiple columns of data meeting a criterion (e.g., date as well as a specific location), we might not have a preexisting Lookup & Reference function that gets the job done, though we are not without development options here as well. As we will find, Logic functions and the use of logic in general provide a host of means by which to identify complex criteria cases.

For example, we could introduce a column that checks for both of these conditions being met, returning and storing the values, such as TRUE, to be searched for. Or, as an alternate tactic, consider the use of Text functions in the creation of composite index columns (where the “&” is used to merge column data to create unique keys). If you have column data on both location and date (e.g., City, Month, Year), you might be in the position to create a column of unique identifiers allowing data to be quickly extracted from a row containing specific joint content (e.g., “ColumbusFeb2020”). Both of these approaches are a bit inelegant, but they would be functional. And,

critically, they represent a proof-of-concept stepping stone toward more elegant approaches that could be built with back-end code (VBA, which we will visit soon enough). As we will keep seeing, just a simple combination of basic building blocks gets us far in this environment.

### **2.3.2 Logic Function Examples**

Speaking of the family of Logic functions, let's have a closer look at these as well. Conditional logic is more fundamental to computing than most individuals realize. Until quantum computing takes off, most of what we work with is based on a series of ones and zeros, TRUEs and FALSEs. It should be no surprise to see that things like conditional statements abound in the tools that we develop. The most common form of these, in the spreadsheet environment (as well as in VBA), is the IF function.

Examples: `IF(B2=B3, B3*B3, "Not Applicable")`  
`IF(OFFSET(Nameslist,1,1)=MAX(B2:B3), "Maximum", "-")`

Other extremely useful Logic functions in the spreadsheet environment include combinations of the IF statement concept with arithmetic functions (COUNTIF, SUMIF, AVERAGEIF), which allow only select values from an array of cells to be included in such calculations. A limitation of this family of functions is the fact that other critical calculations have not been packaged with conditional logic. It is this limitation in fact that led to the development of additional functions such as StdevIf and PercentileIf, which we'll discuss as part of the Blackbelt Ribbon in Chapter 5 (though we will discuss the Ribbon itself earlier). A more nuanced, though appreciated limitation is the fact that IF statements are designed to look for either-or scenarios rather than an array of cases. This results in the perceived need to embed multiple IF statements within each other to get complex conditions handled. These are really just limitations of convenience and spreadsheet management, though they can nevertheless inspire us to build functions of our own (like StdevIf and PercentileIf) to fill up that gap. We'll be ready to do so by the end of the book.

In general, however, the basic IF statement is flexible enough to accomplish a lot of what you might want to do. Solutions based on IF alone won't always be elegant and they can end up being a bit patchwork (hence harder to debug). And if you get too used to just using IF to solve problems, you can paint yourself into a corner in some analysis efforts (more on the challenge of conditional discontinuity in Chapter 7). But, these caveats aside, when push comes to shove, IF statements at the very least hold the potential to help you get your head around the structure of a problem you're facing, even if you might end up replacing their use with an alternative approach further on. The Chp2\_TaxBrackets workbook is a nice

example of how IF statements are used profusely, just to help us get a sense of the somewhat complicated relationship between data inputs and a specific data outcome. In short, IF is a very handy proof-of-concept function as long as you are willing to consider alternatives to replace it down the road.

To gain still more insight into the nature and usefulness of conditional statements in this environment, a little more time with the gymnastics of compound logic can be useful. For this, I'll refer the reader to this chapter's Supplement. For those not familiar with the nature of logical statements (perhaps from past philosophy or computer science coursework), this supplement will provide some useful insights. Similar approaches to and perspectives in logic will be assumed throughout subsequent chapters in this text.

## 2.4 Data Management (An Introduction)

Continuing down this discussion of the tools available for working with data, it is valuable to mention spreadsheet capabilities that extend beyond cell-embedded or otherwise accessed functions. For example, you may have also noticed that the prior spreadsheet examples (Chp2\_IndentitiesList and Chp2\_IndexingMatches) included buttons of some kind seemingly housed in the header row of the main table. With these we can sort and filter the data in the table. But where did they come from? In this environment, we have a number of ways in which to add further attributes to the cells in our spreadsheets. We'll see more examples of this when we discuss data acquisition. In this particular instance we are seeing the Filter option found within the Data ribbon (Figure 2.24).

Symbol	Company Name	Last Price	Change	% Change	Volume
2 GE	General Electric Company	24.3	-0.09	-0.0037	25544173
3 CSCO	Cisco Systems, Inc.	31.24	0.32	0.0103	21425654
4 PFE	Pfizer Inc.	33.23	-0.01	-0.0003	11518341

Figure 2.24 Accessing the Filter application tool

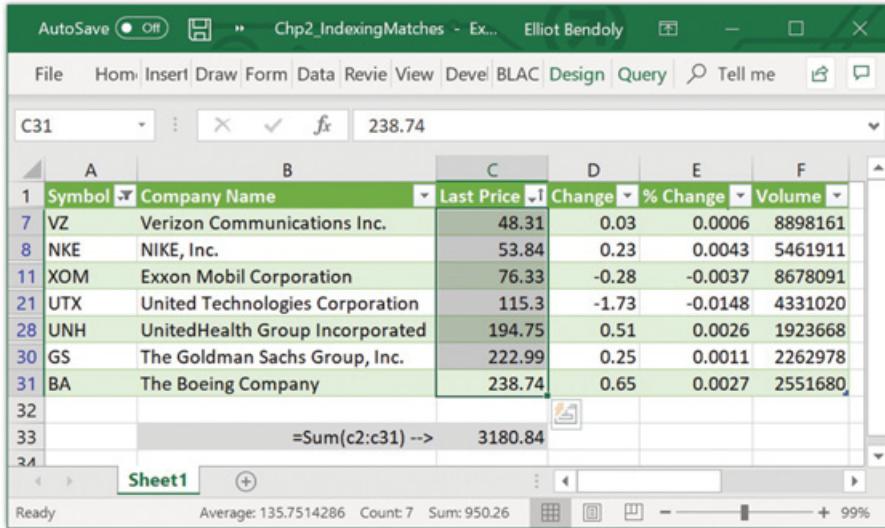


Figure 2.25 Calculations retaining hidden/filtered rows

With a continuous table in place, absent data in immediately adjacent columns and rows, the Filter option embeds these sort and filter options in the first row of the table (ideally, headers are also in place). As you might imagine, being able to quickly sort and filter data provides a major convenience when examining it. However, it doesn't come without risks. Let's imagine calculating the sum of data in the Last Price column for the Chp2\_IndexingMatches workbook, in cell C33. That complete sum is presented in Figure 2.25.

If several of the rows are filtered out, that sum calculated in that cell will appear unchanged, even though the summary at the bottom of the window for the selected cells will change based on that filtering (see bottom of the figure). In retrospect this shouldn't be a surprise, since the range over which the sum occurs hasn't itself been changed; nevertheless, the appearance of omitted cells and the summary provided at the bottom can seem at odds with this calculation. Unfortunately, most users of Excel often don't make this distinction. If you are going to use a tool, you need to know what it does and does not do. Filtering does not eliminate cells from cell calculations.

## 2.5 Copying Content

As shown in the use of the copy prompt in copying fixed data across cells and, in doing so, making implicit use of Excel's pattern recognition capability, functions of all kinds, as well as cell formats and other attributes, can easily be pulled across (or down) ranges of other cells. However, a few caveats are

worth mentioning here as well. Cells conditionally formatted relative to a group of other cells may impact the conditional appearance of other cells in the original range. If not expected, this can occasionally prove frustrating, particularly if new cells in the formatted group represent outliers of some kind. They may reduce the apparent distinctiveness of certain cells in the originally formatted group.

Another caveat deals with functions that include cell references (as many do). When cells with such functions are copied, if they use soft references to cells [for example, “=MATCH(B3, A1:A20,0)”), Excel will automatically change referenced cells as well [for example, “=MATCH(B4, A2:A21,0)”], if the cell copied to is one row below. Obviously, this is often not what a user wants. In this case, a user would probably not want the key range A1:A20 to change. To avoid this change, the use of cell and cell range names can do the trick beautifully because Excel will not attempt to alter the use of such named ranges as it copies among other cells.

Alternatively, one can use hard referencing (aka absolute cell referencing) to prevent any such changes in references as they are copied. A hard-referenced cell in a formula has its column and row each preceded by a dollar sign (\$), such as \$A\$1 or \$A\$1:\$A\$20. When typing such a reference into a cell or function within a cell, you can toggle between soft and hard referencing by pressing the F4 key on the keyboard or, more directly, by adding a \$ as needed. Partial hard references that allow changes in either row (e.g., \$A1) or column (e.g., A\$1) but not both can also come in handy, as we’ll see in some of the more advanced examples presented throughout this book. If there are only a handful of cells or cell ranges that you plan to regularly reference, however, then the best tactic is to use the cell and range naming approaches discussed previously.

## Supplement: The Nature of Conditional Statements

In most management settings, decisions that are made at one point in time affect the kinds of decisions that need to be confronted down the road. For example, consider the choice to expand the number of services offered by a firm to its clients.

If an option to expand is rejected, perhaps no additional decisions on the matter need to be made. However, if an option to expand is accepted, additional questions need to be answered. Should the expansion be targeted toward acquiring new clients or toward better serving existing clients? If we simply want to better serve existing clients, is our end goal to increase their patronage or to increase the likelihood of retention? Are we concerned about encouraging mid-term or long-term retention?

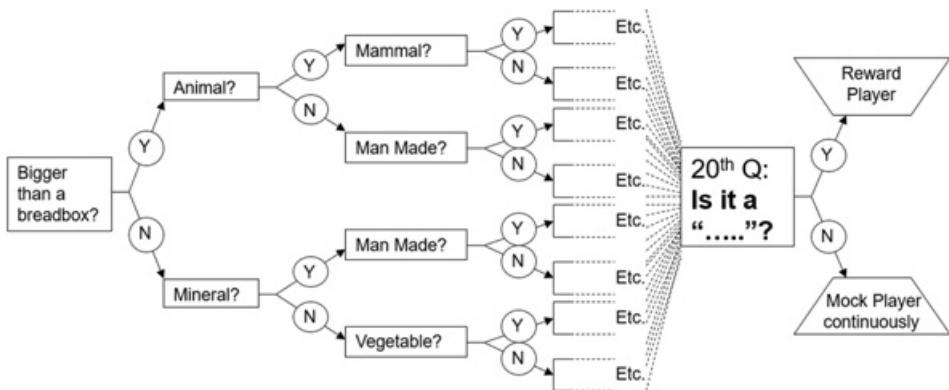


Figure 2.26 Example tree structure for determining identity

The structure of these complex multiphase decisions can be mapped out in a decision tree, a straightforward and commonly used framework. Decision tree structures (shown in Figure 2.26) are useful not only in outlining the course of a decision-making process but also in outlining the course of a set of questions that might be asked in an attempt to assess the specific state of a management scenario. We can draw an analogy here with the common game of 20 Questions, using answers that are TRUE or FALSE (we stick to this assumed limitation for now).

Different types of questions may be relevant when trying to determine what calculations to make based on the current information available to a business analysis (such as data contained in a spreadsheet). For example, let's say we're a firm that manages large advertising projects for other businesses. We have a facility with a limited number of rooms and we typically assign an individual room to a single advertising project. Other rooms may be used for a variety of other activities that we manage: for example, printing, secretarial duties, storage, management offices, and maintenance offices for ongoing campaigns. Occasionally, we may run short of space and need to consider renting additional space. We might want to determine the risk of an event in planning for rentals. But our calculation of risk might be based on a complex set of issues including the number of projects, past space reserved, managers involved, nature of the projects and clients, and so on.

It wouldn't take much for the calculation of risk, conditional on so many variables, to become complex. Let's say that based on the information we've started to lay out, the nature of risk (or uncertainty) we face regarding our need for capacity might be spelled out in a decision tree as shown in Figure 2.27.

Calculating the risk associated with any aspect of this tree can prove challenging, especially since there are implied differences in the risk-dynamics of coordinating different mixes of projects. But no matter how

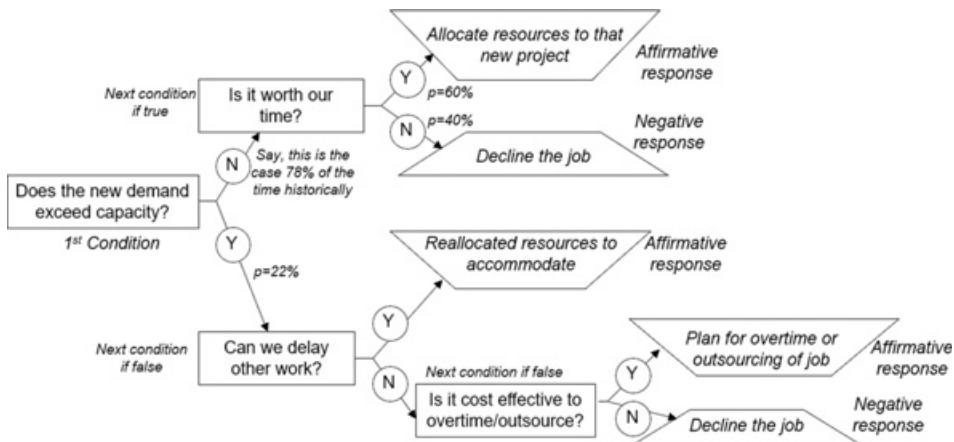


Figure 2.27 Tree structure characteristic of current professional application

Conceptual Condition <u>In 'Plain English'</u>	Computational Condition <u>In 'Plain English'</u>	Computational Condition <u>Formulaic Condition</u>
Does the new demand exceed available capacity?	Is the value in the cell storing "available capacity" (eg. A4) at least equal to the value in the cell showing "capacity demanded" (eg. A2)?	A4>=A2 Where at least A2 comes from an external source
Is it worth our time even if we have the capacity?	Is the expected net revenue generated from the job (eg. calculated in C27) at least equal to the expected net revenue derived from keeping capacity available for other future work (eg. calculated in C108)?	C27>=C108 Where both cells are based on complex calculations
Can we delay other work?	Is there enough slack in between all the existing job due dates (eg. sum(H4:H23)) to accommodate labor hours needed for the new work?	Sum(H4:H23)>A2 x LaborHours
Is it cost effective to overtime/outsource the work?	Is the calculated cost of the best overtime/outsource option (e.g. in Sheet2!G4), less than the revenue from the job?	Sheet2!G4<A1

Figure 2.28 Steps in transitioning from conceptual to computational formulaic logic

strange or complex the conditions of a work system may be, they shouldn't be ignored; rather, they should be captured as faithfully as possible with regard to their potential impact on decision making and performance. Similar decision structures, as well as much simpler ones and much more complex ones, are possible in Excel. The most challenging aspect is not picking the right Excel function to use but rather the translation of all the critical details into a set of succinct logical statements, using words, as a step toward that computational support.

Examples of how analytics transition from real-world conditions into conditions that can be built into a decision support tool are provided in Figure 2.28. We can start simple here, and doing so will give us the best chance of getting this right. When you identify conditions, spell them out as you might explain it to someone on the street. Your first shot will probably be wordy, but try to economize – the more succinct, the easier the next step. Our first goal is to move from the wording of conceptual logic to wording representative of computational logic, which will typically be a little more explicit.

Once this transition is accomplished, the transition into formulaic statements that capture that computational logic can be straightforward, or it may suggest flaws and omissions in thinking. As we will discuss in our sections on structuring decision support models, we need to be willing to recognize these flaws and make corrections.

Once sufficiently captured, however, our remaining work, with data assumed as given, is largely one of syntax and reference. In Excel's spreadsheet environment, the simplest form of the IF statement has three components:

- 1) A condition to test for (should be something that can be shown to be either TRUE or FALSE)
- 2) An affirmative response (what to do if TRUE)
- 3) A negative response (what to do if FALSE)

A series of interdependent conditional statements, as is common in 20 Questions, can be constructed in a number of ways. You can embed an IF statement within another, serving as the latter's input, or you can break the sequence into steps with each IF statement's results residing in separate cells in the spreadsheet. This alternative takes up more real estate on the sheet, but it can be much easier to debug and explain to others (or to yourself two weeks later).

While we've noted some of the limitations of the family of IF statements earlier in this chapter, the usefulness of the IF statement concept can (and often does) extend beyond the contents of any single cell. For example:

- 1) We could use conditional formatting to have the cell appearance change as the contents change (subject to the results of the IF statement).
- 2) Other cells in the spreadsheet may have their values based on the contents of that cell, hence the result of an IF statement can trigger an array of alternative calculations and visual representations.
- 3) Similarly, in recursive use (iteration mode, which we will learn about), we can use an IF statement in a single cell to basically serve as an ON/OFF switch to initialize and execute a host of other repeated activities in the spreadsheet (such as external data record population, Monte Carlo simulation, and so on).

As a final note on how Excel interprets "conditions" or "logic" statements, it is worth mentioning what actually happens when we use a statement such as  $A1>B1$ , either as part of an IF statement, or all by itself. Excel's processing of these expressions amounts to a translation as well, with a result of either TRUE or FALSE. Try it. If you type " $=(A1>B1)$ " into a cell in a spreadsheet, with some values in the cells referenced, you'll find TRUE or FALSE as the cell output. TRUE and FALSE are actually reserved terms (by default they are center-justified in contrast to left-justified text and right-justified numbers). They also have numerical equivalents. TRUE is equal to

1 and FALSE is equal to 0. If you multiply a TRUE by a TRUE, Excel will give you the value 1 ( $1 \times 1 = 1$ ). If you add a TRUE to two FALSES, Excel will give you the value 1 as well ( $1 + 0 + 0 = 1$ ).

This dual interpretation can be extremely handy when you want to do quick calculations with existing data that already take the form of TRUE or FALSE. For example, say you're trying to keep track of the capacity of a network of warehouses. In one column of a spreadsheet you might have records of the storage space for each warehouse. In the adjacent column you might have TRUE or FALSE statements on whether the space is accessible on a given date (perhaps some of these are in regions you can't get to, don't allow you to store what you want to, or simply have been shut down). You could create a third column that multiplies the first (#) and the second (TRUE or FALSE) columns and have a sum at the base of that column to let you know how much space is really available.

This binary equivalency also helps us when making decisions that have cost or revenue implications. Just multiply the cost (or revenue) of the options under consideration by the decisions regarding their pursuit. The sum of those products (e.g., using a function such as SUMPRODUCT) will simply be the sum of all options whose pursuit is TRUE. No IF statements are required. This kind of utilization is a cornerstone to a significant domain of prescriptive analytics referred to as optimization and also proves critical to efficient code designs. Don't be surprised when we continue to return to this idea in later chapters.

## PRACTICE PROBLEMS

### *Practice 2.1*

Each of the capabilities discussed in this chapter may prove useful on its own, though when used in combination their strength is highlighted. As an example, complete the following steps to create a threshold table for z-scores.

- 1) Type the following text into the cells of a new spreadsheet: In A1, type z-score; in B1, type Cumulative; and in C1, type Density.
- 2) In the cells below A1, create a list of numbers from -3, -2.9, -2.8, up through 3. Use the copy prompt to do this. Name the full numeric range of cells' z-scores.
- 3) In the cells below B1, use the NormsDist function to convert the values in the range z-scores (using the name, not the column-row designation, as your reference) into percentages (fixed format them to be viewed as %). Use the copy prompt to copy down the rest of column B. Name that full numeric range of cells Cumulative.
- 4) In cell C2, type =Cumulative. In cell C3, type =Cumulative-B2. Use the copy prompt to copy the contents of C3 down to the rest of the cells in column C.

- 5) In D1, type Threshold =, and in D2, type some number between 0 and 0.04. Name D2 Threshold.
- 6) Use conditional formatting to make the background of any cell in column C green if it is greater than that threshold.

*Practice 2.2*

Select an approximately 200-word paragraph to analyze. This doesn't need to be relevant to your area of expertise, although that might make the exercise more meaningful to you. Import the paragraph into a single cell in a new workbook. Using the FIND and MID functions, decompose it into a list (column in Excel) of individual words. Use the COUNTIF function to create a second column that specifies how many times each word is found in the list you created. Then use conditional formatting to color numbers (in that second column) red if they are greater than 2 and green if they are equal to 1.

# 3

## Acquisition, Cleaning, and Consolidation

**Objective:** *Understand the options and processes for bringing data into the spreadsheet environment, as well as concepts and methods for making it ready for further analysis and reporting.*

Apart from simply typing information into Excel, we have a wide array of options when it comes to introducing new data into spreadsheets. These methods include opening structured, plain-text files in ways that are meaningful to Excel; using other desktop applications as data sources (such as tabular results from SPSS); drawing information from online sources (such as content from cloud databases like Azure, public data resources like baseball-reference.com); and developing techniques for tapping into data that is out there, just perhaps not in tabular form. In this chapter we'll consider options in each of these categories at some level, along with tactics for handling the associated data once accessed.

### 3.1 Text File Imports and Basic Table Transfers

If you have a text file that contains information, such as a survey or downloaded data in text-file format, it can be opened in Excel as well. You simply need to specify how data in that file are organized, such as separated by spaces, tabs, commas, and so on. As an example, let's look at a text file titled Chp3\_MultRespsFinal.txt. Each record in this file occupies a new row, and the information relating to each record is organized sequentially with each field separated by a comma. This kind of data organization is referred to as "comma delimited."

There are at least a couple of ways to get the data in a delimited text file into Excel, regardless of whether it is delimited by commas or by other separators. One is by attempting to open the file as one might any other file in Excel. Selecting File and then Open (File>Open) and specifically searching for the text file of interest will bring up a version of the Text

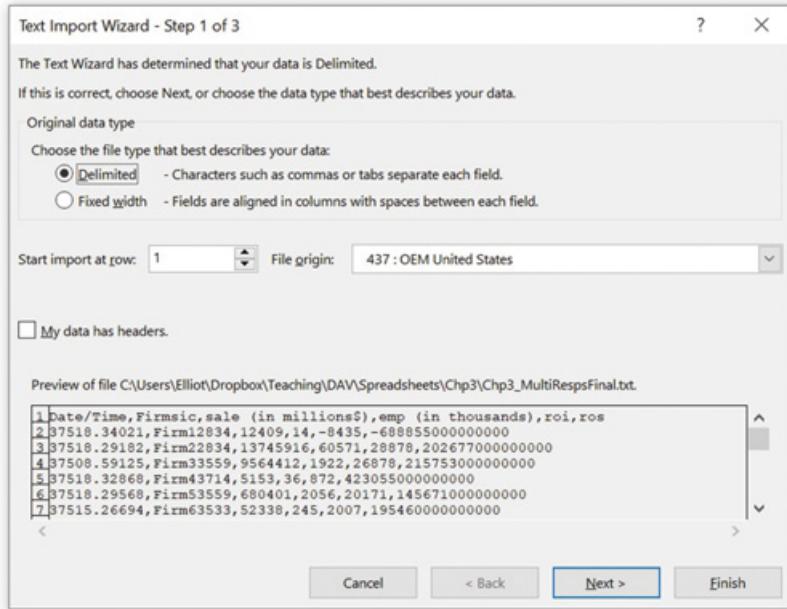


Figure 3.1 Text Import Wizard interface

Import Wizard dialog box. A contemporary view of this wizard appears in Figure 3.1.

As noted, in this case, we have a delimited file. Because of these demarcations, most applications are designed to be able to make sense of data like this. Clicking on “Next” in this wizard allows you to modify how delimiters are used for reading this file (Step 2 of the Text Import Wizard). Clicking on the “Finish” button at this point will complete the data import. What you get is that data split into columns and rows, in the workbook, as requested. You now have the opportunity to save this workbook as a file distinct from, albeit connected to your original text.

If the imported content does not appear to be as clean as you’d like, Excel’s existing functions can provide a range of capabilities post-import. In particular, functions such as FIND, MID, and SUBSTITUTE are some of the usual suspects in spreadsheet-based cleaning operations, at least when moderately sized data sets are involved. The FIND function allows you to identify where other special delimiters (e.g., multiple characters of text) might exist. It opens the door to more nuanced parsing, or simply to parsing that doesn’t require you to restart the importing process.

FIND specifically provides the location of a character or string of characters within a larger string (say the contents of a specific cell). With FIND you are also provided the opportunity to indicate where to begin your search for content, which can be very useful if you are looking for subsequent

occurrences of the same delimiter (e.g., a space) within a body of characters. The MID function is often used with the FIND function to extract portions of a string beginning at a specific character location and for a particular length of characters beyond that point. A nice way to draw out a critical portion of imported text, SUBSTITUTE is similarly useful in removing or replacing specific characters for additional processing (e.g., replacing double spaces with single spaces prior to the use of FIND and MID).

In recent versions of Excel, an additional mechanism for importing has been made available for similar work. Under the Data ribbon, the leftmost section includes a family of tools referred to as “Get & Transform Data.” The “From Text/CSV” option similarly allows you to search for a text file of interest and import its contents. The default setting in this case is comma delimitation (hence the CSV reference). Depending on your version of Excel, this alternate process may take a little longer to execute and the presentation of content in the spreadsheet may appear different (rows in alternating fills to emphasize table structure). This is depicted in Figure 3.2.

In this case as in the previous one, the import is not simply a one-shot transfer operation. What you have built is a connection between your workbook and the source text file. You can see this connection alluded to by right-clicking on any cell in the imported range. The option for “Refresh” should be visible. This means that any time you suspect the source file might change (perhaps it is a file being regularly shared with you from an enterprise source), you only need to right-click and Refresh to draw in the most updated content, provided that the text file, even if updated, remains in the file location it was originally found in. This Refresh option and variations on it exist in other contemporary imports. It can prove to be fairly convenient, as we’ll see.

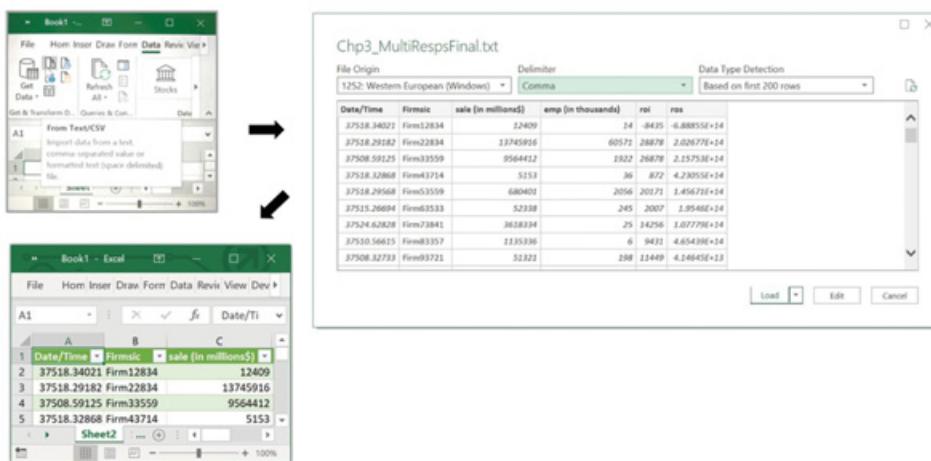


Figure 3.2 Get & Transform Data from Text/CSV

## 3.2 Online Data Acquisition

The integration of alternative data-rich applications with Excel has the potential to open up numerous opportunities for developers and managers who might otherwise be unaware of these convenient resource integrations. At this point let's look at how we can bring the Internet into our discussion of data resources.

### 3.2.1 Generalized Web-Data Pulls

Many online sources of potentially valuable data are already publicly available and updated on a regular basis. The number of these sources is certainly on the rise, as are the means by which to access these. Understandably, some sources are much harder to tap into than others. Further, just as the data presented through these sources may change regularly, the availability of these sources yesterday may not be the same as today. This also applies to some extent to the tools available in Excel for getting at this data.

As a case in point, consider the legacy Web Query mechanism that for a long time was the go-to tool for importing online content into Excel. The second edition of this text focused on that tool, despite some of its drawbacks. More recent versions of Excel make access to this tool somewhat less obvious but if you happen to have a workbook in which one of these legacy connections was built, this traditional mechanism can still be explored. We have this available in the form of the workbook Chp3\_LegacyWebQueryExample. Opening this book, we find a table of the Dow Jones industrial components and their most recent stock prices. There's a lot of extra content as well, but right-clicking on any cell in this table reveals that the cell content is in fact linked as a query to an online source (Figure 3.3). The nature of that source can be shown by selecting “Edit Query” after right-clicking.

What follows is fairly tedious and certainly one of the reasons that this particular kind of query has been sidelined. The presentation you get is something that looks a lot like a web browser and one that is usually followed by an often long series of script warnings and errors that you have the option of accepting or ignoring. They are in part associated with the variety of nontabular data content that is embedded within the online source (ads, images, form fields, etc.) – things you probably couldn't care less about. Eventually you'll see the presentation of the page, with some additional yellow arrowed check boxes, which used to be fairly common tools for selecting individual tables within web pages for import (Figure 3.4).

In newer sites these curious additional check boxes prove less useful, as individual tables are often not recognized by the legacy mechanism. Regardless, what you can do from here often is very useful. For example,

	A	B	C	D	E	F	G	H
66	Day's Range	25,914.37 - 26,143.92						
67	52 Week Range	21,712.53 - 26,951.81						
68	Avg. Volume	350,25						
69	Currency in USD							
70	Top 30 Components							
71	Symbol	Company Name						
72	HD	The Home Depot, Inc.						
73	PFE	Pfizer Inc.						
74	MMM	3M Company						
75	JPM	JPMorgan Chase & Co.						
76	VZ	Verizon Communications Inc.						
77	UTX	United Technologies Corporation						
78	KO	The Coca-Cola Company						
79	TRV	The Travelers Companies, Inc.						
80	CAT	Caterpillar Inc.						
81	PG	The Procter & Gamble Company						
82	BA	The Boeing Company						
83	DWDP	DowDuPont Inc.						
84	MRK	Merck & Co., Inc.						
85	MSFT	Microsoft Corporation						
86	INTC	Intel Corporation						
87	MCD	McDonald's Corporation						
88	CSCO	Cisco Systems, Inc.						
89	GS	The Goldman Sachs Group, Inc.						
90	IBM	International Business Machines Corporation	139.2	1.07	0.77%	3,030,044		

Figure 3.3 Spreadsheet content once imported

Symbol	Company Name	Last Price	Change	% Change	Volume
HD	The Home Depot, Inc.	185.17	0.03	+0.02%	5,481,417
PFE	Pfizer Inc.	43.36	0.01	+0.02%	25,301,489
MMM	3M Company	207.49	0.10	+0.05%	1,754,444
JPM	JPMorgan Chase & Co.	104.43	0.07	+0.07%	13,813,310
VZ	Verizon Communications Inc.	56.96	0.04	+0.07%	10,942,737
UTX	United Technologies Corporation	125.77	0.10	+0.08%	3,822,045

Figure 3.4 Example legacy Web Query interface

Top 30 Components					
			70 <- =MATCH(A1,Sheet1!A:A,0)		
Symbol	Company Name	Last Price	Change	% Change	Volume
HD	The Home Depot, Inc.	185.17	0.03	0.0002	5481417
PFE	Pfizer Inc.	43.36	0.01	0.0002	25301489
MMM	3M Company	207.49	0.1	0.0005	1754444
JPM	JPMorgan Chase & Co.	104.43	0.07	0.0007	13813310
VZ	Verizon Communications Inc.	56.96	0.04	0.0007	10942737

Figure 3.5 Using Excel functions to clean Web Query imports

you can modify the URL of interest by direct editing or pasting as one might in a regular web browser. You can even navigate links as with a regular browser. And ultimately you can import all the text associated with any page you land on, even though much of it will not be of interest. This includes data content that isn't recognized as a legitimate table (an important point worth recognizing).

In reality getting extraneous data shouldn't be a serious barrier to the use of these data queries, but it might create an impetus to take additional action to clean things up a bit. What solutions exist? Here is where the convenience of Excel functions comes up once again. Particularly useful are MATCH (for finding key markers in the imported data, beyond which critical data exist) and OFFSET (for pulling out data located near the result of MATCH), as illustrated in Figure 3.5. Also useful again are both FIND and MID (for parsing data that gets imported as merged with other content). In the case of numbers that are results of that parsing, or which otherwise appear to be read as nonnumeric text (i.e., left justified), a number of simple corrections are also available such as the addition of 0 (=A1+0) to prompt numeric interpretation by Excel.

Whatever cleaning or modeling work you do should ideally be positioning on a separate sheet, or at least sufficiently to the right of the import space so updates don't overwrite this critical work. Losing information can be frustrating, so I'd also recommend saving various versions as you develop your workbook. It's good practice for all workbook development efforts, frankly.

Speaking of data updates, as with the import of content from text files, once the data is in place you have a few options. Right-clicking on any cell in the query also gives access to the Data Range Properties of the query. This

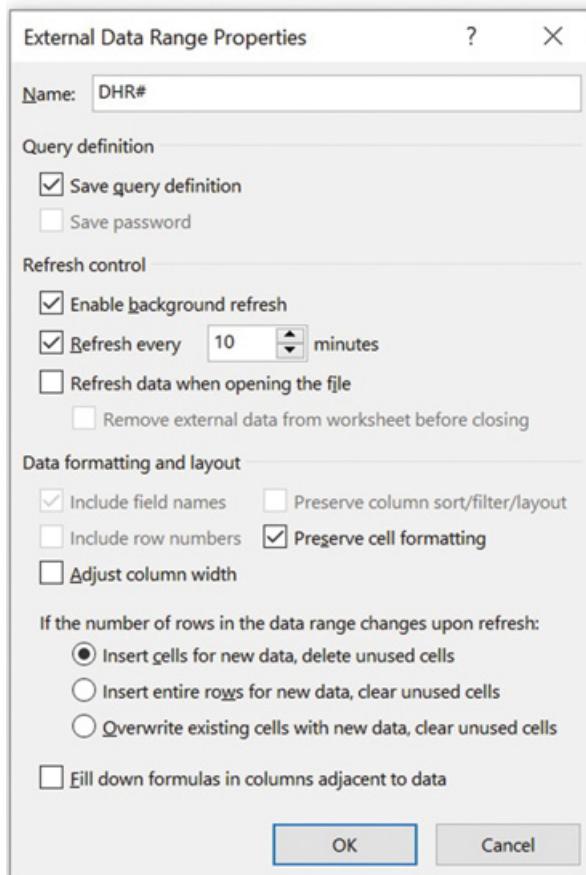


Figure 3.6 Changing Data Range Properties for a Web Query

dialog box, as shown in Figure 3.6, gives you the opportunity to specify how often you want Excel to refresh the query (assuming that the external content is being updated in a way that would be useful to you). In this example I've asked for 10-minute updates. Through this interface you can also turn off default adjustments to column widths and toggle the retention of cell formatting, among other options. Further on in this chapter, we'll take a look at how we can take greater advantage of automated refresh settings on such queries.

### 3.2.2 Add-in Assisted Web Queries

While the process of getting a legacy Web Query off the ground can get a bit frustrating, despite the fact that it permits access to a fair amount of content and some nice data range options after the fact, we are not without alternatives

for accomplishing similar results. These approaches bypass the browserlike interfaces and the associated issues that attempts to work with web page scripts can create. One of these alternatives comes available with the Blackbelt Ribbon add-in. Originally designed with a focus on Google documents, the ReadWrite-G tool allows you to specify any URL and attempt a pull, with or without rich content, retaining or not retaining an associated new query, all without opening up the mini browser that the traditional Web Query approach requires.

First, let's make sure that we have a little introduction if this is the first time you've used the Blackbelt Ribbon, or any add-in for that matter. Add-ins are additional applications compatible with Excel, designed to extend the capabilities of the spreadsheet environment. The most recent version of the Blackbelt Ribbon add-in can be downloaded from the main Blackbelt site, [www.excel-blackbelt.com](http://www.excel-blackbelt.com). Once downloaded, the same procedure for including any third-party add-in can be applied. Your first step is to go to File and Options (the same place we went to edit custom lists in Chapter 2). With the new window open, select the Add-ins option in the list to the left (see Figure 3.7). Clicking on “Go . . .” with Excel Add-ins selected from the drop-down at the bottom of the screen provides a view of add-ins recently available to your application. Checked boxes will denote those add-ins currently active. To include the Blackbelt Ribbon in this list, browse for where you've downloaded it and select “Open.”

Now since the Blackbelt Ribbon is designed, in part, to help us connect to outside sources (as in the current Web Query task), you are likely to see a pop-up Excel Security Notice regarding data connections. Don't worry! First off, we do want to connect to data here, which is sort of the point. So hitting Enable is justified here. Granted, if you choose Disable, most of the tools in the Blackbelt Ribbon will still work just fine. It's just that some tools that expect these connections to be enabled might not. So, if you choose Disable, don't be surprised in those cases. In any event, the add-in should now be active on your version of Excel. When you don't need to use the Blackbelt Ribbon tools, just uncheck this box. You won't be erasing the add-in, just deactivating it until needed next.

The Blackbelt Ribbon is designed for best use with contemporary versions of Excel, operating on PCs. Excel operating off of MacOS may encounter issues, even with parallels/bootcamp running, both with regard to this add-in as well as others. When the Blackbelt Ribbon is added in, you will find a new tab in your ribbon structure labeled BLACKBELT. Expanding the BLACKBELT tab reveals the Blackbelt Ribbon's suite of tools, of which there are several (see Appendix B).

The first few buttons on this ribbon include a quick link to the Blackbelt site (where updates, demo workbooks, and other content can be found),

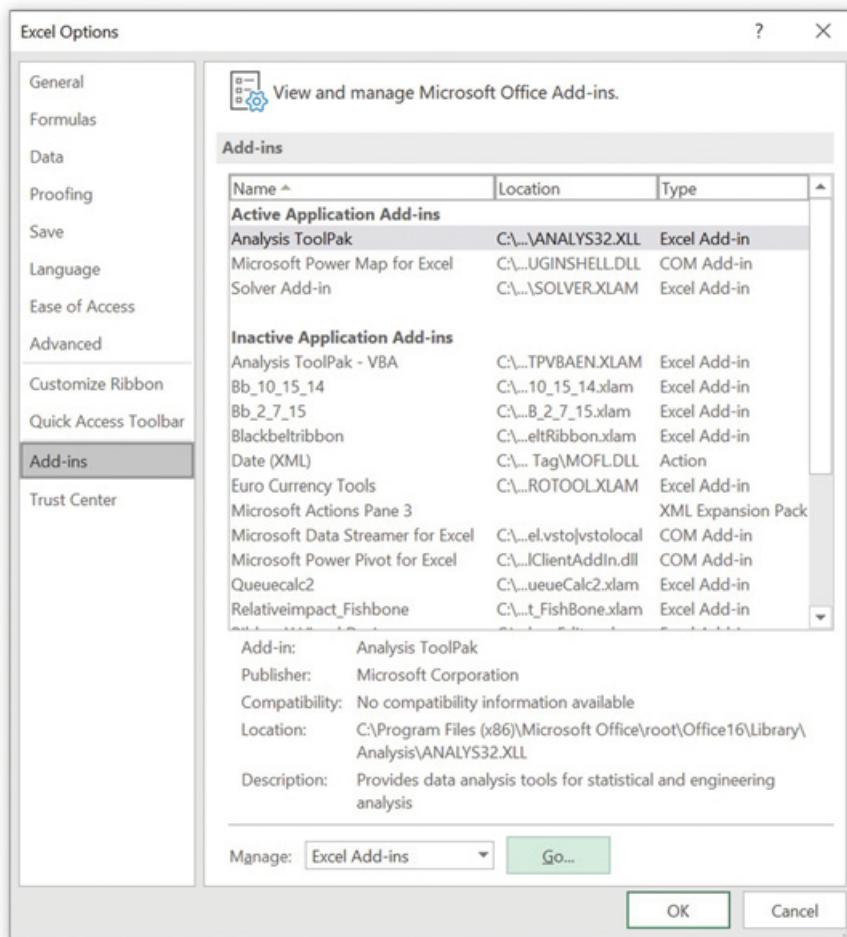


Figure 3.7 Including and activating Add-ins in Excel

a shortcut key setup for the ribbon's various tools, and a default page installer. The defaults page installer is designed to help you customize the use of the Blackbelt Ribbon tools uniquely to specific workbooks, allowing inputs to these tools to be saved as defaults specific to the tasks you work on. The shortcut setup uses specifications in the defaults page to assign keyboard activation of the Blackbelt Ribbon tools if desired. It also registers the special worksheet functions that come with the add-in, which we will cover along with the other Blackbelt Ribbon tools in later chapters.

To the right of the Initialize Default button we find the main Blackbelt Ribbon tools. The one we want now is the Read Write-G tool, found under Data Acquisition. In this tool we have both the ability to generate legacy Web Queries, without having to confront the browserlike interface and script reading issues, as well as write to a specific kind of web destination, Google

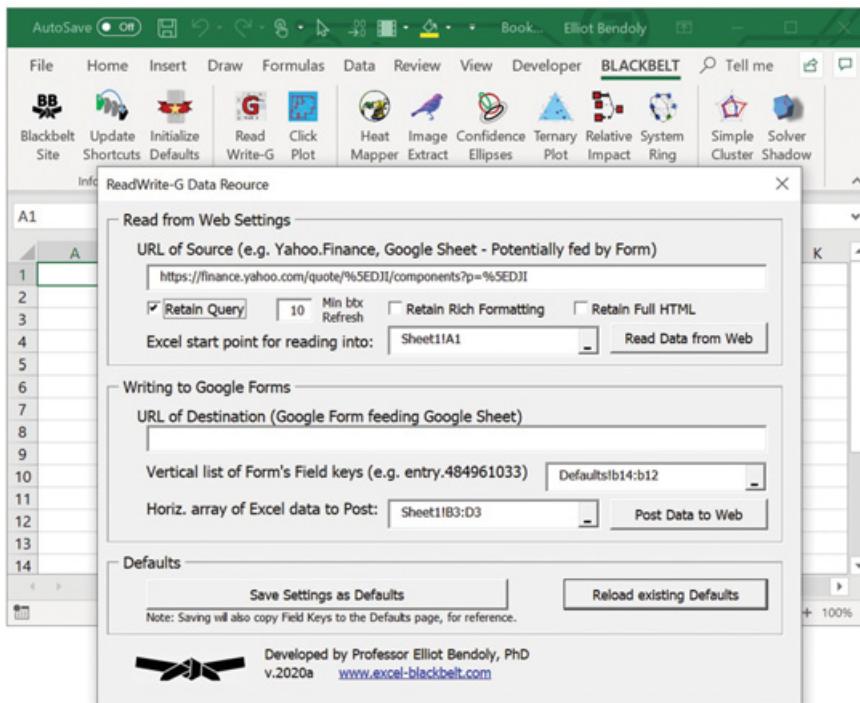


Figure 3.8 Read Write-G query development interface

Forms. We will discuss the latter down the road but for now let's focus on the Web Query generation. In the first half of the Read Write-G interface, we have the ability to specify a source URL (like <https://finance.yahoo.com/quote/%5EDJI/components?p=%5EDJI> from the prior example). We can also request the query link to be preserved and data range updates to be made at specific intervals. Figure 3.8 illustrates this specification.

Clicking on “Read Data from Web” creates the query and pulls in the data available. Ultimately, all of this is possible because Excel currently permits queries to be created and edited using its back-end VBA programming language. As with other things, we will spend more time on this later. Regardless, knowing that otherwise cumbersome tasks can be simplified through the use of code can be inspiring. Here, I've simply packaged that code as a new user interface. You'll be able to do something along the same lines by the end of this book.

Since we've activated the Blackbelt Ribbon and imported some content from the web, we can also consider additional operations the Blackbelt Ribbon might provide. For example, if you've imported content in Full HTML form, you might be importing not just numbers and text but also hyperlinks. Apart from inspecting these elements manually, or clicking on them and viewing the pages

they refer to, you could use a function to extract the URL of those links directly. Of the several custom functions that the Blackbelt Ribbon provides, we have available to us the ExtractURL function. This is a function you wouldn't encounter from Excel itself in your examinations of the functions in Chapter 2. But it is a simple function nevertheless. Just enter =ExtractURL() with a cell reference as the parameter within the parentheses, and if that cell has a hyperlink, the function should return those details.

Keep in mind that if you save a workbook with a Blackbelt Ribbon function in place, that function should continue to work when you reopen that book, provided two things: (1) You have the Blackbelt Ribbon active. (2) You have the Blackbelt Ribbon add-in located in the same place it was when you saved your workbook. If you've moved the add-in, even if reactivated, the function's instance on the workbook may try to reference the add-in from the original location (file path). You might encounter this if someone else shares a workbook with you that they developed on their own computer. You can reenter the function but that might be tedious if there are multiple instances. There is a better fix. I've included in the Blackbelt Ribbon a set of code that strips file paths from any selected range of cells. If you have the add-in and have established the default keyboard shortcuts, selecting the range of cells of interest and hitting Ctrl-N should clean things up, removing the extraneous paths and allowing direct function referencing.

### 3.2.3 Power BI and New Web Queries

Microsoft's latest push into extended application capabilities is its Business Intelligence (BI) tools, with lessons learned from past experiences with Web Queries, the earlier MS MapPoint and World Wide Telescope experiments, and database interactivity. The BI tools currently include Power Query, Power Pivot, and Power Map, with aspects of each increasingly built in ubiquitously within Excel, as well as an associated desktop platform.

We will examine the Power Map utility when we discuss visualization. For the time being, it's worth at least exploring how this advancement has impacted the most readily available forms of Web Queries. In modern versions of Excel, in the same section of the Data ribbon where we established an active import to a text file, we have a Web Query option. The interfacing for this modern version of the Web Query is distinct from the legacy form and avoids the same concerns that we have already discussed. The associated process for accessing our example web data set is presented in Figure 3.9.

Note that the same kind of tabular presentation formatting applied to our text import is used here in the resulting import from the web page in question. It is further worth noting that only certain content from that page is getting pulled in. This presents both pros and cons to the import, especially if

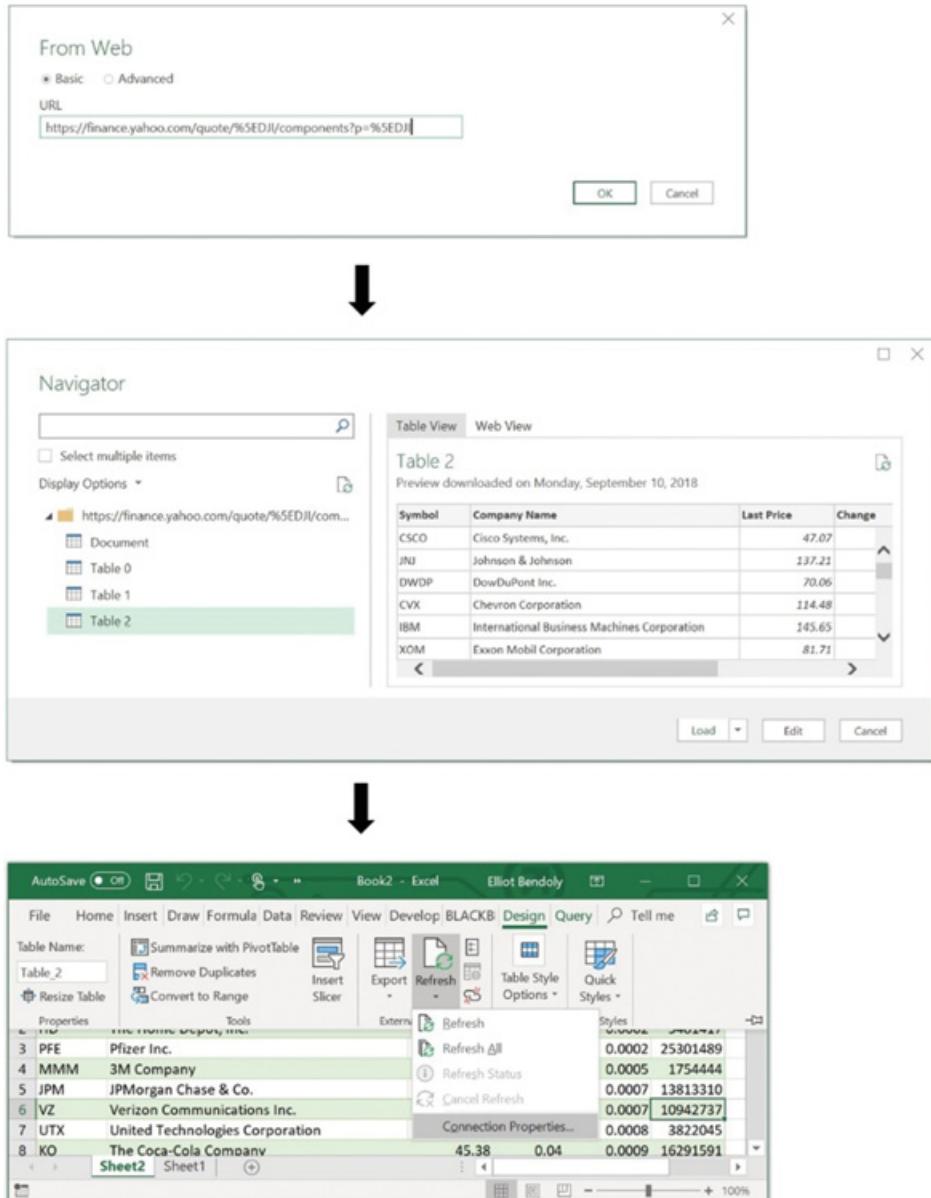


Figure 3.9 Contemporary web import interface for data pulls

nontabular content in the web page is desired (that content generally won't be recognized by this query). Again, that may be exactly what you need, but it might also be less.

Once the content is imported, or more specifically once the web page link has been established, a similar interface for modifying data range properties appears, as with the legacy pulls. Access is a bit different, however. Selecting

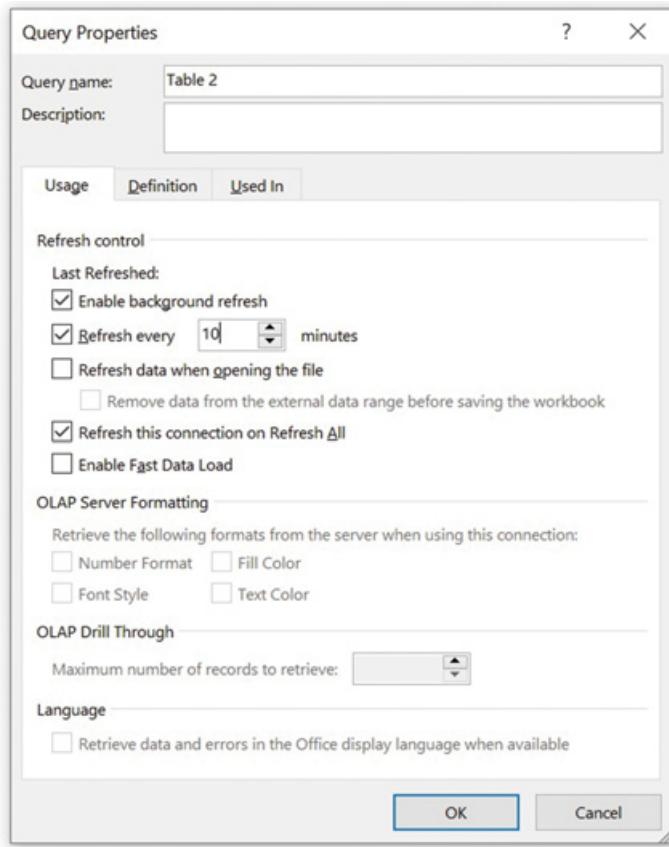


Figure 3.10 Data Connection options from the Design tab

any cell in this new query makes available new Design and Query tabs in the ribbon structure. In the Design ribbon you will find a Refresh drop-down and “Connection Properties . . . ”. Figure 3.10 provides a view of the associated options, which should look familiar.

There is a lot to like with this approach. However, if you want content that isn’t recognized as tabular, you may find yourself returning to either legacy Web Query editing or the Blackbelt Ribbon tools. There’s nothing wrong with using different tools for different tasks. It’s common. And it’s often the most effective route.

### 3.3 Living Data Records: The Basics

A discussion of how to connect to data and how to refresh associated imports is interesting but incomplete without some plan for storing important content over time. All we’ve seen so far are ways to overwrite older imports. To start us thinking about processes for retaining past content, it’s useful to

think about how the spreadsheet environment might create some opportunities for this sort of thing.

To begin this discussion, consider the standard calculation mode in Excel, something that most people take for granted. Specifically the default setting for Excel is what we might refer to as noniterative automatic, meaning that every time you make a change in the workbook, all cells are updated. The results of past calculations are not automatically stored. But sometimes we want to keep track of the past. The iterative calculation option enables you to gain some of that control.

### 3.3.1 Circularity and Iterative Calculation

The iterative setting allows for calculations within cells that build iteratively on the results of prior calculations (e.g., in A1, having the formula A1+1). In general this is referred to as a circular reference because you're asking the computer to base the value of something off itself. If the computer was told to do this continuously, that value would soon become huge (and would keep growing). When faced with this situation, some software will give you an error message saying that something like a circular reference, or circular loop, or circular calculation has been detected and that there is no defined final calculation to arrive at. Hence no effort to calculate will be made. Try entering the calculation =A1+1 into cell A1 of a standard workbook and see what you get.

With an adjustment to the calculation options of Excel, specifying iterative calculation, you can overcome this barrier. Figure 3.11 shows where this

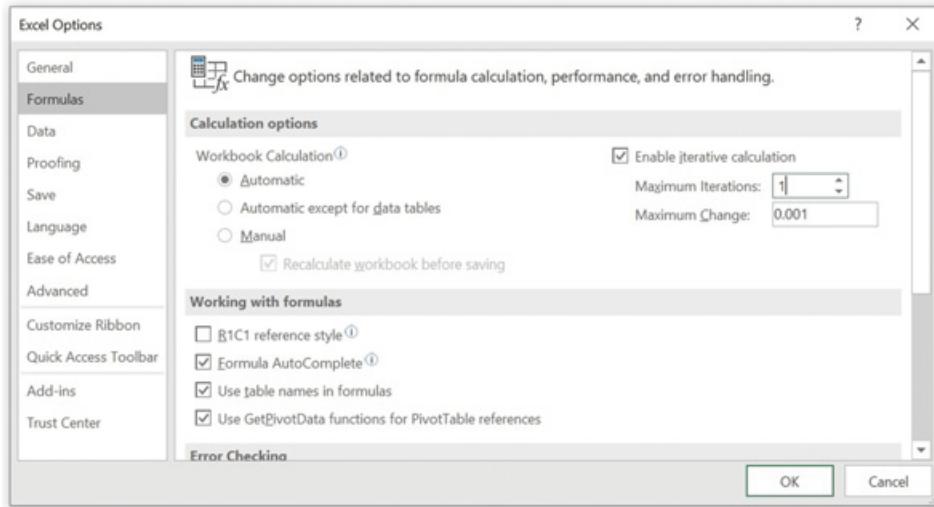


Figure 3.11 Initiating Iterative Calculation mode

feature can be modified within Excel Options. Iterative mode allows developers to say exactly how many times the computer should repeat a circularly referenced calculation before stopping. For many cases, such as Web Query data collection, one iteration at a time is appropriate.

Note that you should use iterative mode only when it's your best option. Trying to create other types of spreadsheet workbook tools can be tough in this mode, as well as frustrating. Most work is done in the noniterative mode. Our current examination is an exception.

Under the iterative calculation setting, there is a standard and predictable sequence in which calculations are performed involving circular references (i.e., all cells referring to themselves or part of a system of calculations that are circularly referencing). These calculations begin with the upper-left cell in such a system (e.g., A1 if part of a circularly referencing calculation) and then progress through all such cells in the first row of the spreadsheet from left to right until that row comes to an end. Calculations then resume, starting at the first column of the next row and progressing again from left to right, until all cells containing calculations are handled. This is illustrated in the workbook Chp3\_IterLoop (see Figure 3.12).

In iterative mode, recalculations are started by pressing F9 (the recalculate key). Other actions in the workbook (e.g., data refresh, entering a new formula) will also trigger this when Automatic recalculation is enabled. In

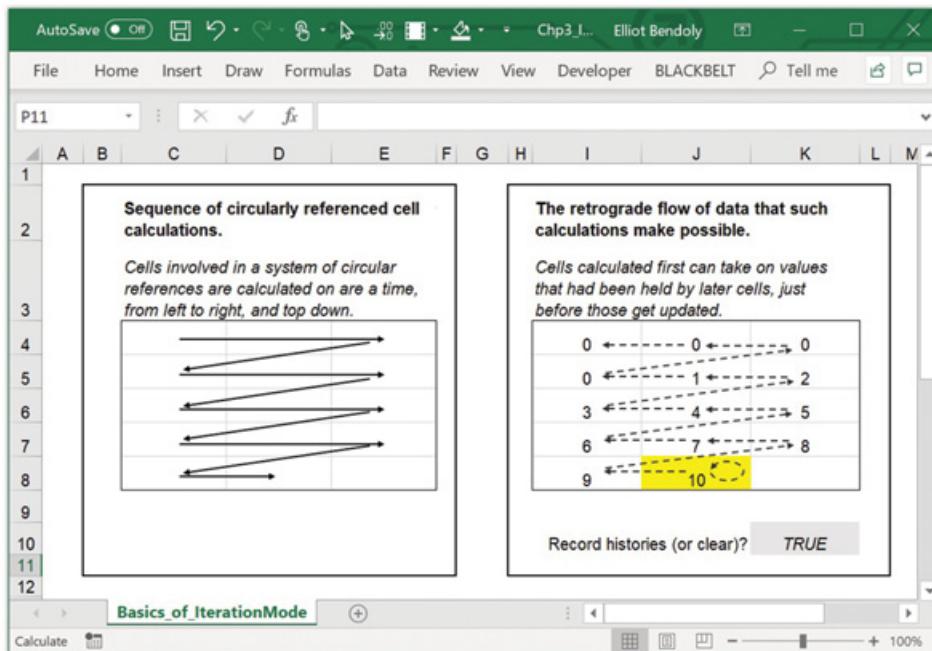


Figure 3.12 Order of calculations in a Circular Reference system

single iterative calculation mode, if there is a calculation to be made in cell A1, whenever F9 is pressed Excel will base that calculation off all information currently available (information in all other cells as well as that currently in A1). If there is subsequent calculation to be made in cell B1, that calculation will take into account all current information as well as the value just now calculated for A1. Similarly, any calculation for B2 would involve the updated values for both A1 and B1. After calculations are made for cells during a single iteration (by pressing F9), they will not change until another iteration is started. This holds for all mathematical calculations, logic statements, and text functions, as well as random number generation. What we end up with is what is referred to as a “living record” in that it is a record of past information that can be allowed to continuously evolve.

The Chp3\_IterLoop workbook contains a simple working example of such a living record. In this example I have set up fourteen cells such that the values contained in each of the first thirteen are based on the value of the cell following it. I’ve depicted this dependency with dashed lines. Notice that flow of data in the record is retrograde (in the opposite direction) of the order by which cell calculations are made in the iterative calculation mode. The value in the fourteenth cell is calculated by increasing itself by 1. In other words, it contains a circular reference. Since the other cells are connected to this cell’s calculation, they are all part of the same system of circular calculation.

The specific structure of this series of calculations is often referred to as a “bottom-up” design. This design allows new data generated in lower cells to “percolate” up to cells above them (or to the left) rather than simply overwriting each other with a single value in each update (which would occur if the calculations were instead pulling from the top). In this case, I’ve also restricted calculations so that this recording occurs only when the cell named Restart is set to TRUE. When its value is FALSE, all cells in the system are reset to 0. The value of resetting systems such as this will come up again when we discuss simulation modeling, though it can also prove relevant to our current objective: collecting Web Query data.

### **3.3.2 Example: Web-Query Histories**

So, we’ve now seen examples of how to draw data into our workbooks from various sources. We’ve also seen the potential for a workbook to house a living record based on prior data developed. Now let’s try to combine the two approaches to create a simple mechanism by which to collect subsequently imported query results.

Let’s think about our approach for a second before jumping in. With online data provided by external sources, it often makes sense to record new data only when we’re sure that they’re different from past data.

Otherwise we are bound to be collecting a lot of duplicate content. If we collect the time at which any new data is collected, we can simultaneously track periods over which data stagnates. In a spreadsheet this will mean basically selecting at least two columns in which you want to store records (one column for your imported data, another for the time at which the data are recorded). This approach will need only a very limited variety of cell calculations to get off the ground. In this case, two of these calculations will be repeated throughout most of your data record.

A typical spreadsheet layout for this purpose is provided in the workbook Chp3\_WebQueryLivingRecord and depicted in Figure 3.13. This workbook is just a version of the earlier workbook in which the legacy Web Query was

Top 30 Components						
70 <- =MATCH(A1,Sheet1!A:A,0)						
3 Symbol	Company Name	Last Price	Change	% Change	Volume	
4 HD	The Home Depot, Inc.	185.17	0.03	0.0002	5481417	4:38:48 PM 208.23
5 PFE	Pfizer Inc.	43.36	0.01	0.0002	25301489	4:39:01 PM 207.88
6 MMM	3M Company	207.49	0.1	0.0005	1754444	4:39:47 PM 207.76
7 JPM	JPMorgan Chase & Co.	104.43	0.07	0.0007	13813310	4:40:01 PM 208.05
8 VZ	Verizon Communications Inc.	56.96	0.04	0.0007	10942737	4:40:47 PM 207.56
9 UTX	United Technologies Corporation	125.77	0.1	0.0008	3822045	4:41:00 PM 208.13
10 KO	The Coca-Cola Company	45.38	0.04	0.0009	16291591	4:41:47 PM 207.80
11 TRV	The Travelers Companies, Inc.	133.03	0.12	0.0009	1360403	4:42:00 PM 208.02
12 CAT	Caterpillar Inc.	137.47	0.13	0.0009	3810952	4:42:46 PM 207.72
13 PG	The Procter & Gamble Company	98.44	-0.11	-0.0011	7733231	4:42:59 PM 208.21
14 BA	The Boeing Company	440.62	0.66	0.0015	5124178	4:43:46 PM 207.77
15 DWDP	DowDuPont Inc.	53.34	0.11	0.0021	11375992	4:43:59 PM 207.52
16 MRK	Merck & Co., Inc.	81.65	0.36	0.0044	10137977	4:44:59 PM 208.17
17 MSFT	Microsoft Corporation	112.53	0.5	0.0045	23501169	4:45:45 PM 207.64
18 INTC	Intel Corporation	53.3	0.34	0.0064	18359325	4:51:43 PM 208.35
19 MCD	McDonald's Corporation	185.05	1.21	0.0066	2690988	4:52:21 PM 208.13
20 CSCO	Cisco Systems, Inc.	51.41	-0.36	-0.007	23683679	4:52:37 PM 207.53
21 GS	The Goldman Sachs Group, Inc.	198.2	1.5	0.0076	2580797	4:52:40 PM 207.92
22 IBM	International Business Machines Corp	139.2	1.07	0.0077	3030044	4:52:42 PM 207.51
23 V	Visa Inc.	149.47	1.35	0.0091	7016918	4:52:46 PM 208.10
24 DIS	The Walt Disney Company	114.01	1.17	0.0104	6996911	4:52:51 PM 207.99
25 AAPL	Apple Inc.	174.97	1.82	0.0105	25886167	4:52:55 PM 208.28
26 WMT	Walmart Inc.	97.93	-1.06	-0.0107	10352471	4:52:59 PM 208.20
27 AXP	American Express Company	108.9	1.16	0.0108	2682041	4:53:21 PM 207.65
28 XOM	Exxon Mobil Corporation	80	0.97	0.0123	15419422	4:53:34 PM 207.56
29 JNJ	Johnson & Johnson	138.35	1.71	0.0125	5641586	4:54:20 PM 208.17
30 UNH	UnitedHealth Group Incorporated	246.15	3.93	0.0162	6921274	4:54:34 PM 208.09
31 NKE	NIKE, Inc.	87.16	1.65	0.0193	6315062	4:54:42 PM 207.52
32 CVX	Chevron Corporation	122.03	2.45	0.0205	7743561	4:55:11 PM 208.19
33 WBA	Walgreens Boots Alliance, Inc.	66.61	-4.58	-0.0643	10299296	4:55:12 PM 207.63
34						4:56:50 PM 208.46
35						5:00:38 PM 208.01
36						5:00:38 PM 208.35
37						5:01:01 PM 207.49
38						5:01:01 PM (J37,"")
39						TRUE

Figure 3.13 Specific structure of living record in Web Query example

demonstrated, though the “living record” principles certainly can be applied to any query created by the Blackbelt Ribbon or MS Power Query. In this example three cells have been named: MostRecent, Comparison, and Restart. I’ve also added conditional formatting to create a kind of embedded bar chart (we’ll talk about graphics in Chapter 5), to help emphasize relative differences across the data collected.

The Restart cell (J38) is used in the same way as in the simpler example but in this case cell content is set effectively to a null value rather than 0. The cell named MostRecent (J37) simply references one of the stock values in the cleaned section of this sheet, which in turn is being updated along with the Web Query refresh rate. In the Comparison cell (J36) we have the following formulation: =IF(Restart,IF(MostRecent=Comparison,J36, J37),""). In other words, if Restart is TRUE, this cell at the base of the living record is being compared to the most recent data of interest. If those values are the same, this cell (J36) retains its value, but if they are different at the time of calculation, the cell takes on the value of the one below it (J37 in this case). Copying this equation across the bulk of the living record (I2:J36) ensures that differences in those two particular cells, and only those two, trigger the entire record’s update. This avoids data replication but retains time stamping.

Just sit back (or leave your desk) and allow the collection to cook. After a while you’ll have some potentially interesting data available to work with; that is, provided the markets remain open and provided you don’t need to use Excel outside of iterative calculation mode during this time. If you do, you may have some alternatives to consider, based on lessons in coding on later in the text.

### **3.4 Selective Filtering and Aggregation**

Now accessing and collecting data is one thing. Knowing what to do with it is something else entirely. Increasingly professionals are bogged down in vast amounts of available data. The seemingly basic task of selecting which data should be used to develop solutions often becomes a stumbling block and slows the development of meaningful analysis and solutions. The overwhelming nature of large amounts of data applies to all aspects of decision making, particularly to the ability to apply logically designed heuristics to solution development. This highlights one of the general misperceptions as we approach greater access to data – that more data, broadly defined, is always better. Clearly if you are getting lost in the data available or if you lack the capability of processing it, you aren’t getting additional value through volume. When there are specific questions to answer, the claim that a million records can provide much more “practical” insight than ten

thousand is a hard one to justify. Increasingly huge amounts of data may give us more to do but it's not always value added.

At the same time we are also a global society that is generally misled by data aggregates. We hear a lot about averages. Many make plans based purely on those averages and face pitfalls because of it. Massive data, used purely to construct more accurate averages, is a misguided approach in general. If we are going to leverage this data at all, it is going to be through identifying nuances in the data: significant separations (groupings) across observations and attributes and significant interrelationships that can help us predict changes. We'll talk about the latter in Chapter 4, so for the moment let's talk a bit about grouping.

Experience (even for lay persons) leads us to recognize that, when it comes to rationalizing the data we have access to, what to leave in or take out isn't always obvious. Fortunately we have some approaches we can at least consider in tackling this challenge. Let's think about our options by analogy to the spreadsheet environment. We have rows and columns in our spreadsheets. Often the rows represent individual observations. Often the columns represent attributes of these observations (price, time, name, type, etc.). Not always, but often.

Imagine being faced with a huge number of observations as well as a huge number of attributes per observation – many rows and many columns. One thing we might consider doing before attempting to extract some intelligence from this data is rationalizing those columns. We might, for example, select a set of attributes that represent similar or associated issues and then somehow consolidate them into a smaller set of representative characteristics for each data record. Here, we are grouping together attributes (elements that describe records of people, places, and things) and the consolidated result should have the same number of individual data observations but with a consolidated number of characteristic attributes (see Figure 3.14). This results in fewer columns in your spreadsheet to consider in subsequent analyses.

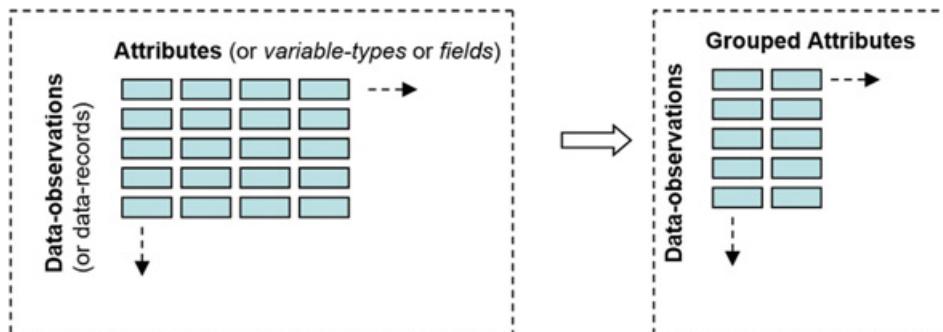


Figure 3.14 Consolidating attributes of large data sets

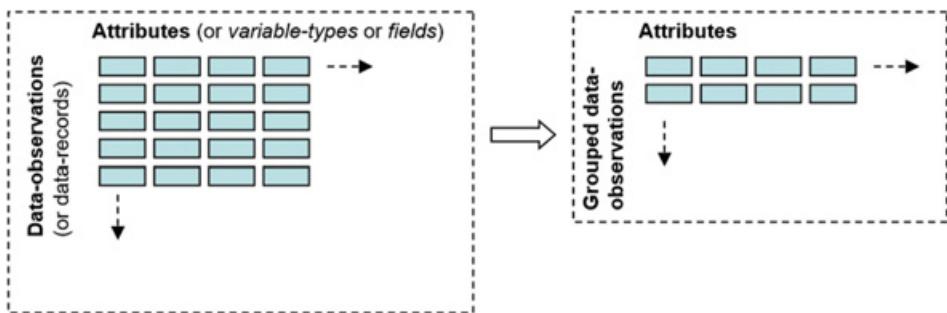


Figure 3.15 Consolidating observations (individual records) of large data sets

The other approach, as you might guess, deals with grouping together similar kinds of data observations (i.e., people, places, or things) to reduce the noise that may be inherent across individual observations. A simple example would be to designate groups of students (e.g., by year, major, or fraternity) and create averages and measures of variation (e.g., standard deviations) of the attributes by which they are characterized (e.g., GPA, starting salary, summers spent as interns). We would be left with a set of consolidated data with the same number of attributes but with much fewer observations (see Figure 3.15).

Ultimately, we could do both – again attempting to reduce the complexity of the decision-making task – while avoiding the actual elimination of specific types of data or descriptors. Neither should be done arbitrarily or without a justified plan of subsequent analysis in place. There are, however, multiple tactics you can use to get the work done, plan in hand.

### 3.5 Guidance in Attribute Grouping

If you think you simply have way too many attributes describing the people, places, or things that you have data on and you feel that several of these attributes may actually represent the same issue (or closely related issues), you have a number of options available to you for consolidating your data.

#### 3.5.1 Rudimentary Attribute Consolidation

One of the simplest ways to consolidate a group of attributes is by creating an average value across them. If you are justified in creating an unweighted average (i.e., simply add attribute  $X$  + attribute  $Y$  + attribute  $Z$  and divide by 3 to create some new overall attribute,  $XYZ$ ), the critical point is to make sure the attributes are similar in units and scale so that the calculation actually makes sense. For example, it might make sense to average the

three student attributes – *grade in BUS330*, *grade in BUS331*, and *grade in BUS432* – to create something general (perhaps referring to it as “average Organization and Management area grade”). It would make less sense to average *score on 330 final*, *bowling score*, and *score on breathalyzer test* (which are different scales and hopefully unrelated issues).

Alternative ways to consolidate attributes may be to take advantage of other standard functions in Excel. For example, if it makes more sense to create something similar to “max performance in Organization and Management,” you might use the MAX function to consolidate all final grades in organization and management classes into a single, best attribute. Similarly, if you’re less interested in the overall performance of students and more interested in their tendency to perform inconsistently in a particular discipline, you might calculate the STDEV across related scores. Maybe those who tend to perform more inconsistently in a discipline are doing so for a particular reason that might be meaningful to you.

These are examples that hit close to home for a typical management student, but the same simplified attribute-grouping technique is used in the marketplace:

- Maybe we don’t care what consumers spend on individual items but rather what they spend on groups such as all perishables, electronics, and so on.
- Maybe we don’t care about how much all stocks vary on an hourly basis but we are concerned with their morning and afternoon ranges.

In any case, the methods suggested put the responsibility of consolidation decisions entirely on the shoulders of individual decision makers or DSS designers. They are assumed to be entirely conscious of why certain attributes should be grouped together and why either unweighted averaging or the use of other summary measures is appropriate. Furthermore, an unmentioned assumption in these examples is that these schemes are applicable regardless of the nature of the sample population as a whole. That is, they would provide the same summary results per record regardless of how many subsequent records are involved.

### 3.5.2 Statistical Attribute Consolidations

Sometimes we don’t really have a strong feeling about how attributes should be weighed in consolidation, only that there’s probably some redundancy in the information they provide and that our decision process would be easier if we could crunch them down to more generalized (yet still meaningful) groupings. Fortunately for us, someone has already done the work here. One popular method freely available to us is Principal Components Analysis (PCA).

PCA is a statistical technique that attempts to create a reduced subset of attributes based on a larger set of potentially closely related (perhaps redundant) attributes. It is not constrained to the assumption that all attributes have equivalent relevance in such consolidation but it does base its statistical approach to the development of weighting schemes on the entire sample of data as a whole (and hence can be sensitive to the size and constituency of that sample). There are numerous third-party application options available for helping us with this kind of work (Palisade, XLStat, JMP, R, etc.). As an example of the kind of results that might be derived through PCA consolidation, let's consider the case of Dodecha Solutions, Ltd.

Dodecha is a small consulting firm that manages a number of information technology implementation projects over the course of a year. It has been collecting both preproject selection and postimplementation data for several years and now wants to base its consideration of future client requests on that data. Dodecha had the foresight early on to recognize that many preproject characteristics could not be assessed by anything but subjective (opinionated) reports of their own consultants. They developed a highly structured set of evaluation questions for their consultants to fill out every time they were given a potential client project request to consider. They specifically designed multiple questions aimed at revealing similar higher-level issues (e.g., potential problems with client participation in projects, uncertainty relating to specific technologies), knowing that the use of only a handful of potentially biased questions could provide an extremely misleading view of project potential.

Table 3.1 provides the full list of the higher-level issues (left column) and the associated more specific set of items (right column) their consultants were asked to evaluate for each client project proposed. For each of Dodecha's 115 past projects, prior to making the decision to accept the proposed projects, managers evaluated each of these 33 characteristics on a scale from 1 to 7 (with 1 indicating strong disagreement with the statement and 7 indicating strong agreement with the statement). The full data set along with the complete set of analysis to follow is provided in the Chp3\_DodechaSolutions workbook.

Again, aware that any one of these items alone might provide a misleading impression of how managers viewed these projects prior to signing on, Dodecha went back to the full data set and attempted to consolidate it into the original eight factors the management highlighted as being useful in later categorization and analysis. In doing so, they would be reducing their analysis set from essentially  $115 \times 33$  (3,680) pieces of data into what they view as a more meaningful set of  $115 \times 8$  (920). To do this they plan to use PCA.

Table 3.1 *Items thought to reflect higher-level management issues for Dodecha*

Ops Fit Issues	A1 : Will require redesign of many processes A2 : Will require elimination of many processes A3 : Will require resequencing of many processes A4 : Will require addition of many processes A5 : Will require changes in work assignments
Industry Instability	B1 : Tech capabilities in client industry change frequently B2 : Process capabilities in client industry change frequently B3 : Market of client industry changes frequently
Inter-Org Concerns	C1 : Client demonstrates poor information sharing C2 : Client demonstrates lack of interest in involvement C3 : Client demonstrates poor worker-resource sharing C4 : Client requires fairly limited time windows for system access/change C5 : Client requires fairly restricted access to stakeholders
New Tech Concerns	D1 : New technology untested by related firms D2 : We lack familiarity with nuances of the technology D3 : Patches to new technology are forthcoming D4 : Value of new technology to market still uncertain
Expertise Issues	E1 : Will require new training for our project staff E2 : Will require work outsourcing to experts E3 : Will require repeated contacts with vendor
Legacy Concerns	F1 : Much data stored in legacy systems to be replaced F2 : Much data formatting based on noncompliant standards F3 : Culture of use/championship of legacy system F4 : User skills highly tuned to legacy system need changing F5 : Many ties to external systems customized to legacy
Client Issues	G1 : Client has demonstrated process design and control problems G2 : Client has demonstrated managerial leadership problems G3 : Client has demonstrated poor technical know-how
Org Complexity	H1 : Involves many separate facilities of client H2 : Involves many functional groups of client H3 : Involves many decision makers at client H4 : Involves many stakeholders at client

To demonstrate the use of the PCA consolidation, in the second edition of this text we focused on the use of XLStat. In this third edition, since we will be visiting Palisade again in later chapters, we are going to take a look at the StatTools offering by this application suite. When installed, Palisade's StatTools ribbon functions similar to any other in Excel. Like many

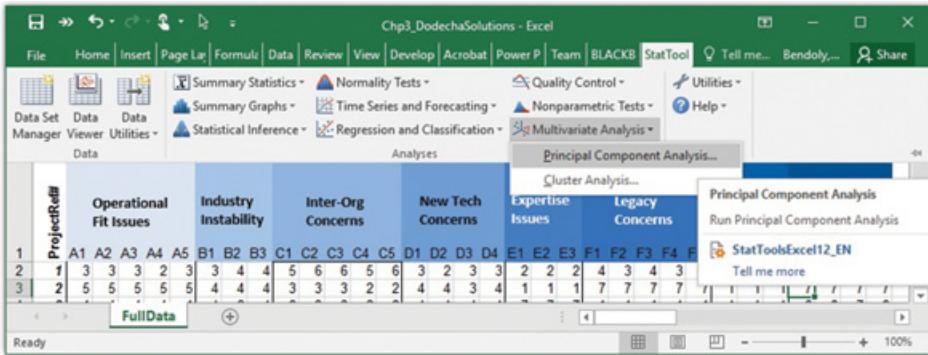


Figure 3.16 Principal Components Analysis in the Palisade StatTools ribbon

statistical packages, the request is that we first specify the data of interest within the Data Manager of this tool (left side of the StatTools ribbon). Once the data is specified, we can engage the PCA engine of StatTools, found under Multivariate Analysis (see Figure 3.16).

The application of the PCA tool can be fairly straightforward. The critical step required by the dialog box is the selection of the data set to be consolidated and some impression of how to consolidate the selected attributes (e.g., into eight components/factors). Although numerous options are made available for professional use and professional users of this technique are certainly encouraged to investigate their benefits, a demonstration of the basic kinds of results to be expected is sufficient for discussion here. Figure 3.17 depicts some of the options immediately presented by this tool.

The next steps happen largely behind the scenes; however, the output provided by Palisade is extremely rich (separate reports and visuals can be provided detailing the PCA process). In the most familiar terms for our purposes, the task the PCA algorithm faces can be viewed with regard to three issues: an Objective, a set of Utility Variables, and their Connections. In the context of data analysis, an Objective is a calculation representative of performance in a given task, while Utility Variables are the numerical or nonnumerical practical choices available to influence that Objective. Connections define how Utility Variables impact the Objective and how certain Utilities constrain changes in each other. In this case we have the following:

**Objective:** Construct new factors based on the existing set of attributes, such that the new factors are as uniquely distinct from one another (uncorrelated) as possible.

**Utility Variables:** The extent to which each attribute contributes to each new factor (e.g., coefficients in each factor equation).

**Connections (Constraints):** Construct exactly eight new factors based on 32 original attributes (in the Dodecha Solutions, Ltd., case).

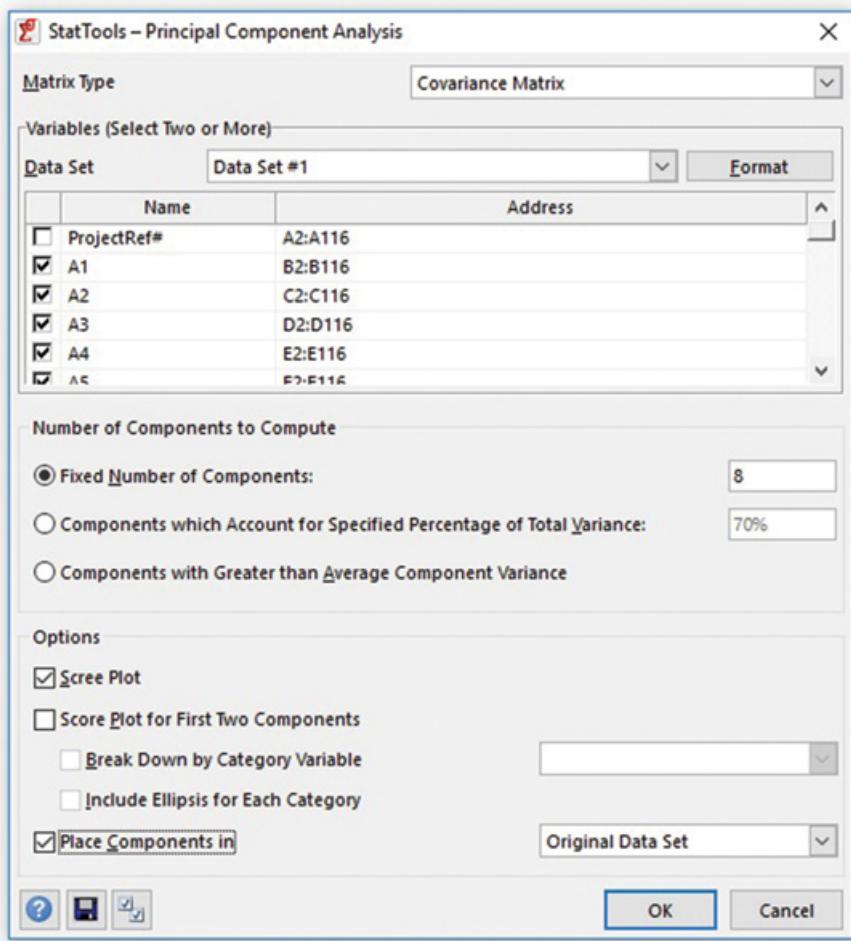


Figure 3.17 Data and component specification in StatTools PCA

Figure 3.18 presents some of the results to come out of this procedure. Assisted by a little conditional formatting, we see an output table of what are referred to as component scores, which are higher in absolute magnitude the more each item relates to one of the eight factors derived from analysis.

Although there is no guarantee that individual attributes will naturally group into the kinds of factors originally conceived of by the designers of the survey (i.e., the eight factors conceptualized in the early table), there is often some correspondence between what was originally conceptualized and the PCA results. What is certainly clear here, as is usually the case, is that the attribute items don't load purely on one factor alone (i.e., some attributes provide information that is

	A	B	C	D	E	F	G	H	I	
108		Components	1	2	3	4	5	6	7	8
109	Attributes									
110	A1	0.202	-0.29	0.005	-0.26	0.053	-0.02	0.067	-0.05	
111	A2	0.208	-0.3	-0	-0.23	0.08	-0.02	0.093	-0	
112	A3	0.21	-0.3	0.001	-0.23	0.07	1E-04	0.046	-0.04	
113	A4	0.214	-0.3	0.008	-0.23	0.058	-0.02	0.064	-0.07	
114	A5	0.212	-0.3	-0	-0.25	0.067	-0.02	0.046	0.008	
115	B1	0.011	0.005	-0.06	0.045	0.128	0.183	-0.17	-0.47	
116	B2	-0	0.018	-0.07	0.014	0.088	0.244	-0.17	-0.5	
117	B3	-0	-0.02	-0.06	0.057	0.134	0.189	-0.16	-0.5	
118	C1	0.01	-0.03	-0.1	-0.07	0.106	0.2	-0.29	0.174	
119	C2	-0	0.016	-0.11	-0.11	0.1	0.233	-0.28	0.238	
120	C3	0.004	-0	-0.1	-0.1	0.076	0.2	-0.3	0.201	
121	C4	-0	-0.01	-0.11	-0.07	0.091	0.173	-0.26	0.21	
122	C5	0.004	-0.01	-0.12	-0.05	0.089	0.216	-0.33	0.241	
123	D1	0.025	-0.01	-0.02	0.047	-0.05	0.409	0.299	0.057	
124	D2	0.009	-0.01	-0.01	0.048	-0.09	0.407	0.291	0.035	
125	D3	0.046	-0	1E-04	0.041	-0.03	0.34	0.25	0.058	
126	D4	0.003	-0.01	0.012	0.068	-0.08	0.418	0.304	0.076	
127	E1	0.275	0.125	0.268	-0.06	-0.37	0.073	-0.17	-0.03	

Figure 3.18 Example components structure derived from PCA

helpful in forming other factors as well). The specific values of these coefficients are far from immediately intuitive. This is clearly not an equal-weighting scheme as might be derived through a simple averaging approach to attribute consolidation and not one that could simply be guessed.

It should also be emphasized that although this outcome suggests a consolidation of 32 attributes into eight components, as the result of our request to the StatTool engine, the PCA process can take many different directions depending on tool and specification. For those interested, the Chp3\_DodechaSolutions workbook also retains the results from XLStat, including the use of an approach known as varimax rotation. The loadings can appear rather distinct by tool and method, which is all the more reason to scrutinize the results you are presented with. Blind adoption of analytical results can lead you down some very bad roads. Take the time to understand what you have.

### 3.6 Guidance in Record Grouping

Similar to the discussion of grouping attributes, we can start this discussion by considering approaches that might be relevant, provided we have a strong understanding of how we should group, based on simple statistics. We can then proceed to discuss approaches where such group structures are fairly unknown. In some real-world cases, decision makers already have a logical notion of which types of records should be similar enough to group together as a subsample. For example, maybe we have a reason for grouping student data by major, or companies by industry, or projects by technology type. If this is the case, we have a preexisting and well-reasoned structure for grouping. On the one hand, if this categorization scheme sufficiently reduces the complexity of our decision-making process, we're set and can move on to other forms of analysis and decision making.

On the other hand, maybe the scheme doesn't sufficiently reduce complexity. Or maybe we don't have sufficient reason for grouping our data by some prespecified category and would like some analytical assistance. Or maybe we just want to group by some numerically driven scheme, such as a sort or percentile split, where we divide our data based on whether it is positioned in the top quarter, bottom quarter, or one of the quartiles in between. We can easily do all of this in Excel.

#### 3.6.1 Percentile-Based Categorization

There are a host of approaches available for us to at least consider in this particular task. One is the Rank and Percentile tool, part of the Data Analysis resource under the Data ribbon (Figure 3.19). This tool is a standard add-in that comes with most Excel installations. If you don't see it active, just go to where we added the Blackbelt Ribbon and make sure this one is checked as well. This tool asks you for a set of observations (e.g., a couple thousand quarterly EPS figures for various firms in a spreadsheet) and the location of the relative ranking of the observations based on a single

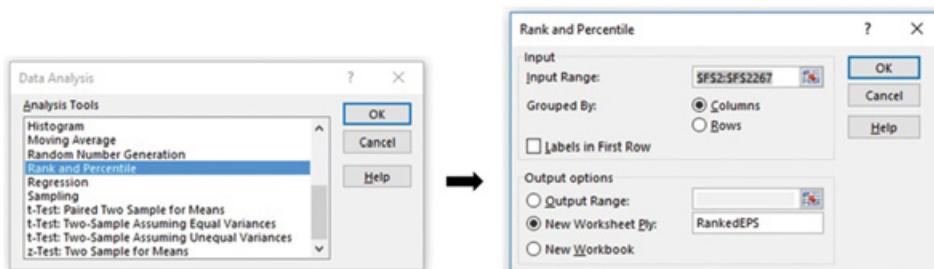


Figure 3.19 Rank and percentile analysis for observation grouping

	A	B	C	D
1	Point	Column1	Rank	Percent
2	74	2.28	1	99.90%
3	748	2.28	1	99.90%
4	367	2.25	3	99.90%
5	578	2.24	4	99.80%
6	1414	2.24	4	99.80%
7	957	2.23	6	99.70%
8	1047	2.23	6	99.70%
9	857	2.22	8	99.60%
10	899	2.22	8	99.60%
11	2073	2.22	8	99.60%
12	1010	2.21	11	99.40%
13	1462	2.21	11	99.40%
14	1816	2.21	11	99.40%
15	435	2.18	14	99.30%
16	2151	2.18	14	99.30%
17	1702	2.17	16	99.30%
18	1893	2.16	17	99.20%
19	375	2.15	18	98.90%
20	789	2.15	18	98.90%
21	839	2.15	18	98.90%
22	1136	2.15	18	98.90%
23	1860	2.15	18	98.90%

Figure 3.20 Example output of rank and percentile analysis

attribute. As would be expected, the more data you have, the lengthier the sorting process.

As an example of how to use this tool in a consolidation effort, I've requested a simple percentile ranking based on no other criteria than quarterly sales across all quarters, for which we have data, from the Chp3\_QuarterlyData spreadsheet (see Chapter 4). Figure 3.20 shows an example of the results generated by the Rank and Percentile tool.

Now there are a few things worth noting here. Look at the rankings presented. Notice that some ranking numbers (1, 18, etc.) are repeated whereas others (2, 10, etc.) are absent. Now notice the actual numeric differences in these observations. The numerical difference between the score of those ranked 11th and those ranked 8th is 0.1 but so is the numeric difference between those ranked 3rd and 4th. So, is the difference between the 11th and the 8th three times ( $11 - 8 = 3$ ) the difference between those ranked 3rd and 4th ( $4 - 3 = 1$ ), or is the difference the same?

Most reasonable individuals would say that the original continuous data is the yard stick for comparison here, rather than the rankings themselves. However, reason can be inconsistent. Consider for a second all the rankings that are presented to the public and how much information they mask or skew. In most of these cases, unfortunately, it is fitting that the adjective “rank” means foul smelling. Ultimately there aren’t a lot of good reasons for drawing conclusions based on rank when real data is available and there are a whole lot of bad ones we should seriously worry about. If there is a belief that individuals are unable to make decisions in the absence of this simplification, then maybe some of this is warranted. Personally I have a bit more faith in the ability of individuals to make sense of actual data.

On the other hand, there are some potentially responsible approaches to using percentile conversions. Typically these would involve looking at differences in the aggregate of percentile bins rather than trying to make skewed comparisons between observations. Considering differences between such aggregated bins can help us derive some generalizable intelligence regarding potentially complex landscapes that our observations are alluding to. But methodologically we might want to stop and ask, is the Data Analysis resource the best tool for a more involve analysis? The Data Analysis resource is, admittedly, a somewhat curious add-in given that so many of its capabilities are immediately available as Excel functions: Correlation, t-Test, random number generation, regression, rank and percentile, and so on.

In part this is the result of Excel’s advancement over the years, with minimal modifications to this legacy resource. The major drawback to the use of the Data Analysis features, compared to related Excel functions, is the static nature of results. After all, if any errors are found in the original data, unless additional automation is put in place, those percentile determinations (and t-Tests, regressions, etc.) will all have to be done manually. The use of Excel’s functions, on the other hand, could provide equivalent and immediately updated percentile bins as such numbers change. Over the range of EPS data in this example we could have each observation placed into approximate quartiles for aggregate quartile comparisons using something like:

$$=\text{TRUNC}(4*(\text{PERCENTRANK}(\$F\$2:\$F\$2267,F2)-0.0001)+1$$

We’ll see some parallels with regards to regression analysis in Chapter 4.

### 3.6.2 Categorization Based on *p*-Levels and *z*-Scores

But even this kind of an approach (percentile binning) has some fundamental drawbacks, which come back again to the troublesome loss of information that occurs when one takes otherwise meaningful information and tries to

make sense of it in a strictly ordinal manner (i.e., as if the magnitude of differences in data didn't matter but only their order did, a generally troublesome perspective to take). We can do better. And we typically have enough data available to us to in fact do better.

Measures of standard deviation from the mean provide the seeds for other meaningful and related approaches to grouping. We live in a world full of distributions with central tendencies but also long and sometimes truncated tails. If you are going to calculate a mean, why not take into consideration the nature of variability around that mean (e.g., standard deviation as a simple metric, though understanding skewness will also prove critical). With that in mind, z-scores and associated p-level of observations can be determined and more effective separation and binning can take place to provide representativeness for second-level aggregate considerations.

Let's assume for simplicity that we currently have data whose distribution is statistically indistinguishable from a normal (bell) curve (i.e., let's assume that there is no need to use any of an array of reasonable continuous transformations that do not require the loss of data that ranking would). A z-score is simply a rescaling, without loss of relative magnitude differences, that centers data at the mean, dividing those values by the standard deviation of the distribution. With data in this form, you have the basis for designating whether any given observation is typical (e.g., z-scores between -1 and 1) or atypical (e.g., outlying z-scores that are greater than 2 or less than -2). While these cutoffs may seem arbitrary, they do at least benefit from establishment in the statistical and process control literature. We don't need to guess how to split up our data if we have some reasonable existing procedure that we can adopt, especially if it doesn't involve a loss of information. Similar tactics will be discussed as we get into prediction and risk profiling in the next chapter.

### ***3.6.3 Multidimensional Bins and Pivoting***

If you think that there may be multiple, independent, noncategorical measures that could meaningfully distinguish subsets of your data, make use of them. Let's say that you suspect certain firms – such as those that are simultaneously on the fringes of both (a) EPS performance and (b) research and development (R&D) investment – are relatively unique with regard to other aspects of their operations or financial performance. You have the ability to designate such groups independently, using either percentile or statistically based approaches as discussed. With those bins established, you can then consider the creation of cross-tabs to examine the aggregate constituencies at the intersection of these bins. Your approach might result in a summary descriptive structure similar to that in Table 3.2.

Table 3.2 Example structure of quartile cross-binning

		EPS (earnings per share)			
		Bottom 25%	Third 25%	Second 25%	Top 25%
R&D Investment	Bottom 25%				
	Third 25%		<i>Summaries of other measures (e.g., profitability) for firms that fall into each cross-tab</i>		
	Second 25%				
	Top 25%				

Multidimensional summary presentations such as this are simple but avoid assumptions regarding the performance landscape and can provide a great deal of insight into the effects of multiple factors on additional performance measures. Such insight might suggest further investigations into specific regions of the data (e.g., those firms in the third 25 percentile in R&D but the top 25 percent in EPS). Much as in the case of the data visualization tactics to be discussed in Chapter 5, don't pass up opportunities to look at your data from multiple vantage points. You might find something worth looking into further, or you might temper your convictions regarding other assumptions typically made.

### 3.6.3.1 Creating Pivot Tables

We've already breezed through filtering in general with the introduction of Excel's data Filter tool in Chapter 2. As we've discussed, that Filter tool is convenient but can be somewhat misleading. It is also somewhat limiting. In particular, it doesn't provide much insight into summary data relating to how multiple variables jointly impact other issues of interest. You might be able to filter along more than one variable, but joint summaries of observations that meet a range of joint criteria will not be simultaneously provided.

Pivot Tables provide an alternative filtering and data presentation approach that overcomes this limitation. Consider the Pivot Table that already exists on the Pivot Table sheet of the Chp3\_QuarterlyData workbook. Click on the table to view available options (shown in Figure 3.21).

The following describes the three mechanisms you can use to prune your presentation and analysis data using Pivot Tables.

- **Pages (global filters):** These allow you to restrict data presented in the table as a whole to certain cases (e.g., firms in the industrial equipment industry and

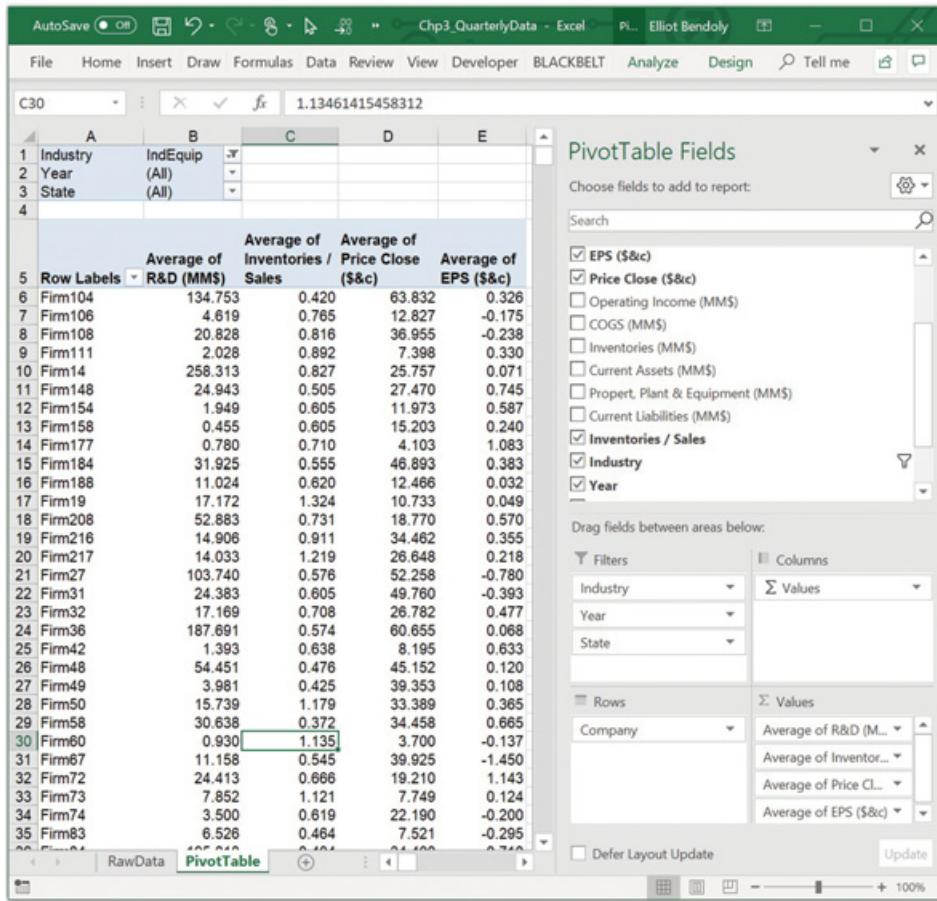


Figure 3.21 General structure of a Pivot Table interface

nothing else, students in the junior class and nothing else). Add a page filter by using the cursor to drag and drop a variable (e.g., firm type or student type) from the field list (shown in the right portion of Figure 3.21) to the Page Fields box above the Pivot Table. You can also drag and drop a variable to the Report Filter box at the bottom of the Pivot Table Field List.

- **Row and Column filters (local filters):** These allow you to restrict the data presented in specific rows and columns of the table to certain cases (e.g., each row provides summaries across a single state. Fifty rows of information would then supposedly be presented). Add a row or column filter by using the cursor to drag and drop a variable (e.g., firm type or student type) from the field list to the Row Fields box at the left of the Pivot Table or the Column Fields box on table header. You can also drag and drop a variable to the Column Labels or Row Labels box at the bottom of the Pivot Table Field List.
- **Data elements:** These allow you to designate what is actually summarized in the meat of the table. You might want to view summary statistics for any

number of issues, subdivided by cross-tab cases; for example, earnings per share, or spending on research and development, depending on location and size of firms. If you divide your table by placing location categories (e.g., state) in the Row Field box, the size categories (e.g., 100–999 employees or 1,000–9,999 employees) in the Column Field box, and earnings per share (or spending on research and development) in the Data Items box at the center of the table, you could get a glimpse of any potentially notable distinctions across these bins.

By default, Excel's Pivot Tables tend to pull data in as counts (i.e., how many pieces of data exist for the row-column combination), as opposed to, say, averages in the data selected for that combination. You can change this by, among other approaches, right-clicking on any header and selecting Value Field Settings. You'll have the option to specify what kind of data summary pops up at that point (see Figure 3.22). And if you are ever curious about the specific data records that form that numeric summary, double-click on that number and Excel will create a separate sheet for you to view, including all those records involved.

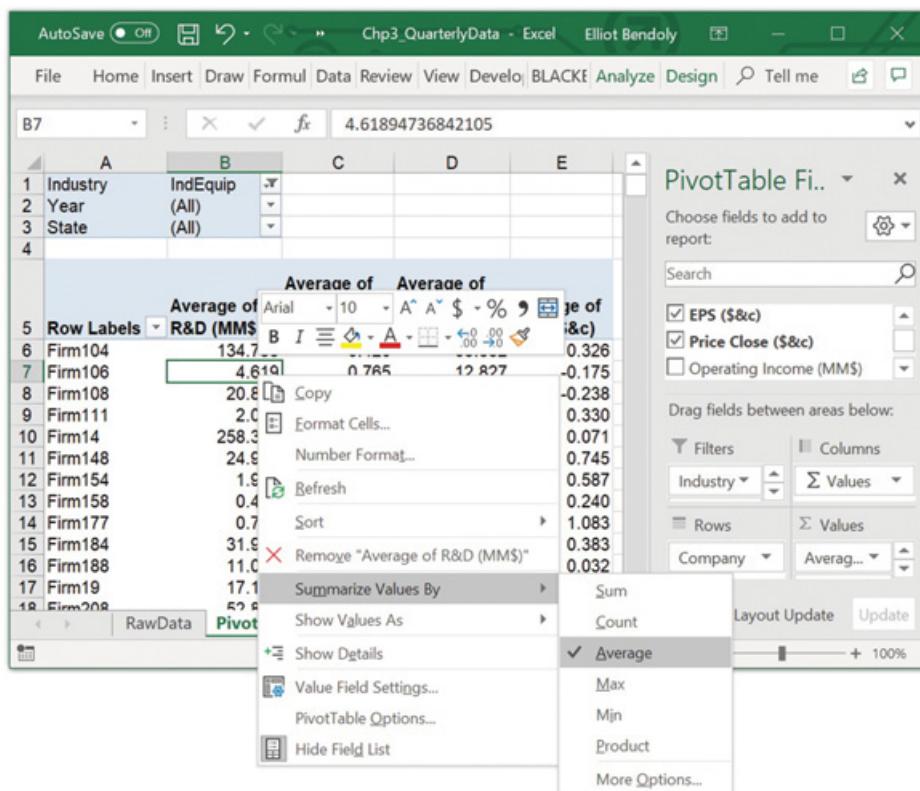


Figure 3.22 Modifying field settings in Pivot Tables

### 3.6.3.2 Filtering by Row and Column

Aside from limiting the entirety of the data viewed in Pivot Tables by what you put in the Data Fields and Page Fields boxes, you also have the opportunity to limit the number of rows and columns for which data are shown. For example, if you have placed a location category (e.g., state) in the Rows Field and are only interested in comparing specific locations (e.g., California, Oregon, and Washington, as opposed to all fifty states), you have the ability to do so easily in a Pivot Table. Such selective pruning of presented data can greatly increase the clarity of points you may be trying to make with the data. Click on the category listing you want (e.g., STATE-Name) and then check off the items for which you want data displayed (e.g., California). This is shown in Figure 3.23.

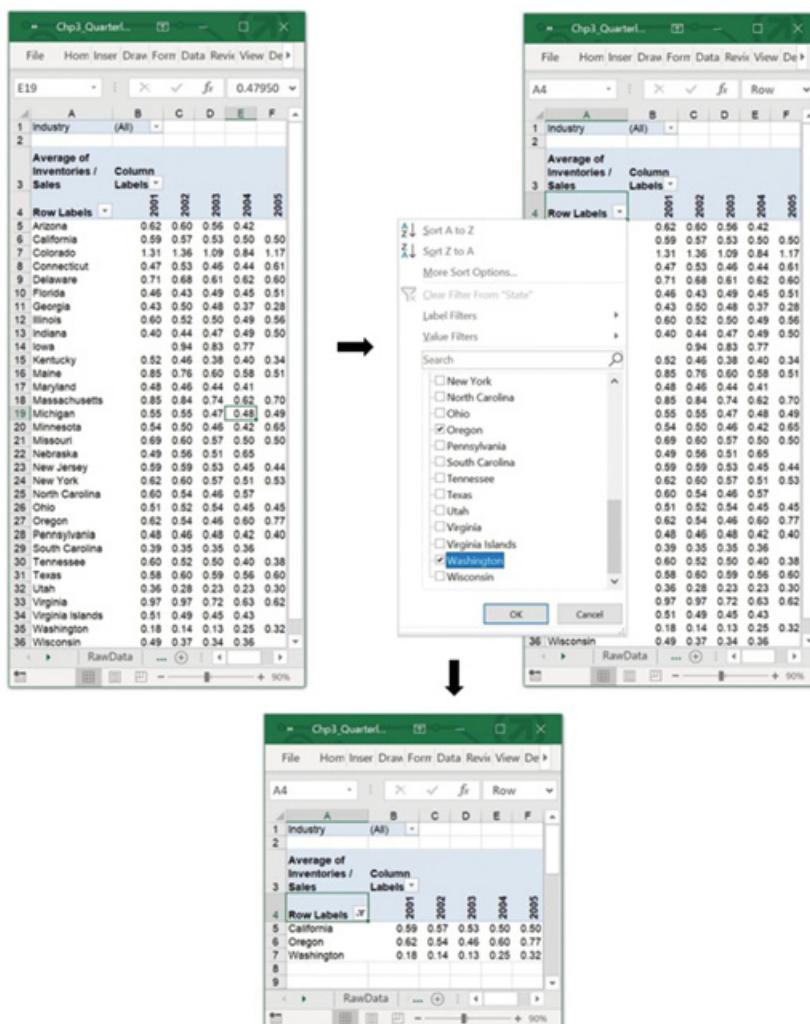


Figure 3.23 Pivot summary manipulation by selective filtering

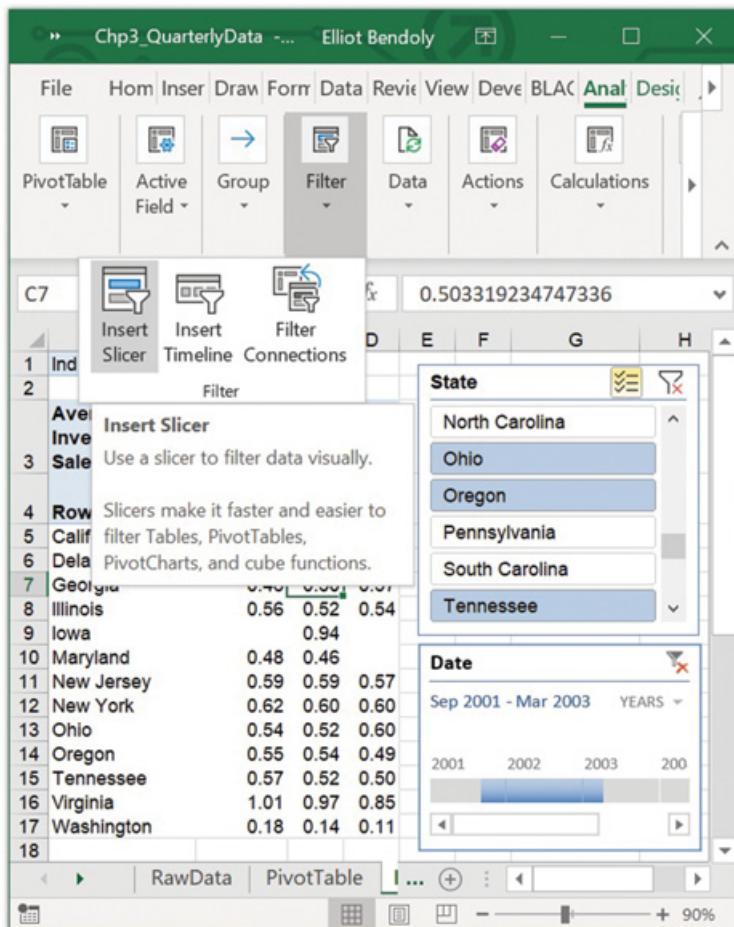


Figure 3.24 Slicer and Timeline filters with Pivot Tables

Still more versatility in the examination of slices of your data set is available through, appropriately, the Slicer features of Pivot Tables. Found in the Analyze tab, when a Pivot Table is active and in use, the Slicer and the Timeline options provide special filter options for the Pivot Table. The result is a much more accessible way to modify data aggregation and presentation (see Figure 3.24). We will consider additional embedded controls of this kind in Chapter 4.

### 3.6.3.3 Power Pivot for Databases

The natural extension of the Pivot Table concept focusing on spreadsheet data is the BI tool known as Power Pivot. Simply stated, while the Pivot Table tool is designed to work with single tables of data, Power Pivot is designed to be used with databases consisting of multiple tables connected by shared key fields. If you have a connection to a database through the Azure

Table 3.3 *Example subpopulations associated with quartile cross-binning*

		EPS (earnings per share)			
		Bottom	Third 25%	Second 25%	Top 25%
R&D Investment	Bottom	2	8	9	5
	Third 25%	4	5	7	9
	Second 25%	8	6	5	6
	Top 25%	11	6	4	4

cloud service, an internal company instance of Access, or something else of the sort, the Data ribbon permits connections to these (with credentials) just as it permits connections to delimited text files and web pages. Once connected, if the key fields connecting database tables are accurately described, Power Pivot permits the use of data from one table in the database to serve as the criteria for filtering or aggregating data from a second. Using Power Pivot is no more difficult than using Pivot Tables in spreadsheets. The hardest part, frankly, is making sure you have the correct credentials access and ensuring table connections are correctly defined.

#### *3.6.3.4 Caveats to Pivot Table Summaries*

As powerful as Pivot Tables may be, there are some warnings that must be given with regards to their application. Although the number of data records that fall into each of the four categories for either the EPS or R&D splits may be equal, there's nothing that will guarantee that the number of observations in each cell of a cross-tab will be similar. Consider the example in Table 3.3, where I've entered the number of records that fall into each cross-bin. There are big differences in the sizes of these bins. Does this make it more difficult to compare performance measures for some of the smaller groups? Should we be concerned if there are only, say, two observations that an average performance calculation is based on? Do we know how many observations are behind the averages our colleagues use when making their arguments?

#### **3.6.4 Cluster Analysis for Multidimensional Splits**

Now it's quite possible that you have data that can be split across multiple dimensions but you don't think it's a situation that the above binning and cross-tabbing can help with (at least not directly). Maybe you feel that natural splits in your data occur in more complex ways, with the separation along one attribute or component somehow dependent on the level of

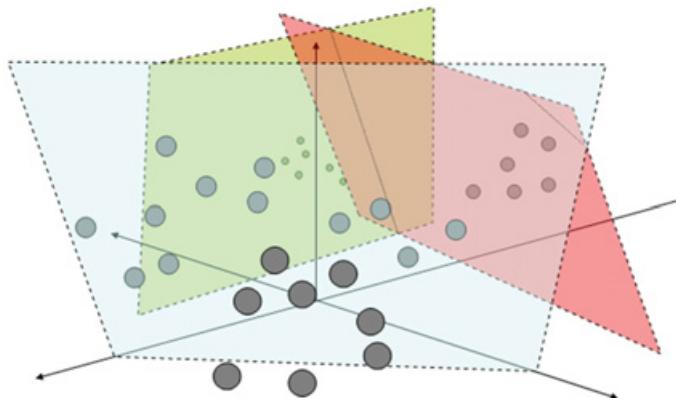


Figure 3.25 Multidimensional complexity in group distinction

another. The problem is, you just don't know what that relationship is up front and you don't want to just guess at it.

To try to illustrate this common situation, consider a bunch of points characterized by three attributes: X, Y, and Z. Upon visual inspection of this three-dimensional space (perhaps using tactics such as those in Chapter 5), you might see these attributes roughly cluster into four groups. Maybe you could even sketch out some dividing planes, superimposed on this plot, to emphasize those separations. Confidence in this ad hoc work could be shaky, though. And what if the clustering was less clear? What if there were more points, or more dimensions, or multiple ways to subjectively split them up? (See Figure 3.25.)

Consolidating observations through nonobvious (although perhaps statistically supported and managerially meaningful) divisions of their attribute space usually falls into the realm of what is called cluster analysis. This technique (broadly) for grouping data is popular in marketing, where firms attempt to classify specific groups of customers based on a wide range of characteristics (attributes). To get a sense of how this method is used, suppose you collect data on individual entities such as:

- Customers of a business
- Stores of a retail chain
- Students of a university
- IT implementation projects (as in the Dodecha case)

And let's say these individual entities can be characterized by a range of specific attributes or higher-level factors such as the following:

- For customers: age, monthly income, loyalty, tech savvy
- For stores: geographical location, sales, market presence, efficiency
- For students: grades in different courses, determination
- For projects: anticipated costs, interorganizational concerns

To make these data useful, a decision maker or manager might want to identify smaller groups of individuals based on their similarities to one another (i.e., the similarity of their shared attributes) and then make separate policy decisions for each group, rather than trying to come up with an all-encompassing and perfect solution for everyone. What kinds of distinct policy decisions might be realistic?

- Designing and deploying different marketing campaigns for different market segments of customers characterized by distinct interests
- Designing and deploying different business and operating strategies to promote sales in different groups of stores facing different challenges
- Designing different courses, programs of study, or sources of assistance based on strengths and weaknesses of different groups of students
- Designing response plans for dealing with (or rejecting) specific projects requested by existing or future clients

The key to applying customized strategies is being able to know for whom or what you are customizing (i.e., the nature and distinction of the group). Conceptually, the decision maker or analyst must be able to make these distinctions, often prior to designing customized policies. This conceptual goal translates into the following analytical task of cluster analysis.

**Objective:** Identify a set of fairly distinct groups of entities (records) in the data set, based on some measure of group separation (e.g., minimizing the ratio of within-group to between-group variance, with variation of the entire sample based on the full set of record attributes used for grouping).

**Utility Variables:** Which entities or records should belong to which group.

**Connections (Constraints):** Often some criteria for limiting the number of groups that might be formed (e.g., limit to the formation of four clusters of projects as in the case of Dodecha Solutions, Ltd.).

That is, the algorithms used in cluster analysis are designed to locate clusters of data records that possess similar characteristics (whatever those attributes might be), which are distinguishable from other clusters and ideally have few records in between (where group membership is less clear).

Let's consider a concrete example in which we can apply another tool from the Blackbelt Ribbon arsenal: Simple Cluster. This tool provides the bare minimum when it comes to clustering; but if clustering is new to you, that's not a bad place to start. Much more sophisticated clustering tools are made available in packaged form through tools such as Palisade (see the drop-down depicted in Figure 3.26) as well as through calls to available code (e.g., R). More fundamentally, if the principal objectives, utility variables (relevant actionable decisions), and connecting constraints outlined here are consistent, the mechanism used in Simple Cluster can be easily augmented to take on a more nuanced role with a little extra code development. Once

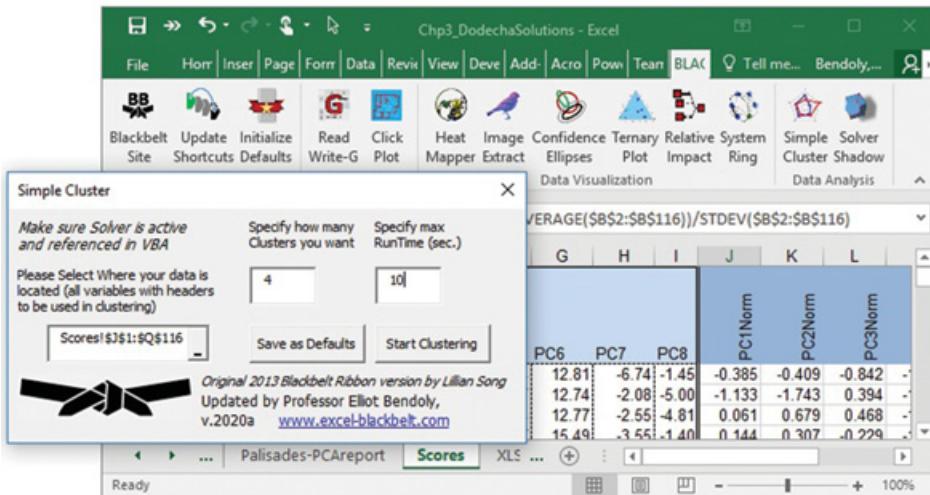


Figure 3.26 Accessing k-means clustering inputs and parameters in Simple Cluster

you know what you want to do and what the results might look like and once you familiarize yourself with just a little coding (as in later chapters), it's worth taking advantage of.

With just the Simple Cluster tool in hand (no coding for now), let's reexamine the Dodecha Solutions case. We will capitalize on some of the attribute grouping work conducted in the earlier PCA analysis, focusing on eight rather than 32 distinguishing traits. In contrast to the aim of the earlier PCA analysis, the main takeaway from the cluster analysis effort will be grouping of projects (rows). These are what policy makers are looking for in this case in that they would like to classify future projects and effectively assign them to the most fitting project teams and leaders. Specifying the data range, the number of clusters to be sought out, and the amount of time you are OK with allowing Excel to spin in its search for clusters, you can activate the search. After the specified duration of time is spent, you have the option to continue the search or accept the result derived. In this case we've just run for 10 seconds, looking to place the 115 observations into four clusters. The results that one might encounter, dependent on your computer's processing speed, are captured in Figure 3.27. The plot is automatically generated by the add-in, but I've also introduced a Pivot Table showing the averages of the eight components for each group. I've also modified the colors a bit and introduced conditional formatting to highlight differences in these average component levels.

Clustering results are only as good as the intelligence brought to bear in their use. Assuming the clusters themselves are fairly robust to other clustering methods that could have been applied, at least two sets of immediate questions typically come into consideration:

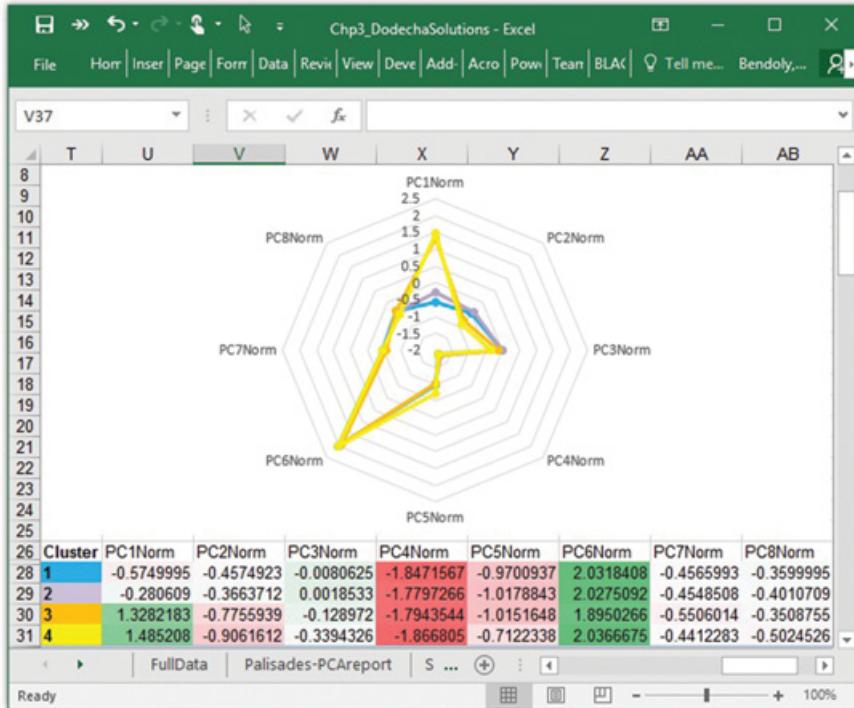


Figure 3.27 Clustering results (classes/groups assigned) from k-means approach

- 1) How do the groups differ across the set of factors upon which they were based?
- 2) Are only some of these factors critical in distinguishing these groups?

From looking at the results in Figure 3.27 it seems that not all of the eight factors have been very helpful in distinguishing these groups; our conditional formatting doesn't suggest differences in the averages beyond the first few components. Unfortunately, the existence of less distinguishing factors tends to deemphasize the distinctiveness of the derived groups as depicted by measures such as the ratio of within-group to between-group variance. (This ratio will tend to go up as more factors are included that don't contribute to group distinction.)

However, we don't need to be thrown by a single measure taken out of context. We also don't need to be dogmatic about the number of clusters that we feel must be discovered. We might do better to focus on distinctions that are salient in the data set. For example, we might derive more meaning if we focused on just a few key factors – maybe the first three, along which the greatest differences exist – and reduced the total number of clusters to three as well. If we rerun this analysis with this focus, we see results more along the lines of those captured in Figure 3.28.

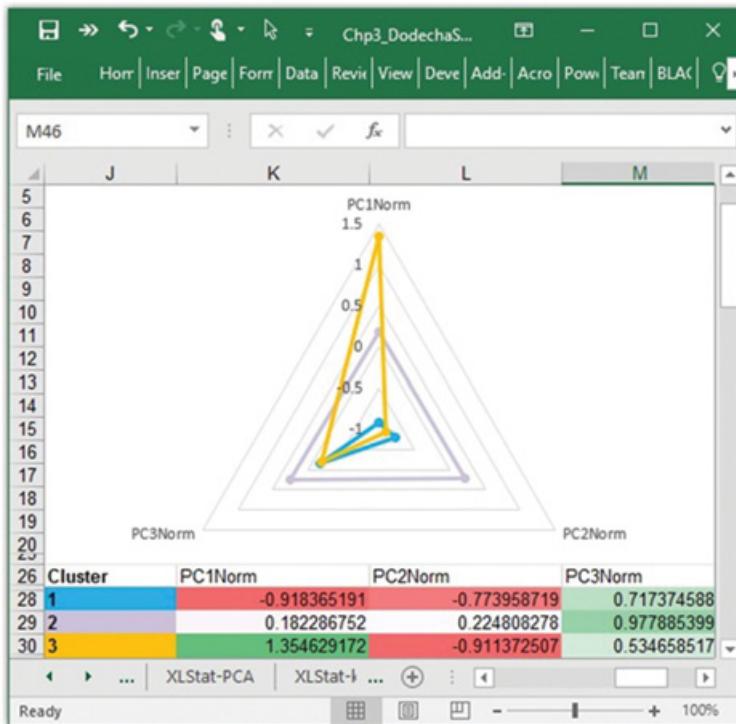


Figure 3.28 Clustering results from a reduced component set

Although the story isn't a simple one, the results do support a fairly clean multifactor tale of how the sample might be split into three groups. Projects in Cluster 1 tend to have lower values of PC1 and PC2, components that are also low when legacy system concerns are high and client concerns are low. In other words, Cluster 1 projects appear somewhat distinct in that they seem to be ones in which the legacy technology landscape is complex (attributes F) but where client management and technical know-how are not barriers (attributes G). Projects in Cluster 3 also have a low PC2 score but are high in PC1. Looking at attributes that PC1 and PC2 draw on in an opposing manner but which are more in line by these contrasting results, Cluster 3 projects appear to face fit issues relative to processes in place (attributes A) but few challenges in the way of organizational complexity (attributes H). Cluster 2 presents middle-ground levels of PC1 but higher levels of PC2 and PC3 than the other clusters. This might suggest that such projects face considerable issues in Dodecha's existing expertise (attributes E) and may suggest the benefit of future training or talent outsourcing.

These are nuanced interpretations and ones that would need to be further scrutinized. And in practice, this investigation certainly wouldn't stop there. For

instance, there is no need to limit our consideration of group distinctiveness to preproject data, nor should we. After all, these are supposedly completed projects. Postproject performance data would not be available for categorizing future projects into such groups, but Dodecha would surely be interested in knowing how these groups might ultimately differ in performance as well. All this leads to our natural follow-up question for analysis: How do the groups differ across other related outcome or performance measures that might be of interest to managers? In the presence of objective performance data (ROI, Time-to-Completion, etc.) we could begin to consider these questions as well – outstanding and stimulating questions, albeit ones that will have to wait for the discussion of predictive analytical approaches in the next chapter.

### **3.7 Caution with Data**

We will forge ahead in the following chapters with discussions of prescriptive and predictive model development, execution and interpretation, and automation as well as ways to make this work and its results accessible and intuitive to our end users (who may include ourselves). Our assumptions beyond this point will generally be that the data we have is clean and accurate – that the aggregation and filtering that has been applied has not lost critical content; that transformations of the data have not mangled the interpretability of original observations. Further, we are going to assume that the data collected can be corroborated, comes from reliable sources, and has faithfully and carefully been transcribed and curated.

In reality some of these assumptions can be more extreme than others, depending on the context. In an age where individuals without any credentials whatsoever are free to make claims that are entirely without foundation and are accepted as fact by those seeking cheap self-confirmation, there is reason to be careful. There is a lot of data out there. But if you are going to use it, make sure you know what it is that you are using. Confirm. Triangulate. Scrutinize. Be open to alternative sources but be thoughtful about it. Put more weight on evidence provided by credentialed and verified sources. Put less weight on data from sources that repeatedly are shown to be incorrect, lack support, or just always seem to be in contrast to the perspectives of professional experts – even if that means a contradiction in your own assumptions. Be open to that and you'll go right in your analysis and decisions far more often than wrong.

### **Supplement: Unique Data Generation Hacks**

We will soon learn more about how data examination, and by extension the search for flaws and nuances in data and assumptions, can be assisted through a range of visualization tactics. Visuals, however, are not just an

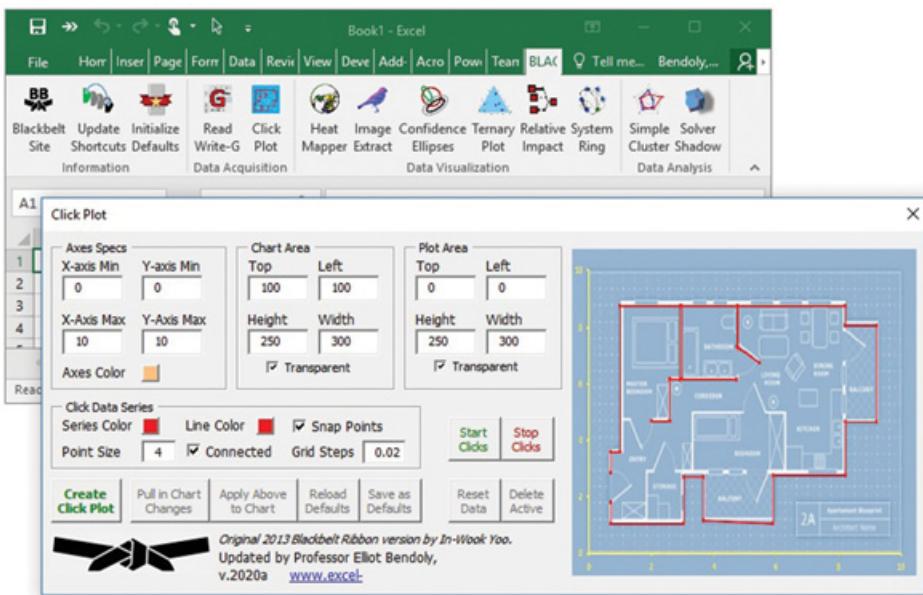


Figure 3.29 Access Click Plot as a means for visually prompted data extraction

“outcome” of data collection efforts. They also, in themselves, can facilitate data development. As a case in point, consider the Click Plot tool in the Blackbelt Ribbon add-in. Click Plot allows individuals to effectively generate data points by moving their mouse and clicking on targeted locations superimposed on imported graphics. Access to Click Plot is found alongside Read Write-G in the data acquisition section of the ribbon. The interface options available for starting a data extraction process, including positioning, color, and grid-snapping, are presented in Figure 3.29.

The process begins with making sure that you have a template image to work with. You can paste any image of interest into the spreadsheet environment to this end. Clicking on the Create Click Plot button then initializes an empty and transparent plot of the dimensions specified in the form (axes specs, chart area, plot area, etc.). You are free to either change these fields or resize that plot manually, shift your plot to ensure it overlays your imported image, save your changes as default, or reset according to saved defaults (which exist on hidden sheets unless revealed). When ready to begin clicking on various points of interest in the overlaid image, hit the Start Clicks button. You will be asked to click on a couple of points for calibration purposes. After that, each subsequent click will result in a new data point being displayed on the plot and the relative position of that point being recorded in the leftmost columns. Figure 3.30 presents the results associated with the mapping of vertices in an example pasted blue print.

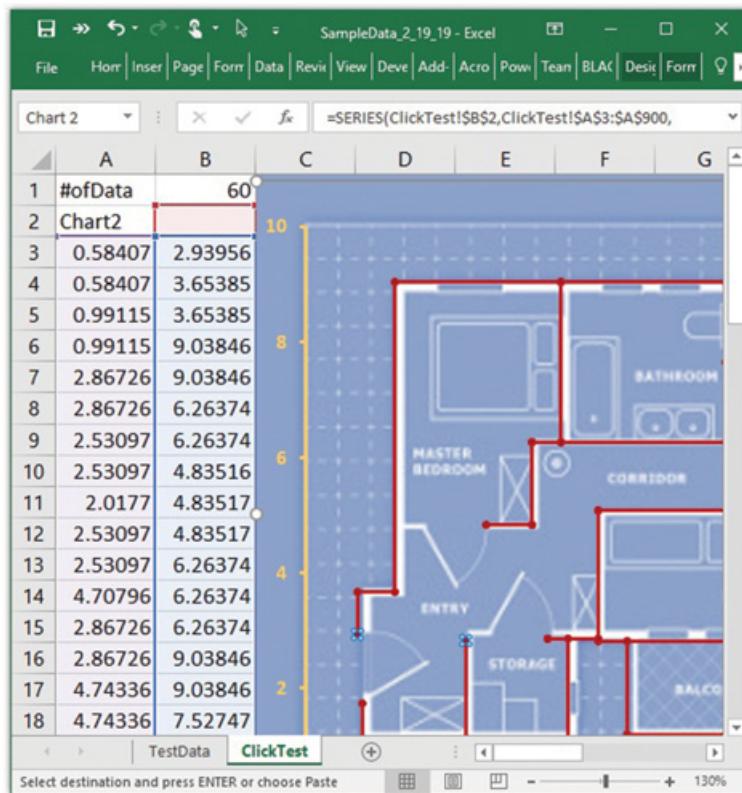


Figure 3.30 Data extraction results from the use of the Click Plot tool

This technique can be applied as easily to graphics of stock-price charts, in lieu of tabular data, as to maps of geographic locations and facility layouts. The result is not just a relatively positioned set of data but also the foundation of what might become paths for analysis or further visualization.

Now, if you prefer to use drawing tools to capture data, translating the vertices of drawn polygons into relative data sets, you're in luck. The Blackbelt Ribbon comes with several additional tools that can accomplish this as well. Once active the ribbon allows use of the PolyPtsExtract function, which takes the name of a drawn polygon and destination-start cell as inputs and returns a textbox containing the relative coordinates of the vertices of that polygon. Clicking on that textbox will transfer them to the destination cells specified. To accomplish the reverse (build a polygon based on X and Y data), the PolyPtsBuild accepts a start cell and a scale factor. And if you ever wanted the length of the path comprised of a set of sorted points, regardless of whether they are in just two dimensions or across many

more (e.g., X1 through X10), the PathLength function will convert that set of sorted points into a total Cartesian length of path calculation (the final parameter should be TRUE if data is ordered vertically). PathLength results can be useful for calculating relative distances and perimeters.

The principles behind these additional functions are also key to two of the visualization tools that we will discuss in Chapter 5: Heat Mapper and Image Extract. We'll learn about another special Blackbelt function that can help us when we are ready to consider depicting data "by area" through the use of these tools. And eventually we will be designing our own User Defined Functions and Userforms to accomplish other complex tasks. So stay tuned.

## PRACTICE PROBLEMS

### Practice 3.1

Examine a possible source of online data, such as *The New York Times* job board (e.g., <https://jobs.nytimes.com/jobs?startindex=25>). When you have sources with multiple pages of content, the associated URL often provides detail regarding which part of the larger set is being displayed. In this case the last two numbers represent the index of the first item on that page. Try specifying a city for your search as well to see how the URL changes. What is the URL for Columbus, Ohio? What is the URL for San Antonio, Texas? On separate pages of a workbook, use a location-specific URL to create a Web Query, trying each of the tactics described in this chapter (the legacy approach, the Read Write-G add-in, etc.). Do not set these queries to automatically refresh. Pick any one of these results and delete the rest.

### Practice 3.2

Find an online source of data that updates more frequently (stock data, sports data, weather data, etc.). Make sure that a Web Query of some kind can actually pull in recognizable numerical data and then set up a living record collection mechanism along the lines described in this chapter to collect unique observations with associated time stamps. Set the autorefresh rate at one minute and allow this to run for an hour. Once the data is collected, copy your record and paste on a separate sheet "as values." Ensure that the time stamps can be read as data-formatted. Add a column that provides a normalized value of your data. Apply a Pivot Table to your static set of data to filter out observations that are more than one standard deviation from the mean. Try applying a Timeline or Slicer to further filter your collected data.

### Practice 3.3

If performance differences are so critical, one might ask, "Why not just group by performance to begin with and then see what premeasures might be useful in

predicting membership in performance groups?” Aside from getting into issues related to potential robustness in the results, the fact is that such seemingly straightforward approaches, as simple as they may be, often don’t yield very strong mechanisms for future prediction (and may turn out not to be all that useful).

Regardless, using the existing Dodecha Solutions data set, give this a try. Use some of the original 32 attributes and either the Palisade StatTools clustering device or the Blackbelt Simply Cluster tool to extract three groups. What differences do you see relative to the earlier derived results? Suggest the pros and cons associated with this additional analysis and interpretation.

## **Section 2**

### Structuring Intelligence

# 4

## Estimating and Appreciating Uncertainty

**Objective:** *Gain an understanding of how observations can vary, how to estimate effects and account for some of that variation, and how to incorporate variation not otherwise controllable into modeling and risk profiling efforts.*

With data in hand, we might be feeling pretty good. We have something to work with. And if we think we have more than we need, we now clearly have a few ways to simplify things. But every reduction of data comes with risks of loss. The motivation for some reduction efforts (e.g., rank conversions) can seem more dubious than for others, but essentially any choice we make to reduce massive amounts of data into something that informs action has its pros and cons.

So, what steps can we take to help ensure that we are, in fact, providing appropriate characterizations of reality when we begin to squeeze our data for the insights we need? Perhaps the most important question to begin with is not “what’s the average” value of an attribute captured in my data but rather “why does it vary” across observations. After all, if your data didn’t vary, we’d have no need to calculate averages. Every value would be the average, the maximum, and the minimum.

Why does our data vary? Well, let’s put this in context. Consider some fairly common kinds of data that we might encounter in our work:

- the amount of demand we need to cover in the next few days
- the amount of time it takes for a worker to serve a customer
- the nature of the transportation infrastructure and traffic along it, on which we base cost and time estimates in routing
- the rate of return on stocks and other options in portfolio selection
- the actual cost to complete a project or new venture we may be considering (among a set of other investment options)

These can be critical pieces of information on which to base decisions. But are they captured by only one number? Some “average” that we can be sure of today? Many of these issues can only really be observed in the future – that

is, they haven't actually happened yet. We have to make decisions in anticipation of those values. Is it possible that an average calculated based on past data might be different from observations closer to the point at which our decisions are played out? If there are shifts over time, how frequently should we expect them to occur? How big can these differences get? What actually drives these differences?

Ultimately, most real-world data can be described not only by one or more central characteristic values (a mean, a median, a mode, etc.) but also by the degree to which it can vary. The latter can be extremely complex. We can talk about the distribution of values around some central estimate, using summaries like standard deviation or coefficients of skewness (e.g., relevant when the mean is notably different from the median or mode). We can also talk about the extent to which that variation can be accounted for by other factors. In other words, not only can we talk about the uncertainty associated with something we need to base a decision on, we can work to reduce the unexplained uncertainty through some intelligent prediction. In doing so, we set ourselves up for smarter prescriptive decisions that control some of the issues that contribute to risk, reduce our exposure to risk that we don't control, and pave the road for contingency plans that accommodate our expectations of remaining risk.

Appropriate and robust prediction is therefore crucial in taking us beyond simple description. It represents a shift away from calling all of the complexity in the world around us “noise” to understanding the world as a system of interrelated and interdependent elements and processes, stocks and flows, constraints, and, yes, some background noise that we simply won't be able to account for. We seldom have the data on all of the factors in the systems we make decisions within and the relationships among these factors (one might say their functions) are often far too complicated to comprehensively model. So there is bound to be some variation we are just going to have to end up treating as unpredictable. But we can at least try to capture some of it, and there are plenty of approaches and tools to help us.

## **4.1 Prediction and Estimating Risk**

Many convenient modeling tools come standard with Excel. Some exist but are not active until you specify that you want them to be. These include the common add-ins such as the Data Analysis tools, which we have seen already, and Solver, which we will spend time on in Chapter 6. Other tools don't come with Excel but can be acquired from external sources. These again include packages such as the Palisade suite, JMP, R, SPSS, and others. As in our discussion of PCA and cluster analysis, the value of each of these tools varies depending on what you are attempting to accomplish.

The lack of free access to some of these tools, the challenge of getting started in others, data processing limitations, and the often static nature of results generated are each common drawbacks that differ by tool. Fortunately, many common statistical processes that help in our prediction efforts are standard and will yield equivalent results irrespective of which package/tool you choose to use.

### 4.1.1 Regression Using Data Analysis Tools

As an example let's consider the regression capability of the Data Analysis tools add-in and then consider an alternative approach available in Excel that yields equivalent results. We'll take another look at some data we considered earlier when exploring data tables. In this case we will reference the Chp4\_QuarterlyRegression file to demonstrate a couple of approaches. Opening this file, we will again access the Data Analysis tool set on the Data ribbon (if you don't see it, just go to File>Options>Add-ins to ensure this add-in is active). We will specifically be selecting the Regression tool from this set (see Figure 4.1).

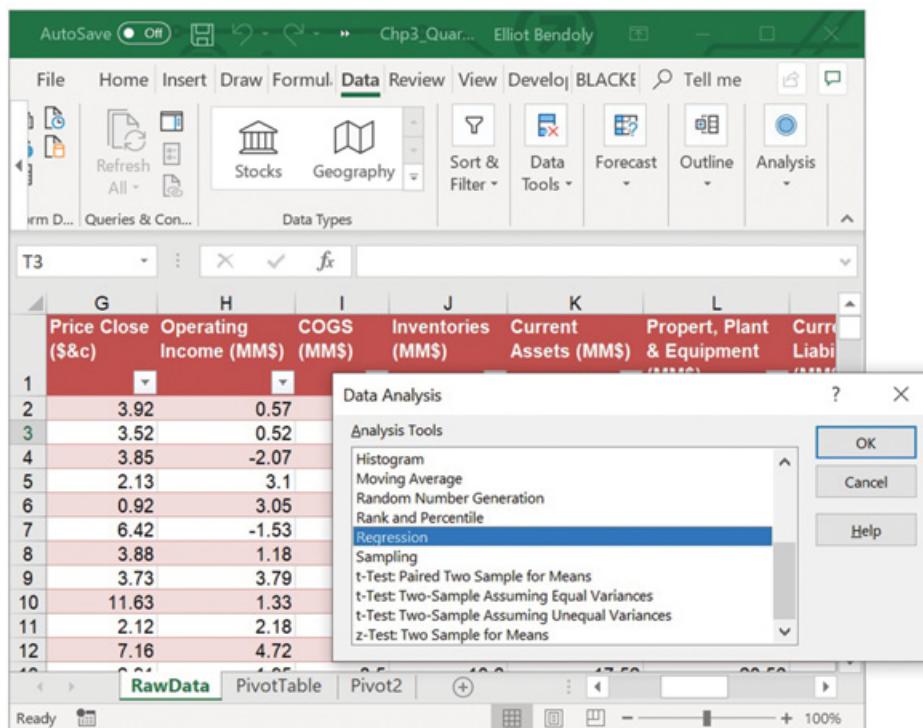


Figure 4.1 The regression option in the Data Analysis tool

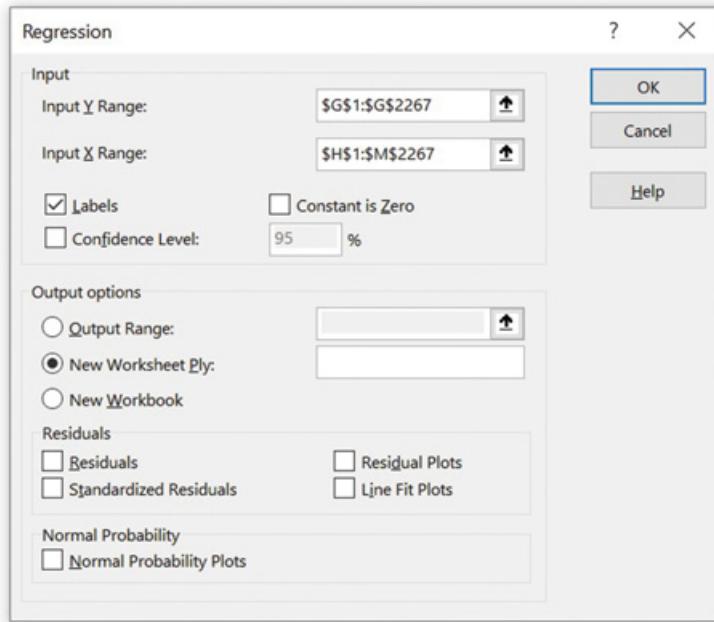


Figure 4.2 Specification of regression inputs based on structure of spreadsheet

Choosing this option provides a pop-up screen, as shown in Figure 4.2. It allows you to specify a dependent (Y) variable and any number of X variables that you might want to include in a regression. If we're interested in whether or not the end-of-quarter close price of a publicly traded stock (Price Close) was somehow a function of other activities in that quarter (operating income, cost of goods sold, etc.), we might select the data in the Price Close column as our Input Y Range and the data in the other columns as our Input X Range. We are allowed here to specify whether the first cell of these ranges includes a header label.

Setting aside whether we were justified in selecting the specific variables as predictors of our Y and ignoring the potential for much greater insight if we controlled for the firms involved (or at least scaled price and other factors relative to within-firm averages), the resulting output, shown in Figure 4.3, is still fairly rich, especially from the perspective of the needs of a junior analyst looking for a quick snapshot of the presumed data relationships.

Let's take a look at some of the details. First, we consider the R Square value. Its value of 0.0615 gives us some sense of the amount of variability in stock price (our Y) accounted for by differences across the predictors (our Xs). Something like 6 percent of variance accounted for may seem small, but it's not uncommon for linear regressions grounded in specific arguments. It might be low for exploratory modeling or for models that

The screenshot shows an Excel spreadsheet titled "Chp3\_QuarterlyDataRegression - Excel". The data is organized into several sections:

- SUMMARY OUTPUT** (Rows 1-8):
 

4	Multiple R	0.2480
5	R Square	0.0615
6	Adjusted R Square	0.0590
7	Standard Error	19.5579
8	Observations	2266
- ANOVA** (Rows 10-14):
 

11	df	SS	MS	F	Significance F
12	Regression	6	56604.843	9434.140	24.664
13	Residual	2259	864095.492	382.512	
14	Total	2265	920700.335		
- Coefficients** (Rows 16-24):
 

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%	
17	Intercept	26.5751	0.4754	55.9042	0.0000	25.6429	27.5073	25.6429	27.5073
18	Operating Income (MM\$)	0.0038	0.0012	3.0981	0.0020	0.0014	0.0062	0.0014	0.0062
19	COGS (MM\$)	-0.0007	0.0004	-1.8835	0.0598	-0.0014	0.0000	-0.0014	0.0000
20	Inventories (MM\$)	0.0222	0.0007	3.2446	0.0012	0.0009	0.0030	0.0009	0.0036
21	Current Assets (MM\$)	-0.0005	0.0002	-2.2325	0.0257	-0.0009	-0.0001	-0.0009	-0.0001
22	Property, Plant & Equipment (MM\$)	-0.0001	0.0002	-0.5998	0.5488	-0.0006	0.0003	-0.0006	0.0003
23	Current Liabilities (MM\$)	0.0010	0.0003	3.1845	0.0015	0.0004	0.0016	0.0004	0.0016

Figure 4.3 Sample output from the Data Analysis regression tool

take more complex nonlinear relationships into account but it can suggest some powerful insight nevertheless. To that end let's look at the impact that each of our X variables seems to have on Y.

At this point many junior analysts will focus on two columns in the lower table of Figure 4.3, specifically those labeled Coefficients and P-value. The estimated coefficients provided in this table are estimates of how much Y might change with each unit increase in each X (i.e., increasing the value of a given predictive variable by +1). In this case our coefficients seem pretty small (average scale in the ten-thousandths) but then again our X variables are pretty big (average scale in the thousands). The scale of the impact that one X predictor can have on Y is important but so is clarity around the specific value of this impact. The size of an estimated coefficient can appear large but it is still just an estimate. And all estimates are surrounded by the potential for error (the Standard Error column gets at this), some levels of which make our ability to bank on those effects tenuous (i.e., when P-values are greater than benchmarks like 5 percent, as is the case with PPE in Figure 4.3).

#### 4.1.2 What Regression Is Doing

Where did these estimated coefficients, their estimated levels of error, and significance come from? How did the tool come up with calculations of R square and Adjusted R square (which provides a deflated measure based on the number of variables relative to observations available)? Stepping

back, we find it's worth understanding a little about what is actually going on behind the scenes in simple (multivariate) linear regression.

Ultimately, regression is simply an attempt to assign coefficient values ( $\beta$ ) to predictive variables (X) with the intention of minimizing the sum of squared differences between all of the observed values of Y and the corresponding summed products of the predictors and those coefficients ( $\beta X$ ). Here, just as Y represents multiple rows of data, X represents multiple rows and columns of predictor data (depending on selection) and  $\beta$  has as many values as there are predictor variables selected. Typically we also include an intercept (essentially a hidden column of X full of the value 1), with an additional  $\beta$  to represent its impact. As earlier, we can view this task in three parts:

**Objective:** Minimize the total of squared differences between the individual values to be predicted (Y) and the overall fitted equation attempting to predict them ( $\beta X$ ) (i.e., minimize  $\Sigma(Y - \beta X)^2$ ).

**Utility Variables:** The coefficients  $\beta$ .

**Connections (Constraints):** Strictly optional and various depending on need.

Notice that unlike some of our earlier examinations, there is no specific constraint being listed here. We could require or omit the presence of an intercept, though in our above procedure we have included it (it is a significant value at around 26.75). We could require all coefficients to be strictly positive if we had good justification to do so, though we had better have very good justification. Ultimately every constraint that we might introduce to this process will, as constraints always do, hold us back from doing better in our Objective. The less constrained we are, the greater the fit between the estimation and the actual data and the greater the R square.

The coefficients are simply a means to that end, and their error levels are simply artifacts of this process. We are not forcing coefficients to become significant, but the search for coefficients that help us use all of our X data to predict Y will involve looking for ways to make the most of that X data; minimizing the difference between actual and predictions de facto involves the search for accurate coefficients of effect. And we are left with additional intelligence that we can use to control for some of the variability in Y, insights into the decisions that can be made to push Y in specific directions, and better estimates of remaining variability.

### **4.1.3 Regression Using LINEST**

Now, what drawbacks do we find from our use of the Data Analysis tool? First and foremost, the results generated are not responsive to changes in the

original data. This is an artifact of the Data Analysis tool and of many other tools as well. It is by design. When faced with increasingly large sets of data for predictive analysis, you simply don't want the analysis always being rerun since it can tie up a fair amount of computational processes. When data sets are not so large, however, it might be particularly convenient to have analysis that adjusts to cleaning and editing performed on the data, or analysis robust to trivial changes in the set of variables and records considered. Why? It's because predictive analysis is seldom the end game. It is often a feeder into subsequent prescriptive analysis. For that reason, it's handy to have a simple way to ensure real-time updating of analysis without having to worry about coding or the use of third-party mechanisms.

And there are in fact other, more dynamic capabilities in Excel that allow for similar results. One of the most convenient is a function called LINEST. Unlike some of the previously discussed functions that provide single outputs into single cells, the output of LINEST is rather special. Specifically, the output of LINEST spans multiple cells, when used appropriately. The nature of this output is akin to that of other matrix functions such as MMULT, which allows two-dimensional matrices to be multiplied by one another, the result of which may be more than a single cell of values. In order to use LINEST to regress an array of Y values (e.g., A1:A100) as a function of up to fifteen X values (e.g., B1:P100), the following could be entered into an anchor cell:

=LINEST(A1:A100,B1:P100,TRUE,TRUE).

The third and fourth parameters specify whether a constant should be included in the model estimation and whether all statistical details of the regression are desired. The value TRUE can be replaced with 1 in both cases, since they are equivalent, as discussed earlier.

The result will be a single value in that cell, at least to begin with. However, if that same cell is selected as the start of a larger range selection (multiple rows and columns) and Ctrl-Shift-Enter is hit while the cursor is in the formula bar, a fairly complete set of regression estimates becomes visible across those cells. Figure 4.4 provides an example of LINEST applied to the earlier regression. (I've renamed range G2:G2267 as PriceData and H2:M2267 as X\_Data, so I've excluded the nonnumeric headers.) In this figure I've shaded the outputs provided by LINEST (which occupies the range C4:I8). I've also added labels to the right and left of this range and just above that range to help with our discussion. Also note the curled brackets around the LINEST function as it appears in the formula bar. This denotes that a matrix function is being used.

To be clear, this isn't a simple output structure to interpret. As I said, I had to add those labels. Also, if you have more than one X variable, you are going

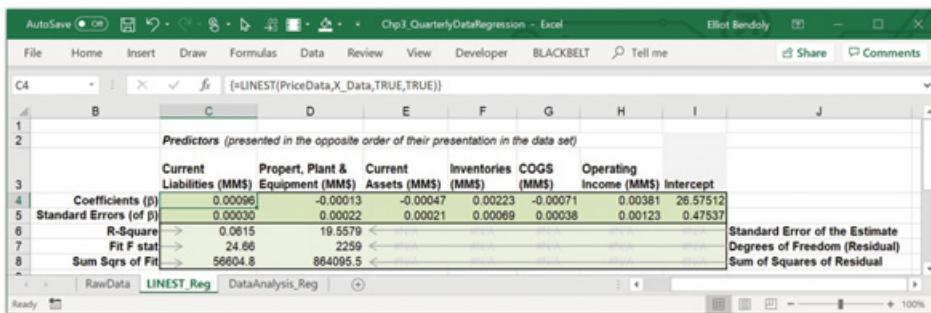


Figure 4.4 Sample regression output from the LINEST function

to get a range of output cells containing “#N/A.” Finally, it has a strange quirk: the estimated coefficients and standard errors around those coefficients are presented in an order opposite to the appearance of variables in the original data set. But if you can get beyond this, there are definitely virtues to the convenience and flexibility this function offers.

Now let’s compare the results to the Data Analysis regression results (back in Figure 4.3). Top two rows in the LINEST output (Figure 4.4) should match the coefficient and SE estimates in Figure 4.3. They do. The R-square in the third cell of the first column of LINEST should match the R-square in Figure 4.3 (0.0615). It does. Linear regression is linear regression. These are not two different statistical methods that we are seeing; they are just two different tools and two different presentations. The results tell the same story.

The nice thing is that when any of the original data is changed, LINEST, as a live function, will react to those changes and change all the output cells accordingly. That’s considerably more flexible than the Data Analysis function, albeit less straightforward for the casual user to initiate. Most methods have trade-offs, but flexibility usually wins out over an analyst’s need to decode a bit. What if you created a separate sheet where you allowed an individual, perhaps yourself, to select a specific set of six predictor variables simply by typing their names, using MATCH to find where they existed in the original set, and employing OFFSET to pull their data into that new sheet? If LINEST was based on the data appearing in that new sheet, you’d have the means of exploring the fit of multiple subsets of predictors, allowing LINEST to simply adjust its output automatically every time the choice of predictors changed.

Now that can appear a bit exploratory if there is not a hypothesis involved, backed by an argument that one subset will prove more effective than another. In either case interactive flexibility is a good thing in general. Also keep in mind that, regardless of the tactic used in analyzing data, pursuing a single approach to analysis can be a recipe for disaster. Taken alone,

a single set of results can misrepresent patterns among otherwise unrelated random numbers. So be cautious. Adjust your analysis, perhaps the selection of predictors or the range of observations, to see if the predictions are solid. It's always worth your while to scrutinize the kinds of results that tools provide, so touch base with reality before you take the output of analysis as infallible.

## 4.2 Simulating Data: The Basics

Say that we've come up with a decent model of how a set of predictors, X, can be used to estimate a key outcome, Y. But a lot of the variability in Y still isn't explained by X, and even the estimates of the effect of the predictors (the  $\beta$  coefficients) have a fair amount of uncertainty (those SE values). Still more, we don't control all of the X predictors and they seem to vary in ways that we can't completely predict. What do we do? Or, more specifically, how do we make use of the information that we do have.

Certainly it would be irresponsible for us to just use the "average Y" to make our decisions. Similarly, if our decisions can be distinct in different X variable conditions, or if our decisions actually involve some of those X variables, we should certainly be considering estimates of Y adjusted by different values of X. But we should also consider the uncertainty around our values of X, uncertainty around the effects of X on Y, and generally the variability that surrounds the estimated Y. All of this provides insight into the risk profiles of outcomes our decisions might lead to. And those risk profiles can be critical when we go about looking for good prescriptive solutions down the road. We shouldn't ignore them.

How do we begin to construct risk profiles based on our data and predictive analysis? Profiles that represent variability that we've observed but can only partly account for? One approach involves simulation and can depend on our ability to generate similar token data whose values we also can't entirely account for – that is, data that we might refer to as being random noise, albeit random in an appropriately defined structure. Fortunately for us there are many different ways to calculate random numbers in Excel. The following sections discuss some of the more common distributions that might be encountered for X variables that one doesn't have complete control over. We'll discuss incorporating such randomly distributed variables into our estimations of Y following this.

### 4.2.1 Continuous Random Variables

We can start by breaking random variables into two general classes: Continuous and Discrete. Within these classifications we have an array of

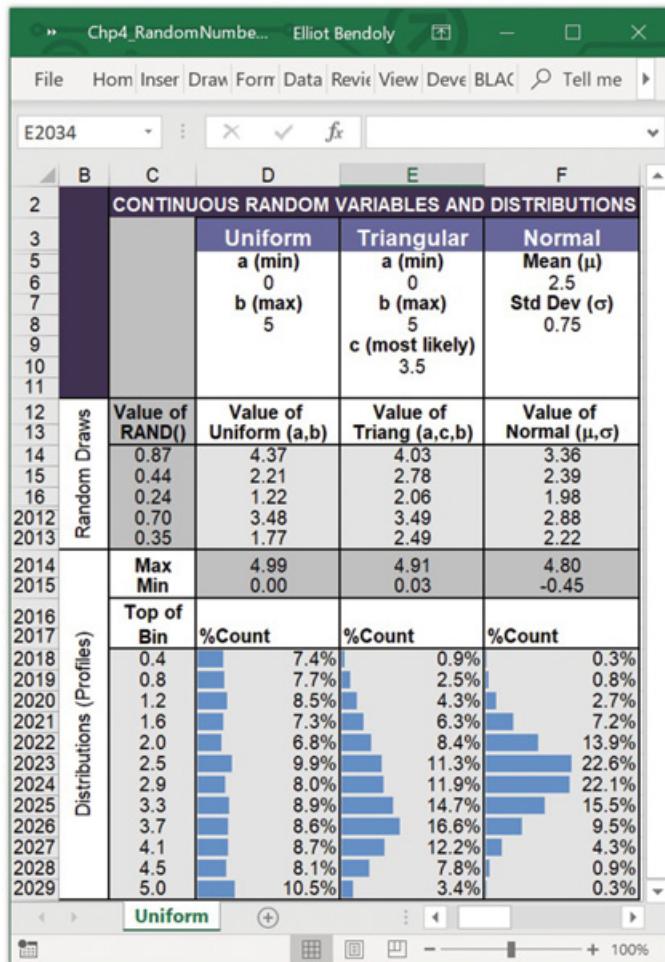


Figure 4.5 Example draws and sample distributions from continuous random variables

distributions that describe the probability of various values being observed. Examples of these, generated in a spreadsheet setting, are provided in the Chp4\_RandNumbers workbook. Figure 4.5 provides some of the example observations generated in the continuous random variable space, along with their observed distributions. We can think of these as the risk profiles around our X variables, though the concept of a risk profile, or at least what it alludes to, will be more relevant when discussing our estimations of outcomes Y.

Uniform random variables are those most readily available in Excel. They range from some minimum value ( $a$ ) to some maximum value ( $b$ ). To create a random number based on this distribution in Excel, start with the

random number generation function RAND(). RAND() automatically gives you a random value between 0 and 1. To change that to a range from  $a$  to  $b$ , enter the following information into a cell (replacing  $a$  and  $b$  with real numbers):

$$= a + \text{RAND()} * (b - a).$$

In the example workbook we include both the use of RAND() over 2,000 cells as well as its transformation into the above distribution with  $a = 0$  and  $b = 5$  in column D. The *sample* distribution of numbers generated in this way is depicted in part through the use of binning, a COUNTIF function, and some conditional formatting (cells C2018 and below).

Triangular distributions also represent random values ranging from some minimum value ( $a$ ) to some maximum value ( $b$ ). However, unlike uniformly distributed random values, the chance of picking a number in this range peaks at some value ( $c$ ) and is essentially zero at both  $a$  and  $b$ . Excel uses the IF statement along with the RAND() function to provide the following form (the working function excluding the semicolons and the annotations that follow):

= IF (RAND() < (c-a)/(b-a); for example, is RAND() below the peak?  
 $a + \text{SQRT}((b-a)*(c-a)*\text{RAND}());$  for example, if yes, then use this calculation.  
 $b - \text{SQRT}((b-a)*(b-c)*(1-\text{RAND}()));$  for example, if no, then use this calculation instead.

You're probably also familiar with what are called normal random variables, at least in theory – the normal distribution is often referred to as the bell curve. It is generally the form that observations in Y are assumed to take when you're conducting simple multivariate regression. (If your Y observations follow a distribution notably different from a normal curve, you should think about transformations to address this, or alternative estimation methods.) Fortunately, Excel makes the generation of normal random variables extremely simple:

$$= \text{NORMINV}(\text{RAND}(), \mu, \sigma)$$

Here,  $\mu$  represents the mean and  $\sigma$  the standard deviation of the variable and they should be substituted for with appropriate numerical estimates or references to such.

Each of the Uniform, Triangular, and Normal random variable forms are illustrated in Chp4\_RandomNumbers, over 2,000 random draws, with binned distributions depicted (mini histograms in advance of our Chapter 5 discussion of bar charts and other graphics). Excel's built-in functions allow for several other common continuous distributions to be handled the same way (such as CHIINV() for the  $\chi$ -dist, FINV() for F-dist, TINV() for t-dist, etc.).

### **4.2.2 Discrete Random Variables**

Sometimes it's useful to consider the chances that alternative discrete events occur. For example, a person decides to buy Brand X instead of Brand Y; or three people don't show up for hotel room reservations on Friday (as opposed to one person, or two or four people); or a competing firm decides to build its new facility in Jacksonville instead of Des Moines or Toledo; or items 2 and 17 are dropped from a federal bill outlining tax incentives for small exporters.

Each of these events is discrete – it either happens or it doesn't. If it does happen, the implication is that alternative events that could have occurred in its place didn't, at least for the specific timeframe considered. Sometimes, the alternative events are related in an ordinal fashion – for example, three people not showing up is greater than two people not showing up. Sometimes alternative events are simply nominal; they're not easily comparable by a single measure but they're distinct from one another nevertheless (buying Brand A instead of Brand B, for example). The consideration of these kinds of variables and their uncertainty are just as important to good decision making as is the consideration of more continuous variables (those that can take on meaningful decimal values). Several discrete random variable forms are shown in Chp4\_RandomNumbers and depicted in Figure 4.6.

The simplest discrete random variables describe multiple discrete events, with each event having exactly the same chance of occurring at a given point in time. Regardless of what each event is, it can be represented by coding, such as Event 1, Event 2, and so on, up to Event  $n$ . This makes it easy to use Excel to generate random events with equal (uniform) chances of occurring. For  $n$  random events, we can use:

$$= RANDBETWEEN(1,n), \text{ or } =INT(n*RAND()+1)$$

Both will provide equally weighted random integers between 1 and  $n$  that can correspond to each of the  $n$  events under consideration.

In some circumstances, we are interested in only one of two events taking place, such as a potential customer either signing up or not signing up for an offered service contract. The chances of either event are often not equal. These are called Bernoulli events, and the outcomes are typically coded numerically as 0 (doesn't sign up) or 1 (does), a bit like a weighted coin flip. If the probability of 1 occurring is  $p$  percent (29 percent, 73 percent, 8 percent, or another percentage), we can generate a 0,1 variable value in Excel by using the following IF-based statement:

$$= IF(RAND()<=p, 1, 0)$$

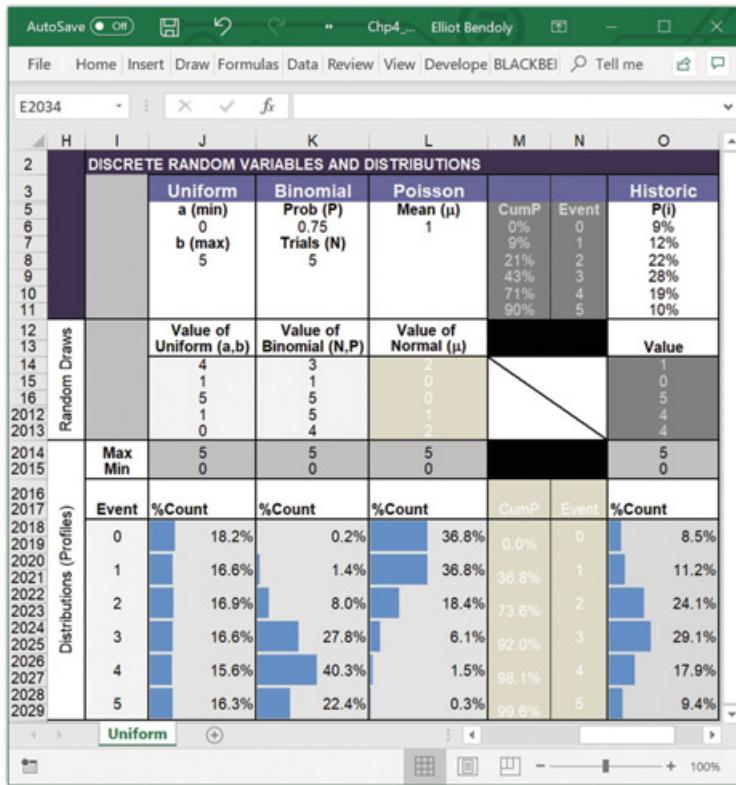


Figure 4.6 Example draws and sample distributions from discrete random variables

We could even replace the 0,1 coding directly with meaningful information, such as:

= IF(RAND()<=29%, "Signs up", "Doesn't sign up")

More interesting is when we are faced with a number of repeated random events of this type; in other words, a bunch of independent flips of the same potentially unevenly weighted coin. If we are interested in estimating the total number of “heads” (one of the two events whose probability is the focus), we are talking about binomial random variables. This is depicted in the Chp4\_RandomNumbers workbook (a single Bernoulli can be very relevant; it just doesn’t make for a very interesting depiction of a distribution). The function we use to pull a number from a binomial distribution is generally as follows:

= CRITBINOM(BinomTrials,BinomProb,C15)

Here the cells K6 and K8 have been named BinomProb and BinomTrials, representing the probability of any event happening (e.g., chance of “heads”

on our weighted coins) and the number of trials (number of “flips”). Reference to C15 is again a reference to the outputs of the RAND() function.

The Poisson distribution is another one that often comes up in modeling. Among other things, it has been used to characterize things like the number of customers arriving at a service in a given window of time, the number of insurance claims to be expected over a year, the number of times a stock will experience a price spike while held, and even the number mutations in a DNA segment by length. Again, there are aspects of this activity that might be predicted by other factors, if appropriate prediction approaches are taken, but a good deal of their variability remains hard to account for and has been observed to follow the Poisson curve.

In contrast to the binomial distribution, which is fundamentally bound between 0 and the number of trials, the Poisson distribution has no upper limit and often appears stretched to the right (long right-tail). We also don’t have a quick-access Poisson random number generator available in standard Excel (though the Blackbelt Ribbon provides one, as we’ll see in a second).

On the plus side, standard Excel does have the following function for calculating the probability of any number of events being observed in accordance with the Poisson curve:

`= POISSON(Events,PoissonMean,FALSE)`

In cells L2018 and below, within the Chp4\_RandomNumbers workbook, this function generates the theoretical Poisson population distribution across a limited series of events (0–5, named Events) and for the average value of that distribution (Poisson Mean). Note, as in Chapter 2, I’ve named a range of cells “Events” but reference to that range in this case only pulls the value in the corresponding row of the function. As a result, this Poisson function is only pulling the probability of 0 events, or 1 event, or 2 events, and so on, at a time.

To draw individual random events, we can actually make use of the distribution outlined in these cells (L2018 and below), in contrast to deriving a sample distribution from those random draws as in the previous cases. All we need is the construction of an offset cumulative probability distribution, starting at 0 and adding incrementally each event probability. This is built in cells M2018 and below, with the event values replicated in the adjacent cells N2018 and below. With this structure it is simply a matter of looking for the closest offset cumulative value that doesn’t exceed a value of Rand() drawn and retrieving the associated event. The following formulation accomplished this:

`= VLOOKUP(RAND(),M$2018:N$2029,2,TRUE)`

Yes, this is VLOOKUP but applied in a very special way. The fourth parameter here is set to TRUE, indicating an inexact search. And that is precisely what we need. We have only a handful of percentages in our offset cumulative probability list (for a Poisson mean of 1, the first six in the offset cumulative list would be 0.0 percent, 36.8 percent, 73.6 percent, 92.0 percent, 98.1 percent, 99.6 percent). There is very little chance that a random number between 0 and 1 will hit any of these. But there is a 100 percent chance that it will be either equal to or greater than one of these. And the inexact search is looking for a number that is equal to or lower than the value provided by RAND(), without going over it. This means the first event, the value 0 in this case, will be drawn if RAND() is between 0 percent and 36.8 percent. The fifth event, the value 4 in this case, will also be drawn 1.5 percent of the time (when RAND() is between 98.1 percent and 99.6 percent), also in line with a Poisson distribution with a mean of 1. We've chopped things off at a value of 5 in the example, giving it a slightly greater chance of happening, but otherwise this principle applies throughout. By this approach we draw 2,000 values from a Poisson distribution, with the value of 5 really representing "5 and above."

If you want a less complicated way of getting at Poisson random variable values without creating an offset cumulative list and without using VLOOKUP, we have a solution for that as well. Based on the same principle of searching for events that have specific cumulative probabilities associated with them, I've included the PoissonInvBB() function as part of the Blackbelt Ribbon add-in. The function takes up to two inputs, the latter of which is optional. The first input is the Poisson mean. If you don't provide a second parameter, the function will simply provide a value from a Poisson distribution with that mean. If you do provide a parameter, you can use that value to either input a specific percentage as input or specify RAND(), which will continue to generate new Poisson draws. I wouldn't suggest that you use this or any random number generator for huge quantities of on-recalc repeated draws. The more calculations you have that always require updating, the slower things will run. There are limits to keeping everything completely updatable!

Finally, in some circumstances, we have multiple alternative events, each of which has its own unique probability of occurring but for which no canned mathematical function is an accurate description. Instead we rely on actual historical data of these occurrences and base the likelihood of their future occurrences on these historical frequencies. Because these are real alternative events, adding up the chances of each event should give us 100 percent (all possible outcomes need to be accounted for). We can consider these events and their probabilities in a tabular format just as we did with the theoretical Poisson distribution.

Table 4.1 *Example history of events during a specific time window*

Number of “no-shows”	Probability of that number of no-shows occurring	Offset cumulative probability
0	9%	0%
1	12%	9%
2	22%	21%
3	28%	43%
4	19%	71%
5	10%	90%

Consider an example of this from the restaurant industry. Lobo’s Cantina, in an effort to better meet the expectations of their customers, has begun to scrutinize how best to staff their operation and manage reservations. Demand is uncertain, and just because a party has reserved a table doesn’t mean that they will show up. Empty tables and idle wait staff are things that Lobo’s Cantina generally wants to avoid. To get a better handle on things, it has started to collect data on the number of parties making reservations but not showing up (aka “no-shows”). Past data shows that the following number of no-shows occur in the 8 pm–9 pm time frame on Fridays (Table 4.1):

As with the Poisson case, we are steps away from drawing possible events from this distribution if we want to simulate a set of representative 8 pm–9 pm Friday scenarios. Inexact searches with VLOOKUP can get this job done. In the Chp4\_RandomNumbers workbook, we in fact use this same approach. In that workbook cells M6:M11 contain the offset cumulatives, with the no-show counts to the right of these. The draws from this, using VLOOKUP, are presented in cells O14 and below. Is there a more refined way of doing this? Certainly. If I were going to provide PoissonInvBB() as a function using a similar principle, it would make sense to take just an additional step to provide HistoricalInvBB(). This function takes up to three inputs: the range of events (which can be text or numerical), an equally long array of noncumulative actual event probabilities, and a third optional parameter (as in the PoissonInvBB() case). In the present case, the values in cells O14 down could be generated as follows using this custom Blackbelt function, without the need for separate cells containing offset cumulatives:

=HistoricalInvBB(N\$6:N\$11,O\$6:O\$11)

### 4.3 The Use of Simulation in Analysis

We used the term simulation in the last case, though we've really only scratched the surface with regards to using all of the intelligence that predictive modeling might provide. To reiterate, if we've done our homework in collecting data, understanding its structure, and beginning to understand its relationships through predictive methods such as regression, we are much better positioned. Nevertheless, we have at least the following forms of variability to consider when thinking about the risk around any outcome ( $Y$ ) that we care about:

- $\varepsilon_1$ ) Uncertainty in the outcome ( $Y$ ) that our predictive models don't quite account for.
- $\varepsilon_2$ ) Possible error in estimating the impact (the  $\beta$ s) of our predictor variables.
- $\varepsilon_3$ ) Uncertainty in those predictor variables that we don't control (some of the  $X$ ).

This might seem like a lot that we still don't know. And it can be. Some predictive models are pretty poor (e.g., very low R-square), hence we face bigger issues in  $\varepsilon_1$ . Some of the estimated impacts can be pretty unclear (e.g., higher absolute ratios of SE to  $\beta$  and less statistically significant), creating bigger concerns in  $\varepsilon_2$ . If impacts are not linear but something more complex, you'd expect that estimates of linear effects as substitutes for these more complex effects would be quite a bit off. Similarly, some of those predictors ( $X$ ) we simply don't control and they can vary greatly in even fairly specific settings (higher  $\varepsilon_3$ ). Finally, there are issues beyond the sheer complexity of the system, sometimes coming from flaws in measurement itself. Whoever was measuring, recording, and reporting these values of  $Y$  and  $X$  might have made some mistakes. Maybe these things are really tough to measure. That could heighten uncertainty in all three forms.

Fortunately, often in management we have some control over some of the predictors. In fact, missing the opportunity to include such predictors in a model would be pretty foolish. After all, we'd like to know not only what impacts what but how to push those levers to get the outcomes we want. If we are only creating models that predict outcomes but provide us no opportunity by which to impact those outcomes, we are only setting ourselves up for an accommodation game, where separate decisions are designed to accommodate specific scenarios we don't control – which is helpful but limiting. On the other hand, if we are building at least some predictive models, where we control some of the predictors, then we are setting ourselves up for a much more fruitful management prescription.

To get a grasp of how all of these pieces fit together, to provide a better model of the system that drives the outcomes we would like to manage, we have two routes available to us in analysis. The first is closed-form normative

modeling, which typically assumes that distributions follow particular, relatively convenient patterns. This essentially results in a system of equations that can be solved to provide a prescription. If these assumptions cannot be made, however, we tend to rely on simulation tactics informed by prescriptive exercises such as regression.

Simulation-based models can take on many forms. Some are much simpler single-shot models, where a range of scenarios in X are considered, dependent relationships are evaluated (e.g.,  $\beta$ ), and variability is considered across sources ( $\epsilon_1-\epsilon_3$ ) to generate variations (risk profiles) on an outcome (Y) of interest. Others are more complicated in that feedback loops exist between outcomes in one period and ultimately the dynamics and outcomes of another (e.g., the value of Y in one period actually serves as a predictor of subsequent values of Y or even of subsequent values of X). These system simulations are going to be necessarily more complex and the predictive models leading up to these similarly tend to be a bit more sophisticated.

Regardless of complexity, there are two general approaches to using simulation to help inform prescription. The first is to use simulation models to generate a range of inputs to decision making. For example, consider a simulation describing a range of productivity levels, error rates, and levels of demand, all based on historical observations. The associated prescriptive decision making might involve determining which staff assignment decisions would work best for each of these scenarios. A manager might look for commonalities across these possible decisions in an attempt to divine which aspects of this decision are good regardless of scenario. This approach to the use of simulation, in advance of the development of prescriptive solutions, is referred to as a *preconstructive* approach to considering variation.

Alternatively, a manager might like to take a derived solution (based on “best” guesses of things; e.g., mean estimates of demand) and then reexamine how well the resulting solution fares if the numbers describing the problem change. Would it always ensure profitability? Would it always be technically feasible? What are the upside and downside risks (the risk profile) associated with the outcome for this solution? We call this a *postconstructive* approach to considering variation. In practice, ideally, both approaches should be considered. For example, a manager might derive a set of potential solutions based on slightly different initial problem parameters and then see how each solution does under a range of alternative parameters. The manager might in fact take into consideration the magnitude of the upside and downside risks in identifying a prescription and it might not be the prescription with the best “average” outcome.

This might seem like a lot of work ahead of decision making. Is it necessary? Would our decisions really be that far off if we just worked with the averages? Sadly, there are countless examples from practice in which such shortcuts have proven disastrous. But appreciating the danger of ignoring risk requires no more than a simple thought experiment. Eli Goldratt's famous novel *The Goal* has a great process example of how not accounting for variance/uncertainty/risk can cause havoc. Taking a page from his writings, imagine a process broken into five steps in sequence (a serial process). The context could just as easily be one of loan processing, automobile assembly, or jewelry cleaning (as in the Chp5\_SynchSwim workbook). The work at one step or by the station/individual executing that step flows down to provide a supply of work in progress (WIP) for others. For simplicity each station can complete between one and six units in a given period, all outcomes having an equal probability, as in a roll of a die. After twenty periods, if each station had four units of WIP to start with, how many units should we expect to see fully processed?

Well, the average value of a die roll is 3.5. This isn't an actual roll but, if you rolled a fair die 10 times and added each roll up, your best guess of that sum would probably be 35. So, if this process plays out over 20 rounds, shouldn't we expect that 70 units would be processed? We would, perhaps, if there were just one step. But if you actually let the simulation play this out, you'll see notably smaller numbers. Try it out. Let the simulator run 20 rounds of this process 20 times. I'll bet you'll never see 70 as an outcome.

The simulation is not broken. The system of variation is just a bit more complicated than any one average would suggest. If we had to make decisions about how much to invest in the capacity of this system to yield specific outcomes and we based these decisions on only a few averages, we would be disappointed.

In Chapter 6 we will begin to discuss prescriptive analytical approaches (i.e., the search for ideal (optimal) solutions for problems where uncertainty is not an issue). We will then move on to joint methods involving the search for solutions in the presence of various forms of uncertainty, leveraging simulation methods in both *preconstructive* and *postconstructive* approaches (Chapter 7). In those discussions, we will use the term *simulation optimization* to broadly describe these composite tactics. Before we undertake all of this, however, we need some practice in building simulations.

## 4.4 Examples in Simulation Modeling

Regardless of the type of simulation being built, the development and use of any simulation model fundamentally involves a set of common steps. Some

simply involve more features than others. We will discuss these through the illustration of some concrete, albeit stylized examples. The stylization applied here is intended only to assist in discussion, as these are actually based on much more complex real-world cases. We will start with the examination of a scenario in which an outcome's risk profile is assumed to be simply replicable across time. We will then consider a simulation context in which a single decision's outcome in one period impacts subsequent and final outcomes of interest. In that case we will be building final risk profiles of outcomes on the backs of earlier ones. Lastly, we will consider a much more complex, albeit still stylized simulation context in which a decision plays out at different points in time and in which intermediate outcomes trigger additional decisions as well as final outcomes and their risk profiles.

#### **4.4.1 Example: Revenue Management at Lobo's Cantina**

Consider once again the case of Lobo's Cantina, introduced earlier. They've been keeping tabs on people not showing up for reservations. Appropriately, one of the many policy decisions they are looking at is how to manage reservations for certain time windows (e.g., 8 pm–9 pm on Fridays). However, they have more data now, collected across various kinds of reservations. At this point they have found that a binomial distribution fairly consistently captures the no-show phenomena. That is, the chance of any one party showing up ( $P$ ) or not showing up ( $1-P$ ) seems independent of others doing the same. This percentage chance also seems to be consistent no matter how many reservations are made, though it does appear to differ by hour and by day. For the 8 pm–9 pm Friday timeframe, for reservations that would occupy one of their 15 four-seat tables, about 73 percent of all those who make reservations actually show up.

Lobo's not only hopes to build a model to simulate these shows/no-shows but more broadly wants to model the kinds of costs that might be incurred by a particular reservation policy over a particular time frame, so that they might extend the model to others. Ultimately they'd even like to distinguish between the dynamics of different kinds of parties seated (four-seat reservations vs. two-seat) and come up with a grand plan for minimizing cost and perhaps containing other risks across the board. This kind of challenge is faced by a wide range of service firms wishing to capitalize on fixed investments and facing uncertain demand. The approach, broadly, is to overbook (think airlines, hotels, etc.). The big question is, by how much. To begin to answer this question the firm in question must consider both upside and downside costs and the overall risk profile of each of their options.

In the Lobo's Cantina case cost (and policy performance) will be based on two factors: opportunity losses associated with unfilled tables and the loss of goodwill from customers arriving to find nowhere to sit. The first of these factors might be based on data the firm has on parties previously occupying these tables during the time frame in question. Despite the fact that all parties spend differently and meals cost different amounts to prepare, Lobo's Cantina at least has that past data. It turns out that the distribution of these prior per-party profit numbers nicely follows a bell curve with a mean of \$150 and a standard deviation of \$50 (we might as well simplify these for demonstration as well). This means that there are now at least two forms of uncertainty to be included in the firm's simulation: (1) how many people don't show up; and dependent on that, when tables aren't occupied, (2) how much profit is foregone.

The second factor (loss of goodwill from being turned away reservation in hand, etc.) is notably more difficult to assess in terms of dollars. It might include losses of future business from disappointed customers as well as from some of those they share their complaints with. Unfortunately, Lobo's doesn't have a good way of distinguishing whether people opt out of future service because of being turned away in the past, or whether they simply no longer live in the area. Nor can they know whether the same party is coming but someone else is paying, or whether for that matter the same people are now paying with cash. Nevertheless, Lobo's does have some anecdotal evidence: turning away doesn't help your reputation on the street. It's a critical piece of the puzzle, but just how big is unclear. It's worth keeping track of in some form, even if it's hard to convert into dollars.

Even with these limited details, the structure of a simulation model begins to take form. The workbook Chp4\_LobosReservations and the snapshot in Figure 4.7 provide an example of how this might be constructed. A few of the connections among the spreadsheet cells are annotated with drawn arrows in this example. The flow of calculations is fairly straightforward: the historical show rate of 73 percent and the existing policy (the decision) of how many reservations are inputs to the random draw on the number of parties showing up, in a single iteration of this simulation (a single F9 hit). Along with this, the number of parties seated is simply the minimum of two values: the number of parties showing versus the number of tables available. When the number of parties showing is greater, we have a nonzero count of parties turned away. When it is less, we end up with empty tables. Random draws on profit for only those tables occupied are calculated in turn.

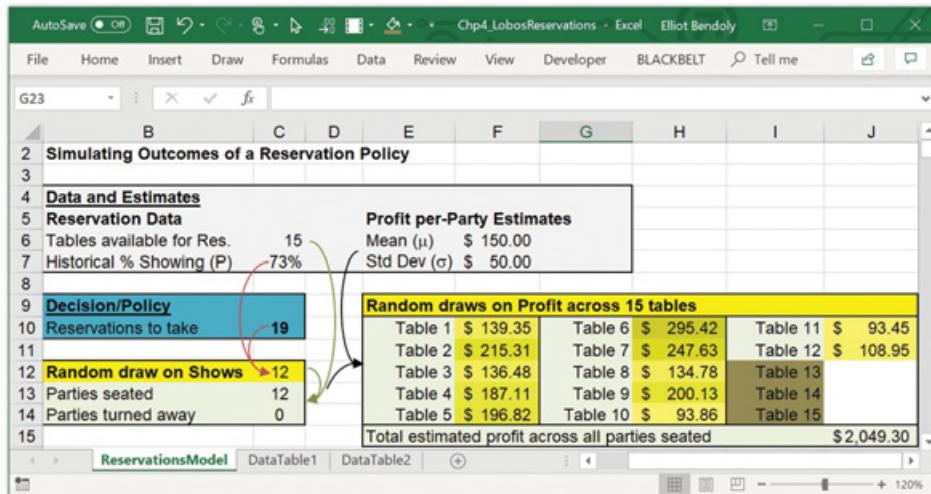


Figure 4.7 Model structure for the Lobo’s Cantina reservations simulation

#### 4.4.1.1 Data Tables for Multiple Evaluations

Although this spreadsheet structure is sufficient for providing an estimate of how well a specific policy might do given one set of random variable draws, a single set of draws can be misleading. We could be lucky or unlucky with that set. Alone, it certainly shouldn’t form the basis for final consideration by decision makers. We need to be able to have stable estimates of policy performance that are representative of a full range of random draws and eventually we will want to be able to compare these results for various policies assessed the same way. A data table is a convenient tool that can be leveraged to this end.

The Data Table tool is found by selecting Data>What-If Analysis>Data Table ... (Figure 4.8).

A data table is used to provide a series of iterations on a set of calculations. It can communicate input into that set of calculations and record associated outcomes, or it can simply trigger recalculations. The latter is useful in situations where one desires model outcomes change without manual manipulations of input (e.g., when multiple random number sets are being drawn, or even in some cases when a living record is active in iterative calculation mode). Data tables can also be set up to record more than one outcome of the table for a given recalculation. Because there is more than one way to use this tool, we will need to set some things up before we ask a data table to do its work.

We’ll begin on the sheet DataTable1 in that same workbook. Let’s store our “Number of Reservations to Take” decision variable in cell A1 of that

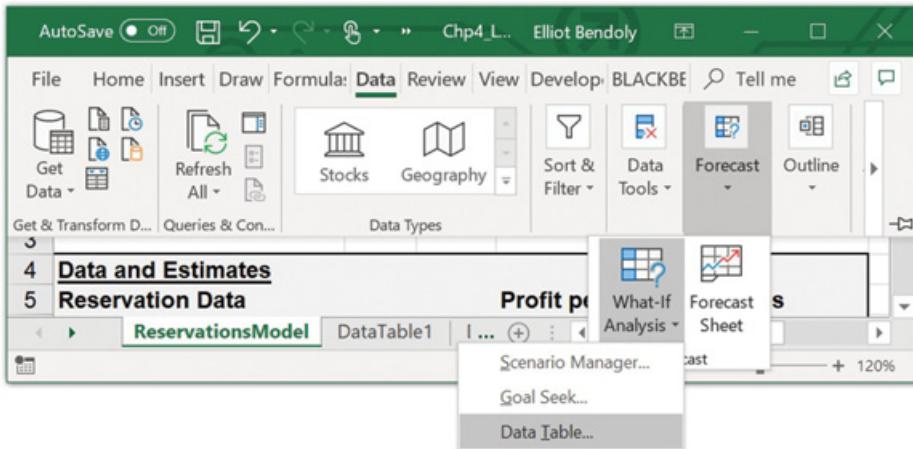


Figure 4.8 Accessing data table functionality in Excel

sheet. In cell A2 we can store an optional term representing the monetary cost of turning away a single party. We can choose to keep this empty but the person in charge of Marketing at Lobo's has been arguing that this value should be at least \$150, likely more. Their argument is as follows: Since that party might have been OK with an alternate hour-day combination, booking them for a slot and then turning them away is akin to the opportunity cost of another empty table. Beyond that there must also be some future loss of business from them and those they talk to. Marketing puts this at an additional \$50 for each party turned away, giving a proposed cost of \$200. There are a few assumptions being made here and perhaps some subjectivity, but we'll use this piece of information for now as a placeholder with the option of removing it later.

Now, when using a data table, communication is really everything. And it's easy to get things wrong, largely because of the sparse design of this tool's interface (as we'll see). Specifically, we will be detailing to the data table a set of possible input values, stored in the left column and top row of an empty table we are about to create. It will be expecting additional information on where (cell locations) those values should be placed for each calculation. Unfortunately, the Data Table interface will not accept direct instructions to place values on sheets other than the sheet it is created in. So, if we want a new Data Table on the DataTable1 sheet, where our model is not located, we'll have to jump through a little hoop.

Going back to the ReservationsModel worksheet, we need to make sure that the cell named NumReserves (C10, containing the reservation policy) is not just some static number but rather contains a reference to =DataTable1! A1. In this way any change to cell A1 on the DataTable1 sheet will trigger

changes to the main model and will generate a new set of outputs from that model.

Returning to the DataTable1 sheet, we now need to outline the structure of the data table. If we want a table of 500 runs, we might enter the numbers 1 through 500 in the cells DataTable1!B6:B505. These are purely labels and for our own reference, not part of any calculation in the model. To designate the kinds of decision/policy scenarios we want the data table to examine, we enter a variety of alternative reservation policies: for example, in this worksheet, 15 to 25 reservation bookings in cells C5:M5. Finally, we need to designate what kind of outcome measure we want to summarize in the table. For now, in cell B5, let's go with the Total Adjusted Profit figure (a merger of estimated profit across seated groups minus Marketing's estimated costs due to overbooking: =TotalEstProfits-Overbooked\*A2). The location of B5 is not arbitrary but is what is often referred to as the *keystone cell* in this type of data table (at the intersection of the left column and the top row). Our specification of the fields in the Data Table interface will instruct it to collect values of this calculation for 500 reexaminations (left column) of each policy 15–25 (top row).

Now we're ready to let the Data Table tool do its work. Select the entire area of the table that contains the row labels (1–500), the calculation reference (cell B5), and the various scenario inputs (cells C5:M5). Select the Data Table tool to generate the Data Table dialog box, shown in Figure 4.9.

As shown in Figure 4.9, the Data Table interface is extremely minimalist – simple but not very informative and most people don't find it immediately

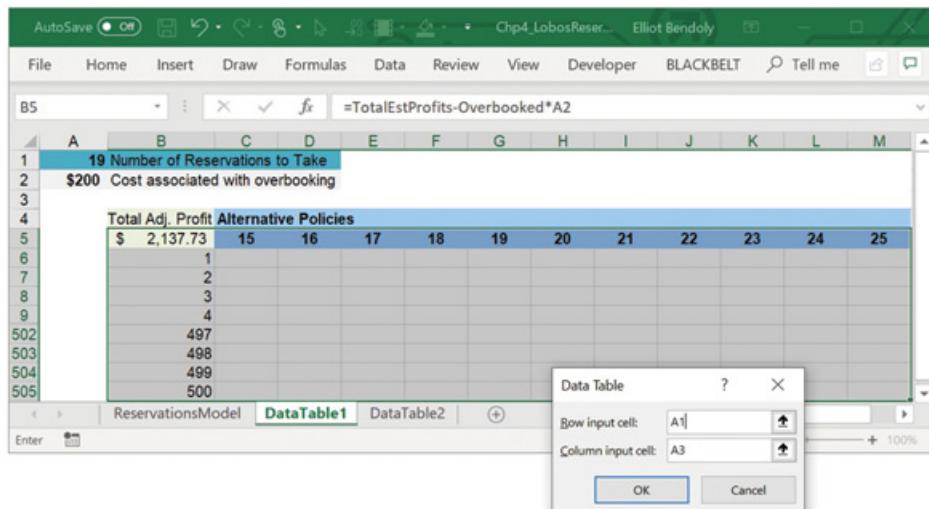


Figure 4.9 Developing a data table in Excel

intuitive. Essentially we have two fields to fill out, each of which can take a cell location as a destination for the content already mapped out in the top row and left column of the selected range. The field labeled “Row Input Cell” is asking where the values in the top row of your range should each in turn be placed. The “Column Input Cell” is asking the same for the content in your left column. At least one of these fields must have a cell location specified. What locations we specify depends on what we have in that top row and left column:

- 1) INPUTS: If the content in either your top row or left column is content that you want your model to make use of, then you clearly need to tell the data table where to place this content so that your model does in fact know about it. As we've said, data tables like to work with the cells on the sheet they are placed on and will not send data directly to other sheets. That is why we created cell A1 on the DataTable1 sheet. Our model looks to this cell, taking that cell's value as the decision/policy to evaluate. Therefore, any number that the data table places in this cell will impact the model. The top row of values 15–25 in this case are exactly the kinds of inputs we want to send to the model, so we need to specify cell A1 in this case as the destination for these inputs.

Rule of thumb for INPUTS: Pick a cell the model is dependent on.

- 2) LABELS: In some cases, the row or the column will be used to simply provide space for multiple iterations on the same input scenario. In simulation this is useful because the relationships between inputs and outputs are not certain and are subject to some randomly modeled noise. Hence repeated evaluations of the conditions in subsequent rows below the first top-row cell, for example, could provide the basis for generating statistics on the effects of that input (means, standard deviations, and so on) as well as for each of the other columns under the remaining top-row cells. In order to get this to work, the leftmost column would contain some “dummy content” (e.g., 1–500, representing replications 1 through 500) and the “Column Input Cell” would be some cell in the sheet that has *no impact* on the model's calculations (e.g., A3). After all, these labels should not mean anything to the model itself. They're meaningful only in the sense of assessing variants of the model's outputs. And that's exactly how they are used in this example.

Rule of thumb for LABELS: Pick a cell that doesn't impact other calcs.

- 3) FORMULAE: In the current example, the kinds of calculations to be stored throughout the data table are designated by the *keystone cell* in the upper-left cell (highlighted in the example). That's fine for this example. However, occasionally you'll want to assess multiple output calculations. We'll see this need in the second portion of this example. In order to do this, before opening the Data Table tool, use either the top row or the left column to contain a set of

distinct formulae or references to different outputs calculated by the larger simulation model. In the example we'll see shortly, we will demonstrate such an assortment in the top row of an alternatively structured table. In order to tell the data table that you want to apply each of the top row formulations in populating the cells below them, you'll need to keep the field "Row input cell" *blank*. It contains neither inputs nor labels. This instruction will populate these calculations down the length of the table (across if they were in the leftmost column instead). Because of how this is executed, it would never make sense for both the top row and left column to have formulae (at least one field in the Data Table form must have a destination outside the table).

Rule of thumb for FORMULAE: Keep the field (at most one) blank.

So, to reiterate, in the present example here's how we would go about specifying the table. First, in the field labeled "Row Input Cell," select A1, which is the cell we are using to store our decision variable and which we are using now to control the nature of the simulation. This tells the table generator that you will want to substitute the various alternative policies you've designated at the top of the table into this input field to generate your results. In the "Column Input Cell" field, select any cell not currently being used (and outside of your table range). I've chosen A3 in the example. The result should be a table populated with outcome data for 500 iterations of the simulation, for each of the scenarios represented in the top row of the table. An example of this output is shown in Figure 4.10, with additional descriptive statistics that I've generated in rows below.

The nice thing about this table is that you can change just about any of the features of the simulation model, including both constants, such as cost figures, and the nature of relationships (i.e., formulae), and the data table should provide updated information more or less instantly in response to these changes. For example, try changing the cost figure in cell A2, or editing the formulation in B5, or changing a policy level in the top row of your table. What is less flexible is the data table's interior content. Just as with LINEST or any matrix function (note the curly brackets in the formula bar when the Data Table range is selected), you can't change these interior cells one at a time. If you need to rebuild the interior, select only the interior, retaining the top row and left column, and clear its contents.

As an alternate design of analysis, we could consider creating a data table that reports on more than one outcome under a specific decision (policy) scenario. The worksheet DataTable2 does this. The table was developed by starting with a structure similar to Figure 4.11 in which the top row references each of the three outcomes, two of which are calculated in the main model and another is calculated on the spot (E3 contains =15-NumServed).

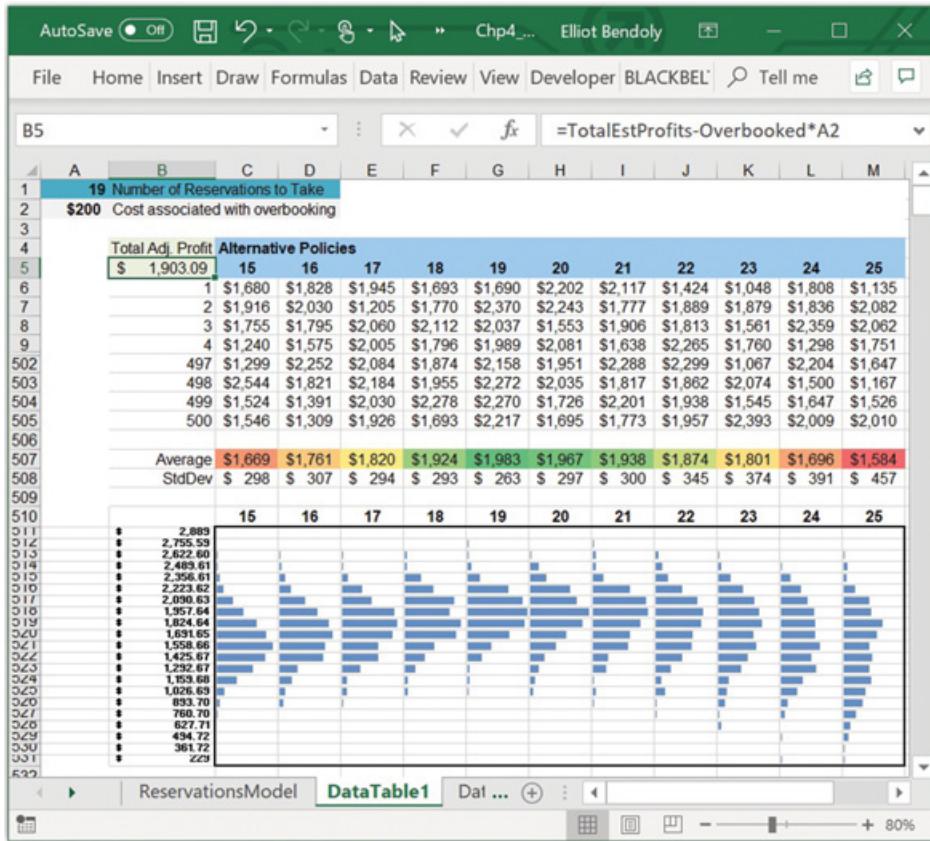


Figure 4.10 Data table content for multiscenario runs on a single outcome

After we select the table area that includes the leftmost column and all three subsequent columns for which calculations are provided and again call on the Data Table tool, the Data Table dialog box will ask for information. This time the only information we want to provide is a reference in the “Column Input Cell” field to a blank cell outside of the table such as cell A4. We leave the other field blank. This indicates to Excel that we want to use the default calculations in the top row to build the rest of the table and we don’t want to use any information for separate calculations. The result is shown in Figure 4.12.

The values in each individual row of this table are related. Because we’re evaluating only a single-policy scenario, each row essentially represents a single iteration (a single set of random number draws, a single F9 hit). In any row, you won’t have both people turned away and empty tables (only one can happen in an iteration). This means that complex relationships between the outcomes recorded in any given row can be further examined and these row-wise relationships analyzed as a connected set.

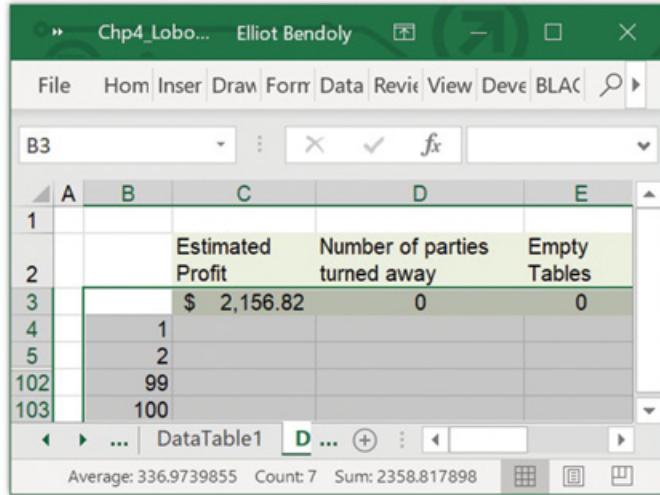


Figure 4.11 Setting up a tabular structure for multioutcome Data Table use

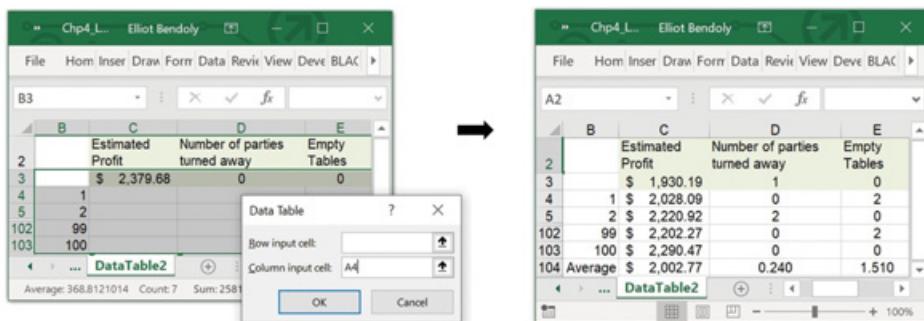


Figure 4.12 Interfacing with the Data Table tool for multioutcome specifications

Let's reiterate a key point in all of this. Data tables are, at their core, communication tools. They are listening for outcomes but also often talking to the source of those outcomes. We need to make sure that whatever they are saying is being listened to as well. In our first example, we had to make sure of this. Our model had to contain some reference to cell A1. In terms of overall flow, that allowed the following to take place: Data Table Row data → A1 → Model (to use in its calculations). I've seen people do the equivalent of having a data table send input into A1 and then have A1 instead reference the model – in terms of flow that's more like Data Table Row data → A1 ← Model (whatever default input exists). The result would be both the model and the data table sending information to A1 and the model not being affected by data table inputs at all. Any outputs pulled into the data table would not be meaningful in terms of input effects.

Data tables offer a lot of potential but only minimal guidance. If you aren't careful with how you design the flow of information when using these tools, you are bound to get results that are misleading.

#### 4.4.1.2 *The "What If" of Modeling Opportunities Missed*

What if we didn't do this work? What if we didn't build a simulation or begin our examinations through tools like Data Table? What if Lobo's Cantina simply tried out a handful of reservation policies, observed outcomes and their variations over time, and attempted to draw conclusions about a best policy based on that information? The relationships between the reservation decision and the outcomes, both their averages and their risk profiles, are not simple linear functions. It would be hard to imagine a clearer picture emerging through this kind of an approach. Plus it would be costly to run a trial-and-error approach to managing this and collecting data. If you know something about your system, even the simplest of cause-and-effect structure, take it into consideration. You'll get farther faster.

Now, what if we were doing this analysis for alternate time frames as well, but we didn't quite have enough data to provide representative historical distributions for each time frame – not yet at least. Do we throw up our hands then or start thinking about some of that trial-and-error stuff? Not necessarily. First, you would imagine that the likelihood of people not showing up might be different per time frame. But maybe experience in other settings suggests that a binomial distribution is a pretty decent estimate of that dynamic, such that there is a consistent  $p$  percent chance any given party will in fact show up when making a reservation. Again, the  $p$  percent might differ by the hour and by the day, but with the actual data in hand you could estimate those values. And you might be able to create a predictive model (e.g., regression) for  $p$  percent of any specific hour or day. Similarly, you might create a predictive model for profit per party as a function of hour and day, or at least blocks of hours for some days.

These models would no doubt fall short of explaining all the variability in the dynamics observed, but they could reduce that variability considerably. As a result, the description of the risk around specific reservation policies could be much more precise for specific hour-day combinations than if all of this data were simply aggregated to create single estimates of performance regardless of hour. It could even potentially help to make up for a lack of data during certain hour-days. In cases where we have an abundance of data for a specific scenario, we capitalize on that. When we have an abundance of data spread across a range of scenarios, we capitalize on what we can interpolate and extrapolate from predictive modeling.

#### **4.4.2 Example: Acquiring a Diamond in the Rough**

In our first example we saw how models can provide structure, connecting decisions to simulated draws based on historical evidence and in turn to outcomes of interest. Now, let's consider an example in which decisions lead to a cascade of outcomes, each subject to uncertainty and building on a previous one. RoughCut Media is a major Internet company thinking of purchasing DashQap, a struggling online service provider. DashQap's current annual revenues are \$100 million, with expenses of \$150 million (again keeping some base numbers simple). Current projections suggest that DashQap's revenues will continue to grow on average at 14 percent per year, while its expenses are likely to grow at an 8 percent annual rate. Variations around these rates are expected from year to year.

An in-depth analysis of prior acquisitions has suggested that additional investments by RoughCut Media, if they choose to acquire DashQap, could influence these growth rates. Their predictive model has estimated coefficients for the impact of additional up-front investments in technical redesigns of the data management system and user interface of DashQap's web portal. The model suggests that each additional \$1 million invested up-front in improvements in data management can yield 1 percent reductions in the growth rate of costs, down to a minimum of 2 percent growth. Another estimated coefficient in their model suggests that each additional \$1 million invested in the user interface can yield half a percent increase in the revenue growth rate (up to a maximum of 19 percent). However, there is uncertainty around these impacts (these  $\beta$ s). Further, these predictors account for only a fraction of variation in their data. In total, RoughCut expects that revenue growth in a given year will follow a roughly normal distribution with a standard deviation equal to 2 percent plus an additional 0.05 percent for each \$1 million invested in the user interface, while the cost growth will face a SD of 1 percent plus an additional 0.1 percent for each \$1 million invested in data management (yes, investments here contribute to uncertainty on both fronts).

These descriptive and predictive analytical details provide a roadmap for assessing the potential of the acquisition, perhaps justifying it. Ultimately RoughCut would like to get an impression of various measures of performance that might be associated with the acquisition, under various investment scenarios. The setup for this example can be found in the workbook Chp4\_RoughCutAcquisition. As shown in Figure 4.13, there is a lot going on here, so it's useful to separate things out into sections. The first block at the top of the MainModeling sheet outlines the estimates and the impact of investment decisions just described. In this case we use target Adjusted Rates to back-calculate assumed investment amounts, since we will be

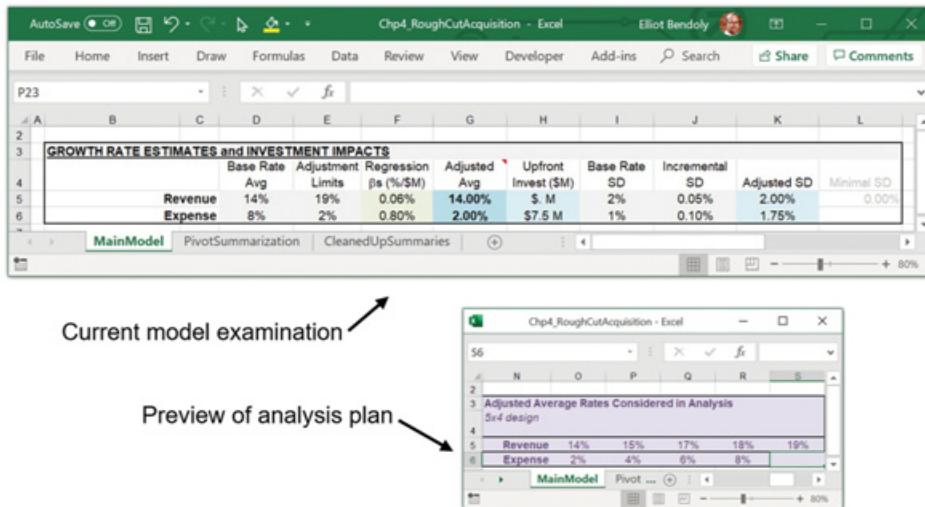


Figure 4.13 Details and estimations of averages and standard errors in decision outcomes

focusing on alternative scenarios under which certain adjusted rates are considered. Associated uncertainty around those rates is calculated in turn. A preview of the adjusted rate scenarios to be considered in analysis (with investment scenarios implied) is provided in the lower right panel of Figure 4.13. These scenarios involve adjusted expense growth rate averages of between 14 percent and 19 percent and adjusted expense growth rates between 2 percent and 8 percent.

Beyond these essential details we have a series of calculations needed to assess what these growth rates will translate to in terms of the potential appeal of the acquisition and associated upfront investments. First, we'll need to calculate the expected annual revenues and expenses for each future year to try to figure out when the firm would start to see a positive annual return. We start from the point of acquisition, incorporating any additional up-front investments as one-time expenses. Growth in revenue and expenses draws on the normal distributions described by the first block, with both averages and standard deviations linked to implied up-front investments. These yearly estimates are developed across a 13-year time frame in this case and are accompanied by simple logical tests for annual revenues exceeding costs. The calculation of when a within-year breakeven takes place requires leveraging the MATCH function to find the first year in which the value TRUE emerges from this test. Additional calculations relating to total discounted ROI are also included at this point, with key performance summaries emphasized in their own summary block (see Figure 4.14).

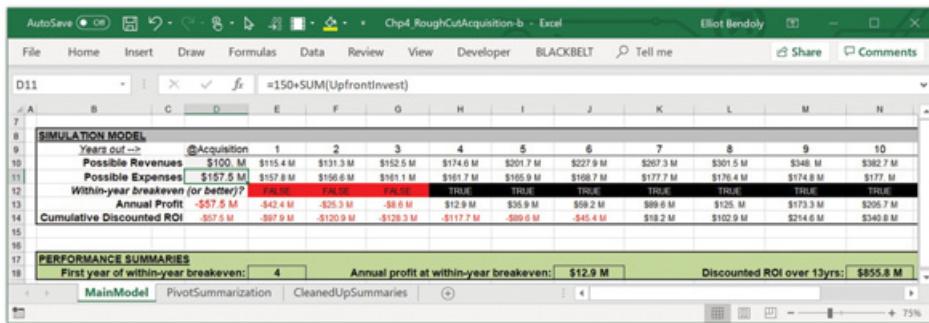


Figure 4.14 Main calculations of interest as for the RoughCut simulation

The next few steps in evaluation, though not in actual model building per se, also require some deliberate planning. For this problem we have quite a bit of analysis that we'd like to conduct. We'd like to look at multiple outputs generated simultaneously. We'd also like to consider more than two changing factors as inputs (both expense and revenue growth rate investments). And we would like to ensure multiple iterations of the simulation model for all scenarios considered so that we aren't misled by the results.

Doing all this with a single data table might seem daunting to those new to these methods, but a general approach to aid us does exist. It involves encoding combinations of multiple inputs into a single meta-input, using that meta-input as the lone data table source of input to a model, and disentangling the various input levels represented by that meta-input for use in the model calculation. The approach, borrowing from the research community, is referred to as the Design-of-Experiments (DOE) method to data table development.

We've already outlined five adjusted revenue growth rates (14 percent, 15 percent, 16 percent, 17 percent, and 19 percent) and four adjusted expense growth rates for consideration (2 percent, 4 percent, 6 percent, 8 percent). This gives us a total of 20 ( $5 \times 4$ ) combined scenarios to investigate, as previewed in Figure 4.13. For emphasis, Figure 4.15 fully articulates these 20 factor-level combinations. Essentially, this is the experimental design we would like a data table (of some kind) to evaluate multiple times, delivering a range of outputs of interest to us.

Note that in this table I'm making ample use of reference numbers to help label both the scenarios as well as the individual rates being considered. This again is aimed at assisting in the communication of details elsewhere in the sheet (we are pulling in the rates listed separately in the upper-right block of this sheet's details). It also makes the translation of our data table's input extremely mutable (easy to change).

SCENARIOS & DESIGN of EXPERIMENT (DOE)		
Scenario #	Adjusted Revenue Growth Rate Average	Adjusted Expense Growth Rate Average
1	1 14%	1 2.0%
2	1 14%	2 4.0%
3	1 14%	3 6.0%
4	1 14%	4 8.0%
5	2 15%	1 2.0%
6	2 15%	2 4.0%
7	2 15%	3 6.0%
8	2 15%	4 8.0%
9	3 17%	1 2.0%
10	3 17%	2 4.0%
11	3 17%	3 6.0%
12	3 17%	4 8.0%
13	4 18%	1 2.0%
14	4 18%	2 4.0%
15	4 18%	3 6.0%
16	4 18%	4 8.0%
17	5 19%	1 2.0%
18	5 19%	2 4.0%
19	5 19%	3 6.0%
20	5 19%	4 8.0%

Figure 4.15 Design of Experiment (DOE) to be executed by a data table simulation approach

Let's take some more time to think about this approach and its effectiveness. We want to extract four performance measures from the simulation process. To do that with a data table, this means sacrificing either the top row or the left column to store calculations or references to these measures. With a top row, say, occupied with formulae, we're left with only one opportunity to provide input to the data table. But we can make the most of it. Because we have these scenario labels, if we simply specify the scenario number we want the data table to make use of (here a number between 1 and 20), we are really specifying two pieces of information for the price of one. A simple VLOOKUP of those scenarios' numbers (finding them within the DOE table) can readily extract both the expense and revenue growth rates for incorporation in analysis. Cells C5 and C6 (RevGrowth and ExpenseGrowth), in the first block of rate details, contain these references to the DOE based on the value in the cell C22 (CurrentScenario). As long as the data table knows to

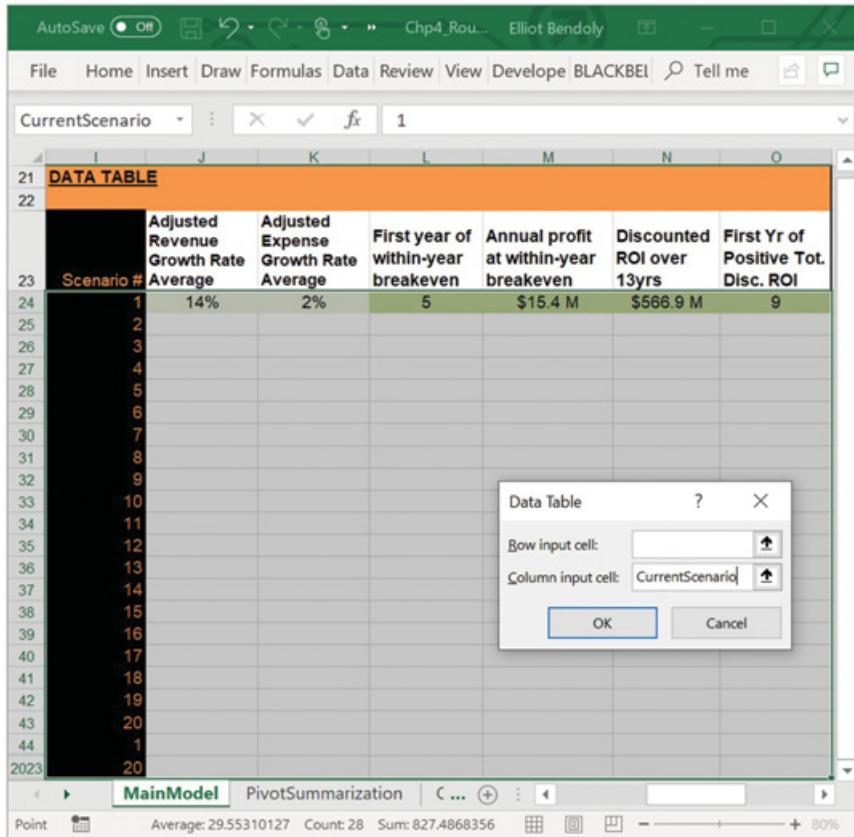


Figure 4.16 Multiformula data table set up for DOE execution

communicate information to that one cell, everything else will fall in line.

With this plan in mind, the data table structure we want is similar to that used in the second data table of the Lobo's Cantina reservation case (Figure 4.12). In Figure 4.16 we see the top row occupied by formulae and references to outcomes of the modeling, as well as the extracted growth rates. The left column has the scenario numbers 1–20, repeated 100 times to drive 100 iterations of simulation for each growth rate (investment) combination. To populate this table we use the same rules as before. The top row contains formulae, so we keep the Row Input Cell field empty. In this case the left column isn't just some set of labels to help us count from 1 to 2,000. This is actually input that drives the way the model works. So we need to make sure it isn't just tossed to an unused cell but rather deliberately placed in the CurrentScenario cell (or C22).

The resulting content occupying the interior of the data table, once populated, will appear similar to the example simulated iterations shown in Figure 4.17.

DATA TABLE							
Scenario #	Adjusted Revenue Growth Rate	Adjusted Expense Growth Rate	First year of within-year breakeven	Annual profit at within-year breakeven	Discounted ROI over 13yrs	First Yr of Positive Tot. Disc. ROI	
	Average	Average					
1	14%	2%	4	\$6.5 M	\$889.8 M	7	
2	14%	4%	6	\$4.9 M	\$271.6 M	11	
3	14%	6%	7	\$10.6 M	\$167.9 M	12	
4	14%	8%	8	\$26.3 M	\$104.5 M	13	
5	15%	2%	5	\$5.2 M	\$707.7 M	9	
6	15%	4%	7	\$20.7 M	\$311.4 M	11	
7	15%	6%	8	\$14.7 M	\$10.7 M	13	
8	15%	8%	9	\$15.2 M	<span style="color:red;">-\$219.6 M</span>		
9	17%	2%	7	\$41.2 M	\$661. M	11	
10	17%	4%	8	\$15.6 M	\$48.3 M	13	
11	17%	6%	7	\$11.9 M	\$46.1 M	13	
12	17%	8%	10	\$8.1 M	<span style="color:red;">-\$423.5 M</span>		
13	18%	2%	5	\$9.3 M	\$1,352.2 M	9	
14	18%	4%	6	\$6.6 M	\$193.4 M	12	
15	18%	6%	9	\$32.7 M	<span style="color:red;">-\$35.2 M</span>		
16	18%	8%	13	\$4.5 M	<span style="color:red;">-\$1,018.5 M</span>		
17	19%	2%	6	\$13.8 M	\$490. M	12	
18	19%	4%	7	\$7.3 M	\$258.2 M	13	
19	19%	6%	8	\$19.6 M	\$164.8 M	13	
20	19%	8%	8	\$63. M	\$283.3 M	13	
2023	1	14%	2%	4	\$9.7 M	\$689.9 M	8
	20	19%	8%	9	\$8.9 M	<span style="color:red;">-\$93.7 M</span>	

Figure 4.17 DOE applied to generate multiple input and multiple output responses

There are a couple of things to note. As long as the default scenario is scenario 1, I'm comfortable using the top row as just another record of data collected. This allows me to avoid any separation between the headers I've placed above that top row and the data I want to look more closely at below. Note also that I've started to clean the outputs up a bit, replacing things like "#N/A" with null content. These are both useful steps for the simulation data summarization I'm about to conduct using a tool we've seen before: a Pivot Table. In such a Pivot Table, the multiple simulated runs of each combination can be averaged and the standard deviation calculated for each of the performance measures. These summaries can be organized by the two growth rate variations (since they were also included in the top row of the data table structure). Figure 4.18 depicts what this looks like on the PivotSummarization Sheet.

We can develop still cleaner versions of the Pivot Table Summaries to make what we want to demonstrate more transparent to another user. For example, we could create a mirror of the Pivot summary on

The screenshot shows a Microsoft Excel spreadsheet titled "Chp4\_RoughCutAcquisition - Saved". The PivotTable is located in the range E29:N29, with the table header spanning B1:N1. The data rows (1-11) contain various financial metrics such as Average Annual profit at within-year breakeven, StdDev of Annual profit at within-year breakeven, and StdDev of First year of within-year breakeven. The PivotTable itself displays these metrics categorized by the adjusted revenue growth rate (14%, 15%, 17%) and provides summary values like Total and StdDev.

Figure 4.18 Using a Pivot Table to summarize a complex data table

a separate sheet and then modify the formatting (see Figure 4.19 and the CleanedUpSummaries sheet in this workbook). There are a lot of details to look at in this presentation but then again there is a lot going on in the modeling. What is particularly striking in this case are the high numbers in the Annual Profit at Breakeven and Average Discounted ROI estimates at the high levels of adjusted (invested in) Revenue Growth rate. However, these are accompanied by high levels of uncertainty as well (potential for major downside risk). Less risky are lower investments in Revenue Growth rates but high investments in keeping Expense Growth down. They also see earlier points of breakeven. All of this will eventually need to be considered when the acquire/don't acquire decision is ultimately to be made.

#### 4.4.3 Example: Reorder Point System Performance

The RoughCut example incorporated changes across time in its modeling structure. In that case the levels of revenue and cost evolved over time without encountering additional events and without triggering additional events themselves. That is not always the case when considering the impact of policy decisions. Often these decisions drive not only a series of outcomes but also processes that impact other model factors and thus what happens later on in the model. These feedback systems, structured by policy decisions, can be particularly challenging to model. They may require the use of additional tactics for evaluation beyond those we have considered in our discussion of simulation up to this point.

As an illustration, let's return to Lobo's Cantina and consider another key decision they need to make: How to manage their inventory. Inventory policies critically impact the availability of certain stocked items (liquor and dry goods, in particular at Lobo's) but are also highly dependent on uncertain issues such as periodic demand and fulfillment lead times. For some complex policies, or those subject to complex forms of uncertainty,

		Chp4_RoughCutAcquisition - Excel																				
		Elicit Bindley																				
		Tell me what you want to do																				
K	L	C	D	E	F	H	I	J	K	M	N	O	P	R	S	T	U	V	W	X	Z	
2		Adjusted Revenue Growth				14%				15%					17%			18%			19%	
3		Rate Average																				
4		Adjusted Expense Growth	2.0%	4.0%	6.0%	8.0%		2.0%	4.0%	6.0%	8.0%		2.0%	4.0%	6.0%	8.0%		2.0%	4.0%	6.0%	8.0%	
5		Rate Average																				
6		Average of First year of within-year break-even	4.6	5.3	6.5	8.0	5.3	6.0	7.2	9.0	5.8	6.5	7.6	NA	6.0	6.8	7.7	NA	6.2	7.0	7.8	9.4
7		(Std Dev)	(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)	(0.0)	
8		Average of Annual profit at within-year break-even	\$11.5	\$9.7	\$9.5	\$8.8	\$13.3	\$12.6	\$13.1	\$13.4	\$17.9	\$17.3	\$16.5	\$18.0	\$20.6	\$18.6	\$22.5	\$24.9	\$25.8	\$24.2	\$26.9	\$28.9
9		(Std Dev)	(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)	(0.0)	(0.0)
10		Average of Discounted ROI over 13yrs	\$691.3	\$483.1	\$226.2	\$-41.1	\$642.9	\$441.0	\$154.8	\$-150.6	\$691.7	\$417.5	\$111.1	\$-251.0	\$740.0	\$396.2	\$143.4	\$-352.7	\$816.5	\$489.9	\$147.8	\$-364.7
11		(Std Dev)	(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)	(0.0)	(0.0)
12		Average of First Yr of Positive Tot. Disc. ROI	8.3	9.5	11.3	12.5	9.6	10.7	11.9	12.7	10.2	11.3	12.0	12.6	10.5	11.5	12.0	12.6	10.9	11.5	11.9	12.4
13		(Std Dev)	(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)	(0.0)	(0.0)
14		Investment in User Interface																				
		Investment in Data Management	\$7.5M	\$5.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$5.5M	
		Upfront Investment Total	\$7.5M	\$5.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$7.5M	\$5.5M	\$5.5M	\$5.5M	

Figure 4.19 Results cleaned for presentation purposes and interpretation

the use of system simulations may be the only mechanism for assessing their overall effectiveness.

The flowchart shown in Figure 4.20 presents our current challenge and the kind of work needed for analytical consideration.

The primary policy decision in this case is the reorder point; that is, how low should we allow our inventory to get before we reorder. Ostensibly, we could also be deciding on how big those orders should be and making decisions across multiple products we stock. For this demonstration, however, we'll keep things simple. We'll set the amount we reorder equal to the demand expected (average) across the expected (average) lead times for deliveries, plus any difference between the current amount on hand and the reorder point (in case actual demand has depleted inventory a great deal beyond this point). If delivery time is measured in periods and demand occurs per period, then we can ultimately build this formulaically into the spreadsheet.

In this examination we could have multiple outcome measures worth keeping track of. One would certainly be the average amount of inventory held per period. This outcome would be implicitly traded off against the occurrence of stockouts (the less you hold, the more likely you are to stockout). In practice both can have explicitly measurable costs, with inventory held taking up space and rents, and stockouts associated with expensive expedited shipping, late discounting, substitution, or lost sales, depending on context. We'll keep track of both the stockout occurrence as well as the amount of demand lost (e.g., people asking for a particular wine when it is out of stock). Together they form the basis for decision making (policy selection).

The workbook Chp4\_LobosInventory provides an example of how these issues might be incorporated into a model. This is a considerably more

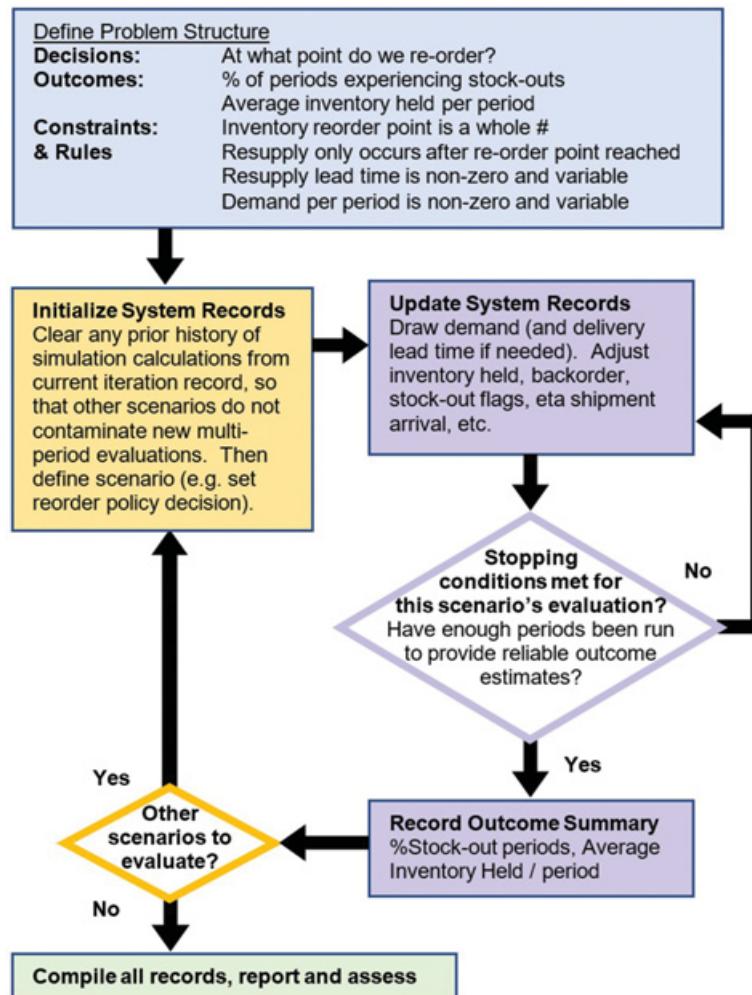


Figure 4.20 Design structure of the Lobo's inventory system simulation

complicated model than the prior examples. Specifically, we need a multiperiod evaluation of this policy. However, in contrast to the 13-year evaluation in the last example, the number of periods here will be in the hundreds. To minimize the amount of space used and to make the number of periods evaluated easy to change without massive redesigns of the model, our cell calculations will have to be a bit trickier. In this example we are storing information in cells that are dependent not only on the value of other cells but also on their own previous values (e.g., how much inventory we hold in period 110 is dependent on how much we held in period 109). This is in place of hundreds of cells otherwise dedicated to storing various details for each period explicitly. The cell calculations, in short, involve circular references to allow us to evolve their values in the

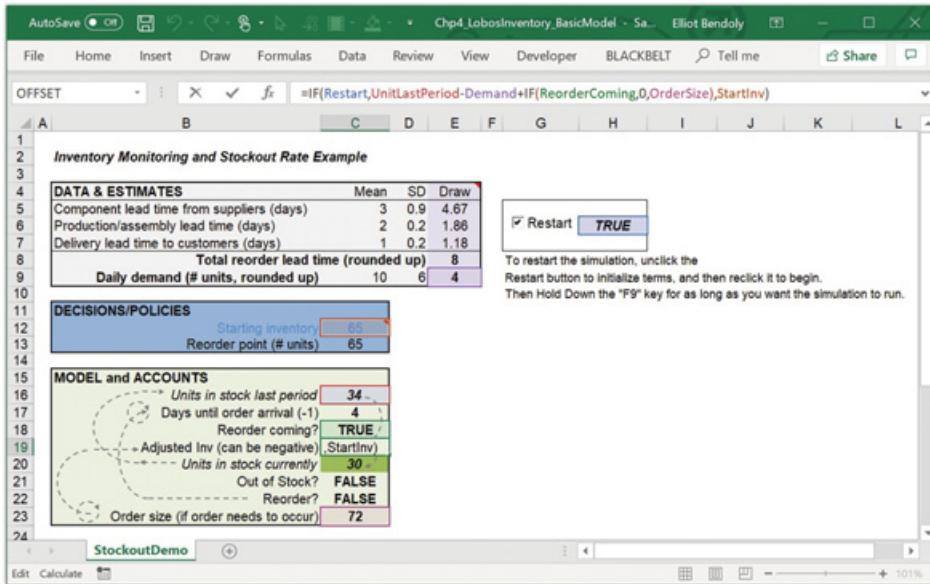


Figure 4.21 Module structure for Lobo's inventory system simulation

multiperiod evaluation of this system; economizing on spreadsheet real estate used, storage, and computation. What we have is a model that itself is a Living Record (Chapter 3) but one that is building a simulation history rather than a query history.

To emphasize this, the circular flow of the activity associated with some of these cells has been annotated with dashed lines. To avoid making the presentation too busy I haven't annotated all of the forward calculations in this case but, if you look inside the cell, you will be able to see these as well. The other thing that you will notice almost immediately is the pervasive inclusion of IF statements referencing a cell named Restart, statements that serve to reinitialize the compiled records when needed (see Figure 4.21).

For example, as shown in Figure 4.21, cell C19 (Adjusted Inventory) contains the following calculation with references to named cells:

$$=IF(Restart,UnitLastPeriod-Demand+IF(ReorderComing,0,OrderSize),StartInv)$$

There's a fair amount going on here. The outer IF statement is the reinitialization control, resetting this cell to the initial inventory level if Restart is FALSE, an action that can take place either by typing FALSE into the Restart cell (H6) or by clicking on the check box next to it (we'll see how to create these check boxes and other controls shortly). If that value is true, cell C19 (AdjustedInv) takes on the difference between the inventory held

most recently (stored in the UnitLastPeriod cell) and the most recent Demand draw. It adds in the potential arrival of orders that might have been placed IF an order has arrived (i.e., is no longer “coming”). This cell can therefore take on a negative value, though the actual inventory level, shown in the cell below it, is set to zero if that is the case. A separate account tracks how much of this demand is actually lost due to stockouts but that is demand never recovered. Lobo’s doesn’t place backorders for customers wanting glasses of sold-out wine.

While this specific calculation in C19 doesn’t reference that same cell directly (doesn’t involve C19, or AdjustedInv), it is part of a circular system of equations, as suggested by the dashed lines in the workbook and in Figure 4.21. Hence, in order to do these calculations we need to be in some form of iterative calculation mode. This workbook is operating in the same mode as the living record example of Chapter 3; Each F9 hit triggers just one recalculation of all cells involved in circularly referenced systems of calculation.

Hitting F9, you can observe the changes taking place one period at a time. In each period, some of the inventory on hand is used up or sold off. After a number of periods the inventory might dip low enough (below the “reorder point”) to trigger an order of more goods, or it might run out entirely, resulting in the placement of backorders. And in certain periods, a truckload of additional goods might arrive, as ordered, to replenish the inventory. These dynamics in some form repeat themselves across the history recorded by the living record.

It should be clear that no single period can capture the overall performance of any given policy in this case (since any given period can be a stockout or delivery, for example). We’ll need to think of system performance in a systematic way; for example, creating averages to get a better sense of policy performance. That is exactly what is taking place in the lower cells in the workbook (see Figure 4.22). Here we are incrementally adding to our counts of the number of periods examined, the cumulative amount of inventory seen, and the occurrences of stockouts and demand foregone. Dividing the latter three counts by the number of periods examined at any given point gives us some systematic measures of policy performance. The more periods we examine, the more representative these average performance levels will be.

Beyond these summary measures, actually seeing the dynamics over a window of system activity can also be insightful. Since we are already using the living record concept to run calculations, we might as well throw in a multiperiod snapshot of how inventory behaves. I’ve provided a 40-period window along these lines below the summary calculations. If you reset the system and hold down the F9 key for at least 40 periods of calculation,

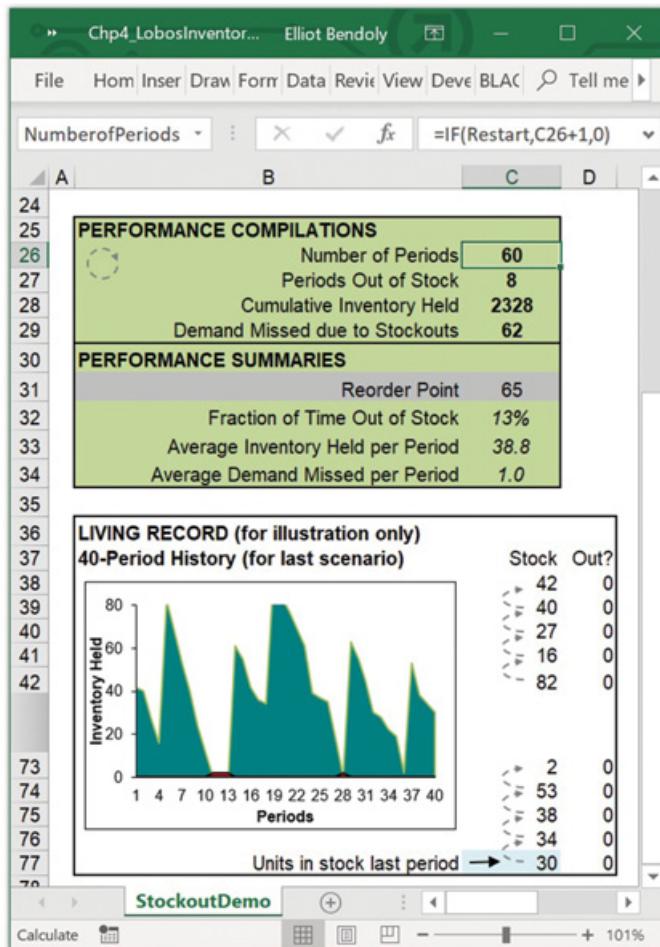


Figure 4.22 System performance summaries and dynamics

you should see the prototypical sawtooth diagram graphed alongside this window of records. Again, we'll discuss how to create graphs of various kinds in more depth in Chapter 5.

To reiterate, in the single iterative calculation mode, each press of the F9 key allows for an additional period to be evaluated and introduced to the base of the living record. So, if you wanted a 200-period evaluation of any single policy (e.g., reorder point at 60 units rather than at 65), you would have to reset the system (Restart = FALSE, then Restart = TRUE) and hold down the F9 key until the number of periods examined was 200. That's a fair bit of work in itself. You could alternately create a 200-row data table to both trigger the 200 periods of calculation and keep track of the performance of this system over time. But you might not actually want to store 200 periods of data. The limitations faced by either approach alone are not unlike those

faced in the RoughCut acquisition example. We'd like to do more: multiple policies reviewed for comparison, a sufficient number of periods examined, multiple performance measures tracked, and potentially multiple reexaminations of a given policy (if we felt that simply increasing the number of periods of one examination was not sufficient).

#### *4.4.3.1 Macros for Repeated Examinations in Iterative Calculation Mode*

Let's consider what this would require in our current single iterative calculation mode (maximum iterations=1). Say we want to slightly change the policy under consideration so that the reorder point is 75 units. To record the summary results for this policy we would have to first change the policy we wanted to evaluate by typing in 75 for the reorder point value. Then we would have to carry out the following steps, at minimum:

- 1) Reset the spreadsheet's summaries (type FALSE and TRUE for Reset).
- 2) Press F9 as many times as you think is necessary; some logical stopping rule could be applied here (e.g., stop when NumPeriods gets to around 200).
- 3) Pick a clear spot in the workbook to save these summaries.
- 4) Copy and “paste-special” the values of those summaries.

If this is our plan for creating records to compare different policies, it's going to get very tedious very fast. Fortunately, there are some alternatives available to us. At least one set of approaches allows us to remain in this specific iterative calculation mode but requires us to allow Excel to create a little code in the background (called a “macro”). An alternative approach will involve changing the number of calculations each F9 hit (or each data table row) triggers. Let's look at each approach in turn.

We can use the Macro function to record a set of actions so that we could get Excel to repeat that same set of actions on command (and in one click). Macro functionality is found under the Developer tab, as seen in Figure 4.23. If you don't see the Developer tab, go to File>Options>Customize Ribbon and make sure the Developer tab is checked.

The Record Macro button in this ribbon allows the actions you take within the spreadsheet environment to be recorded as code, as part of your file, so that you can ask Excel to repeat that set of actions in a single command. We will learn more about how this works in the coming chapters. For now we'll just start with a demonstration of recording the steps 1–4 listed earlier for reinitializing the simulation model's system of calculations and storing the evaluations of a policy under examination. To keep things organized we'll store this data in an additional sheet called MacroRuns.

We begin our recording session by clicking on the Record Macro option in the Developer tab to open the Record Macro dialog box, as shown in Figure

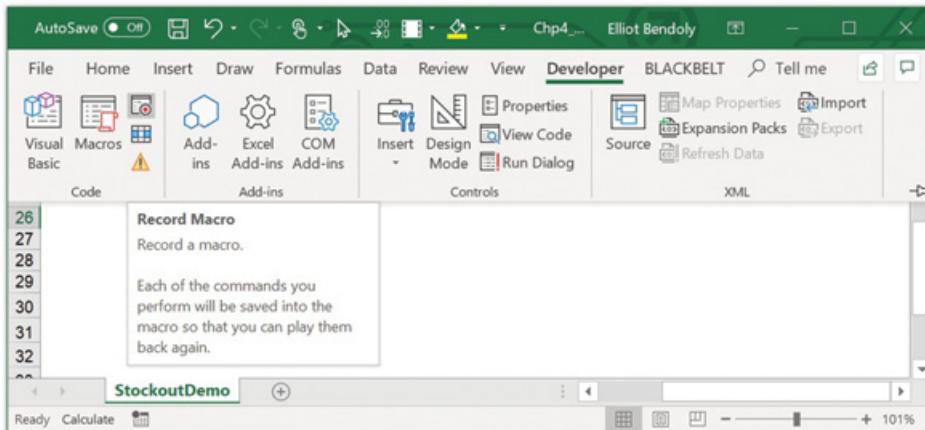


Figure 4.23 Accessing macro recording capabilities in Excel

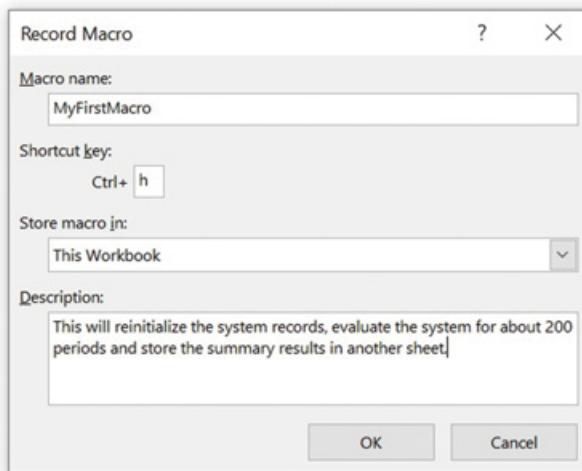


Figure 4.24 Initial specification interface presented for macro recording

4.24. In this dialog box, we can name our macro, describe it in depth, and even create a shortcut key for later execution (we will choose Ctrl-h for demonstration in this case). The default name Macro1 has been selected by the system but I've replaced it in this case (when you try this yourself, use a different name to avoid confusion). Once we are satisfied with these details, we can simply click OK and the macro will start recording (most of) the actions we take.

With recording “on” you should now see a small square where the Record Macro once was. You should also see a similar square at the bottom right of your Excel application window frame. This is critical. Those buttons allow

you to stop your recording session – and yes you always have to stop, otherwise recording will just continue.

Now we are just going to do everything that steps 1–4 have outlined, plus a couple of additional moves. First, after typing FALSE into the Restart cell, I’m going to hit F9. I’ll do the same after typing TRUE, just to make sure the other calculations respond to these changes. Second, when I paste those final summary values, I am going to flip them from vertical to horizontal. I’ll be using cell A1 of the MacroRuns sheet as a destination for that paste, right-clicking, selecting “paste special,” and specifying values as well as the check box that says “transpose” (for the horizontal flip). With A1 still selected, I will insert a row using the Alt-i-r command (or right-clicking and selecting Insert>Entire row). This will shift my record down and prevent it and other records from being overwritten whenever these actions are recalled. I’ll then return to the Stockout Demo workbook. Finally, I’ll stop the macro recording session (clicking on the small square at the bottom right of the Excel window, for example).

The macro is now saved and available for future use, as long as this workbook is open. If you want to have it available to your workbook in the future, make sure to save your file as a Macro-Enabled workbook, otherwise the macro (which is now just VBA code temporarily retained in a hidden section of your file) will be lost. The workbook I’ve provided is already of this format (note the slightly different appearance of its icon). We’ll look at the code later on.

There are several ways to activate a recorded macro at this point. The first is menu driven and involves returning to the Developer tab, selecting the Macros option, and opening the Macro dialog box. Choose the macro you want and then click Run. Later we will discuss ways to create special buttons elsewhere to call the macro. For now, a much faster alternative, however, is simply pressing Ctrl-h (or whatever the shortcut is that you’ve specified). With this option available, we could run this macro any number of times, manually modifying the reorder policy decision before each call, and in turn create a large record of summaries to evaluate.

#### *4.4.3.2 An Alternate Iterative Calculation Mode and Auto-toggle for Repeated Examinations*

Entering into the world of macros and other code structures in Excel provides us with almost endless possibilities for development, at least at the proof-of-concept level. But let’s talk about the drawbacks in the method just described.

First, it’s very easy to miss a step, or to include one you don’t want; for example, forgetting to return to the StockoutDemo sheet at the end, changing the reorder policy while recording, forgetting to turn off the recording

session at the end, and including extra steps. Any mistake can be edited on the back end but you might not want to get into code yet. The alternative is to just start the recording from scratch. Further, with all of the hits to F9 (holding it down until you get to about 200), the macro recording process might actually register a couple more of these than you intended – not a huge deal but it can be a bit surprising. Apart from this, the replay of the macro can seem slow; you are editing the content of a cell 200 or so times and everything else (including the graphic) is getting updated as well. These are not necessarily deal-breakers for this method, but they inspire us to consider some alternate approaches.

The other approach we've suggested involves changing the way in which iterative calculation mode runs. Right now each F9 hit recalculates everything once. But if we go back into File>Options>Formulas, we can adjust this. I have this set up for us on another workbook but before we open it, shut down Excel. As mentioned in Chapter 3, Excel will operate in only one mode of calculation at a time and your setting, whatever it is, gets saved with the books you are working in when you save those. To avoid having other books inadvertently set to this new adjusted mode, it might be best to just close them down. Now, opening the workbook Chp4\_LobosInventory\_200it+DT, you will find the same model structure we were working with before, plus some additional features. We are in a 201-iteration mode here. Each F9 hit now recalculates all random number pulls and cells in the circular reference system 200 times. The additional iteration (the "1" in "201") is an additional step that automatically toggles the Restart cell and hence drives the system reinitialization process prior to these 200 periods of evaluation. Let's talk about this automatic toggling of Restart first.

Figure 4.25 focuses in on this mechanism. Recall that in the one-iteration example we had a single check box adjacent to a single cell (Restart).

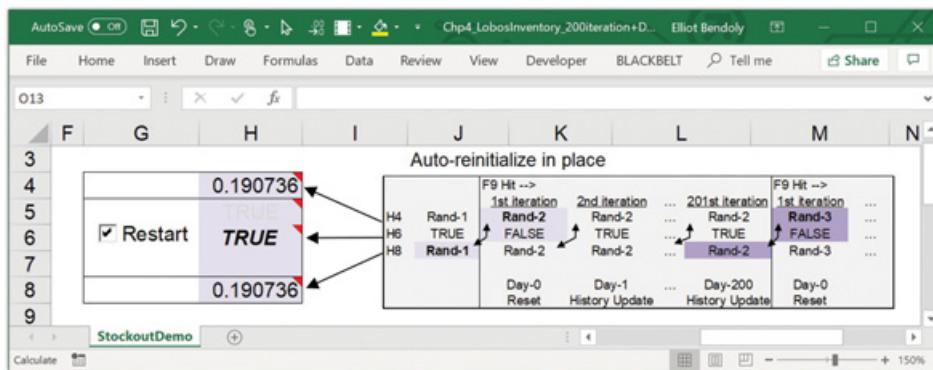


Figure 4.25 Auto-reinitialization mechanism in multi-iteration mode

Checking would change the value in Restart from TRUE to FALSE and back. This still happens here but now another mechanism is in place to do this as well. Now we have several cells working together. The check box directly affects cell H5 (named “CheckboxCell”). The Restart cell (H6) looks to that cell for cues but it is also looking to see if cells H4 (named “newrand”) and H8 (named “pastrand”) are the same. H4 draws a simple random number. And even though H8 pulls on the contents of H4, it also references itself. That means it will be calculated only after H8 and, because of this, H4 and H8 can in fact differ momentarily at each F9 recalc (or anything else that prompts a recalc).

The graphic to the right of these cells describes what happens when F9 (for example) is hit. First H4 (newrand) draws a new random number. Then H6 (Restart) executes its formula:

$$=IF(\text{CheckboxCell}, IF(\text{pastrand} <> \text{newrand}, \text{FALSE}, \text{TRUE}), \text{FALSE})$$

If CheckboxCell is TRUE, H4 is compared against H8 and they are found to be different (since H8 has not yet been updated). Then H8 takes the value in H6. That’s iteration 1 of 201. In the next iteration, H4 isn’t changed, since it isn’t part of a circular reference (i.e., it has no reason to be recalculated). This is in contrast to other random draws like E5–E9, which have circular references, so they will in fact be updated in iterations 2–201, just like H8. H4’s draw happens only once because its calculation is not dependent on any circular reference and thus iteration mode sees no reason to attempt to do more.

The end result? We don’t need a macro to reinitialize our system. We also don’t need a macro to get us through 200 periods of evaluation. All we need is something that prompts a recalculation. And while F9 is one such prompt, there are others; data tables, for example. Each row in a data table can involve sending content to a model for model recalculation. I’ve added a data table to this workbook to demonstrate how we can take further advantage of this in the presence of the auto-reinitialize trigger in 201-iteration mode. In Figure 4.26 we see the data table structure, not unlike the multioutcome structures of the previous two examples. Here, as in the first example, we are using the left column to contain different values of a single decision (if we had more decisions to consider, we might use a DOE approach as in the second example).

Once again, this is a multioutcome, single-input data table. Clicking on any cell in the interior shows this to be case in the formula bar: We see `{=Table(,C13)}`, with the curled brackets designating the matrix nature of the data table, the first parameter blank (designating that the top row should contain formulae), and the second parameter equal to C13 (meaning that this cell, the reorder policy cell, is the destination for the values in

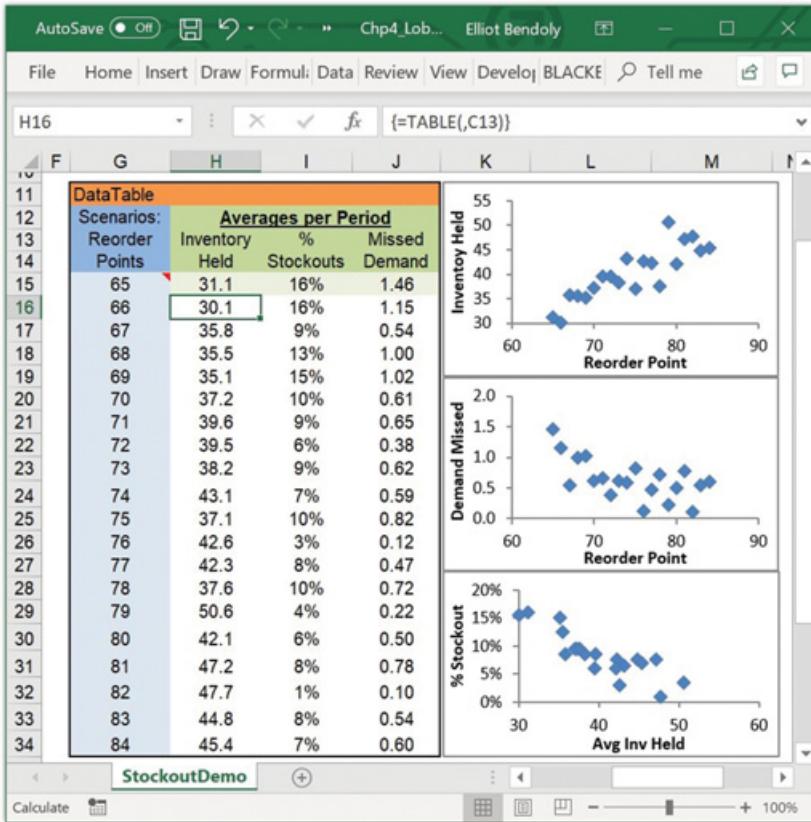


Figure 4.26 Data table structure and assessment in multi-iteration mode

the left column prompting outcome recording in the table). To further illustrate what the modeling is yielding and to emphasize that each row contains data from the same set of 200 iterations in this case, I've thrown in some simple scatter plots to visualize things. As one would anticipate, as the reorder point increases, orders are placed earlier, while more inventory is available, yielding higher average inventories per period but lower levels of demand missed. Higher inventories held are similarly negatively related to overall stockout incidents. How one trades the cost and risks of stockout versus that of excess inventory holding is another matter (for Chapter 7). How we maximally leverage graphics such as these scatters to tell our stories is the subject of the next chapter (Chapter 5).

## Supplement: Control Made Friendly

In our discussion of simulation we've come across a couple of departures from the traditional spreadsheet environment, just in passing. These were

small buttons present in the final cleaned report of the second simulation example and the check box active in the third example. What are these and where did they come from?

Excel provides at least two sets of resources with which to develop visually intuitive and user-friendly graphical object interfaces. One set is referred to as ActiveX controls; the other is referred to as Form Controls. They offer seemingly very similar capabilities and often you can get the same task accomplished with both. These controls allow developers to add elements such as spin buttons, check boxes, option buttons, and combo boxes (drop-down menus), among other resources, to their spreadsheet as alternatives to changing values in cells directly. Form Controls often provide a simpler interface for developers, but I find that ActiveX controls ultimately provide more versatility and a greater range of options. We'll stick to the ActiveX controls for this discussion. A functioning assortment of these controls can be found in the Chp4\_SampleControls workbook (see Figure 4.27).

A text box ActiveX control duplicates the contents of any individual cell in a workbook, but, because it's an object unto itself, it can be positioned

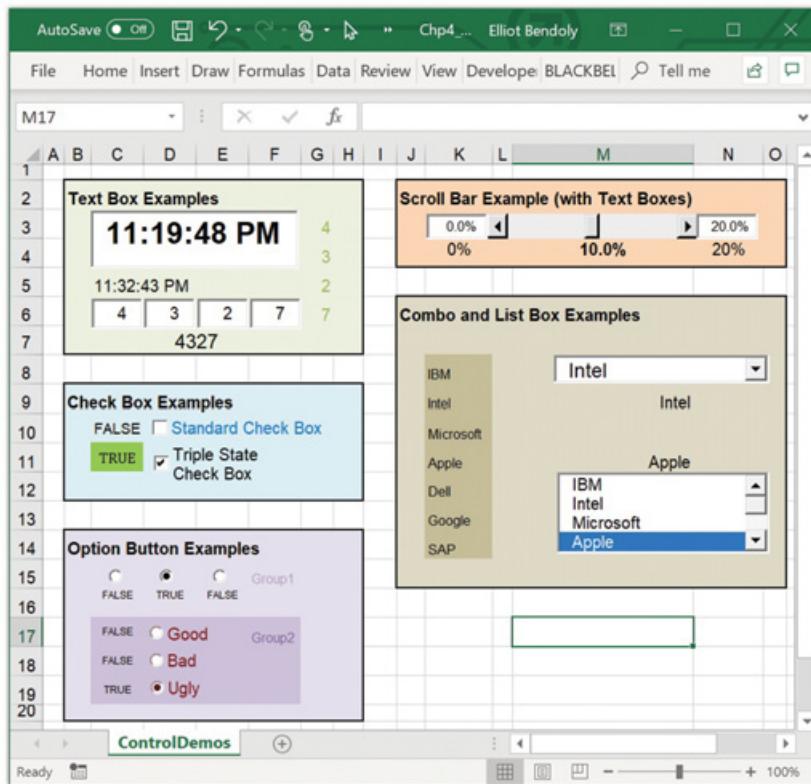


Figure 4.27 Example embedded ActiveX controls

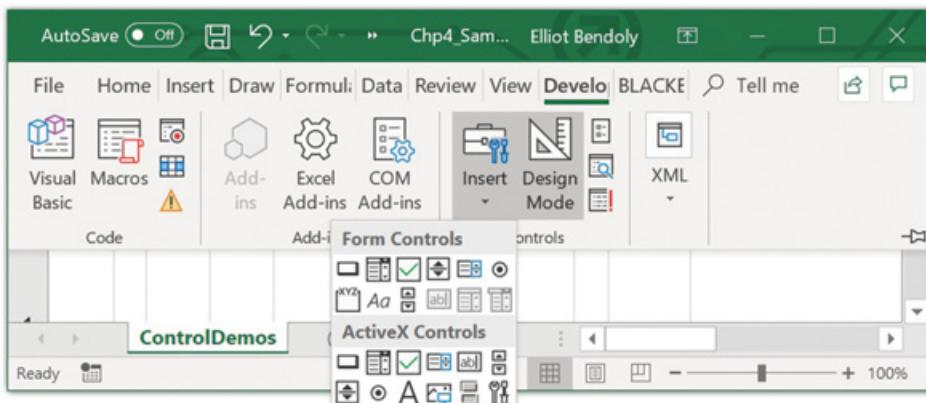


Figure 4.28 Controls available through the Excel menu interface

anywhere in a workbook. This can be valuable because although cells in complex decision-support environments may be difficult to relocate meaningfully (without messing up other parts of the design), the locations of these text boxes has no impact on what's going on behind the scenes. You could even group the text box with other objects, such as graphs, so that when you move a set of objects in your spreadsheet, the text box moves along with that group. To create a new text box (or any other control for that matter), select Insert under the Developer tab, as shown in Figure 4.28.

Click the Text Box icon in the toolbox to generate a new text box at any point in the spreadsheet. When the text box is created, you will automatically enter Design Mode, where you have a wide range of control properties that you can edit. Right-click on the new box and select Properties to open the Properties dialog box. In this view, an array of attributes can be modified, most critically in this case the LinkedCell. Specifying a single cell in this field (e.g., C3) will allow for direct communication between the text box object and that cell. Changes to either one will now be reflected in the other.

The Properties listing for all other ActiveX control objects will appear similarly. Those associated with an option button in the Chp4\_for example are shown in Figure 4.29.

Option, or radio, buttons are designed to be used as one of a set; in other words, each radio button is used to represent specific alternatives, only one of which may be active or relevant at a specific time. An example would be the selection of a single candidate for a specific position during voting, a specific shipping option for an order, or a specific accounting classification for filing an item on an expense report.

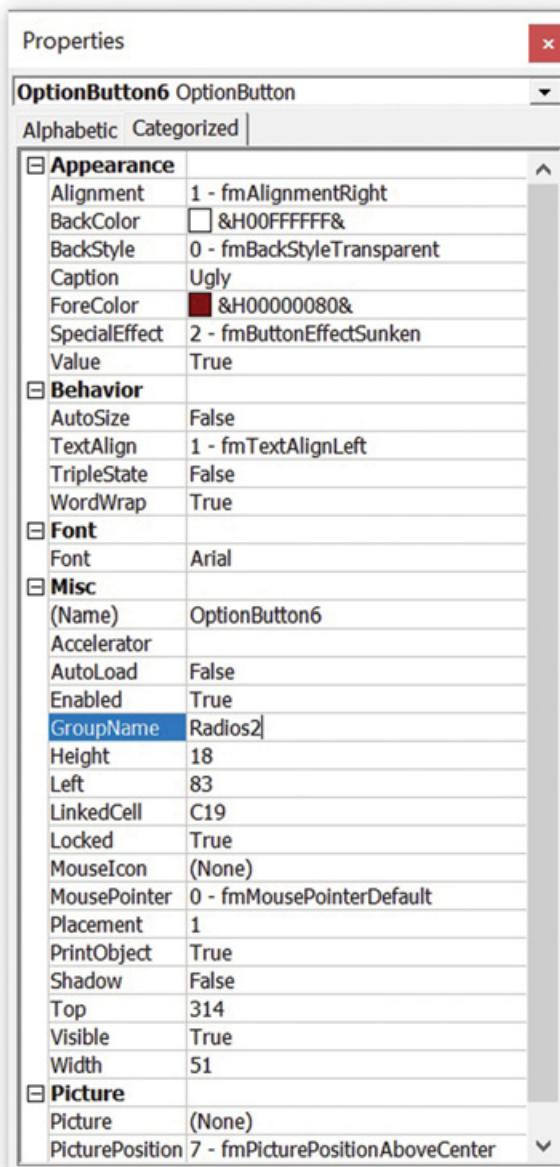


Figure 4.29 Modifying the properties of a text box ActiveX control

In light of this, a critical property of option buttons is the `GroupName`. This property designates the set of alternative radio buttons that each individual button works in conjunction with. By default, new radio buttons created on a worksheet will receive the name of that worksheet as their group name. But similar to all other properties, you can (and usually should) change this. For example, for one set of buttons that represent

interchangeable options, you might use the group name Radios1; for another set of buttons that are not dependent on the choice made in the first set, you might use the label Radios2 (see Figure 4.29 where this is used). Specifying different groups for independent sets of radio buttons will ensure that they don't interfere with one another's functionality. It is also worth noting that, unlike text boxes, option buttons also allow you to add fixed text captions that accompany them on a page. These are not to be confused with the name of these buttons, just as you would never confuse the text contained in a cell with the actual name of a cell. Names are always unique identifiers in the ActiveX control realm (as with cell names within a workbook). You can have two controls with the same content, but no two controls in the same environment can share the same Name property.

Check boxes, which we've already seen applied in the third simulation example, offer another convenient control that allows users to toggle between two settings (e.g., 1,0; Yes, No; On, Off; Restart, Stop) rather than having to type these values into cells, for example. In contrast to option buttons, check boxes are typically not restricted based on the selection of other check boxes. An interesting feature of check boxes is the ability to use them in a *triple state*. In such a case, each click on the box will move the user through the states of TRUE, FALSE, and #N/A (other). This could be useful if, for example, you had three alternative settings that you would like to make available through this kind of object-oriented interface.

Scroll bars are also illustrated in the Ch4\_SampleControls workbook. These controls provide mechanisms by which to select a relatively continuous range of values bounded by upper and lower limits (rather than by a discrete set of options described by check boxes and radio buttons). For example, you might want to allow a user to specify an interest rate between 5 percent and 7 percent, a minimum average labor force scholastic competency between 900 and 1,200 (e.g., on the SAT), or a maximal budgetary allowance between \$10,000 and \$35,000. A scroll bar might be a nice choice for an interface on such a decision. The biggest limitation here is that the upper and lower bounds specified in the properties of scroll bars must be integers. As long as you are comfortable rescaling after the fact, this isn't really an issue. In the example provided I've even shown how you might couple two text boxes to a 0–10000 scroll bar, allowing for the specification of any decimal/percentage end points in those text boxes, which in turn provides input for the rescaling of the scroll bar data to a fairly specific decimal/percentage.

Lastly, it is worth discussing the list selection controls available. Most common are the combo box (drop-down menus) and list box options. One

of the additional properties of these controls requires the specification of a list of alternative options, which can be presented as a sequence of cells in a column of a worksheet (as in Chp4\_SampleControls). A combo box with such a reference list specification provides a compact form that can be expanded by the user for item-selection purposes. List boxes also make available special “MultiSelect” property options, permitting multiple entries to be selected. These choices will not appear directly in the LinkedCell but can be accessed using code. We will see more of this in Chapter 9.

It is certainly worth noting that an alternative to this kind of functionality might involve use of the Data Validation resource under Data>DataTools. This approach provides some listing options that are limited but fairly easy to implement and modify. Essentially, it allows you to restrict the content of a cell to match only one of the elements in a range on that same sheet. However, we'll find that mastering the use of similar noncell structures and ActiveX capabilities will be more effective and come much more in handy when developing more sophisticated user interfaces.

## PRACTICE PROBLEMS

### *Practice 4.1*

Using the approach demonstrated in Section 4.4.3.2, build a report that shows not only the average but also the best (MIN) and worst (MAX) cases for the cost (holding) and stockout rates (as before, across 200 iterations or trials). Create a data table with six columns of formulae or references. Extend your data table to allow this to be run 10 times for each of 11 reorder point policies ranging from 60 to 70 (i.e.,  $11 \times 10 = 110$  rows in your data table). Now summarize these performance measures in terms of overall average levels, average mins, and average max values. Comment on how average performance might not be completely informative with regard to “best policy” selection.

### *Practice 4.2*

Reconsider the Lobo's Reservations example simulation from the previous practice section. We selected random numbers based on a specific level of variation and we limited ourselves to a few specifically structured scenarios. But we might want to find out how different the performance of this system might be given alternative overbooking charges and alternative levels of variation (more or less) in customer dollar contributions.

Create three option buttons to depict different possible levels of variation in the profit per party: standard deviations of \$25, \$50, and \$100. Create an ActiveX combo box to allow a user to choose among the application of three distinct overbooking costs (\$150, \$200, \$300). Create a data table that evaluates just the estimated profit for policies of 15 to 25 reservations (11 columns), each 200 times. In the two rows

below the table, calculate averages and standard deviations for the estimated profit of each policy and use conditional formatting to highlight the highest profit and its policy in blue. Test to make sure the results change as expected when option and combo box selections change. Four rows below the table (two rows below these summaries) paste “as values” the results for the following two scenarios: [standard deviation in profit-per-party: \$100, overbooking cost: \$150] and [standard deviation in profit-per-party: \$25, overbooking cost: \$300].

# 5

## Visualizing Data, Dynamics, and Risk

**Objective:** *Develop an awareness of the variety of both simple and more nuanced data visualization tactics and tools available. Appreciate how to develop visual renderings that best meet the needs of data examination and storytelling for the audience in question.*

I've had an internal debate around where to best position this particular chapter. In the first three chapters of my book *Visual Analytics for Management*, I describe human biases pervasive in visual interpretation and best practices in visual development that take these into consideration. And while later chapters in that book describe a range of cases in practice, what the third edition of the current text required was a discussion of the tools and tactics available to common platforms such as Excel and associated applications. Why not broach this subject immediately after a discussion of data acquisition?

The answer has everything to do with what people tend to focus on in their design of visuals, namely, measures of centrality: means, medians, modes, and so on. And that is a big problem, since visuals of these summary statistics only reinforce a reliance on these in decision making (and a sidelining of other data features). Individuals already struggle to incorporate risk into planning, so it would have been irresponsible to discuss visualization in greater depth prior to demonstrations of how risk might be evaluated and used in modeling. Now that we have seen some of that variability and where it comes from (variability that we can't account for or control, uncertainty around the impact of predictors, etc.), we can take a more comprehensive look at our data, its relationships, and contexts.

Visual examinations are key preludes to our eventual search for prescriptive solutions. They will help justify assumptions and close gaps in these. They will help us scrutinize whether the recommendations made by our tools make sense and are practical, whether the risk they suggest is acceptable, and whether there are other solutions worth further consideration. They will

prompt us to examine alternative predictive and prescriptive models and may inspire us to collect additional data. As we will discuss in depth in the next chapter, and have alluded to in prior discussions, most management problems for which prescriptive solutions are sought can be represented by three standard elements: Objectives, Utility variables (decisions impactful to target objectives), and Connections (including those that limit actions).

### **Objectives**

**For example:** Maximize profit

Provide earliest entry into market

Minimize employee discomfort and turnover

### **Utility variables**

**For example:** Determine what price to use

Determine the length of time tests should be run on a new product or service

Determine the responsibilities to assign to each worker

### **Connections:**

**For example:** Profit is revenue less cost but expenditure can also increase revenue in relatively complex ways.

We must test enough to meet minimum safety regulations.

Ensure responsibilities are shared by two workers at most.

Objectives, utilities, and the connections that bind them can be visualized graphically and in many cases should ensure that mathematical reductions of reality are still actually representative of that reality (and not simply a mathematical exercise). Clearly, this kind of scrutiny can be to the benefit of analysis and general insights. Our initial discussions in this chapter will focus on the visualization of content predominantly characteristic of these relationships to prepare us for our journeys into the prescriptive analytical realm.

## **5.1 Approaching the Task of Visualization**

In most business scenarios, managers are faced with making a set of decisions that impact a final outcome (objective). Simultaneity of multiple decisions tends to make the decision process more complex and can make the rationale for specific decisions difficult to describe. As the old saying goes, a picture is worth a thousand words. Some pictures are cute but may say very little – at least initially. Misleading suggestions can throw a decision maker off his or her game. It's the responsibility of individuals charged with providing decision support to clarify what limitations exist in a graphical representation – that is, what to take with a grain of salt, and where consistency and relevance exist.

We are not without numerous options here either. There are plenty of graphs that can be built through Microsoft products and affiliated applications, ranging from simple bar charts to more sophisticated plots. For

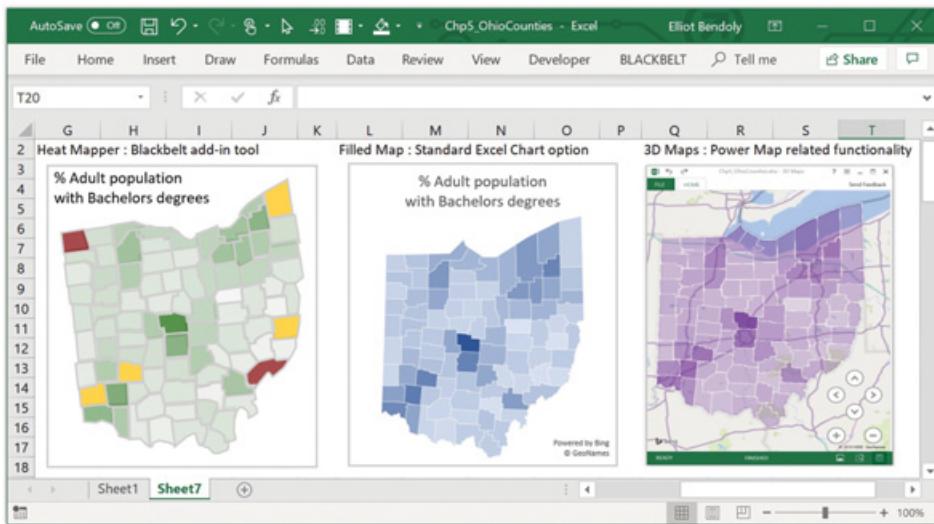


Figure 5.1 Heat maps (chloropleths) generated by Power Map and the Blackbelt Ribbon

example, geographic heat maps (chloropleths) are readily available through standard filled map charts and MS Power Map capabilities built into contemporary versions of Excel, but they are also easily embedded through add-ins such as the Blackbelt Ribbon (see Figure 5.1). Depending on what you need, in terms of detail, flexibility, and dynamics, in order to convey your message effectively to your audience (which may be just yourself), one option alone, or multiple in tandem, may prove most effective. In many other situations, multiple in tandem may be required. What works best is necessarily going to be context-specific.

While we will be examining each of the options shown in Figure 5.1, chloropleths are certainly a special case in terms of the kind of data relevant to such visuals. Other kinds of plots, for other kinds of data, can be just as nuanced and various in terms of tool selection and design. And certainly, depending on the data, message, and audience, alternative visualization options, even much simpler ones, can be much more effective. The availability of enterprise level visualization tools and mastery of their options can only get you so far without an appropriate understanding of your data, message, and audience.

Fortunately, the basic technical logic required to create and manipulate any graph in Excel and most affiliated applications is largely the same. Given a selected subset of data to graph, you need to specify how to use it (e.g., as a label, or as data to be plotted) and then specify the particular aesthetic features of the resulting graphical presentation. What is always more

challenging is that which is nontechnical, specifically, ensuring natural flow, separation, emphasis, access, and so on, so that our visual gets (or will get) the job done, given the nature of our audience, our data, our analysis, our reality. For this reason we will begin our review with the construction and manipulation of simple but representative types of graphs in Excel – bar charts and scatter plots – thinking about both technical steps and best practices in design. We will then consider alternative visual renderings of data, the incorporation of analytical intelligence into graphs, and the dynamic enrichment of visualizations.

## 5.2 Working with Standard Charts

While there are a variety of standard charts available to Excel, most of these operate using comparable interfacing and editing options. As a result, becoming familiar with how just a couple of these forms can be used (and misused) can provide sufficient preparation for the various alternatives available.

### 5.2.1 Bar Charts

Because bar charts represent a fairly expansive range of graphing possibilities, from the simplistic to the information packed, they serve as a useful foil for describing the various options available when developing visuals in Excel. These kinds of charts are useful when you have a small set of discrete categories, each of which describe and compare along a single measure. For example, we might want to chart the degree to which legacy concerns differ across individual projects or clusters of those projects. Or we could be interested in depicting how common it is for one, two, three, or more parties to not show up, despite making reservations. Or we could be interested in depicting the kind of per-period inventory levels that our system simulations project. After all, we've seen data of this kind in the previous two chapters. In all scenarios, the X-axis delineates discrete cases for which details of observations (Y) may differ.

The arrangement of these cases along this X-axis can be obvious because the cases have both an inherent order and consistent intervals (e.g., period 1, period 2, period 3), but that's not always the case (Software Development X, "Harry's" Project, Implementation-B, etc.). Whatever the situation may be, just make sure that you are being transparent in your decisions regarding arrangements along this axis. If values are strictly ordinal but their separations are not equal, make that clear. If they are largely nominal, describe the justification for their ordering (e.g., by timing, alphabetical, grouped by type, sorted by the Y value). This is one of the most common issues people do not

spend enough time on before making bar charts. So think about what would be most meaningful before moving forward and be absolutely clear on what your choice is.

### 5.2.1.1 Preparing Data for Robust Graphics

Let's look into a simple example of how to create the most basic of bar charts and what kinds of modifications can make the use of a bar chart more informative. If we open the file Chp5\_DodechaGraphs, we'll find some data that we've seen before. Specifically, this book contains raw data and the analyses derived from data reduction exercises such as those in Chapter 3. Recall that we used Principal Components Analysis to consolidate a set of 32 factors (columns) and then used a form of cluster analysis to group our 115 records (rows) into three to four groups. At this point we might be interested in seeing how truly distinct these groups are with regard to some of the original attributes.

I've provided some of the summary statistics for a given attribute (AX6) and a specific clustering (AW7) in tables to the right of the data (see Figure 5.2).

The construction of these tables is worthy of a little discussion before we start using them to create visuals. Unfortunately, while the main data in the left of this sheet is rich with detail, it might not be in the best shape for our current needs. For example, although we have all of the group assignments (columns AS–AU), they are not sorted in any particular fashion. So we can't simply select a range to average. Sorting all of the data by one clustered grouping (e.g., using a Filter) would almost certainly require unsorting the groupings in an adjacent column. A Pivot Table could allow an organized

	AW	AX	AY	AZ	BA	BB	BC	BD	BE
5	Cluster Differences in B3 (Industry Instability)								
6	B3 (Industry Instability)								
7	Clustering_3	N	Avg	Stdev	0%	100%	50%	25%	75%
8	Group 1	51	4.14	0.82	2.00	6.00	4.00	4.00	5.00
9	Group 2	34	3.47	1.19	1.00	6.00	4.00	3.00	4.00
10	Group 3	30	4.17	1.16	2.00	7.00	4.00	3.00	5.00
11	Group 4	0							

Figure 5.2 Conditional variation and percentile summaries via the Blackbelt Ribbon

summary, but let's consider another alternative for extracting summary statistics for a given clustered grouping – something robust to both our selection of attributes to examine as well as our approaches to visual examination.

All we really want are ways to get summary statistics of a selected measure, specific to each clustered group. Fortunately, Excel gives us exactly the kind of formula we need for part of this task: AVERAGEIF. It requires specifying the range over which particular cases should be searched for (e.g., the range AS:AS if we are just interested in the first clustering results), the criteria that a subset of those cases must meet for use (e.g., the value 1 if we are interested in just the first group extracted in that clustering), and the range of associated numbers that will be drawn from in forming that case-specific average (e.g., the range B:B if we are interested only in the first original measure in our data). Altogether, the AVERAGEIF formula for this specification would look like this:

$$= \text{AVERAGEIF}(\text{AS:AS}, 1, \text{B:B})$$

Simple enough. If I wanted to make sure that this worked for a set of cluster numbers in the range AW8:AW11, I could upgrade this to the following so that I could copy it down and it would work across rows 8:11.

$$= \text{AVERAGEIF}(\text{AS:AS}, \text{AW8:AW11}, \text{B:B})$$

That is, assuming I didn't want to name the range AW8:AW11 something meaningful, like "clusternumbers," which I'd be justified in doing. Now let's take another step. Let's say that I have a formula that calculates the number of records in each cluster (COUNTIF usage, for example) and I've named that range Ncount (the range AX8:AX11 in the workbook). With that in place I could avoid bothering to calculate averages if some minimal threshold of observations (at least two) isn't met by embedding the AVERAGEIF statement in another conditional statement as follows:

$$=\text{IF}(\text{Ncount}<2, "", \text{AVERAGEIF}(\text{AS:AS}, \text{AW8:AW11}, \text{B:B}))$$

Finally, let's make this even more robust. Let's allow the range of clustered group designations (AS:AS) to shift based on a choice and allow the column of data to be averaged (B:B) also to shift based on choice. As for these choices, we can select a cell like AX6 and equip it with a kind of data selection mechanism, available through the Data Validation tool. As in Figure 5.3 I will select a range of values to serve as a "List" that I want to select from (say the headers in B1:AG1).

With Data Validation in place, AX6 (or AX6 merged with AY6) provides a drop-down menu of these options. The MATCH function (in AZ6, for example) can then provide the specific location of the column containing that

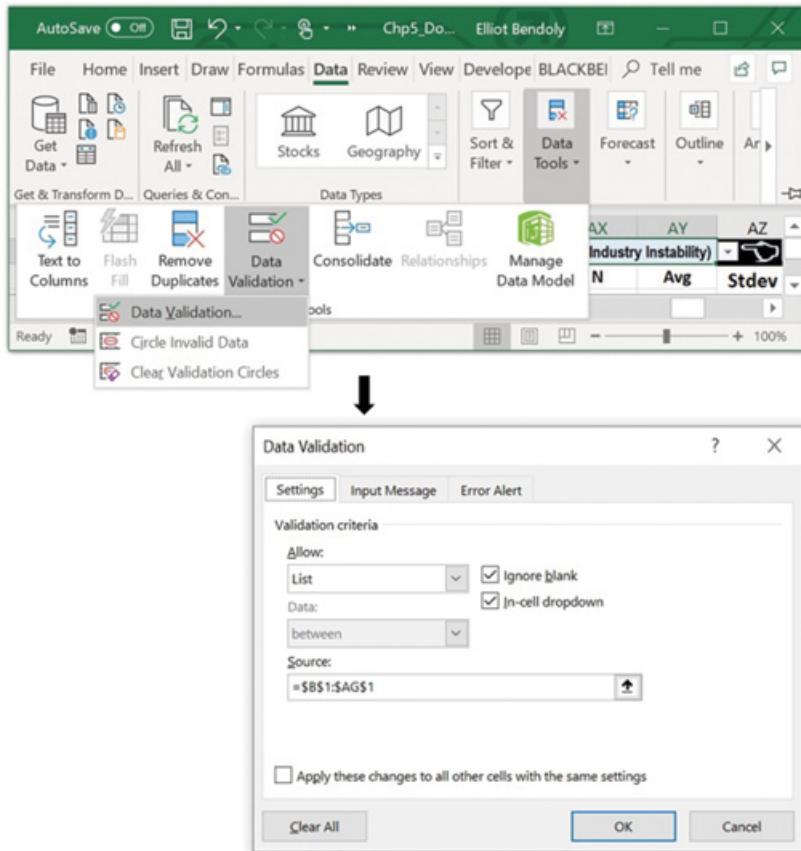


Figure 5.3 Adding Data Validation rules for data interface enhancement

selection (=MATCH(AX6,B1:AG1,0)). With that in hand, the OFFSET function can shift our data range from B:B to the column we actually want to build an average of. And we can do this for both the selection of the data attribute to be averaged (AX6) as well as the cluster grouping to use (AW7). With some additional naming (Cluster1 is AS:AS, Data1 is B:B, and ClusterPick and DataPick are the respective MATCH results of the Data Validation selections), we then have the following equation in cell AY8 for our first average:

$$= \text{IF}(\text{Ncount}<2,"",\text{AVERAGEIF}(\text{OFFSET}(\text{Cluster1},0,\text{ClusterPick}-1), \$\text{W}\text{8},\text{OFFSET}(\text{Data1},0,\text{DataPick}-1)))$$

It's a longer formula than those we have seen so far – OFFSETs embedded within AVERAGEIF, within an IF – but it gets the job done. Every time a different selection is made in AW7 or AX6, the right averages get calculated for the associated clusters.

Now AVERAGEIF, in tandem with other tactics, does a great job for the task of calculating these conditional averages. Unfortunately, in the current version of Excel, there are not a lot of other combined conditional calculations – nothing for standard deviation conditional on a criterion or for the minimum value conditional on a criterion. That's understandable since there are already so many functions that come standard and it's fair to ask how many more Microsoft would really want to build in as standard. Still, given the aforementioned challenges people already have in appreciating risk, it's a problematic omission.

To cover this gap I've written a couple of additional functions of my own of this type, using the same structure as the AVERAGEIF function. They also come along for the ride with the Blackbelt Ribbon add-in. The first is StdevIf, which does exactly what any user of AVERAGEIF would imagine. It calculates not the average but the standard deviation of a set of cells in a column of data, if those cells are in a row where the conditions of the selection criteria are met. The simplest form of this function, calculating the standard deviation across the values of the attribute in column B:B, for the first group in the first clustering would look like this:

$$= \text{StdevIf} (\text{AS:AS}, 1, \text{B:B})$$

As in the case of averaging in this workbook, I use a more complex form to accommodate alternate choices in the clustering results and data to be summarized. A similar but even more flexible function is provided by PercentileIf. This function looks at all the data in rows meeting the specified criteria and returns the value at the kth percentile specified (a fourth parameter for this function). If I wanted the minimum (0th percentile), my fourth parameter would be 0. For the maximum, I would specify 1 (equal to or greater than 100 percent of all relevant data); for the median, 0.5; and all percentiles in between in a similar fashion. A simple formulation for the 25th percentile could be as follows. Here I'm swapping in the range names for AS:AS (Cluster1) and B:B (Data1):

$$= \text{PercentileIf} (\text{Cluster1}, 1, \text{Data1}, 0.25)$$

### 5.2.1.2 Working with Simple Bar Chart Options

OK, enough talk about prepping the data. Let's build some graphs, as promised. Our simplest bar chart begins with two steps: selecting the data of interest, say AY8:AY10 for those averages, going to the Insert tab, selecting the Column/Bar Chart option, and selecting the first icon on the top left (see Figure 5.4).

Just a note in passing: You'll see many graphic options here and elsewhere among the chart tools. Most you will never and should never use. You might,

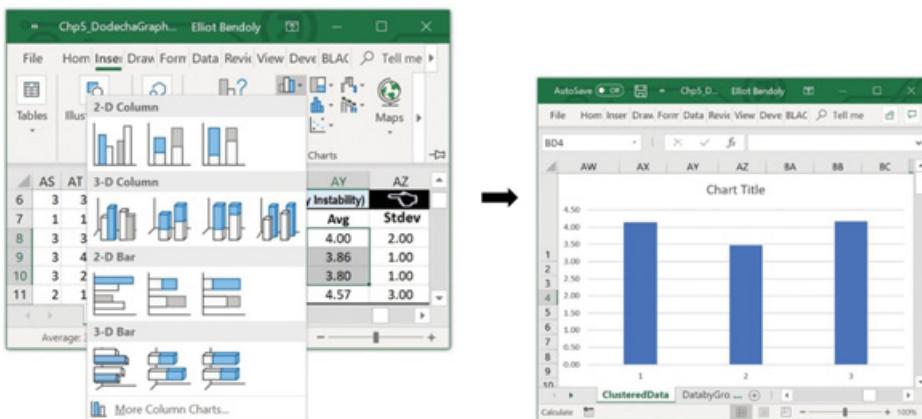


Figure 5.4 Accessing standard graphics available through Excel

for example, be tempted to use one of the 3-D options presented in Figure 5.4. They might look pretty cool – but be wary. The moment you introduce visual elements that do not actually visualize data (the depth of those 3-D bars), you chip away at the actual message you are trying to convey. If you think a third dimension that doesn't actually represent data is the only way to get people to read your report, throw them a bone on a piece of info that doesn't actually matter – maybe make an analogy between your fancy 3-D bar chart and the city skyline – but then quickly get back to the serious business of clean presentation.

We now go back to our current graph. Having made your selection, you are now in the position to make modifications to the graphic. Now, if you've selected a set of data, Excel will often guess correctly about how you want to use it (e.g., different rows representing different groups). However, it's also not uncommon for Excel to pick the wrong way to transform selected data into a plot. When you select multiple rows and columns of data, sometimes separate observations are confused with series delineations. Or, sometimes headers get included as data. Fortunately, we can correct for any initial stumble and further improve the presentation through the Chart Tools ribbon, which becomes available when charts are created or selected. From here simple corrections like “Switch Row/Column” can occur. If more complicated data usage fixes are needed, clicking on Select Data from the Chart Tools’ Design tab displays the screen in which both the series of values and the series of labels can be manipulated. If we examine the chart generated in our case, we should see something like that presented in Figure 5.5.

Since we have only a few data cells and we've created a fairly simple chart, we shouldn't have any data selection fixes to make. However, our main responsibility at this point is to make sure the presentation meets minimal

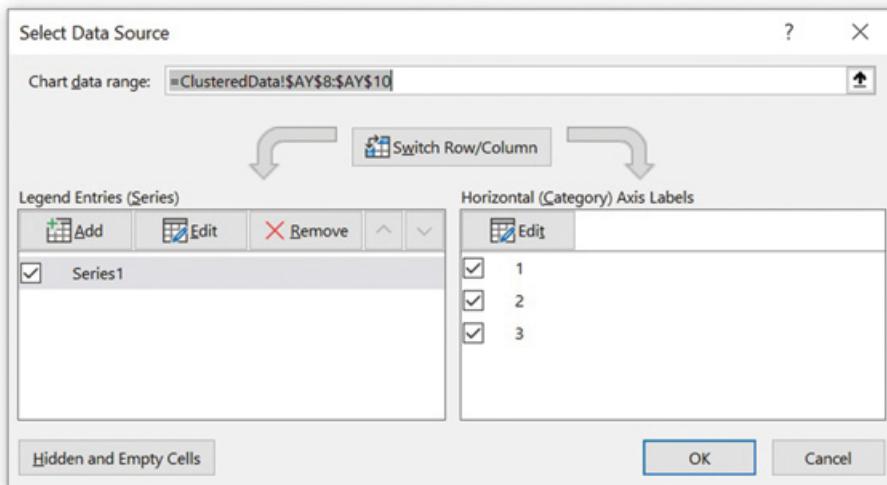


Figure 5.5 Chart data source editing interface

informational requirements. That is, fundamentally critical components that you will generally never want to leave out should now be included. This includes specifying clear axis labels and ensuring representative ranges and units for numerical axes. Chart titles, legends, and labels for specific data points may also prove useful, but getting the axes correct is something that applies to every graph we create. Start by clicking on Edit for the Horizontal (Category) axes, an option shown in Figure 5.5. Then simply select the range in which the names (Group 1 through Group 3) are located (AW8:AW10 in our example).

The Add Chart elements of the Design tab provide a quick means by which to add other elements, such as the Y-axis title. Once created, an element can be further modified by typing directly into it, or, with the new title selected, going to the formula bar and creating a reference to text in an existing cell (e.g., =AX6). And ultimately, as with controls (Chapter 4), right-clicking on any element in our graph similarly provides detailed attribute modification access (e.g., modifying the Y-axes to a range of 1 to 7). Eventually we'll end up with something like the example in the workbook, as shown in Figure 5.6 (where the third set of clustering results is the basis for the three groups and a specific attribute has been selected for examination).

Once again, this is a simple chart. But is it telling the story we need, or is it leading us to guess about too many things? Is it misleading us? You might think that all of these groups are basically the same in terms of the attribute shown in Figure 5.6 (all around 4). It's hard to say definitively, unfortunately, because we don't know how representative these averages are in this plot (i.e., we are not given a depiction of the risk profiles, the distributions,

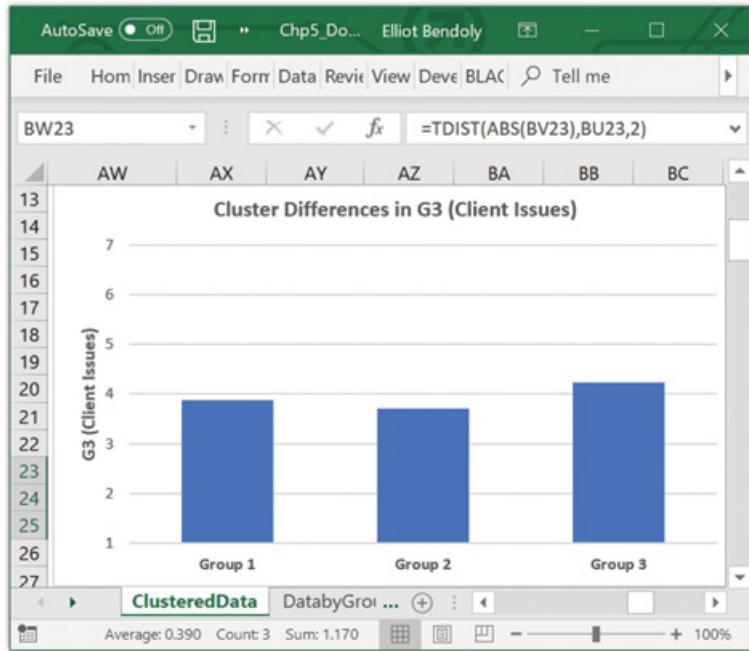


Figure 5.6 Simple Bar Chart presentation after chart element editing

around these means). Compare this to the box-whisker plot presented to the right of this simple bar chart, in the same workbook, shown in Figure 5.7.

The dotted lines in this plot are equivalent to the position of the top of the bars in the simpler bar chart. But here we also see additional summaries depicting the broader distribution of values for this attribute across each group. Group 1 is highly variable in this attribute with the middle 50 percent of all observations limited only by the measurement scale (they are all over this scale). Groups 2 and 3 are more contained and notably it looks like Group 2 and Group 3 might even be statistically significantly distinct from one another. A t-test comparing the data in Groups 2 and 3, assuming somewhat different levels of variation overall, provides a 2-tailed p-level between 5 and 10 percent, in line with the graphical depiction. While the additional test reinforces the observations in this plot, the richness of this latter depiction (Figure 5.7) is a big departure from the limited insight that might be gained through the simpler bar chart (Figure 5.6).

Yet, both charts are constructed and made dynamic using Excel's bar charts options. The box-whisker, in this case, was built using the Stacked Column chart option and selecting the range BG8:BK10 as a data source. Stacked Column charts build incrementally, creating a first bar from a minimum value of 0 and stacking additional data increments above. BG8:BK10 include just the increments between points in the distribution

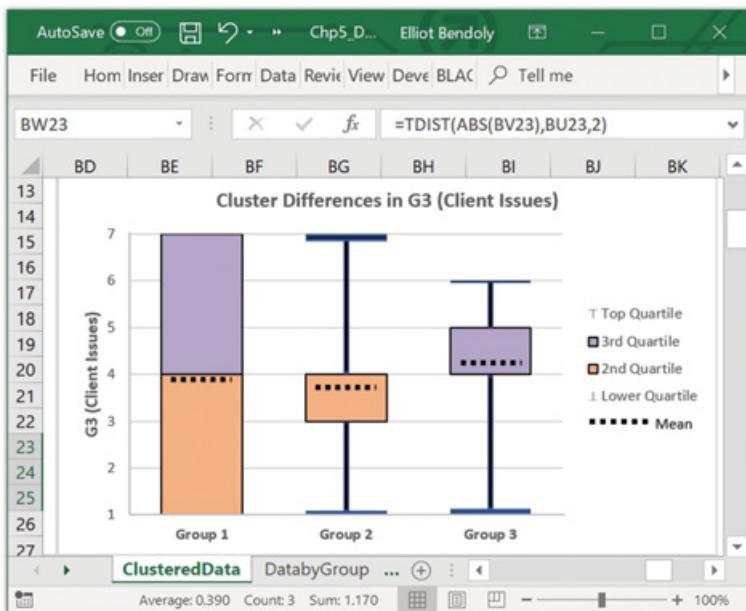


Figure 5.7 Stacked bar chart converted to box-whisker using conditional summaries

(0 to Min, Min to 25th percentile, 25th percentile to Median, etc.). Right-clicking on the base bar (0 to Min), we can make the fill of that bar transparent so it doesn't appear in the graph (it's still there). Note that Gantt charts, which are popular in project management, can also be created to depict nonactive time frames through transparency.

We can modify the color and border of the interior quartile in a similar way, retaining the view of these bar sections. The whiskers (the bottom and top quartiles) are a slightly different matter but relatively simple nevertheless. They can be created by first drawing a "T" shape (e.g., using the drawing tools), copying it (Ctrl-c), and then selecting the upper bar section and pasting (Ctrl-v). The image in buffer then appears in place of the default rectangle. Flipping the "T" and doing the same for the bottom quartile provides the whisker effect. A Legend element added to this chart will reflect these images as well. Lastly, we can superimpose another plot (e.g., a scatter, which we'll talk about next) with perhaps another pasted image (dashed line) to provide the position of the average in a traditional box-whisker format. Placing a copy of that image in line with the legend completes the depiction.

Now, getting to this point involved a lot of separate operations, though it was worth it from an interpretation standpoint. Any one of these operations might seem interesting by itself but the power really comes in the combination of all of these approaches. The combination capitalizes on the

capabilities available through the platform but avoids the limitations of certain canned approaches that would be more difficult to work around. The box-whisker approach available in Excel, for example, assumes data is already separated into groups. Pivot Tables allow some similar chart development options but not all (as we will see). The greatest strength you can develop in this environment isn't knowing all of the tools but being familiar with how they can be used in combination and how those combinations can be realized.

#### *5.2.1.3 Cell-Embedded Sparklines*

Now, while there are clearly more sophisticated ways of leveraging bar chart structures, there are also simpler graphics of the bar chart variety, which may nevertheless prove useful when data and messaging is simple, or when a collection of simple graphics is used systematically to present a comprehensive view. We aren't always graphing averages with a concern for associated risk. Sometimes all we have are a handful of observations – that doesn't mean they aren't worth looking at.

With contemporary versions of Excel we see a number of convenient, simple, and accordingly limited visualization options. Certainly we've already seen how conditional formatting can be applied to cells to create visuals similar to bar charts (Chapter 4). Another option that provides a similar functionality is the Sparkline features. These simple graphical elements allow any single series of data, divided into discrete bins, to be depicted visually directly within cells of the spreadsheet. Although the vast majority of standard chart formatting options are not available with these elements, some formatting is possible and the results are often visually straightforward.

In contemporary versions of Excel, Sparklines can be found under the Insert tab. A userform pop-up will ask for the location of the series and the destination for the resulting Sparkline graphic. As shown in Figure 5.8, Sparklines for multiple series can be generated simultaneously. As with embedded charts in Excel, any changes to the source data will automatically be reflected in the Sparklines graphics. In the Chp5\_DodechaGraphs workbook, the sheet Spark&Pivot contains the results of specifications described in Figure 5.8.

#### *5.2.1.4 Pivot Charts: Conveniences and Limitations*

Our experience with Pivot Tables in Chapter 3 has also exposed us to some of the conveniences in data organization that they offer, along with some caveats to their use. Pivot Tables are designed for ease of specific actions, with some notable loss of customization and often a risk of misinterpretation. Pivot Charts, also found on the Insert tab and designed to interact with

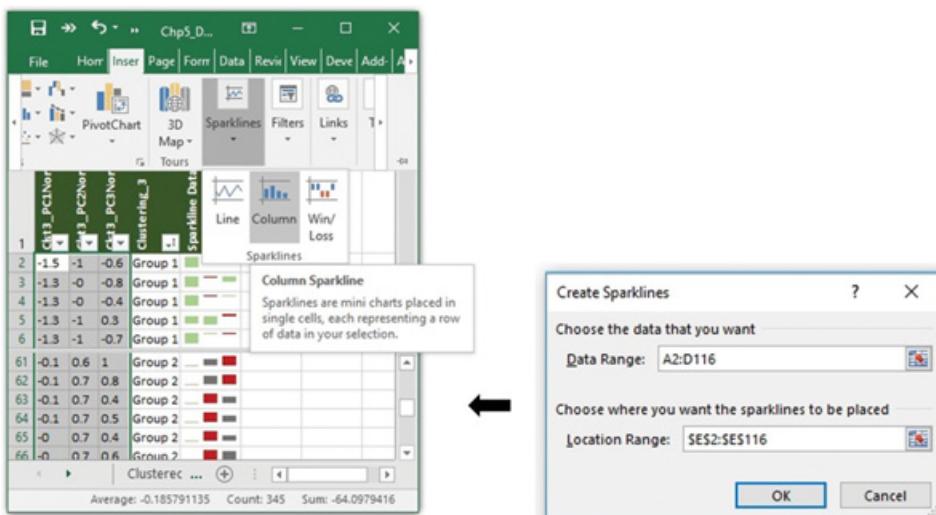


Figure 5.8 Sparkline access and specification for cell-embedded graphics

existing Pivot Tables, have similar pros and cons. First off, the choice of graphical options available when working directly from Pivot Table summaries is greatly limited. We can build bar charts directly but can't build Excel's packaged box-whisker plots or many others. There are workarounds for graphics based on Pivot Table summaries but not within the Pivot Chart realm. We'll revisit this limitation shortly in Section 5.2.2.

Second, although the appearance of Pivot Charts can be easily manipulated through filtering choices in the Pivot Table or through associated Slicers, as well as through drop-downs embedded within Pivot Charts, getting things to look the way you want can be more of a struggle. Chart fields used in the visualization will appear by default at the top of resulting graphics and are a challenge to move around or mask. This in turn makes the alignment of displayed content relative to these field headers somewhat cumbersome.

Figure 5.9 provides an illustration of how Pivot Table summaries might be combined with both Slices and a simple Pivot Chart of the clustered-column bar chart variety. It's incomplete in terms of its representation of variation (which, again, other graphical approaches could accommodate, if only they could be built through Pivots directly). And it leaves something to be desired by virtue of remaining field content (essentially chart junk in this case). But the flexibility gained through easy cutting and slicing of the data could be all you need in an early stage of analysis. Certainly there's little justification to stopping here with respect to visual depiction, but you don't have to consciously avoid this capability either.

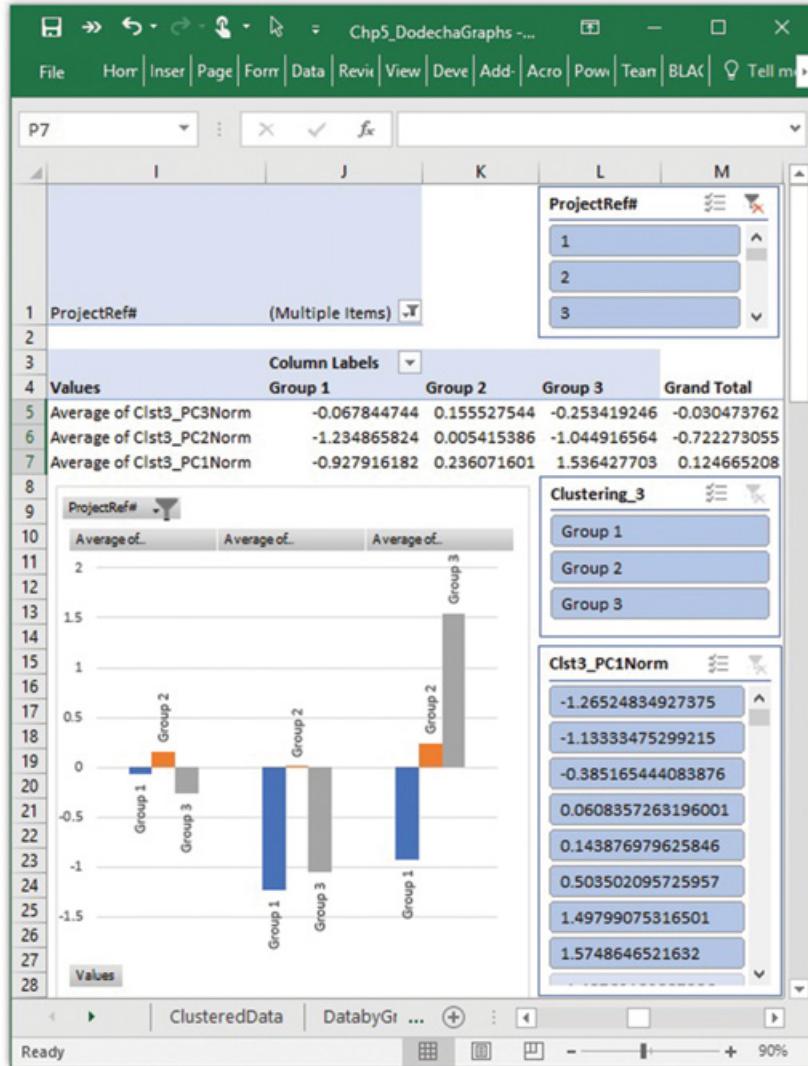


Figure 5.9 A Pivot Chart integrated with Pivot Table summaries and Slicers

### 5.2.2 Scatter Plots

In the presence of two ostensibly related continuous variables, scatter plots are often the default go-to graphical options. They can be used to describe general closeness or proximity among comparable observations as well as to tee up quick examinations of implied association. Continuing with our example of clustered groups, let's say that we strongly believed that the group distinctions are not simply a matter of examining a single dimension and hence can't be effectively captured by looking at a single bar chart. After all, we did allow for cluster analysis to use multiple dimensions to distinguish

these groups. Even side-by-side box-whisker diagrams could fail to capture a key aspect of this multidimensional space, specifically the tendency for one partially distinguishing dimension to be related to the value of another. How these measures might vary with one another, captured by summary statistics such as covariance, correlation, or certain fit statistics, can be much more directly viewed through the use of scatter plots.

### 5.2.2.1 Working with Basic Scatters

On the second sheet of the Chp5\_DodechaGraphs workbook, I've created a quick mechanism for pulling in data for two dimensions of Groups 1–3, depending on the choice of clustering. Data Validation is used in the headers of the first few columns on this sheet in the same way that it was earlier. The first three columns on the sheet use a kind of floating approach to MATCH, where the range in which entries that match the group name criteria is shifted down by location of each subsequent match. Eventually matches are not found, so we receive an #N/A result (which I've conditionally converted to white font). Those match values are simply row locations, so, with the column locations derived from the attribute selections in the adjacent headers, I can quickly gain access to the data specific to each group and for each dimension selected.

A scatter plot of the data in columns D:I requires a couple of steps, since Excel might not immediately recognize which columns represent X-Y pairs. In our case we are pairing column D with G, E with H, and F with I, representing two dimensions of data from each of three groups. Let's just start with the first pairing (D and G). Selecting the range D2:D68 and holding down the Ctrl key, you should be able to also select the range G2:G68 without selecting the columns E:F in between. With only these two paired data columns selected, we can go once again to the Insert tab and select the simplest of Scatter Chart options from the Charts section of that ribbon. As with the bar chart example, you should generate a precursor to what we ultimately want to end up with, something along the lines of what we see in Figure 5.10.

As with the bar chart editing, with this new chart selected, we have access to the Chart Tools tabs and ribbon options. Clicking on Select Data will give us a presentation similar to that in Figure 5.5, where we again can add data series or modify how they are used. In this case we will be adding two additional series and specifying X and Y series values by selecting the associated ranges for Groups 2 and 3 (E2:E68 and H2:H68, and then F2:F68 and I2:I68). Each new series will receive a default marker coloring to distinguish them from each other. Deleting the grid lines, editing the axis ranges, adding axis titles, and editing the overall chart title allows us to ultimately develop a clearer picture of how all three groups differ in

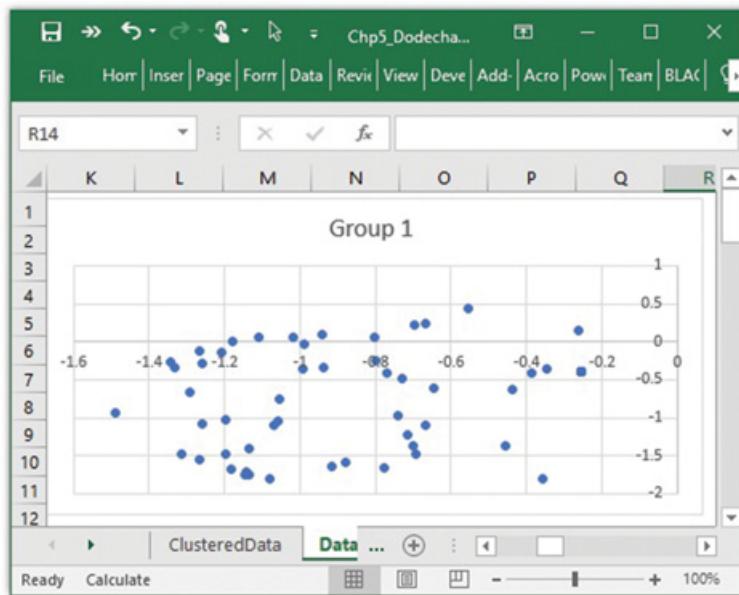


Figure 5.10 Default scatter generation prior to editing/formatting

a two-dimensional space. In Figure 5.11 we see the three groups as they differ along two of the PCA-derived factors from the third cluster analysis.

In this figure it is clear that some of these groups are more different than others along particular dimensions and that the variability within these dimensions differs by group. However, we might also notice some possible distinctions in the way in which data seems to trend or covary within these groups. We might begin to get a sense of these differences by right-clicking and adding Trendlines to each series. These are essentially univariate regressions (default trends are linear model estimations). If you look at the R-square of these series-specific trendlines, accessed by right-clicking to edit once you've created each, you'll see that in some cases the X is a stronger predictor of the Y variable than in other cases. These are the same R-square values, incidentally, that you might get from LINEST or the Data Analysis tools, or JMP, R, and so on. Also, while the R-square estimates won't change depending on whether you regress Y as a function of X, or X as a function of Y in these cases, the slope of these trendlines will change in ways that are more complex. That is, the estimated impact of X on Y is not simply the reciprocal of the estimated impact of Y on X, as shown in the additional regressions starting in row 37 and below in that sheet (Figure 5.12).

One might ask, what is the actual relationship between X and Y? In Group 2 of this example there seems to be something going on. And we can in fact

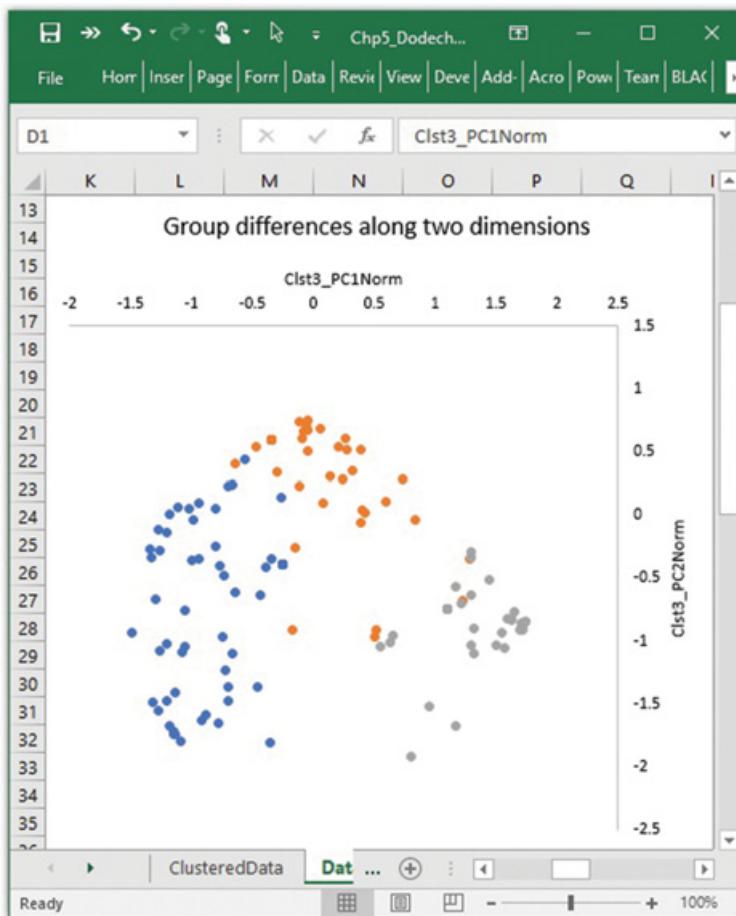


Figure 5.11 Edited scatter plot depicting data points by group

make more general statements about the relationship between these dimensions without getting into assumptions about causality (it may be that a third unseen factor is driving both). As long as we know that there is a general tendency for both factors to increase or decrease together, we've gained some insight into the composition of this group. In a later section we'll consider how we might summarize these relationships and their covariance visually to further enhance group comparisons.

#### 5.2.2.2 Higher-Dimensional Options in Scatter Plots

Let's also now look to another set of data, provided in the workbook Chp5\_QuarterlyGraphs. We've seen this data before, both in the discussion of Pivot Tables as well as in the discussion of regression. I've kept some of these resulting sheets for our current use here as well but I've added some additional calculations to the RawData sheet and added another Pivot Table

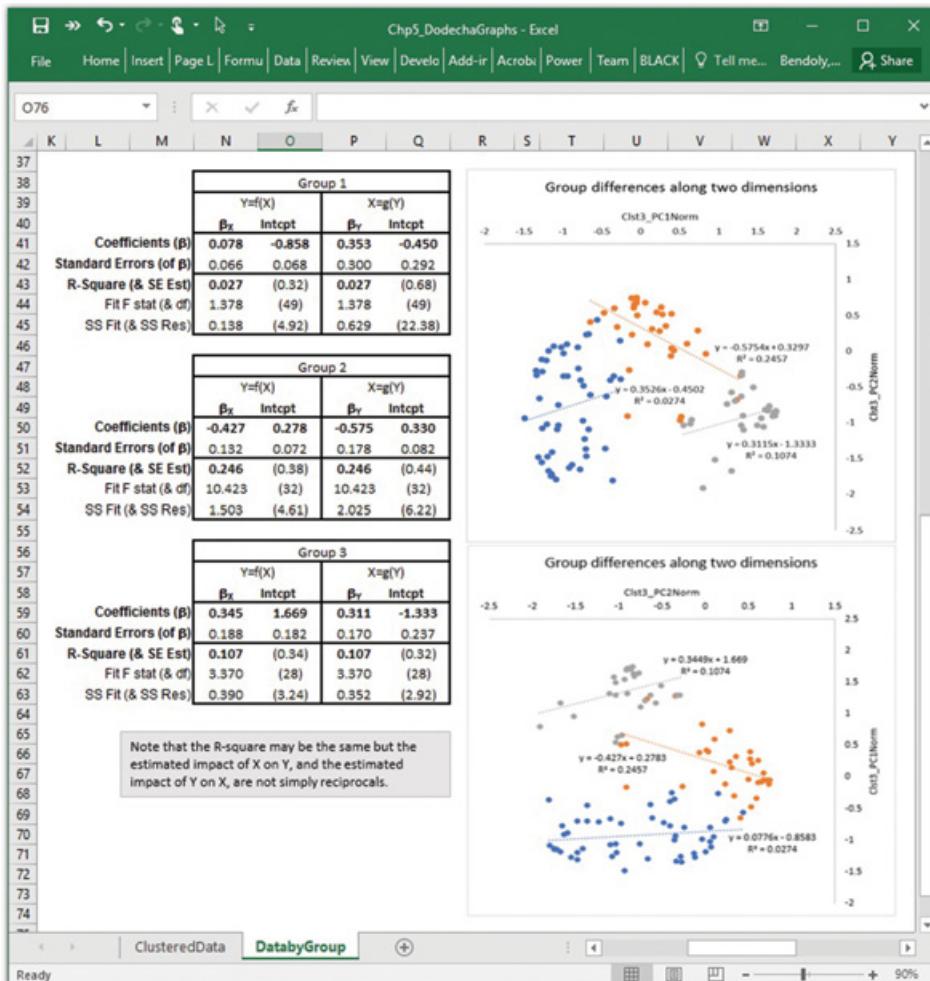


Figure 5.12 Consistency in association, differences in predicted impact

on a sheet named BubbleChart. On this sheet we are looking at not two but three dimensions and beginning to think about how variation across these three might be related. The bubble chart graphing option is available under the same Chart section as the more traditional scatters. What the bubble chart attests to provide is the means by which to look at the third dimension, beyond the X-Y Cartesian plane, captured by the size of the data marker.

There are a few things to note here. First, I have a Pivot Table with data but I'm not using it directly (I'm not creating a Pivot Chart). Why? Well, give it a try. Select a range of data in the first three columns of this sheet, where the Pivot Table exists, and try to insert a bubble chart or a scatter. You'll likely get the following error or something like it: "Some chart types cannot be combined with other chart types. Select a different chart type." Pivot

Tables allow for a great deal of packaging and flexibility when it comes to some very particular and common tasks (e.g., building a bar chart). Sadly, the current design is not friendly to things like scatters. But we're not going to let that stop us. The simple workaround is to create a mirror, or modified mirror, of the Pivot Table data in some nearby cells (as I've done in columns G:J). In a cell called "Choice" (L9) I've added one of these selection mechanisms to restrict this mirroring to only certain values from the Pivot Table (all values, only those with positive EPS values, or only those with negative ones).

Second, I've constructed a bubble chart by selecting the full range from I6:K127, clicking on the Bubble Chart option, and making some choices regarding the axes and the chart title. I've thrown in a best-fit line and the regression statistics as well. As for the bubbles, right-clicking gave me the option to scale the markers so that the area of each represents the EPS value magnitude, and I've chosen to show any negative values (which appear with transparent white fills as in Figure 5.13). Modifying the choice of data to examine demonstrates the potentially different kind of relationship that exists between observed changes in our variables, coincident with growth or loss in EPS.

In this depiction I've also attempted to tackle one of the biggest problems with bubble charts and similar efforts to capture scale with marker size, namely, people have a terrible time interpreting data differences that are anything other than linear. This is one of the reasons that Pie Charts have

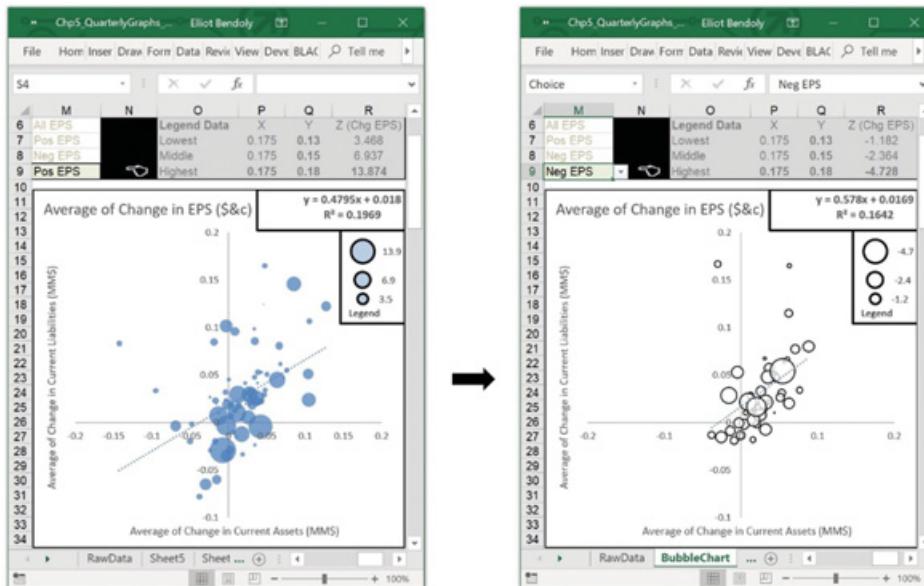


Figure 5.13 Bubble chart comparisons based on data subset selection

such a lousy reputation (and why I'm not going to spend time on them here). This isn't simply an opinion – it's cognitive processing phenomena that have been studied across multiple disciplines. Still, we can make a little lemonade from this lemon by assisting interpretation through the use of a custom legend and a separate data set representative of several levels of EPS magnitude. It's not a perfect solution, but as we go up in dimension and try to nevertheless depict things in a flat plane, we are going to come up against trade-offs. Grouping by EPS and showing Group 1 versus Group 2 as separate colored series would represent a loss for interpretation as well.

As an additional side note, there are occasions, and they are fairly rare, when yet another approach to leveraging bubble chart capabilities can prove handy. This is less common in management than in engineering and the sciences, where the depiction of spatial depth is more often relevant. Specifically, you might notice that the points in all bubble charts will, by their very nature, appear "stacked" in a plot (unless they are transparent). The points at one end of the column of data may appear to partially or completely cover those at the other end (which is also not always ideal). Sometimes large bubbles effectively hide smaller ones as a result. Sorting the Z variable from largest to smallest at least increases the chances of these points being visible, with partial transparency either helping or hindering the depiction.

But in the sciences and engineering, as well as in management, as I've learned from personal experience in both, perspective can be everything. If we had spatial data points in three physical dimensions, atoms in a lattice, sensors in a tornado, or imperfections in a design, we could make the most of something like a bubble chart to emphasize perspective. In this case things closer to our view should appear larger and we'd accept the associated risk of data being partially obstructed, provided we had the ability to rotate our perspective and adjust our view. We can apply a similar principle to reexamine the data in Figure 5.11. Instead of looking at three dimensions, the second of which was characterized by marker size, with an appropriate legend, we could consider developing a means of rotating our perspective, allowing that rotation to permit the view of yet an additional dimension. If we truly felt we needed to see four dimensions simultaneously, the Chp5\_QuarterlyRotational workbook provides a depiction of how we might go about it. The example, as shown in Figure 5.14, is not complete with regards to a robust design for a user experience but it demonstrates how far we might leverage perspective (and flexibility in that perspective) across multiple dimensions.

As a final thought on higher-dimensional depictions in the Scatter class of plots, it's worth considering a scenario that might be somewhat simpler to depict – still three dimensions and a fourth outcome. However, in this special

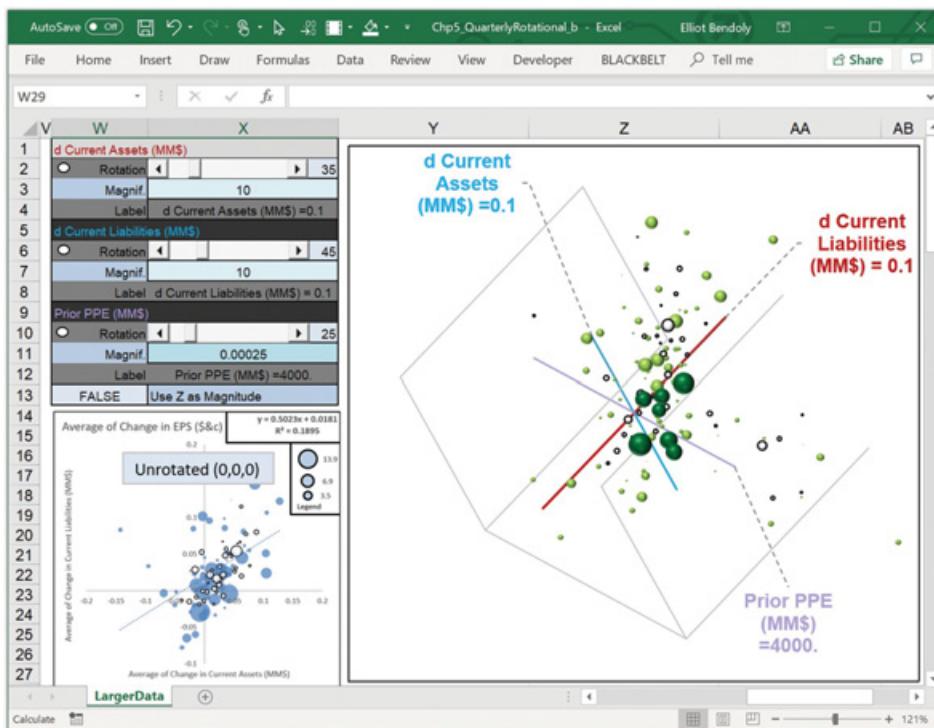


Figure 5.14 3D Rotational bubble chart with a fourth dimension by magnitude and color

case, the first three dimensions are fundamentally related such that the sum of the three must equal some fixed quantity. Maybe these are three dimensions of expenditure limited such that their sum must equal the budget (no less, no more). Maybe these are components contributing to the weight of a new product, all of which are percentile and must sum to some limiting percent. In any case, we would say that these three potentially impactful decisions (utility variables) face “connected constraints.” Each is limited, but the more you do of one, the less you can do of the others. Knowing two allows you to know the third. In a 3D space they define a surface of possible decision sets. If it’s a simple sum that they must equal, then we are talking about a triangular area in the 3D space, anchored by each of the maximal values that each decision can take on in isolation.

And if that’s the case, then all we have to do is focus our perspective on points within that triangular surface, with each data point on that surface possessing potentially a fourth attribute that can be large, small, positive, or negative. Such a 4D representation is made available through the Ternary Plot tool in the Blackbelt Ribbon add-in. Selecting an appropriately scaled set of data, such as that presented on the TernaryPlot sheet of the

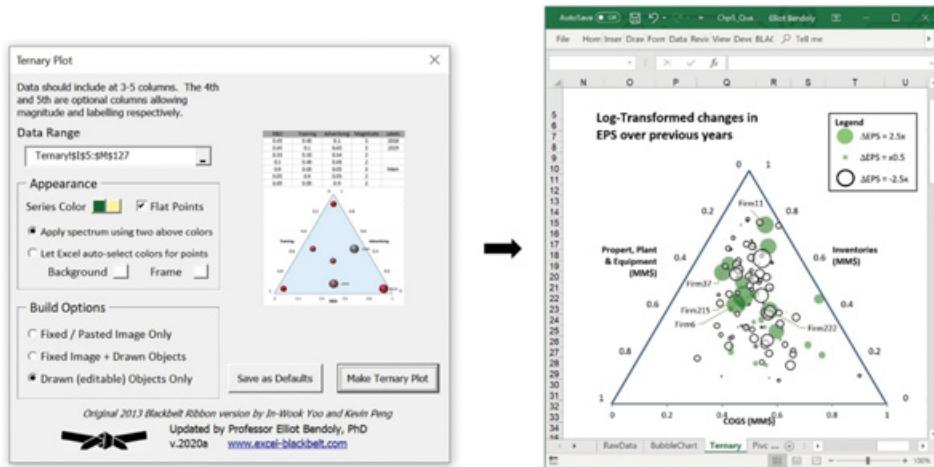


Figure 5.15 Bubble charting through the Ternary Plot tool on the Blackbelt Ribbon

Chp5\_QuarterlyGraphs workbook, allows the generation of a variety of plot options, including the use of color scaling to emphasize distinctions in those fourth values. The further incorporation of an informative legend, as included in the rendering of Figure 5.15, would again be critical for interpretation.

#### 5.2.2.3 Connections in Scatter Plots

Another data feature you might consider depicting within scatter plots, apart from the association between groups of data, is the “connectedness” across individual data points. In some cases data points are connected to others in the representation of networks of systems, people, facilities – anything that might constitute a holistic system. In some cases data has the additional feature of being meaningfully ordered across time. You might not want to represent time in an X or Y axis if you have other uses for such an axis. As an alternative, visually connecting these points in sequence could potentially demonstrate, in a static sense (at least), the history of how these data points evolved.

This doesn’t always yield the results you might want and may end up being far inferior to a nonconnected plot (since visual connections can also be greatly misleading in their implications of interpolated data between any pair of points). However, if you have reason to believe it might be informative, it’s worth thinking about your approach.

In the Chp5\_QuarterlyGraphs workbook, we have interval data by year and by quarter. Let’s say that we had relatively scaled X-Y data within each of three firms, sorted in time over a 16-quarter timeframe. We might think that since this data is sorted by time, we could connect the dots and have

a useful depiction of the paths of change over time. In Figure 5.16 I've added each firm's data to a line-connected scatter accordingly.

What can we gather from this plot? That depends on what you are used to viewing and what you are comfortable with attempting to interpret. A lot of information is being shown and as a whole this kind of presentation can make the extraction of meaning a bit challenging – perhaps not encumbering for all audiences but certainly for some. A slightly modified mix of information might prove helpful. As shown in Figure 5.17, we leverage some simple ActiveX controls once again to provide options to our user in examining various discrete time windows. This kind of flexibility and a small amount of guidance through the use of labels goes a long way toward helping others

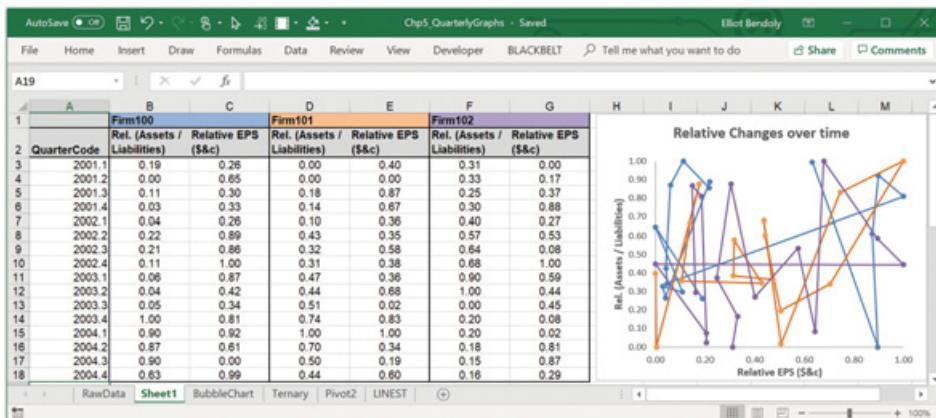


Figure 5.16 Unfiltered line-connected scatter plot by series

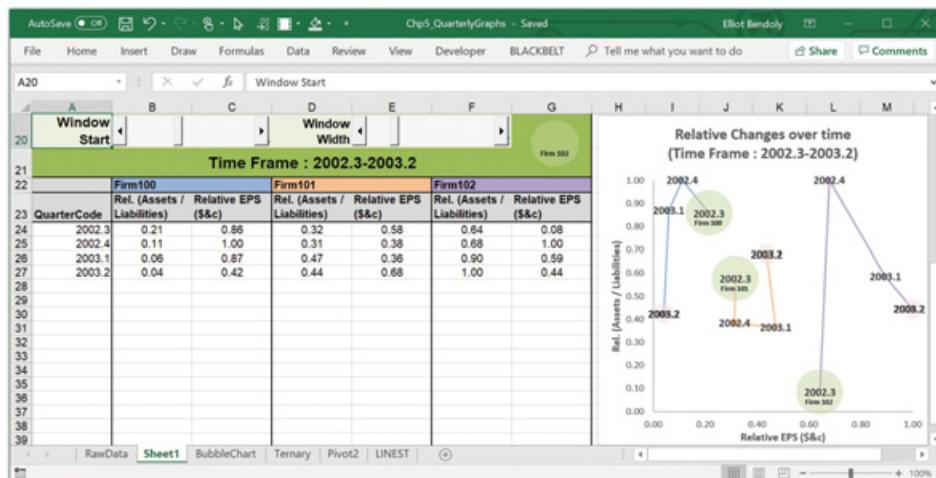


Figure 5.17 Time-window filtered and directionally labeled scatter plot by series

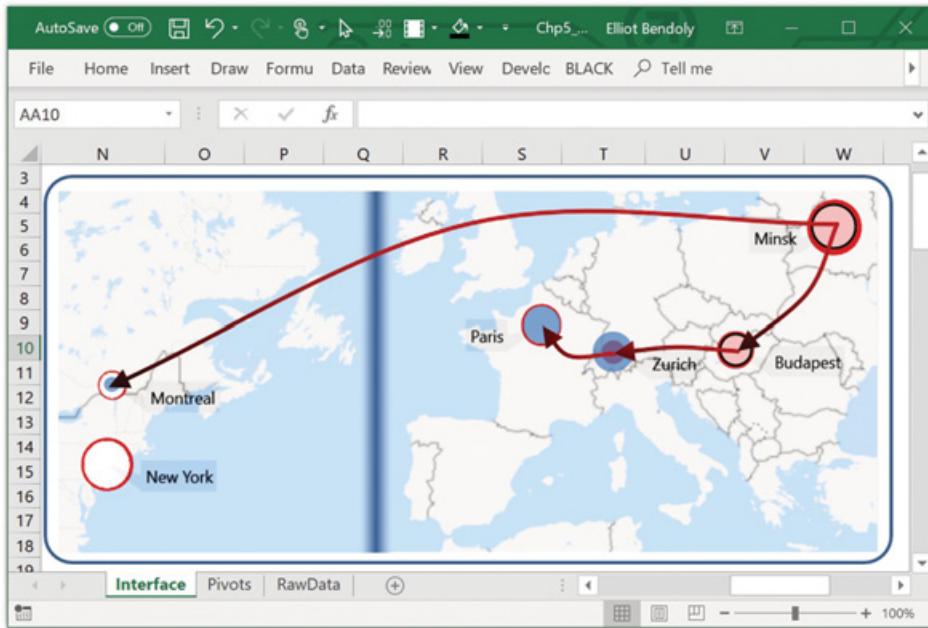


Figure 5.18 Activity filtered and directionally labeled spatial scatter plot overlay

interpret the kinds of trends that might be unique or shared among data series (firms in this case) over time.

#### 5.2.2.4 Depicting Flow in Scatter Plots

Used wisely, line-connected scatter plots can be used to portray paths across not only time but also space. An example is shown on Figure 5.18. In this particular depiction (from the Chp5\_Madrenko file that we'll look at again in the next chapter), each arc between the X-Y pairs is defined as a separate set in the scatter in order to provide an easy mechanism to turn selected arcs on or off. An intermediate point is used to enforce curvature and reduce the risk of straight path overlap and obfuscation (we'll talk about the pros and cons of this in a second).

To see how these paired coordinate paths are put together in the first place and how they allow connecting arcs to be turned on and off, we can look briefly into the associated cells in row 47 and below in this sheet (see Figure 5.19). Looking at the cell that contains the X coordinate for Zurich, we see the product of some cell (\$H49) and the result of a VLOOKUP. The VLOOKUP is pulling the actual designated X coordinate that is representative of Zurich's relative placement on the graph. The cell \$H49 contains a value that is either +1 or -1, which itself is dependent upon the result of a decision higher in the sheet (in this example, whether the route is being

	C	D	E	F	G	H	I
47	Paired Arcs (negative quadrant values don't appear on graph)						
48		X	Y				
49	Zurich	0)	-3.82	0	-1	<-Selected	
50	to	-3.47	-3.995	0	0.5	<- X-Y Shift	
51	Montreal	-2.23	-3.17	0	-1	<- (copy)	
52	Zurich	-4.71	-3.82	0	-1		
53	to	-3.4575	-3.3975	0	0.7		
54	New York	-2.205	-1.575	0	-1		
55	Zurich	-4.71	-3.82	0	-1		
56	to	-5.015	-4.34	0	0.5		
57	Budapest	-5.32	-3.86	0	-1		
58	Zurich	4.71	3.82	6	1		
59	to	4.485	3.785	0.05	0.3		
60	Paris	4.36	4.35	6	1		
61	Zurich	-4.71	-3.82	0	-1		
62	to	-5.2575	-5.17	0	0.1		
63	Minsk	-5.805	-6.32	0	-1		

Figure 5.19 Structure of source-destination arcs for filterable directional scatter

used for shipping). When the value is +1, the coordinates are positive and the arc appears in the positive quadrant of the scatter plot. When the value is -1, it appears in the negative quadrant but, because the scatter plot's axes are fixed and nothing below (0,0) is displayed, the negative arcs never show up. A bit of a vanishing (and appearing) act is therefore being played, without the loss of information in the graph.

The other visual aesthetics designed for messaging include the use of colors and arrow heads at the end of each arc. Movement is from red to black, flat to arrow head, capitalizing on fairly established accounting and iconic symbolism. Bubble charts are used to represent coverage and initial states, so that as decisions are made we can see where deficits are being covered and where surpluses may still exist. There is a slight curvature in the arcs, made possible by an intermediate point that is somewhat shifted off the line between sources and destinations, to ensure that backwards routes do not entirely overlap forward ones (to help with error checking). Again, we'll be returning to this particular example in Chapter 6 when we start looking for optimal solutions to our decision-making scenarios.

### 5.3 Complex Area Visualizations

While common charts, such as those just discussed in the bar chart and scatter chart families of graphics, focus on the depiction of data points with, yes, the potential for those points to be used in capturing distribution (coverage, variability, risk, etc.), other approaches to visualization focus more deliberately on areas of effect. It's an important distinction and a focus for which we would like to have visualization options. Individual data points can often be misleading and be taken out of context. Areas possess multidimensionality in forms not always easily described by even a handful of numeric summaries. Because of this, it is useful to familiarize ourselves with several area-focused visualization options readily available, any one of which might prove more useful to us.

#### 5.3.1 Visualizing Areas of Confidence and Uncertainty

One thing that individuals – analysts and nonanalysts alike – are biased toward is a simplified depiction of reality. This isn't our fault because most of what we are educated on either comes in the form of assumed facts or measures of central tendency (averages, medians, and so on). Although professionals are aware that averages are only one representation of a distribution of values, the vast majority of professional decisions are still based on considerations that exclude other characteristics such as variance. Even our graphical depictions of data tend to be light on depictions of variance (our discussion of commonplace bar charts).

While depictions such as the box-whisker plot, discussed earlier in the chapter, can give use a sense of a single variable's (univariate) distribution or risk profile, looking at two such plots for two variables might not tell the full story. The two variables and their distributions and levels of risk may in fact be intertwined (e.g., correlated). Certain approaches to scatter plotting (e.g., distinctions by group or some other additional dimension) may provide some hints into how multiple variable distributions/risks might be related. However, the task of supporting/corroborating statistical tests visually through the eyeballing of what that relationship and joint (bivariate) distribution appears to be and whether that joint distribution is distinct relative to other groups is a daunting one.

What we really could use is a way to summarize bivariate distributions in a manner similar to what the box-whisker approach does at the univariate level. The Confidence Ellipse tool in the Blackbelt Ribbon add-in was designed to do just that, making the work of more appropriate bivariate renderings of risk and their visual comparison a bit easier. This tool takes the means and standard deviations of a series of X and Y groups, along with

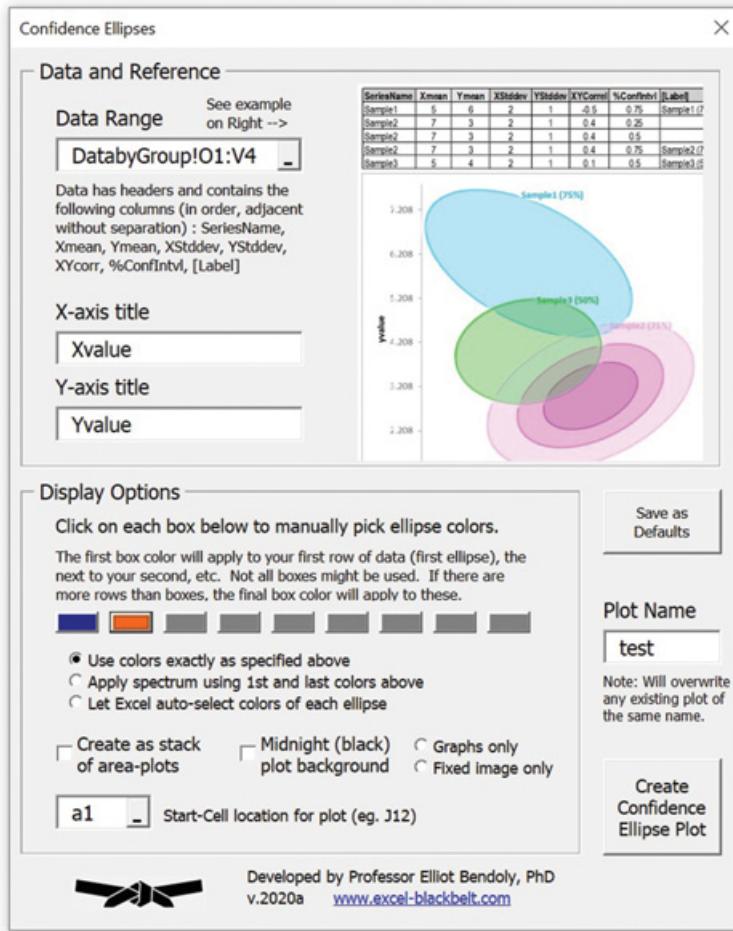


Figure 5.20 Interface of the Confidence Ellipse tool of the Blackbelt Ribbon

correlations of X-Y in these distributions, and plots confidence ellipses for each pair based on user specifications. Figure 5.20 shows an example of the tool's interface.

The features of this interface are not unlike those of the Ternary Plot tool. Color specification options and choice of image versus editable graphic outputs are made available. With specifications set, the tool generates confidence ellipses that are estimated to include a certain percentage of all data represented by each variance pairing, centered at each pair of means. The greater the percentage confidence level, the larger the area of the ellipse generated. Input data range selections can also include optional labels to these ellipses. As with all tools in the Blackbelt Ribbon, you can also save your preferences to your current workbook as default settings that can be readily accessed.

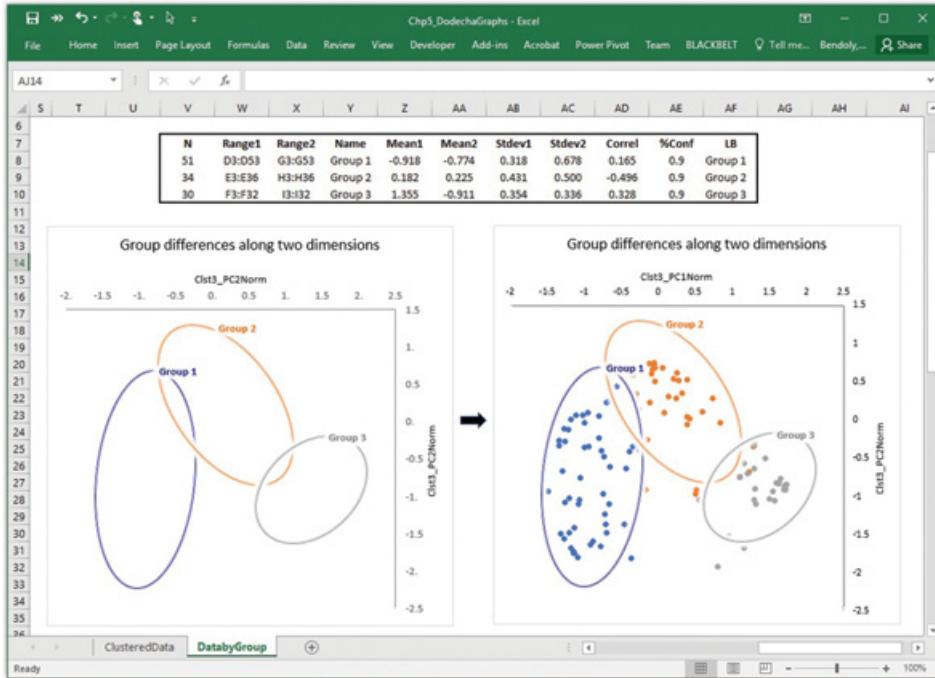


Figure 5.21 Resulting confidence ellipses alone and in overlay

Figure 5.21 provides an illustration of the tool's output applied to our clustering efforts with the Dodecha Solutions data set. On the left of this image I have the visual generated by the Confidence Ellipse tool, using summaries of the last clustering effort's three emerging groups. To the right I've simply made the fill of the graph transparent and superimposed these results on the earlier data (shown earlier in Figure 5.11). The ellipses by themselves describe distinctions between the groups in a much more concise and explicit manner but, if you want to retain full transparency to the data, that is certainly an option as well, as shown in Figure 5.21.

### 5.3.2 Surface Maps and Cartesian Heat Mapping

The ellipses created through the Confidence Ellipse tool are perimeters derived through a backend calculation. By specifying alternate confidence levels you could ostensibly generate any kind of contour/topographic mapping with smaller confidence ellipses within large ones – if you really need to see these differences explicitly, that is. If you feel you need more of a relief mapping of the entire distribution, not limited to any specific set of confidence levels, you do have some alternatives. The first is to generate a relatively high definition grid (table) of equally spaced X-Y pairings within

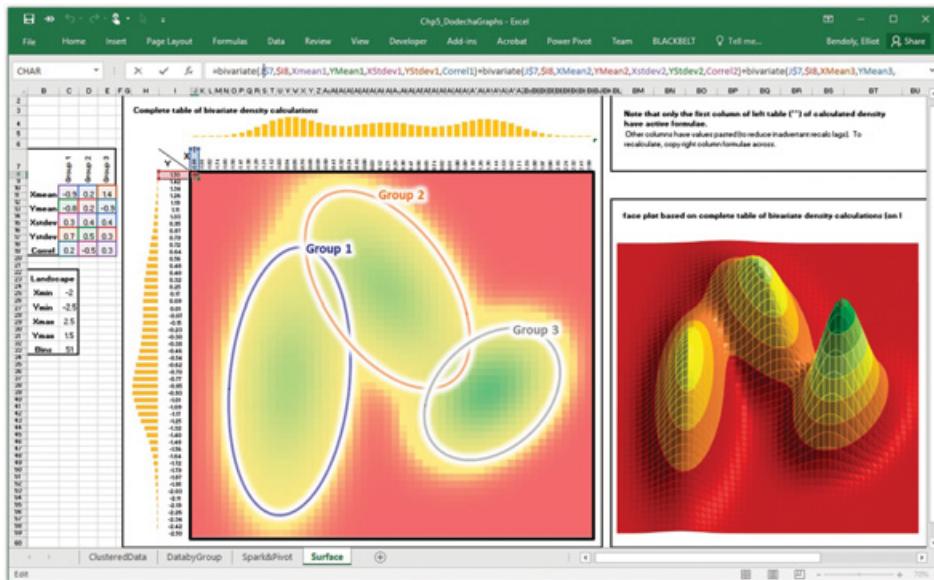


Figure 5.22 Conditional formatting of a calculated matrix and linked 3-D Surface plot

a range and their associated bivariate densities. If you wanted equally divided values of X from 0 to 1, you might divide this into 11 values incremented at 0.1, for example. With X values presented in the top row of your grid and Y on the left column, for example, the interior of the grid can be built through individual calculations or through the use of a data table on a single calculation. The Bivariate function itself, found in the Blackbelt Ribbon, can be used in these cells to generate these density calculations. The function requires a specific X-Y point for its first two parameters and, like the Confidence Ellipse tool, it requires values of means, standard deviations, and the correlation of the two dimensions.

The result is a bivariate density calculation for that pairing, which can be copied across all pairings in your grid. In the example provided on the Surface sheet of the Chp5\_DodechaGraphs workbook, I've conditionally formatted the interior (see Figure 5.22). Note that I'm actually combining all three group distributions simply to demonstrate proximity and overlap. I've also added conditionally (data bars) formatted averages by row (left of the Y values) and a sparkline capturing averages by column (above the X values) just to show how some of these simple visualization tactics can be useful. I've superimposed a graphic of the ellipses derived in the last section to emphasize consistency in what is being calculated. I've also pasted the vast majority of the cell calculations in this grid “as values” to cut down on possible lags in inadvertent recalculations. The formulae remain live in the first interior column, for copying.

To the right of these calculations we have another chart option available in Excel known as a 3-D Surface plot. Right-clicking on the plot grants access to the rotation and perspective of this view. It's a slightly cumbersome graphic, with relatively limited customization opportunities that can be hard to access. For example, to manually change the specific color bands, you have to access the chart's Legend and recolor through legend element selection – that is, unless you are OK with one of the preset color schemes, which are only a handful. You also can't easily modify the number of color bands or ensure that the number of bands you want will always appear as you change your perspective on the chart. Further, the axis labels are fairly difficult to edit manually (without code). These kinds of issues aside, it's worth knowing that there is at least an option along these lines. You likely won't find yourself using it too much to make your points, but it could serve you as you examine your data and modeling efforts.

### ***5.3.3 Defined Region and Polygonal Heat Mapping***

The previously discussed examples provide a way of describing information that is otherwise associated with two dimensions by incorporating the graphical depiction of a third dimension. In these examples specific X-Y data gave rise to the potential for showing the Cartesian positions and third variable summaries point by point but also allowed the development and depiction of summaries by area. In these cases the areas were also positioned in the X-Y plane and the areas themselves were mathematical functions of X and Y. Areas of effect, however, are not always simple (or even moderately complex) mathematical functions. When these areas are complicated in ways that are not captured by math alone, other graphical techniques must be sought out to assist with visualization and understanding.

Polygonal heat mapping is a general term that refers to the coloring or shading of typically nominal, often unique areas in a larger graphical depiction. The graphics are still typically two-dimensional, with the areas often combining to create a kind of jigsaw puzzle of reality (think of a map of the contiguous states in the United States). Each distinct color or shade of the areas drawn is associated with different levels of a trait shared by each area. When applied to geographic contexts, these maps are also commonly referred to as choropleths and are certainly the most common form of polygonal heat mapping that you will encounter. Other examples in practice involve the heat mapping of levels of quality across the components of a manufactured good, the levels of foot traffic intensity in a mall or hospital, or the levels of injury sustained to a human body or organ.

These latter examples are not the kinds of things that geographic information systems or the more recent mapping tools in Excel or other visualization

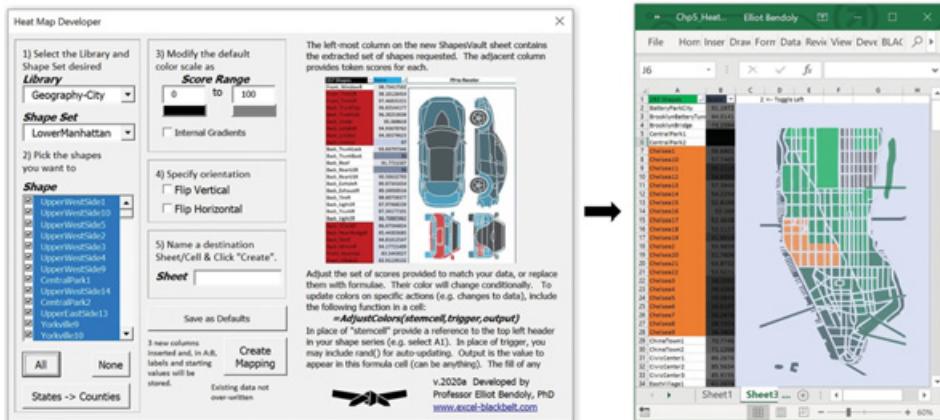


Figure 5.23 Illustration of the use of the Heat Mapper tool interface

applications are typically equipped to handle. Fortunately, we have a solution for this as well.

#### 5.3.3.1 Heat Mapper and Image Extract

The Heat Mapper tool in the Blackbelt Ribbon provides access to a limited but growing library of image sets to which polygonal heat mapping can be applied. These image sets fall into broader library categories of geography-national, geography-global, geography-city, anatomy, art, and blueprints. An example of the userform pop-up that appears after activating the Heat Mapper tool is given in the left panel of Figure 5.23, where a user has selected the shape set for Lower Manhattan and its associated images. A resulting heat map in use (i.e., following some user changes to the data and color indicators) is provided in Chp5\_HeatmapperExample. An image of this is provided in the right panel, along with the column A listing of all images and associated token values of “heat” (e.g., real estate costs, population density, etc.) provided in column B.

Regular recoloring of these generated heat map images involves the use of the Blackbelt Ribbon function `AdjustColors`. An example of the function in use is included in cell D1, as follows:

```
= AdjustColors(A1,C1,"<- Toggle Left")
```

The first parameter of that function is the upper-left cell in the two columns generated (the left column header). The second parameter toggles refreshes to the graphic. Replacing this reference with `Rand()`, or placing `Rand()` in C1, or simply making a manual change to the contents of C1, will automatically trigger color adjustment. The last parameter is simply a placeholder and describes what will appear in the cell where the function resides when called. The function will

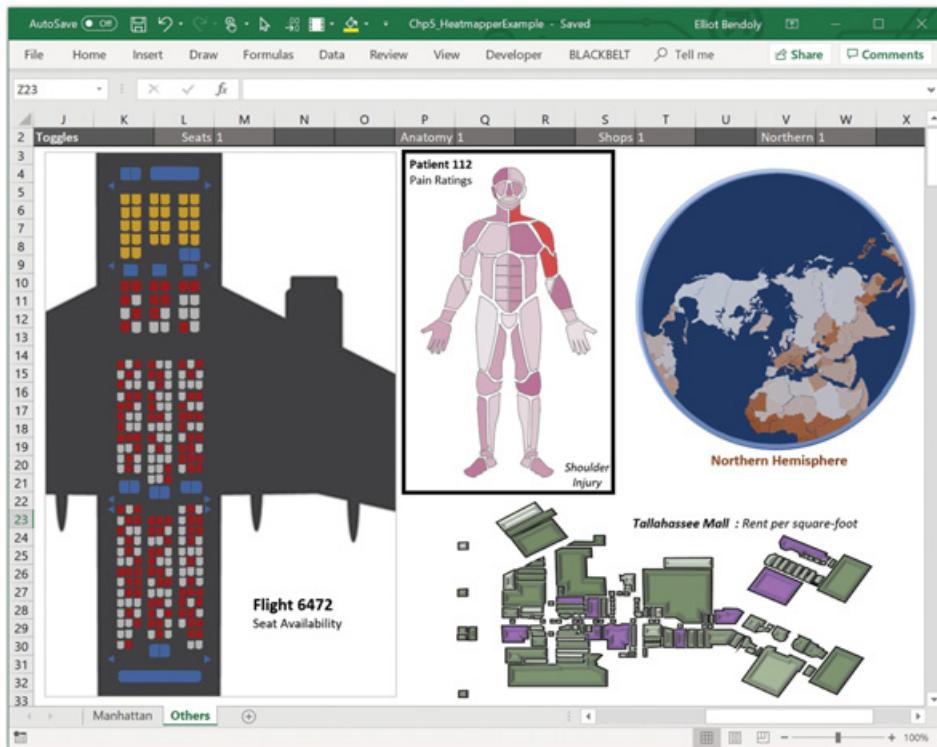


Figure 5.24 Examples from the Blackbelt Ribbon’s Heat Mapper tool libraries

take the fill color of the top row cells of the two columns of data as the new end points of the relative color spectrum. Any numerical changes in column B will be reflected according to that color scheme if the `AdjustColors` function is recalled. Individual entries in column A can be colored as well, for emphasis, with resulting adjustments to the images via `AdjustColors`.

Other examples of area-based depictions through the Heat Mapper tool are provided on the second sheet of this workbook (see Figure 5.24) as well as in the Chp5\_OhioCounties workbook (previewed earlier in Figure 5.1).

All images in the sets are drawn polygons (scalable vector graphics of a sort). If a user would like to leverage an image set on a regular basis that is not available in the shape libraries, any set of drawn images can be extracted using the Image Extract tool on the Blackbelt Ribbon. Feel free to forward any such extracts to me (my contact information is on [www.excel-blackbelt.com](http://www.excel-blackbelt.com)) and we can consider adding them to the libraries for all to use.

### 5.3.3.2 Filled Maps and 3D Maps

Developments in the MS Power BI domain have impacted not only data access (Power Query) and data manipulation (Power Pivot) but also the availability of

alternate visualization resources – that is, beyond the ability to create a few Pivot Charts based on Power Pivot summaries. Notably, for the current discussion of area-based renderings there are at least three Excel-affiliated options worth considering beyond the Heat Mapper add-in. These include the Filled Maps option within the Insert ribbon (for Excel 2019 or 365), 3D Maps (within the tours section of the Insert ribbon), and the MS Power BI Desktop application. Each of these is progressively more detached from the spreadsheet environment but nevertheless offers some interesting capabilities.

The Filled Maps option is the most natural extension of the traditional embedded graphics we've discussed up to this point. That is, the graphics generated are housed within the spreadsheet environment and therefore can be presented proximally to other embedded graphics and, often critically, adjacent to any spreadsheet model and decision-making structures (see Figure 5.25; also from Chp5\_OhioCounties). These maps can also update

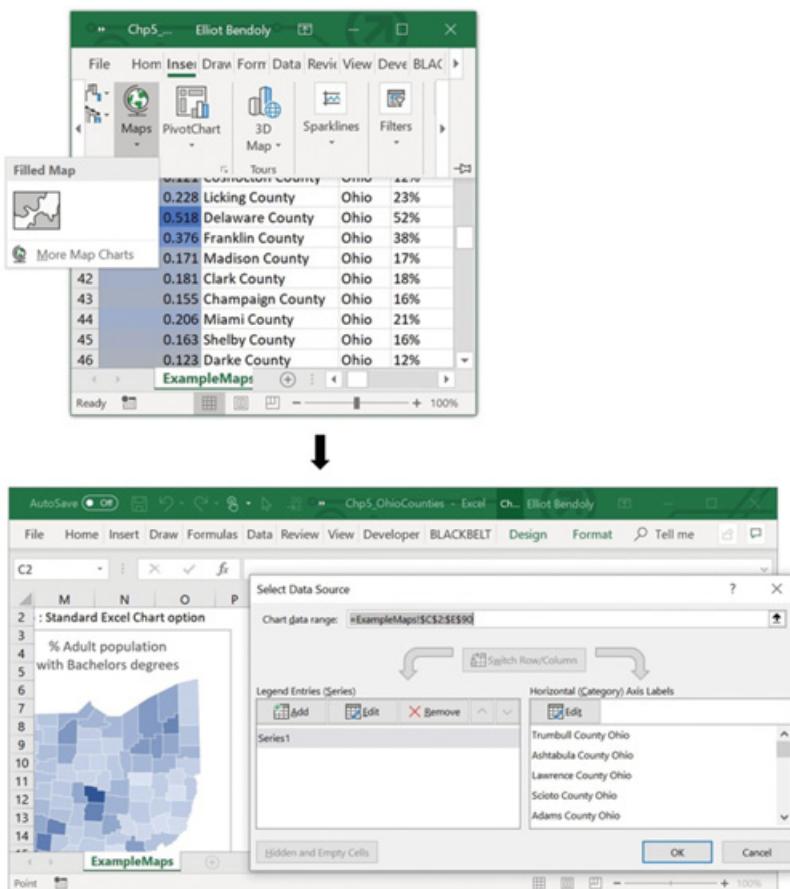


Figure 5.25 Filled Map tool and series specification

immediately upon changes to their source data, just like typical bar charts and scatters. The drawback is in the limited array of geographic partitions that are immediately recognized by the tool (e.g., you won't be able to graph data things at the block or neighborhood level, as you might with the Heat Mapper). You'll also be hard pressed to adjust the color scheme beyond the options available; highlighting any one area may require adding a series to your plot, whereas it only involves recoloring a cell in Heat Mapper with AdjustedColors dynamically active. And there are no visuals beyond these limited geographies. Nevertheless, the tool is easy to configure, cleanly packaged, and a fairly convenient option for many geographic data interests.

On the other hand, if you feel that you need an option that is not only capable of plotting data by area but also has the ability to superimpose specific data points by address or longitude-latitude, you might want to consider additional options. Manual superposition of something like a scatter on top of a Heat Mapper set, or a Filled Map, can get frustrating due to approximations that come with transforming the curvature of the Earth into the X-Y plane. An alternative for this kind of work could be the 3D Maps option, which currently draws on the Bing engine for surface details and location translation. 3D Maps, essentially what is currently access to the MS PowerMap functionality, will take either longitude-latitude coordinates, recognizable street addresses, or recognizable geographic areas (states, countries, etc.) and provide several options for plotting data as tours, saved within the Excel data file but presented through a separate interface.

As shown in Figure 5.26, where some of the tallest buildings in San Francisco are shown over time (lighter being newer), individual values can be plotted as 3D bars with height representing the value or level of the main data to be represented. For those interested, the data and two constructed tours can be found in the Chp5\_Buildings workbook. Here, in contrast to 3D bar charts, for example, the 3D nature of the bars is an acceptable addition since all three dimensions do in fact have "interval" meaning (and you can reduce the bars to thin lines if desired). Data cards describing details of each point are customizable and can be activated upon hovering over each point. Color modifications of each bar, to emphasize distinctions in some dimensions (e.g., time), can be done manually. Time panning/play can also become available if your time content in your original Excel source is "formatted" as time (interestingly, simply having time data isn't enough for 3D Maps at this point).

The option of plotting data by area is also possible and again you also have the option of having both area and point data on the same plot (if you have both kinds of data). It's akin to plotting two different series in different ways in a single chart. Bubbles or intensity plotting are both options here, with transparency customizable. With recognizable time data, the tool even

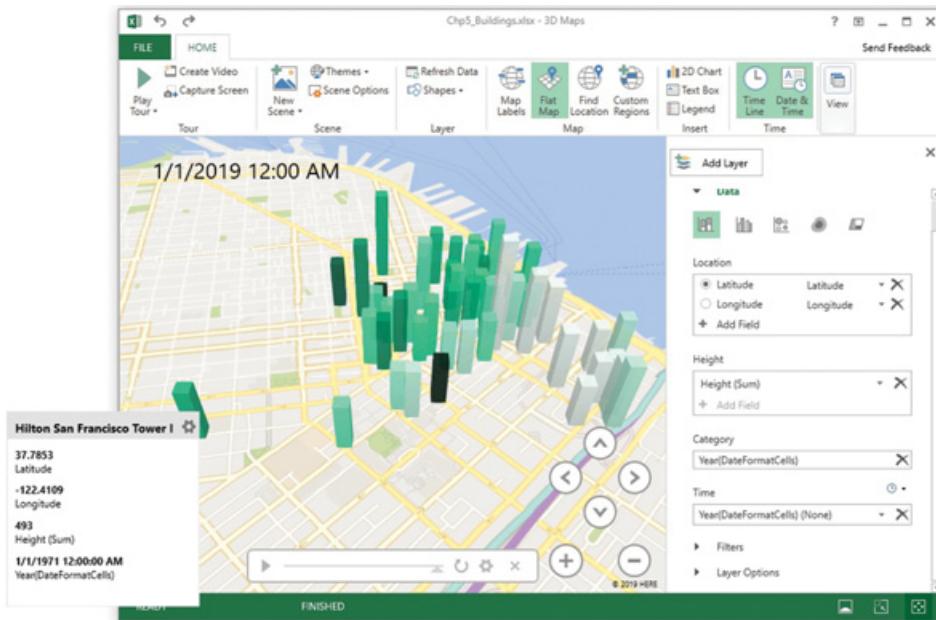


Figure 5.26 3D Map interface available through PowerMap and Bing engines

allows you to save your tour as a video for replay independent of MS Office (see Figure 5.27 and [www.excel-blackbelt.com](http://www.excel-blackbelt.com)).

#### 5.3.3.3 Power BI Desktop

Lastly, while the 3D Maps/PowerMap option represents a notable departure from the spreadsheet environment and at least a step away from more seamless options, it does retain a clear connection to the spreadsheet environment, enabling rapid refreshes when data changes. The additional interface environment is also fairly similar to the Excel environment. A much greater departure but one with some alternate added value is the MS Power BI Desktop. This is a separate application that is currently available from MS as a free download. As with 3D Maps, the tool utilizes the Bing engine and geographic data set for graphing geographic data (see Figure 5.28).

Looking at the Power BI Desktop application, you'll note a markedly distinct presentation of options relative to what we are familiar with in MS Office (at least currently, these things do tend to change). Gone is any perception of seamlessness with workbook modeling. The Power BI Desktop can import developed data from a workbook but, once it has, it's all about graphing. And depending on the type of data you have, there are many more options beyond geographic plots to work with. Some you will be familiar with, others will appear more exotic (with good reason, given the challenges they pose to interpretation).

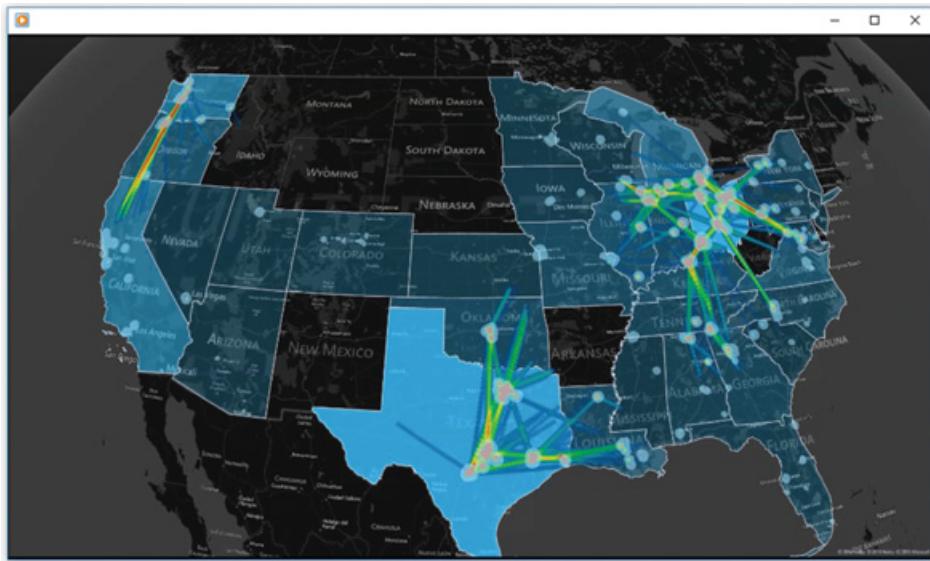


Figure 5.27 Video for playback of timelines on a 3D Map rendering

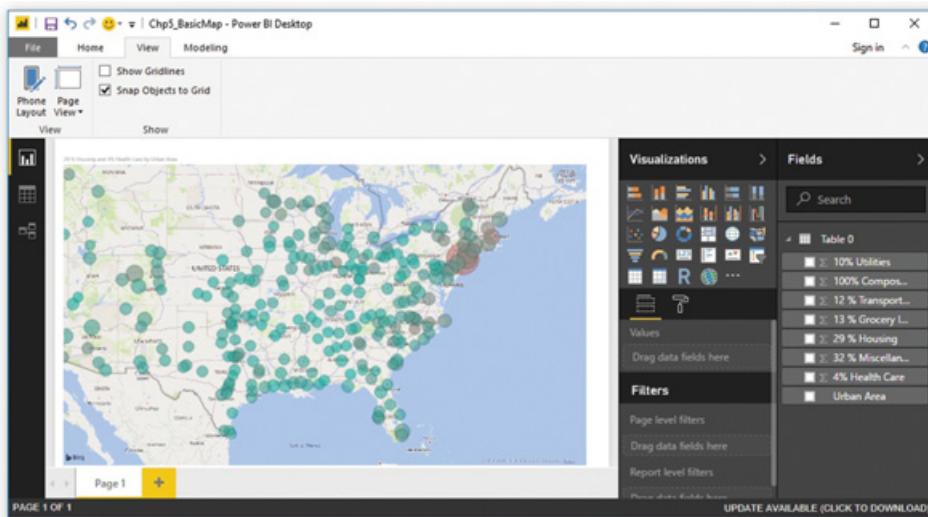


Figure 5.28 Power BI Desktop development environment

The functionality and limitations are a lot like those of Pivot Charts, which is to say, there is a lot of automatic packaging and a lot you simply can't customize (easily). It also takes much more massaging to get what you "can do" done, at least with the current interface. The visual placement of options isn't entirely intuitive, the color customization can be cumbersome, and control over seemingly basic things such as axis labeling can be just painful (even if you get used to the layout).

So with all of the current limitations, why invest time in the Power BI desktop? There are at least three excellent reasons to consider. First, as noted, there is a large library of externally developed visualization tools (e.g., Sankey diagrams) that can be added to the Power BI Desktop. These include tools that might be hard to replicate in Excel without the help of an add-in. Whether they are helpful in your work is another matter.

Second, and this is a big one, the Power BI Desktop is not just another application for making visualizations. It's also designed for sharing those visual renderings in ways that are accessible to mobile devices, via the Power BI App. Making graphics available in this manner currently requires appropriate 365 accounts, but, once available, the visuals are capable of providing interactivity for those accessing them through the App. That is, if there is any panning, filtering, or zooming that could be done through the visuals in the Desktop environment, those same capabilities can be made available to mobile viewers. The Desktop's mobile device view helps designers preview what their audience might see through the App (see Figure 5.29).

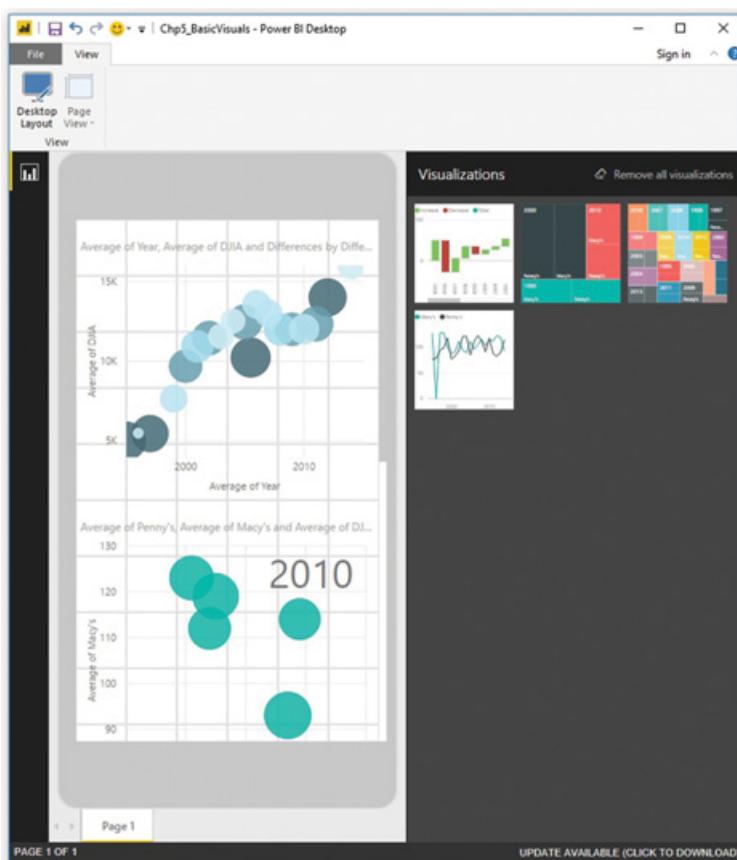


Figure 5.29 Power BI Desktop visual design for mobile sharing

Lastly, the investment in Power BI over the years suggests that the capabilities built into the Desktop tool now are just a preview of capabilities that we may see more common to regular versions of Office. The Filled Maps and 3D Maps resources after all are features only of more recent versions of Excel. Their predecessors were experimental applications like MS MapPoint and World Wide Telescope. We don't tend to encounter the latter tools by name, but much of the work that went into their designs and the study of their use ultimately gave rise to much of the geographic mapping capabilities we have in things like Filled Maps and 3D Maps. Similarly, we can see how experience with Pivot Tables and Charts and Power Pivot provided insights into the Power BI Desktop designs (with both its pros and cons). With Office 365 already dipping toes into data sharing, it is natural to expect the best parts of Power BI Desktop's graphical sharing capabilities to become more standard, streamlined, and intuitive.

## **5.4 Visualizing System Structures**

Now that we have seen a wide range of visualization tools and tactics, it is valuable to take a step back once again to talk about intent. What are we trying to capture in visualization? Will our renderings present clear messaging to our audience (even if that's only ourselves)? Do our renderings provide a sufficiently complete picture of the reality in which we are going to be making decisions?

Up to this point, we have been focusing on ways by which to demonstrate differences and trends in points and areas. In some cases we've considered the illustration of possible relationships that individual variables might have with one another and how those relationships might differ from situation to situation, group to group. These relationships may be challenging to define, particularly when there is considerable variation, where knowing the level of one variable only allows for a better "guess" at the value of another. However, sometimes these relationships are much clearer. In fact, sometimes they are explicitly defined and are a matter of the rules in place – that is, sometimes a small increase in one issue ( $X$ ) absolutely mandates a predictable shift in another ( $Y$ ); or perhaps it fundamentally limits (constraints) the values that this other issue can take on.

If there was only one linear and entirely unambiguous relationship for us to consider in our efforts to make decisions, life would be easy. If we had only two decisions to make and one was nothing more than a clear function of the other, then we would really have only the one decision to make. That, however, isn't reality. Any two decisions we face are bound together not just by a single thread but often by many. Furthermore, there are often many more than two decisions that we must face, or many facets of each decision to

be dealt with, simultaneously. The relationships among these may not be linear and they may be far from certain.

Decision making in this fuzzy web of relationships is something we'll return to in future chapters, where we discuss techniques by which to leverage technology toward developing and assessing prescriptive decision options. Before we get there, however, it is worth taking a moment to think at least a little about how we can best represent these complex systems.

### **5.4.1 Depicting Constraints**

Systems are collections of parts that meaningfully interact with one another to give rise to and impact holistic dynamics and outcomes. We can think of systems like those that operate on our computers or those that drive patterns in weather. But they can also be collections of workers and clients specific to an organization or coworkers within a team. Systems are complicated by a number of issues that are embedded in the network of connecting relationships between their collective elements. These issues include the amount of time between cause and effect (lags), the often nonlinear nature of continuous relationships between a cause and an effect (e.g., does Y follow the square of X), the potential role of discontinuities or limits (constraints) beyond which change itself pivots or potentially ceases to be possible, and unaccounted-for uncertainty (risk) regarding all of these issues.

We've spent some time talking about the connection of observable issues to their distributions (uncertainty and risk) in Chapter 4, as well as visualizing some forms of that in this chapter. Relatedly, we've spent some time talking about the estimation of connections between variables via tactics such as regression. In a similar sense, we can (and have) consider(ed) approaches to modeling that incorporate lags between predictors/decisions and outcomes as well. But we haven't spent much time talking about special limiting connections, restricting changes in utilities (decisions), and thus limiting changes in objective outcomes as well. These special connections are typically referred to in management as "constraints."

Constraints can be rules that physics imposes on us or that earlier human decisions do (think regulations, organizational policies, the number of tables that exist, etc.). They often include financially based considerations, as was implied in the Ternary Plot of Figure 5.15. In that example we have a fixed budget. It's been decided that all of it must be used. Therefore the percentage of budget allocated to each of three areas of investment must sum to 100 percent (if those are the only options). A less strict constraint would suggest that whatever is allocated simply cannot exceed 100 percent of the budget but can be below it.

Sometimes, even in the presence of a limited number of utility variables (impactful decisions), our systems impose a wide range of constraints on our decision making. Not all of these constraints will matter much, but some will be fundamentally limiting. We'll use the term "binding" to describe these latter kinds of connecting constraints in Chapter 6. But how do we know what the impact of a system of constraints might have on our ability to make decisions? How can we get a glimpse of how constraining these rules may be on the space we have available to navigate in our search for good solutions? As with all other aspects of analysis, the answer can come from visualization.

Let's say that we are working with Dodecha Solutions once again, this time in a hiring capacity. We are in growth mode, seeing the potential for a much larger number of projects, provided we have sufficient staff to market our service offerings and to actually handle the technical work of a greater number of projects. How can we develop a plan for hiring that involves both marketing/sales roles as well as technical roles? We have to start by outlining all of the facts and rules that might prove relevant to our decisions. Dodecha's management has come up with seven:

1. Dodecha has recently lost some Marketing/Sales people to competitors and retirement. No matter who else Dodecha hires, at least six should be Marketing/Sales **replacements**.
2. Dodecha has **office space** available to accommodate no more than 35 additional workers (of any kind) total. It can't hire beyond this.
3. Technical staff costs twice as much to hire as Marketing/Sales. **Hiring budget** is limited to either 60 Marketing/Sales hires, or 30 Techs, or combos in between swapping two Marketing/Sales for each Tech.
4. **Company transport and corporate relations** budget is capped. Can support either 28 Marketing/Sales alone, or 56 Techs alone; Techs draw half as much from this budget.
5. **Training** is complex for Techs. Dodecha resources can train 25 Techs alone, with no Marketing/Sales hires. Or it can train three Marketing/Sales for any one of those 25 not hired.
6. If Dodecha doesn't have enough Techs, **new work** won't get done. Every Marketing/Sales person added above 23 must be accompanied by an additional five Techs.
7. **Cultural Dynamics:** If the number of new Techs gets to 15, we can hire more but each subsequent addition should be accompanied by at least two Marketing/Sales people.

While derived from real-world issues, in this case each one of these rules is also amenable to a translation into a mathematical form, with Y as a function of X (more on tactics in this regard in Chapter 6). A summary of these rules in mathematically translated  $Y=f(X)$  form is provided in the Chp5\_Constraints workbook (see Figure 5.30), with X representing our

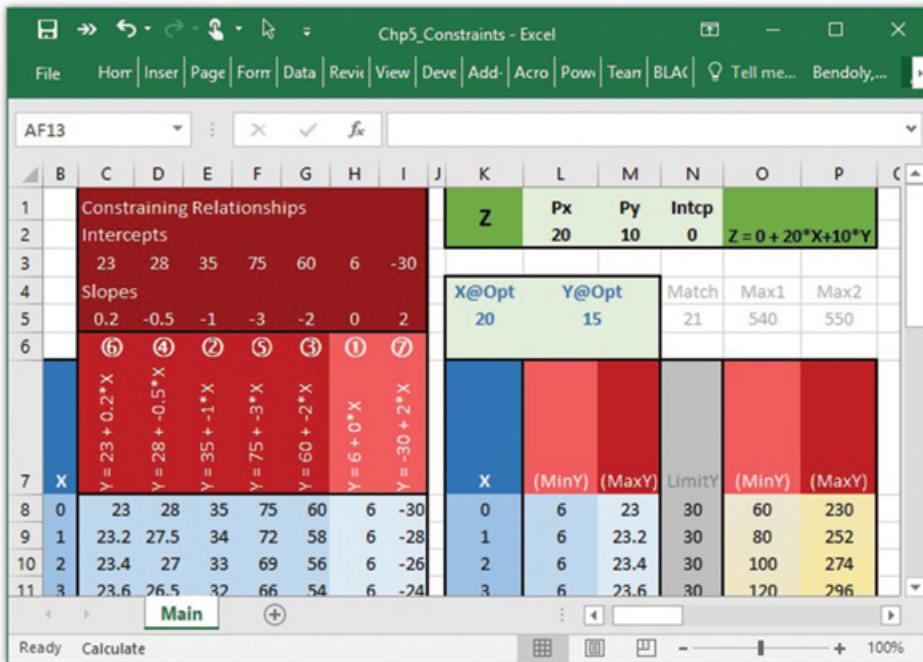


Figure 5.30 Flexible structures for mathematical relationships in decision context

first utility variable (how many technicians to hire) and Y our second (how many marketing/sales people to hire). In this workbook, these are also used to calculate the largest or smallest values that Y can take on for a given level of X. Dodecha wants the outcome of their hiring to be as positive as can be, measured by an expected Profit (Z) model based on work done by Technical staff and margin growth by Marketing/Sales efforts.

In short, all of these rules are lines in the X-Y decision space that should not be crossed. Combinations of the X,Y decisions that are beyond these lines are off limits. Because we only have two decisions in this case, we can in fact visualize the space that they cordon off. A line-connected scatter could accomplish this alone, with each equation for  $Y=f(X)$  providing data in a separate data series. Since we assume that Dodecha is looking for whole numbers for both X and Y, in a couple cases we have lines that look a bit more like staircases (“step functions”). We’ll further assume that X and Y cannot go below zero, allowing us to finalize constraint maps similar to those in the workbook and Figure 5.31.

Clearly, there is more going on in these visualizations than simply the depiction of the constraint lines. I've also created an underlay of a 2D Area Chart (using Excel's standard tools) to highlight the region that utility variables (decisions X and Y) are limited to by these rules. I've also

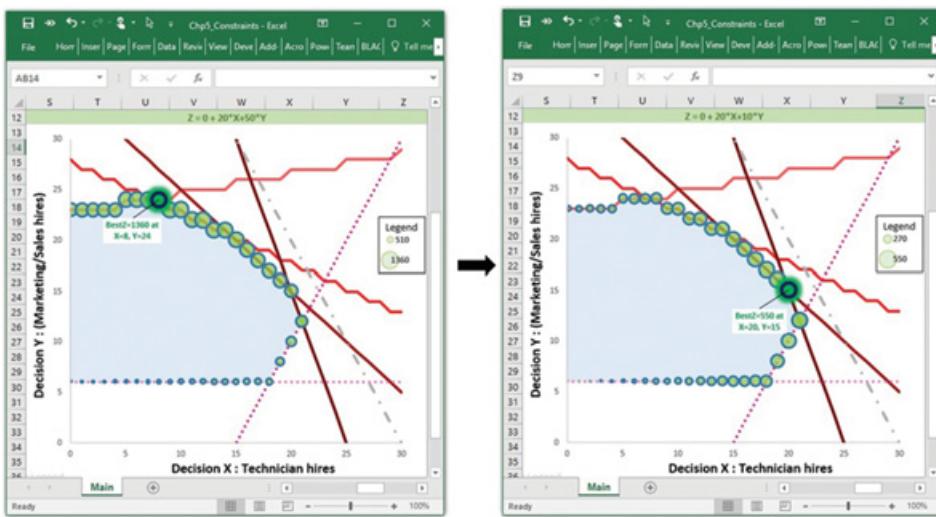


Figure 5.31 Constraints, decision space, and optimality in a two-decision context

superimposed a bubble chart of the expected Profit equation at each value of X and Y pair along the outer perimeter of this constrained space, highlighting in particular the best decision dependent upon the specific form of the Profit model and the constraints.

On the constraints side, it is worth noting that not all constraints have to actually impact decision making. In the present Chp5\_Constraints example, the cost-of-hiring differential and budget limit (appearing as a gray dashed line) is superceded by the impact of the rest of the constraints. Although it may have been something Dodecha was originally focusing on, it is an issue that won't ever come up in this situation. The others may. Understanding decision making involves understanding what you are trying to achieve and the issues that most hold you back from getting there. If we don't look closely at the rules of the game, we are likely to focus on the wrong things, play the game badly, and lose.

#### 5.4.2 Causal Mapping and Systems Perspectives

While we can get wrapped up in the wrong rules to focus on in decision making, we can also be drawing the wrong conclusions regarding the relationships embedded in these rules and the relationships that translate decisions into performance. This is not simply a matter of running good analytical models to describe and predict these relationships (though that is critical). It is also a matter of deciding on what our objective is; what we are actually measuring to gauge performance; which decisions to include in a predictive

model that describes intermediate outcomes and constraints, and which directly impact the objective. These decisions heavily draw on contextual experience. Because of this, in complex settings the conception of these potential connections often starts best a bit detached from subsequent modeling steps. We want to make sure the considerations of the connections actually make sense (practically, theoretically) before asking a tool to estimate the numerical form of relationships.

Accordingly, visual concept mapping can prove to be one of the most valuable approaches to professionals as they begin to wrap their heads around the complexity of the systems in which decisions have to be made. Once sufficient representation is achieved conceptually and supplemented by more complicated models (closed form or simulation equipped), we can then consider the systematic implications of the results to see if they still make sense. To that end it's worth considering visual approaches to both simple concept mapping as well as post hoc visual inspection of models developed.

#### *5.4.2.1 Conceptual System Structure Mapping*

One of the most impactful contributions to management over the last 100 years has been the development of tools, frameworks, and approaches in the Operational Excellence space (including the history of study in Total Quality Management, Six Sigma, Lean, etc.). In Chapter 6, we'll talk more about how our approaches to prescriptive model development have similarly been influenced by this history. For now, let's just consider one of the most impactful (albeit humble) tools used in the pursuit of Operational Excellence: The Fishbone Diagram (also known as the Ishikawa diagram or the Cause-and-Effect diagram).

Dr. Kaoru Ishikawa introduced the idea of this approach to visualization in the 1960s with the intent of supporting quality control in manufacturing settings. The diagram was designed to summarize the impact of issues within six categories (Measurement, Method, Man, Material, Machine, and Milieu/Mother Nature). And yes, when these issues are mapped out as potential causes of problems to be minimized, the diagram looks a bit like the skeleton of a fish.

However, not all of the causes of problems or the sources of performance need to fall evenly into these six categories. In some settings, alternative categories can be more meaningful. The key, however, is in demonstrating an understanding of how issues, which may be decisions or intermediate outcomes of those decisions, are related thematically and how some may be much more critical in pursuit of an objective than others. With this in mind, we can now increase the effectiveness of the traditional Ishikawa diagram by supplementing it with information critical to understanding systematic impacts. Specifically, equipped with a little descriptive data, rules, and

estimates, our conceptual map of what might be critical to decision making in our systems can provide a much more complete guide of next steps in model development and prescription.

The Blackbelt Ribbon includes a couple of concept mapping tools toward this end, one of which allows the creation of a Relative-Impact (RI) Fishbone diagram. To create such a depiction, all one needs is a set of issues/decisions/intermediate outcomes, the categories they roughly fall into, estimates of the magnitude of their impact on the final outcome of focus, levels of certainty around those magnitudes, and potential estimates of lags and recent trends in these factors. Upon selection the visual rendering of these issues is provided in a RI Fishbone (see Chp5\_ConceptMaps and Figure 5.32).

The source numbers that construct this visualization (for magnitude, trend data, etc.) may come from descriptive statistics or from earlier models. Or, with appropriately indicated levels of certainty, they may simply be best

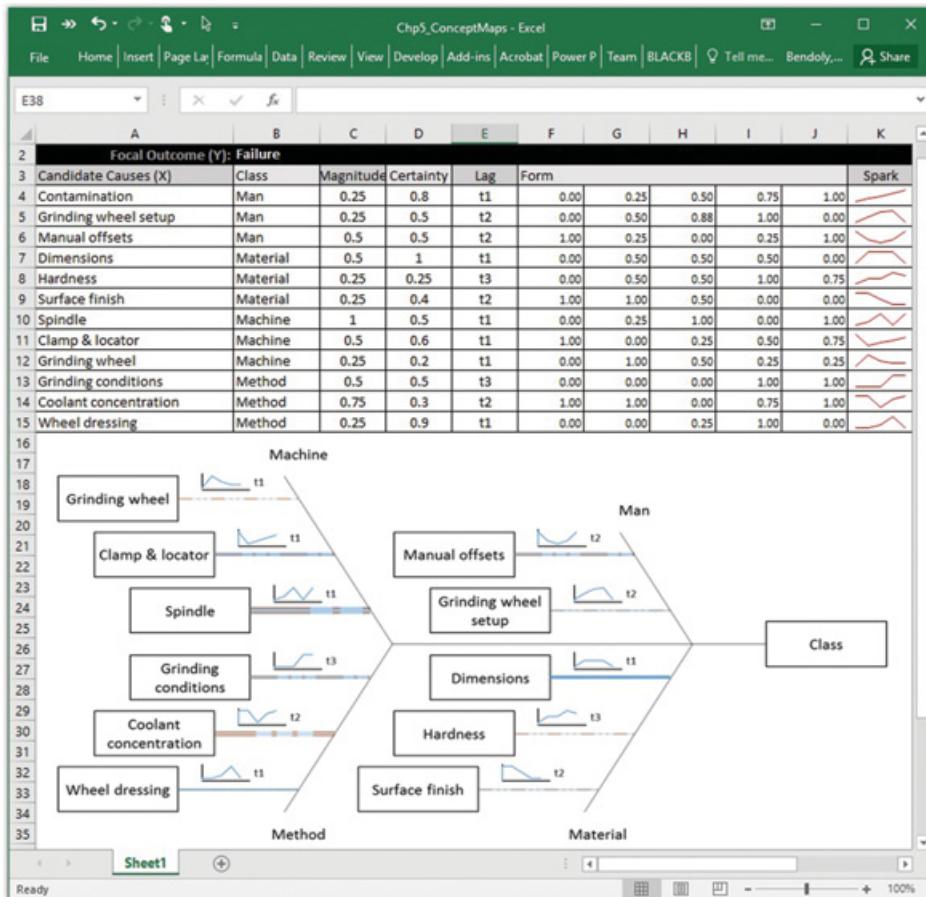


Figure 5.32 Relative-Impact Fishboning potential causes of failure

guesses. The importance is not in the precision of any individual number in this case but the relative levels represented. If factors are viewed as having little impact, or are issues that are either out of direct control or have spurious impacts, or are viewed as having trended toward limits (constraints) that cannot be exceeded, they are less likely to be levers toward performance. The intent here is to advance concept conversations and drive next steps in modeling rather than conclusive arguments for prescription.

An alternate approach to concept mapping is provided by the System Ring tool. As much as we may be interested in the impact that one factor may have on another, decision making in systems really requires looking at series of impacts holistically over time. It may be that a feedback dynamic is at play such that changes effected by one decision trigger a series of events that impact the conditions in which that decision was made, either forcing the decision to be backed away from because it is no longer tenable or making subsequent decisions of a similar kind much harder to make or modifying their anticipated impact. Feedback loops exist in more places than we often realize, which makes conclusions based on strictly linear thinking so problematic. The more we can capture our experience with a sense of how decisions and outcomes impact one another, often distinctly (e.g., X impacts Y in a manner that is not the reciprocal of Y's impact on X), the less likely we are to be surprised by subsequent system self-corrections or death spirals.

If we have a set of factors of interest and estimates of both the  $\Delta X \rightarrow \Delta Y$  and  $\Delta Y \rightarrow \Delta X$  types of impacts experienced in the past, we can start to get a sense of the dynamics of pushes and pulls that characterize systems. Once again, knowing a little about the extent to which change might be limited (upper and lower bounds) puts these changes in additional context (some issues may be close to their limits of change in some direction). These directional impacts and levels relative to limits give us a sense of the stocks and flows that characterize the complex dynamics of systems. A current stock and flow matrix, similar to the one presented on the right side of the data in Chp4\_ConceptMaps, is all we need to create a simple visualization along these lines with the System Ring tool. In this table the diagonal represents current state stock levels. Selecting an  $M \times M$  area of such a table, starting in the upper left with the diagonal bisecting the selection, the tool generates a depiction of relative impact and the extent to which changes may be approaching upper or lower limits (see Figure 5.33).

In this case we see a strong feedback mechanism by which the outcome we are concerned with (Failure) has a subsequent (in time) negative impact on a factor that we might have viewed as simply a predictor of Failure. That is, the occurrence of the processing Failure either directly or indirectly appears tied to variation at the process Setup, which, although possibly approaching

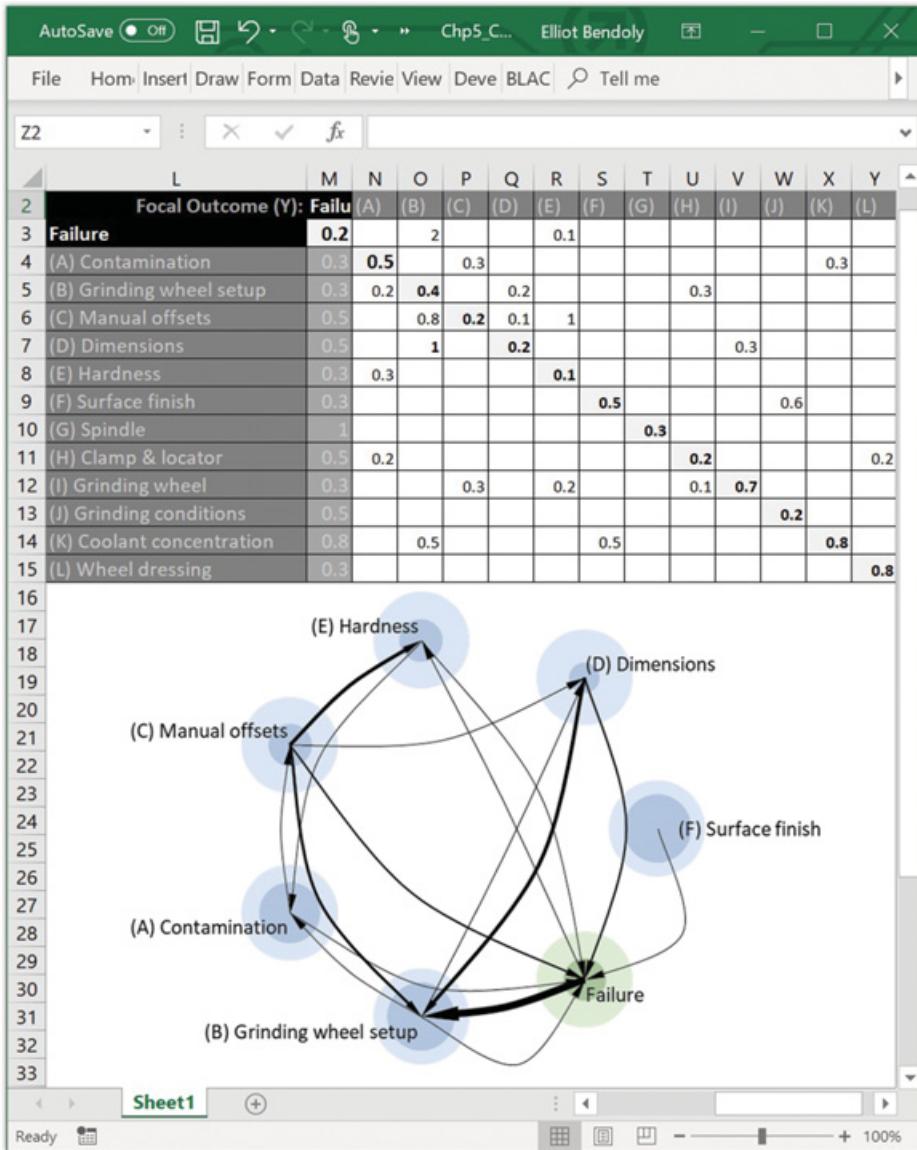


Figure 5.33 System Ring tool applied in capturing feedback relations in a complex system

an upper bound, is expected to contribute at least marginally further to Failure levels down the road. This is how processes can run out of control.

#### 5.4.2.2 Depictions of System Dynamics

Concept maps get us thinking about what might be going on in our systems. They enable us to build representative models to help us further analyze dynamics and ultimately make better decisions. In turn, the system models

that we develop can be useful in helping us understand what we are up against when assessing the risk profile of possible solutions and ultimately in selecting the best candidate solutions. Because models of complex systems are complex themselves, they are similarly prone to error. They may also possess insights embedded within them, which we might miss if we don't look closely enough. To help in both of these regards (finding errors in modeling and extracting additional insights), visualization can once again provide value. In this case we find it most effective to visualize the workings of our systems dynamically.

In Chapter 4 we in fact saw a number of systems built into simulation models. Some had dynamics that evolved over time in somewhat complex ways. The Lobo's Cantina inventory ordering case provided one example of a system simulation that we found a way to examine through repeated periodic evaluation and descriptive statistic comparison. But that same example, starting further down in row 36, also includes a visualization of the periodic stocks and flows of the system, driven by a living record – a convenient application, given that the simulation itself was already working in iterative calculation mode.

Figure 5.34 shows the nature of inventory levels as they are chipped away at and resupplied (the prototypical sawtooth diagram of inventory management) as well as the occasional occurrence of stockouts. The spikes in inventory represent the arrival of vehicles, whereas the dashed line represents the

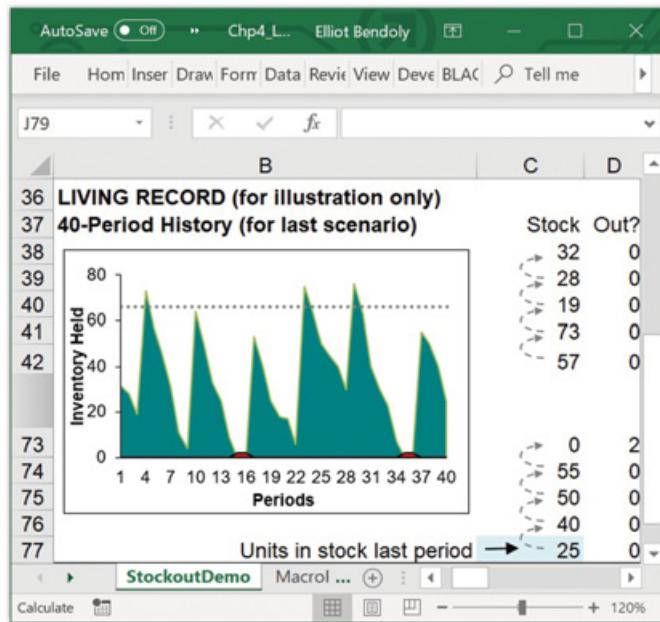


Figure 5.34 Dynamic depiction of inventory held, stockouts, and replenishment in simulation

reorder point. Adding a visual of a truck en route would add little to our understanding here, but the dynamics of stocks and flows depicted and the ability to see these evolved dynamically (F9 in single iterative calculation mode) can confirm that our model is operating in a representative way.

Having said this, the development of dynamic representations of physical movement can also be useful toward both discussion and modeling. On the Blackbelt web site ([www.excel-blackbelt.com](http://www.excel-blackbelt.com)), I retain the video rendering of the 3D Maps visualization presented in Figure 5.27. Seeing the timing at which various flow patterns emerge and cease and where others initiate can again help in understanding trends that may be more complex than simple linear ones. Similarly, in a pinch, a scatter plot could be retrofitted to depict the movement of clients or personnel in a service setting (see Chapter 5 Supplement) or vehicles along supply routes. Scatter plot points after all can take on the appearance of images just as portions of bar charts can (Figure 5.7). The Chp5\_Crossings workbook provides a somewhat alternative approach, controlling the movement of vehicles based on decisions specified, not through an Excel chart but rather through VBA (we saw macros in Chapter 4 and will learn more about these in later chapters). In this case the visualization was used not for the development of a managerial solution but for the development of an educational and professional training exercise. Eventually this kind of proof-of-concept build provided the foundation for the development of an instructional App (Figure 5.35). Visualization can provide the inspiration for many things.

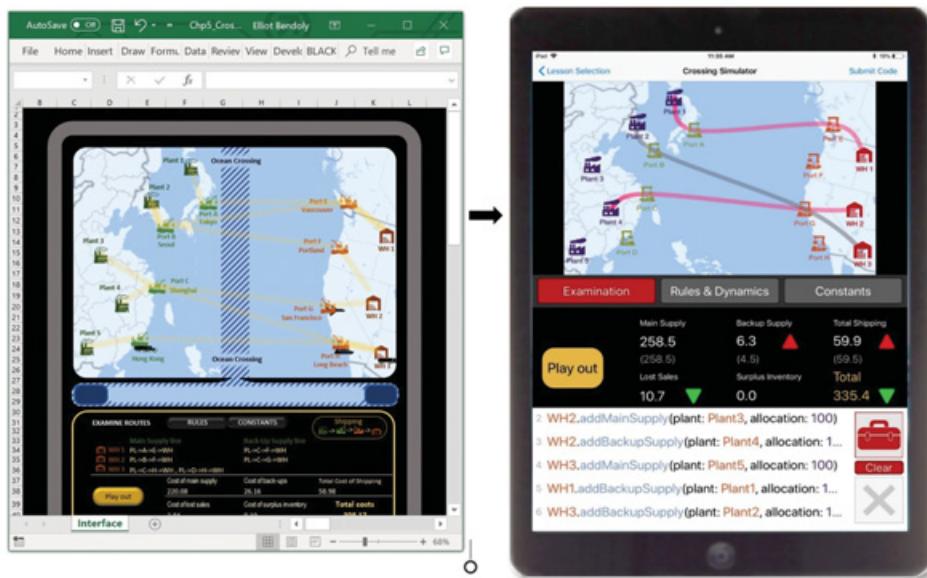


Figure 5.35 Dynamic flow along directed paths in a scatter plot and App translation

## 5.5 Options in Other Platforms

It goes without saying that there are many alternatives to Excel when it comes to visualization. Tableau is an incredibly easy-to-use alternative that permits drill downs, aggregation, filtering, panning, and many of the user interaction options we've demonstrated in Excel. It is designed with an outstanding capacity for handling large, mutable data sets and for sharing renderings across platforms. If you've used Tableau, you can see what MS Power BI Desktop is competing against. Both are designed for interactivity predominantly with nonanalyst audiences who won't feel put out by the very well-packaged but notably limited options. It does the job it was designed to do, however, and is worth getting to know through any number of online guides. It's a fast learn and can come in handy if available at work.

Google provides another option for sharing data and visualization, integrated with the many mechanisms for selective access, form completion, and so on that can augment that collaboration. While visualization options are not outstanding, the most common visual forms are perfectly workable. Google Sheets are also available ubiquitously for everyone to use, regardless of platform choice and application investment. Further, there is a level of interactivity between Google and Excel that has grown over the years through the use of Google API, Apps Script, and to some degree developments on the Excel side. The Blackbelt Ribbon's Read Write-G was in fact designed originally to focus on reading from Google Sheets and writing to Google Forms. If you are aware of the Google Form key codes for a form you want to write to (and hence to a Google Sheet linked to that form), the bottom section of the Read Write-G tool will make it happen for you. Those writes can in turn impact any shared models and visuals dependent on such data. More on this in Chapter 9.

Other resources such as R are also worth mentioning. R is another ubiquitous resource and something worth getting to know if you are working with extremely large data sets and rather sophisticated statistical analyses. It's certainly much harder to ramp up on than something like Excel, Google, or Tableau and, to be clear, it is not really designed as a visualization tool. Nevertheless it has an increasing number of interesting application components that can contribute to visual discussions. Take, for example, its facility with 3D scatters, the ability to easily depict best-fit planes (bivariate regressions) and confidence ellipsoids (confidence ellipses extend into an additional variable dimension), as shown in Figure 5.36. Perspectives for these renderings are seamlessly rotatable through cursor interaction. While edits to the nature of these graphs are not typically point-and-click, their features are editable in R script. And if you are working with R, you are working with script, so it's not much of a departure from standard use.

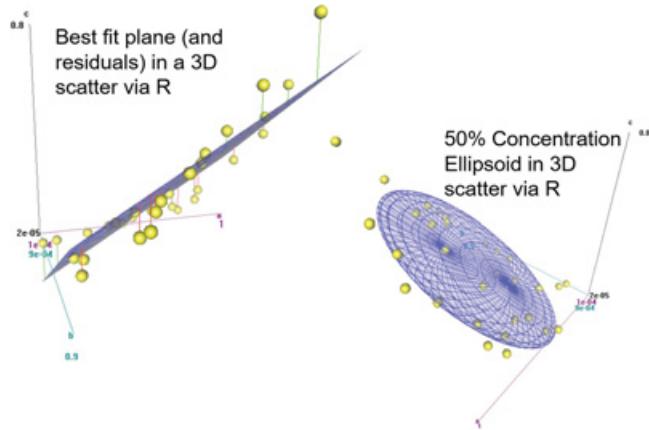


Figure 5.36 3D scatter plots, best-fit planes, and confidence ellipsoids in R

As we will see, the Palisade suite, which we visited in discussions on data work in Chapter 3, also provides some useful visualization capabilities either as artifacts accompanying analysis reports or as live views of the progress of lengthy analysis procedures. As with R, if your task is to visualize something, you aren't likely to go to Palisade first. But if you are leveraging the Palisade suite for analysis, it's worth taking advantage of the visual utilities it provides. You'll learn much more about the nature of the analysis and its conclusions by doing so.

### **Supplement: Dynamics along Visually Derived Paths**

In many real-world situations the kind of dynamic paths experienced take on complex forms that aren't easily described by a series of formulations. Some of these involve spatial considerations (e.g., internal or external physical infrastructures of facilities); others involve nominal transitions that are conceptually diverse and complex (e.g., representing the processing of new design ideas, or the retooling of staff in preparation for new work deployment). Let's consider once again the Lobo's Cantina context from earlier (Figure 5.37).

This is nothing more than a scatter plot superimposed on a graphic of an extremely stylized restaurant floor plan (*Chp5\_LobosFloorPlan*). The waiters are scatter plot points along more complete paths, whose additional points are not shown here. A closer examination of these paths is provided in Figure 5.38. If equipped with a schematic of where the lines of traffic occurred, either the ClickPlot tool on the Blackbelt Ribbon add-in or the PolyPtsExtract function in that same add-in would be sufficient for translating a series of clicks along those routes, or

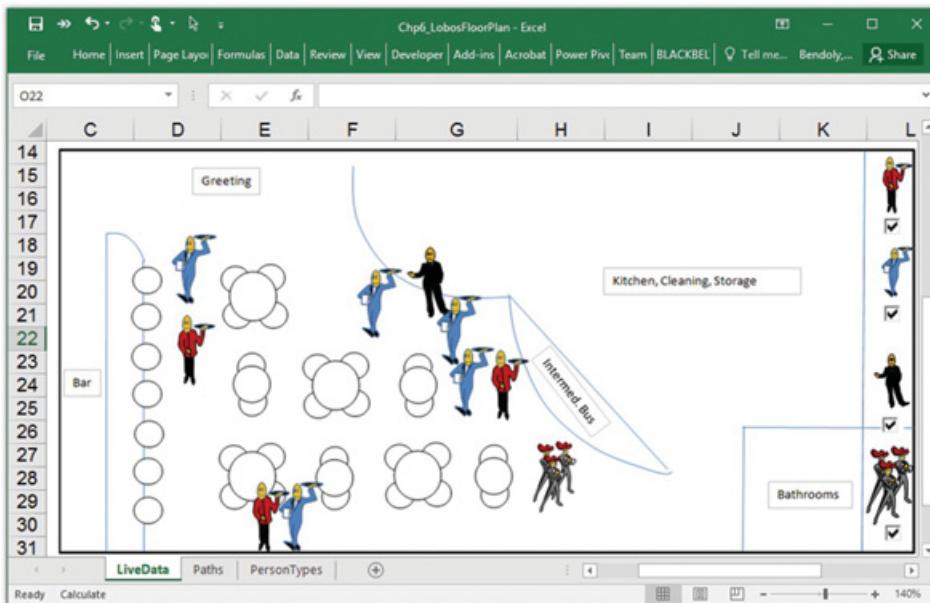


Figure 5.37 A dynamic multielement path-based simulation

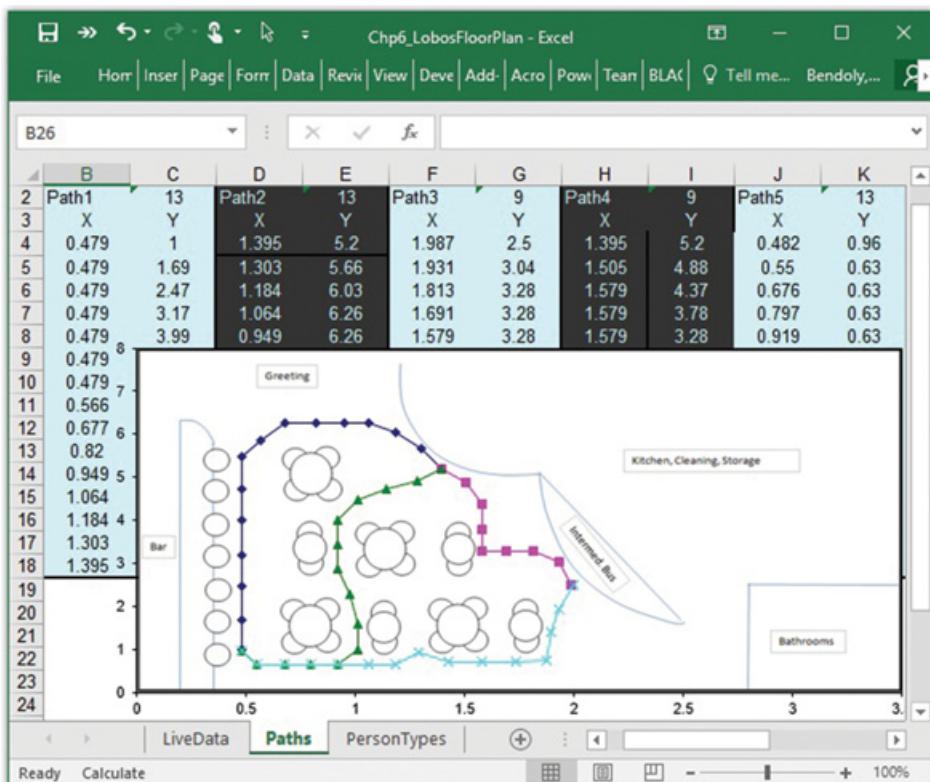


Figure 5.38 The point-structure used in the floor plan example

a drawn polygon of that route, into a relative set of coordinates at the foundation of these visual simulation paths.

On the PersonTypes worksheet, I specify the transition matrices relevant to this graphical simulation. In this case these matrices specify the probability of an individual shifting onto an alternate path upon completion of the one they are currently on. A lookup mechanism is used to determine where to go next based on a random number pull among eight possible states. Different workers may be described by different types of movement and so the transition matrices of the busboys are not the same as that of the manager or the mariachi band, for example. Along with these transition matrices are cumulative matrices that are used when Excel actually needs to look up what random path the worker(s) will embark on next.

Ultimately the logic used here is no more complex than the structures discussed in other examples of iterative calculation mode or those discussed regarding discrete random number pulls. The complexity comes into play only when you consider how all of these elements and techniques come together to form a single integrated system. The caveat here is that even the coolest (or simplest) graphics can get messy at times. When complete graphical depictions limit the capability to get a point across or come to an understanding, partial depictions of representative or critical data are more useful.

## PRACTICE PROBLEMS

### *Practice 5.1*

- a)** Develop two scatter plots based on the data in Chp3\_QuarterlyData. Specifically, use the data summarized by the Pivot Table on the PivotTable sheet of that workbook. Do not apply any filters on these company averages. This should give you about 200 records to work with.

First plot:  $\ln(\text{Avg R\&D})$  on the X-axis; Avg Price Close on the Y-axis.

Second plot: Avg Inv/Sales on the X-axis; Avg Price Close on the Y-axis.

Add best-fit parabolas (also known as second-order polynomial fits) to each plot. Display the best-fit equations.

- b)** Use these equations in a separate set of cells to calculate the difference between each observed value of Avg Price Close and the equation's estimate of Avg Price Close for a given  $\ln(\text{Avg R\&D})$  value. Do the same for the second fit. This is variance that the fits have not picked up on. Create a third scatter plot of these differences against one another for each value of X (first Difference vs. second Difference).
- c)** Create a fourth plot, a bubble chart, of  $\ln(\text{Avg R\&D})$  vs. Avg Inv/Sales with Avg Price Close as size.
- d)** As a fifth and final plot, assuming normality, use the Confidence Ellipse tool in the Blackbelt Ribbon to create a 75 percent and a 95 percent confidence area for

the  $\{\ln(\text{Avg R\&D}), \text{Avg Inv/Sales}\}$  joint distribution. Superimpose this on the bubble chart so that the axes overlay and correspond in scale.

### *Practice 5.2*

Create two columns of numbers. The first column should contain integers from 0 to 20. Label this column Apples. The second column should contain the following formula:

$$= \text{SQRT}(20^2 - \{\text{Whatever \# of Apples are in the adjacent cell}\}^2)$$

Call the second column Oranges. Create a line-connected scatter plot of the two columns and call the plot Production Frontier.

Pick one cell in the sheet and label it Location. Label the cell to the right of it Direction. Turn on single iterative calculation mode. Use the following calculation to determine the value inside the Location cell:

$$= \text{Location} + \text{Direction}.$$

Now use an IF statement within the Direction cell that makes Direction equal to 1 if Location is 0 or is less than or equal to 0, if Location is greater than or equal to 20, or unchanged at all levels in between. (You can figure that last part out for yourself.)

Use a VLOOKUP to set two additional cells in the sheet (below the Location cell) equal to the number of apples and number of oranges that are given in the row corresponding to the value of Location (e.g., the 0th row, the 1st row, the 20th row). Add that X-Y data point as a new series to your plot. In iterative calculation mode, F9 should move that point back and forth along the production frontier you've plotted.

Note that you may want to include an IF statement in the Location cell tied to a Restart (or Toggle) cell so that you can make sure that things start the way you want them to (i.e., at Location=0) when the iterations begin.

### *Practice 5.3*

This problem focuses on practice with nonstandard heat mapping.

- a) Select a sample shape set from the Heat Mapper add-in. Create the set using the tool. Modify the default color scheme using the header row of the labels and data generated. Also apply different color formats to five of the left column labels. Use the AdjustColors() function as described to make sure changes are playing out in the visual mapping.
- b) Now create a shape set that doesn't already exist in the library of Blackbelt Heat Mapper tool. This might be based on a city or regional map, a blueprint, or even a photograph. Critically, you'll want to draw individual polygon areas and name each meaningfully. Convert them to a data set using the Image Extractor tool in the Blackbelt Ribbon. Submit them to your instructor, or to [bendoly.2@osu.edu](mailto:bendoly.2@osu.edu) with your instructor's class and contact information, along with a name for your set as a whole and a library placement recommendation. If your set is well constructed, we will consider adding it to the global library.



## **Section 3**

### Prescription Development



# 6

## The Analytics of Solution Search

**Objective:** *Gain a familiarity with approaches to constructing models for prescription development, as well as basic approaches to deriving prescriptions from those models.*

So, you've acquired and cleaned your data; figured out some likely relationships within your data, more or less; visualized risk and limitations representative of the system you have some knowledge of. Where do you go from there? Where would you like to go?

You'd probably like to get to a prescription. A proposal of some specific action. Some answer that would pay you back for all the descriptive and predictive efforts you've invested in. But even having a comprehensive view of the relationships between decisions and outcomes, decisions and constraints, doesn't mean that you have an immediate view of a prescription. In fact, once you've spelled out all of the relevant system relationships, the task of identifying a best solution in that landscape may be daunting.

Certainly, you can always attempt a little snooping. For example, the visualization presented in the Chp5\_Crossing workbook or in the associated app-based activity (SupplyRisk QB) relates to the task of finding an effective way, using some combination of overseas manufacturers, to supply three warehouses of differing quality and cost, transportation modes, and order amounts. You could just start trying out one possible solution and see how it plays out; then another, maybe just a slight variation on the first; then maybe another, a more radical alternative; and so on. You make big pivots here, small tweaks there (for a recent study on manual search patterns, see the *Management Science* article titled "How Do You Search for the Best Alternative?" (Sommer et al. 2020)). You could learn a great deal from trial and error this way, but you could also miss a great deal. To be sure, unless you have a remarkable ability to recall the performance of all prior attempts, documentation is going to go a long way.

The importance of documentation in fact applies to both the process of searching for solutions as well as the process of constructing models – sufficiently representative of real-world scenarios – on which those searches are conducted. At the simplest level, structured documentation allows you to avoid retrying efforts that have failed. At a more sophisticated level, structured documentation can allow you to triangulate multiple pieces of information so that you can draw conclusions about likely best paths for improvement. It facilitates the cohesive sharing of ideas, input into their modification, and iteration on their integration. It can also give you a sense of the returns to your search effort as you get closer to a good solution (i.e., it can help you know when to stop searching).

The same principles can be and are leveraged by computerized search mechanisms. The best computerized searches take what they have learned from each step in their examination and use that information to move forward intelligently toward improvements, making incremental changes when refinement is needed and pivoting big when otherwise stuck, and knowing when enough time has been spent searching by examining trends in objective improvement.

In this chapter we will begin by looking into structured approaches to documenting and thereby developing models of decision-making contexts – systems with objectives to pursue, utilities (impactful decisions) that can be modified, and connections that may limit them. We will then get into a discussion of how we might methodically search these model landscapes for the best, or at least acceptable, prescriptive solutions. We'll also examine the structured documentation that accompanies some of the searches that computerized approaches produce.

## **6.1 Encapsulating Decision-Making Contexts**

Developing models that sufficiently connect objectives to decisions and constraints requires steps involving fact collection, relationship description, and the systematic examination of assumptions regarding these and their combined implications. In some cases the modeling challenge faced is at least similar to ones encountered in the past, and therefore either an explicit or implicit structure exists that we can borrow from. However, a given decision-making context can also be relatively foreign to our past experience. In these cases, in particular but not exclusively, we are aided by frameworks that can guide us in documenting and advancing our modeling work.

Any structured process of proposing, discovering, deploying, and scrutinizing models of reality, particularly in otherwise unfamiliar contexts, inevitably gives rise to critical thinking. And we need to think critically if we want to be in a position to propose thoughtful prescriptions. The value

of such structured processes is tied to the value of repeatedly returning to earlier stages in model development, to revise assumptions and directions taken, before some decisions are sought out and acted on. Too often individuals are not willing to go back and revise assumptions. Structures that require documentation of these assumptions fundamentally change these dispositions. They force greater scrutiny at each step and encourage the most appropriate assumptions of real-world systems to emerge.

Ultimately, good structuring processes will tend to be reminiscent of many other established concepts and frameworks that have been discussed in disciplines such as those of management. These include frameworks focused on continuous improvement (PDCA or A3 frameworks), the Thinking Process of TOC, the six steps of critical thinking, and so on. For our purposes we can capitalize on a process that has proven to be effective in model development in the past: the OUtCoMES Cycle (or Roadmap in its earlier discussion in the second edition of this text).

### ***6.1.1 Stages in the OUtCoMES Cycle***

Similar to some other frameworks, the OUtCoMES Cycle is a fundamentally recursive one. It involves refinement after stepwise documentation and assessment. Figure 6.1 depicts the components of this discovery cycle. Note that although the first four stages here might typically fall under the umbrella of Plan (P) in PDCA, in reality artifacts are created at each stage (D) and the effectiveness of their development is checked (C) with recourse to revisit earlier stages. Acting (A) on derived prescriptions will modify context conditions, setting up the next stage of inquiry, and repetition as well.

The OUtCoMES stages include three items that sound like nouns (Objectives, Utilities, Connections) and three others that sound like verbs (Manifest, Explicate, Scrutinize). This is deliberate. A critical part of planning (and one often mismanaged) involves outlining specific points of information

Figure 6.1 Stages in the OUtCoMES Cycle of problem structuring and resolution

prior to operationalizing workable models for decision making. Further, descriptions of some of these points must precede drilling into others predictively. Explicitly spelling out the sequence of these considerations (Objectives → Utilities → Connections) reinforces the flow of descriptive and predictive analysis in advance of the act of prescriptive model construction (Manifest), its evaluation and extraction of possible solutions (Explicate), and close examination of their realism and implications (Scrutinize). Again, all these stages are prior to acting upon these recommendations and are often followed by regular returns to earlier stages of consideration.

Let's look into each of these stages in depth.

#### *6.1.1.1 Objective Specification*

Every decision-making task should begin with the following question:

**What do you believe you can reasonably accomplish, given what you know about the context?**

Although it can be challenging to come up with the most salient objective in a new, relatively unstructured (at this point) scenario, the development of “candidate” objectives is always within reach. The best thing that can be done in the first stage as well as subsequent iterations of this stage is simply to come up with a list of these candidates. It may be a short list, but try to come up with two or more outcomes or directions for improvement that could be reasonable and worthwhile pursuing, with as much definition of each as necessary to avoid ambiguity.

Some of these candidates may simply be variants or derivatives of others. Some of these ultimately describe subobjectives that might be folded into other aspects of your solution. Developing a prioritized list of these will help set priorities for and guide subsequent structuring stages, while keeping records of these lists, and changes to them as they are revisited, provides a history of thought that can inform later aspects of the structuring process. Notes serve to track rationale regarding the priority of candidate objectives and the selection of each as structure evolves. Specifying details such as the natural bounds on these objectives (i.e., an objective might not be something that can go above or below certain levels) will help rationalize their priority as something to be pursued. Table 6.1 provides an example record-keeping mechanism for outlining, ranking, and annotation of candidate objective considerations.

In the above example the notation “Iteration: 1” and “Prior State: NA” suggests that this is the first (probably not the last) time the consideration of candidate objectives has taken place in the current process. If the decision to reconsider objectives is made at a later stage of structuring – for example, “Manifest” – an additional record would be logged with associated notation (e.g., “Iteration: 2” “Prior Stage: Manifest”).

Table 6.1 *First iteration on documenting candidate Objectives in problem structuring*

Objective Specification		Iteration: 1			Prior Stage: NA	
Candidate	Definition	Current State	Bounds	Priority	Notes	
Profit across ...	The net difference ...	###.## [units]	(##-##)	1	Incorporates ...	
Cost within ...	The sum of ...	###.## [units]	(##-##)	2	Only captures ...	
...	...	...	...	3	...	

### 6.1.1.2 Utilities Specification

Specification of likely objectives sets us up for the next natural question:

**What are the factors that you (or decision makers in question) are in control of, and which do you believe you can use in pursuit of the highest-priority objectives stated?**

Utilities are managerial decision variables that specifically impact the objectives of interest. Typically, managers have a host of decisions they can make, not all of which bear on specific objectives. Still other variables that impact objectives are simply not practical managerial levers. Here the focus is on outlining those particular levers that both are viewed to be impactful and can be reasonably manipulated by management. Hence the term “Utilities” is used to emphasize this subset of decision levers among all others available to managers. Similar to the case of outlining candidate and likely objectives, record keeping on candidate utility factors can be instrumental to the development of structured solutions. Describing the anticipated relative impact (on the main objective) of each utility, accounting for both the natural bounds (limits) on these variables, is particularly critical at this point. Table 6.2 provides a comparable record-keeping structure:

To help justify current and future utility choices, and in advance of the next stage, it may be useful to outline some descriptive details and assumed (or well-quantified) connections to a key objective through use of a tool such as the Relative Impact Fishbone of the Blackbelt Ribbon add-in (Chapter 5). As this is a stage in advance of more explicit modeling (Manifest), the intention is not to pin down all impacts statistically but rather in a theoretical form based on experience. Predictive modeling (does one decision in fact appear to historically drive the objective?) will almost certainly follow soon after. But getting these likely levers down and mapped in concept is the first step. It is important to note that some of these factors may be very interrelated, albeit indirectly, through intermediate processes and outcomes – something that will further emerge in stages that follow.

Table 6.2 *First iteration on documenting candidate Utilities in problem structuring*

Utility Specification		Iteration: 1			Prior Stage: Objectives	
Candidate	Definition	Current State	Bounds	Impact	Notes	
Staff count	Total number of ...	###.## [units]	(##-##)	8	Aggregate ...	
Training Spend	Investments in ...	###.## [units]	(##-##)	3	Specific to ...	
...	...	...	...	1	...	

### 6.1.1.3 *Connections (and Constraints) Specification*

Now we need to connect the dots. We ask at least two kinds of questions:

**What limits exist on the way in which utilities can be modified as a set? What are the limits to the impact that those decisions can have on objectives?**

Here we use the term “limits” to refer to both hard constraints in the manner in which utilities can be modified as well as variable phenomena such as diminishing returns to the objectives. Further, as alluded to in the description of the last stage, many limitations that are present in reality are relational in that the severity of their impact on one utility or objective may be dependent on levels of other utility decisions. For example, the time available to pursue one activity can be limited by the time one decides to allot to another activity.

In the Connections specification stage the task is to list all such limits, relationships, and dynamics with sufficient detail to appropriately depict the implied interplay between utilities and objectives. While it isn’t critical to provide math at this point, at least documenting which utilities might be involved/impacted by definition of these connections will prove helpful.

Further, because some constraining connections may be more relevant than others (more likely to “bind” decision making), it is useful to continue to present those perceived to be currently the most limiting, even if we end up having to revise these perceptions later on. We capture this information through estimations of “current slack.” For example, if we are currently using all the space available, there is zero slack for making use of more space. If we are using only half of the space, we may have plenty of wiggle room. Indicating both absolute values of slack as well as subjective descriptions (e.g., High) will help keep our thought process grounded (see Table 6.3). This is true even if all constraining connections are ultimately included in structured solution development.

Table 6.3 *First iteration on documenting candidate Connections in problem structuring*

Connection Specification		Iteration: 1		Prior Stage: Utilities	
Candidate	Utilities involved / Definition	Abs. Slack	Rel-Slk	Notes	
Space limits	Total amount of square feet ...	###.## [units]	High	May need subdiv. ...	
Diminish. Ret.s	Stock buffers variability ...	###.## [units]	Low	Assumed math is ...	
...	...	...	...	...	

In the process of outlining these connections, particularly those that are relational and refer to multiple utilities and/or objectives, it is also natural to begin to develop concept mappings that emphasize the holistic web of Connections between Utilities and Objectives. Hence the “Co” in the OUtCoMES Cycle is not just a prompt for the consideration of connections and related constraints, but also a reminder of the critical role of concept-mapping activities and the outlining of connections. Similar to tabular record keeping, the tracking of versions of these concept maps can prove useful to subsequent iterations.

The System Ring tool (Chapter 5) may provide a starting point for such mapping, with some manipulations and addenda, but so can a standard box-arrow diagram. Figure 6.2 provides an example of such conceptual mapping.

#### 6.1.1.4 *Manifest (Operationalize Your Full System Model)*

The specifications in stages 1–3 certainly provide the backdrop for the development of a structured solution search. However, they don’t venture into the critical stages in which the implications of these connections are considered. This is the role of the latter three stages, with the Manifest stage serving as a transition point.

When an ephemeral idea takes on a codified model form, we often say it Manifests itself. Now its appearance isn’t something supernatural (as in novels like *Estate* or *The Peddler of Wisdom*). Rather, it’s often the result of a great deal of effort and consideration. To get from the details we’ve compiled in the last three stages to a fully codified model, we ask questions such as the following:

**Given the web of connections that exist in concept, what is the magnitude of impact when one utility decision is increased twofold? What is the related impact if another is brought to zero?**

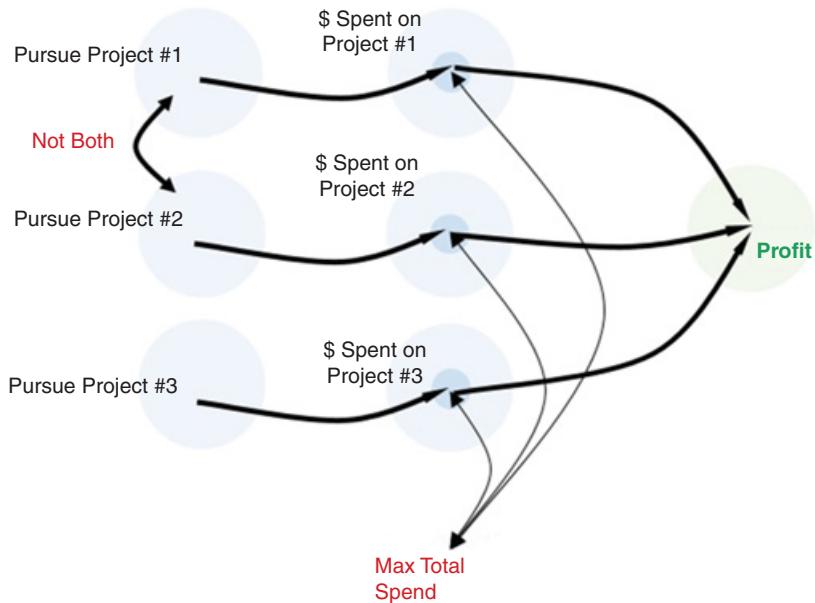


Figure 6.2 Example concept mapping of Connections between Utilities and the Objective

In the Manifest stage we attempt to develop an understanding of the degree to which specific levels of change in Utilities impact the primary Objective (and sub-Objectives) as well as the extent to which such changes further limit or open up opportunities for other Utility decisions. We focus here on how the magnitude of input changes impact the context for and degree of objective changes. Although this is often an ideal place to begin putting down mathematical representations of these relationships, math itself is not a fundamental requirement. Magnitudes of impact can be depicted in general terms if they cannot be fully codified (e.g., “A large investment in Plan A will yield greater gains than will similar investments in Plan B”). In fact, in many cases general models of magnitude impact can be far more useful than forcing one to make unjustified mathematical claims. In other cases estimations of magnitude are derived statistically from available data.

Ultimately the Manifest stage is simply about expressing the dominance of specific impacts so that these impacts can be viewed with relatively greater prioritization when solutions are being sought. However, when we do have the advantage of sufficient data, descriptive and predictive analysis results, and codified rules, we are well positioned for the development of mathematical models with the prospect of using such models as the foundation for computationally assisted search.

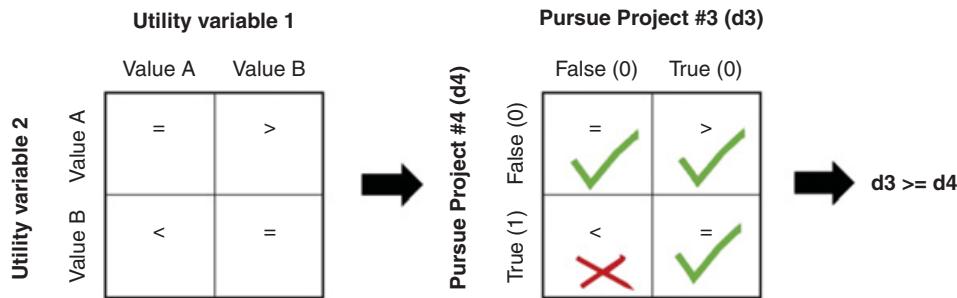


Figure 6.3 Example use of a logic matrix for developing math forms in constraints

If we are to develop a math model with this intent, we will benefit from the development of some notation for short reference to variables; for example, letting “ $d1$ ” represent a binary choice (0,1 variable) on whether project #1 is pursued, letting “ $x1$ ” represent the amount spent on project #1 and “ $m1$ ” represent the max spend on project #1, and so on.

Notation development is typically viewed as simple, but it is not without its own pitfalls if done in a haphazard way. The more challenging task here is typically the *translation* of a concept map (suggesting linkages between the objectives, utility variables, and constraints) into specific mathematical forms using this notation. Again, building on the above example, the relationship between deciding to pursue project #1 and the “spend” on project #1 might take on the following mathematical form, given the stated notation: “ $x1 \leq d1*m1$ .” The requirement for only either #1 or #2 being pursued, or neither but not both, might be stated as “ $d1 + d2 \leq 1$ .” If the connections between two variables were a bit different (e.g., if you can’t pursue project #4 if project #3 isn’t pursued), then we’d have to come up with a different manifestation.

In general, when working with connections that relate two binary variables, considering which areas of a  $2 \times 2$  logic-matrix apply is often helpful (see Figure 6.3). In such a matrix, if both variables can take on equal values (both 0 or both 1), part of the relationship must be an “=.” If the first variable can also be greater than the second, but not less, the full relationship must be “ $d3 \geq d4$ .”

Record keeping for the Manifest stage may be less amenable to a tabular format, but that doesn’t mean tracking shouldn’t or won’t take place (similar to the case of concept mapping in the last stage). Notation penned for utilities and their relationships to the objective and each other in fact serves this purpose rather well. Keeping track of general statements of impact magnitude, or mathematical notation and formulae as the case may be, will prove useful as the process proceeds and as these points are returned to for further development and revision.

#### *6.1.1.5 Explicate (Extract Prescriptions and Their Attributes)*

Codified relationships in hand, we are now ready to consider approaches to deriving solutions. We now ask:

**What do we need to do to evaluate the model in hand, so that we can generate outcomes that help us understand how effective the model is?**

To Explicate is to analyze with the intention of extracting meaning. That is essentially what we are doing when we make use of Manifested models to search for potential solutions. Often, the means by which to begin to fully evaluate a model requires further assistance from computational tools. But this is not always the case. For example, in order to simply validate assumptions relating to a nonmathematical model (framework), panel evaluations or individual case testing may be required. Responses gathered through this process will eventually set individuals up for the last stage of the structuring process, and more than likely a return to prior stages. Even some numerical tasks have manual tricks that allow for immediate solutions.

In other instances, however, we need some real computational assistance to provide some reliable heavy lifting that individuals (including yourself) would not want to take on. Nor should you. For example, in the case of network analysis, appropriate computational tools are required to assess which components are most pertinent. This isn't a manual examination task. The same holds for a wide variety of optimization tasks that might not be easily solved (well) by simple manual tactics.

In the case of math-driven proof-of-concept structures, making your models accessible to the assistance of tools involves coding the equations you have developed in a manner recognizable by a tool. Entering simple mathematical relationships into a spreadsheet is fortunately fairly straightforward. It's vastly more challenging to come up with the mathematical forms in the first place than it is to key them into the spreadsheet environment. This makes Excel a convenient resource for such work – a convenience made all the more salient since the Excel environment is not only outstandingly flexible with regard to documentation, but also provides ready access to a host of solution search procedures. Although several tools exist, Solver (this chapter) and RISK Optimizer (Chapter 7) are common tools that we will examine for this task.

#### *6.1.1.6 Scrutinize (Search and Conduct Sensitivity Analysis)*

Details of derived recommendations require a final stage prior to becoming realistic prescriptions that can be acted upon. We need to Scrutinize these assembled solutions to make sure that they make practical sense and that we fully understand their systematic implications. Ask questions such as the following:

**What do the recommendations suggest regarding action and expected outcomes? What is the level of risk around these and other recommendations? What makes sense and what actions are practical? What needs fixing?**

In computational modeling scenarios, the employment of applications provides the mechanism by which to search for possible actionable solutions, as well as the development of sensitivity analysis around the performance of those and alternate solutions. This provides an ideal context to scrutinize the end product with regards to practicality, and the sensitivity of results with regard to expectations of model dynamics. If anything seems counterintuitive, or if the solution is untenable, a return to earlier stages is required. Are the model specifications correct? Can additional improvements be pursued? How sensitive are solutions to the present assumptions?

The same would or should hold for nonmathematical (or at least less mathematical) examinations. If the case testing of a proposed framework yields results that don't match expectations or are impossible to interpret, failings exist in either the framework or the understanding of the context. A return to earlier stages of development, in such instances, would be mandated.

### **6.1.2 Using the OUtCoMES Cycle in A3 Frameworks**

Although the original design of OUtCoMES emerged from a desire to help people translate real-world decision-making scenarios into model forms for computational assistance, the process also applies more broadly. As we've now explained, there are detailed aspects of this cycle that both reflect and enrich established Plan-Do-Act-Check (PDCA) processes. By taking time in developing each of these details, and later each of the steps in holistic consideration, and by encouraging returns to prior steps for modifications to assumptions and priorities, OUtCoMES reinforces sense making and increases the odds of getting to the right questions as well as decent answers to them.

This virtue holds for nonmathematical modeling applications as well as those that are well suited to computational assistance. Because of this generalizability, it has become useful to think about the elements of the OUtCoMES Cycle as they relate to other popular frameworks for sense making such as A3. Similar to the PDCA framework, A3 thinking emerged out of the evolution of manufacturing practice. Like the OUtCoMES Cycle, it proposes the benefit of detailed structure on problem-solving tasks, and holistic views of context. The term "A3" in fact is drawn from the notion of getting all of the critical details onto a single A3-sized piece of paper, hence being judicious about the selection of critical content.

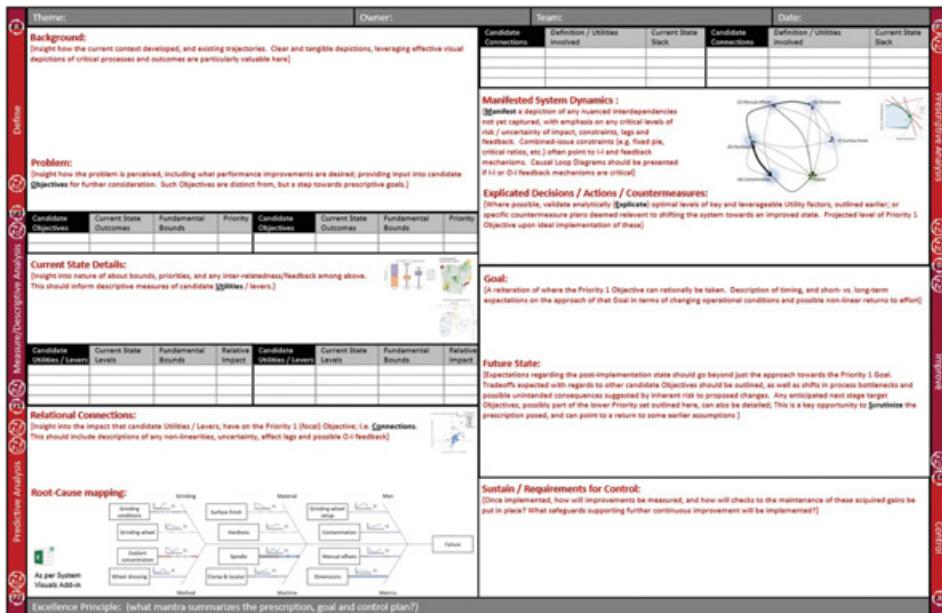


Figure 6.4 The OUtCoMES Cycle applied to a systems-oriented A3 framework

Unfortunately, implementations of A3 tend to lack emphasis on some critical system elements. In particular there is a lack of consistent guidance on details regarding Connections – details such as whether they are highly constraining, what level of risk is associated with them, whether impacts are lagged, and do feedback mechanisms exist. There is also a notable lack of support for iteration within many A3 implementations. For many the task of developing an A3 appears to be a strictly linear process (which can lead to rather unfortunate design choices).

Because of these issues, the OUtCoMES Cycle has recently been incorporated in A3 template designs (as in Figure 6.4 and the Chp6\_SystemsA3 PPT template). Such a merger of structure, and emphasis on iteration in the process of sense making and problem development, capitalizes on the best aspect of both A3 and the OUtCoMES Cycle. It also emphasizes connectivity to associated documents, such as data sets and analytical models, for helping to rationalize assumptions.

## 6.2 Developing Solutions from Structures

You've now rationalized your decision-making task – given it some structure. Great job! Now for the bad news: It turns out that actually finding practical solutions in the representative structures of the real world that you've spent

all of this time and energy detailing will often require a fair amount of additional time and energy.

Hold on, it's actually not that bad. On occasion solutions are pretty straightforward, and prescriptions simple. Often, however, solutions are far harder to identify. And sometimes there is little hope of guaranteeing that the solution you arrive at is in fact the best, either because you simply do not have the time to search more comprehensively, or, as we'll discuss in Chapter 7, there is risk around every solution we encounter.

Uncertainty in our system elements and relationships aside, for the time being we can draw an analogy between the search for solutions and the completion of a maze. Mazes are fairly well-defined problem structures. You can see the whole maze at once, and you know what you're looking for – you just don't know what the specific set of moves are that will get you the solution the fastest. Some mazes take longer to solve. Some mazes are so large that you might not be able to find a way through them given the time available (we're talking massively complex mazes here). But there are also better and worse approaches to solving mazes. If time is limited, those better approaches give you better odds at reaching the end of those mazes.

Consider two approaches to solving a maze, as demonstrated in the Chp6\_CellMazeRun workbook. In this visual example I've built a maze by coloring the cells in a worksheet to represent a start point, an end point, possible paths, and barriers to movement (walls). Changing the color of cells according to the consistent use of this color scheme, we can make the maze smaller or less divergent (simpler), or larger and full of more circuitous dead ends (more complex). We don't always have much choice in the complexity of our task, however. What we do have some choice of is the method used to find a solution.

I've posed two options for us. They are both somewhat weak, but one is appreciably weaker than the other. I've called it dumb, but that's probably too harsh, since it will after all (eventually) find its mark. It basically picks a random direction to travel down until it hits a wall. If there is another direction to go in, apart from reverse, it will attempt that direction. If it hits a dead end, it closes off that path, although because it selects new directions randomly it doesn't always get to those dead ends. The savvier, but still simplistic, approach also proceeds until it hits walls. It just changes directions a bit more deliberately: always clockwise; down, left, up, right (Figure 6.5). It finds dead ends faster and moves on faster. That one rule regarding changes in search direction makes a big difference. On a Surface Pro bought in 2014, running Windows 10, with 4 GB RAM, this approach took about one second to complete the default maze, including time

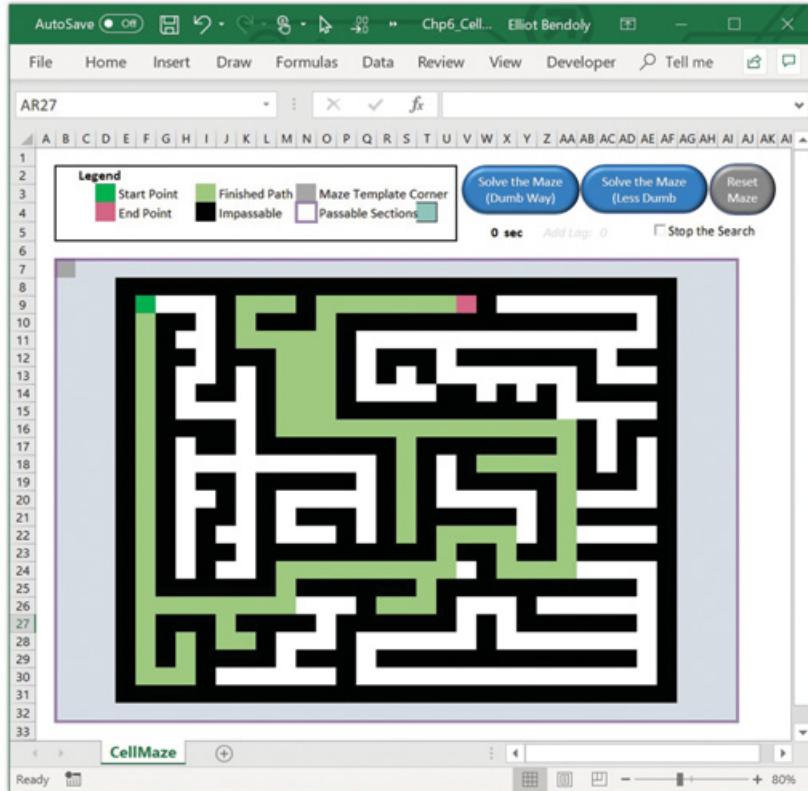


Figure 6.5 Approaches to solving a simple maze

for graphic updates. The less savvy approach took ten times longer on average.

There's something else worth noting, however. The faster approach is covering less territory. In this case that's just fine, since there is only one acceptable end point. But what if there were several end points? And what if the measure was not how fast you get there but whether you find the *closest* end point (solution)? Try it out. Add some additional end points here and there. When we search for solutions, we often face trade-offs between comprehensiveness and speed. We don't want to miss great solutions, but we also don't always have the luxury to wait forever.

Heuristics, or simple rules and approaches designed to help us solve problems, cover a wide range of approaches, tactics, and tricks that pose alternatives to comprehensive searches. Because they are not comprehensive examinations, they have their own caveats and risks specific to the problems they are applied to. But they can also get a decent (if not the best) job done. It's worth familiarizing ourselves with some of these approaches

before we consider a computational one that is a bit more sophisticated (albeit charged with its own limitations).

### 6.2.1 The Recognition Heuristic

One of the simplest examples of an effective, fast, and frugal heuristic is the *recognition heuristic*. Rather than assuming that people act as unboundedly rational individuals (a strangely common assumption in academic literature to date), the recognition heuristic assumes not only that bounds on awareness exist, but also that such bounds are important factors in determining the strength of specific decision options. In fact the foundation of this heuristic relies on at least some level of limited information availability, and the underlying sources of such limitations, to develop good solutions to problems.

As an example, consider a study performed by Bernhard Borges and his colleagues, “Can Ignorance Beat the Stock Market?,” which is detailed in *Simple Heuristics That Make Us Smart* (Gigerenzer, Todd, and the ABC Research Group, 1999). Borges et al. examined the ability of people on the street to develop high-performing stock portfolios based on their personal, albeit limited, exposure to corporate information.

**Objective:** A consistently high-return portfolio (relative to market average).

**Utility Variables:** Set of publicly traded firms to be bought or shorted.

**Connections (Constraints):** In the case of the heuristic, the ability of individuals to recognize companies in domestic and foreign markets (which obviously differs by expertise).

Certain companies in the United States are widely recognized both by nondomestic career financial workers and lay people alike. Likewise, in the United States we recognize only a relative handful of foreign companies by name. Does that signify anything? The assumption of the recognition heuristic is: Yes. This recognition probably indicates the ability of a brand name to penetrate markets abroad and domestically, as well as the resiliency of the reputation it has developed within those markets. In most cases we’d assume this resilient reputation to be positive – companies with negative reputations don’t last very long.

But can something as simple as name recognition by an average individual predict performance even close to the level of highly sophisticated financial techniques? More sophisticated approaches, aside from being vastly more complex, are often proprietary in nature (not so publicly accessible); however, more alarmingly, they often don’t do that well. In fact, the history of major US investment management and mutual companies suggests that the favored selections of many highly

experienced financial professionals perform worse than the market as a whole. This suggests that sophistication and experience might bias professionals toward misleading decision-making approaches. Perhaps less sophistication might go hand in hand with approaches that avoid such biases, at least in some cases.

To compare the recognition heuristic to more sophisticated and expert solutions, a research team asked average people and financial experts in the United States to specify from a list those companies in Germany that they recognized. They did the same thing in Germany with a list of US companies. How well did the ten most-recognized US stocks do, chosen on the basis of German recognition?

Those top-ten German choices (again, based on the recognition heuristic) beat the Dow 30 by about 10 percent, along with a number of funds supposedly based on sophisticated intelligence. They did much better than a random portfolio method. Granted, sophisticated methods of field experts showed better performance against random portfolio assembly as well. However, it costs to hire such professionals – one might question the value of the incremental gain against the recognition heuristic here. Even better was the top-ten portfolio based on US recognition of German firms. The average Joe was able to outperform the Dax 30 by 23 percent over the six-month test period, and similarly over subsequent periods studied.

In subsequent studies this phenomenon has been retested with mixed results, largely by those financial professionals who have a clear interest in suggesting that simple techniques have limitations relative to proprietary expertise. This may be true in some cases, but these and other related results certainly cast critical doubt on what it means to be a financial *expert*. This is not to suggest that there don't exist, or won't exist in the future, successfully sophisticated selection procedures developed to consistently beat lay recognition; however, the best approach perhaps would be to incorporate just a few simplistic rules as checks to those sophisticated models.

### **6.2.2 Nearest Next: A Sequencing Heuristic**

Another well-established fast and frugal heuristic can be taken from the history of the shipping and transportation industry. Before computing power was abundant, shippers relied on professional planners to use their own modes of judgment to develop shipping routes and schedules that economized on cost but still met the service levels their clients expected. The types of problems about which these planners had to make decisions were highly complex.

Table 6.4 *Example 1: Documentation of Objective, Utilities, and Connections*

Objective Specification		Iteration: 3		Prior Stage: NA	
Candidate	Definition	Current State	Bounds	Priority	Notes
Route length	Tot distance btx	NA [units]	(0–?)	1	Straight round trip
Utility Specification		Iteration: 3		Prior Stage: Objectives	
Candidate	Definition	Current State	Bounds	Impact	Notes
Order of trees	Sequence of route	Arbitrary seq.	[1–10]	100	Whole numbers
Connection Specification		Iteration: 3		Prior Stage: Utilities	
Candidate	Utilities involved / Definition		Abs. Slack	Rel-Slk	Notes
Sequence law	Sequence has 10 unique values		0	High	Each tree once

The challenge of sequencing visits to spatially distributed locations in fact applies to a broad variety of contexts. Imagine, for example, the task of an arborist charged with assessing the health of a set of trees spread across a large parcel of land. To keep things manageable, let's say it's just ten trees and the arborist can start wherever she or he likes, helicoptering into that location, walking between assessments, and returning to that spot at the end. Assessment takes time, so the arborist wants a route that minimizes total transit (i.e., requires the least amount of walking). After a few attempts we may have rationalized our decision-making context into the details of Table 6.4.

Although we might not know the upper limit to the objective, we know we can't go below 0. We could probably estimate a more meaningful lower (and upper) bound by multiplying the shortest (or longest) between-tree distance by 10 in this case. But much more critical is pinning down the form of the actual objective function, based on the utility variables outlined.

As we work to manifest a mathematical model relating our Utilities to the Objective, as noted earlier, we'll find the use of notation helpful. Let's allow X and Y to be the fixed (unsequenced) arrays of relative X-Y coordinates approximated using GPS. We can denote the coordinates for the seventh tree in this set, for example, as X(7) and Y(7). These are just facts in our problem. Let's also describe an array of Utility variables, representing the ordered sequence of a possible route as "Tour." If we choose to visit the seventh tree in our set third in the route sequence, then Tour(3) = 7. This

allows the following Objective form to manifest, with  $N$  referring generically to the total number of sites, and  $i$  as a placeholder for each in the set:

$$= \left( \sum_{i=1}^{N-1} \sqrt{[X(\text{Tour}(i)) - X(\text{Tour}(i+1))]^2 + [Y(\text{Tour}(i)) - Y(\text{Tour}(i+1))]^2} \right) \\ + \sqrt{[X(\text{Tour}(N)) - X(\text{Tour}(1))]^2 + [Y(\text{Tour}(N)) - Y(\text{Tour}(1))]^2}$$

Explication of a solution is then just a matter of modifying the whole number values in the Tour array, keeping each one unique and between 1 and  $N$  (1 and 10 in this case). But how many ways can we sequence ten stops? As it turns out, quite a few ( $10! * 9! * 8! * 7! * 6! * 5! * 4! * 3! * 2! * 1 = 3,628,800$ ). With a little help we could in fact search all of these, finding both the best and the worst routes overall. The CompSearch\_TSP function of the Blackbelt Ribbon add-in can tackle this for us in less than half a minute (running on the same machine that ran our earlier mazes). The formulation used in cell G13 of the Chp6\_ArboristTSP workbook, before it was pasted as values, was the following:

= CompSearch\_TSP(D\$4:E13)

Half a minute isn't too bad. The arborist could afford it. But there might be a decent shortcut as well. The *nearest next heuristic* provides one. Its rule is "Wherever you are, go to the closest site next" and then repeat this until all sites are visited and return again to the start. A version of this is built into the Blackbelt Ribbon as well, with the results of both of these compared in the Chp6\_ArboristTSP workbook and Figure 6.6. The formulation used in cell I13 of the Arborist sheet, prior to fixed pasting, was:

= NearestNext\_TSP(D\$4:E13)

Figure 6.6 Comprehensive vs. heuristically developed TSP solutions

For completion, I've also created a FurthestNext function, which changes the rule to "go to the furthest site next" (in case you wanted to approximate how bad a route could get). An additional optional parameter in these three functions of a destination cell (in quotes, so as to not retrigger the functions), and the specification of TRUE or 1 as a third parameter (e.g., =NearestNext\_TSP(D\$4:E13,"V1",1), creates a textbox that can be clicked on to extract all 10 (in this case) derived sequences into spreadsheet cells starting at the location specified. Once there, any one of these 10 are available for visual examination. Since CompSearch looks at many more possibilities for a given number of sites, I've limited this sort of extraction capability to only become available if the comprehensive search involves sequencing of six sites or less (720 combinations extracted). No need to crash things here.

Figure 6.7 shows an example visually comparing the heuristic solution to the absolute worst and absolute best options.

Note that the heuristic doesn't quite hit the best solution, but it's far closer to the best solution than it is to the worst one. I also haven't designated any start points here, since these are round trips – starting at any point and following the path leads to the same outcome. There might be other considerations, such as the cost of starting at any one of these sites, but once selected the value of the comprehensively best solution, and the nearest-next solution, still holds, relative to others.

Critically, the number of examinations considered to determine the nearest-next solution is only  $10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$  (55), yielding 10 solutions. These can also translate into at most  $10 \times 10$  (100) solutions since each round trip can begin at any initial point, albeit some of the nearest-next solutions derived from given start points may end up being identical (hence <100). More generally, the work of *nearest next* involves  $N*(N + 1)/2$  "examinations" (where  $N$  is the number of sites), and can yield up to  $N^2$  "solutions" as a result. In stark contrast a comprehensive search for the absolute best needs  $N!$  (i.e.,  $N$  factorial) examinations (assuming no other ancillary heuristics are applied). So, if we needed to find a decent solution for  $N = 15$  sites, the nearest-next heuristic would yield solutions after 120 examinations, whereas a complete review of all possible solutions would require considering 1.3 trillion examinations.

And what if  $N = 100$  sites? It's hard to find a physical example comparable to the size of 100 factorial, something like a 9 followed by 157 zeros, or almost one hundred thousand quinquagintillion (say that five times fast), but there are some. For example, the total volume of the entire (not just the observable) universe has been estimated at around this level in cubic meters (Guth 1998), or about that same number of car trunks. So, having to examine  $100!$  sequences would be like opening and comparing the contents of all the car

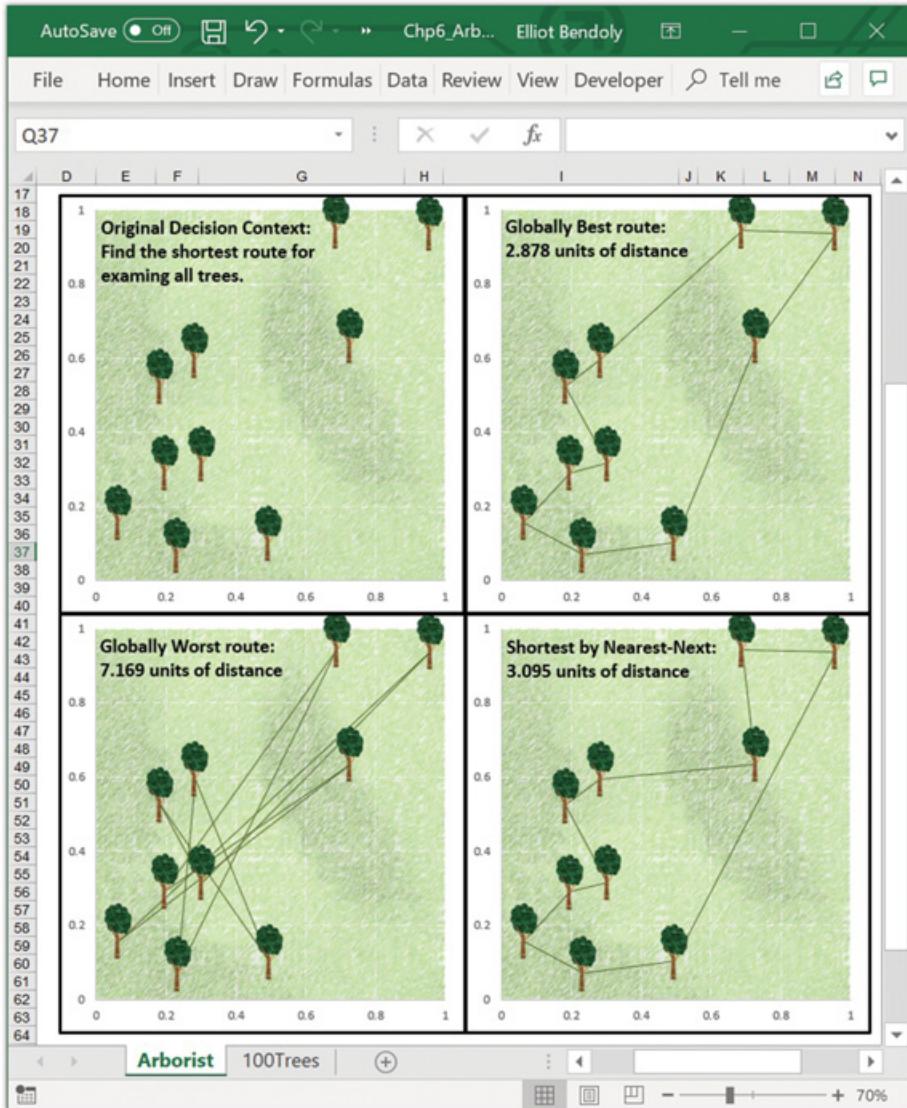


Figure 6.7 Globally best, globally worst, and heuristically derived solutions

trunks you could fit into the entire universe. No arborist has that much time on their hands. In such contexts comprehensiveness is untenable. Heuristic derivations on the other hand can prove highly effective (relatively speaking), even if only as a great start for more-involved computational search follow-ups. We'll revisit this in Chapter 7.

Routing applications today tend to use a mixture of simple heuristics and complex, large-scale analysis to arrive at highly effective solutions. We continue to see improvements made in these applications with increases in

computing power (i.e., speed and memory), hence additional elements of reality can be practically included without an abandonment of fundamentally crucial aspects. Some of the most recent additions to routing applications now attempt to account for the human aspects of shipping, such as individual psychology, relationships between drivers and clients, and morale. Fast and frugal heuristics continue to have their place in these applications. Furthermore, these heuristics continue to provide benefit for managers who need to make decisions on the fly (e.g., immediate rerouting responses to real-time road closures).

### ***6.2.3 MinSlack and SPT: Two Project-Management Heuristics***

Resources, such as workers and machines, often constrain the processes managed by an organization. These hurdles are common in project management, regardless of industry. The process of deciding which activities will be given access to potentially limited personnel first is a critical one in these settings, and these decisions need to be made quickly. For this reason, a large number of fast and frugal heuristics have been developed by managers in an attempt to make these decisions simple.

Two common fast and frugal heuristics are MinSlack and Shortest Processing Time (SPT). MinSlack assigns resources and starts comparing activities based on which ones will be most costly or problematic to project completion times, if they are otherwise delayed. The idea here is that limiting activities should be addressed as soon as possible so they don't delay subsequent activities.

SPT assigns resources and starts comparing activities based on which ones can be completed quickest. If you can get these activities done quickly, you can free up resources and assign them elsewhere while allowing the start of other activities that might not need those resources.

### ***6.2.4 The Punchline: Relevance to DSS Designs***

I could keep giving examples of heuristics that have been used in practice to make quick decisions in complex decision-making settings, but that's not really the goal here. Some people may be of the mindset that they don't want a simple approach; they want the best solution. Aside from the perception that a single, best solution to all management settings exists, this mindset seems to assume that simple approaches cannot be as effective as ones that require several tools used together to deliver ideal solutions. But that's just not universally true.

Each of the heuristics I've discussed represents a structured set of rules that somebody came up with because the rules made sense in a specific

setting. For people who develop decision support systems, there are at least three reasons to provide structured rules that are backed by logical explanations.

- 1) By definition, the rules tend to be both easy to apply and informative.
- 2) The rules can offer a great starting place for more sophisticated support.
- 3) Perhaps most important, the rules can offer a performance benchmark from which to gauge the efficacy of more sophisticated solutions. This, in turn, helps to assure users of the support the system is providing to them. The effectiveness of decision support system designs comes largely from selling the effectiveness of the support they are designed to provide.

For DSS designers, the question of the practical application of heuristics comes down to how much is gained and lost through the use of such techniques, and, perhaps more fundamentally, whether these techniques can be automated for integration with other techniques useful in the DSS design.

### **6.3 Optimality Searches under Moderate Complexity**

When we don't want to think up a solution approach from scratch, we also have some nicely packaged and readily available tools to draw on. Excel gives us Solver, a great tool that helps us determine what specific levels of our Utility variables should be used to best attain our Objectives, in a manner that helps us account for the various Connections among things (notably those that constrain decision making). Generally, Solver can be accessed under the Data tab in the Analysis section (Figure 6.8). If you do not find Solver in your Excel Data tab, it means that either Solver was not selected for installation at the time your copy of Excel was installed, or it is currently not

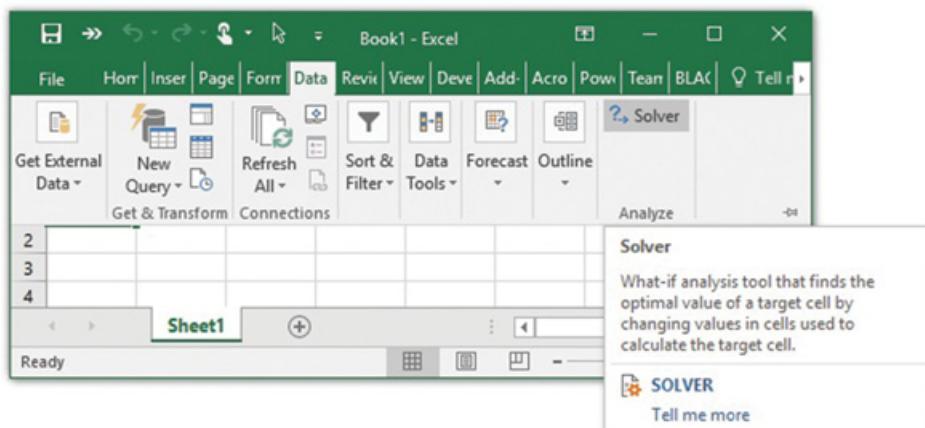


Figure 6.8 Landscapes of linear versus nonlinear connections

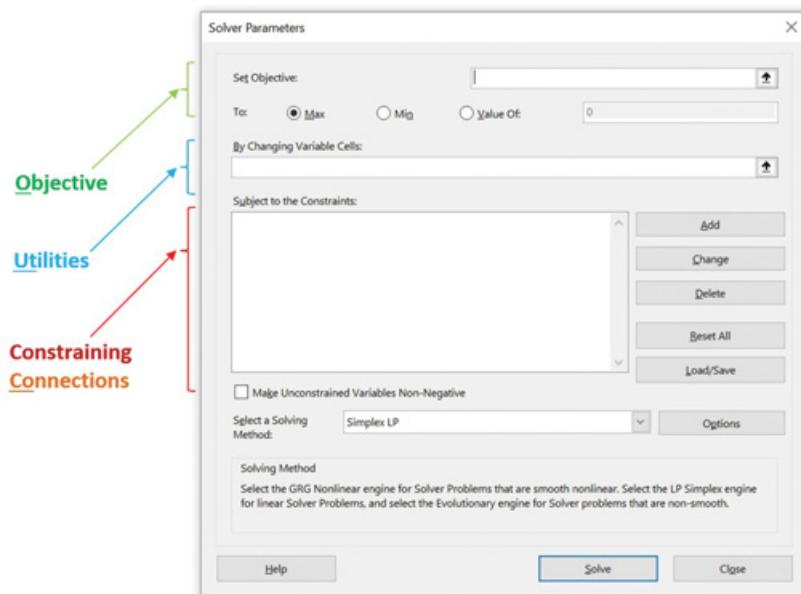


Figure 6.9 Locating Solver in the Excel menu interface

activated. To activate Solver, as with the Blackbelt Ribbon, click Options>Add-Ins. Select Excel Add-Ins in the Manage drop-down menu and then click Go.

The general structure of Solver fits perfectly with the first three stages of the OUtCoMES Cycle: objectives, utility variables, and connections (constraints) (Figure 6.9). Solver is designed to provide the best solutions possible, based on the information we provide. It has its limits (it breaks down with extremely complex or large problems), but it does a nice job for smaller problems that still present challenges to decision makers.

Solution search engines, like those embedded in Solver, commonly use what are called “hill climbing” approaches to seeking out optimal solutions. In reality, this is just another fast and frugal (greedy) heuristic. Starting with an initial solution (aka an initial set of utility variable values), a hill climbing approach considers incremental changes in each of these variables and pursues changes that yield the greatest return to the objective.

This approach is a lot like following the most appealing path along the perimeter of the example of visualized constraints, near the end of Chapter 5. When no further change in the direction you’re following yields superior and feasible improvements to the objective (due to the presence of constraints, for example), alternate directions are pursued or the search ends. Solver has several search engines available to it, but both the LP (linear programming) and GRG Nonlinear engines in Solver operate more or less in this fashion.

Like any noncomprehensive approach, it can fail us when things get too complicated. But for simpler scenarios, it's fast and can yield exactly what we're looking for. Ultimately, its usefulness boils down to whether we are thoughtful enough, and lucky enough, to structure our models in mathematically efficient ways.

Let's look into some examples of what this approach can do for us.

### **6.3.1 Example #1: Columbus Professional Training (CPT)**

Columbus has been an up-and-coming hot spot for young professionals over the past two decades. As more twenty-somethings migrate to Columbus, they find both opportunities and challenges in landing the dream jobs available. Extensive market research with the major employers in the area indicates that nonlocal job seekers often lack depth in one of two key job skills: substance (statistical analytics, visualization, coding, and so on) or style (communication, poise, etiquette, and so on). Dorian McAnderstein, a recent MBA graduate, jumped at the opportunity to share his expertise in both of these areas. To do so, Dorian founded a professional training facility that caters to the needs of these individuals.

He has decided to charge \$3,500 for each substance-lacking student (termed "Beatnik") and \$2,800 for each style-suppressed student (termed "Geek"). The plan is to hold a professional boot camp comprised of both combined sessions as well as high-touch one-on-one training sessions. For these one-on-ones Dorian estimates that beatnik students will need 14 hours of training in analytics (substance) and seven hours of training in communication and etiquette. In contrast, geeks will need only six hours of one-on-one training in analysis but 11 hours of training in communication and etiquette.

Dorian teaches the style courses, and he can work up to 114 hours per month. Dorian's partner handles the substance sessions and can work up to 107 hours per month. In addition, Dorian does not feel that the students benefit from class sizes smaller than nine. The question is: What mix of geek and beatnik students should Dorian admit?

For the benefit of discussion, let's follow the documentation of Table 6.5 and assume Dorian's altruism is superseded by an objective to maximize revenue. The formulaic structure of the revenue function might manifest as follows:

$$= \#Beatniks * \$3500 + \#Geeks * \$2800$$

Or maybe,

$$= \#TotalEnrolled * [(1 - \%Geeks) * \$3500 + \%Geeks * \$2800]$$

Table 6.5 Example 2: Documentation of Objective, Utilities, and Connections

Objective Specification		Iteration: 1			Prior Stage: NA	
Candidate	Definition	Current State	Bounds	Priority	Notes	
Max Revenue	Tot \$ Enrolled	0 [\$]	[0-?]	1	Weighted sum	
Utility Specification		Iteration: 1			Prior Stage: Objectives	
Candidate	Definition	Current State	Bounds	Impact	Notes	
# Geeks	Tot geeks enrolled	0	[0-?]	2800	Whole numbers	
# Beatniks	Tot beatniks enr.	0	[0-?]	3500	Whole numbers	
# Total enrolls	#Geeks + #Beatniks	0	[9-?]	Mixed	Whole numbers	
% Geeks	Geeks / Total enr.	0	[0-100 %]	?	Decimals	
Connection Specification		Iteration: 1			Prior Stage: Utilities	
Candidate	Utilities involved / Definition		Abs. Slack	Rel-Slk	Notes	
Min Size	Total enrolls > 9		0	<Neg>	Currently infeasible	
Max Hours	Weighted Sums < Max avail.s		>	High		

Each of these manifestations of the objective function relies on specific operationalizations of utility variables. Knowing either the first two values, or the latter two, tells the story. But is the use of one pair better than the other? In one case we have a linear combination of variables weighted by constants. In another, variables are multiplied by one another, making for a nonlinear function (Figure 6.10 compares the two).

In a single relationship like that in Figure 6.10 it may seem like a subtle difference. But even a few nonlinearities in combination can generate a landscape that can throw off a search. Solver is outstanding when it comes to solving problems that have only linear connections between utilities and an objective, and between utilities and each other. It's less reliable when this isn't the case. We'll talk more about these complications in Chapter 7, but for the purpose of optimization the basic rule for structuring the mathematical forms of story problems is to "Keep it simple" (often translated into "Keep it linear").

Regarding other connections, such as those constraining the decisions to be made, we'll try to keep things simple as well. We know that Dorian and his partner have a limited number of hours to devote to students and that student training requires time. The total amount of time required for substantive training ( $\#Beatniks \times 14\text{hrs} + \#Geeks \times 6\text{hrs}$ ) must be less than or equal to the amount of time available from Dorian's associate (107 hours). A similar constraint connects the enrollment numbers to the maximum hours

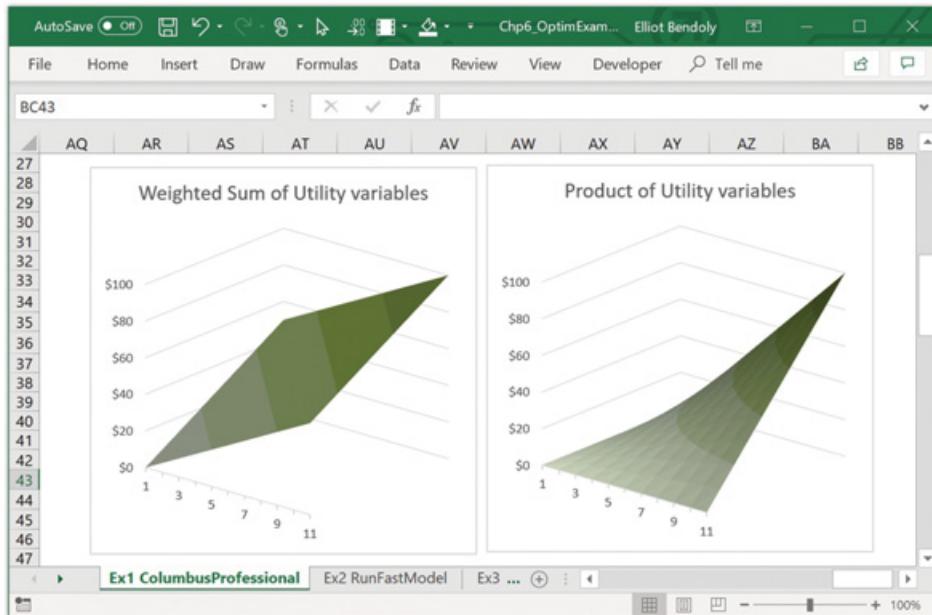


Figure 6.10 Objective, decision, and constraint specification fields in Solver

available by Dorian (114 hours). A third constraint ensures that total enrollment (#Beatniks + #Geeks), relevant for joint session dynamics, is not fewer than nine.

In the Chp6\_OptimExamples workbook, I've put all of the information (and the mathematical structure just outlined) from that story problem into some meaningful order on a spreadsheet. I've also added some graphics, but more important, I've provided sufficient annotation to let you know exactly what each of the numbers on this page refers to. As per Chapter 2 I've also labeled relevant cells and cell ranges with names (e.g., TotRev, NofEachStudent) for easier reference. Figure 6.11 provides a view of both the preliminary concept map as well as the manifested model.

If we didn't have any additional decision support mechanisms to help us at this point, we might start by trying out various values of our utility variables in an attempt to meet our objectives without violating any of our constraints:

If I enter three geeks and three beatniks, I get total revenue of \$27 K.

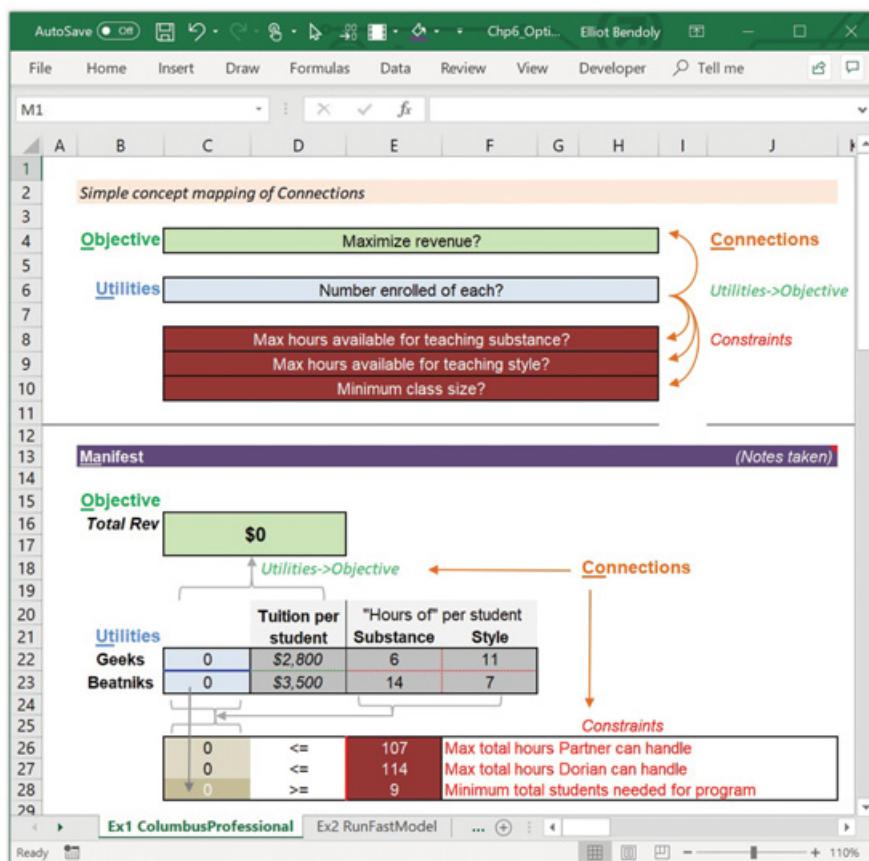


Figure 6.11 Structuring the enrollment decision for CPT

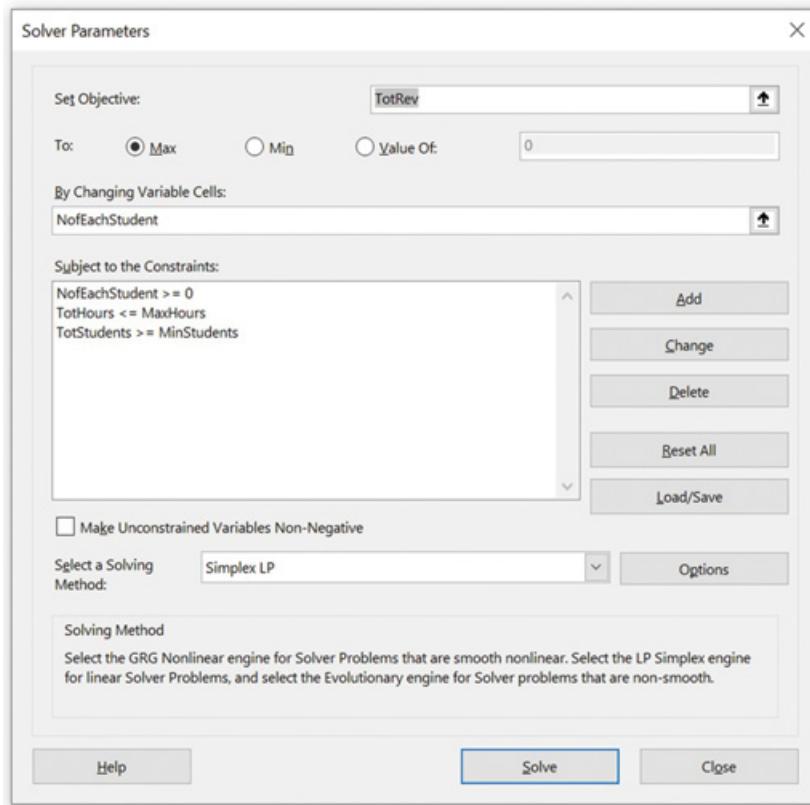


Figure 6.12 Specifying the problem structure for CPT in Solver

If I enter five geeks and three beatniks, I get total revenue of \$45 K.

If I enter seven geeks and seven beatniks, I get total revenue of \$63 K, but this requires more hours than either instructor has available.

Obviously, a manual search for a good solution, even in this case, could take some time. Solver makes it easy for us. Figure 6.12 illustrates how the fields in the Solver interface might be populated to accommodate this problem.

Because all of the information is entered in some intelligent fashion into the spreadsheet with utilities linked to both the objective and connections specifying constraints, we're already set up for Solver to help us. Note how the objective, utility variables, and connecting constraints correspond among the spreadsheet and fields in Solver. The intelligent naming of cells allows for a meaningful representation of the problem in the Solver interface, facilitating error checking. Solver is smart enough to interpret “NofEachStudent $\geq 0$ ” as “all the cells in the NofEachStudent range must be greater than 0” (no antimatter students, please).

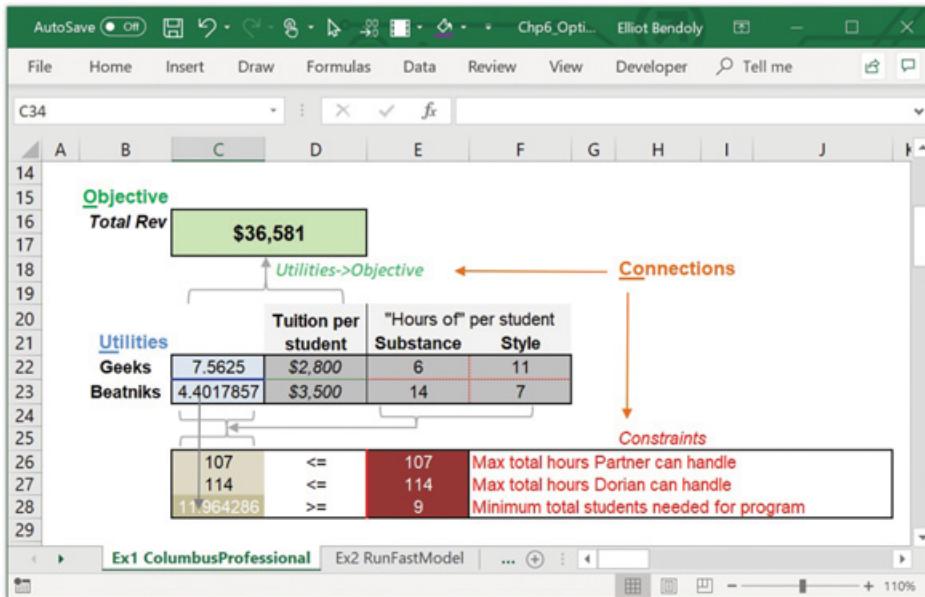


Figure 6.13 Example solution to one specification of constraints for CPT

In this case the problem presented to Solver is the simplest kind. Once again, all connections between the utilities are linear ones: utilities are multiplied by constants at various points, but nothing more complex than the addition of these products takes place (no  $1/x$ , no  $\ln(x)$ , etc.). Consequently, we can get Solver to find a solution using its most fast and frugal approach to these problems: the Simplex LP (linear programming) method – deferring a consideration of other approaches to Chapter 7.

When these selections are made, hitting Solve should provide the solution presented in Figure 6.13.

Let's think about this solution. Are 7.56 students or 4.40 students reasonable options in this problem? If partial students are not acceptable, or at least a liability, we could try rounding these numbers this way and that until we find a feasible solution. But that's just more manual work – not desirable, and it can easily go wrong. Try it and see where you end up, watching those constraining connections. It would be much smarter to just add another constraint limiting our solution to integer-only options, as shown in Figure 6.14.

At this point we also need to provide a cautionary note on using Solver. Some of its default settings can be a bit odd. For example, the default setting for Solver is to “Ignore Integer Constraints.” Yes, really. This means that variables that, in fact, should be whole numbers, as specified by constraints, might take on non–whole-number values. There's not much rhyme or reason

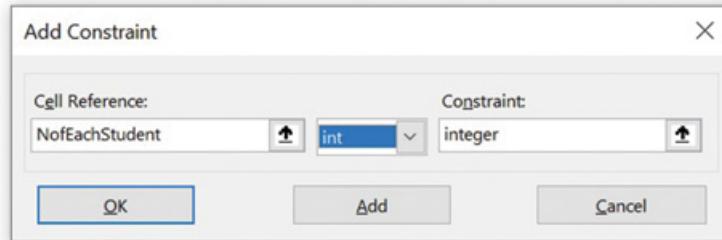


Figure 6.14 Adding the integer constraint on the utility variables

for why this default setting exists, but if you do want to have integer constraints apply in your work make sure to go to Options in the Excel userform pop-up and unclick the “Ignore Integer Constraints” option. Setting Integer Optimality (%) to 0 further enforces interest in any integer constraints you list (see Figure 6.15).

A more reasonable solution for Columbus Professional Training then emerges from this setup. To maximize revenues subject to the constraints listed, enrollment should include six geeks and five beatniks. That’s the best solution that doesn’t violate any of the rules. Is it where you got by rounding the noninteger solution? If not, it’s just another example of the value of DSS. This solution also leaves some extra time for both Dorian and his partner (aka “slack”). However, that’s often the nature of real-world solutions. We’ll talk more on the implications of slack in solutions later on.

### 6.3.2 Example #2: RunFast, Inc.

Now let’s look at a similar kind of search in a very different context. RunFast manufactures running shoes used to supply specialty-running retailers throughout the Southwest. RunFast currently produces three types of running shoes: basic, track, and trail. RunFast wants to determine how many pairs of each type to produce next month. The process of assembling each pair is highly automated, followed by manual inspection of each individual pair to ensure it meets the high quality standards set by RunFast. Each component of the shoe (sole, laces, air wick, insole, and so on) is placed onto a belt that feeds through an assembly machine called the Blazer620. After each shoe is assembled, it is passed on to manual inspection, where its quality is assessed.

Each of the three types of shoes must be processed on the Blazer620, and each pair must be inspected. The times required for these two processes for each shoe type are specified in Table 6.6. There are 90 hours of Blazer620 time available for assembly of these shoes next month. Not all of the hours must be used, but no more than 90 can be used. The inspection team has 125 hours available for testing these products next month.

Table 6.6 Requirements specifications for a production environment

<b>Stage</b>	<b>Shoe Type</b>		
	<b>Basic</b> (minutes/pair)	<b>Track</b> (minutes/pair)	<b>Trail</b> (minutes/pair)
Blazer620	6.75	4.80	6.00
Inspection Area	15.30	17.40	14.25

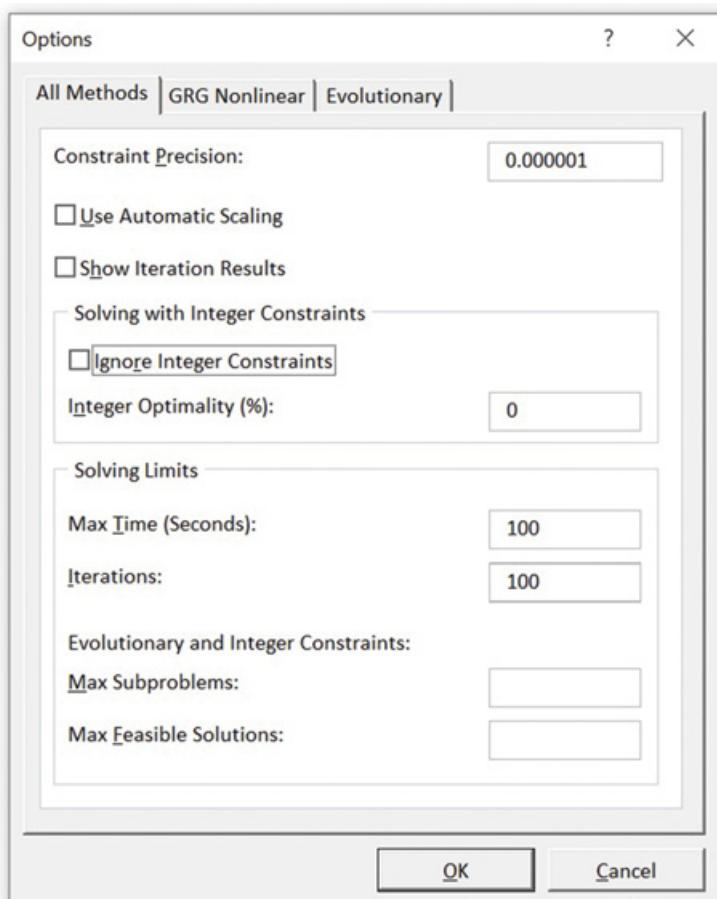


Figure 6.15 Enforcing compliance to integer constraints in Solver

The accounting department at RunFast has provided \$6.38 as the production cost per hour for using the Blazer620. This cost must be considered in the decision process. Likewise, the direct labor cost for the inspection team is \$0.151 per minute. The revenue and overall profit that each pair generates and the cost of the materials used in each pair are specified in Table 6.7.

Table 6.7 *Cost and revenue details for a production environment*

	Shoe Type		
	Basic (\$/pair)	Track (\$/pair)	Trail (\$/pair)
Revenue	23.85	19.88	18.00
Material Cost	7.80	7.34	6.30
Direct Production Cost	3.03	3.13	2.79
Profit Margin	13.02	9.41	8.91

RunFast is known for its high-quality products and special testing of each pair of shoes. RunFast faces tremendous demand for its products and is confident it can sell every pair at the given prices. There is one catch: One of RunFast's largest customers has already placed an order for 30 pairs of the track shoes to be delivered next month. Further, the marketing department wants to ensure that RunFast maintains a full product line to stave off the competition encroaching on its territory, and the department has requested that at least one pair of trail shoes be produced for every ten basic pairs that are produced next month.

We can now formally outline the task confronting RunFast: Determine a production plan that specifies how many basic, trail, and track running shoes should be produced next month in order to maximize total monthly profit. That is, we'll assume that our objective is profit to be maximized, and that our utilities are the quantities of each shoe type to make.

What constraining connections do we face?

- 1) Don't use more than 90 hours of Blazer620 time.
- 2) Don't use more than 125 hours of inspection time.
- 3) Do produce at least 30 pairs of track shoes.
- 4) Do produce at least one pair of trail shoes for every ten basic pairs.

Figure 6.16 shows a manifested structure that we might use to outline our utilities and how they connect to the objective and each other per the constraints we're faced with.

Given the appropriate labeling of cells and cell ranges for reference purposes, Figure 6.17 shows how we'd convey that information to Solver.

Note that, as with the previous example, we would like an integer solution. Therefore, we need to make sure Solver is not going to ignore the integer constraints we've added (see Figure 6.15). We can also leverage the Simplex LP in this case. The solution that Solver gives us is shown in Figure 6.18.

**Utilities**

Pairs of...	Basic	Track	Tail
	0	0	0

**Objective**

Total Profit	Fixed profit per pair
\$0.00	\$13.02    \$9.41    \$8.91

**Connecting constraints**

	Time required per pair			TotalTime	MaxTime
Blazer620	6.75	4.80	6.00	0.00	$\leq 5400$ minutes
Inspection	15.30	17.40	14.25	0.00	$\leq 7500$ minutes
				0.00	$\geq 30$ Min number of track pairs
				0.00	$\geq 0.00$ x10's of basic pairs
				Make sure at least 1 trail pair per 10 basic pairs	

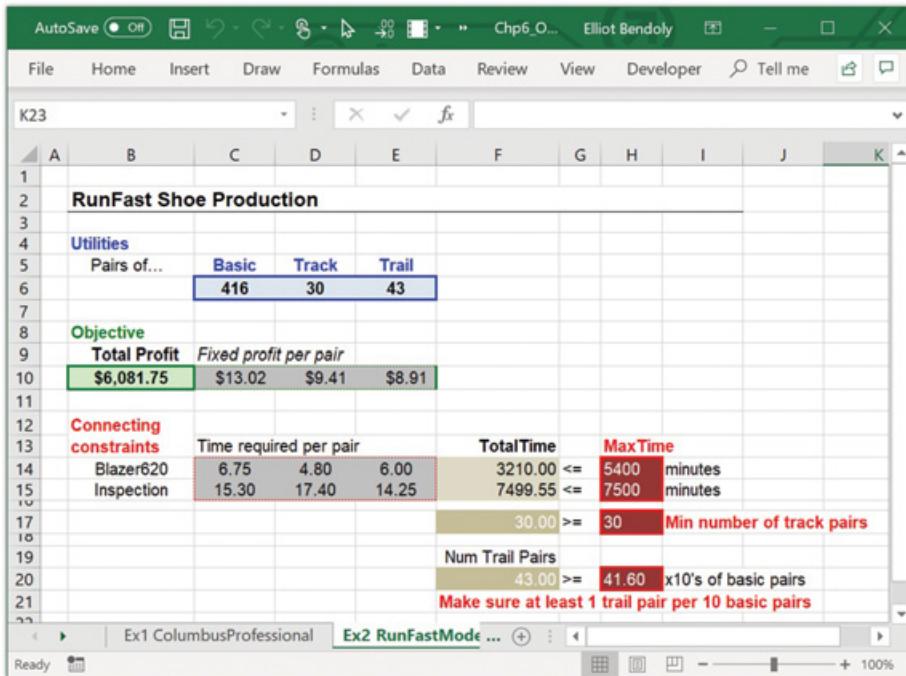


Figure 6.18 Solver's suggested optimal production solution based on specification

An additional point is worth mentioning regarding the visual worksheet layout of this and other examples we're discussing. If you are viewing these workbook files directly or are reading this on an electronic version of the text, you may have observed some consistency at this point in the colors used to demarcate objectives, utilities, constraining connections, numerical constants/facts, and so on. This is entirely intentional. Just as the naming of cells can help you and others make sense of the complex calculations you may be relying on, consistent color coding visually facilitates your own efforts toward building and sharing prescriptive models in spreadsheet contexts. It's another major and underappreciated plus regarding the use of these environments.

### 6.3.3 Example #3: Jumping Java

The Jumping Java coffee shop, located on a local college campus, is open 24 hours per day. Jumping Java employs mostly students who enjoy the flexibility of four-hour shifts (six per day). All shifts start at four-hour intervals that begin at midnight. The expected minimum number of employees required in each time interval in a typical week is given in Table 6.8.

Table 6.8 *Details for employee staffing example*

Day	12 am–4 am	4 am–8 am	8 am–12 pm	12 pm–4 pm	4 pm–8 pm	8 pm–12 am
Monday	4	7	16	5	10	8
Tuesday	3	6	17	7	14	7
Wednesday	6	5	18	14	9	8
Thursday	4	7	16	8	12	4
Friday	8	7	18	8	19	7
Saturday	9	6	16	22	8	15
Sunday	5	12	17	21	11	9

Note that in this example we are assuming that these staffing requirements are good estimates reflecting demand and that they are roughly stable week to week. This is a major simplification, but it's useful to sufficiently demonstrate the structure of the logic and overall nature of the model. In reality, each week may have different staffing requirements, and demand estimates can have substantial uncertainty around them.

In our simplified model, staff members can work either four- or eight-hour shifts. Employees choosing to work an eight-hour shift receive \$13.50 per hour. Those working only a four-hour shift receive \$12.75 per hour. The coffee shop also incurs an overhead cost of \$7.50 per person working either shift. Therefore, the total cost of having one person work for eight hours is  $(8 \times 13.50) + 7.50 = \$115.50$ . The total cost of having one person work one four-hour shift is  $(4 \times 12.75) + 7.50 = \$58.50$ . So the question is: How many employees should be working in each time period on each day to minimize total staffing costs (subject to minimum staffing requirements)?

In this particular problem, because there are two separate kinds of staffers (four-hour-shift and eight-hour-shift people), and because we are concerned with assigning those staffers to specific time slots, the description of the utilities might seem a little more complex than in the previous examples. It would help if we could design utility variables that would simplify the model regardless of whether we were using a manual search or Solver.

The most straightforward definition might at first seem to be simply deciding upon and keeping track of the number of people working in each of the time intervals for each day. Unfortunately, if we know only the total number of people working from midnight to 8 am, we don't know how many of these people are eight-hour employees and how many are four-hour employees. Even if we kept track of these workers separately, we would not know how many eight-hour shift workers were ending as opposed to beginning their shifts in that interval. Using that approach, we could easily get confused in our policy efforts.

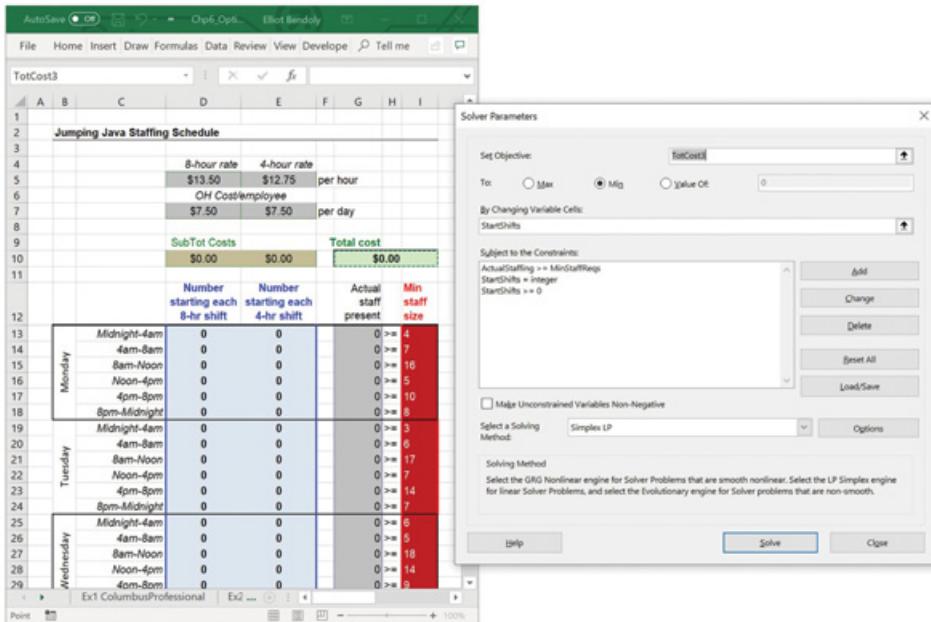


Figure 6.19 Spreadsheet construction and Solver setup for staffing problem

To clarify the schedule and avoid this problem, we define utility variables that tell us how many people of each type simply begin their shift in each time interval. With this information, we can compute total employment costs as well as determine the total number of people working in each of the 54 four-hour time intervals. Figure 6.19 shows how we might structure the problem in a spreadsheet and in Solver.

Provided we make sure Solver does not ignore integer constraints (Figure 6.14), Figure 6.20 furnishes an example of the kind of solution we might get. Consider trying to come to this solution through a manual trial-and-error approach!

Now, if you give this problem to Solver, you might not get exactly the same solution. The final objective function should be the same, but values of the utilities might not be the same. This isn't a mistake. There's actually a good reason for this kind of a discrepancy. What distinguished this problem from the previous two examples is the sheer number of variables relative to the constraints. The increased number of variables introduces some additional flexibility. It so happens that in this case there are multiple ways to get to the same total cost number in this problem.

I've presented a few in the workbook, as shown in Figure 6.20. These are akin to multidimensional isoquant frontiers, borrowing from economic-speak, each peppered with several potentially different variants on utility

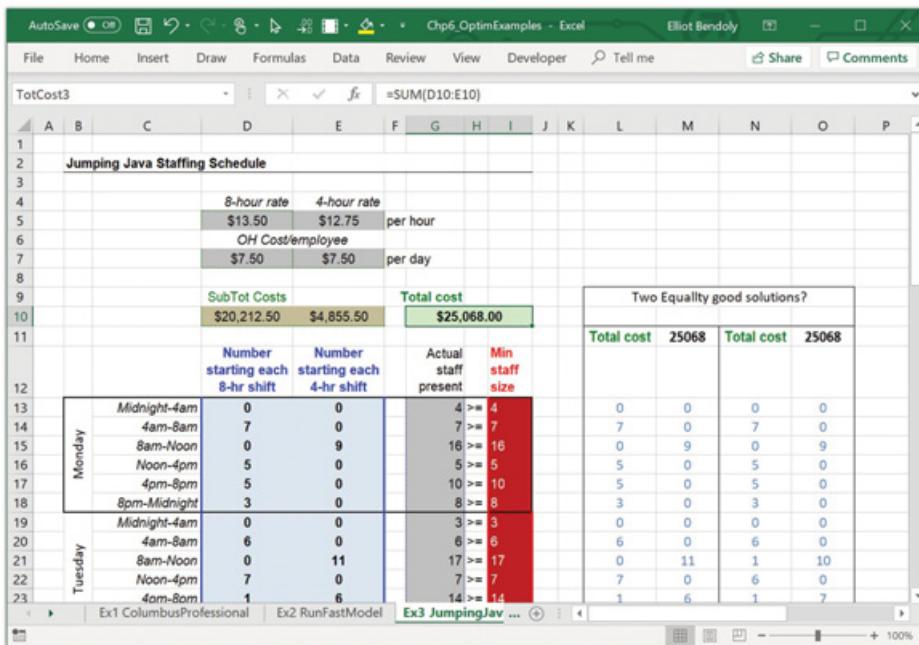


Figure 6.20 Staffing solution suggested by Solver

variable value sets. Which one Solver comes up with depends on a few things, including the nature of the install, whether you are using the LP method rather than another alternative (GRG Nonlinear, or Evolutionary), and even the starting solution you give Solver. However, aside from proposing different utility variable levels, if all you care about is “optimizing the objective subject to constraints,” these differences might not matter much.

This raises the question once again of how exactly Solver is doing its job, starting at one set of utility variable values and transforming these into solutions that reach good objective levels while obeying the rules set by connecting constraints. So let’s peel back the veil a bit. As noted earlier, Solver’s approach is to consider the many ways to improve upon solutions and capitalize on the most impactful changes before moving on to others. The same applies as it seeks to resolve violations of constraints. Said more simply, Solver is greedy but it takes things in steps.

We can actually observe steps taken through the use of the Shadow tool in the Blackbelt Ribbon add-in. Clicking on this feature of the ribbon gives you access to an interface along the lines of that shown in Figure 6.21.

The Shadow interface does expect that your objective and utilities are positioned in your spreadsheet in a single column, with the objective at the top. This makes for simpler documentation of the various solutions encountered

and their impact on the objective. The first field in this form asks that you outline where that column is positioned, followed by the question of whether you are looking to minimize or maximize the objective value (which relates again to the recording of solutions). Whatever you have Solver set up to do, just reflect that here.

The other fields in this form allow you to specify initial values of the utility variables, where you want records of solutions to start, and whether you have a specific cell that designates whether solutions are globally feasible or not (e.g., some cell with a conditional statement that returns TRUE if none of your connecting constraints are violated by a given solution). Your first use of the Shadow tool may also require you to copy and paste some code in the backend VBA environment, and since we haven't discussed that environment in depth, you might want to get back to this tool only after Chapter 8.

If you're intrepid, however, the tool works with any Solver setup and any Solver engine. In using the Shadow you might get more intermediate steps revealed than you had expected, or fewer, depending on how many utilities are subject to change, the nature of the constraints, and whether you are filtering for improvements and/or feasibility in the records. Using the settings described in Figure 6.21, and within a version of the Jumping Java problem set up in Chp6\_JavaShadowing workbook, you can generate a record of

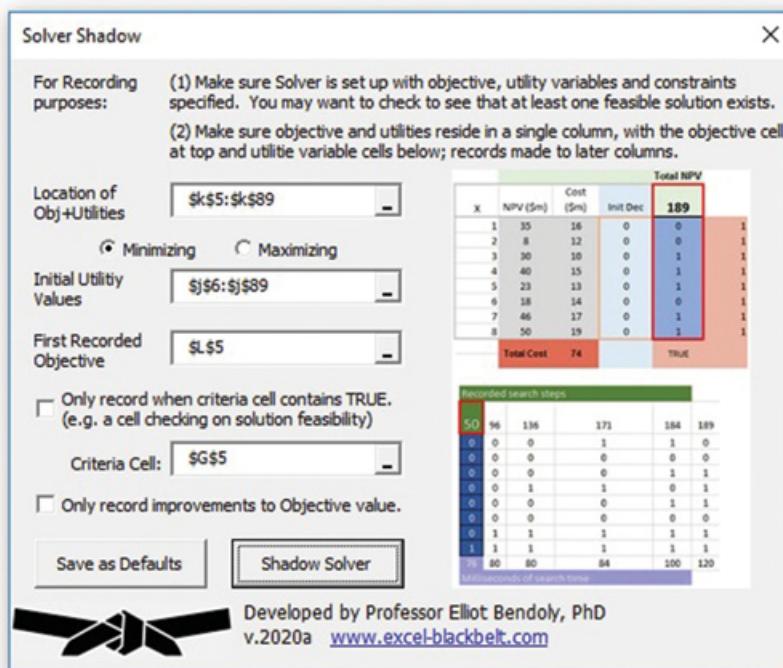


Figure 6.21 Solver Shadow tool interface of the Blackbelt Ribbon

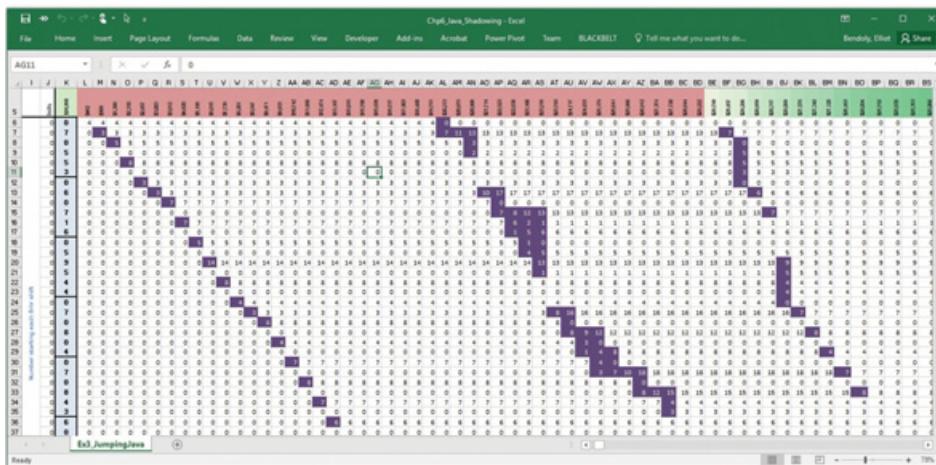


Figure 6.22 Steps of the solution search recorded by the Shadow tool

solutions encountered such as that presented in Figure 6.22. Each subsequent column is a solution examined by Solver in its search. I've added in some conditional formatting to help emphasize the incremental tweaks between subsequent solutions (purple cells), as well as which solutions are infeasible (top row reds), and improvements encountered near the end of the search process. At the bottom of these column records we have the amount of processing time taken to get to each solution.

From this depiction, or a closer inspection of the actual file, some of the black-box mystique around Solver might be dispelled. Solver doesn't come up with a final solution instantly. It's very methodical, a lot like the Nearest-Next heuristic or the savvier of the two MazeRun approaches, building up a solution step by step. As it further develops solutions and works to adjust these in increments to refine them (the last several columns of the sheet in Figure 6.2), it's learning a great deal about the nature of the problem and the extent to which the final solution is limited by various connections.

In Section 6.4 we'll talk a bit more about what Solver learns by the end of this process and how we can use this information.

#### 6.3.4 Example #4: Madrenko Fine Art Galleries

As a final example, before we drill further into what is learned from the search process and where further opportunities might exist, let's reconsider the work of Madrenko Fine Art Galleries (introduced in Chapter 5 in the discussion of directional path visualization). Madrenko is a buyer and reseller of fine Eastern European paintings and sculpture. Based on input from local staff, top management foresees a likely need to restock its galleries in

Montreal, Paris, and New York in the next six months. Madrenko has identified works in its Minsk, Budapest, and Zurich offices that have not received recent offers for purchase but are likely to do well in their more westerly markets.

The Montreal, Paris, and New York galleries are also better equipped to showcase these pieces. Based on past experience, top management doesn't believe it matters which works each of these galleries receive, as long as *most* of the anticipated gallery need is covered (equal sales prospects). However, it does want to get formal buy-ins from local gallery chiefs, to check against unanticipated issues, before making transport arrangements. All told, top management anticipates a demand for three additional units in Montreal, ten in New York, and six in Paris. Surpluses of two units exist in Zurich, five in Budapest, and eleven in Minsk. Madrenko already sees that not all expected demands will be covered; regardless, it would like to cover as much of that demand as possible at the lowest cost. What lowest-cost reallocation of works should top management propose?

Figure 6.23 provides an example setup for this problem, including details on the costs of transit (in hundreds of dollars) between the six cities (see Chp5\_Madrenko). These costs supposedly include transport as well as courier fees and the cost of paperwork filing and international tariffs.

In this instance we can manually run Solver as we have, ensuring utility variables are integers representing the flow of units from one city to another. Additional connecting constraints ensure physical laws are obeyed: total

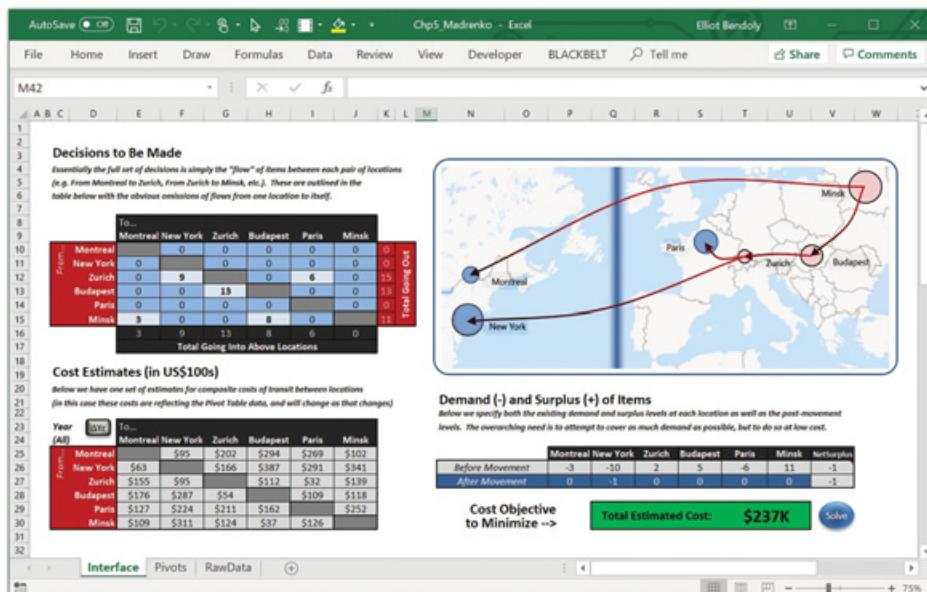


Figure 6.23 Facts, setup, and visualization for the Madrenko shipping problem

units leaving a city must be less than or equal to preexisting units plus any units arriving. An optimal solution should furthermore involve no surplus remaining, and only one unit of unfilled demand.

Further, as an alternative to our standard manual activation of Solver, we have a couple of buttons embedded in the sheet of this example. They allow changes to cost data referenced (modifications to Pivot Table summaries) and also permit the automated initiation of Solver's search (more on both of these automation activities in Chapters 8 and 9). Both the tabular and graphical depictions are updated when Solver completes its search. The graphic in particular emphasizes the elimination of surpluses and the coverage of the majority of demand.

## 6.4 Deeper Insights into Optimization Solutions

In our examples costs certainly deter certain solutions, yet they don't appear as explicitly as do constraints in Solver. Issues listed as "constraints" can almost be viewed as infinite in cost – that is, if one attempts to cross these lines. But not all connections listed as constraints end up having a major impact on the solution that Solver provides. Those that do are referred to as "binding." Those rules that are active but are less severe than binding constraints, and therefore don't actually hold us back, are called "nonbinding."

The concept of a binding constraint is analogous to the concept of a bottleneck in operations management. Bottlenecks are always the key elements that hold us back, even if other rules simultaneously apply. Managers are (or at least should be) always trying to find ways to break down bottlenecks and find new ones to tackle. When bottlenecks (binding constraints) are broken, we expect other limitations (originally nonbinding constraints) to take their place as we adjust utility levels and use up slack around those constraints.

Solver provides a few handy mechanisms for describing which specific constraints in a problem are binding, how much they can be modified (presumably at some cost) before becoming nonbinding, and to what degree other currently nonbinding constraints can be modified (often at some marketable gain) before becoming binding themselves. We're going to focus on the interpretation of the most straightforward of these tools in this regard: Answer Reports. These reports can tell us specifically which constraints are binding and which still have some slack. It does this by reporting on cell references involved in a problem – that is, cells containing utility variables (and implied decisions based on these) and the cells that contain the limits to which these utility variables are subject.

### 6.4.1 Example #5: Fashion Denim, Inc.

Fashion Denim, Inc., is a designer jeans manufacturer that currently produces all their goods in the United States. To cut costs, Fashion Denim is considering sourcing some of its materials from overseas for next month's production. Three of the major raw materials used in jeans manufacturing (cotton, zippers, and indigo dye), as well as how much of each material is available, are shown in Figure 6.24.

In addition, the procurement costs of each material from the United States and overseas are shown. Next month Fashion Denim wants to produce 100,000 pairs of jeans. The raw material requirements for this plan are as follows:

Cotton (yards): 300,000

Zippers: 100,000

Indigo Dye (ounces): 25,000

Comparing this to the total amount of components available, this goal seems generally achievable. However, given high import tariffs on certain goods purchased overseas, Fashion Denim needs to limit the quantity of zippers and dye imported to 50,000 and 1,500 respectively. What portion of materials should Fashion Denim acquire from the United States and

The screenshot shows an Excel spreadsheet titled "Chp6\_OptimExample...". The first table, located in rows 3 to 6, is titled "Amount Available" and shows the quantity of three materials (Cotton, Zipper, Indigo Dye) available in the USA and Overseas, with a total column. The second table, located in rows 9 to 13, is titled "Procurement Costs" and shows the cost per unit for each material in the USA and Overseas. Both tables have columns for "USA" and "Overseas".

	Amount Available		
	USA	Overseas	Total
Cotton (yards)	395000	20000	415000
Zipper (per unit)	400000	85000	485000
Indigo Dye (ounces)	54000	48000	102000

	Procurement Costs	
	USA	Overseas
Cotton (per yard)	0.03	0.07
Zipper (per unit)	0.21	0.16
Indigo Dye (per ounce)	0.68	0.54

Figure 6.24 Facts relevant to the Fashion Denim case

overseas to minimize the total cost, meet production requirements, and limit import tariffs? To find the answer, we might set up the problem as shown in the workbook. Provided integer constraints are not ignored, Solver gives us the solution shown in Figure 6.25 (when specified appropriately).

But just before it gives us this solution, it asks us what kinds of reports we might be interested in (Figure 6.26). We've been skipping over these options up until now, but let's select Answer Report on another fresh search of this problem (utilities initially set to zero). Figure 6.27 shows what we get when we select that option and hit OK.

	A	B	C	D	E	F	G	H
15								
16								
17								
18	Cotton (yards)	300000	0		Acquired Materials		Production Requirements	
19	Zipper (per unit)	50000	50000		300000 >=	300000	yards	
20	Indigo Dye (ounces)	23500	1500		100000 >=	100000	zippers	
21					25000 >=	25000	ounces	
22								
23	Total Cost	\$44,290						
24								

Figure 6.25 Solution to purchasing problem by Solver

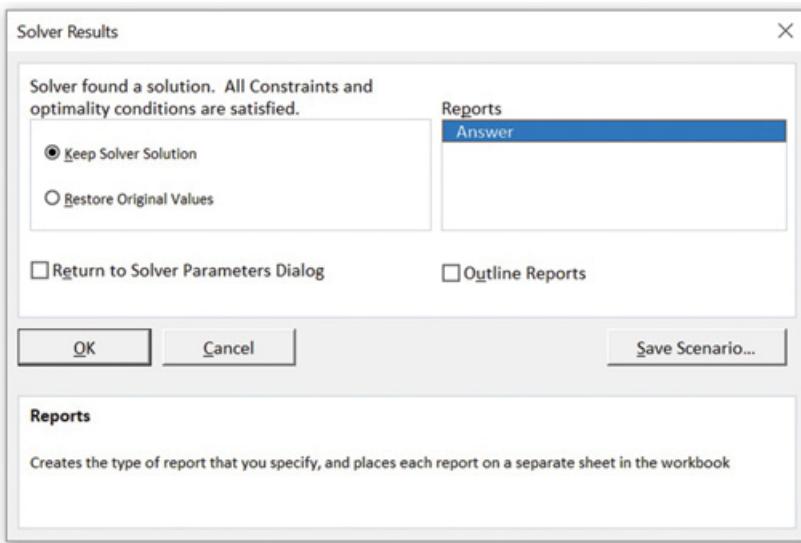


Figure 6.26 The Answer Report option

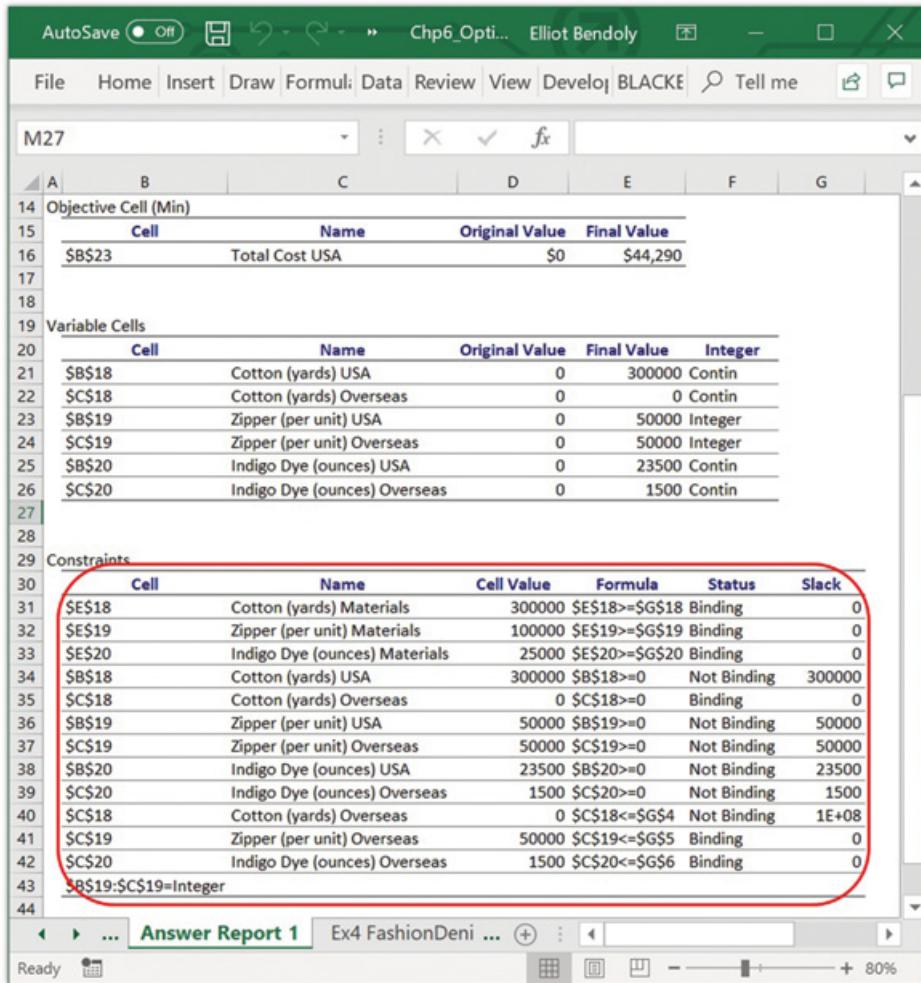


Figure 6.27 Sample output of the Answer Report

We could figure out most of this information from the solution provided, but it's explicit in the report. Solver is revealing which connecting constraints bind our decision (no slack, no wiggle room without getting worse, violating an explicit rule, etc.). Keep in mind that integer constraints are always binding. From this report we might be inspired to ask a next very natural question – specifically, what the situation would be if some of these constraints were modified. For example, what if import tariffs were eased on zippers (which would essentially break the binding nature of this constraint)? If we eliminated that constraint entirely, Figure 6.28 shows what we'd get.

This is good news because our total cost drops from \$44,290 to \$41,790 – provided, that is, that there is some mechanism by which to ease this limitation in the first place. Sometimes extra time and money can be spent in the

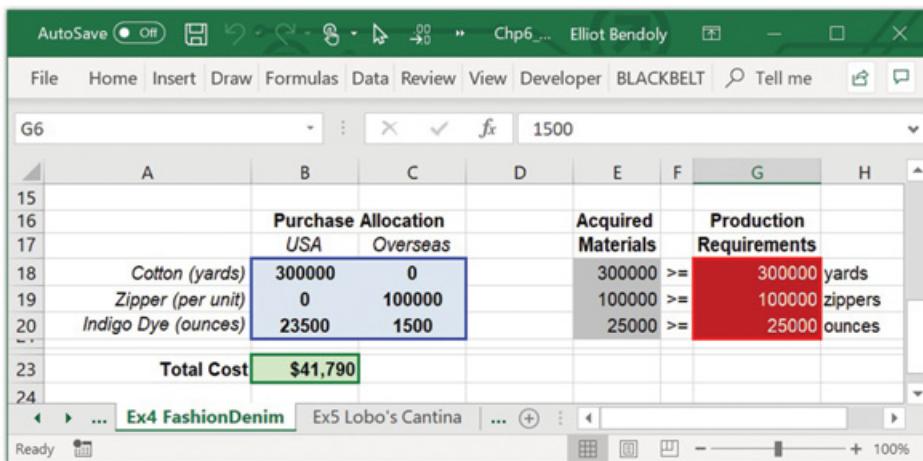


Figure 6.28 Alternate solution after eliminating a binding constraint

real world to modify the constraining nature of the landscape in which we are seeking solutions. The question is whether the gains are worth the additional time and money. After all, sometimes binding constraints are only slightly more severe than the nonbinding constraints that exist below them. Real added gains often require the consideration of a series of calculated changes.

#### 6.4.2 Example #6: Lobo's Cantina

Lobo's Cantina is considering a redesign of its current layout with the interest of generating a higher profit margin while continuing to cater to the desires of its target market. It currently has a total area of 30 feet by 60 feet (1,800 square feet) to work with. The owner realizes that for everything to work, he will need to consider a variety of implied service and operating requirements when making a decision. The space needs to house its dining area, bar, kitchen, wash area, restrooms, storage, and host stations, which include both the greeting station and the rear cashiers.

Although the owner has considerable freedom in terms of how to specifically lay out each area to provide for an aesthetic look, he does have to follow some rational rules in making the initial space division decisions. For example, if he plans to devote a large area of the floor to dining, he will want to make sure that he has enough kitchen and wash capacity to meet the implied demands of a population seated in that area. Demands for kitchen work (and, thus, requirements for kitchen space) will decrease as the size of the bar increases. Implied storage requirements (and storage space) will change depending on how much the restaurant is focused on bar service, as will restroom designs and the need for host stations. The following is a breakdown of the assumed restrictions with which the restaurant owner is dealing.

**Fundamental Seating Dimensions:**

Tables designed to seat four take up 24 square feet of space.

Tables designed to seat two take up 8 square feet of space.

Any seat at the bar (plus bar-counter space) takes up 5 square feet.

**Additional Fixed Requirements:**

There must be at least one greeting station that takes up a minimum of 20 square feet. Each host station is expected to provide enough capacity for handling 24 individual tables at most. Each additional host accommodation requires an additional 20 square feet of space.

There must be at least one rear cashier station (taking up 10 square feet of space) because bar customers pay at the bar and they don't need rear cashier service. In other words, the need for additional rear cashiers depends only on the amount of dining volume anticipated. Each rear cashier is thought to be able to handle work for up to twenty tables in use.

Minimum kitchen amenities begin with a required 40 square feet of space. Every additional 20 chairs in the dining area translate into another approximately 5 square feet in the kitchen. However, the current building and sewage code does not allow for a total kitchen capacity beyond 400 square feet.

Minimum cleaning amenities require a minimum of 20 square feet of space.

Another 5 square feet of kitchen space is needed for every additional 40 chairs in the dining area or the bar. Building and sewage codes limit this total space to 200 square feet.

Because of inventory management policies, the restaurant tends to buy alcohol less frequently than food, which it obtains throughout the day from neighboring markets; this means it generally stores more alcohol than other consumable inventories. Each additional seat at the bar increases the storage needs by 5 square feet. Each seat in the dining area increases storage needs by only 1 square foot.

Lastly, there must be one male and one female restroom. Each must have a total of 25 square feet to provide the minimal functionality (one sink and chamber per restroom). Each additional chamber adds another 15 square feet. The total number of chambers needed in each restroom should be based on expected average need at any given point in time. The forecast the restaurant will use to derive this total is as follows: number of chambers per restroom must be equal to at least  $(\# \text{ bar seats})/30 + (\# \text{ dining seats})/60$ . The sewage code allows for a maximum of five chambers for each restroom.

**Profitability and Market Issues:**

The owner believes that the average profit his restaurant can generate per dining seat over the course of a day is \$340 for two-seat tables and \$400 for four-seat tables, which are more likely to spend money on appetizers and \$560 per average bar seat. However, he is concerned that an excessive bar space might damage the image that the restaurant attempts to portray to its overall market

segment (which consists of customers who might be attracted to either the bar or seated dining on any given day). With that in mind, the owner wants to make sure that the bar area never takes up more than half the area specific to dining. If we want Solver to help us, we'll have to translate those specifications into something more formulaic.

Interdependency Constraints in brief:

The number of greeting stations must be at least ((# of two- and four-seat tables combined)/24); total greeting station space equal (# of greeting stations × 20).

The number of cashier stations must be at least ((# of two-and four-seat tables combined)/20); total cashier space equals (# of cashier stations × 10).

The kitchen space should be at least (40 + (5 × ((two-seat tables × 2) + (four-seat tables × 4))/20)); total cannot exceed 400 square feet.

The cleaning space should be at least (20 + (5 × ((two-seat tables × 2) + (four-seat tables × 4)) + bar seats)/40)); total cannot exceed 200.

The storage space should be at least equal to (1 × ((two-seat tables × 2) + (four-seat tables × 4)) + (5 × bar seats)).

Total number of chambers should be at least ((bar seats/30) + ((two-seat tables × 2)/60) + ((four-seat tables × 4)/60)); total restroom space should be equal to ((25 + 25) + (15 × (number of chambers))).

Total area taken up by the bar can't be more than half the area taken up by dining.

Figure 6.29 shows an example of how we might structure the various utility variables and their connections to profitability and the functional requirements spelled out in this problem in spreadsheet form. (All constraints and relationships are built in as they were in previous examples.)

Again, if we're using Solver, we'll need to spell out which cells represent utilities and which cell relationships represent constraints. With the appropriate, meaningful cell labeling, we might have the results shown in Figure 6.30 in Solver (lots of constraints here; only a subset is shown in the figure).

And if we use this set of constraints, Figure 6.31 shows the results from Solver. Those noninteger recommendations appear dubious – I doubt anyone would feel comfortable using a quarter side of a toilet. So, let's rerun the search adding appropriate integer specifications. We'll get what's shown in Figure 6.32.

That's more like it, but it's a very different solution seatingwise than what was previously suggested. You wouldn't get here just by rounding the numbers in Figure 6.31. Again, integer constraints can have a major impact on the general nature of a solution.

Take a look at the Answer Report generated for this search in Figure 6.33. Again, Solver's labeling in these reports leaves something to be desired, but we get a general idea. A little close examination shows what's holding us back from doing better, where our next greatest challenges exist, and where excessive slack might provide other implications on how we use resources.

Seating Decisions				
	Actual			
Number of 2-seat tables	0			
Number of 4-seat tables	0			
Number of bar seats	0			
Other fixed element decisions (implied)		Actual	Implied Minimum	Absolute Minimum
Number of greeting stations	0	0	1	NA
Number of cashier stations	0	0	1	NA
Number of 'chambers' / restroom	0	0	1	5
Space taken up by...		Actual	Maximum	Also total bar space can't be more than 1/2 total dining.
...2-seat tables	0	NA		
...4-seat tables	0	NA		1/2 Dining: 0
...barseats	0	NA		Bar: 0
...greeting stations	0	NA		
...cashier stations	0	NA		
...kitchen	40	400		
...cleaning	20	200		
...storage	0	NA		
...restrooms	50	NA		
Total Space	110	1800		
Total Expected Profitability		\$0		

Figure 6.29 Possible spreadsheet setup for layout example

Interestingly, while we come close to the limits of several of the rules, the only one that is truly binding, aside from the integer limits, is the minimum chamber requirements. (We actually have a little slack on the use of cashier stations, too.) In reality, this constraint works in tandem with the total area requirement. We have additional space, but adding even one more seat would increase the restroom size requirements beyond the space available.

Oddly enough, coming up with some way of reducing the per-chamber size can impact our solution; reducing it to 14 square feet allows full use of our total space (now a binding constraint) with greater profit (about \$500 more) and without major changes in our overall solution. A change to a chamber size of 12 square feet has a huge impact on our design (no more four-seat tables, but another \$1,000 or so profit; and now the binding constraint is the sewage ordinance). The ‘real’ Lobo’s, on which this case is grounded, in fact made an equipment change of this very kind.

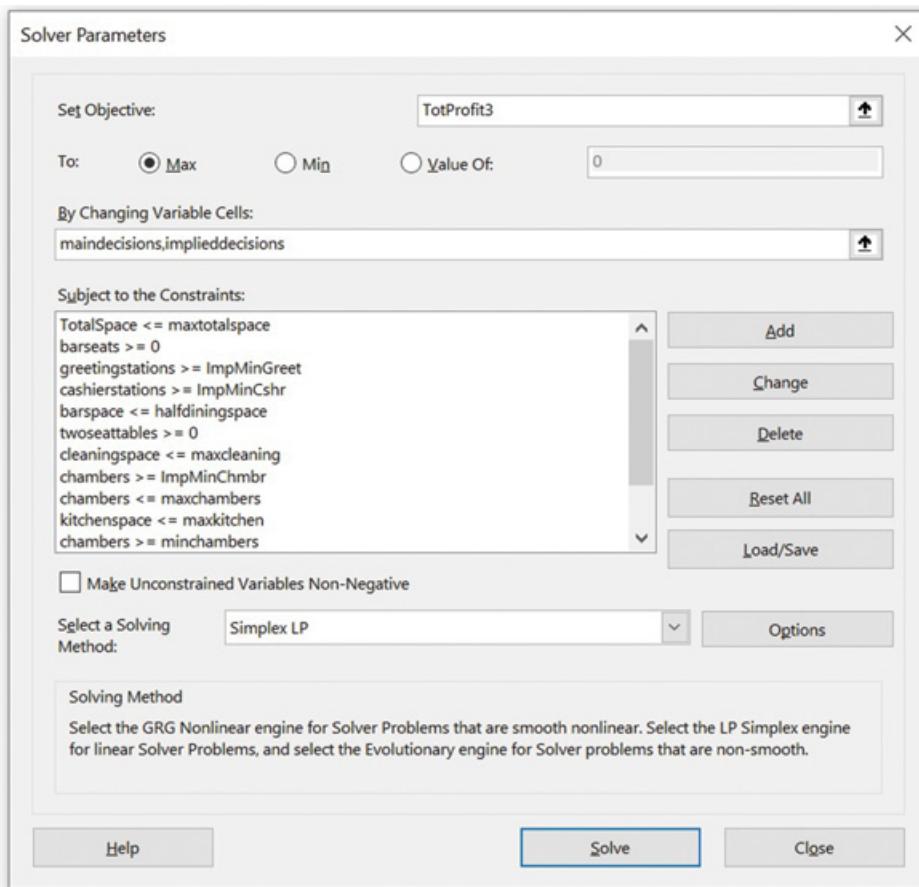


Figure 6.30 Example Solver specification for layout example

Another feature that the Answer Report provides is a summary of which constraints seem to be the least binding in a given scenario. The two most slack constraints in the original integer solution are those relating to the building code limits on the kitchen and cleaning spaces. A manager might ask, “If we’re so far below our maximum allocations, could we sell the rights to such allocations to another neighboring firm?” – perhaps one that finds such limits to be currently restrictive. Furthermore, the implied requirements of the cashier station are more than fulfilled by the number of stations we have in the original integer solution. One implication might be that of underutilization. A manager might then ask, “Could these stations be used for other activities, as well?”

As we see, the search for solutions often ends with the inspiration to ask yet more questions. The more detail a decision support system can provide regarding recommended solutions, their risks, and what holds them back from being better, the more likely subsequent questions are to be fruitful.



The screenshot shows the Microsoft Excel interface with the title bar "Chp6\_O... Elliot Bendoly" and the ribbon tabs "File", "Home", "Insert", "Draw", "Formulas", "Data", "Review", "View", "Developer", "BLACKBELT", and "Tell me". The active cell is A1, and the formula bar shows "Microsoft Excel 16.0 Answer Report". The main content is a table titled "Answer Report 1" with the following data:

Cell	Name	Cell Value	Formula	Status	Slack
\$C\$10	cashierstations	3	\$C\$10>=\$E\$10	Not Binding	0.65
\$H\$16	barspace	285	\$H\$16<=\$H\$15	Not Binding	31
\$C\$23	TotalSpace	1799.75	\$C\$23<=\$E\$23	Not Binding	0.25
\$C\$11	chambers	4	\$C\$11>=\$E\$11	Binding	0
\$C\$9	greetingstations	2	\$C\$9>=\$E\$9	Not Binding	0.041666667
\$C\$20	cleaningspace	65.75	\$C\$20<=\$E\$20	Not Binding	134.25
\$C\$19	kitchenspace	166	\$C\$19<=\$E\$19	Not Binding	234
\$C\$3	twoseattables	31	\$C\$3>=0	Not Binding	31
\$C\$5	barseats	57	\$C\$5>=0	Binding	0
\$C\$10	cashierstations	3	\$C\$10>=\$F\$10	Binding	0
\$C\$11	chambers	4	\$C\$11<=\$G\$11	Binding	0
\$C\$11	chambers	4	\$C\$11>=\$F\$11	Binding	0
\$C\$4	fourseattables	16	\$C\$4>=0	Binding	0
\$C\$9	greetingstations	2	\$C\$9>=\$F\$9	Binding	0
\$C\$3:\$C\$5=Integer					
\$C\$9:\$C\$11=Integer					

Figure 6.33 Answer Report for layout problem

### Supplement: Decision Logic with Interdependent Utilities

As mentioned earlier, it's easy for IF statements to become go-to mechanisms for real-world model development, since the real world is full of conditional rules. We've made many of these rules ourselves, often for simplification in practice. Ironically, these same tactics can make analysis harder rather than easier.

We also discussed, and will consider again in the next chapter, the fact that Solver's often successful approaches to solution development are hampered by models including statements that are not explicitly of a particular mathematical form (linear, continuous). This is a problem with regard to IF statement dynamics, which, like reference functions, more or less play out as discontinuous step functions in Excel. So, what do we do when we have to capture real-world conditional logic, relating multiple utility variables in models we want Solver's help with?

The logic matrix presented in the Connections section of the OUTCOMES Cycle discussion helps us at some level. For example,  $X_1 \geq X_2$  would be the translation the matrix provides for the example condition that allows a second action ( $X_2$ ) to occur only if another action ( $X_1$ ) is taken. The utility variables

$X_1$  and  $X_2$  are (0,1) binary variables, equivalent to “False/True” or “Don’t Do/Do.” If either a third or fourth action (or “project” from the earlier discussion), but not both, can be pursued, the application of the logic matrix would be different. We could cross “ $X_3$ ” with “NOT  $X_4$ ” (or the quantity  $1 - X_4$ ) and arrive at the following relationship:

$$X_3 \leq 1 - X_4, \quad \text{or} \quad X_3 + X_4 \leq 1$$

And if  $X_3$  must occur if  $X_2$  hasn’t been performed:

$$X_3 = 1 - X_2, \quad \text{or} \quad X_3 + X_2 = 1$$

Other conditional relationships can also clearly apply between Boolean and non-Boolean utilities. Sticking with a “projects” example, they can be based on simple ideas such as “the number of project group members must be less than the maximum” or “the number of group members must be greater than the minimum.” The role of Boolean decisions can give meaning to what these minimums and maximums are. For example:

$$\text{MaxN} = (1-\text{NoMax}) * \text{CriticalMax} + \text{NoMax} * \text{Limitless}$$

where CriticalMax is some constant integer that would considerably constrain membership, and Limitless is a constant far beyond any possible group size. The Boolean decision NoMax, perhaps an option associated with some additional cost, essentially serves to toggle what the effective maximum membership is and thus drives the nature of the simpler membership constraint. Costs for overriding or “breaking” constraints are also common in practice.

What about scheduling projects? What if you wanted to handle all of these projects simultaneously (with different workers) but needed to make sure they finished before a certain common time (MaxT)?

$$X_1 * T_1 \leq \text{MaxT}, X_2 * T_2 \leq \text{MaxT}, \dots \text{etc.}$$

As long as the values  $T_1$  and  $T_2$  are constant, we still retain linear conditions that Excel can work with. Even if these projects were not simultaneously handled but rather handled by the same worker team in serial, we’d have no issues constructing a connecting constraint relationship that retained linearity:

$$X_1 * T_1 + X_2 * T_2 + \dots \text{etc.} \leq \text{MaxT}$$

If we needed to track the project that took the greatest (max) amount of time, we could also come up with a linear tactic to determine that length of time. In the simultaneous project case, imagine that we were paying for space for doing these projects, and that the entire space needed to be paid for as

long as projects were being conducted (regardless of how many projects remained). We could define an additional utility variable (rather than a constant) called Shortest and simply apply the following constraining connections:

$$X_1 * T_1 \leq \text{Shortest}, X_2 * T_2 \leq \text{Shortest}, \dots \text{etc.}$$

The value Shortest would then be used as part of the Objective function (e.g., penalizing by  $C_s * \text{Shortest}$ ) or in other budgeting constraints (e.g., subtracting out  $C_s * \text{Shortest}$ ). We could even replace the bulk of the MaxT constraints at this point with the simple statement of  $\text{Shortest} \leq \text{MaxT}$ . True, it's an extra utility variable to decide upon, but it connects things in a flexible and meaningfully constraining way.

We could go on with examples, but they would simply be permutations of the above linear inequality forms. The short version of this is as follows: Complex logical conditions that constrain our decision making don't need to also constrain our ability to use tools like Solver. With just a bit of linear math, there are often a range of workarounds available, as long as we are willing to think in binary rather than just with words like True/False, Yes/No, Do/Don't. The more you get used to seeing the complexity of the systems we work in and create through multiple alternative lenses, the less daunting the challenges you face will seem.

## PRACTICE PROBLEMS

### *Practice 6.1*

A Fortune 500 company needs to hire a group of consultants from Just Right Consulting to assist with a large technology rollout that will require around-the-clock support. A day is divided into eight-hour shifts. The total number of consultants required during the day shift is shown in Table 6.9.

The client would like to minimize the labor costs associated with staffing the project. Consultants from Just Right work five days a week and are entitled to two consecutive days off each week. We know there are only seven ways that each consultant can have two consecutive days off, with each break starting on a different day of the week (staggered approach). One straightforward way to structure the problem is to determine how many consultants of each break type to have staffed each day.

One twist in the problem is that any consultant required to work on Saturday receives an additional 5 percent overtime pay (think of their total weekday pay level as 100 percent; on Saturday it's 105 percent), whereas those required to work on Sunday receive an additional 7 percent of overtime pay (107 percent of typical weekday pay for Sunday). This will undoubtedly result in a tendency to avoid excess weekend scheduling whenever possible.

Table 6.9 *Details associated with consultant staffing example*

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Consultants Needed	9	18	15	18	15	15	15

Determine how many consultants of each break type (i.e., beginning their two-day break on a specific day) would be needed to at least cover the demand, while trying to keep the total cost as low as possible (again, accounting for weekend premiums).

### *Practice 6.2*

Fun Viewing, Inc., is a new television network that has become extremely popular in recent months. Fun Viewing is trying to decide which TV shows to air during their fall lineup. The creative staff has worked tirelessly reviewing market research reports to give Fun Viewing an idea of how profitable each show will be (based on the number of viewers, ad sales, competing shows, and so on). Fun Viewing has allocated \$28,000 for new TV shows. The objective is to generate the highest possible return on investment.

Eight scripts have been reviewed and deemed worthy to air on the network. Table 6.10 specifies the estimated return on investment (ROI) for each of these shows.

If a script is chosen, at least \$700 must be allocated to developing the show. To maintain a variety of shows, the board of directors has stipulated that at most \$7,000 can be invested in any project. A show's total return is the product of its ROI and the amount invested in the show. The decision to finance some shows cannot be made independently of other funding decisions. The rules outlined in Table 6.11 must be met:

Determine which shows to invest in (1 = yes, 0 = no), and how much to invest in each, so as to maximize total returns for the resulting portfolio of TV shows (subject to the rules previously outlined).

Hint: The OUtCoMES roadmap (in this chapter's supplement) presents some examples that may be particularly useful in this problem. You should be able to solve this as an LP with Solver.

### *Practice 6.3*

Football Fanatics, Inc., makes three types of footballs: the Pee Wee, the College Canon, and the Pro Player. The labor and materials requirements for each football are shown in Table 6.12 with information regarding the maximum amount of labor hours available (for a given activity) and the maximum amount of each material available.

Table 6.10 *Details associated with TV script selection example*

Show#	1	2	3	4	5	6	7	8
ROI	7%	9%	4%	25%	21%	14%	8%	16%

Table 6.11 *Constraints associated with TV script selection example*

<i>Show 2 can't be pursued unless Show 1 is</i>	<i>Show 4 can't be pursued unless Show 3 is</i>
<i>Show 3 can't be pursued unless Show 1 is</i>	<i>If Show 6 is funded, so MUST Show 7 (and vice versa)</i>
<i>Show 4 can't be pursued unless Show 2 is</i>	<i>Shows 5 and 6 cannot be funded at the same time</i>

Table 6.12 *Details associated with football fanatics example*

		Labor (hours/football)		Material Requirements (gram/football)		
<b>Pro Player</b>	<b>Revenue</b>	<i>Cut</i>	<i>Assemble</i>	<i>Rubber</i>	<i>Leather</i>	<i>Stitching</i>
55		0.3	0.5	250	200	60
<b>College Canon</b>	48	0.2	0.4	225	180	50
<b>Pee Wee</b>	30	0.1	0.3	150	100	40
<i>Max Available</i>		40	100	35,000	28,000	5,000
<i>Cost issues</i>		15	20	0.02	0.05	0.03
		<i>[\$/hour]</i>		<i>[\$/gram]</i>		

The firm purchases only as much of the three materials (rubber, leather, and stitching) as it uses at the prices shown in the table. It wants the Pro Player to comprise at least 10 percent of all the footballs it produces and sells. In addition, the number of Pro Player and College Canon footballs (in total) must not be more than 50 percent of the total number of footballs produced and sold.

How many of each type of football should be produced to maximize its profits, subject to the availability and sales-planning constraints given? Which input constraint appears to be most limiting? How does this result differ when integer decisions (i.e., whole numbers of footballs) are no longer assumed necessary for planning estimates? You should be able to solve this as an LP with Solver.

## References

- Gigerenzer, G. Todd, P. M., ABC Research Group. 1999. *Simple heuristics that make us smart*. Oxford.
- Guth, A. H. 1998. *The inflationary universe: the quest for a new theory of cosmic origins*. Basic Books.
- Sommer, S., Bendoly, E., Kavadias, S. 2020. How to Search for the Best Alternative? Experimental Evidence of Search Strategies to Solve Complex Problems. Forthcoming at *Management Science*.

# Searches in Highly Complex and Uncertain Landscapes

**Objective:** *Understand the limitations of certain prescriptive analytical approaches and how to leverage evolutionary searches for acceptable-risk solutions under complex and uncertain conditions.*

The approaches in Chapter 6 were effective in the contexts they were applied to. Solver's LP approach gave us solutions that yielded the very best outcomes for the linear problems it was given. In comparison, the Nearest-Next heuristic gave us good solutions, if not the best, that would have been very hard to find using less organized tactics. Neither approach looked comprehensively at these landscapes. Solver didn't have to, given the linearity of the problems. Why explore all points on a straight line, when you can explore the highest endpoints? Nearest-Next looked at only a handful of points as well, deliberately.

But there are caveats to these approaches. Nearest-Next can miss things, just as our savvy MazeRun approach could (in a multiexit maze). Solver can miss things too, when landscapes get more complicated – sometimes very big things, sometimes everything. Understanding how tools and approaches can fail and being familiar with alternatives that can help fill the gaps in these situations are critical to successful prescriptive analysis.

## 7.1 Limitations of Simple Hill Climbing

The unfortunate reality is that even small problems can have nuances that make Solver's job extremely difficult and the resulting solutions thus prone to poor performance (i.e., substantially less-than-optimal managerial recommendations). Why? As discussed in Chapter 6, most hill climbing algorithms typically only look into how to step by step, greedily but gradually, improve on a single, most recent solution (or one in the making). There are implications to this.

### 7.1.1 Simple Nonlinearity

As a thought experiment consider the following hypothetical performance surface, where performance along the Z-axis is some function of the two decision variables X and Y. In Figure 7.1 a circular point near the middle of the image represents a possible solution, one that at this point appears to be less than ideal. From a local perspective it certainly doesn't represent the apparent peak value of Z attainable (shown by an ellipse).

Based on our limited view of this landscape, we might immediately conclude that the best decision exists at the top of the hill. This immediacy is really the result of the graphic visualization of a portion of this terrain as well as our ability to interpret it. Such a capability (holistic or selective visualization and interpretation) is not baked into an algorithm like Solver; it doesn't work the way we might. Instead, and once again, such algorithms are more or less restricted to information on the most recent solution considered (starting with the solution initially provided) and the nature of the terrain incrementally around it. It will pursue changes in the solution that improve upon the current objective (i.e., to climb a hill) and then stop when it gets to a point where it can't make any more improvements (at the peak). Critically, in a nonlinear landscape such as this, this approach won't recognize the peak as

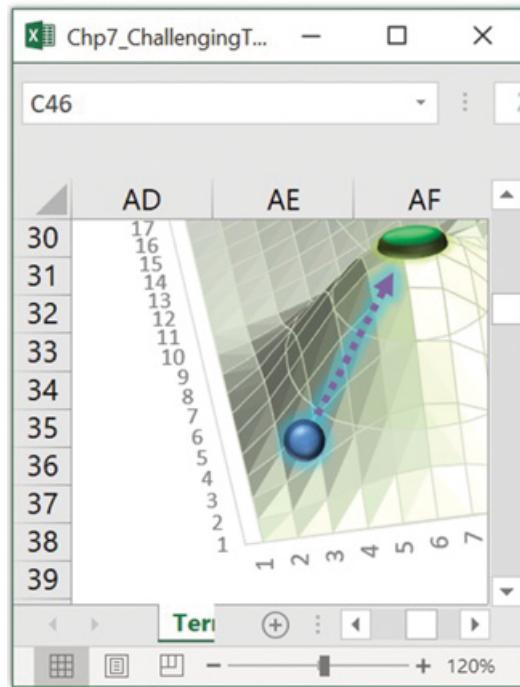


Figure 7.1 Hill climbing active in a nonlinear frontier

the best solution until it is actually there. And Solver's GRG Nonlinear approach, a hill climber, would do exactly that, more or less.

If there is a single nonlinear hill to climb and that is the only major complication of the landscape (i.e., we would say this is a unimodal landscape), Solver's GRG Nonlinear approach will prove sufficient. Or if there is a single well to delve into, defined by continuous curvature, we should be just fine. And there are many examples in which this is the case. In fact, simple linear regression is a nice example in which the minimum of an objective is sought out through changes in utility variable values. The utilities are the beta coefficients in the linear model of fit and the objective involves the sum of square differences between observed values of the dependent variable and the estimations provided by the model (based on the utilities in their role as coefficients). If you set up an optimization to minimize that objective by modifying these utilities, you should get the standard solution that packaged regression tools will otherwise provide (see the workbook Chp7\_FitbyRegression). Linear regression is linear regression.

### 7.1.2 Complex Nonlinearity and Discontinuity

All of this seems straightforward enough and not really anything to be concerned with at this point. However, we can get into some pretty serious problems in our reliance on hill climbing when performance landscapes are even slightly more complex; for example, if we have a landscape with not one but two or more hills. Depending on where our first guess is, we might climb the wrong one, one that's not the highest, and get stuck. Even we, as visual observers and integrators, might make such a mistake if we limit our overall view to the landscape shown in Figure 7.1 as opposed to the more global view shown in Figure 7.2.

In such cases the algorithm provides us with what we call a local optimum, whereas the global optimum (best solution, represented by the shaded dot above the ellipse) eludes it. Because the standard Solver GRG Nonlinear method uses such an approach to handle nonlinear objective functions in optimization, problems with more than one peak may be improbable for Solver to solve to optimality – or at least impossible for Solver to guarantee that it has provided the best option available.

Is this something to be concerned with? Do difficult objectives and decision terrains really exist in practice that would make reliance on hill climbing algorithms alone problematic?

**Marketing example:** Demand is often dependent on price. If revenue is a multiplicative function of both demand and price, it's going to be nonlinearly related to the price decision. Beyond this, demand for one good often impacts demand

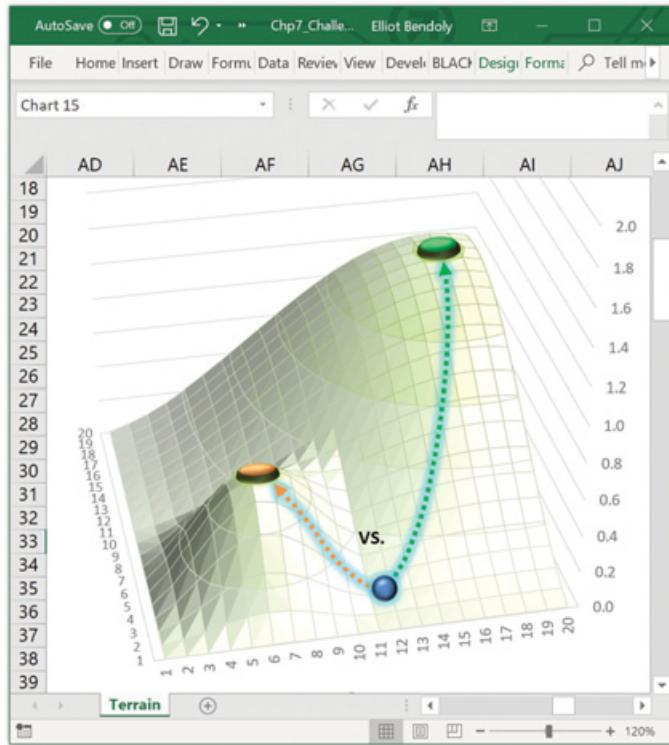


Figure 7.2 Hill climbing stopped at a local peak, missing global optimum

for other goods that are either complements or alternatives. Therefore, when faced with multigood pricing decisions, the objective landscape for expected revenue may have many distinct peaks. For some goods there may also be areas in which demand is not sensitive to price (i.e., demand is inelastic to price over these ranges). That in itself can give rise to multiple peaks in a nonlinear terrain (as in Figure 7.3 and the workbook Chp7\_Inelasticity).

**Finance example:** Think about all the complex calculations financial planners have to deal with on a regular basis, beginning with even some of the simplest like net present value (NPV). Nonlinearity typically pervades their work. In cases where complex portfolio management decisions need to be made, particularly when the performances of options are thought to be interrelated, it is difficult to simply assume that single-peak dynamics automatically apply. More to the point, making such an assumption and relying on a hill climbing approach cannot only result in a suboptimal decision but will also reduce the overall value provided by such analysts.

**Production example:** Consider a firm that makes and ships 12,500 low-end wrist watches over the course of a week. The total cost to run this operation is \$28,125. We want to reduce how many watches we produce. If we produce only one watch, would we assume it would cost us just \$2.25 ( $\$28,125/12,500$ ) to

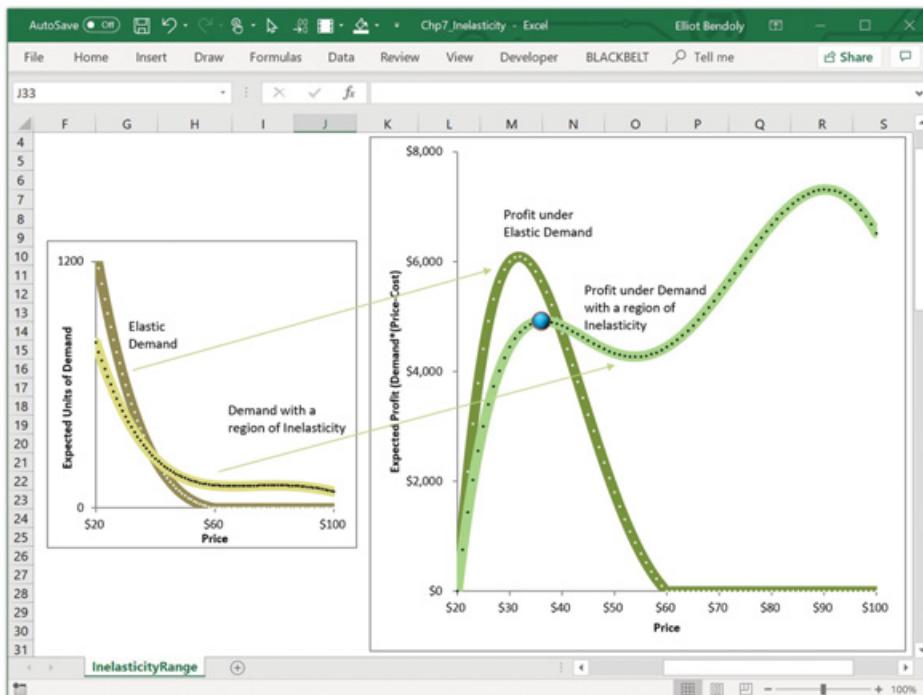


Figure 7.3 Hill climbing efforts further complicated by discontinuity

make? No, we wouldn't. We assume that there are fixed costs that don't diminish and that nonlinear economies of scale, due to efficiencies in bulk processing (buying, assembling, shipping, and so on), exist. We also expect conditional dependencies. For example, bulk purchase rates are adjusted not on a continuous scale but on a tiered one such as \$0.5/unit for 0–99 units of raw materials, \$0.35/unit for 100–499 units, and so on – a pretty complicated but realistic context for decisions.

As suggested by this last example, complex continuous nonlinearities aren't the only thing we have to think about. Figure 7.4 provides a much more complex visual example, where the objective is connected to the two utilities through a combined step function as well as an otherwise continuous nonlinearity. These kinds of sudden rises can be the result of rules that kick in only under certain circumstances. They are limits to ostensibly surrounding continuity, a bit like the thresholds that constraints impose, though associated with major shifts in the objective rather than limits to decision making. Indeed, in this visual example, the independent upper and lower bounds on our utilities are the only active constraints, although more complex ones could certainly be introduced if appropriate.

Solver's LP and GRG Nonlinear approaches also expect relationships with continuity in between the objective function and the utility variables

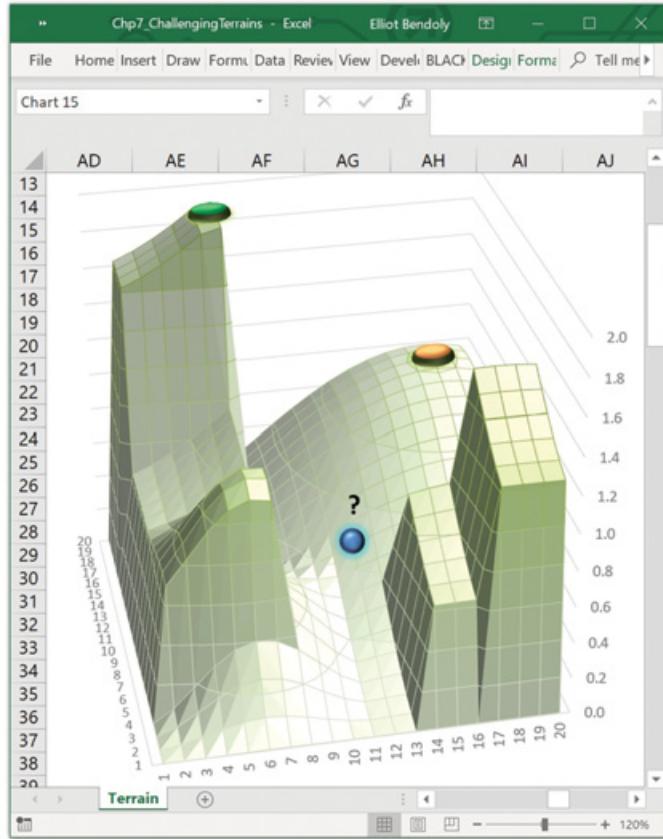


Figure 7.4 Five-city traveling salesman scenario

available, not a choppy, subdivided, or staggered one, nor one that is subject to random events. If your objective model includes elements such as IF statements or LOOKUP values – or if it relies on noisy, real-time or simulated data – Solver has its work cut out for it. Don’t bet on getting a great solution from Solver in these cases.

### ***7.1.3 An Example of Search Challenged***

As a buildup to more effective approaches to complex optimization, let’s further scrutinize the capabilities of the hill climbing algorithm through a full-blown numerical example.

The classic “traveling salesman/vehicle routing” problem is the same one we introduced when talking about heuristics in Chapter 6. It’s a tough one to solve, especially when there are a large number of sites to visit. It’s worth noting, yet again, that this is essentially a sequencing problem, and although it is often applied to routing, it has similar applications in work scheduling (what

to work on first), investment planning (where to transfer cash to next), training (what sequence of skills should be taught and in what order), marketing (what sequence of marketing activities will yield the best results), and so on.

Because we're already familiar with this setting, we'll stick to the routing case. Figure 7.5 from the Chp7\_FiveCityTSP workbook shows a pretty simple version of this problem, involving just five cities. We'll travel by plane to justify straight-line distances here, but in reality any approximate driving distance could just as easily be substituted.

A five-city scenario is something for which we could guarantee solution optimality manually without taking shortcuts. There are only five places to

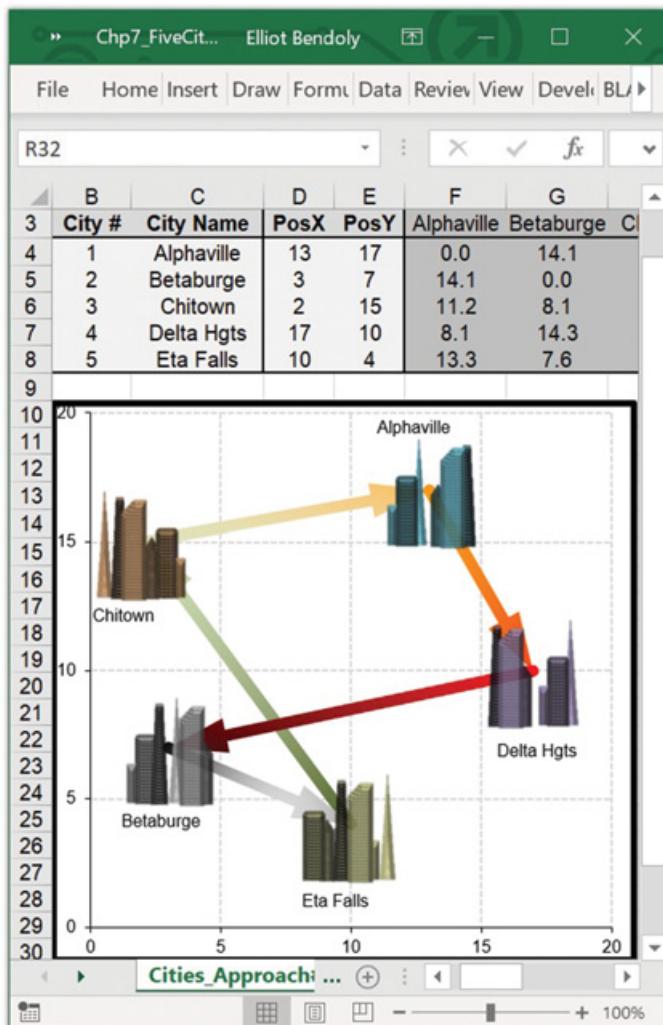


Figure 7.5 One approach to traveling salesman problem setup

start. Once that choice is made, there are only four remaining to choose from, then three, then two; in other words, five factorial ( $5!$ ) or 120 solutions. Furthermore, if we are indifferent as to where we start and we are doing a round trip (e.g., 3-4-1-5-2-3), the total distance for any given route, including the shortest, might be obtained by starting at any given city in that circuit in both forward and reverse order. This all assumes that there is no penalty associated with direction and, again, that the origin of the round trip doesn't matter. Both can be impractical assumptions, but unless they are explicitly specified, this leaves us with 10 equally good combinations out of those 120; Simple circuits around the perimeter of the polygon, formed by these five points.

Since we now know 10 good solutions exist – and it wasn't hard to come up with them by inspection – let's see if Solver's hill climbing can do the same. As with many problems, there are multiple ways to manifest the decision-making framework in this case, in preparation for an automated search. We already know what the objective is, and we have a sense of the utilities and connections conceptually from Chapter 6. Still, manifesting a complete model in a spreadsheet can get a little tricky. For purposes of discussion I have provided two approaches in the document Chp7\_FiveCityTSP. Figure 7.6 shows an image of the first approach. I'm using both city names and city numbers to refer to the sites that need to be visited. All distances between cities are presented in matrix form.

In a matrix (K4:O8, or "DecisionSet1") of equal size to that of the between-city distances I have the utility variables; in this case they answer the following questions: "For a specific city will it appear in the route first? Second? Third? Fourth? Fifth?" These 1s and 0s basically represent true or false. The row below those decisions provides the sum of the (0,1)

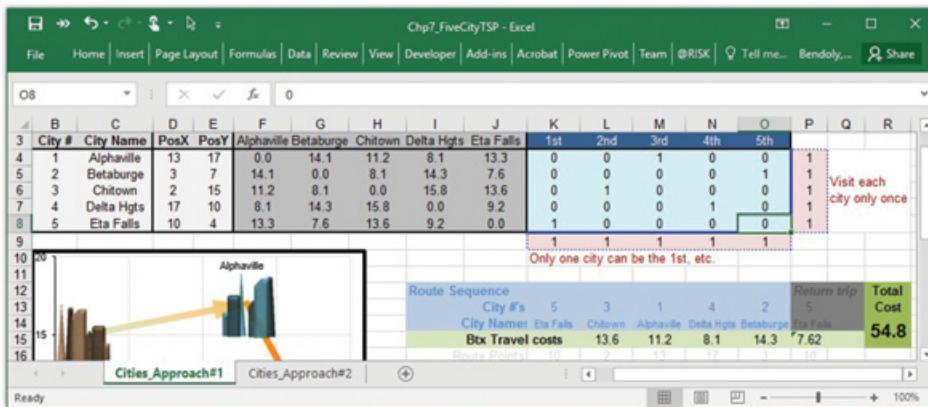


Figure 7.6 Solution provided by Solver for traveling salesman problem (no change from original)

decisions above it. Because exactly one of the cities should be visited first and only one should be visited second, that lower row should be all 1s. Similarly, the column to the right of that table is the sum of all (0,1) decisions to the left of it. Because each city must be visited exactly once, those sums should also be all 1s.

The numbers starting in cell K13 outline the actual routing sequence based on all those (0,1) utilities. From this sequence we just selectively extract the distances between each subsequent pair of cities and sum them, getting the associated value of the objective in cell R14 (named “Cost1”). With the objective, utilities, and connecting constraints in place, I could now try to use Solver to come up with an optimal solution, attempting to minimize total travel cost by making changes in the matrix of binary (0,1) utilities. Unfortunately, we are going to encounter a little problem.

The relationship between the objective and the utilities is far from a simple linear connection. If I try to use Solver to minimize Cost1, changing cells DecisionSet1, and subject to constraints which include ensuring DecisionSet1 are binary, we get an error message from Solver’s LP engine: “The linearity conditions required by this LP Solver are not satisfied.” If we switch to Solver’s GRG Nonlinear engine, it’ll run, but we will be told that improvements cannot be found – which is funny because we clearly know about improvements to the route shown in Figure 7.6. In other words, Solver doesn’t help us much at this point.

Perhaps the real issue was in the way I decided to set up the problem for Solver. Maybe we can make things more direct, manifesting a model form with fewer utilities for Solver to modify, as shown in Figure 7.7.

Here the utilities are perhaps a little more obvious. They answer the question “What is the sequence of city reference numbers associated with the route?” Our utilities are literally just a sequenced list of city reference numbers. The only thing I have to ensure is that each city appears exactly

	I	J	K	L	M	N	O	P	Q	R	
12	Route Sequence					Return trip				Total Cost	
13	City #'s	5	3	1	4	2		5			
14	City Names	Eta Falls	Chitown	Alphaville	Delta Hgts	Betaburgle		Eta Falls			
15	Btx Travel costs	13.6	11.2	8.1	14.3	7.62				54.8	
	Cities_Approach#1	Cities_Appro ...									

Figure 7.7 Alternative approach to traveling salesman problem setup

once in this five-stop sequence, which is what I'm doing in the range named "ExactlyOnce3," leveraging the COUNTIF function.

What do I get from Solver using this approach? The same thing as before. Solver can't devise how to improve the current solution. Again, we know the solution, and it took us only moments to get there manually. The problem in both manifestations is, ironically, our use of rather convenient albeit non-mathematical functions in Excel. OFFSET is not a mathematical function. It looks for content and returns associated content. It could be looking for text just as easily as numbers. Its use is creating a step-function relationship between the utilities and the objective – plateaus in the decision-making terrain. COUNTIF, and anything involving IF for that matter, similarly adds discontinuity to our model – cliffs rather than hills.

Now let's think about the implications of getting stuck on problems like this. When problems become much more complex – for instance, 100 sites to visit – what do we do? UPS delivery drivers make something more like 150–200 stops in a given day. How do you think the firm figures this out? Not just by eyeballing the landscape visually, though that can certainly help, at least with reality checks. They and we also may feel a bit reluctant relying purely on heuristics like Nearest-Next, which have their own limitations, although they perhaps provide decent starting points for smarter searches. It might be useful to have an alternative mechanism that is more sophisticated and possibly more effective to get to a truly excellent solution.

## **7.2 Genetic Algorithms for Evolutionary Search**

Genetic algorithms represent an alternative to search tactics such as simple hill climbing, and one that can overcome some of the fundamental difficulties that hill climbing encounters in nonlinear and discontinuous landscapes. Genetic algorithms, as examples of evolutionary search approaches, are based on at least two principles that are fundamental precepts in the biological sciences:

- 1) Entities, scenarios, and solutions that do well in a context have traits favored for replication in these contexts ("survival and progeneration of the fittest").
- 2) For performance improvements to develop there must be opportunity to capitalize on diversity. Such diversity is made possible through the intermingling of sufficiently large populations and/or is the result of perturbations (mutations) that introduce novel changes.

These same principles can be used to help analysts develop high-performing solutions for professional practice.

Consider once again the landscape depicted in Figure 7.4. While simple hill climbing alone may offer little help, evolutionary approaches to solution

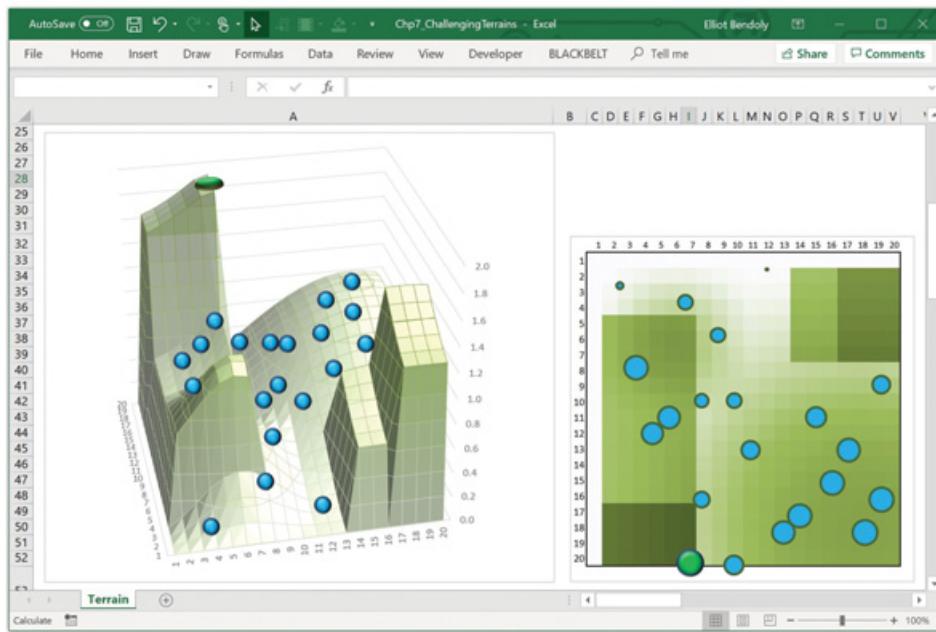


Figure 7.8 Possible initial solution set used by a GA

search, driven by these two principles, are far less impeded by false signals relating to local optima and complex nonlinearities. Genetic algorithms (GAs) begin with not one solution but rather a pool of solution possibilities, an initial sample population in which traits (utility values) differ more or less randomly. Some of these solutions will certainly be better than others with regard to objective performance and constraints. A starting population of 20 example solutions is depicted in Figure 7.8, using the surface plots from earlier as well as a simpler top view via a bubble chart superimposed over conditionally formatted cells.

The next step allows the survival-of-the-fittest concept to kick in by eliminating many of the poorer solutions (perhaps half of them). If our initial set consisted of 20 solutions, this first cut would bring our solution set down to 10. To reinforce the potential for future diversity and development, the algorithm might then temporarily duplicate each of the 10 remaining solutions, bringing the number of solutions back up to 20. I say “temporarily” because the nature of some of these solutions is about to change considerably.

Again, drawing on the first natural principle, a random pairing of each solution with another in that set of 20 (10 pairs of two) provides the foundation of a mechanism by which new solutions can be generated for consideration. The generation mechanism itself requires that some of the utility

variable values get swapped with those of their partner. In the case illustrated in Figure 7.8 this might involve the X values only, with the Y values retained. This swapping activity is referred to as crossover and is akin in nature to passing along a mix of genetic traits to offspring.

The result is not only new solutions with mixed traits (i.e., X and Y values in the graph), but also potentially novel performance characteristics (i.e., Z values) based on those recombined traits (see Figure 7.9). In this depiction the inferior solutions are still depicted though faded, while the superior initial solutions are recombined to create a replacement set of solutions (arrows designate where X or Y traits are used to create new solutions). Here we have only two utility variables in our search, so each child gets just one value from each parent. If we had 100 utilities to modify, then something more like 50 could be inherited from each parent – a major

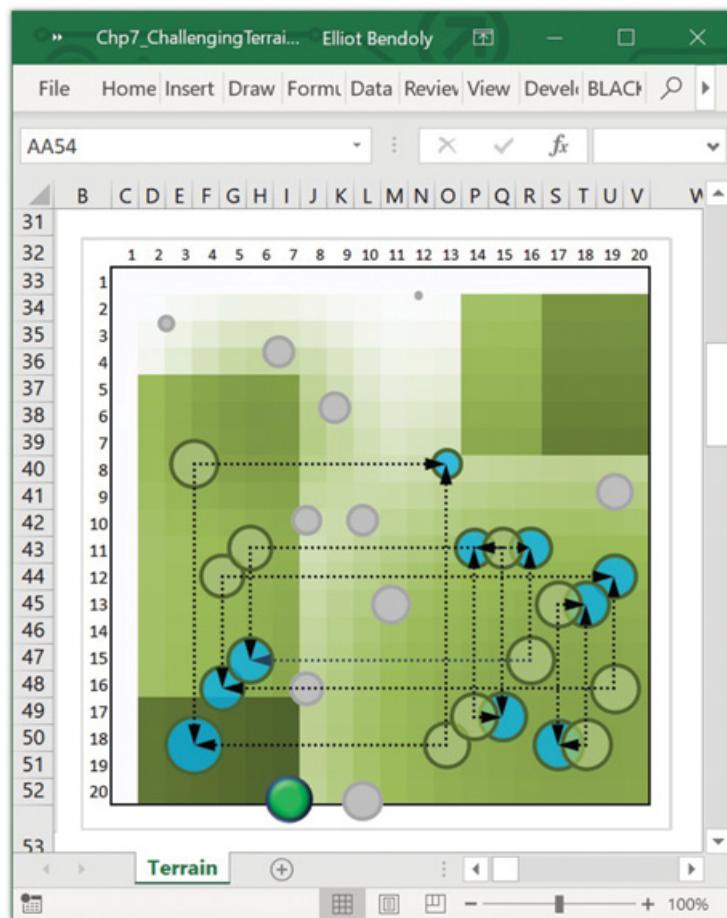


Figure 7.9 Possible first evolutionary iteration by a GA

departure from a single-variable value adjustment common to, and limiting, hill climbing. It allows plateaus to be ascended and chasms crossed.

Overall this process will give us a new population of possible solutions that *may* be notably different from the initial set but also strongly related. Some of the new solutions may be much worse than their parents and some may be equivalent; in other contexts some may violate rules (constraints), but some, however, may be truly better – and that’s the important point because we’re going to repeat this procedure, cutting out worse performers and basing the next generation off the best. If we allow the population to evolve (eliminating the worst and then reproducing and swapping from the best) for yet another generation, we might find further improvements in our solution set, some of which may be very close to a globally optimal solution.

With only a fixed initial population and the crossover mechanism we’re going to hit some limits in the extent to which we can find better solutions. With only 20 solutions our gene pool is fairly restricted. However, even small gene pools are capable of seeking out and attaining globally optimal solutions if we throw in the element of mutation. Along with the crossover mechanism we could also pick a specific attribute (X or Y) in a solution and randomly alter it in a way that creates something that crossover would never achieve by itself. The subsequent solution may put us in a position much closer to the global optimum or further away, but in any case it’s different and its strengths and weaknesses as an alternate line of investigation will become apparent in future generations.

### 7.3 Evolutionary Search Using @RISK

While Solver does have an Evolutionary search engine, even this requires some special added tactics to yield improvements in a TSP setting as well as many others relevant to practice. Instead it is worth seeing what a package uniquely designed for evolutionary search can do. Here, as in Chapter 3, we will leverage Palisade’s suite of tools. With the application running, we will see the appearance of the @RISK ribbon and a variety of analytical tools. In this instance, we are interested primarily in resources that will help us solve complex nonlinear and potentially discontinuous optimization problems, specifically the RISK Optimizer (RO) tool available in this ribbon (see Figure 7.10).

#### 7.3.1 Evolutionary Search on the Traveling Salesman Problem

To demonstrate the use of RISK Optimizer’s genetic algorithm in optimization, we’ll use the second traveling salesman setup for illustration. Make sure this file is open and the Palisade’s @RISK suite active. RISK Optimizer is

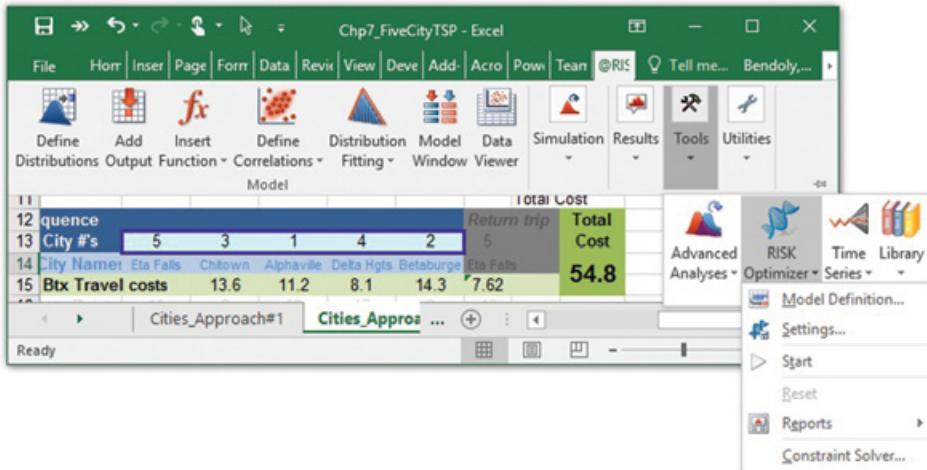


Figure 7.10 RISK Optimizer interface access

found under tools (as in Figure 7.10). When you select the Model Definition option from the RISK Optimizer drop-down, a dialog box should appear with assorted form fields. As with Solver, we need to first say what we want to do. In this case, we want to minimize the total cost (named “Cost2”). And, as with Solver, we also need to specify where the utilities are so those cells can be modified as part of the search process (see Figure 7.11).

With RISK Optimizer, however, we’re given a somewhat more sophisticated interface that allows us to help the computer approach the task we’ve assigned. In this interface we can essentially describe the nature of the utilities and how they might most meaningfully be modified. In this case, we want the search to simply consider alternative sequences or orderings of our nominal variable (city reference numbers). We can specify this by selecting the adjustable cells of interest (named “Sequence”) and then hitting the Group button (displayed in Figure 7.11). The result is a pop-up (Figure 7.12) that gives us the option to declare these variables be handled as a special type in the search.

The Order method for handling variables simply takes the preexisting values in the selected cells and mixes them up in an effort to search for improvement. In this case such an approach guarantees feasibility (no city ever comes up twice, for example). The genetic algorithm’s crossover approach focuses on swapping the relative order of a given set of site visits. For example, if 1–4–3–5–2 is paired with 2–3–4–1–5, the GA might focus on swapping the order of 1, 4, and 3 in each, giving rise to 3–4–1–5–2 and 2–1–4–3–5. Both result in feasible solutions, either better, worse, or equal to the performance of the original parent solutions. Other available solution

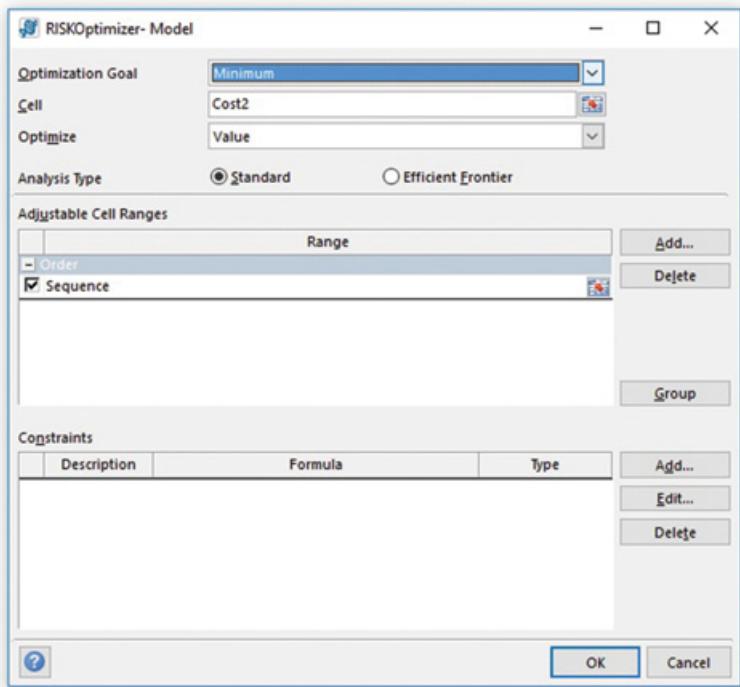


Figure 7.11 Specifying objectives, utilities, and connecting constraints in RISK Optimizer

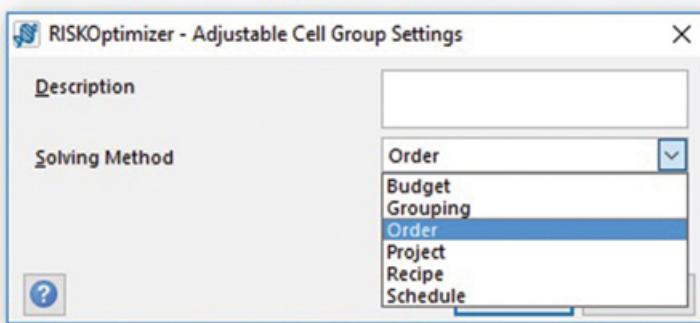


Figure 7.12 Selection of solving method when specifying decision variables

mechanisms are better suited to other kinds of problems. A summary of these methods is provided in Table 7.1.

RISK Optimizer also works in conjunction with other tools in the Palisade suite, including those that permit repeated examination of the impact of utilities on objectives subject to uncertainty in these connections (as might be picked up in predictive efforts in regression). As a result of this

Table 7.1 *Types of solutions available through RISK Optimizer*

Method	Type of Decisions	Special Assumptions
Recipe	Typically for decisions that can take on a continuous or semicontinuous range of values. Common for decisions involving money invested, hours allocated, number of resources used, and so on.	None. Decisions may be varied independently (such that constraint feasibility/costs, for example, bounds on individual decisions).
Budget		Decisions may be varied independently provided the sum of all values is no greater than some specified value (and s.t. other constraint feasibility/costs).
Order	Typically for decisions that take on ordinal or nominal meaning. Order is common for vehicle routing tasks; Project would be common for project scheduling tasks.	Each of the initial decision values (e.g., 1, 2, 3, and 4; or even 3.14, 2.31, and 2.41) is used exactly once in final solution (only order is manipulated).
Project		As with Order, with the additional assumption that some decisions must take on smaller values than (i.e., “come before”) others.
Grouping	Typically for decisions that take on nominal meaning. Grouping would be common to cluster analysis for example. Schedule would be common to appointment or independent course scheduling tasks.	Only the initial decision values (e.g., 1, 2, 3, and 4; or even 3.14, 2.31, and 2.41) can be assigned to the decision variables. Multiple variables will be assigned the same value.
Schedule		As with Grouping, with additional assumptions relating to the maximum number variables that can take on each value and (similar to Project) any applicable precedent constraints.

integration, RISK Optimizer in its search for the best values of the utility variables will try to evaluate each solution it comes up with numerous times (e.g., 100) unless we tell it otherwise. It does so in the hope of simulating a range of possible objective outcomes that the set of utility values might give rise to, in order to estimate some typical performance level for each solution.

Much as in our discussions of Chapter 4, the repeated calculation of an outcome based on uncertain or systematically changing inputs, perhaps even feedback mechanisms, is referred to as an “iteration.” It is not exactly

equivalent to being formally in iterative calculation mode in Excel, which permits circular references/feedback mechanisms to be part of calculations. Rather it's much more like having @RISK hit the F9 key multiple times (and storing average simulated results and perhaps variance) or like having a multirow data table with labels in the left column (repeated evaluation) so that descriptive statistics by column can provide meaning for analysis. If you happen to also be working in Excel's iterative calculation mode, then you can also have circular references active. Just keep in mind that if you are in multiple (N) iterative calculation mode in Excel and are asking for multiple (M) iterations from RISK Optimizer, the total number of calculations per solution examination is  $N \times M$ .

Anyhow, as we'll see in Section 7.4, this is a truly wonderful feature – it's just not one that we need for our current TSP example, since there is no noise in the connection between a selected sequence of visits (our utility variables) and the objective. For a given sequence the total distance is fixed. So for now we can turn off multiple examinations of each solution within the Simulation section of the @RISK ribbon (Figure 7.13). We'll revisit this option later on in problems where the connections between utilities and the object are less certain.

Although there are some additional settings we can tweak at this point, we'll put those to the side for now so that we can dive in and see an example of how RISK Optimizer generally does its thing. Click on the Start RISK

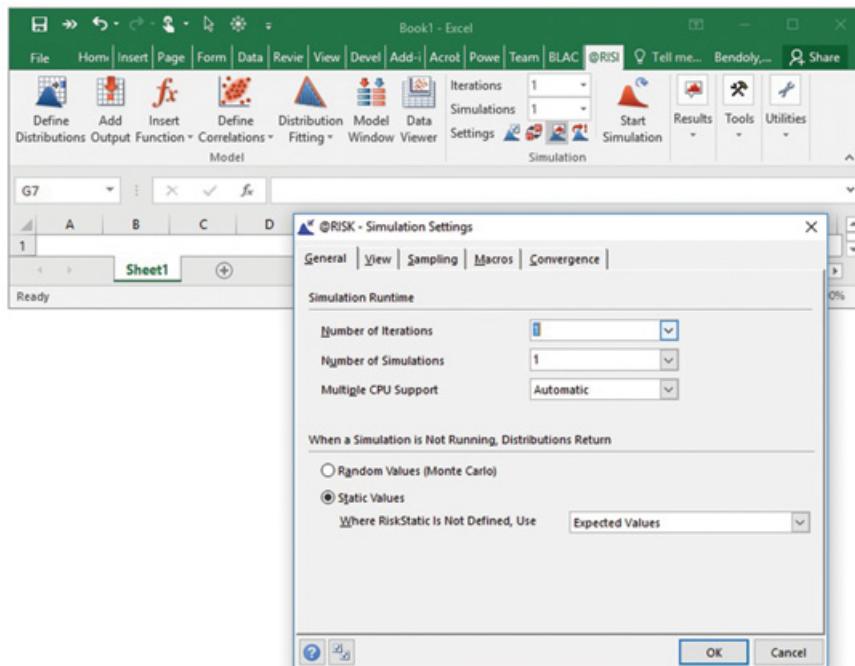


Figure 7.13 Specification of single iteration conditions on search

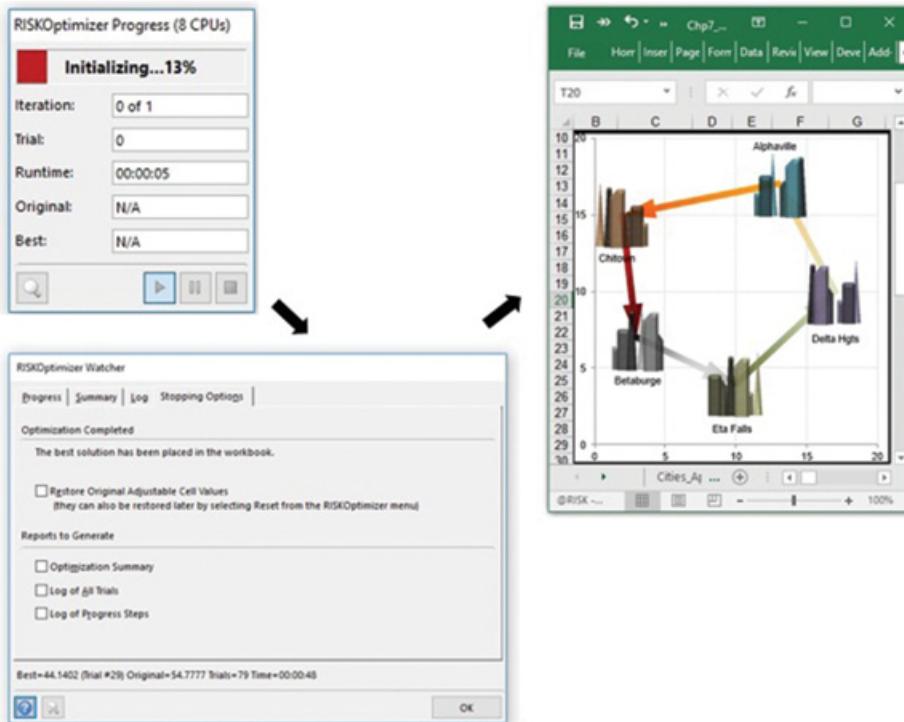


Figure 7.14 Solution derived by RISK Optimizer

Optimizer button (a right-pointing triangle, akin to a Play button). Depending on any extra settings you've put in place, RISK Optimizer may first go through an initialization process before it gets into formal progress in the evolutionary search. When the search ends, a new dialog pops up providing details on the search, an option for returning to the original utility variable values, saving a log of the search, and so on. In this case, a best solution is derived moments after initialization (Figure 7.14).

The one limitation is that RISK Optimizer doesn't always know when to stop trying. Fortunately, if there is a limited number of solutions (recall only 120 in this case), the search will automatically end after these are comprehensively reviewed. In other words we would get the same result if we explicitly requested an examination of only (exactly) 120 unique "trials" (in the terminology of RO). In bigger problems we'll need to use different tactics to end the search. We could, for example, choose an option to stop the search after, say, five minutes (more on that and related topics later on), or we could just hit the little square "stop" button provided. Extra specifications will typically initiate some upfront initialization prior to search.

Another useful feature of RISK Optimizer worth noting at this point – particularly for those interested in getting a feel for how much progress is

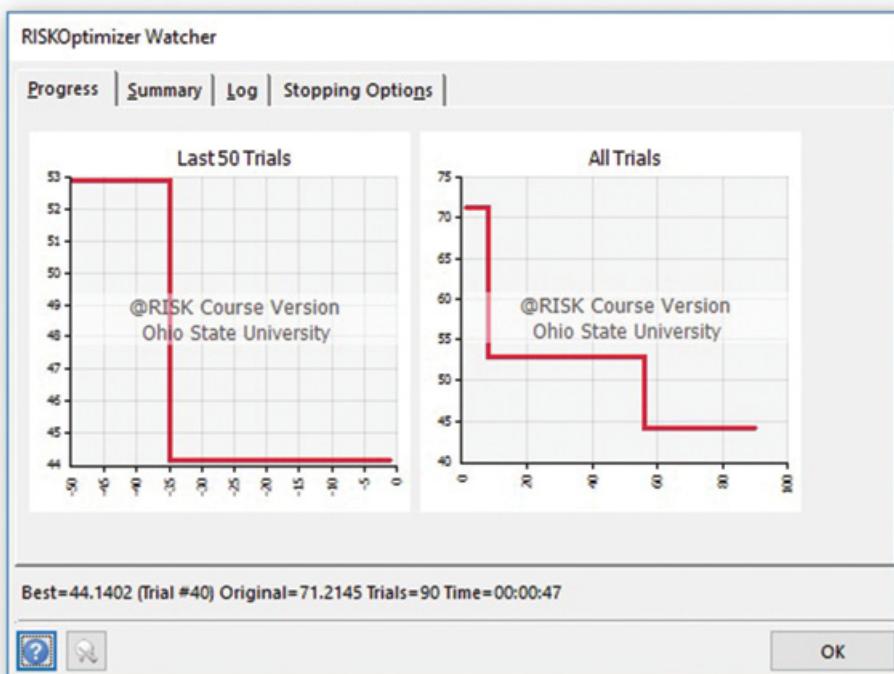


Figure 7.15 Progress graphs created by the RISK Optimizer Watcher during a search

being made toward better solutions as time goes on – is the RISK Optimizer Watcher. When you start a search, the bottom of the progress dialog displays a small magnifying glass icon that opens up this resource. Figure 7.15 provides an example of the visual you might see applied to the current five-city problem, starting from the worst solution and seeing improvement until reaching the optimal. This presents a graphical mapping of the objective function as it evolves with alternative (and better) solutions encountered. Through such a visual representation an analyst may be able to assess whether it's worth continuing the search for an appreciably greater length of time, or if a termination of the search can be made early on.

In a post hoc sense a similar capability is provided by the Log of Progress Steps check box or, more extensively, Log of All Trials, which triggers new sheet presentations of rich data that an analyst might be interested in after the search has terminated. A demonstration of the nature of these data summaries is provided in the next example.

Before we get there though, what about that 100-site problem? If RISK Optimizer can solve a five-city TSP, where Solver gave little support, can it also find a decent solution to the 100-site problem? A problem where a comprehensive search (searching across a universe of car trunks) is

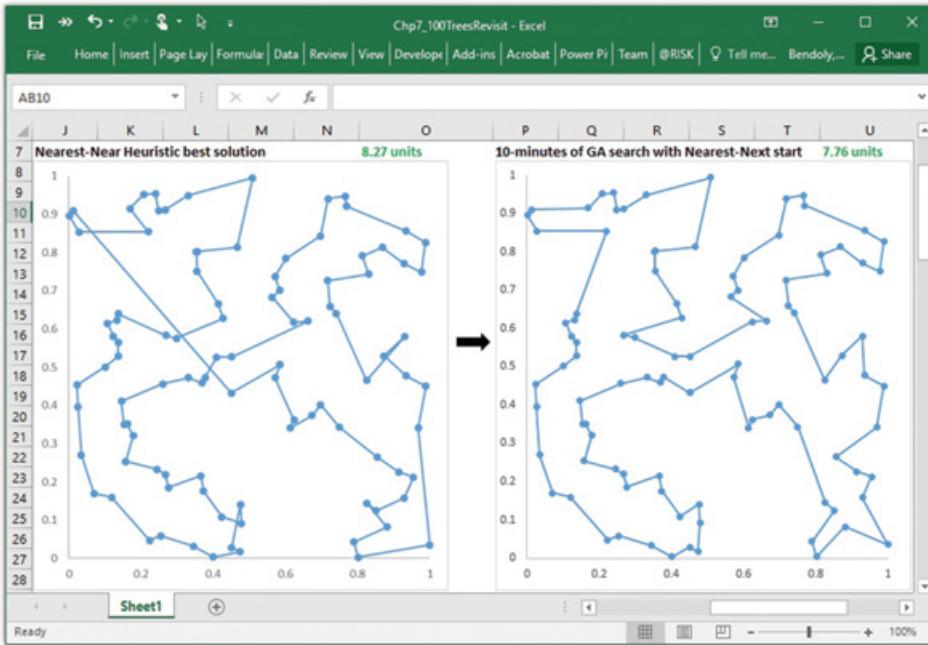


Figure 7.16 From the Nearest-Next heuristic to a Genetic Algorithm solution

untenable? We have at least a couple of benchmarks to compare its performance against. Nearest-Next (NN) provides 100 pretty good solutions. Furthest-Next (FN) provides 100 ridiculously lousy ones. If we're smart and start with where Nearest-Next stopped, can RISK Optimizer's genetic algorithm (GA) approach pick it up and provide important improvements? Perhaps measured by some percentage improved from the Nearest-Next solution ( $1 - \text{GA}/\text{NN}$ )? Or such that the ratio of the RISK Optimizer solution to an upper limit is below a certain level (e.g.,  $\text{GA}/\text{FN} < p$ )?

Let's give it a try. Figure 7.16 provides a before and after shot of the solution derived by Nearest-Next in a 100-site problem (from Chp6\_ArboristTSP, copied into Chp7\_100TreesRevisit), followed by 10 minutes of the RISK Optimizer GA search. Considering the amount of total time these tours entail, a reduction of 6 percent (from 8.27 units of distance to 7.76 units) might be a valuable return for 10 minutes of time.

### 7.3.2 Evolutionary Search on Cluster and Group Development

As alluded to in Table 7.1, RISK Optimizer is also capable of finding solutions for clustering and grouping problems. We've already discussed the potential value of cluster analysis in the Chapter 3 Dodecha case. However, the ability to develop meaningful clusters often transcends

standard assumptions made by statistical programs (i.e., that the best clustering solution minimizes the ratio of within-group to between-group variance). In practice, the objective of our clustering may be much more idiosyncratic to our context and the perceived interpretation of managers. Let's consider an example with a very different sort of objective function.

Imagine a scenario wherein a manager at Dodecha is now given the task of breaking her technical workforce up into four groups, each of which will be responsible for one of four projects. Dodecha maintains an outstanding enterprise and 360 assessment system that provides both objective and subjective assessments of employee capabilities and work satisfaction. It has used this data to create predictions of the technical qualifications and likely interest of its workforce. It refers to these predictions as Ability (A) and Motivation (M), values which differ for each individual and depending on the project assigned.

Given an interest in high levels of both (i.e.,  $A^*M$ ) as well as in keeping group sizes relatively equal (all groups between 18 and 22 people), with Ability levels within each group that do not differ by more than four units between any two members (group dynamics), how should the manager assign the workers? The workbook Chp7\_WorkGroupSelection sets up this problem for us, with the RISK Optimizer model expressed in Figure 7.17. For the time being we will ignore uncertainty in the predictions of Ability and Motivation; the problem is complex as it is.

Unlike the traveling salesman problem that required RISK Optimizer to only consider different sequences of a fixed collection of five numbers (1 through 5) or of the 100 numbers in the larger arborist example, our utility variables in this case can use four values each (1 through 4), with each number used multiple times (between 18 and 22). Because this is not a sequencing problem, I'm going to ask RISK Optimizer to use a different approach to evolving solutions: Grouping (specification shown in Figure 7.17).

We can use MAXIFS and MINIFS (or PercentileIf from the Blackbelt Ribbon) to calculate within-group differences on Ability scores, constraining these to 4 or below. We can also use COUNTIF to determine group sizes to compare to the 18–22 range. Or, if we wanted to avoid this last tactic, we could split the column of 80 integer (1–4) utilities into four columns, one per group. This would leave us with 320 binary (0,1) utilities, and we would have to make sure each row of four values summed to 1 and each column of 80 summed to something between 18 and 22. But that's a lot of effort and extra space given that we're already in nonlinear territory (e.g., with the conditional functions) and we've already hitched our horse to a GA method. Since RISK Optimizer is able to use the Group method to assign things, we aren't doing the tool any favors by switching from 80 integers to 320 binaries. We'll stick with 80.

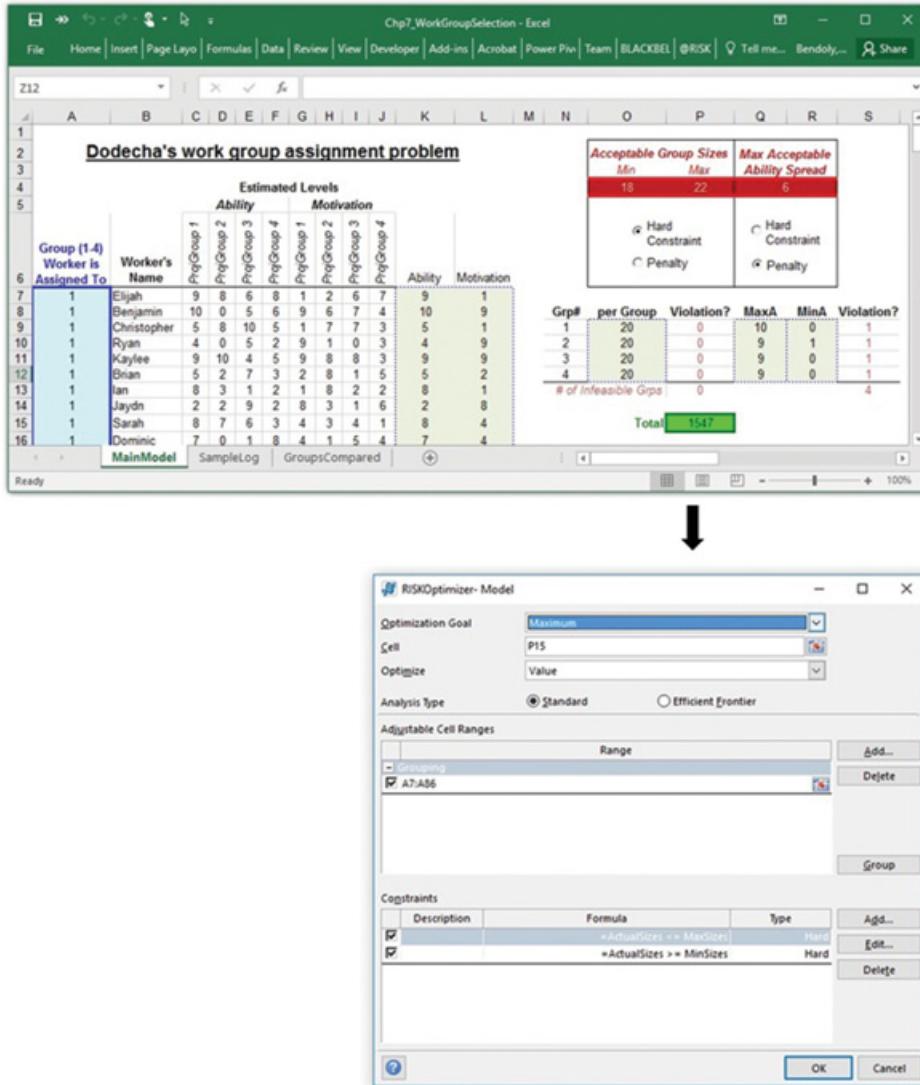


Figure 7.17 Specification of workgroup formation problem in RISK Optimizer

The potential use of a penalty function for group sizes outside the desired range could be an alternative approach to specifying connecting constraints. I've added a radio button to toggle between our more familiar approach to constraints and this one. In this approach penalties become part of the objective function, in the direction opposite to our desired objective (subtracting from the objective, if we are maximizing). If the penalties are appropriately designed, that is, large enough to ensure the desired group sizes, there shouldn't be a need to explicitly designate constraints. Such a tactic can sometimes make RISK Optimizer's search a little easier and is

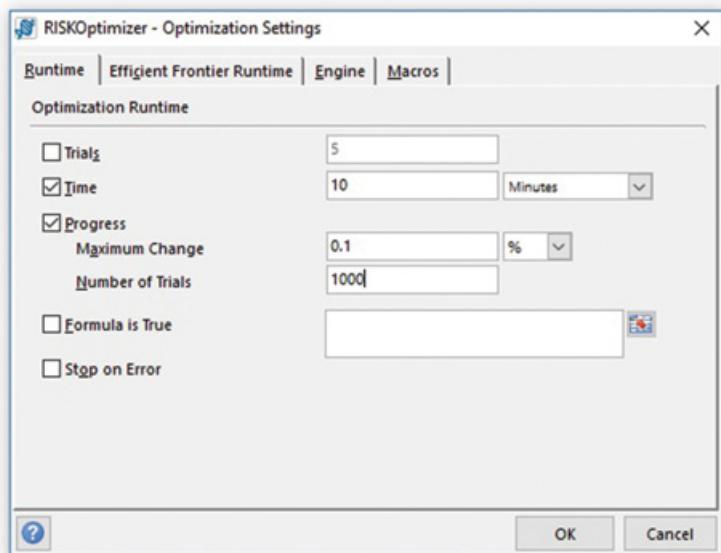


Figure 7.18 Specification of search-stopping conditions in RISK Optimizer

similar to designating constraints as “soft” in RISK Optimizer. In this example I’ll keep the group sizes as a hard constraint and impose a penalty for large spreads in Ability (just so that we can get the search moving).

One additional step, before we start the search, involves once again thinking about how the search will end. We might manually stop the routine, or perhaps we could have been savvy enough to preset stopping conditions as an option of the run. For example, Figure 7.18 shows a designation of the amount of maximum search time in minutes or the degree to which the objective solution appears stable. Here the run is set to stop only when very little (0.1 percent) additional improvement is generated over the last 1,000 solutions examined. If less than that improvement is seen, the search will stop, or after 10 minutes, whichever comes first.

For me, just the first time around, allowing RISK Optimizer to run for about one minute, the algorithm is able to develop a grouping solution with an objective value of 2,743 (versus the original 1,547). After one minute you might see a different level of improvements, since the approach does involve some randomness in the pairing and crossover of parental solution traits. After two and a half minutes I’m at 3,253. My “at least 0.1 percent improvement over the last 1,000 solutions” limit kicked in about a minute later, but without this search limit I was at around 3,383 after 5 minutes. After 10 minutes, at 3,394. Things aren’t changing much over time at this point – around 5,000 feasible solutions were found in those last 5 minutes, and for about a 0.05 percent change in each rolling set of 1,000.

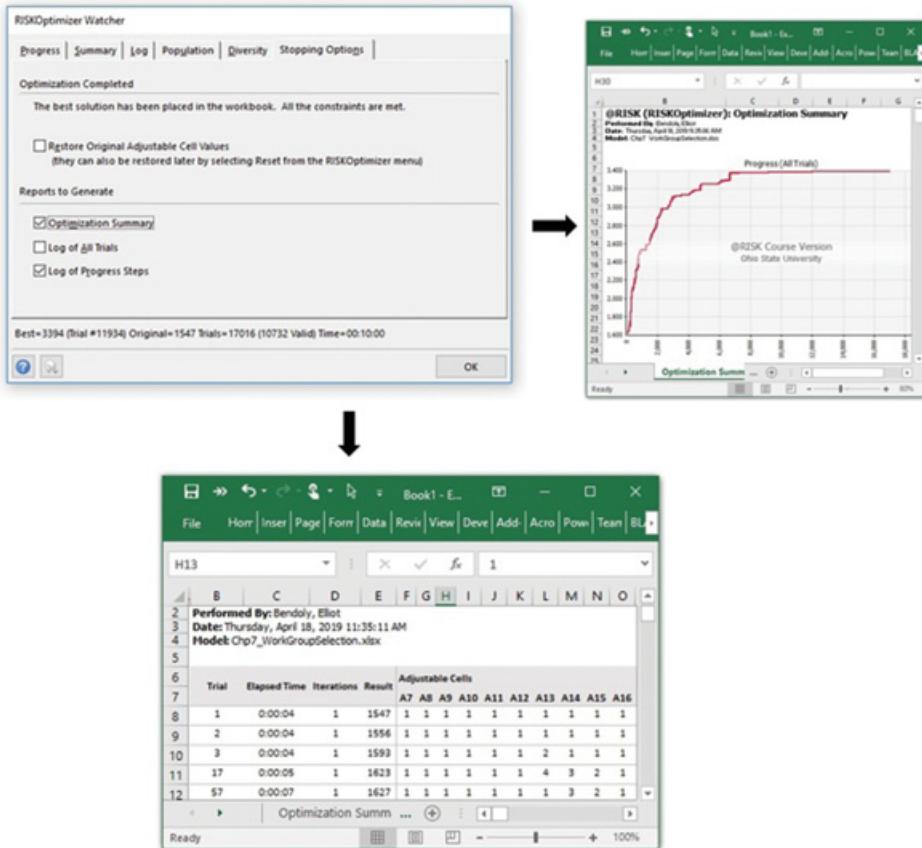


Figure 7.19 Sample output reports available from RISK Optimizer

After any search comes to a stop, RISK Optimizer provides the option of giving you a summary and a log of all solutions it has considered up until that point. This could be useful if you wanted to do further analysis on other solutions that are nearly as good. My recommendation is that you ask for reports only on progress steps (aka improvements) (Figure 7.19) unless you really have a very strong reason for doing otherwise. The number of solutions considered by RISK Optimizer over the course of ten minutes can be enormous, and simply reporting all these solutions (both great ones and very poor ones) is going to take up a lot of time and space, often with little added value in analysis.

This in itself is quite a bit of detail. How could we use it to interpret the effectiveness of our investigation? Well, at the bare minimum, we could use the log file to get a visual impression of how quickly progress was being made on increasingly stronger solutions. We could also use it as an analytical foundation on which to extrapolate how long it would take to get

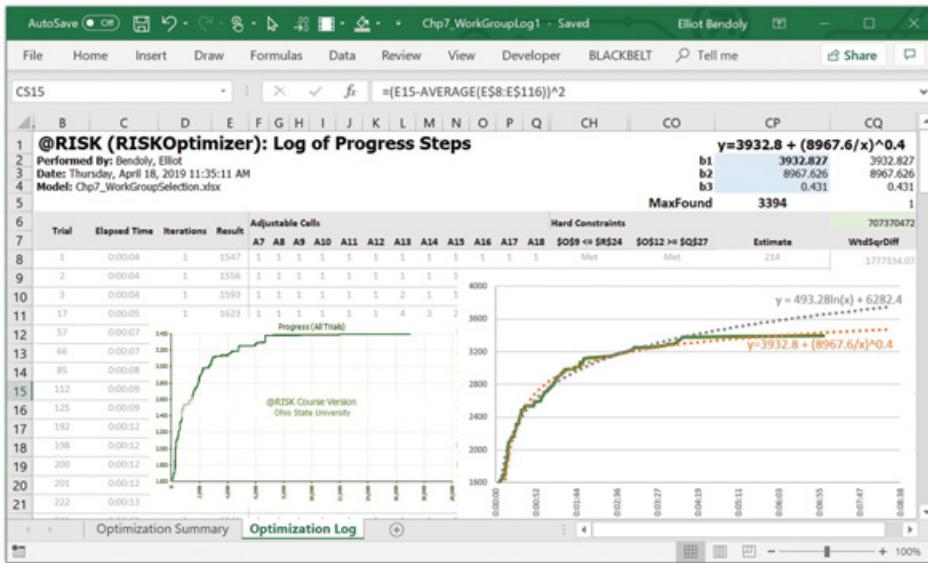


Figure 7.20 Improvements in search over time

a solution that yielded a specific target objective (performance) level. For example, using the Elapsed time and Result columns, I could generate a connected scatter plot to show how much improvement we were getting in the last five of the total ten minutes that elapsed (see Figure 7.20 and the Chp7\_WorkGroupLog1 workbook).

That plot (on the right of Figure 7.20) will look a lot like the graph of performance improvements versus solution number (which RISK Optimizer has already provided, copied in Figure 7.20 on the left), since each solution takes about the same amount of time to examine in this case. In the plot I've generated I've also added a best-fit logarithmic curve (pretty good) as well as a negative power curve estimate (a better fit to the more recent observations). The latter was derived through the use of Solver's evolutionary search mechanism to estimate coefficients with the aim of minimizing the sum of square differences between a fit and the observed data, weighted toward more recent observations.

A decent further use of Solver's GA; Analysis on analysis, to inform further analysis. As we've seen, regression is optimization, after all. The negative power curve is a bit more meaningful in that it presumes an upper bound on performance, estimated to be around 3,932 (15 percent greater than where the 10-minute search ended). An additional hour of search might get us a 10 percent improvement. A full day of additional search might get us to a 14 percent improvement. But from a practicing manager's perspective, these are bound to be long projects, for which any staffing decisions will be

persistent. A day of prep in analysis could be worth it. Extending the length of the examination is ultimately the analyst's call, informed by details of the context. There is virtue in weighing the trade-offs between up-front work such as this and the bigger picture of how decisions are implemented.

### **7.3.3 Evolutionary Search on Schedule Development**

As a last example of the type of complex problem for which RISK Optimizer might provide analytical strength, consider the task of developing a schedule that outlines the sequence and potential simultaneity of the work required for large-scale projects, such as the technology implementations that Dodecha manages.

Such projects typically consist of a series of discernible steps, many of which cannot be started before work on other steps has been completed. The availability of individual project workers is an added complexity. In many cases some workers cannot easily handle more than one step at a time. However, because of the costs of transferring project experience from one worker to another, it is often beneficial (and sometimes necessary) to make sure that certain sets of steps are handled by the same individual. The implied human resource and organizational issues, and complex constraints on the decision-making process coupled with potential nonlinearities in the value of getting a project done quickly and with a high level of quality, make project management a complicated duty (see Figure 7.21).

The spreadsheet within which these facts are applied when developing relationships among the utility variables (to whom to assign project steps and when to start them) and the objective (how much total time should be expected for project completion) should also contain the mathematically codified forms of all relevant constraints (i.e., both precedent and “no double-working” rules). An example workbook that captures this is provided in Chp7\_ProjectScheduling. The main decision interface with a sample start solution is shown in Figure 7.22. In this case, in the RISK Optimizer model we're specifying two kinds of utility variable search methods: Recipe for the worker assignment, and Schedule for the start dates.

Keep in mind that even RISK Optimizer may be sensitive to the nature of starting solutions, especially when so many possible solutions and discontinuities exist. The more thought you put into your initial solution based on what you know as a planner, the better RISK Optimizer (and similar algorithms) will serve you – as was the case when we jump-started the 100-site TSP problem earlier with the Nearest-Next heuristic upfront. Even keeping the worker assignments from Figure 7.22 (which are entirely arbitrary 1–5 repeated), we could use the earliest start time estimates provided as input to a “more savvy” initial solution for RO's genetic algorithm to work from.

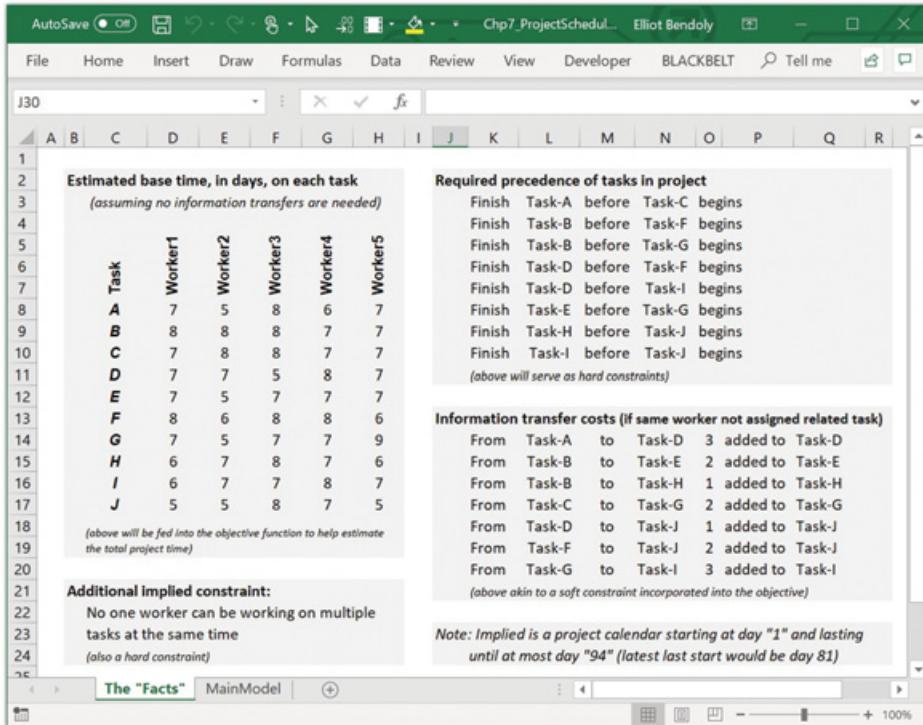


Figure 7.21 Facts and rules critical to the project scheduling problem

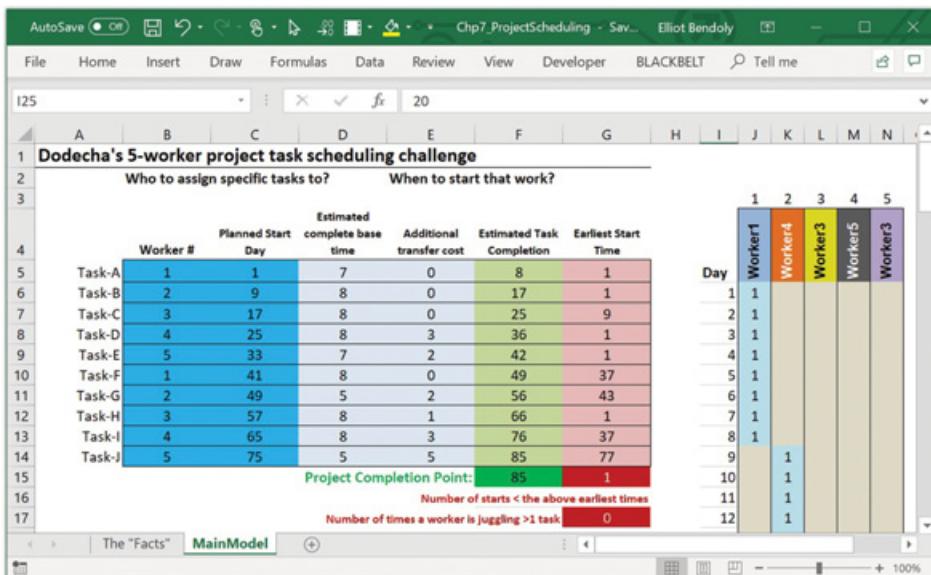


Figure 7.22 Setup for project scheduling optimization

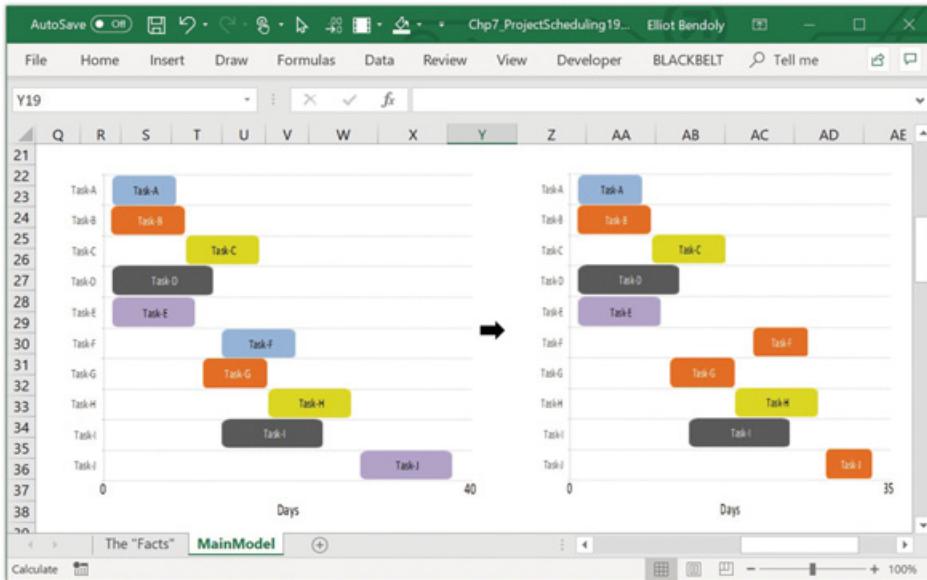


Figure 7.23 Before RISK Optimizer versus after 6.5 minutes of search

In any event, and even starting with a reasonably intelligent starting solution, RISK Optimizer is still able to provide an improvement – from a starting solution’s project completion time of 38 days to a project completion time of 33 days after about 5 minutes of search time. To help visualize any meaningful changes in the solution I’ve also included in this workbook a modified Gantt chart, common to project management (here just a stacked bar chart with the first stack transparent). For contrast Figure 7.23 visually compares the savvy start to the resulting schedule. The benefit provided is pretty striking: a 5-day return on 5 minutes of search.

As a final reminder, as valuable as graphical depictions can be, and we’ve seen several post hoc examples, their presence during the search can appreciably increase processing time. If you want RISK Optimizer (or any other routine) to quickly run through a large number of solutions and calculations, you might consider postponing graphing until after every search has been conducted.

### 7.3.4 Genetic Algorithm Options in @RISK

Now that we’ve seen the tool in action, it’s worth looking a bit deeper into the options available to us in these kinds of searches. RISK Optimizer uses both *crossover* and *mutation* in its search for globally optimal solutions to complex business problems. To make use of these mechanisms, and in accordance with the conceptual nature of genetic algorithms just described, such a tool

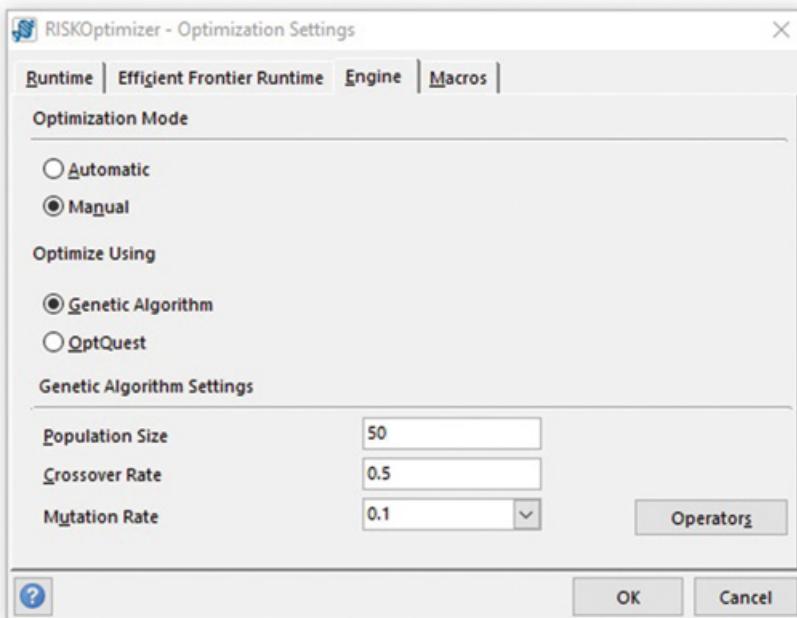


Figure 7.24 Specifying solution population size in RISK Optimizer

will need to have a set of alternative solutions available from which to draw and develop next-generation solutions. Analysts can specify the size of this pool, as well as the crossover and mutation rates, in the RISK Optimizer Settings dialog box, under the Engine tab. Once Manual is selected, the interface appears as in Figure 7.24.

The specification of the size of the retained population (the gene pool) can have a significant impact on the effectiveness and efficiency of the search. Too small a pool can impair the development of novel superior solutions because only a limited set of existing ideas are available upon which to draw. Too large a pool can relate to the retention of too many inferior solutions that can similarly distract from effective searchers. So, what's a good population size? This is likely to differ for each and every problem type. The problem is that, early on, most analysts won't know what that size is and they learn only through the experience of running similar optimizations repeatedly. However, the makers of RISK Optimizer do suggest a population size of 30 to 100 for most problems, with bigger populations relevant for bigger problems (i.e., those with large numbers of variables and more complex relationships between the utility variables, connecting constraints, and objective function).

As for the nature by which the solutions in this population are used in the search for better solutions, RISK Optimizer also provides the means by

which to specify how certain utility variables are mixed and matched by the GA and then potentially subjected to mutation during the search. For any set of utility variables (the settings can differ for different sets), these options are available when the variable manipulation methods (“groups”) are initially specified or when users elect to edit them in the RISK Optimizer interface.

The crossover rate specified in RISK Optimizer can be anything between 0.01 and 1.0. For any two solutions being used in crossover to generate new offspring solutions, a crossover rate of 0.85 would mandate that 15 percent of all utility variables involved in a composite solution be substituted by other existing utility variable values (from the partnering parent solution) during crossover. In contrast, a crossover rate of 0.5 suggests that only half of the existing variable values will be retained. A crossover rate of 1 essentially equates to zero crossover (the generation of new solutions is supposedly handled predominantly through mutation instead).

The mutation rate is also modifiable and can be specified as anything between 0.0 and 1.0. A mutation rate of 1.0 specifies that any individual utility variable value in a composite solution involving multiple utilities be subject to random modification for the generation of new solutions. A mutation rate of 0.5 suggests that half of all decisions are subject to random mutation, while 0.0 relates to zero mutation in new solution development. The abundant use of a mutation rate also relates back to the specific selection of population size. If new solutions are being generated largely by mutation rather than by crossover, the need for large populations is reduced. (Random mutations allow even relatively small genetic pools to evolve.)

#### **7.4 Simulation Optimization Basics**

Population size, as well as the consideration of crossover and mutation specifications, requires still more attention in increasingly complex problem settings. When searching for ideal solutions in landscapes that are not only structurally complex but also fundamentally uncertain, we clearly benefit from considering a wide range of possible outcomes for any given solution. As discussed in Chapter 4, any one randomized assessment of a model involving uncertain connections can provide a misleading picture of the effectiveness of a policy. Therefore, it is critical to guide our methods and tools (e.g., RISK Optimizer) in the consideration of such a range of assessments when conducting a formal search for optimal solutions. Increasing the number of iterations specified in a tool like RISK Optimizer provides exactly this kind of capability.

### 7.4.1 Simulation Specifications in @RISK

There are at least a couple of ways to specify how many repeated examinations of a model and its outcomes to conduct for a given solution set (i.e., for a given set of utility variable values). The simplest approach is made available directly through the @RISK ribbon, under the simulation section (see Figure 7.25). This is the same place we have been going up to this point, switching the number of iterations from 100 to 1 (since the examples in Section 7.3 did not involve models of uncertainty or system dynamics). We'll want to reconsider that choice in the presence of simulation models.

As a side note on the use of multiple iterations for evaluating policy performance that includes uncertainty, one convenient mechanism to ensure policy comparability across even a small set of iterations is that of the Random Number Seed field (found under the Settings section of the Simulation area of the @RISK ribbon). An analyst can specify that the same set of random numbers is ostensibly drawn in evaluating each new policy decision under consideration during the search (see Figure 7.26).

By specifying the All Use Same Seed drop-down option, the analyst can have greater assurance that RISK Optimizer will be making apples-to-apples comparisons with regard to the conditions under which each possible policy solution is being judged. Typically, this is a default setting, but it's one worth paying attention to, particularly when a relatively small set of random number draws are being specified during solution assessments.

Alternatives to specifying a discrete number of iterations for the evaluation of each policy considered (e.g., generate 100 possible outcomes associated with each decision considered) include the Stop on Actual Convergence and Stop on Projected Convergence options. In cases where you might not know how many iterations to run to get a stable characterization (average) of performance, you can actually let RISK Optimizer try to figure it out on its own. This takes a little control out of the hands of the

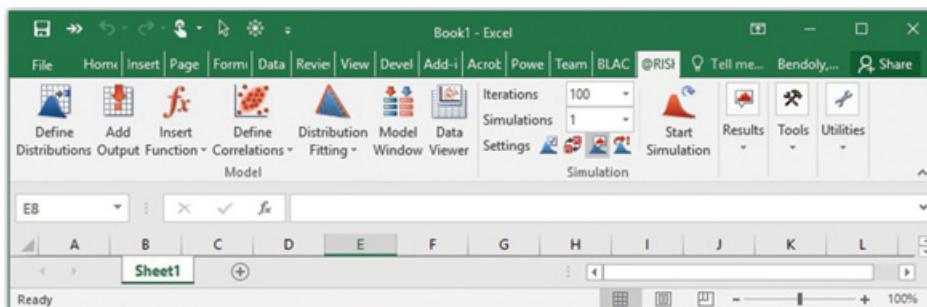


Figure 7.25 Simulation stopping conditions: How many examinations of each solution

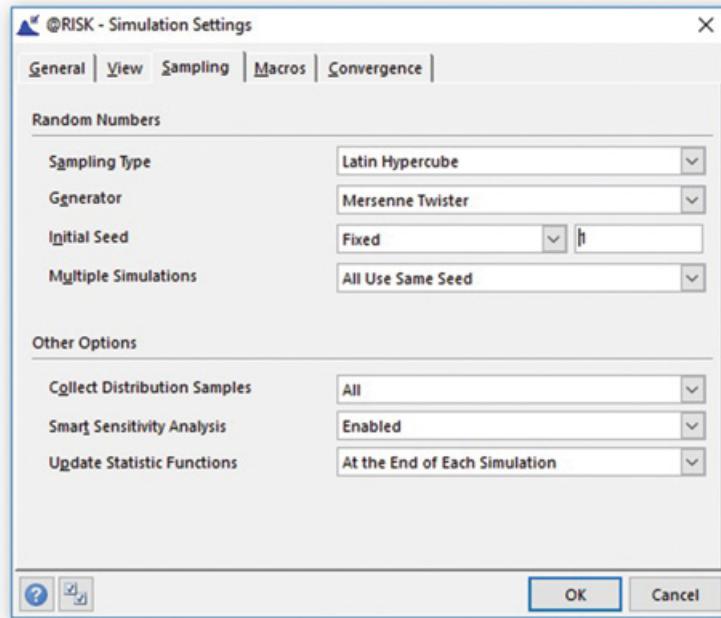


Figure 7.26 Specification of random number generation conditions

analyst, but it can be helpful if the number of iterations needed to assess certain policies is appreciably greater (and unknown) relative to some more-simplistic policies.

#### 7.4.2 Using Averages from Data Tables in Optimization

As a first example of how RISK Optimizer might be applied toward seeking out best solutions to simple simulations, consider once again the Lobos Reservations example. In the Chp7\_LobosReservationsOpt1 workbook we have a 500-row version of the second data table from this problem's Chapter 4 discussion temporarily embedded in the model sheet. For simplification no additional Data Table calculations or graphics have been retained. In that Data Table, as before, performance is tabulated for a single overbooking policy in terms of both earned direct profit, parties turned away, and empty tables. Five hundred scenarios are simulated and averages tabulated. If we were interested in finding only a policy to maximize expected profit, subject to a service constraint (e.g., on average, only overbooking at most by one party one third of the time, or an average  $<0.33$  turned away), with the standard deviation of empty table counts limited to 1.75 (to reduce over-staffing risks), we might end up specifying the optimization search in RISK Optimizer as shown in Figure 7.27.

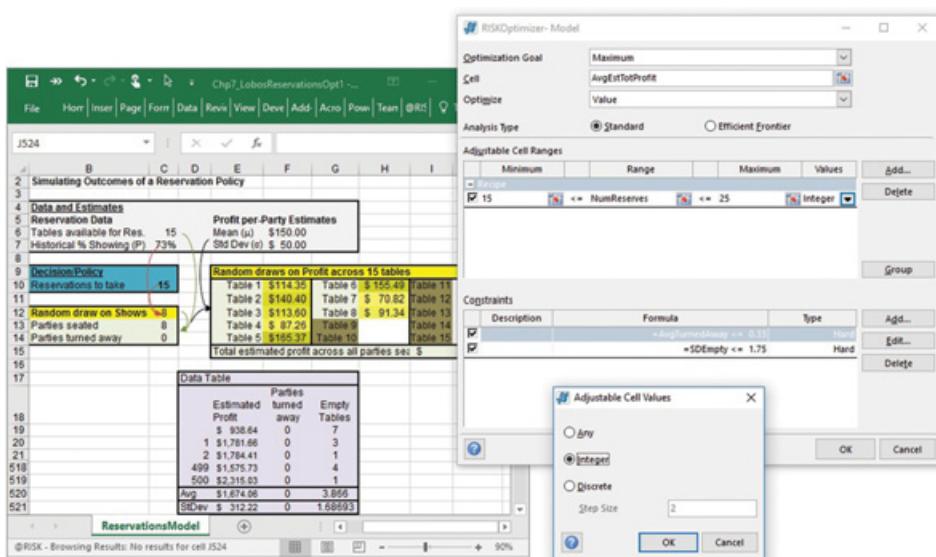


Figure 7.27 Problem specifications for simulation optimization of reservation problem

Because our iterations are basically being covered by the data table, we won't request any more than a single RISK Optimizer-based iteration per decision policy evaluated. After initiating the search, it doesn't take long for RO to examine all of the policies between 15 and 25 bookings considered – a fairly limited landscape to examine. The best result typically seems to be a reservation policy of around 19 bookings, less than what simply working with averages might have lead us to (e.g.,  $15/(73\text{ percent})$ , or more like 20 or 21). We could confirm this through manual inspection as we did with the five-city TSP. Critically, however, if RO can get us here without manual perusal, it can also tackle much more complicated versions of this problem (e.g., those in which parties occupy tables for longer periods than anticipated, or parties are given the opportunity to be waitlisted for subsequent periods where tables may open up).

As in Chapter 4's discussion of this case, closer inspection of the details of each option, here made available through the log of all trials (only 11 solutions examined), suggests that the solution of 19 bookings along with neighboring solutions – for example, 17, 18, or 20 bookings – are fairly comparable. Certain aspects of risk are more severe for some of these. Ultimately tolerance for risk itself can pose challenges, but that doesn't mean risk should be ignored, rather, just thought about a bit more deliberately.

### 7.4.3 Using RISK Optimizer Iterations without a Data Table

An alternative approach to generating the same result is to capitalize not on data table functionality for developing simulated scenarios (i.e., iterations in RISK Optimizer lingo) but instead on RISK Optimizer's built-in iteration mechanism. To accomplish this, we need to modify the source of the performance and constraint information drawn into RISK Optimizer and we need to specify to RISK Optimizer that we would like for it to run five hundred iterations for each solution considered, just as the data table had been providing (see Figure 7.28 and Chp7\_LobosReservationsOpt2). Because data tables won't be necessary in this approach, those associated sheets should also be deleted from the workbook to reduce the amount of processing required in each step.

It is worth noting that to get the constraints to apply equally in this case, we will need to specify that we are interested only in meeting the service constraint "TurnedAway on average" (i.e., across all iterations of each policy assessed). We could be more stringent, but our search might yield different results and wouldn't be a fair comparison given that the data table approach was basically just looking at satisfying the constraint, on average, across its trials. We can make this modification to the service constraint by asking that the Simulation Statistics of mean TurnedAway for each policy (across all iterations) be considered rather than the final value (of each iteration, or the last one viewed). In a similar modification, the Empty Table variation constraint will focus on standard deviation

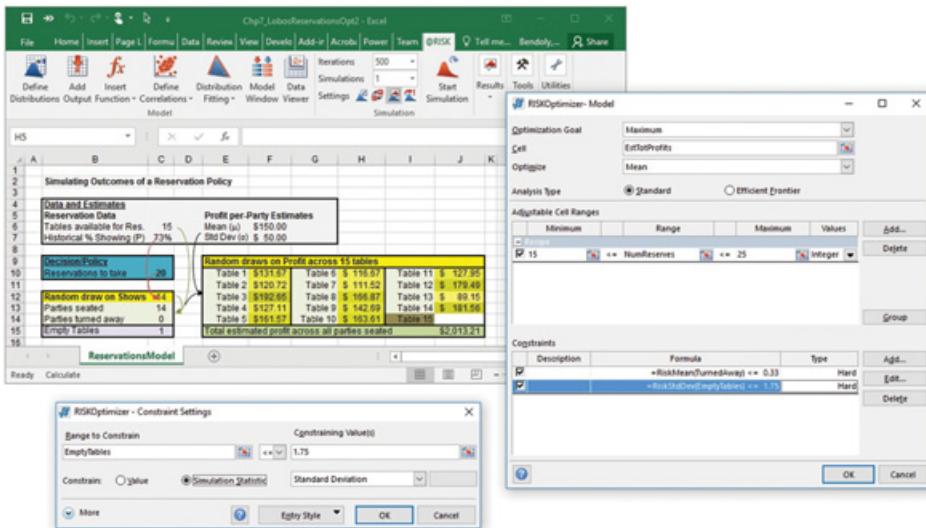


Figure 7.28 Alternative specification leveraging RISK Optimizer iteration capabilities

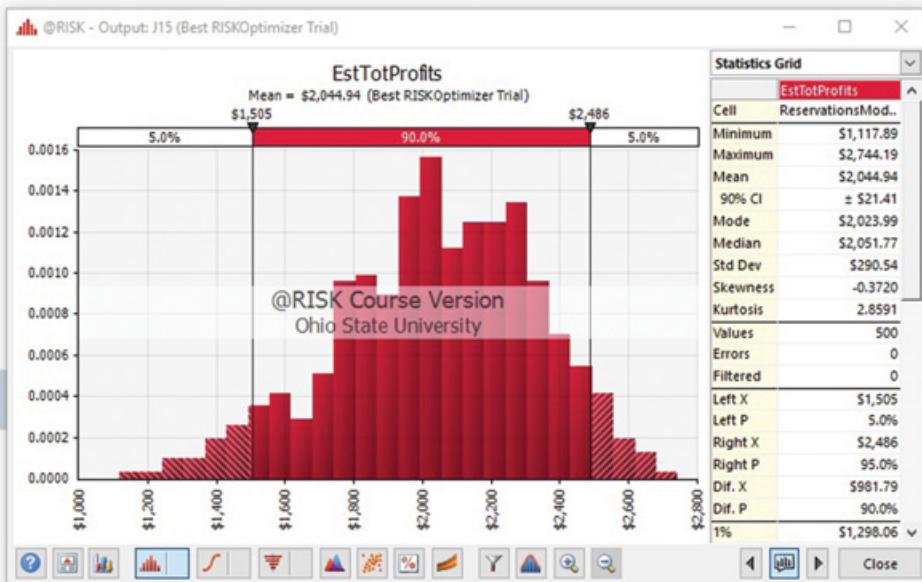


Figure 7.29 Descriptive statistics on simulation examinations by RO in search

limits rather than any one scenario. These specifications are already made in Chp7\_LobosReservationsOpt2 (see Figure 7.28).

Since RO is doing the simulations for us in this case, it will also present the descriptive statistics around each solution it encounters in the form of a histogram during its search for the best solution found (see Figure 7.29). Ultimately, the result of an otherwise preferred policy of around 19 bookings is comparable to that derived by the other method. A close inspection of neighboring results, as before, shows them to be similar as well. It's a limited search and thus a fast one; however, RISK Optimizer's handling of iterations may tend to converge upon this conclusion notably faster than the data table approach.

## 7.5 Optimization for System Simulations

Now that we have seen how RISK Optimizer can work with simulation models, with data table summaries as well as in place of these summaries, we are ready to consider more complex simulation examples where combined tactics may prove useful. For this, let's now reconsider the Lobos Inventory example from Chapter 4. The most notable difference between the Inventory example and the Reservation example was the fact that the former's policy under consideration (the utility variable whose optimal value is being sought) has an impact that explicitly triggers events that play out

over time. One could argue that a reservation policy implicitly impacts table availability in a future period, we just didn't manifest that dynamic in our example. In contrast the reorder point policy is explicitly triggered by inventory levels that dip below its threshold, and explicitly triggers the placement of replenishment orders, the initiation of a shipment, a period of shipping transit (during which stockouts can occur), and a replenishment of inventory – all of which is manifested in the Lobos Inventory model.

We've also seen a couple of approaches to evaluating this manifested system. In our original discussion we looked at a model for this system in single-iteration calculation mode to simply accommodate the circular reference needed (*Chp4\_LobosInventory\_BasicModel*). In that first approach a macro was recorded such that striking Ctrl-h would reset the model and run 200 periods worth of evaluation. Resets are critical here, since each new reorder point policy (utility value) must be independently evaluated over time without contamination of results from prior policy evaluations. In the second version of this model we entered into a more complicated iterative calculation to permit all 200 periods of evaluation simply with each hit of the F9 key, with an automated "system reset" in place (*Chp4\_LobosInventory\_200iteration+DT*).

In that second approach we included a data table approach to summarize the 200-period performance of a discrete set of reorder point policies. As with the reservations problem, we may feel that the latter case provides enough information for us to derive a prescription. But, as earlier argued, in anticipation of more complex problems, with many more complex decisions to be made (e.g., ordering policies across a range of goods, sharing purchase and shipping costs, joint storage constraints), we need to be familiar with how prescription derivation can be automated here as well.

To recap, there's a lot we need to do to fully automate prescription development in this case, and there is more than one approach that can be taken in manifesting and explicating these prescriptions. In line with the OUtCoMES cycle, and with some understanding now of what goes into simulation optimization, Figure 7.30 provides a revision of the flow chart presented in Figure 4.20.

According to this figure three looping mechanisms are not uncommon in system simulation optimization. The first of these evaluates a single system scenario (a solution defined by a set of utility values) over as many periods as might be necessary to capture system dynamics. The second of these repeats such evaluations from scratch to avoid the risk that such system dynamics are skewed by chance. The last is the search for improved solutions (alternate utility variable scenarios) and is the optimization that we typically depend on tools like RISK Optimizer to manage for us, unless we are in the very fortunate position of examining all possible solutions comprehensively, in

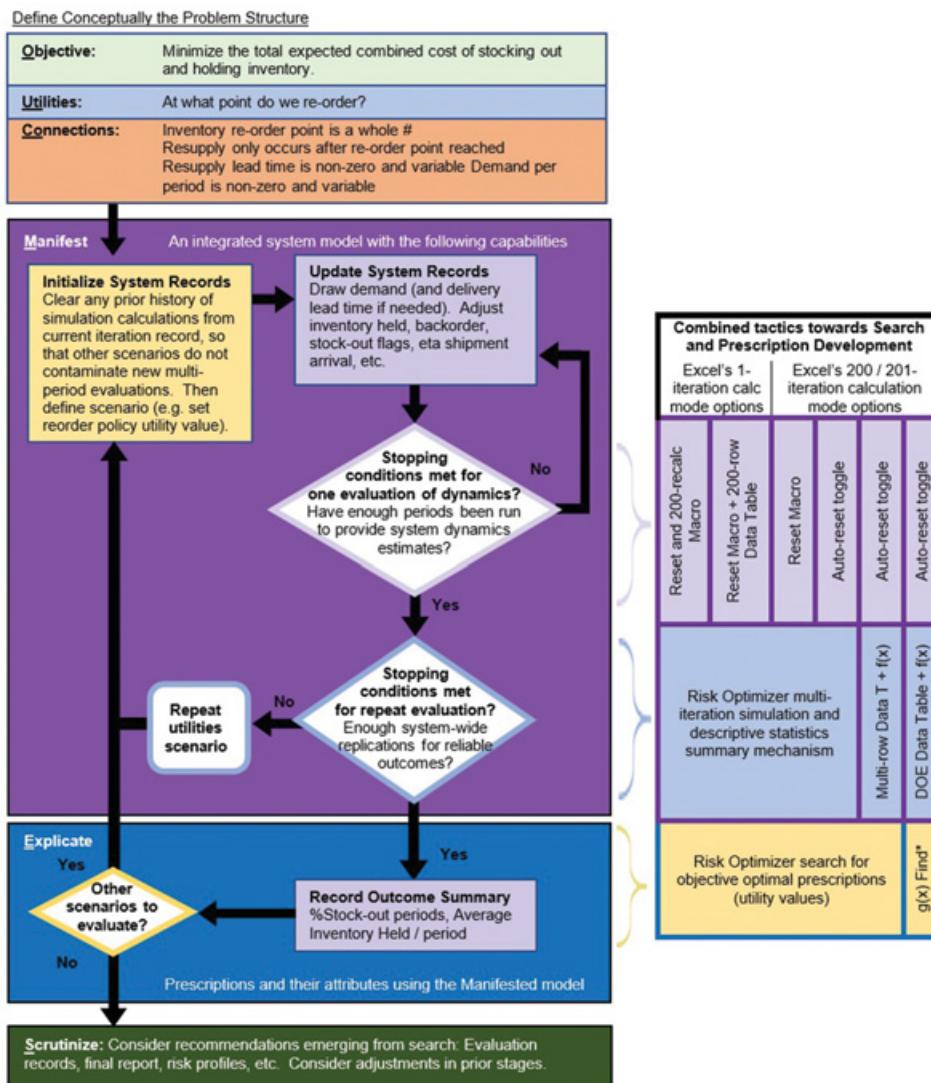


Figure 7.30 Flow and combined tactics in system simulation optimization

which case functional examination (use of MAX or MIN, and MATCH overall results) may suffice – often not a luxury we have.

### 7.5.1 RISK Optimizer with Excel's Single-Iteration Calc Mode

Fortunately for us, RISK Optimizer not only has the ability to search for good solutions and execute repeated calculations when needed, it can also help manage these investigative loops by doing something very important: call macros. And that's the kind of thing we would need if we wanted to use our

first approach to system simulation (Chp4\_LobosInventory\_BasicModel). To reiterate, here's what we are working with if we follow this approach as is (for now):

- 1) Running in single-iteration mode in Excel Options
- 2) Evaluating each solution only once (simulation runtime iterations = 1 in RISK Optimizer)

To make the setup in RISK Optimizer as simple as possible, let's define a single objective that integrates the two costs represented in this system: holding costs and shortage costs. Let's just assume for now that every unit of inventory held per period, on average, costs the same amount as each additional percent chance of being out of stock (hence expediting and/or losing goodwill). We assume that all those who want these goods will ultimately get them, albeit not instantaneously, via backorder. They won't give up on us; they'll just get angry.

To manifest this as part of the overall model let's create a cell named NetImpliedCost equal to AverageHeld + FractionOut\*100. We'll use that sum as the objective function we're aiming to minimize. Because these two components of the cost function trade off against one another and are nonlinear functions of our policy, it is likely that the combined objective function will be nonlinear as well, with considerably high costs appearing at both ends of the reorder-point policy spectrum (another trait well suited to the use of RISK Optimizer).

#### *7.5.1.1 Searches Calling a Reset and Multicalculation Macro*

What we specify to RISK Optimizer as the objective, utilities, and connecting constraints might be something as simple as shown in Figure 7.31, assuming we have reason to suspect the best policy exists somewhere between a reorder point of 40 and 120 and that assumptions around reorder quantity determination need no reconsidering. Following the first approach in the table in Figure 7.30, we'll begin using a version of one of the Chapter 4 files in the form of the Chp7\_LobosInvBasicFullSimMacro\_Opt1 workbook. In this workbook, operation in single-iterative calculation mode in Excel, we have Net Implied Cost calculated, and the macro we developed to reset the system and examine 200 periods remains available on a Ctrl-h strike.

As for actually getting that macro to run, we aren't actually going to be striking Ctrl-h repeatedly (that convenience is just for testing at this point). Instead we need to make sure RISK Optimizer itself calls our macro when needed. In fact, we'll want RO to call our macro every time a new policy is considered. In the parlance of RISK Optimizer, this means calling this "Before Each Simulation Begins." It sounds a bit odd, since the macro is essentially driving the simulation of results for a possible policy, but the term

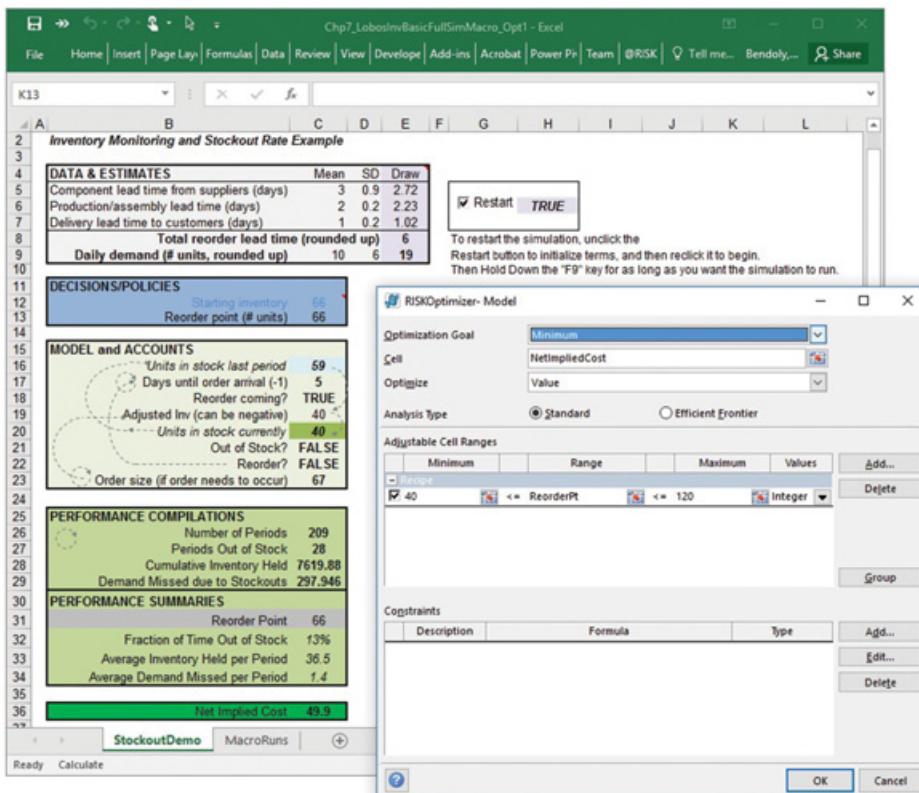


Figure 7.31 RISK Optimizer model specification for the reorder point problem

“Simulation” in the RO form is equivalent to “next solution under consideration” in this case. We find this option in the @RISK ribbon under the simulation section (set to 1 iteration). The Settings option gives us access to macro calls tied to RO examinations of solutions (Figure 7.32).

Figure 7.33 shows an example graph describing the nature and variability of the relationship between the reorder-point policies considered and the objective function (Net Implied Cost), spelled out in the logs of this search. Given the objective and everything else specified in the system model, the best reorder-point solution suggested by RO was just shy of 60, at 57 – that is, inventory reorders placed whenever the inventory level reaches or goes below 57 units. But hold on a second. Before becoming satisfied with this recommendation, take a close look at Figure 7.33. There’s a lot of noise around these policy performance estimates; a cloudy band that seems to otherwise follow something of a parabolic form. That should raise some red flags regarding the appropriateness of the final recommendation.

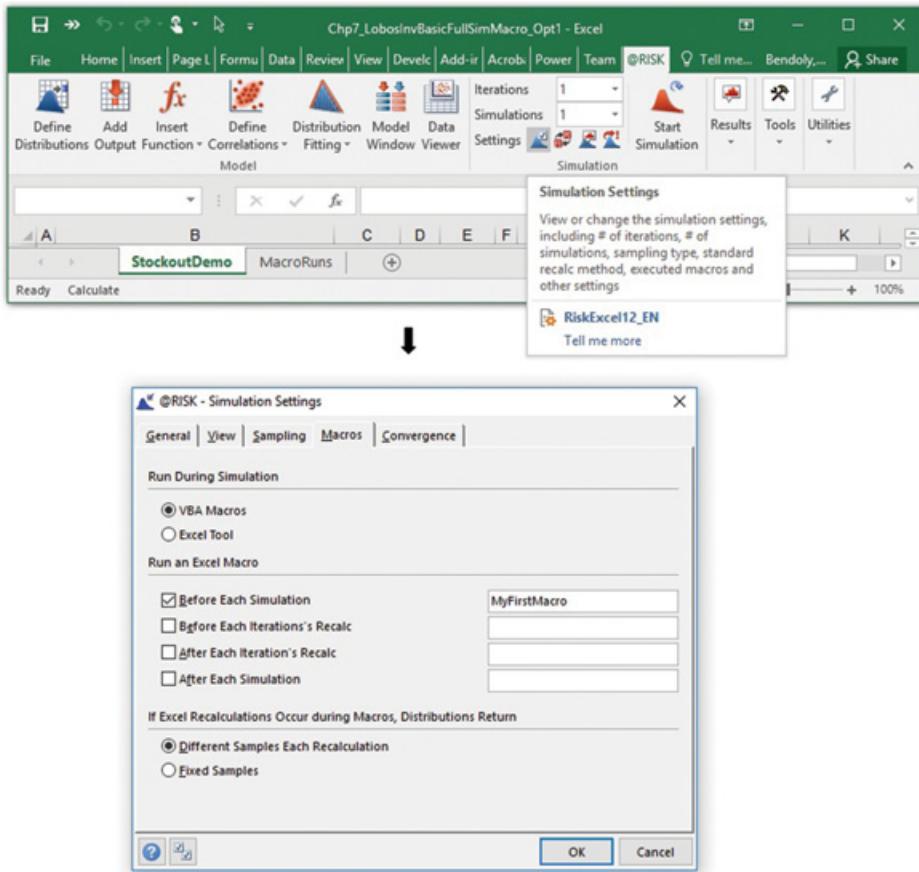


Figure 7.32 Specifying a macro call in RISK Optimizer before each solution is evaluated

### 7.5.1.2 Refinement through RO's Multi-iteration Mechanism

Now, that noise is actually meaningful. It, in part, gives us a sense of the uncertainty around the average performance of each reorder point policy considered. However, it makes comparisons more challenging. It makes it harder for us to have faith in RO's recommendation, which, after all, is based on these point estimates. Lucky for us, there are ways to neaten things up, if the goal is to minimize based on more robust central estimates of cost.

What's needed for this is a repeated consideration of each of the possible reorder-point solutions (that second loop in Figure 7.30), which takes us beyond simply looking at each solution over a single 200-periods. It permits looking at each solution multiple times (each time over 200 periods). After all, something really unlucky, or lucky, might have happened at the beginning of any one of those 200-periods that completely threw the system off.

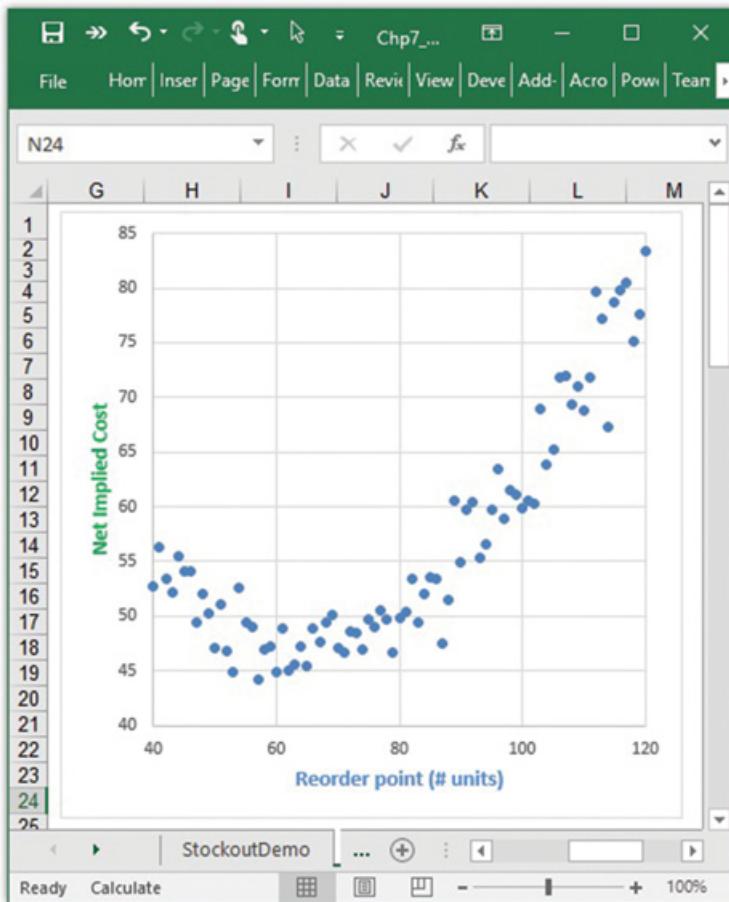


Figure 7.33 Reviewing nonlinearities inherent to the solution space encountered

Because of this, we want all three loops outlined by Figure 7.30: (1) a multiperiod examination of each solution to allow for system dynamics to play out; (2) multiple reexaminations of each of these solutions; (3) a search that will consider multiple alternative solutions to our decision-making problem, or multiple simulations in the terminology of RISK Optimizer.

RISK Optimizer already handles the third by design, and our current macro handles the first loop. The second loop can also be handed off in a variety of ways. Our first approach will be giving this task to RISK Optimizer as well. Only a couple of modifications are needed. First, we will need to change the simulation settings from 1 to some larger number: 10, or 50, or as many repeated examinations of each policy's 200-day evaluations as we feel are necessary. We can even ask RO to continue reexamining until little change in estimates' descriptive statistics take place (a nice feature of

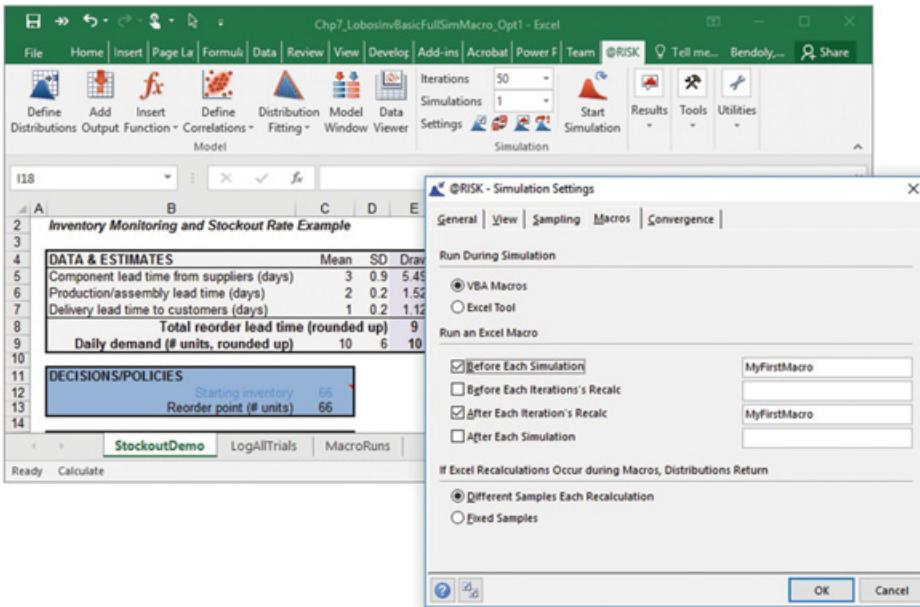


Figure 7.34 Use of RISK Optimizer iteration and macro calls for repeated system simulation

RISK Optimizer's simulation tools). Apart from this we will also need to make sure that our macro is called not just when a new policy begins to be considered but also at each of these reassessments – as in Figure 7.34, both before each new simulation (in the terminology of RISK Optimizer) as well as after each subsequent Iteration recalculation. To ensure that these various reexaminations are considered in policy comparison and search, we must also change the consideration of the objective from “Value” (in Figure 7.31) to “Mean.”

Now, if you are asking for 50 repetitions of each 200-day policy evaluation, things will take about 50 times longer to run. But you will benefit from a bit more clarity. The result of this setup in Excel’s single iteration mode should look considerably leaner in terms of variance than the previous results. But with the summary information provided by the Log of All Trials we have the added benefit of being able to create approximate box-whisker plots of performance distributions across policies.

Specifically, since RO is doing the repeated simulating here, for each 200-day evaluation of each policy it is collecting mean, min, and max performance data along with standard deviation summaries. Even if we don’t get 25th and 75th percentile data points, we could approximate those with the mean and standard deviation, if we can assume the observations are close enough to following bell curves (i.e., Normal). With that information in

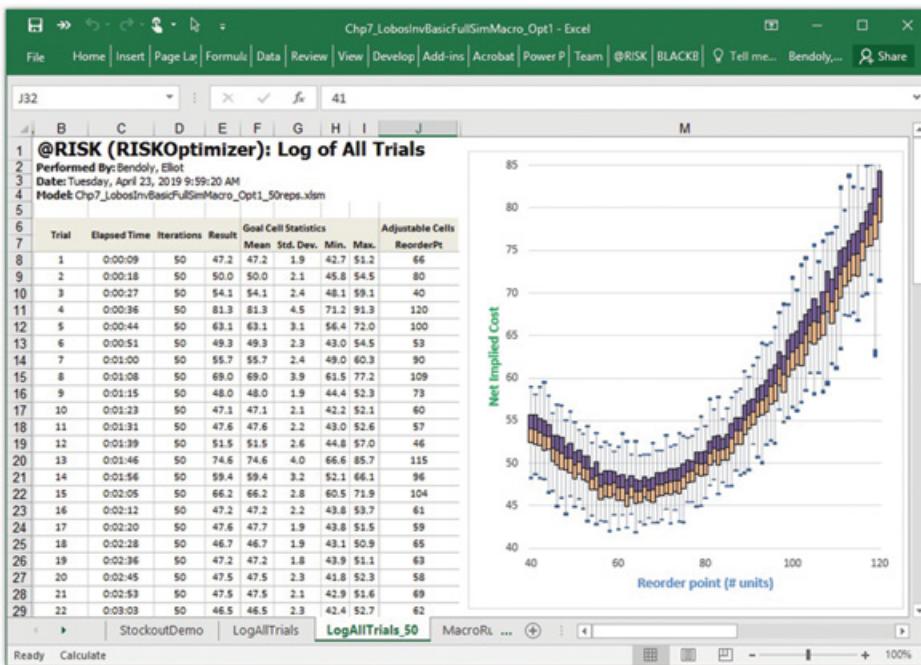


Figure 7.35 Improved estimation in a multievaluation, multiperiod system simulation search

hand, we could get a more convincing picture of the value of specific policies relative to others and relative indifference within a specific range. A visual summary of all of this might look a bit like the depiction in Figure 7.35.

Although the nature of these searches cannot guarantee that optimal solutions are pinpointed, we can gain insights into the goodness of the best solutions encountered relative to others encountered. Graphical illustrations of the relative strength of solutions based on search logs can be compelling exhibits in reports to higher-level decision makers. They emphasize the salience (or lack thereof) of differences between alternative competing policy options. Graphs that include representations of risk, as in Figure 7.35, provide further critical insight to decision makers in these contexts.

If the need arises, we can develop yet additional descriptive insight into recommended prescriptions post hoc through tools we've already seen: data tables and macros. Imagine recording in a macro the actions involved in developing a data table that described numerous intermediate and final performance measures associated with a policy (RISK Optimizer will generally provide only summary measures on the Objective). We might not need to have such a table live and active in each step of the search (it might slow down run-time), but it might be something nice to have as one of the final artifacts describing the best policy reached (such a macro might be called "At End of Optimization" by

RISK Optimizer). With a table like this, any variety of descriptive details can be presented with regard to the option, including its upside and downside, as well as the dynamics of intermediate performance measures.

#### *7.5.1.3 Searches Reliant on Data Tables*

An alternative approach in single-iterative calculation mode could involve eliminating the 200-period evaluation of our macro, reducing its task to simply that of a system resetting tool (we will dive more into the code associated with this change in the next chapter). The 200-period evaluation could after all be handled, less elegantly, through the use of a 200-row data table, with the final row of that table representing the final 200-day average. Essentially each subsequent row in the table advances us one period closer to our destination, but we really only care about the destination (that final row), not the trip itself, hence my reference to this approach as a bit inelegant.

Nevertheless, in Chp7\_LobosInvResetMacro+DT\_Opt2 we do just that. It's representative of the second method of the Figure 7.30 table. RISK Optimizer would still need to call a macro, a much less involved one in this case. And RO would still be a useful resource for repeated examinations of that last data table row tabulation (to generate good policy descriptives for comparison and search). And in the end the search results should be comparable. But what is gained through perhaps a little more transparency isn't the best use of spreadsheet real estate.

#### **7.5.2 RISK Optimizer with Excel's Multi-iterative Calculation Mode**

If we shifted only slightly from our Basic model toward a 200-iteration mode in Excel, what would we do, or have RISK Optimizer do, differently? We would still need a macro that could reset our system for each policy evaluated, even though each evaluation (for RISK Optimizer) would still involve the 200 periods desired (any F9 hit would as well). The setup in RO therefore won't be much different from our setup leveraging the multiperiod macro; the macro itself would just be a lot simpler.

##### *7.5.2.1 Searches Including Calls to a Reset Macro for 200 Iterations*

The approach outlined in the third column of the method table in Figure 7.30 begins with this move to a 200-iteration mode in Excel, and a simplified macro that only flips the switch for the system tabulation off and then on. It's a lot like using a data table, but no additional space is taken up for calculation and storage (Chp7\_LobosInvResetMacro\_200It\_Opt3). Each examination of a solution by RO needs to just call this macro in this multi-iteration Excel mode (Figure 7.36). The associated search and its results will ultimately be comparable to that of the first two approaches, albeit potentially less efficient.

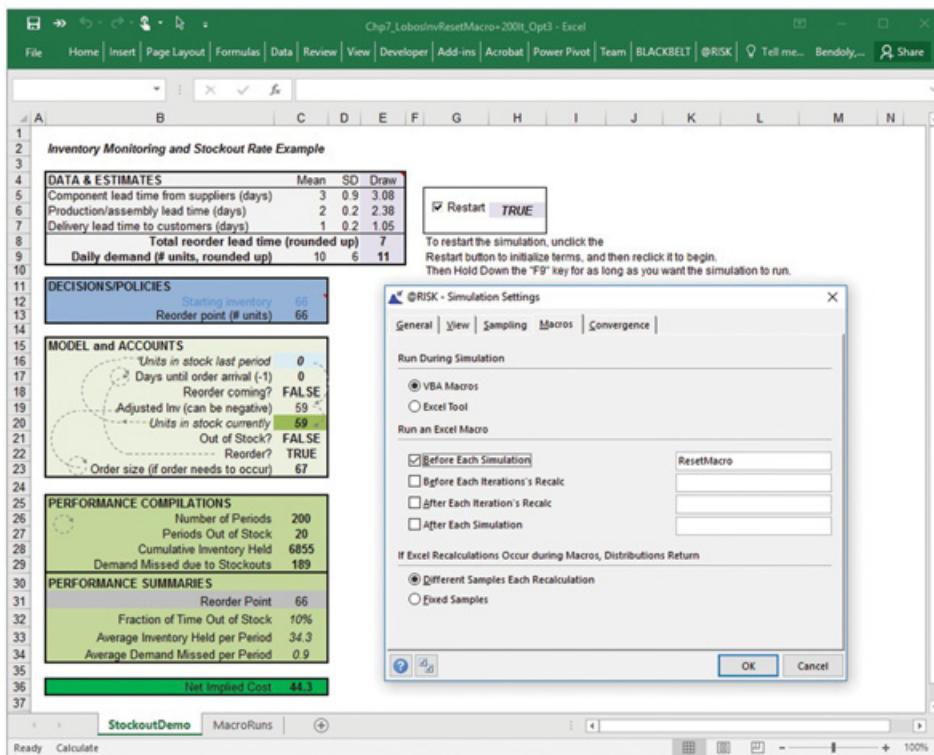


Figure 7.36 An alternative search approach using Excel's multi-iteration calc mode

### 7.5.2.2 RO Search Using an Auto-reset Toggle

The fourth and fifth columns in Figure 7.30's table of combined methods suggest two related approaches that do not make use of macro calls. Clearly, RO can take care of repeated examinations for us; however, in the absence of a macro call we still need a way to reset our system. Fortunately, we have one. We introduced it as an auto-reset toggle in the workbook Chp4\_LobosInventory\_200iteration+DT. That workbook operates in multi-iteration calculation mode, with each initial iteration serving to reset the system and the remainder of the iterations taking us through the 200 periods desired. Thus, each hit of F9 both resets and executes a fresh 200-period evaluation.

How can this be of use to us in our RISK Optimizer searches? There are at least a couple of possibilities. The first (column 4 of Figure 7.30) involves giving RISK Optimizer the same task as in the previous example, absent the macro call. After all, it's no longer needed since the system is resetting itself after every 200 periods. We will still want RO conducting repeated evaluations (10, 50, etc.) per policy, but the system

dynamics simulation work is all taken care of by the spreadsheet. The workbook Chp7\_LobosInv200ItAuto\_Opt4 can be used as a foundation for this kind of approach as well as the next with some tweaks.

The second approach (column 5 of Figure 7.30), also macro-free, involves the addition of a data table to this workbook to generate repeated evaluations. In RO we won't need to request multiple evaluations, since the DT is doing that work, and we can return to allowing RO to consider "Values" rather than "Means" for comparison (no real impact on processing). Descriptive statistics of the repeated 200-period evaluations, provided by simple spreadsheet functions (e.g., AVERAGE) can then provide performance comparisons and the search landscape for RISK Optimizer. Once again, the results should be comparable to those described up to this point, although the continued presence of a large data table in multi-iteration Excel can get a bit tedious as it might slow other edits down. The same caveat is representative of our final approach, as we'll see.

#### *7.5.2.3 Evaluation and Selection Including Use of an Auto-reset Toggle*

Before we end our discussion on simulation optimization it is worth emphasizing a method that could be used on this fairly limited search landscape that does not make use of a search engine like Solver or RISK Optimizer. It's the last columns of the table in Figure 7.30. What it requires is a comprehensive evaluation of the solution space, which again is possible only if the landscapes are in fact small enough (i.e., not something like 100 factorial). In this case we have just been examining 81 reorder points – small enough for a comprehensive evaluation. In the auto-reset toggle model that we've looked at, it is possible to create a multievaluation data table, 81 rows deep and  $N$  columns wide (where  $N$  is the number of repeated 200-day assessments).

In the workbook Chp7\_LobosInv200IterAuto\_DT+Find, 10 columns are used for these repetitions per policy (Figure 7.37). The resulting averages are plotted directly but also serve as the basis for a search for the minimum net implied cost policy. Noise can still pose an issue, but it can be further reduced through iteration, best fits established, or ranges of policies targeted. Ultimately it's not elegant and not robust to noncomprehensive searches, but it can serve to confirm the value of the other methods we've reviewed.

### **Supplement: Leveraging Solver's Evolutionary Search Engine**

Before we close, let's consider Solver once again. Although we have focused on Solver's hill climbing approaches, an evolutionary search capability does exist in Solver. In the absence of a tool like @RISK, and when

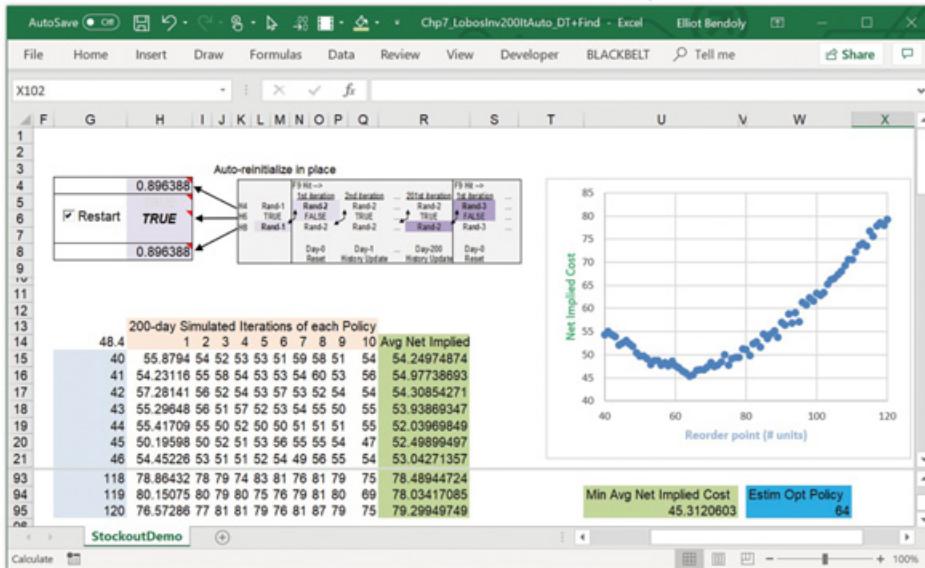


Figure 7.37 Results leveraging an auto-reset toggle and data table

comprehensive searches are not possible, this highly accessible engine can actually help us get through some of the challenges that we face in these complex terrains. It may take some finessing – occasionally unorthodox finessing – but if we set things up correctly, we can even retrieve some of the search history and conduct time and value analyses as we considered with @RISK.

The retrieval of search history itself is somewhat straightforward given the availability of the Shadow tool in the Blackbelt Ribbon add-in. However, similar standalone code can also be developed (as in the Chp7\_30TSPSolverEvolv workbook). In this example we have a 30-location TSP, hypothetical drone deliveries in downtown Chicago. The starting solution is lousy with lots of back-and-forth crisscrossing (more than 12 standard units of travel distance). The NearestNext\_TSP function from the Blackbelt Ribbon whittles this down to 5.47 units of distance in less than a second. But Solver's evolutionary search engine can get there as well and a bit beyond (5.15 units after 15 minutes), if you're lucky – which is pretty good and competitive with RO's “from scratch” improvements over that time. There's a better solution with a <5.08 distance out there, but it's a nice result regardless. Figure 7.38 presents the manifestation of this model along with a visual history playback of the route (utility values) and the improvements in the objective.

The luck we can't do much about, apart from increasing the population pool size (1,000 used here) and increasing the amount of time spent without

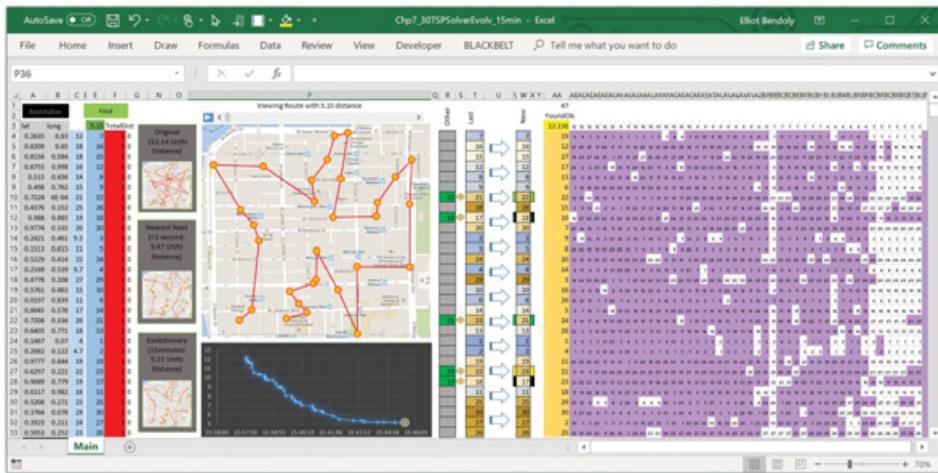


Figure 7.38 TSP model adjustments to leverage Solver's Evolutionary engine

improvements (both of which increase the odds of finding improvements eventually). A little random mutation (0.1 rate) can help as well, although too much can spoil the richness of the solution pool. The real trick is getting Solver's evolutionary approach to make modifications to the utilities in a way that ensures feasible solutions.

Since this Evolutionary tool doesn't have a special search approach like "Order," as RISK Optimizer does, we'll need to try to force order on whatever it comes up with. And that's not that hard to do – after all, any set of thirty values, whether whole numbers or otherwise, can be sorted and replaced with rank-order values (perhaps with a tiny bit of random noise added in to guard against tie-breaking). It's one of the only good uses of rank transformations from continuous data and in fact is only useful because the underlying data modified by Solver in this case isn't meaningful on its own. Here we are actually making lemonade out of lemons, where most commercial ranking efforts sadly tend to do the opposite. As with all methods and results of analysis presented, understanding the "how" and "why" is fundamental to applying (or ignoring) the "what."

## PRACTICE PROBLEMS

### *Practice 7.1*

Recall the application of cluster analysis to the Dodecha Solutions case. One of the implied goals of the clustering algorithm was to minimize the ratio of within-group to between-group variance, subject to the constraint that we were interested in distinguishing a total of four groups. The criteria for grouping (and the source of variation)

included composite factors that were derived through Principal Components Analysis (PCA) conducted on an original set of 32 items. Recognizing the semirandom nature of the grouping process, attempt to replicate the groups derived using RO and your own custom criteria. Using the data set from the Chp5\_DodechaSolutions workbook, show how the resulting groups compare across the performance measures.

Can a similar story be told? Given our previous discussion in this chapter regarding the implications of random features in the grouping process, what might any significant differences imply for an analyst attempting to discriminate among project types? How much do you think these differences might be dependent on your choice of grouping criteria? On the time spent in the search?

### *Practice 7.2*

Recall the issues of randomness brought up in the clustering discussions of Chapter 5; the same kind of randomness inherent to XLStat's search for groups applies here. RISK Optimizer isn't providing a complete search for more than  $4^{80} = 1.46 \times 10^{48}$  solutions, just an evolving series of smart guesses.

Try to make a major manual modification to the initial solution while making sure that group sizes remain between 18 and 22. Look at the best solutions log and then compare it to that derived in the example solution presented in this chapter. Are similar objective function values obtained? Does convergence seem to occur much earlier for one-start solutions?

Try to develop a plot of best solutions at each of the 10-second intervals with your solutions on the y-axis and the example solutions on the x-axis (i.e., organize the data so that best solutions at each 10-second interval are outlined for comparison purposes, then line up and plot the solutions corresponding to each interval). How could this depiction help to describe the relevance of starting solutions?

Regardless of group number, how similar are the group constituencies derived at the end of the two initial solution approaches? An example of how to depict this might involve the use of a scatter plot for four separate series of data (designated by the groups derived from the first best solution). Group membership from the first approach might be graphed against that from the second set of runs, with a little "jitter" added in (not graphing Person1's first group# versus Person1's second group#, but rather something more like  $x = \text{first group\#} + (\text{0.5-rand}())$  and  $y = \text{second group\#} + (\text{0.5-rand}())$ ). The result gives an impression of the population at each combination of results, something along the lines of that shown in Figure 7.39. Try it and describe what you see.

### *Practice 7.3*

Recall the project scheduling problem from Chp7\_ProjectScheduling. In reality, the time to complete specific tasks can only be estimated and is automatically viewed as associated with some level of uncertainty. In each of the cells where estimated completion times are being tabulated (D5:D14), introduce an additional term  $+\text{RANDBETWEEN}(1,4)$  to allow for the simulation of such uncertainty. Use a RISK Optimizer procedure similar to that originally applied in Chapter 7 for this

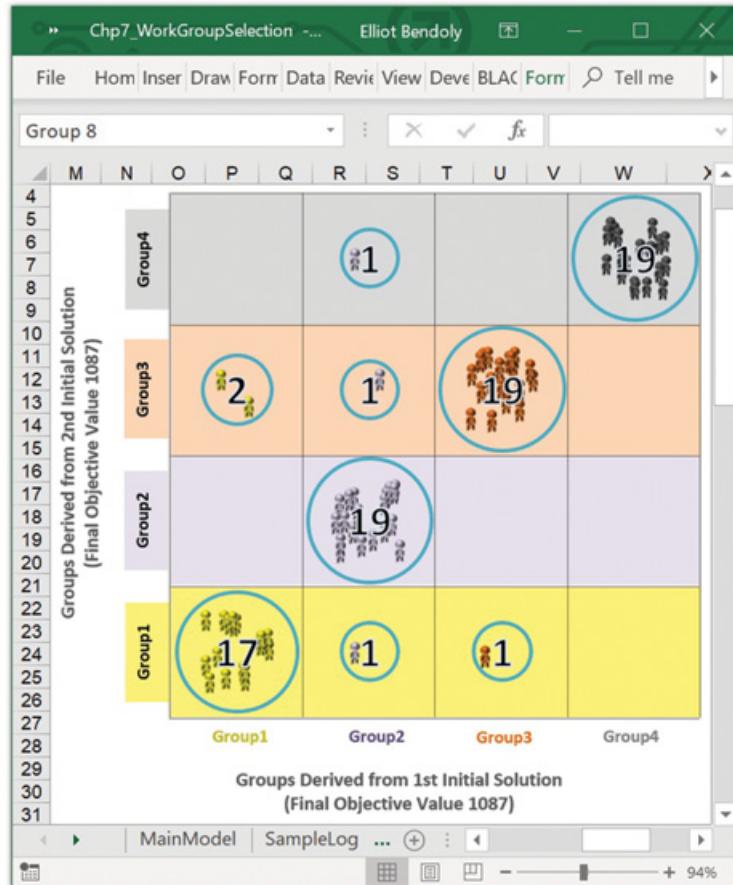


Figure 7.39 Example of a jitter-enhanced comparison of worker assignment solutions

problem, but additionally specify that 10 iterations are considered for every solution considered. Make sure to convert “Value” specifications to “Mean,” for example, where appropriate in RISK Optimizer. After five minutes of run time use the RISK Optimizer log to comment on the nature of the performance variability of the final solution suggested relative to other best solutions encountered along the way.

## **Section 4**

# Advanced Automation and Interfacing



# 8

## VBA Editing and Code Development

**Objective:** *Delve into programming logic and flow, VBA syntax, and debugging tools. Become familiar with code structure, communication with spreadsheets, dynamic data storage, conditional statements and loops, calling worksheet functions, and creating user-defined ones.*

Effective decision support systems rely not only on the ability to present information, analysis, and meaningful dynamics (e.g., through visualization) but also on enabling users to easily customize functionality to their own interest without the developer holding their hand in that process. This is often going to mean providing sufficient documentation that might go beyond cell labeling and embedded comments. It may mean coming up with structured guidance for users, checks on effective usage, help resources, or “wizards” as part of the DSS.

This in turn is often going to mean a level of automation that stretches the limits of the kind of work that can happen at the spreadsheet interface alone. In fact, it may be untenable using only the top layer of an Excel workbook. We’ll need to dig below this layer to supplement our spreadsheet front end at times, sometimes replacing what we had originally designed to function on the front end by much more efficient and effective approaches on the back end. Let’s see how macros and the Visual Basic for Applications (VBA) Editor might provide us with some new options in this regard.

### 8.1 The Visual Basic for Applications Editor

We’ll begin by taking a deeper look into one of the first macros we’ve recorded in the course of this text, originally created in Chapter 4 and then leveraged in analysis in Chapter 7. Opening either the Chp4\_LobosInventory\_BasicModel or the Chp7\_LobosInvBasicFullSimMacro\_Opt1 workbook provides us with an opportunity. To see the code associated with this macro, select the Developer tab on the main menu bar and then select Visual Basic (which

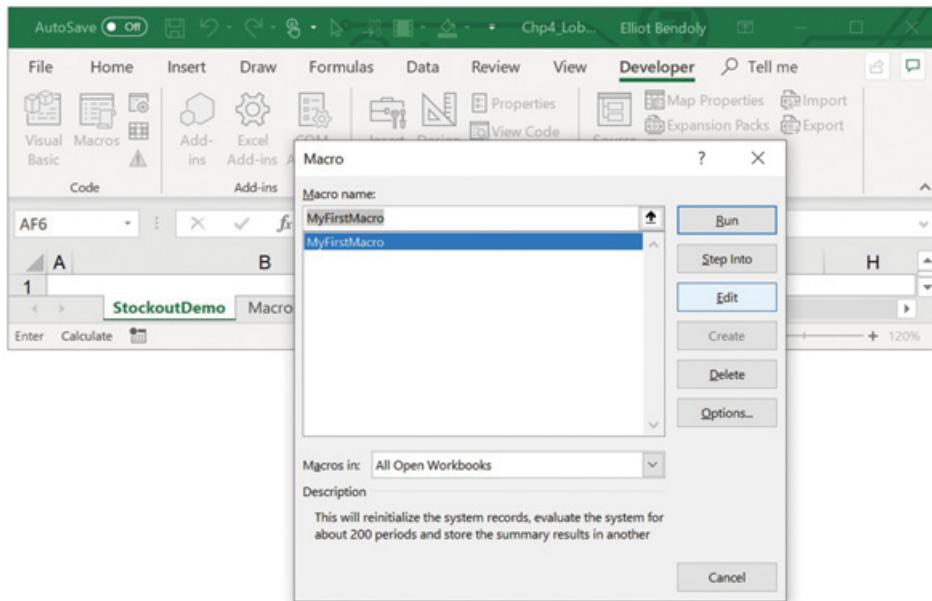


Figure 8.1 The Developer tab and associated elements

will open the general VBA Editor screen); or click Macros (see Figure 8.1) and from the associated dialog box select the specific name of the program code you are interested in viewing (in this case, generically called MyFirstMacro) and then Edit.

The VBA Editor in Excel has its own distinct structure, markedly distinguished from that of the front-end Excel spreadsheet environment. Because any given macro may be specific to a single workbook, worksheet, or even refer to an included add-in (such as Solver or the Blackbelt Ribbon), the VBA Editor provides a mechanism to categorize the macros that are associated with any workbook currently open in Excel. That mechanism is the Project window that appears to the left of the VBA Editor interface (shown in Figure 8.2). It's a bit like a file directory, with components such as Sheets, Modules, and Forms, each capable of storing code.

If what you see looks like it's missing something that Figure 8.2 shows, that's OK. I have the Blackbelt Ribbon active; you might have other add-ins, or none, so your Project window will reflect that. If you don't see any Project window whatsoever, go to View in the VBA environment's ribbon and select Project Explorer (or Ctrl-R in this environment). You can similarly open the Properties window that I'm showing (which we will use more in Chapter 9). If you don't see the Immediate and Watches windows at the bottom, the same deal applies.

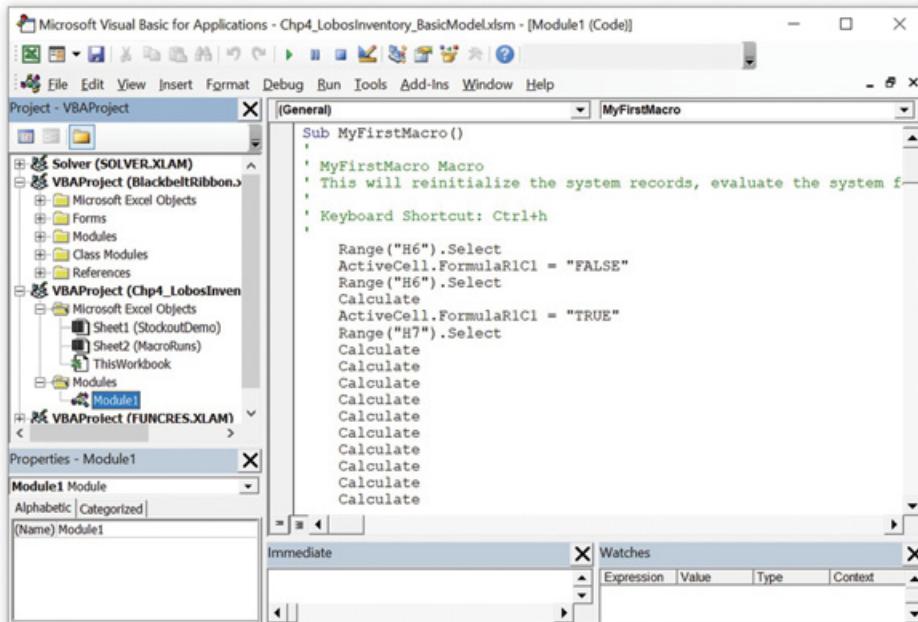


Figure 8.2 Basic elements of the VBA Editor environment

As for the code, which you should be seeing if you have elected to “Edit” the macro in question, the language we are seeing is that of VBA (which all macros in Excel are recorded in). We’re actually viewing a Module window here, in the bulk of what we see presented in Figure 8.2. All code recorded as macros gets automatically stored in these project components called modules, within the general Modules folder of the workbook file or add-in – that is, unless you copy/paste that code into other components for editing and use; or unless you accidentally save a workbook with a macro as a non-macro-enabled workbook, in which case no code gets retained anywhere in your book. If you accidentally close the current module window, just find that module (Excel defaulted in naming this “Module1”) in the Project window, and double-click to reopen it. In this chapter, we’ll begin with considering these module components for housing our code, with a deeper consideration of another code-housing component in Chapter 9.

### **8.1.1 Confronting Code**

The critical feature of the VBA Editor is access to viewing, editing, testing, and applying VBA code and eventually creating it from scratch as needed. In fact the code we are currently considering (as in Figure 8.2) wasn't typed or copied from some other source and pasted in, but rather was created in our

application (Excel), based on actions we took in the spreadsheet environment. It was the application's best attempt at capturing verbatim the user actions taken from the point at which macro recording began to the point at which it was terminated (hitting "Stop"). Often more is recorded than is necessary for your needs, but that's far better than getting less than what you want.

Modules are also where much of the ground-up code development is done by users (in lieu of macro recording). This is a critical point, in part because not all useful code can simply be generated through macro recording alone. Admittedly, coding is probably one of the most intimidating areas for new developers, particularly those without computer programming backgrounds. Fortunately, it's much easier to learn to code in VBA than you might think. And it's worth it. Those who know how to do even a limited amount of coding in this environment not only can make their own work easier, since these skills apply across MS Office products, they are also highly valued in the assistance they can give to their organization. Experience coding in VBA also translates into coding acumen in other languages (e.g., Swift, R, Python). People who can help the entire organization (and ostensibly other organizations) with work tend to reap dividends.

As an experiment in the accessibility of VBA coding, let's try to interpret the language Excel and the VBA Editor use to keep track of some of the actions we may have recorded. Table 8.1 shows an abridged version of the code written and used in our example.

To help us get through a first pass at translation, I've fully annotated the code (the text in Table 8.1 is not actually present in the code itself); but even if you didn't have this annotation, do you think you could have guessed what some of these lines did? Some are fairly obvious, or at least suggestive. For example, consider `Range("H6").Select`, `Selection.Copy`, or `Selection.EntireRow.Insert`. What do you think these lines of code are doing? VBA capitalizes on simple elements of the English language, making it easy for even laypersons to navigate. Some code is more cryptic, but in general we could get a sense of what's happening here by attempting to read through the code as if it were steps in a set of cooking instructions.

As with any good cookbook, we also have tactics in VBA that will help maximize the potential for readability. Here we see, for example, the use of "\_" to denote a line continuation, sort of like a hyphen breaking a word up. In the next subsection we'll see some more sophisticated tools for better understanding code. The bottom line is this: Although it requires a little getting used to, the value of getting to know code logic and syntax is immense. And specific to VBA, at least, you'll find that recording and editing macros is one of the best ways for those who aren't career programmers to learn some amazing things that lock in that value.

Table 8.1 Annotation and code for Chapter 8 example

Annotation	Actual Code
<i>Name/Start of macro/subroutine</i>	Sub MyFirstMacro() ‘
<i>Notes/Comments</i>	‘ MyFirstMacro Macro ‘ This will reinitialize the system records, .... ‘ ‘ Keyboard Shortcut: Ctrl+h ‘
<i>Selection of active cell</i>	Range("H6").Select
<i>Specifying “reset/reinitialize”</i>	ActiveCell.FormulaR1C1 = “FALSE”
<i>Reselection of active cell</i>	Range("H6").Select
<i>Call to Recalc (e.g., F9) to reset</i>	Calculate
<i>Specifying “start simulation”</i>	ActiveCell.FormulaR1C1 = “TRUE”
<i>Selection of alternative active cell</i>	Range("H7").Select
<i>Call to Recalc to iterate</i>	Calculate
<i>And again Recalc</i>	Calculate
<i>~ 200 of the same lines</i>	Calculate
<i>(for 200 periods iterated)</i>	:
<i>Select key cells</i>	Calculate Calculate Range("C31:C34").Select
<i>Copy them as a set</i>	Selection.Copy
<i>Select the “record” sheet</i>	Sheets("MacroRuns").Select
<i>Select a starting cell there</i>	Range("A1").Select
<i>Paste the copied cells there (note that the “_” is used to denote next-line code statement continuation)</i>	Selection.PasteSpecial Paste: =xlPasteValues, Operation:=xlNone, _ SkipBlanks:=False, Transpose:=True
<i>Closes out the copy/paste action</i>	Application.CutCopyMode = False
<i>Inserts a row ahead of more pasting</i>	Selection.EntireRow.Insert
<i>Reselects the original model sheet</i>	Sheets("StockoutDemo").Select
<i>End of macro/subroutine</i>	End Sub

### 8.1.2 Walking through Code and Checking for Bugs

Just to set expectations appropriately: Things will go wrong with code. You'll make occasional mistakes and omissions, and some of your designs will not always work as fast or as robustly as you would like. Let me emphasize that for every minute that beginner (and even advanced) developers spend on coding, they may find themselves spending that same amount of time (and often much more) figuring out where flaws exist and how to resolve them. Fortunately, there are resources available to help pinpoint where things go wrong, shedding light on the causes and context, and facilitating correction and improvement.

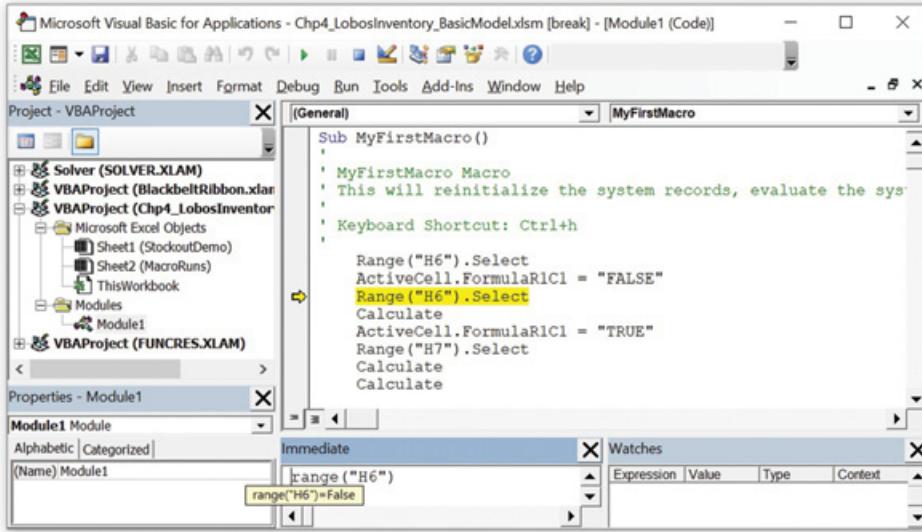


Figure 8.3 Step-through code execution

The first feature is the ability to “step through” code line by line as opposed to just letting code run until it ends or breaks down. In VBA Editor, after a specific set of code is selected and the blinking line cursor appears within the code (as for editing), pressing the F8 key will step the developer through the code line by line. Any line about to be run will be highlighted in yellow, signifying that the particular line of code is on deck. A yellow arrow also appears at the side bar of the code window to emphasize the current line (see Figure 8.3).

Some code that might contain text, numerical or Boolean (T/F) content (such as a referenced spreadsheet cell), command parameters, or variables can also be examined at this interim point. Hovering your cursor over such elements (such as `ActiveCell.FormulaR1C1` in our example) will provide insight into the most recent values they contain. If that doesn’t work, perhaps because the line of code isn’t related to content (e.g., `Range("H6").Select`), then simply including it in the Immediate window (typing `Range("H6")` in that window) creates that same opportunity.

As an alternative to using F8 for close examination of everything, often it’s preferable to allow larger bodies of code to run and halt for inspection only when specific points of interest are reached. Those specific points are often chosen by developers because there is some evidence (or intuition) that something around that point is likely to cause some problem – something that has either just recently gone wrong or has the strong possibility of

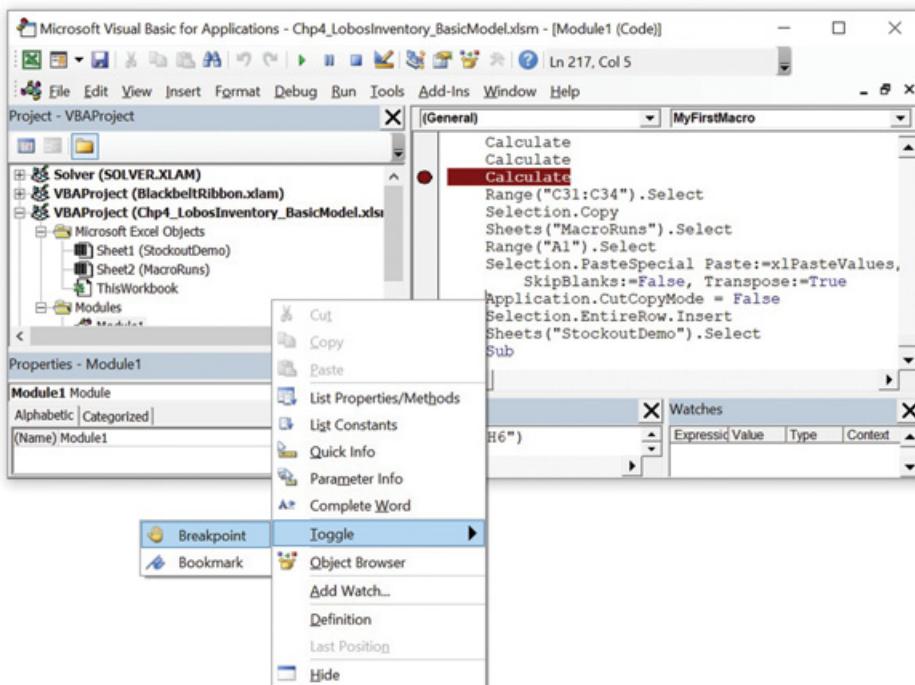


Figure 8.4 Toggling breakpoints in code

going wrong under certain future conditions. To insert these breakpoints, the developer can select the line of code of interest, right-click, and select Toggle>Breakpoint (see Figure 8.4). The line then gets shaded maroon and a maroon circle appears to the left of the code. A much faster way is to select a line of code and press F9 in the VBA Editor or, more simply, to toggle breakpoints on and off by clicking on the bar area immediately to the left of the code (which presents the above-mentioned maroon circle indicator).

There can be as many breakpoints as there are functioning lines to a body of code (although there would be little point in toggling so many). When these are added, you can start the execution of code within the VBA Editor (using the Play button). The code will run up until each breakpoint and then stop until the next breakpoint is reached, the code ends, or the code breaks down because of a processing error. At each break, you have the opportunity to check what the spreadsheet looks like, or again check on the status of some data maintained or referenced within the code (cursor hovering or Immediate or Watches windows). Methodical examinations can provide a wealth of insights and seriously reduce frustration that can accompany debugging.

## 8.2 Previewing Unique VBA Capabilities

Before delving further into the syntax and logic of VBA code structures, let's preview a few more of the activities that VBA makes possible in coordination with Excel. Understanding this landscape will help us appreciate possible approaches to leveraging VBA automation. At this point it should be clear that there are many other objects you might encounter, construct, or manipulate aside from cells in a workbook. Objects such as graphs, buttons, and interactive computational elements of various types can reside within a workbook and may ultimately serve as the primary vehicles for users to interface with a DSS. Yet, as with cells, there's often much more to these objects than you might first expect.

Similar to cells, all the things you encounter in the Excel spreadsheet environment have a certain set of properties that can be manipulated. It's not unusual to hear developers refer to the full set of properties of any object as constituting a record, using some terminology we will see later, although the structure and content of those records may be very different depending on the objects. For example, objects such as ActiveX controls are specifically designed to possess properties that a simple circle drawing would not. Check boxes contain a property that describes the cell to which they are linked, and whether they are checked or not checked, for instance.

At the same time, simple circle drawings possess other attributes like "Line.DashStyle" that check boxes don't have (at least not in an identical form). Regardless of these nuances, there are some properties that all these spreadsheet objects share in one form or another:

**Name:** How to (hopefully unambiguously) refer to the object

**Location:** Where the object is relative to some top/left reference point

**Basic Appearance:** Size and some color attributions visible to users

As an experiment, try recording the process of adding a drawn circle shape (under Insert>Illustrations>Shapes) somewhere in an open spreadsheet, and move, resize, and modify the color of that object. Then make sure to hit "Stop Recording" to end the macro-recording session. Look at the code recorded (the same way we did in our 8.1 example). That's how Excel has recorded your actions. It includes how Excel refers to the object you've created, how it refers to the attributes that you've modified, and how it refers to those modifications. If you only recorded the creation and modification of a drawn shape object and never selected something else in the process, you should be able to replay this without error.

However, the code recorded can also be fairly specific or entirely specific. It will name newly created objects according to defaults and refer to those same objects elsewhere in code. If those named objects don't exist the next

time around (e.g., have been deleted), or if you expect these same actions to take place on other newly created objects, your code might not work the way you want (or at all). It's yet another reason to get more acquainted with writing code rather than just recording. Macro recording can get you to logic and syntax you weren't already familiar with, but knowing how to code allows you to fix things to be robust to your needs, rectifying the assumptions that Excel knows no better but to make.

Certainly some objects are going to be far more complex than others and therefore the associated code may be harder to interpret or even to get at (not everything you do while recording a macro will actually get recorded in such cases). Still, to encourage our discussion, before getting into the bread and butter of VBA syntax, it's worth going over a few examples to give a sense of the range of VBA applications, similarities, and distinctions in code.

### ***8.2.1 Manipulating Embedded Visuals***

To begin, let's see what we can expect if we decide to import an image of some kind into our workbook. In this case we'll try out one of the newer features: 3D Models. Excel provides an online library of these, but you can import others. Start by going to Insert>Illustrations>3D Models and then browsing for a model to import. For my own reference I'm going to rename this object SampleImage (using the same naming field I would use to name cells and cell ranges). In doing so I'm essentially updating the name attribute of that model. Access to 3D model importing and the model actually imported is shown in Figure 8.5 (Chp8\_ObjectTrials).

Note that when the object is selected in Excel, a new Format ribbon displays in the ribbon structure. Right-clicking on the object and selecting Format provides another view of modifications that might be made. All of these items on the Format ribbon, and accessible via right-clicking, represent attributes of that object's record: location, size, rotation, pan/zoom (in this unique case), background color, and others. You might try recording a macro in which you first select the imported object and then manipulate certain attributes. Notice how syntax relating to these changes often involves a “dot” (“.”) notation, drilling down into aspects of the object in question.

Once again, the code will give you insight into how those attributes are referred to in VBA syntax, but replaying the code might not deliver what you expect. Shifting the location incrementally will result in additional incremental shifts, but recording a rotation may only end in the same rotated position. Some actions recorded by macros take the form of verbs (e.g., move a little to the left), while others are more like adjectives (e.g., set the Y rotation to 45 degrees). Something similar applies to structures you

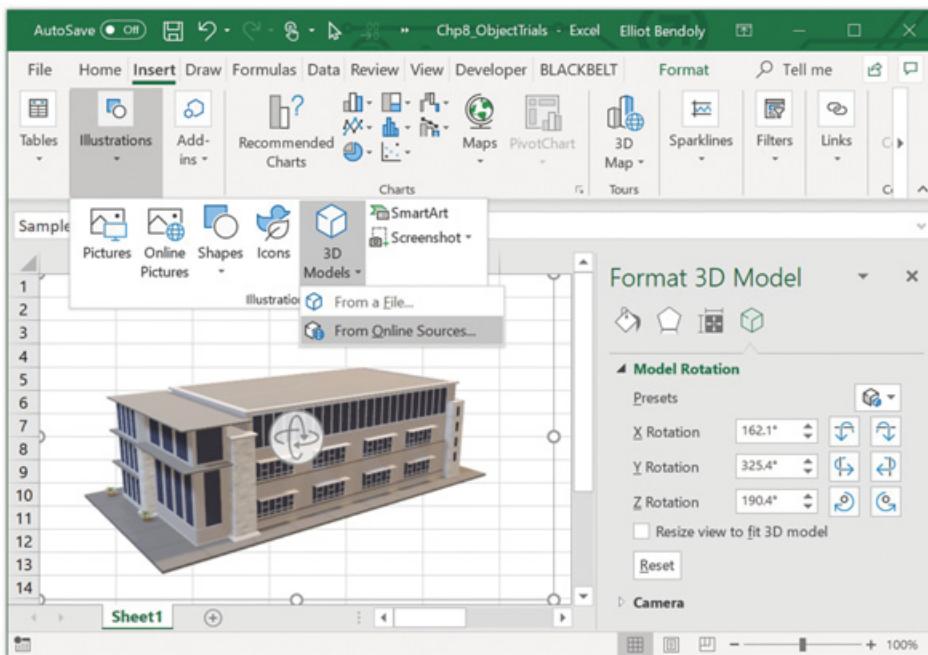


Figure 8.5 Example of picture import and available editing tools

can create in languages such as Swift (e.g., “methods” vs. “properties”), with dot notation again applied in syntax to drill into these aspects of structure. Again, if you haven’t worked with such code in the past, absent other information, you might only discover how things are recorded in VBA by trying to record them. Fortunately, for every inconvenient surprise we might encounter as we learn about new syntax there is always a workaround.

### 8.2.2 *Modifications to Data Tools*

Now, let’s consider another example we’ve looked at early on. In Chapter 3 we introduced Pivot Tables (to those not already familiar with them). The Chp3\_QuarterlyData workbook houses a couple examples of Pivot Tables in use, with the second including Slicers to facilitate user interfacing. Let’s say that we expect our users (or ourselves) to do at least two things with this interface on a regular basis: (a) examine Midwest state numbers for 2003 and 2004, and (b) reset the slicers so that filters are cleared. We could record two macros involving these actions, assigning short-cut keys to each. Chp8\_QuarterlyData\_Macros includes two such recorded macros. One, (b), is considerably simpler than the other, (a). The “reset” code in this case essentially includes only the following two lines:

```
ActiveWorkbook.SlicerCaches("Slicer_State").ClearManualFilter
ActiveWorkbook.SlicerCaches("NativeTimeline_Date") .
ClearDateFilter
```

Note again the use of dot notation to drill into aspects (commands/methods in this case) specific to the Slicer objects housed in the sheet.

The code for the first selection procedure, (a), however, is much more lengthy and takes a much longer period to run. In fact, if you run this one, you might get a little concerned that Excel is hanging up on you. But don't worry; it will eventually finish up. It just has a lot to do, not because we want it to do a lot but because Excel records both the initial and the final states of the Slicers, making them actions that must be performed. That effectively doubles the amount of run time. It doesn't take a computer scientist to select and delete that extra code and cut the run time in half (as I've done in the "alternate" code in that workbook, shown in Figure 8.6). Getting the code to appear a bit more streamlined will take some more work, but at least we can see, fairly clearly, what we're working with.

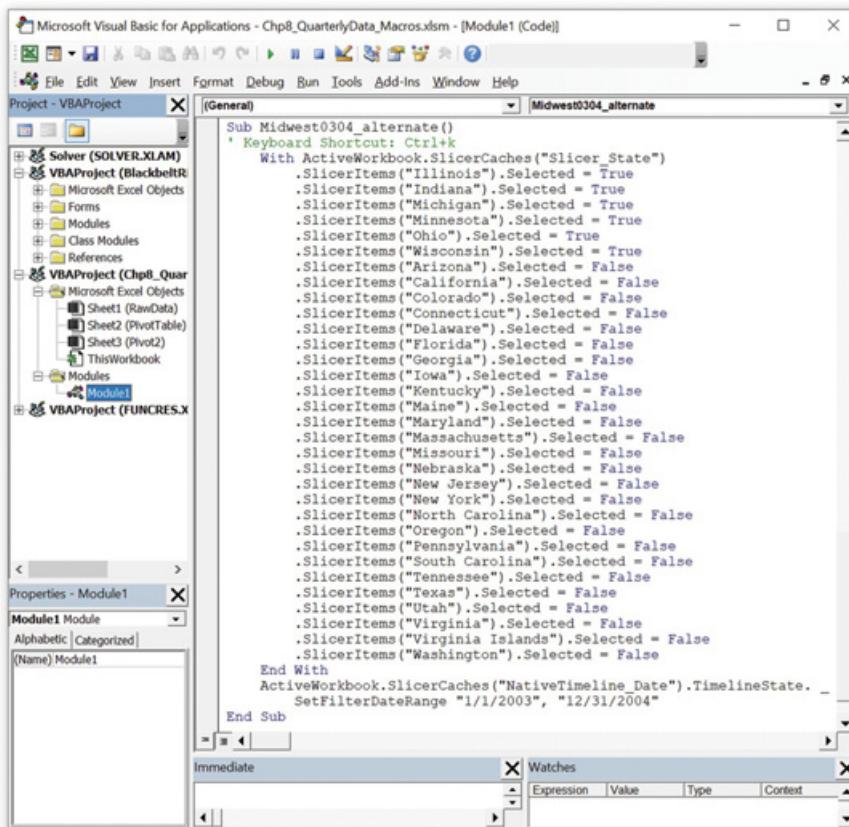


Figure 8.6 Macro-recorded changes to Slicer selections

### 8.2.3 Chart Properties: Regression Revisited

In Chapter 4 we introduced two mechanisms for developing best-fit model estimates in data (regression). Formulaically, we have the LINEST function available in Excel. We also have the Data Analysis tool that can present a fixed report of the same type. In Chapter 7 we noted that these same results could be derived through Optimization and Solver's GRG Nonlinear engine. And earlier, in Chapter 5, we were introduced to yet another mechanism for developing (univariate) best fits within scatter plots, with visibility into the best-fit equation and  $R^2$ . However, ultimately, the numerical results of these best fits were presented only in the graphs themselves and appeared inaccessible to the spreadsheet (where they could be useful in additional analysis). Fortunately, these estimated equations in the scatter plots are simply another attribute of those plots, and, as such, they can be accessed through the same dot notation relevant to the prior object code examples.

Let's look at the Chp7\_FitbyOptimization file that contains all of these methods, including a Scatter with the univariate best-fit information. This is a VBA-enabled workbook, and if we look closely we'll see a comment in cell L4. That comment outlines three shortcut-activated VBA subroutines: one (Ctrl-n) that changes the fit to linear, one (Ctrl-g) that changes it to logarithmic, and a third (Ctrl-l) that extracts the best-fit line information and stores it in a cell named "TrendlineData." Peeking behind the scenes at the VBA code reveals the three subroutines. Let's focus first on the one that extracts the best-fit information (the R-square and estimated fit equation) and store it in a cell labeled "TrendlineData."

```
Sub PullTrendLineData()
    Dim ChartText, LocatR, OldCarriage As String
    ChartText = ActiveSheet.ChartObjects("Chart 1").Chart. _
        SeriesCollection(1).Trendlines(1).DataLabel.Text
    LocatR = Application.WorksheetFunction.Find("R", ChartText)
    OldCarriage = Mid(ChartText, LocatR - 1, 1)
    Range("TrendlineData").Value = Application.WorksheetFunction. _
        Substitute(ChartText, OldCarriage, Chr(10))
End Sub
```

We have some lengthy lines of code here and some new things we haven't seen yet. First, as mentioned earlier, “\_” denotes a line continuation, to help in our presentation of lengthier lines. Second, we are temporarily creating some storage space for information, denoted by Dim and the text that follows it (we'll get to this in Section 8.3). The line that begins "ChartText" is the one that pulls in the best-fit line text from the chart, and the following lines work to ensure that there is true a carriage return recognizable by Excel prior to the  $R^2$  in the text (the chart and the spreadsheet retain carriage

returns a bit differently). In all of this we see some functions that look familiar (FIND) and some that look uncannily similar to others we may have seen in the spreadsheet (Chr vs. CHAR). We'll talk more about these nuances further on.

There are also some assumptions in the code. A huge one is that the scatter plot is actually called "Chart 1." Another is that we are going to call this code while we have the sheet containing that chart open (i.e., it's the ActiveSheet). There is also an assumption that a trend line (and the data series it is based on) already exists. Ultimately the code is functional at this point, albeit not very robust.

The other bodies of code are simpler. If you want to change from a linear fit to a logarithmic one, no problem. The code behind Ctrl-g is as follows:

```
Sub ChngtoLogarithmic()
    ActiveSheet.ChartObjects("Chart1").Chart.-
        SeriesCollection(1).Trendlines(1).Type = xlLogarithmic
End Sub
```

Changing back to a linear fit (code behind Ctrl-n):

```
Sub ChngtoLinear()
    ActiveSheet.ChartObjects("Chart1").Chart.-
        SeriesCollection(1).Trendlines(1).Type = xlLinear
End Sub
```

Additional code capable of deleting the trend line altogether and adding it anew, or additional lines, is presented in Figure 8.7. These are not currently assigned control key shortcuts, since they have the potential to raise errors at this point (if you delete a best-fit line, the code that attempts to convert that missing fit to a logarithmic one won't make any sense). A little safeguarding and/or error proofing, which we will consider again soon, would go a long way to making this full set of subroutines robust to these issues and useable as a collection.

### 8.2.4 Controlling ActiveX Media Objects

As a final example before moving on, let's consider something a lot more exotic. When we considered the introduction of ActiveX objects in Chapter 4 and its Supplement, we really only scratched the surface regarding the kinds of things that Microsoft developers have considered over the years as possible elements worth integrating into the spreadsheet environment. These were standard controls. However, many others exist under "More Controls" (wrench and screwdriver icon likely on the lower right of the ActiveX menu). Not all are still supported. For example, the embedded

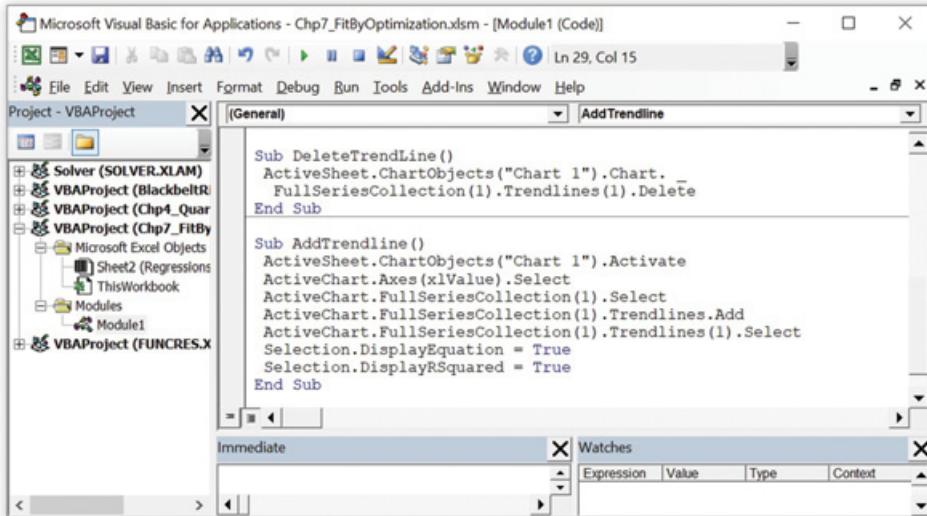


Figure 8.7 VBA code accessing and modifying a trend line

browser control, an incredibly handy element to have available that took up minimal space in workbooks, no longer functions in newer versions of Excel. Why it remains on this list of objects is beyond me, but it's a reminder that not all great things last. Still, others do.

One that is still functional is a Windows Media Player object. Clicking on “More Controls” and browsing for this object gives you the same cross-hairs cursor from earlier, allowing placement and sizing of this object. Creating it also activates the Design Mode and allows us to access the object’s Properties. One of these is Custom, which opens up another interface for modifying things such as volume and frame rate. But the key property (also present in Custom) is “URL.”

This object doesn’t need to work with online media. Your URL could just be the file path of a .wmv file on your computer (and yes the player is particular about the kind of media it is asked to play). But it can access online media as well, which adds to the mobility of any workbook using it. In the Chp8\_ObjectTrials workbook I’ve placed one of these Windows Media Player objects on the same sheet as the 3D model from before. I’m using a smaller version of the Power Map video from Chapter 5 (Chp5\_ThreePointSmall.wmv), which I’m hosting online, as the source (the URL) for this object (see Figure 8.8).

This object, which I’ve named Media1, should be ready to play the designated media file, if you manually click on the play button. But what about automating this and perhaps the selection of the file in the first place? Well, it’s all possible,

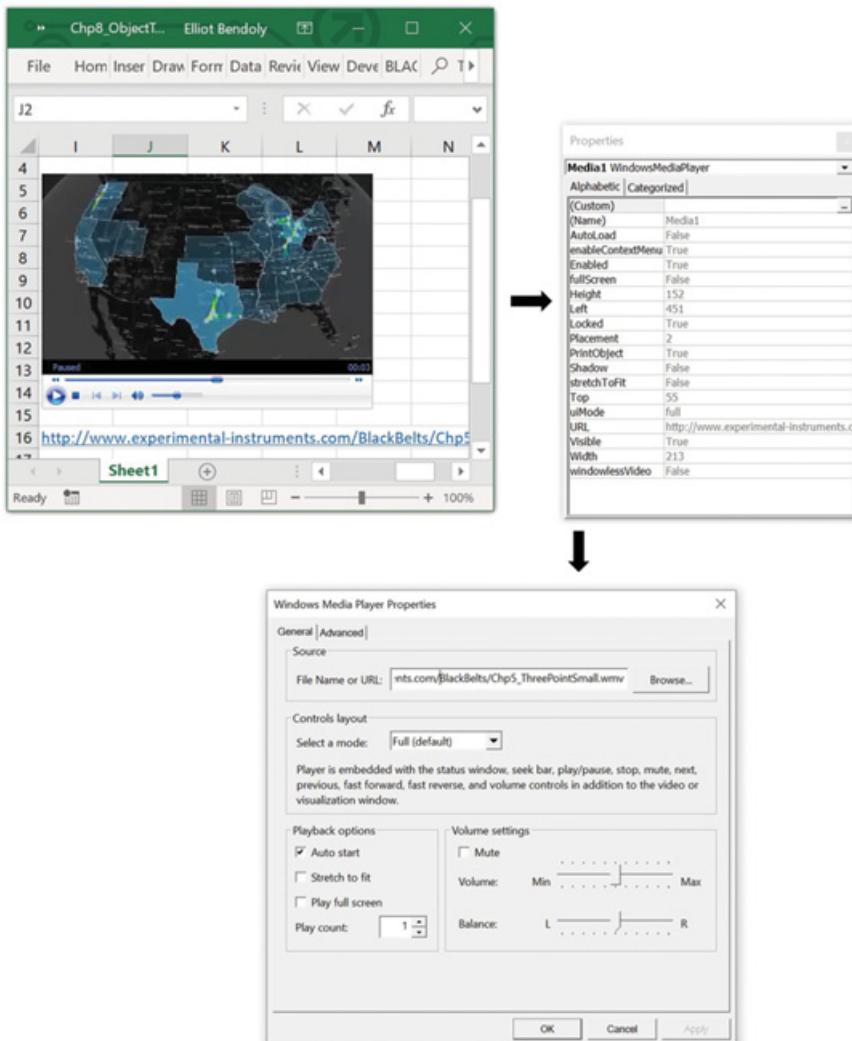


Figure 8.8 Properties of an embedded media object

but you won't have much luck recording a macro to get to that code. If you try to do that, at least at this point, very little will get recorded, apart from maybe the selection of the worksheet object. This is a great example of the limitations of macro recording, and the usefulness of being prepared to get your hands dirty with direct coding. I had to do a little searching, but the code (`RunExistingPlayer`) to simply play the existing media object (called `Media1`) on the `Activeworksheet` is just one line: `ActiveSheet.Media1.Controls.Play`. In order for VBA to understand this line, you might have to give it a kind of dictionary, called a Reference (under Tools), in this case including the Windows Media Player "Reference" (Figure 8.9).

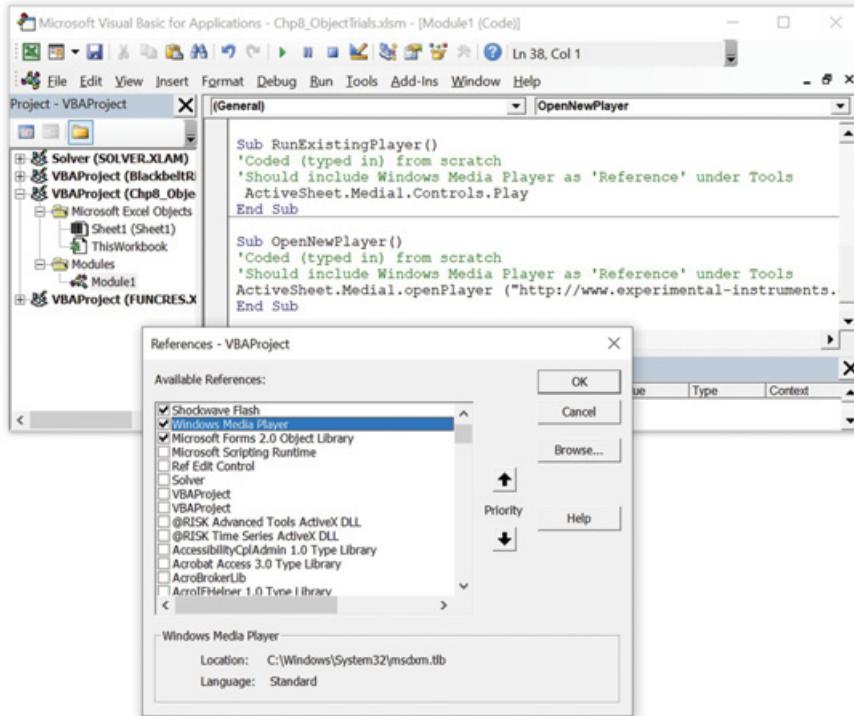


Figure 8.9 VBA code managing an embedded media object

Since you might prefer to call into being a media player that doesn't already exist as an object in your workbook, I've also included code (the OpenNewPlayer code in Figure 8.9 and the Chp8\_ObjectTrials workbook) that opens a new player separate from the spreadsheet environment. Although spreadsheet-embedded browsers are not currently supported, a similar option of calling browsers into being separate from the workbook environment is also still possible. The code may be a little obscure and take some searching for, since it's not macro-recordable; but once you have found it, you won't need much code to get the job done.

### 8.3 Syntax and Storage

Now that we've gotten a taste of the range of what we can accomplish using VBA code, we're ready to take a step back and look at the fundamentals of VBA syntax more closely. We keep seeing the term "Sub," and I continue to refer to the term subroutine. It's no coincidence. *Sub* is short for subroutine, which is just a set of instructions for VBA to follow. Macro recording involves the construction of subroutines. So, when we talk about macros,

we're really talking about subroutines. And subroutines can be built in many different ways without the use of macro recording.

All of the macro-constructed and directly coded subroutines in our four examples start with the following general statement in the VBA Editor, with "SubName" here serving as a placeholder for the actual macro or subroutine name:

```
Sub SubName () 'Where SubName is whatever name you choose
```

The content following the apostrophe is clearly just annotation (nonfunctional). Similarly, each of our examples ends with the following:

```
End Sub 'always the same regardless of the subroutine
```

Everything between those two lines represents what will take place (or "could" take place, depending on certain conditions, as we'll see in a moment) when the macro/subroutine is run. Remember that we can start the run of any public subroutine directly from the Excel workbook interface by going to Developer>Macros>Run or by hitting a control key shortcut assigned to it, among other approaches as we will also see.

### 8.3.1 Introduction to VBA Data Storage: Variables and Types

Since the development of VBA automation in our decision support efforts will often require us to consider specific details of the analysis context, one of the first things we should nail down is how we are managing data in this environment. We will need to be comfortable with the syntax of communicating with the spreadsheet where data describing the context may exist (raw data, model fit parameters, simulation rules, selections made by front end users, etc.), but we will also want to be familiar with the syntax of storing and manipulating data behind the scenes in the VBA environment. In other words, we'll want to know how to work with data separate from the cells of the spreadsheet environment (which will provide greater flexibility and speed in data processing).

As introduced in Section 8.1, the specific syntax for referring to cells and their content in VBA is to use the term "Range" followed by parentheses. Within those parentheses we can either have the column-row designating address of the cell, Range("A1") or Range("Sheet1!G2:H9"), for example, or what we've named that range in the spreadsheet, Range("Rates"), for example. This applies regardless of whether we are acting on that range (e.g., .Copy), or reading information from it, or writing information to it. Alternately we can have some other term (a *variable*) that we've defined in VBA to store the text range of cells we are interested in, for example, Range(Nextcell), where Nextcell contains something like "B17." Note that

the contents within the parentheses will always be text, denoted by quotation marks, regardless of whether that text is being stored within a variable like Nextcell.

Now, what is a *variable* in VBA? Simply stated, it's a part of the computer memory set aside to temporarily store information while the subroutine is running, and which should subsequently free up that memory afterwards. This is unlike the cells of spreadsheets, which, if they have content, will continue to take up memory until the file in which they reside is removed from a computer. Although each content-retaining cell by itself doesn't take up much space, as a collective they certainly can have an impact. Further, the act of referencing and modifying data that's present in the cells of spreadsheets actually takes a little longer than doing the same with data stored only by simpler structures in the VBA Editor during run-time.

Think about it this way. The computer has a large block of memory it makes available for storing content. When Excel is running, part of the computer's resources performs regular checks for changes in open workbook spreadsheets. Recall from Chapter 2 the wealth of attributes associated with each cell in a spreadsheet (static or formulaic content, visual size, fixed or conditional formatting, comments, links to externals, and so on). Whether they are used or not, these attributes come default with each cell. So, cells are already fairly complex things. Data stored behind the scenes in VBA, on the other hand, can be stored in much simpler structures, without a need to define all the bells and whistles that come with cells. Accessing this doesn't need to involve the additional step of asking the Excel environment to access the data-content portion of something more complex. Changes to data stored behind the scenes in VBA also doesn't risk a full-blown recalculation of the formulae contained in other cells in a spreadsheet, which happens if cell content is changed (unless certain precautions are taken such as "manual updating" in Excel options).

In short, if you have a lot of data that you plan to access and base multiple, repeated calculations on, this data simplicity can really make a difference in the amount of time it takes for the program to run. You may not want to do this for all the data you use in the workbook; it makes sense to have some of it stored in spreadsheet form if, for example, you want to make sure it's retained when you save the workbook. But during long and exhaustive data cleaning, manipulation, and calculation exercises, this kind of virtual storage comes in handy. I don't expect many to be able to make this "efficiency" distinction in the smaller tools developed up to this point. However, it is a large enough distinction to spur discussion by analysts on the topic of how to streamline Excel–VBA integration in large Excel-based applications.

And there's another reason why considering back-end data management in VBA may prove useful. Some of the data that we work with takes on

structures that may be difficult, or at least somewhat cumbersome, to meaningfully store or readily access in spreadsheet format. For example, let's say we want to run a simulation that takes into account all the restaurant franchises in a niche market that exist in each of the 48 contiguous states. Based on historical data, some franchises open whereas others close over time. Imagine that for the ease of certain calculations we want to make sure our data remain sorted by state and restaurant chain. Storing and changing this data in a spreadsheet over the time window of the simulation would mean either inserting and deleting rows of data (and knowing where to do such deletions and insertions based on state and chain) or adding data at the end of a list and then resorting that list every time new data is added.

This can be a lot of work, even if fully automated. Short of getting involved with a formal database application, a better way to get work done along these lines might be to temporarily store that data in a flexible VBA-based structure, that is, something that automatically indexes data the way you want (e.g., by state, chain, and franchise number) while keeping track of which franchises are open and which are closed. In particular, certain forms of these variables (often referred to as dynamically linked lists) are used extensively by DSS developers. We'll focus on the simpler forms of arrays in this chapter. But even before we get there let's just outline some of the most basic kinds of data structures in VBA: single-value variables and their data types. Common types include the following:

**Integer:** For whole numbers ranging from -32,768 to +32,767

**Long:** For whole numbers ranging from -2,147,483,648 to 2,147,483,647

**Single:** For all numbers ranging from as small as  $\pm 1.4 \times 10^{-45}$  to as large as  $\pm 3.4 \times 10^{38}$

**Boolean:** For variables that take on the value of either True or False

**String:** For variables that take on text values (such as someone's name)

**Range:** For storing not the text name/address but an actual cell Range (i.e., not the text value "A1:B3" but a direct reference to that set of cells)

**Variant:** The default type for nonspecified variables (when permitted), which can take on any of the above types once assigned a value (inferred).

Certainly there are other types not listed here (double, date, object, etc.), but for beginners, being familiar with this set is typically sufficient until other specific needs become obvious. It is further worth noting, as we will see shortly, that each of these single-variable types can be used to construct a variety of more complex data types. For example, they could be used to build a "record," a grouping of variables each of a potentially different type: for example, the name and role {2 Strings}, business unit code {Long}, and unit revenue rate {Single} for an individual manager. Alternately, they could form variable "arrays," which are lists of values for multiple instances of the same kind of variable, such as a list consisting solely of individual students'

names {An array of Strings}. In combination these can be used to construct variable *arrays of records* – per our example, lists whose entries *each* consist of a record that includes name, role, unit code, and rate.

To ensure that the VBA Editor is handling data the way you want, your best option, at least starting out, is typically to explicitly define the variables you want it to keep track of right at the beginning of your macro/subroutine. For complex variables, such as arrays of records, this is something of a must. Declarations of variables typically start with the term *Dim* and are followed by the name you want for your VBA variable and a description of the type of variable it is. You can think of the term *Dim* as representing “Declare in Memory,” though originally it was simply a short form of Dimension (particularly relevant to arrays). In more complex cases we may come across, or choose to use, declarations of variables using *Private* (which limits variable access), *Public*, or *Global* (for broader reference and access). For our current discussion, however, we will tend toward the use of *Dim*.

For example, we could create a new subroutine called SubroutineName in which two variables are defined – one called *NewInteger* that is designed to hold integer-type values, and another called *NewName* that is designed to hold text type values fifty characters in length (max). The “\*50” designation specifies this, although it is often not a requirement for String declaration. The following outlines how that code might appear, with a comment included to suggest additional code would probably be included after the second declaration. As we’ve seen in passing already, any comment added to VBA begins with an apostrophe and can exist on separate lines or following code.

```
Sub SubroutineName ()
    Dim NewInteger As Integer
    Dim NewName As String*50
    ' Some other would code comes next
End sub
```

After declaration, you can use these variables within your subroutine without any doubt regarding how the VBA Editor will interpret them. You can now develop additional code (e.g., after the comment above) that assigns and changes values for these variables. Here’s the kind of syntax you might use to assign a value to the *NewName* variable:

```
NewName = "Dorian McAnderstein"
```

The universal rule when using the “=” sign in VBA to make one thing (a variable, spreadsheet cell, etc.) take on the value of another thing is that the item on the left side (in this case, a VBA-declared variable) always takes on the content on the right side of the equal sign. Left is destination, right is

source. You can have equations on the right; you'll never have them on the left. This will always be the case. The opposite is never true. Regardless, early programmers often confuse this sequencing. If things aren't working in your code, step through it using F8 (as discussed) and see if you have the correct things on the left and right of your equal signs. Because of this, you can't place a constant on the left side, only VBA variables or spreadsheet cell locations such as Range("A1"). When "=" is being used as part of a more complex statement (e.g., using "If" in VBA, as we will soon see), a bit more flexibility is afforded regarding what appears on the left side.

It's also valuable to note that the name you give a variable in VBA Editor has absolutely nothing to do with any labels or names you've applied in the spreadsheets of the workbook. VBA works with, but is also largely independent of your spreadsheet environment. The origin of this language is one that didn't involve spreadsheets (hence the need to be explicit and use the term Range to designate when you actually want to communicate with the spreadsheet cells). Because of this, you can use similar names for cells and variables without violation, making your back-end work, front-end work, and communication between the two fairly straightforward. For example:

```
Sub AdjustRate()
    Dim RateofReturn as Single
    RateofReturn = Range ("Sheet1!A2") + 0.01
    Range ("RateofReturn") = RateofReturn
End Sub
```

This pulls a value from cell A2 in your Sheet1, makes a very minor modification prior to storing it in a variable called RateofReturn, and then returns that calculated value to the cell in your spreadsheet named "RateofReturn." Although the current calculation (adding 0.01) is trivial, it could be far more complicated and the same communication between VBA variables and the spreadsheet cells would remain functional. Again, VBA is looking at the variable you've declared differently from the text that is the name of the spreadsheet cell in question, which Range is essentially translating to determine a cell address of interest.

### **8.3.2 Declaring and Using More Complex Variables**

The real power of VBA is often realized at higher levels of data processing, therefore understanding how to temporarily store a large set of variable values is essential. As suggested a moment ago, this capability often comes in the form of arrays. To create a variable array (list) of strings for storing multiple (say, 25 or so) names, we would use a declaration like this:

```
Dim NewNames (25) As String * 50
```

I like to use an analogy here: Think of this as a bookshelf with 25 (or so) shelves. Every shelf is exactly the same size and is able to accommodate the same kind of thing – in this case a string of up to 50 characters in length.

Having said this, the specification of 25 is itself a little misleading here, since it really represents the final index of the array (the final shelf number). Since VBA arrays by default begin with an index 0, just a standard convention, we actually have 26 slots to store information. Still, to minimize confusion, many noncareer developers in VBA prefer to either simply ignore that first entry or explicitly avoid its creation (declaring NewNames(1 to 25) for example). In any event, after such a declaration, to set the name at index 3 of that list (shelf 3 in the bookshelf) to a name in your workbook (such as a name you happen to know is stored in cell B30 of Sheet4), you'd use something like:

```
NewNames (3) = Range ("Sheet4!B30")
```

That's fairly convenient. However, we often have still more complex data structures to work with. Rather than just having a list of names, we might have a list of managers, as suggested earlier – each characterized by a name, a role, employee ID, personal salary, associated unit code, unit revenue rate, and so on. The kind of “bookshelf” we'll need to accommodate this data may still have identical shelves, except that each shelf will have various compartments, each designed to store different content. When you compare this structure to that of the cell structure of a spreadsheet, say that from the Chp2\_IdentitiesList workbook, you'll find that the organization is fairly similar, as shown in Figure 8.10.

Similarity is good. Again, in using VBA behind the scenes of Excel, you usually want to be able to pull in data from the sheets and push it back out. When the data is of the same dimensions both in the sheet and in the VBA code, the task of pulling in and pushing out that data is trivial. But remember, we aren't making new “cells” when we declare things like arrays of records. We're creating something much more efficient. To create a variable record like that of Figure 8.10, accommodating multiple entries of groupings of numerous different variable types, you could use the following syntax:

```
Type InfoRecord
    Name As String * 30
    Role As String * 50
    EmpID As Long
    Salary As Single
End Type
Sub DeclareRecord()
    Dim NewRecords (50) As InfoRecord
End Sub
```

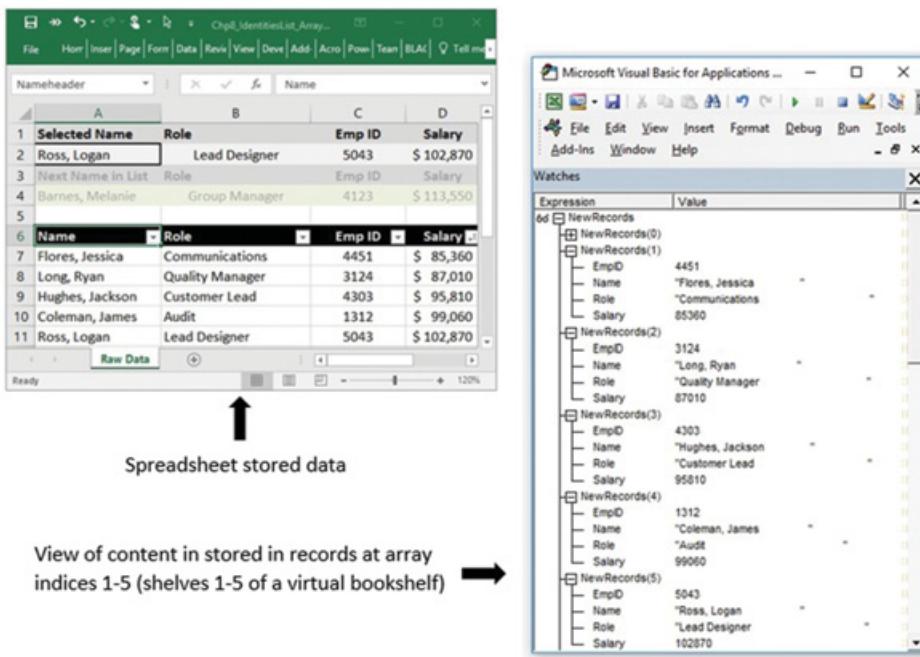


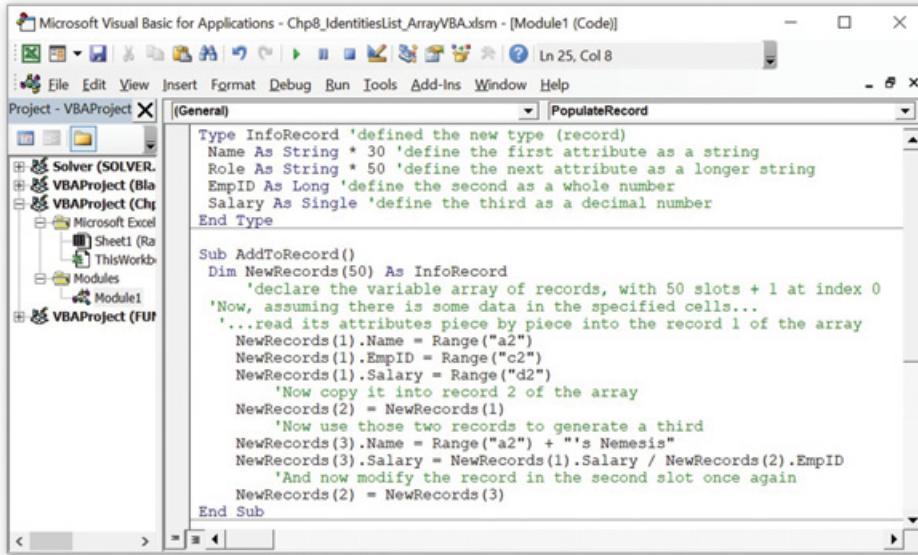
Figure 8.10 Comparison of data stored in a spreadsheet and an array of records

This is a much more complex declaration than the one presented previously, but it's still something we can get our heads around. The first thing that's happening here is the definition of a new type of variable, one that contains four components of Name, Role, EmpID, and Salary. This must be typed at the top of the VBA Editor code before the subroutine starts. The second is a declaration of a variable base on that new type.

Once declared, our new bookshelf is ready to take on some values. Although there's a much better approach to getting data into this array of records, as we shall see soon, at this point an item-by-item assignment can certainly be used. For example, this is how we might assign values to two fields of the entry at index 1 (our shelf 1) of this new array of records variable:

```
NewRecords(1).Name = "Dorian McAnderstein"
NewRecords(1).EmpID = 555442727
```

Note that each field of the record is designated by a dot (.) followed by the name of the field to be used. This dot notation is used to specify an aspect of the larger record element (which compartment of a selected bookshelf) we want to make use of. We've already seen this dot notation in action in our Section 8.2 examples. This same notation will be encountered throughout your experience with VBA to allow you to drill down to more and more specifics of control.



The screenshot shows the Microsoft Visual Basic for Applications (VBA) editor interface. The title bar reads "Microsoft Visual Basic for Applications - Chp8\_IdentitiesList\_ArrayVBA.xlsm - [Module1 (Code)]". The menu bar includes File, Edit, View, Insert, Format, Debug, Run, Tools, Add-Ins, Window, and Help. The toolbar has various icons for file operations. The left pane shows the "Project - VBAPercent Project" with nodes like Solver (SOLVER), VBAProject (Bla), VBAProject (Chp), and VBAProject (FUF). The right pane displays the code for Module1:

```

Type InfoRecord 'defined the new type (record)
    Name As String * 30 'define the first attribute as a string
    Role As String * 50 'define the next attribute as a longer string
    EmpID As Long 'define the second as a whole number
    Salary As Single 'define the third as a decimal number
End Type

Sub AddToRecord()
    Dim NewRecords(50) As InfoRecord
    'declare the variable array of records, with 50 slots + 1 at index 0
    'Now, assuming there is some data in the specified cells...
    '...read its attributes piece by piece into the record 1 of the array
    NewRecords(1).Name = Range("a2")
    NewRecords(1).EmpID = Range("c2")
    NewRecords(1).Salary = Range("d2")
    'Now copy it into record 2 of the array
    NewRecords(2) = NewRecords(1)
    'Now use those two records to generate a third
    NewRecords(3).Name = Range("a2") + "'s Nemesis"
    NewRecords(3).Salary = NewRecords(1).Salary / NewRecords(2).EmpID
    'And now modify the record in the second slot once again
    NewRecords(2) = NewRecords(3)
End Sub

```

Figure 8.11 VBA code reading from spreadsheet cells into an array of records

As a quick example to pull some of these ideas together, consider the following code, presented in Figure 8.11:

Admittedly, the code demonstrated is simply a collection of fairly random data retrieval and storage exercises, but it does provide a little more clarity on the syntax of how these compound variable structures interact with one another. Individual pieces of data, characterizing attributes of larger constructions, can be read and written to, calculated or the basis of other calculations. Entire arrays of records can be copied, as can all substructures in between (e.g., single records within an array). And much more complex structures – the kind of structures certainly not well suited to the dimensions of spreadsheets – work similarly (e.g., arrays of records that possess arrays as record attributes within them).

### 8.3.3 Watching for Changes in Stored Information

All the variables and types just described are helpful because they provide the means of storing data outside the spreadsheet proper, and the means of storing data in forms other than the piecemeal structures often associated with highly complex spreadsheet records. However, the storage of data in VBA variables does leave one particular issue to be desired: visibility. Generally speaking, it often appears to those starting out that what happens behind the scenes in VBA is a bit opaque, or at least much more difficult to get a handle on than the kinds of calculations that take place in a spreadsheet.

In reality, this perception has more to do with the roots of those same developers as they transition from spreadsheet environments to VBA. In the spreadsheet environment it's easy to get used to data storage units (e.g., cell values) whose contents are largely explicit. In back-end VBA programming environments, commonly leveraged data storage units (e.g., variable values) are called into existence only as needed and, unless measures are taken, don't tend to display their contents as a matter of course. Efficiency, after all, is something we are looking for in VBA.

Fortunately, as suggested in Section 8.1 and earlier examples, the VBA Editor does provide the means of making the values stored and modified within these variables crystal clear to developers interested in monitoring their change throughout the course of a subroutine. We have a number of tools available to accomplish this. One, as described earlier, simply involved hovering our cursor over a variable of interest during a step-through process (F8) or the triggering of a breakpoint (F9). Another two options involve the *Immediate* and *Watches* windows.

Right-click anywhere on your code to open a shortcut menu. From there, select Add Watch (shown in Figure 8.12). This process is facilitated if you select the variable of interest and right-click directly on it.

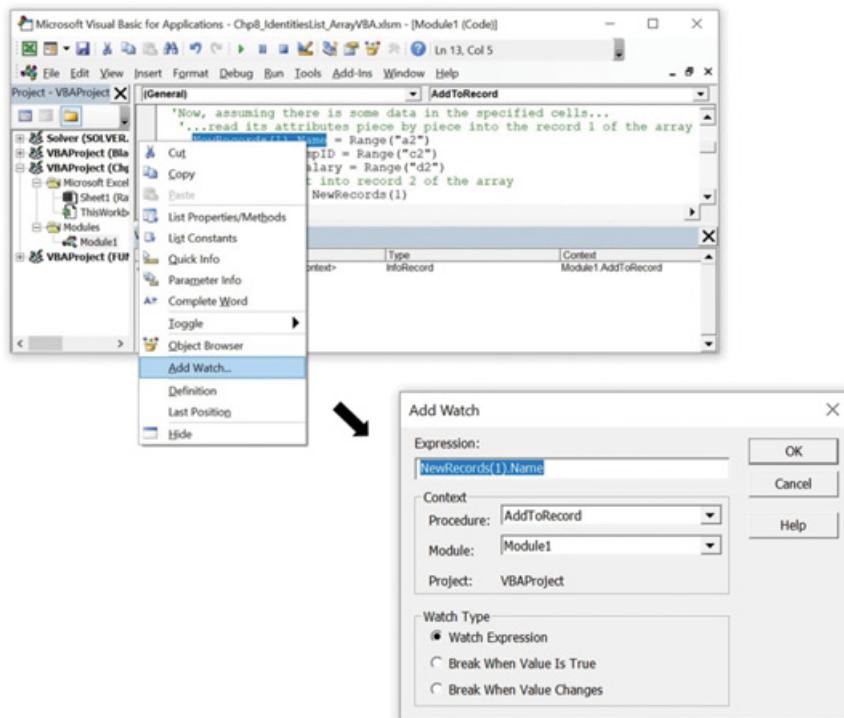


Figure 8.12 Adding a Watch to subroutine code monitoring

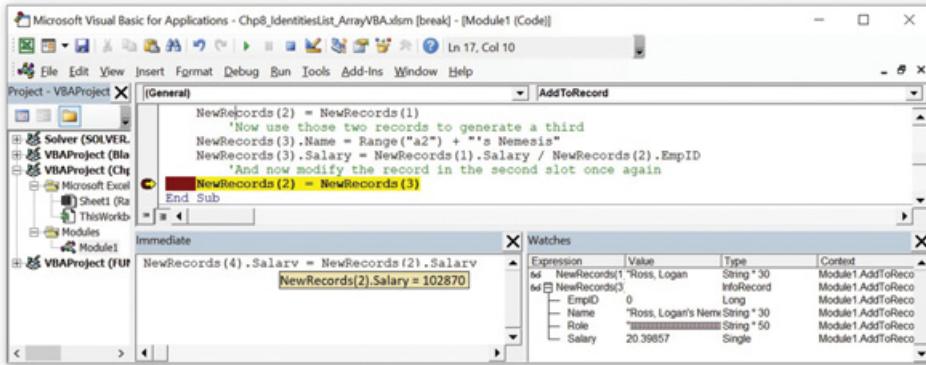


Figure 8.13 Viewing variable contents through hovering, Immediate and Watch windows

Selecting Add Watch opens the Add Watch dialog box, which enables you to specify exactly what you want to watch as you step through the subroutine (using F8, for example, or running the subroutine with breakpoints active). The Watch then appears at the bottom of the screen (Figure 8.13), which is what you'll want to keep your eyes on for changes in the variables selected. After stepping through to the end of this particular subroutine run, the Watch screen should appear as shown in the lower panel of Figure 8.13. Adding the Immediate window (under View) allows a different degree of flexibility, permitting any additional line of code to be executed, using the values of all variables at the point at which the run of your code has paused. This can provide additional tests without requiring changes to your code.

### 8.3.4 Facilitating VBA Interaction in Complex Spreadsheets

Although we can accomplish a great deal of heavy lifting behind the scenes in VBA through the use of data stored in VBA variables, we still frequently have a need to interact with the spreadsheet; often our spreadsheets provide the original source of data as well as the reporting interface for data work conducted in VBA. To avoid delays that otherwise might come from regular spreadsheet usage we can take the following precautions in our VBA–Excel interactions:

- 1) When Running VBA script, unless regular spreadsheet calculation and data updates are needed, run Excel in “manual update” mode in Excel Options. If you’d like to make that specification in VBA, the following code can be used:

```
Application.Calculation = xlCalculationManual
```

- 2) To toggle back to automatic updates, the following line of VBA code can be used:

```
Application.Calculation = xlCalculationAutomatic
```

- 3) If you want to recalculate all cells in the workbook once, or trigger a series of calculations under multi-iterative calculation mode, the following single command can be used (equivalent to an F9 hit) in the spreadsheet, as we saw in Figure 8.2 and Table 8.1.

```
Calculate
```

- 4) If you want to only perform a recalculation over a specific set of cells while in manual mode, the following code can apply (e.g., over the range B10–C15):

```
Range("B10:C15").Calculate
```

## 8.4 Common Operations in VBA

Now that we've discussed basic variable declarations and monitoring, let's move onto the more interesting question: How can we accomplish interactive analysis tasks with data available in VBA? Just as with the long list of functions already built into Excel, a comprehensive coverage of what we can do with data in VBA, as part of an interactive experience, is certainly beyond the scope of this text. However, we can cover the fundamentals that any analyst using VBA should know. These are the building blocks of all things computational and are ultimately the foundations for seamless user experiences with the decision support tools we construct.

### 8.4.1 Syntax for Basic Operators

Many basic arithmetic operators are the same in VBA Editor as they are in the spreadsheet environment (think +, -, \*, /, ^). Other mathematical functions, such as ABS() and EXP(), also work as they do in spreadsheets, as will certain nonmathematical functions such as MID() and LEN() and functions from the Blackbelt Ribbon add-in (if it is active and is referenced under Tools in the VBA editor). Other functions, such as MOD, work just a little differently, others are spelled differently, and still others require us to take the additional step of referring to the Excel application in order to access them (e.g., VLOOKUP).

In a spreadsheet, for example, we might use =MOD(12,5) to find 2 (the remainder of 12 divided by 5). In VBA code our statement is =12 MOD 5. One could ask in this case and others: Why does the spreadsheet use syntax

different from the VBA Editor? Suffice it to say that both Excel and the ancestral Basic programming language (predating VB and the current VBA) have long histories. Spreadsheet function syntax developed somewhat separately from the syntax that became standard in VBA, and the task of reconciling the two sets of syntax would be a daunting one given the amount of legacy applications in place. True, this can create a bit of dissonance and frustration, but it's also something you can get used to. Let's take a look at some of these uncanny similarities and notable discrepancies before we move on to more unique syntax.

#### 8.4.1.1 Random Numbers

In spreadsheets we can use RAND() to generate a decimal value between 0 and 1. In VBA we use “Rnd.” There’s no real difference here, aside from loss of a vowel and the lack of parentheses. RANDBETWEEN() doesn’t have a direct VBA equivalent; however, you could always make do by combining Rnd with a multiplier, converting to an integer (Int() in VBA), and adding 1. For example:

```
NewVariable = Int(Rnd * 9.999) + 1
```

Alternately, as with so many of the functions we are familiar with in the spreadsheet, we can still access the very same RANDBETWEEN() function of the Excel application, provided we precede it with the “Application.”, “Worksheetfunction.”, or “Application.Worksheetfunction.”. One form would be:

```
NewVariable = Worksheetfunction.Randbetween(1,10)
```

Some prefer “Application.” due to certain benefits in error handling, while others prefer “Worksheetfunction.” referencing since the VBA Editor (via Intellisense) provides some marginal advice as to the arguments and outcomes of functions as you type them (albeit not as richly as in the spreadsheet environment for the same functions). Each of these approaches works similarly for functions such as AVERAGE, STDEV, NPV, VLOOKUP, and so on.

As for special random numbers, such as PoissonInvBB, provided by add-ins like the Blackbelt Ribbon, a similar notation can be used. Make sure to have the add-in active. It may appear as “VBAProject (BlackbeltRibbon.xlam)”; however, you can rename it using the Properties windows so that it is distinct from your current project (e.g., “Blackbelt (BlackbeltRibbon.xlam)”). In any event, clicking on Tools in the VBA editor, and References, look for the add-in and make sure it is clicked on. From that point forward you can access any of the functions in the add-in (listed in Appendix B of this book as well). For example:

```
NewVariable = PoissonInvBB(20)
```

#### 8.4.1.2 Date/Time Functions

In the spreadsheet there are a number of ways to get and make use of date/time information maintained by the system clock. For example, the NOW() command provides the date/time signature at any given moment. Like the RAND() function, every recalculation in a sheet containing NOW() updates the value returned by that function to reflect the passage of time. What is returned by NOW() is a composite of the current date and time.

If the value is formatted correctly, then it should be easy to understand, but ultimately it's just a long decimal number; Values to the left of the decimal are the count of days since a baseline (e.g. Jan 1, 1900), while those to the right represent the fraction of the current day completed (i.e. time). On the other hand, if it's formatted like a long decimal number, it will look like one. Another spreadsheet function, TIMEVALUE(), serves in a related capacity by translating a recognizable time text string such as 3:30:15 into a long decimal value of the kind that NOW() and other time functions actually work with. DATEVALUE() serves a similar role with arguments such as January 4, 1982.

In VBA, multiple tools for accessing and utilizing system clock data are also available. For example the term “Now,” without the parentheses, can similarly be used to generate a consolidated decimal value that incorporates today’s date and time. Also, the TimeValue function works just as it does in the spreadsheet. However, in VBA the combined use of these two functions takes on particular relevance because of still more advanced functionality that does not exist directly in the spreadsheet. The Application.Wait function, a mechanism for generating delays during subroutine runs, is a prime example of how these two can be used together.

Why would someone want to intentionally add a delay to a subroutine? Sometimes subroutines, designed explicitly for demonstration or external data communication purposes, run too fast – at least, too fast to allow for a meaningful visual demonstration of dynamics, or faster than the amount of time required for an external data source to update. Application.Wait can help us here. As an extremely simple example, if at any point you want to force a one-minute delay in the middle of a subroutine, just insert the line:

```
Application.Wait (Now + TimeValue("0:01:00"))
```

That’s it. Easy to customize, easy to interpret – and it works.

#### 8.4.1.3 OFFSET, ROW, and COLUMN

In Excel spreadsheets we have a few functions that help us extract information regarding cell ranges or provide adjustments to reference cell ranges in

order to meet certain computational needs. These include the OFFSET, ROW, and COLUMN functions. You won't find these functions directly available in the syntax of VBA, and you won't find them as offshoots of the "Application." or "Worksheetfunction." notation. But they are available in VBA. Since VBA is designed to reference spreadsheet cells using the "Range" syntax, it also extends that syntax to give us access to Offset, Row, and Column functionality.

For example, in the spreadsheet environment we might use OFFSET (C5,2,4) to access the cell two rows below and four columns to the right of C5. In VBA we use the following to access the same spreadsheet data in the VBA Editor:

```
NewVariable = Range ("C5") .Offset (2, 4)
```

As in the spreadsheet environment, we are almost certainly never going to use Offset in VBA with static arguments (2,4), but rather are much more likely to supply variables or references to other cells as inputs, keeping its functionality dynamic and relevant. The same would hold in the use of Row or Column.

#### *8.4.1.4 IF Statements*

In the spreadsheet we type the following into a cell, for instance, cell D5, to set its value to 4 when C5=1, and 3 otherwise: IF(C5=1,4,3). In VBA we would use:

```
If Range ("C5") =1 Then
    Range ("D5") =4
Else
    Range ("D5") =3
End if
```

In VBA, similar to subroutines themselves, we see both statement initiation (If) as well as termination (End If) for multiline conditional statements like this. Alternative cases (Else, or Else If) are optional. At first this may seem more complex than what we're familiar with in the spreadsheet, but it actually represents a fairly robust structure into which a wide variety of actions can conditionally take place. For instance, if C5=1, I might also want to run Solver and/or launch and play a Media Player object, etc. I can do this by entering more code into the framework. I couldn't do this by just using a spreadsheet's IF statement.

To drive this home, let's consider a more complex application in which the VBA If statement proves useful. We've already taken several looks at the dot-notation structure common to what the VBA Editor writes when macros are recorded. We've also seen how to communicate with the spreadsheet

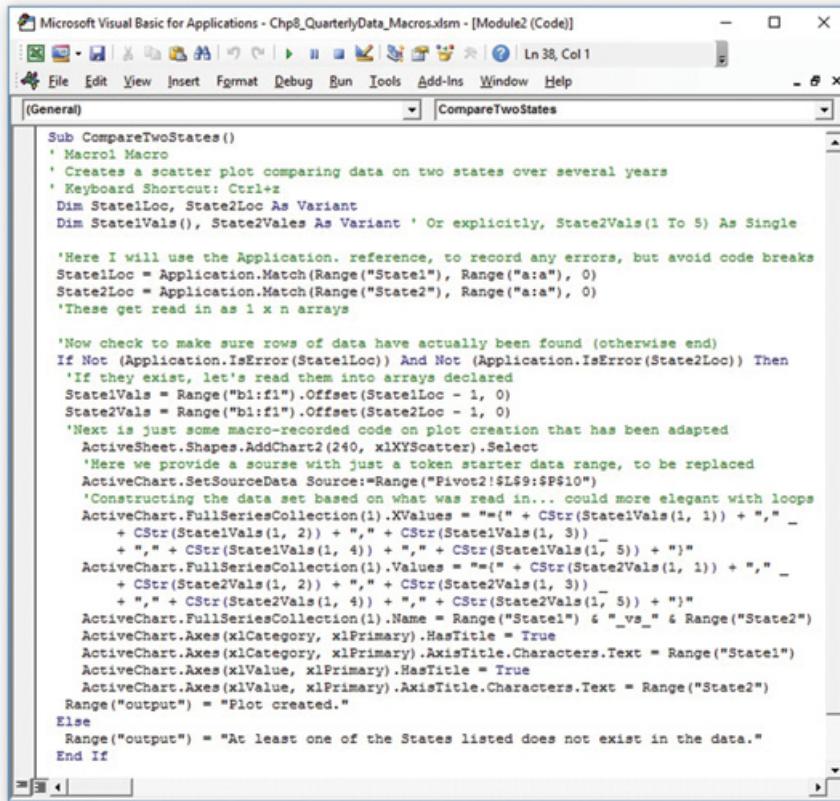
cells, the use of data storage structures, and an example of interacting with a chart. Dot notation was relevant there as well. We'll combine all of this in our simple example of a VBA If statement in use.

Starting on the Chp8\_QuarterlyData\_Macros workbook, I recorded a macro involving the creation of a simple Scatter, using a set of blank cells (not part of the Pivot Table). In that recording session I edited the chart's source (XY) data and then added a chart title and axis labels. After stopping the recording session, which was simply to give me some object code that I might have forgotten or not known about, I then went back into the recorded code and started modifying it. This allowed me to anticipate two geographical selections by a user (in cells named State1 and State2 in the spreadsheet), locate the rows of data for those states in the Pivot Table (if they exist), and use that data for the plot.

Now, since it is possible that the states selected don't actually exist in the Pivot Table (or in the event that a typo in their entry exists), I don't want to create a plot but rather just inform the user that something's wrong. Although we will talk about error handling more deliberately later on, I'm going to use a VBA "If" statement here to direct the code to do one of these two things (either create a plot or send a message) depending on the case. I'm also going to store content in variables using the "variant" option, allowing either numerical data or error codes to be stored in these for this check. Figure 8.14 shows the code implemented (Module2 of this workbook), while Figure 8.15 shows the spreadsheet results when data for both states can be found. When data are not found, an alternate message is provided.

Looking closely at the code in this subroutine (CompareTwoStates), we see things beginning with standard variable declarations, albeit in the flexible variant realm. Then we are leveraging spreadsheet functions and Range, looking for matches in the names provided, and then checking to see if matches occurred. In the latter case the "If" statement's criteria has two parts: Was there no error in searching for the first name provided? And was there no error in finding the second? Both must be true (no errors all around) in order for the bulk of the remaining code to run. If one isn't true, then the code jumps to the "Else" statement and informs the user.

If there is no error in matching the names to the list, then those matched locations are the basis for reading numerical data from the spreadsheet into the arrays (of variant size and content). These variant arrays will take on a  $1 \times n$  size and will involve two indices, the first always equal to 1. While the remainder of the code is simply an adaptation of the macro recording of a plot creation, these adaptations are dependent on content read from the spreadsheet. Token series data is replaced by a constructed string of



```

Sub CompareTwoStates()
    ' Macro1 Macro
    ' Creates a scatter plot comparing data on two states over several years
    ' Keyboard Shortcut: Ctrl+z
    Dim State1Loc, State2Loc As Variant
    Dim State1Vals(), State2Vals() As Variant ' Or explicitly, State2Vals(1 To 5) As Single

    'Here I will use the Application. reference, to record any errors, but avoid code breaks
    State1Loc = Application.Match(Range("State1"), Range("a:a"), 0)
    State2Loc = Application.Match(Range("State2"), Range("a:a"), 0)

    'These get read in as 1 x n arrays

    'Now check to make sure rows of data have actually been found (otherwise end)
    If Not (Application.IsError(State1Loc)) And Not (Application.IsError(State2Loc)) Then
        'If they exist, let's read them into arrays declared
        State1Vals = Range("b1:f1").Offset(State1Loc - 1, 0)
        State2Vals = Range("b1:f1").Offset(State2Loc - 1, 0)
        'Next is just some macro-recorded code on plot creation that has been adapted
        ActiveSheet.Shapes.AddChart2(240, xlXYScatter).Select
        'Here we provide a source with just a token starter data range, to be replaced
        ActiveChart.SetSourceData Source:=Range("Pivot2!$L$9:$P$10")
        'Constructing the data set based on what was read in... could more elegant with loops
        ActiveChart.FullSeriesCollection(1).XValues = "(" + CStr(State1Vals(1, 1)) + "," _
            + CStr(State1Vals(1, 2)) + "," + CStr(State1Vals(1, 3)) -_
            + "," + CStr(State1Vals(1, 4)) + "," + CStr(State1Vals(1, 5)) + ")"
        ActiveChart.FullSeriesCollection(1).Values = "=" + CStr(State2Vals(1, 1)) + "," -_
            + CStr(State2Vals(1, 2)) + "," + CStr(State2Vals(1, 3)) -_
            + "," + CStr(State2Vals(1, 4)) + "," + CStr(State2Vals(1, 5)) + ")"
        ActiveChart.FullSeriesCollection(1).Name = Range("State1") & "_vs_" & Range("State2")
        ActiveChart.Axes(xlCategory, xlPrimary).HasTitle = True
        ActiveChart.Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = Range("State1")
        ActiveChart.Axes(xlValue, xlPrimary).HasTitle = True
        ActiveChart.Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = Range("State2")
        Range("output") = "Plot created."
    Else
        Range("output") = "At least one of the States listed does not exist in the data."
    End If
End Sub

```

Figure 8.14 VBA code to find specified data and generate a plot

values from the State1Vals and State2Vals arrays (Cstr converting numbers to text), and the chart header is simply a construction based on the two states selected.

If you're not fond of the repeated use of "ActiveChart" (beginning eight lines of code) or of "ActiveChart.FullSeriesCollection(1)" (beginning three lines of the code), perhaps because of the amount of space such syntax takes up, there is another option. It involves use of the With/End With statement.

The With statement specifies the object, or property subset of an object, that associated property commands apply to. Beginning with a statement like With ActiveChart avoids the need for repeated reference to ActiveChart until the paired End With statement is made. You can even embed With statements within each other, provided you keep things straight. For example, some of the code from the example could be presented as below. Note that the "." dot notation continues to play a prominent role in drilling down into these properties, even in embedded With statements (e.g., With .FullSeriesCollection(1)).

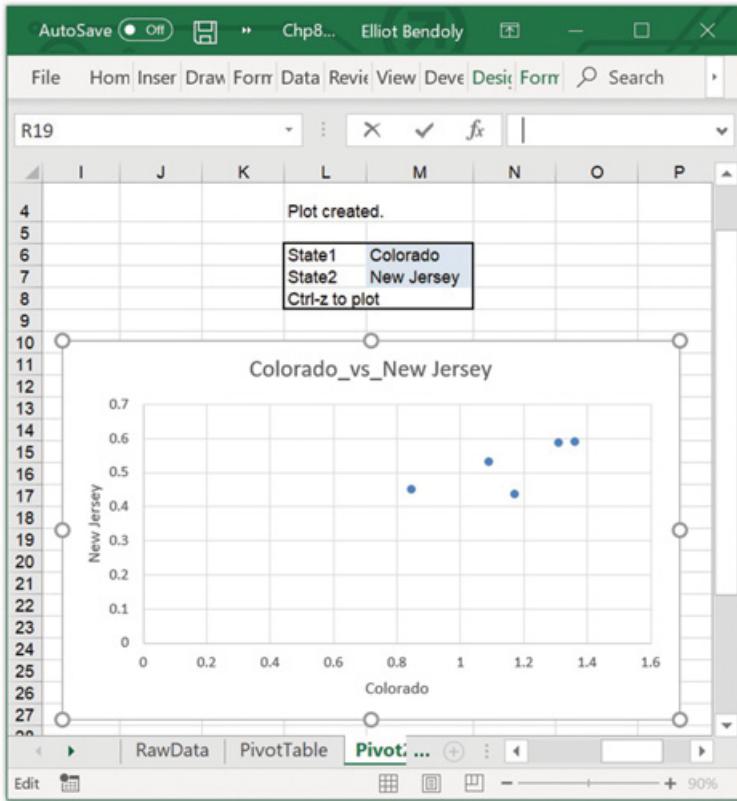


Figure 8.15 Example of plot generation by VBA subroutine

```

With ActiveChart
    .SetSourceData Source:=Range("Pivot2!$L$9:$P$10")
    With .FullSeriesCollection(1)
        .XValues = "=" + CStr(State1Vals(1, 1)) + "," + _
                    CStr(State1Vals(1, 2)) + "," + CStr(State1Vals(1, 3)) + "," + _
                    + CStr(State1Vals(1, 4)) + "," + CStr(State1Vals(1, 5)) + "}"
        .Values = "=" + CStr(State2Vals(1, 1)) + "," + _
                    + CStr(State2Vals(1, 2)) + "," + CStr(State2Vals(1, 3)) + "," + _
                    + CStr(State2Vals(1, 4)) + "," + CStr(State2Vals(1, 5)) + "}"
        .Name = Range("State1") & "_vs_" & Range("State2")
    End With
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text =
        = Range("State1")
End With

```

Now, if you're only referencing an object a couple of times, the With statement won't be that useful. But if you are referring many properties of an

object in a block of code, or repeatedly working with it, the use of With can make things more readable.

Try stepping through this subroutine with F8 and/or using F9 breakpoints. Try entering state names incorrectly. See how it works, breaks down, or might be modified by small tweaks to the code. We'll come back to this one in Section 8.6 with additional discussions of error handling.

### **8.4.2 Iteration Structures: Loops**

Are there better ways of getting done some of the work we've seen up until now? Or at least ways that make what we want to do more robust? Or more controlled and manageable? Certainly. The code structures for getting us there are very much in reach. And these tactics also apply to improvements on the sorts of things we have up until now only considered in the spreadsheet environment. Consider, for example, the use of Excel's iterative calculation mode as a mechanism for getting us through a series of sequential events in system simulation or data collection contexts. Some of this associated work is often better accomplished with code, provided we learn how to leverage structures broadly referred to as *loops*.

#### **8.4.2.1 For Loops (Fixed-Finite Iteration Structures)**

When we know how many times we want some action repeated, or we know how many items to which we want the same or similar action applied, no matter what that action is, For loops (or For-Next loops) can get the job done quickly. The syntax of these statements begins with the term For and a specification of a counter variable to begin at one value and increment toward another (e.g., Count = 1 to 200). Increments can be decimal or negative as well if additional specifications are added in line (e.g., Count =200 to 1 step -0.25).

To make this tangible, here's an example of what the code from Chp4 \_LobosInventory\_BasicModel (Figure 8.2, Table 8.1) might look like if all 200 or so of those Calculate lines were condensed using a simple For loop:

```
Sub MyFirstMacro()
    Range("H6").Select
    ActiveCell.FormulaR1C1 = "FALSE"
    Calculate
    Range("H6").Select
    ActiveCell.FormulaR1C1 = "TRUE"
    Range("H7").Select
    For Count = 1 To 200
        Calculate
    Next Count
End Sub
```

```

Next
Range("C31:C34").Select
Selection.Copy
Sheets("MacroRuns").Select
Range("A1").Select
Selection.PasteSpecial Paste:=xlPasteValues, _
    Operation:=xlNone, SkipBlanks:=False, Transpose:=True
Application.CutCopyMode = False
Selection.EntireRow.Insert
Sheets("StockoutDemo").Select
End Sub

```

In doing this, we've removed around 200 lines of code. As a whole the subroutine is also now easier to read, manage, and edit. If we at some point feel that the system should examine 1,000 days rather than 200, we don't have to add 800 lines of Calculate; We just change 200 to 1000 in our For loop. Furthermore, just as with VBA If statements, there are plenty of other actions that could take place within each iteration of that loop, such as complex calculations or even multiple calls to add-ins like Solver, or modifications to and data scraping with web queries.

For illustration, let's start simple. Consider the following code:

```

Sub SimpleForLoop()
Dim count As Integer
For count = 0 To 10
    Range("a1").Offset(count, Int(count/2)) = count
    Range("a1").Offset(count, Int(count/2)).Interior.Color = _
        count * 24 + 8953100
Next
End Sub

```

When this code runs, the numbers 0 through 10 (the values of the count variable used in the loop) will be written to subsequent cells in the active worksheet, starting at cell A1. Column shifts occur every two entries due to the inclusion of Int(count/2). The color of each subsequent cell's interior will also be modified based on a numeric color code. Each cell will have a slightly different fill color as a result, starting from green and approaching yellow. The result of the run is shown on the Simple sheet of the Chp8\_ForLoops workbook (see Figure 8.16).

This example demonstrates how the count variable in the loop can be used as much more than just an iteration mechanism. In the code it is used to specify an output value, an offset level, as well as a formatting modification.

In a more involved example, more relevant to data analytics and one that we will see again later, we could also consider using a For loop to aid in an otherwise fairly complex calculation. On the sheet NoisyROI in that same

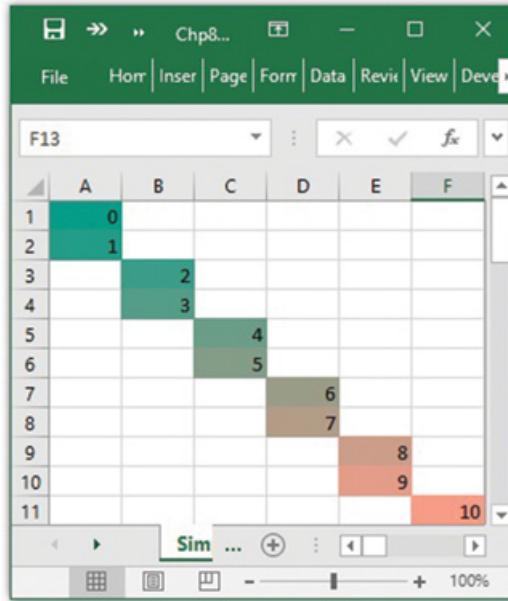


Figure 8.16 Resulting spreadsheet changes associated with a simple For-Next loop

workbook, we have the following task presented to us: Calculate the value of a compounded investment subject to variable rates of return over 30 periods. That is, starting with some initial investment, draw a rate of return from a random distribution with a given mean and standard deviation for each of thirty periods and compound the quantity invested based on each rate every period. All we want is the value of investment at the end of the 30 periods. In the spreadsheet this can take up a bit of space (as in the workbook provided; multiple rows and columns of calculation). However, if we did this in VBA we might only need a couple of inputs and a place to provide a final output. The following code does just that and doesn't require any of the spreadsheet calculations (although it runs alongside them with alternate random draws).

```
Sub CompoundCalc()
    Dim Avgrate, stddevrate, compoundreturn As Single
    Dim counter As Integer
    Avgrate = Range("avg_rate")
    stddevrate = Range("stddev_rate")
    compoundreturn = 1
    For counter = 1 To 30
        compoundreturn = compoundreturn * (1 +
            Application.NormInv(Rnd, Avgrate, stddevrate))
    Next
    Range("outcome") = compoundreturn
End Sub
```

#### 8.4.2.2 While Loops (Open-Ended Iterations)

In contrast, While loops (or Do While loops) are useful when we don't know how many times we want some action repeated or how many things we want the same or similar action applied to, no matter what that action is. These loops end when specific conditions are met, similar to the stopping logic used as part of RISK Optimizer's functionality. The syntax of a While loop begins with the statement "Do While" followed by a condition/criteria, and ends with the term "Loop." Everything in between gets repeated as long as the criterion is met. Consider the following code as an alternative to the For loop code that generates the changes in Figure 8.16.

```
Sub SimpleWhileLoop()
Dim count As Integer
count = 0
Do While count <= 10
    Range("a1").Offset(count, Int(count/2)) = count
    Range("a1").Offset(count, Int(count/2)).Interior.Color = _
        count * 23 + 8953100
    count = count + 1
Loop
End Sub
```

Here, the count variable isn't the driving mechanism of the loop. Rather it serves only to specify the stopping criteria of loop repetition (end the loop repetition as soon as count is NOT less than or equal to 10). In order for this to work, count first has to be initialized to 0, as in the code. Within the loop, count has to be incremented (count=count+1). If this didn't take place, the loop would continue to repeat itself forever – after all, the count would always stay  $\leq 10$ . This is an important point we'll mention again in a moment. Other than that, the same result as that presented in Figure 8.16 is generated. As in the previous example, count plays multiple roles in the output generation.

As an alternate example, combining the use of While loops, some of the date/time functions, as well as some simple controls, consider the construction of a simple countdown timer as provided in the Chp8\_BasicClock workbook. The main sheet appears in Figure 8.17, allowing individuals to enter the number of minutes to be counted down from, a button that initiates the looped countdown subroutine, and another button that continues the countdown if paused. The countdown can be paused by typing anything (e.g., a space) into any cell in the book.

Before getting into the code for this one, it's worth returning to a general warning regarding the use of While loops. They are very handy but notorious for causing headaches for early developers. It's possible that, perhaps due to an omission or error elsewhere in the code, the loop criterion is "always"

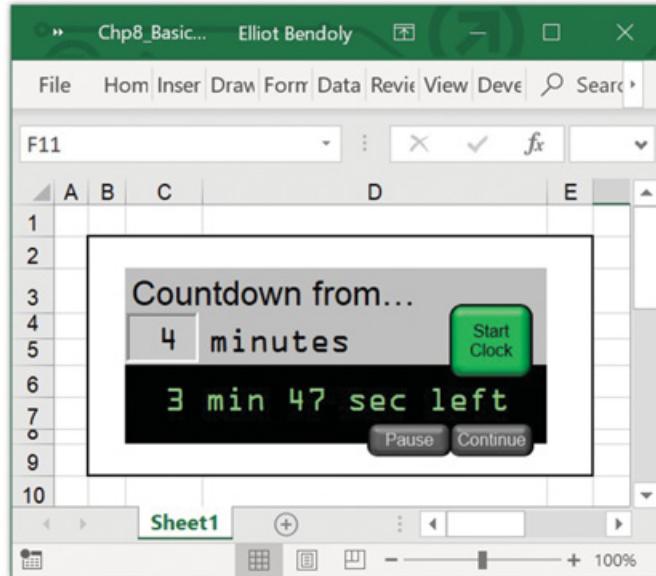


Figure 8.17 Interface for example of use of a Do-While loop

met. The result is an infinite loop that might stop only if the user presses Ctrl-Alt-Delete or when the battery dies – whichever comes first. Neither is great.

Fortunately we can build in some escape clauses, mechanisms that allow users to force exits from loops of any kind, or at least maintain more control. One option is to add a criterion to the While loop statement, requiring some other object (cell, etc.) to maintain the value TRUE, and then have a button or control key shortcut ready that sets that value to FALSE. Another option, a kind of all-purpose escape clause, is the DoEvents statement. Much like Calculate, it typically occupies its own line within a program (in this case somewhere within the While loop). Functionally, it allows items such as forms and controls on the spreadsheet to respond to actions taken by users while running subroutines that contain this line of code. It also permits direct changes to cell contents during this time, which in turn will automatically shut down the running of subroutines (without shutting down Excel).

Let's take a look at the code working behind the scenes of the Chp8\_BasicClock example. The following is a clip of the code, fully annotated in the file:

```
Sub StartClock()
    MaxTime = TimeValue("0:01:00") * Range("CountdownFrom")
    TimeSpent = 0 'Initialize this time keeping record
    StartTime = Now 'Provides the current system time
    Range("Paused") = False
```

```
Do While (TimeSpent < MaxTime) And (Not Range ("Paused")) 'Criteria
    DoEvents'Forms/controls on the sheet respond to actions taken
    'by user while clock runs. Allows general cell access
    'but stops code on entry of data directly into them
    '**Below this point, perhaps included needed calcs/refreshes ...
    TimeSpent = Now - StartTime 'Update the time keeping record
    Range ("TimeLeft") = MaxTime - TimeSpent 'Display time left
Loop
End Sub
```

Activation of the subroutine is made possible through Ctrl-m, or by clicking on the button “Start Clock” (we’ll talk more about buttons like this and other graphical user interface tactics in Chapter 9). Leveraging a While loop structure, the associated subroutine (StartClock) reads in the number of minutes provided in the ActiveX control (linked to the “CountdownFrom” cell), stores that in a VBA variable called MaxTime, and ensures a cell named “Paused” is set to FALSE. It then continues to update the amount of time spent in the countdown (a VBA variable) and updates the time left on the main spreadsheet, as long as the time spent is less than the total requested countdown (MaxTime). It will also stop (or pause) the countdown process under two alternate conditions. If the Pause button is clicked on, which changes the value in the cell named “Paused” to TRUE, the loop criterion will no longer be met. The subroutine will also stop if any cell content is manually (directly) modified, due to the DoEvents statement. With these possible exits in mind, the Continue button will allow the clock to pick up where it left off.

Use DoEvents cautiously. While it can serve as a convenient exit, it can also prove inconvenient if it triggers an accidental exit (if you accidentally enter content when something important is running and can’t easily be continued). If you want to allow interfacing with forms or controls during a subroutine run, and if you want to use DoEvents to give you that functionality, you might want to make sure that users are not actually able to modify cell content. As we will discuss in Chapter 9, this kind of safeguarding is available through worksheet protection tactics, all of which can be activated and deactivated through manual design as well as VBA code.

## 8.5 User Defined Functions (UDFs)

A natural extension of the discussion we’ve had so far regarding the values, capabilities, and structures of subroutines is to spend some time on functions, another development mechanism made available through the VBA Editor.

### 8.5.1 An Introduction to Functions

Figure 8.18 shows an incredibly simple example of a function (“formulais”) that writes out the formula content of another cell, something you would typically have to go into a cell directly to see. You might not find this function automatically built into Excel; it is not listed under Formulas>Insert Function under a typical install. Instead, this is a User Defined Function (UDF). I created it, and I did so with essentially a single line of functional code.

Now let’s take a close look at this code to see how it differs from the subroutines we’ve been talking about up to this point. Below is the VBA code that makes it possible to use the function “formulais()” in the spreadsheet.

```
Function formulais(incell As Range)
    formulais = incell.Formula 'return the formula property of incell
End Function
```

Note that the encapsulating structure here is distinct from that of our subroutines. This distinction has to do with the purpose behind User Defined Functions. The main purpose is to take inputs and kick out individual outputs to the cell that references them, or to VBA objects that make use of such output. What happens within the code of a UDF may be as complex as what takes place in the code of a subroutine, although the intended impact of many subroutines extends beyond the contents of any one cell.

The screenshot shows an Excel spreadsheet interface. The ribbon menu is visible at the top, showing tabs for File, Home, Insert, Draw, Formulas, Data, Review, View, Developer, and Search. The developer tab is selected. The status bar at the bottom right shows '100%'. The active cell is C12. The formula bar shows 'C12'. The worksheet area contains the following data:

	A	B	C
1	Below is an incredibly simply example of a function that simply writes out the formula content of another cell (something you would otherwise typically have to go into a cell directly to see)		
2	\$	388.01	<-- Here's the cell containing the actual calculation and showing the OUTPUT of that calculation
3			
4			
5	=NPV(0.1,124,234,109) <-- Here's a cell containing a function that shows the STRUCTURE of the above calculation (i.e. what was entered to get the output)		
6	=formulais(B4) <-- And here's a cell containing a function that shows the STRUCTURE of the above function call (just to hit home the point on this)		
7			

The 'Developer' tab is selected in the ribbon. The status bar at the bottom right shows '100%'.

Figure 8.18 Interface for a simple applied example of a User Defined Function

User Defined Functions, in contrast to most of the subroutines we've seen, tend to be designed for limited cell outputs but are also designed for repeated activation when needed. They are just like any other function in Excel, except they don't come standard – they need to be built by users. In contrast to being activated by a button or a selection from a "macro list," they are called directly from cells within a spreadsheet or by other code within VBA Editor, with the assumption that a sufficient specification of inputs is provided for them to do the number crunching they were designed to do.

The structure of UDFs in code, in fact highlights this role. First off, all functions begin with the key term "Function" and end with "End Function" (in contrast to Sub and End Sub). You have as much discretion with the name of functions as you do with subroutines, but if you've used that name already elsewhere in code, you can't use it again (as in the case of names used in subroutines and VBA variables). Again, names used in the front-end spreadsheet environment won't present such conflicts. Critically, functions will tend to take inputs, and therefore the parentheses that follow your function name will tend to have the declaration of variables with it. Declarations are separated by commas in these parentheses (e.g., PathLength(DataRange As Range, PointsbyRow As Boolean)). Optional parameters are placed at the end of such listings and are preceded by the term "Optional" (e.g., PoissonInvBB(PoissonMean As Double, Optional InputPerc As Double)). And lastly, most critically, in order for a function to return a value, which again is the guiding design principle here, before End Function you should have a line that assigns your named function a specific value (e.g., formulais= . . . , or PathLength= . . . ).

As for making sure that the right kinds of inputs are fed into a User Defined Function when being used in a spreadsheet (e.g., by some other user), some guidance can be made available by the application. Excel should automatically recognize the function by name, if it is present in the VBA environment. In other words, you should be able to locate it in the Formulas>Insert Function dialog box when the workbook that contains the function is open (or when an add-in that includes that function is active). Furthermore, if the additional step of "registering" the function (or perhaps all of your UDFs) is taken, additional user advice regarding the specifics of what your function expects as inputs, and what should be expected as outputs, can become available.

In Figure 8.19 I provide the code for both the function and the registration subroutine. Registration will impact the description of the function in the Insert Function dialog box and will also impact details provided regarding each input parameter in the Function Arguments dialog (which can be activated if you double-click on the function in the Insert Function box, or if you click on the "fx" icon next to the formula bar for an existing cell

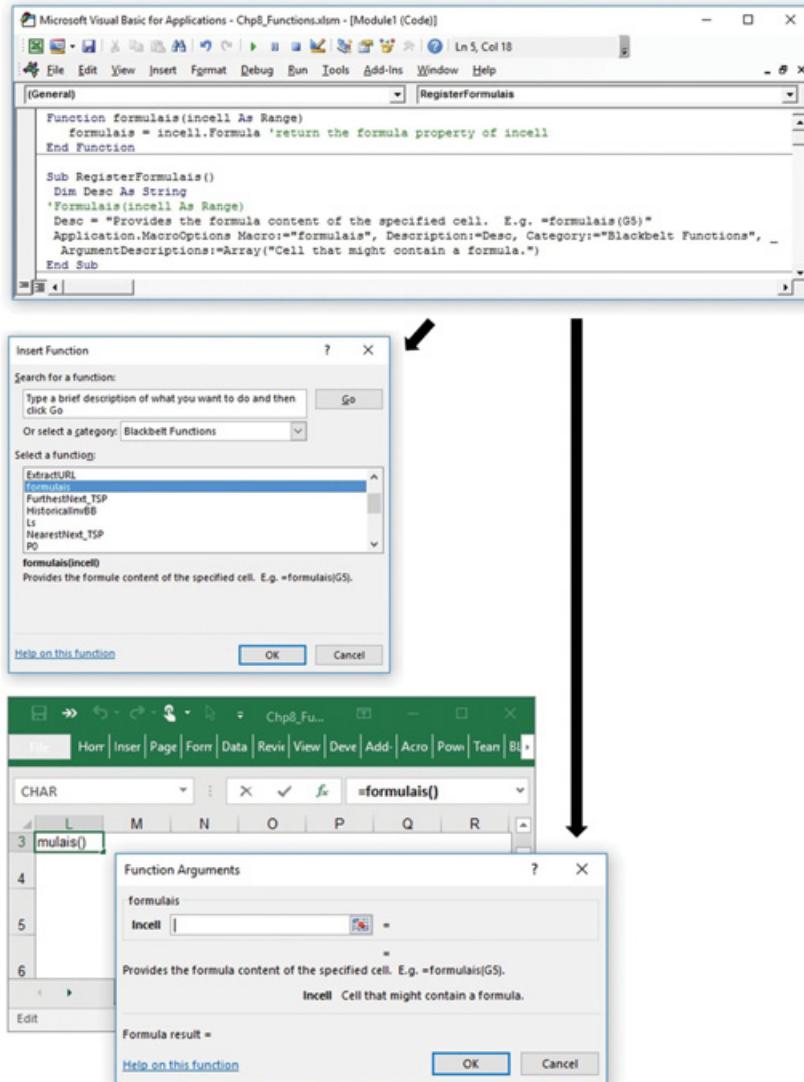


Figure 8.19 Recognition of User Defined Functions by Excel

function). It's pretty handy to have these details available both to other users as well as to developers as UDFs get edited and updated.

You could provide additional comments on the function code within VBA, but this is likely to only be to your own benefit. A typical user is unlikely to want to search for details in VBA. The more you can inform your front-end users' experience through transparent tactics, such as registration options or even user guides of some sort, the more likely they are to use what you've made. We'll see some additional tactics along these lines in Chapter 9.

### 8.5.2 More Complex Examples

Now let's consider some more complex User Defined Functions (UDFs) to really emphasize what we can do.

#### 8.5.2.1 Complex Math Made Easy: Queuing Equations

For those who remember the queuing equations from any Operations coursework you may have taken, you'll recall that they can be pretty hairy. For those not familiar with these equations, consider yourselves lucky – they aren't exactly fun to work with. As an example, the following are some of the formulae needed for estimating associated calculations, such as line-length probabilities and average wait times for situations where you have multiple servers ( $c$  could equal cashiers, clinicians, accountants, and so on) but only limited space ( $N$ ) to accommodate people waiting.

##### **Finite-Queue M/M/c Model:**

$$P_0 = \frac{1}{\left(\sum_{i=0}^c \frac{\rho^i}{i!}\right) + \left(\frac{1}{c!}\right) \left(\sum_{i=c+1}^N \frac{\rho^i}{c^{i-c}}\right)} \quad P_n = \begin{cases} \frac{\rho^n}{n!} P_0 & \text{for } 0 \leq n \leq c \\ \frac{\rho^n}{c! c^{n-c}} P_0 & \text{for } c \leq n \leq N \end{cases}$$

$$L_s = \frac{P_0 \rho^{c+1}}{(c-1)! (c-\rho)^2} \left[ 1 - \left(\frac{\rho}{c}\right)^{N-c} - (N-c) \left(\frac{\rho}{c}\right)^{N-c} \left(1 - \frac{\rho}{c}\right) \right] + \rho(1 - P_N)$$

$$W_s = \frac{L_s - \rho(1 - P_N)}{\lambda(1 - P_N)} + \frac{1}{\mu}$$

Here,  $\lambda$  is the average number of people arriving per unit time (e.g., per minute),  $\mu$  is the average time needed by a server to complete a customer's request, and  $\rho$  is the ratio of  $\lambda/\mu$ . The first function,  $P_0$ , then, represents the probability of having no one in the system (in line plus those being served) at any given moment in time, while  $P_n$  represents the probability of having exactly  $n$  people in the system. (The probability of a full system,  $P_N$ , is given when  $n$  equals the system capacity,  $N$ .) The terms  $L_s$  and  $W_s$ , respectively, represent the average anticipated number of individuals in the system, and the average amount of time an individual can expect to spend in the system (again, in line and at the counter) before their needs are filled.

The calculations of these estimation terms are not trivial. In particular, the summations over a range of values are not particularly fun to do by hand. In a spreadsheet, we could do all the required summations by setting up a table and calculating a sum of all appropriate cells in that table. In the Chp8\_Functions workbook, on the Queuing sheet, the task of trying to get multiple calculations done from scratch in the spreadsheet environment

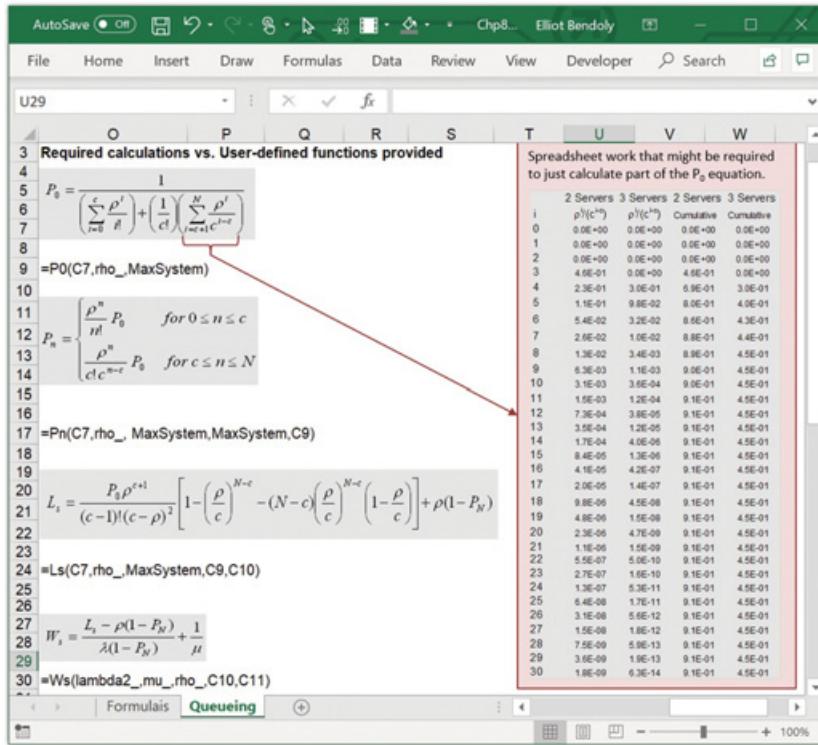


Figure 8.20 Example of extensive calculation space occupied on a spreadsheet

alone is hinted at. On the right, highlighted, we see how many cells just part of the denominator for the  $P_0$  calculations might be (see Figure 8.20).

Fortunately, we can easily create a sum of sequential terms in VBA through using loop structures such as For-Next. Not only does this eliminate the need to take up space in the spreadsheet, it's actually more efficient in VBA. In contrast to spelling all this out in the spreadsheet alone, the associated function codes (for all four functions) take up space largely behind the scenes, apart from the individual cells that call them. The left side of the Queueing sheet (Figure 8.21) gives an example of how functions are being leveraged to provide a much more frugal use of spreadsheet real estate, but also to support multiple calculations across multiple scenarios, illustrated in graphic form. Once User Defined Functions are working in a single cell, they can easily be adapted across models and analyses in a workbook, offering compatibility with such spreadsheet structures as data tables.

Looking behind the scenes at the VBA code making all of this possible, we can find the following functions, accompanied by a registration subroutine (in the file but omitted here):

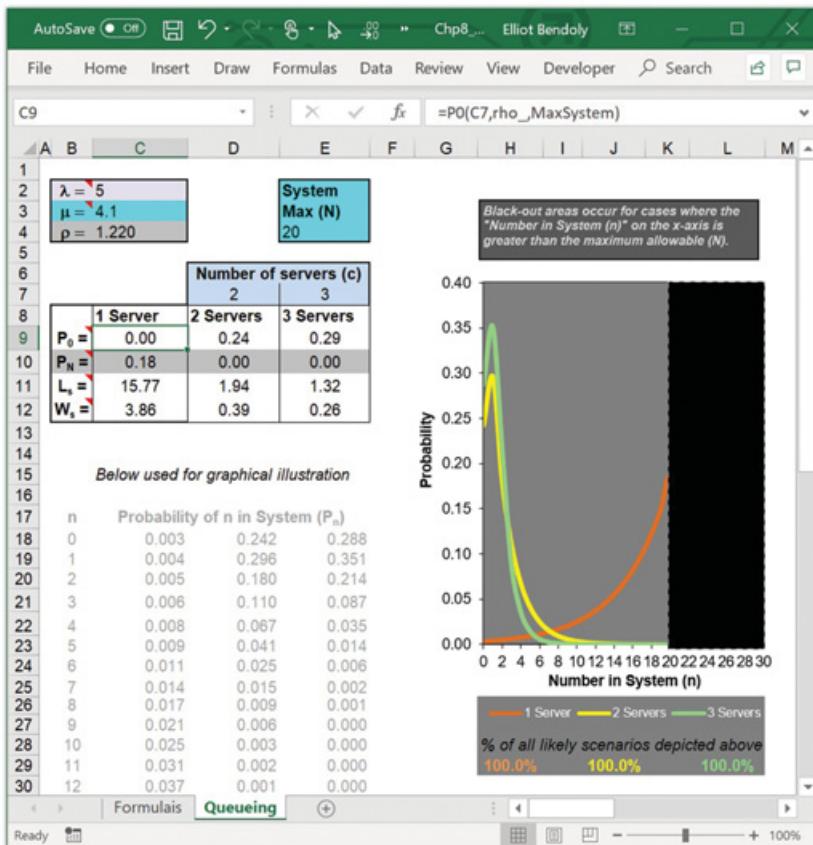


Figure 8.21 Example of function-enabled frugality in the use of spreadsheet

```

Function P0(c, rho, N As Single)
    Dim firstsum, secondsum, cfact As Single
    'declare (create) internal variables
    cfact = WorksheetFunction.Fact(c) 'calculate and store c!
    firstsum = 0 'initialize prior to summation
    For i = 0 To c 'calculate first term in P0 denominator
        firstsum = firstsum + (rho ^ i) / WorksheetFunction.Fact(i)
    Next
    secondsum = 0 'initialize prior to summation
    For i = (c + 1) To N 'calculate second term in P0 denominator
        secondsum = secondsum + (rho ^ i) / _
            WorksheetFunction.Power(c, i - c)
    Next
    P0 = 1 / (firstsum + (1 / cfact) * (secondsum))
    'Calc & return the P0 estimate
End Function

```

```

Function Pn(c, rho, littlen, bigN, Pnot As Variant)
    cfact = WorksheetFunction.Fact(c)
    littlenfact = WorksheetFunction.Fact(littlen)
    If (littlen >= 0) And (littlen <= c) Then
        Pn = Pnot * WorksheetFunction.Power(rho, littlen) _
            / littlenfact
    ElseIf (littlen > c) And (littlen <= bigN) Then
        Pn = Pnot * WorksheetFunction.Power(rho, littlen) _
            / (cfact * WorksheetFunction.Power(c, littlen - c))
    Else
        Pn = " "
    End If
End Function

Function Ls(c, rho, N, Pnot, PbigN As Variant)
    cless1fact = WorksheetFunction.Fact(c - 1)
    rhocpow = WorksheetFunction.Power((rho / c), N - c)
    product1 = Pnot * WorksheetFunction.Power(rho, c + 1) _
        / (cless1fact * ((c - rho) ^ 2))
    product2 = (N - c) * rhocpow * (1 - (rho / c))
    Ls = product1 * (1 - rhocpow - product2) + rho * (1 - PbigN)
End Function

Function Ws(lambda, mu, rho, PbigN, Lsubs As Variant)
    If Lsubs = "NA" Then
        Ws = "NA"
    Else
        Ws = (Lsubs - rho * (1 - PbigN)) / _
            (lambda * (1 - PbigN)) + 1 / mu
    End If
End Function

```

You might think this looks complex (it is). But it's no more complex than cranking out the calculations within the cells of a spreadsheet. The individual mathematical operations are simple, there are just a lot of them in repetition; and that's exactly the kind of thing that back-end code is perfect for. Plus there's still another benefit to developing functions in VBA rather than within spreadsheet cells. Building User Defined Functions for complex calculations tends to be less prone to errors than a pure spreadsheet approach. And if you find a mistake, it's a lot easier to correct for it in code than across a wide range of interdependent cells.

#### *8.5.2.2 Data Access and Presentation: Online Stock Data*

As a different kind of example, let's recall Chapter 3's discussion of linking Excel to external data sources. Our example focused on drawing in stock data from sources like Yahoo, Google, and others. In fact there are multiple sources for this information. Each of these sources tends to organize its

information intuitively, permitting the location of specific stock data both manually and through automation (e.g., VBA). Consider the following code in Chp8\_TickerPFunction:

```

Public Const QueryLocation = "QuerySheet"

Function StockData (Ticker As String, Optional num As Single)
    'This function returns price associated with Ticker;
    'Updates when any input is changed (including the optional)
    Dim Found As Integer ' location of ticker in query
    'Next 2 lines update a pre-existing query based on the user inputs
    Worksheets (QueryLocation) .QueryTables(1) .Connection = _
        'URL;https://finance.yahoo.com/quote//" + Ticker + "/"
    Worksheets (QueryLocation) .QueryTables(1) .Refresh _
        BackgroundQuery:=False
    ' Now find the ticker and price location within the query
    Found = 0
    Do While WorksheetFunction .IsNumber (Range (QueryLocation _ 
        + "!a1") .Offset (Found, 0)) = False
        Found = Found + 1
    Loop
    ' And return the Price to the function (and cell if used)
    StockData = Range (QueryLocation + "!a1") .Offset (Found, 0)
End Function

```

The assumption in this code is that the direct destination for the data pull from the web is a designated worksheet (in this case named QuerySheet). The inputs to this UDF include a ticker symbol (of type string) and an optional parameter allowing for live updatability. Specifically, it takes in a numerical parameter, which could be a RAND(), with changes to that value triggering a recalculation (requery) of stock data – a bit like the AdjustColors function from Chapter 5. A result of the multiple application of this UDF to four separate tickers is presented in Figure 8.22. In this case the content of cells A2:A5 represents the secondary parameters, which can in turn trigger query updates.

We'll return to this example in Chapter 9 to see how we might further develop a more intuitive and guided query and calculation interface based on this approach.

## 8.6 Error Handling

Regardless of how careful we are in coding and how much time we invest in debugging, there is always some risk that the person we hand our subroutines and UDFs to won't use them as intended or will perhaps encounter scenarios we couldn't have dreamed, resulting in failure on the part of our code. There are several approaches we can take to minimize this risk. On the front end

Table 8.2 Typical error handling options in VBA

Syntax setup (pre-error)	What it does when an error is detected
On Error GoTo Handler	Code advances to the “Handler” line and executes any code immediately following it. “Handler” can be any term you want that hasn’t already been used or defined elsewhere (except 0, -1)
On Error Resume Next	Simply advances code to the line immediately following wherever an error takes place (ignores the error line)
On Error GoTo 0	Disables error trapping/handling in the subroutine

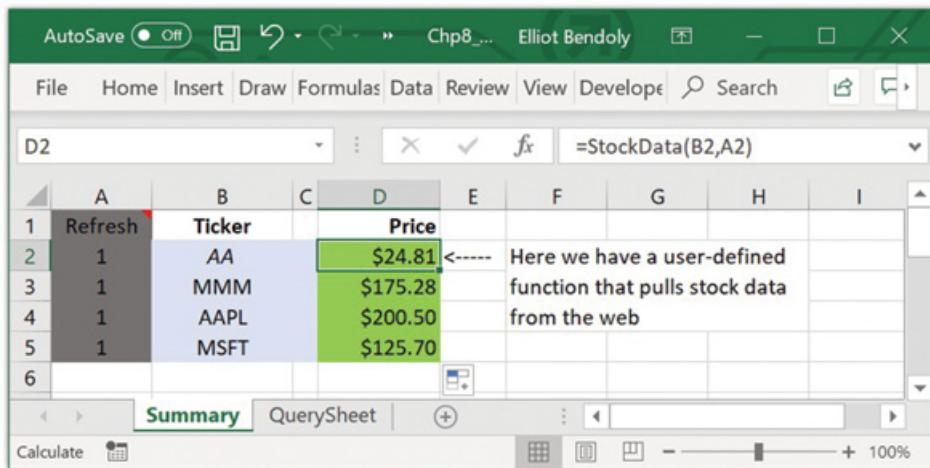


Figure 8.22 UDF for pulling stock data from the Internet

there are foolproofing tactics we can apply to help guide users down the path of “correct use.” We’ll discuss these in Chapter 9. However, at the back end there are also some nice capabilities available. They generally fall under the header of Error Handling. Table 8.2 presents the most common forms of error handling using “On Error.”

Let’s look at an example of these error handlers at work. We could draw on any one of the previous examples in which the processing and/or manipulation of inputs was central. If those inputs aren’t precisely what the subroutines or UDFs were expecting, things could go a bit sideways. Take the example in which a scatter is generated through the selection of two states in a list of possibilities (Section 8.4.1.4 on IF statements). It’s true that in the design there was some anticipation of potential errors; nothing happened if either state specification could not be found in the data. But even if both states are found, errors can occur.

The subroutine expects five data points per state. What if fewer exist in the data (e.g., for Iowa in this example). Running the subroutine will halt the code run and present “Run-time error ‘1004’: Application-defined or object-defined error,” which is to say, the scatter plot doesn’t know how to deal with the occasional absence of data points in a series string. How should we deal with this possibility and others we might not think of?

That’s a double-barreled question. If we can anticipate an error, we can certainly deal with it deliberately (as was done with the use of IF statements checking on the existence of state selections in the set). But if we can’t anticipate all errors or don’t want to trap each uniquely, the On Error options are worth considering. In this case introducing “On Error Resume Next” will allow the code to run without a break, but it will generate a scatter plot with no data points – a bit misleading. On the other hand, we could create a generic trap with “On Error Goto Handler” such that this error and perhaps others might be dealt with more meaningfully. For example, we might include the On Error Goto Handler after our variable declarations (commented out in Chp8\_QuarterlyData\_Macros for the earlier example) and add the code shown in Figure 8.23 between the End If and End Sub statements (also commented out in the earlier example). The earlier End If and End Sub are shown.

In this case we include an Exit Sub statement after the primary work; we don’t want the code associated with the Handler to run if nothing is left to do. Further, we see the availability of noncell messaging by way of the MsgBox command line, which allows details to reach the user without occupying potentially critical spreadsheet cells. It might make us rethink the earlier messaging regarding possible state typos.

```

Microsoft Visual Basic for Applications - Chp8_QuarterlyData_Macros.xlsm - [Module2 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
[General] CompareTwoStates
End If
Exit Sub
Handler:
Description = "Error " + CStr(Err.Number) + " : " + Err.Description + Chr(10) -
+ "Please check to make sure each states has " + CStr(5) + " data points."
MsgBox Description, vbMsgBoxHelpButton + vbExclamation,
":(- Sorry, can't create plot.", Err.HelpFile, Err.HelpContext
End Sub

```

Figure 8.23 An error handler affecting the flow of code and messaging

Also as suggested in this example and particularly salient to this discussion, when errors take place, information about them is temporarily recorded by the system and is therefore accessible by code, which is extremely helpful. In fact, the MsgBox is particularly designed to work with this kind of information (as shown). Here are some examples of common details and actions available to error handling procedures.

**Err.Clear** is called solely to clear the most recent error from the error structure. In the event you fail to clear the error, it would persist and be called again (syntax: Err.Clear).

**Err.Number** is the “ID” number for the event itself (syntax: If Err.Number=6 then ... ).

**Err.Description** is a string expression describing the error raised [syntax: MsgBox (“Uh oh!! : “ & Err.Description)].

**Err.Source** is a string displaying the name of the object or application that generated the error raised [syntax: MsgBox(Err.Source & “friggin’ messed up again”)].

**Err.HelpFile** is the fully qualified path to an MS Windows \*.hlp file used by your application.

**Err.Raise** is called to force an error of the specified, optional properties to occur. Therefore, you can call up the reference to an error at run time to generate a list of all errors, display a specific help context ID, or display a help file for specific errors to the user (syntax: Err.Raise 6).

It is also worth knowing what specific kind of errors (Err.Number codes) are most frequently encountered by new developers. Here are a few:

- 0:** No Error
- 6:** Overflow Program error (when calculations exceed what can be stored)
- 7:** Out of memory (when in a code run, the total amount of available memory for temporary storage is exhausted)
- 9:** Subscript out of range (when you reference beyond the range of an array)
- 11:** Division by 0 (pretty straightforward but common)
- 13:** Type Mismatch (e.g., when you try to assign a text value to a variable defined as integer)
- 35:** Sub or Function not defined (when it looks like your code is trying to call a function or subroutine that doesn’t exist)
- 438:** No such property or method (e.g., when trying to access the “funk-level” property of a cell ... or using the command “Range(“A1”).chillout”)

## 8.7 Increasing Access to Code: Creating Add-ins

If, at the end of all of your efforts to develop VBA subroutines or UDFs, you feel they would be useful across various workbooks that you and others will use in the future, you do have an additional way to save your work. If what

you have created is not dependent on the specific front-end contents of your workbook (i.e., stuff in the spreadsheet environment) or if you expect people to use your tools in similarly structured workbooks, you can consider saving the code you've developed as an add-in. I'm going to emphasize "saving the code" here, because that's really what add-ins are; Files that contain elements such as modules (and Userforms, etc., as we will see in Chapter 9), which may work in conjunction with Excel's spreadsheet environment, don't actually retain spreadsheet work in themselves. This means that some development efforts will be very much primed for this transition, while others may need considerable further development to make them more independent of the workbook and spreadsheet structure contexts across which they might be used.

For example, the Chp8\_ObjectTrials and Chp8\_TickerPFunction workbooks rely on elements that exist in the spreadsheet environment (images, media, queries). These could be adapted to also create these spreadsheet elements, search, or accept input from users as to what to interact with in other workbooks, but that would require some additional code work. In contrast, the Chp8\_ForLoops and Chp8\_Functions workbooks are largely spreadsheet independent. The CompoundCalc subroutine (Section 8.4.2.1) does expect a couple of inputs from users, but that's just a matter of naming a couple of cells and providing values. In fact this subroutine is poised for redevelopment as a function, where one could prompt the user directly for those inputs and possibly others (see the Practice Problems). The Chp8\_Functions workbook already has working functions, so it's pretty much ready to go (assuming that you are OK losing the tabular and graphical summaries based on these).

With this in mind, to transition from a workbook to an add-in, all you need to do after saving the file as a workbook (always do this so you can easily edit it later) is go to File>Save As, and select Excel add-in. The typical location for add-in saves will be a subfolder in the user folder of your computer (something like ... >Roaming>Microsoft>Add-ins). That's where Excel will go by default to look for add-ins to include as options in the future. However, you can navigate to any preferred location for storage that is convenient for you. At this point, if you follow File>Options>Add-ins and click on the Go ... button at the bottom of the dialog box, yours should appear on the list. When active (checked) it will be available in any associated workbook on your computer, as demonstrated in the case of a UDF (P0) within the Chp8\_Functions file in Figure 8.24.

If your add-in is a subroutine that is Ctrl-key activated, that same Ctrl-key activation can be used if the add-in is selected as active (unless it is being overridden by another add-in that has the same shortcut). In Chapter 9 we'll see how to develop strong front-ends in these applications. Stronger user

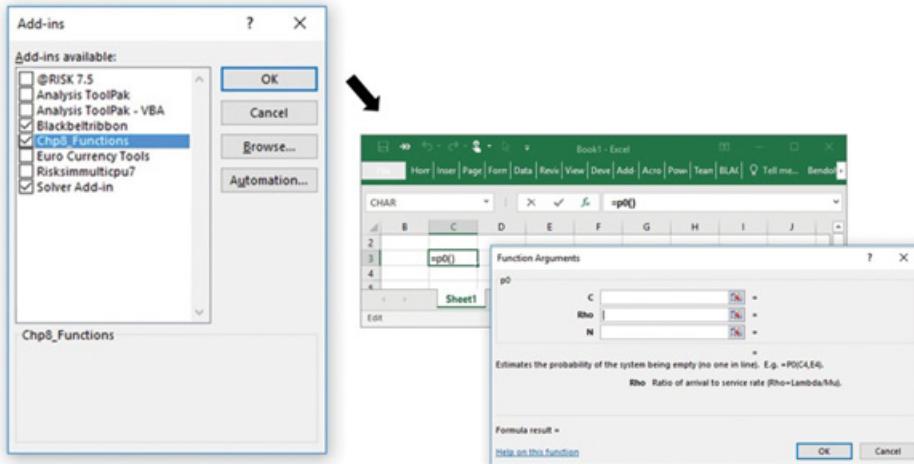


Figure 8.24 Excel's recognition of add-ins and UDFs included in add-in files

interfaces can also be saved as part of an add-in, making for a very nice packaging of otherwise complex work.

### Supplement: Coding Parallels in iOS Swift

We've now seen code and code structures that serve as the bedrocks to basic automated tool development and enhancement in MS Office products such as Excel. In Chapter 9 we will look into the kind of code that is needed for bridging Excel to other applications in seamless automated ways and for the development of interactive graphical user interfaces.

You might be wondering whether these lessons extend in any way to other contemporary languages and coding environments. The answer comes in two parts. First, programming logic in conceptual form is broadly universal. That is, if you understand the value of conditional statements (If), and loop structures (For, While loops), and appreciate the value and structures through which data can be stored and application resources leveraged, that same understanding and associated tactics for solving problems can apply regardless of environment. Syntax is another issue, though even the syntax we see across markedly different platforms will tend to have some similarities. That will make the code you find in other environment at least close to readable, if you are familiar with VBA, though you'll still need some practice in the alternative syntax before writing it yourself.

Let's look at an example body of code from an environment that is notably removed from MS Office applications: the Apple iPad Playground Book environment. The coding language here is Swift and generally not something

you are going to be developing tools with outside of a Mac running XCode, an iPad running Playground Books, or a practice learning app. But it is a critical language to get to know if you're truly interested in application mobility.

The code in Figure 8.25 should look somewhat familiar given the discussion from Section 8.5.2. It's another queuing system calculator (for just the P0 and PN terms). But it's in Swift rather than VBA. Starting at the top, you should see some parallels; after all, the same math ultimately needs to

```
//first summation in P0 denominator
for i in 0...c {
    idoub = idoub + 1.0
    if i > 0{
        ifact = ifact * idoub
        rhopowi = rhopowi * rho
    }
    invPnot = invPnot + rhopowi / ifact
}
//second summation in P0 denominator
var cp1 = 0
cp1 = c + 1
cpow = 1.0
//note that rhopowi now continues to build on prior power calc
for i in cp1...N {
    cpow = cpow * cdoub
    rhopowi = rhopowi * rho
    invPnot = invPnot + rhopowi / ( cpow * cfact )
}
Pnot = 1 / invPnot
//now build Pn for various levels of n
var rhopown: Double = 1.0
var ndoub: Double = 0.0
var nfact: Double = 1.0
var cpownew: Double = 1.0
// this first part accounts for the -c power
for i in 1...c{
    cpownew = cpownew / cdoub
}
//this next part builds the main component
for n in 1...N{
    ndoub = ndoub + 1.0
    nfact = nfact * ndoub
    rhopown = rhopown * rho
    //cfactnew = cfactnew * ndoub
    cpownew = cpownew * cdoub
    if n < c{
        Pn = (rhopown * Pnot) / nfact
    }
    else {
        Pn = (rhopown * Pnot) / (cfact * cpownew)
    }
    Pn = Pn + 0.0
}
```

Figure 8.25 Swift Playground book code emulating the two VBA queuing functions

happen. We see here the same standard math operators, the presence of For loops, conditional statements, and so on – even similar variable declarations.

In the use of calculations the same rules apply. Left of the equal sign still is the destination for calculations on the right of the equal sign. This doesn't change. Comments are a bit different, starting with “//” rather than an apostrophe. Loops and If statements don't have end statements but are defined in their influence by curly brackets “{” and “}.” Variable declarations begin with “var” rather than “dim.” But otherwise there's not much stopping us from understanding what's going on here.

When it comes to simple operations that are complex in their assembly, the ability to decipher code is pretty transferable. Things do get more complicated when it comes to the syntax used in interacting with objects such as graphical elements, or user controls, or in interactions with external applications; but if you gain familiarity with the properties of these things in one language, that understanding too is transferable. It just becomes a matter of finding the right syntax. But at least you'll know enough to ask the right questions in that search. And you might even have a proof-of-concept example to share, built in VBA, to make that search easier.

## PRACTICE PROBLEMS

### *Practice 8.1*

Import a flat image (picture) of your choice into Excel. Create a subroutine that uses the time functions in Excel to make the image slowly fade away over a 15-second period. Then, make the image slowly reappear, again over a 15-second period. Do this by either building the subroutine from the ground up or leveraging code from a macro-recording session. Hint: consider modifying the size of the SoftEdge (its Radius) of the picture, a formatting option.

Here's another hint: For the time functions, use whichever you think are appropriate, but it's actually good practice to do a little snooping around because there are multiple approaches to getting this information (see the functions under the category Data & Time). You may find, for example, the Now function to be useful, as well as the HOUR, MINUTE, and SECOND calculations typically found in Excel. Use Excel's help function to learn how these may help, regardless of how you use these in VBA.

### *Practice 8.2*

Complete the following multistep task in code:

- a. Create a function, called CircY, for calculating the y coordinate associated with the x coordinate of a circle. Your input should be the value of X, the radius of the circle, and some Boolean input that designates whether you want the upper

(TRUE) or lower (FALSE) half of the circle to be referred to for your value of Y. The standard formula for a circle is:

$$R^2 = X^2 + Y^2 \text{ or } Y = +/\sqrt{R^2 - X^2}$$

- b.** Take the code from the CompoundCalc example and convert it into a function called CompoundFunc. Your function should allow a user to provide the average and standard deviation of a rate of return, as well as the number of periods over which compounding is to take place.
- c.** On a sheet of your workbook, put the number 1 in cell A1. In B1 use CircY to get the upper Y value for an X at a value of cell A1. In C1 use CompoundFunc to estimate the value of \$1 invested at an average rate of 10 percent per period with a rate standard deviation of 0.5 percent, over a number of periods equal to the value in A1. Place the values 2–30 in cells A2–A30. Copy your functions down to occupy cells B2–C30 and process those inputs.
- d.** Save your two functions in a single add-in and confirm its functionality in a blank workbook when your original workbook is closed. In your submission of your original coded workbook, include a screen capture of these functions repeating step “c” in that new workbook.

# 9

## Application and User Interfacing

**Objective:** *Gain a broad understanding of tactics for interfacing with associated applications, as well as approaches to graphical user interface development through Userforms, designs for user experience enhancement, and add-in packaging.*

As you've probably guessed by now, decisions can become increasingly complex as we increase the number of variables and constraints in attempts to maintain reality and practical relevance in our decision-making process. Such complexity further challenges the ability to concisely provide visualizations of what is possible, what is recommended, and how these recommendations differ from alternatives and benchmarks. It mandates that we approach any guidance we intend to provide from a systems perspective, appreciating both the complex systems of practice that we work in as well as the technical systems that can help us solve problems. As Bendoly and Clark (2017) argue, the visualization of real-world systems requires systems of visualization. Similarly, analytical guidance in the support of decisions in complex professional systems requires decision support systems that guide analysis.

Sometimes these systems of guidance take the form of what we refer to as "dashboards." From a general decision-making perspective, these are interfaces that allow individual users to simultaneously view various renderings of data and information, as well as various aggregations and subsets of relevant data. Figure 9.1 presents several examples of simple dashboards that have been put into use for research and consulting purposes in the recent past.

Proceeding clockwise from the upper left, we have an interface designed for clients in practice, a dashboard designed for a public consumer audience, a proof-of-concept design for the development of a teaching mobile app, and an application designed for studying human and group behavior. You'll notice that each of these consists of multiple frames for communicating and interfacing. Some make use of interactive controls more than others. Some make use of graphs and charts predominantly, whereas others make rich use of tables with key indices summarized. All of them were designed as



Figure 9.1 Several examples of dashboard developed in Excel

applications that could function through the use of Excel alone, and are highly mobile from a distributional perspective. And, critically, some of these ultimately evolved to take forms that transcended the workbook environment, either as ubiquitous add-ins or as robust enterprise solutions entirely independent of, though still compatible with, MS Office applications.

Now there are obvious advantages to integrating into DSS designs the wide range of capabilities made available through tools such as the Blackbelt Ribbon, Solver, Power BI, and RISK Optimizer. In reality, high mobility is not often a key requirement of decision support systems and can easily be overshadowed by the need for advanced application integration. The most critical issue is practical usability paired with honest and ethical rigor on the part of developers. A detailed understanding of user needs from both a strategic-value perspective (driven by real business goals and decision requirements) and a tactical-use perspective (appropriate context-specific metric depiction and user-oriented visualization/control) is essential to ensuring these attributes.

Unfortunately, in many cases, it may be impractical to assume that one dashboard design applies to the myriad users that a DSS is designed to assist. For this reason the best dashboard designs increasingly allow individual users to independently choose and alter visual presentations, analyses, and

data usage. We've already seen some simple mechanisms that might help with proof-of-concept development toward such flexibility. Pivot Tables and the use of check-box-triggered pruning mechanisms are in this ballpark. Excel's Change Chart Type and QuickLayout options, or the ability to simply adjust the scale of axes in graphs, also represent a rough approximation of the kind of user customization possible in the structuring of graphical depictions. We might not rely on these options directly, but nevertheless such choices can be extremely helpful in thinking about dashboard flexibility opportunities.

Ultimately, even greater flexibility comes into play through the full capitalization of subroutines and functions to reveal customized analyses and visualizations on an as-needed basis. The avenues made available through the utilization of behind-the-scenes VBA coding provide the ability to avoid superfluous data, graphics, and controls that, presented in an extraneous fashion, can lead to confusion, misuse, and disastrous professional decision making. Granted, the art of managing access is often one of the last issues considered in DSS design, particularly for those just starting to hone their skills in development; however, it remains a critical element of DSS excellence overall. With this in mind, the following final sections outline approaches to augmenting DSS interfacing, both with regard to front-end user interaction as well as back-end application integration. We'll begin with the latter, as it can be instrumental in driving design decisions on the front end.

## **9.1 Application Automation and Integration**

Fortunately, many applications such as Solver and RISK Optimizer can not only be leveraged through the primary interfaces with which they were designed; they can also be called from behind the scenes through the same Visual Basic for Applications (VBA) developer environment discussed in Chapter 8. From a decision support development perspective, there are several advantages to making such calls from behind the scenes.

First and foremost, behind-the-scenes control can eliminate the need for users to become acquainted with alternative interfaces in the course of using a DSS that leverages their capabilities. Another advantage is the potential avoidance of outputs that automatically accompany the use of these applications, but are nevertheless visual and information distractions from the main point of the DSS design. The appearance of seamlessness in a designed DSS is also facilitated by VBA-driven automated calls to applications. This has the potential for engendering greater confidence in the developed DSS, as well as in its developers, and, accordingly, increased usage and user retention.

### 9.1.1 Calls to Solver

Solver is perhaps the best place for our discussion of automation and interface enhancement to begin. We're already familiar with its capabilities and limitations. We're also familiar with the extent to which its outcomes can be sensitive to the specific numerical assumptions that go into the problems Solver tackles. VBA-based automation of Solver can open a lot of doors for us. It can prove extremely convenient, for example, if we want to consider integrating Solver with simulation analysis, or with highly structured what-if reporting. It can also be critical if we wish to increase visibility and access to modifications that we (or others) might make to the objective, the set of utilities subject to change, or constraining connections. It can even allow automatic tweaks to search processes (engine specifications) if we feel we that we've hit walls in the use of others.

We just need to make sure VBA understands what we're trying to do, which in turn involves making sure we have the right dictionary in hand. Let's consider what we have to work with in this respect. As an experiment, the following is the kind of code we would get if we simply took a workbook that contained an existing math-programming problem structure (e.g., Chp9\_Examples\_wVBA), and started recording a macro right before we asked Solver to come up with a solution.

```
Sub Macro1()
    SolverOk SetCell:="$C$16", MaxMinVal:=1, ValueOf:=0,
        ByChange:="$C$22:$C$23", Engine:=2, EngineDesc:="Simplex LP"
    SolverSolve
End Sub
```

Even though the VBA Editor created this code, and it correctly respecified the utilities and objectives in the Columbus Professional Training problem in this case, it is not likely to be able to simply replay this procedure without getting a little more information from us. In fact, oddly enough, it may not even seem to recognize elements of the code it has just written. If you're able to record the same action into code comparable to that just presented, and then ask the VBA editing environment to rerun that code, you might encounter a message similar to that shown in Figure 9.2.

The problem here is that although the VBA Editor was told how to turn those actions into code, it wasn't told what to use when executing that code. The Editor needs to know that it should be referencing Solver in running this code. To make the VBA Editor aware of this, we need to formally add Solver as a reference, similar to how we added in Solver as a tool for the Excel workbook in the first place. In the VBA Editor go to Tools>References to open the References-VBAPrjct dialog box shown in Figure 9.3.

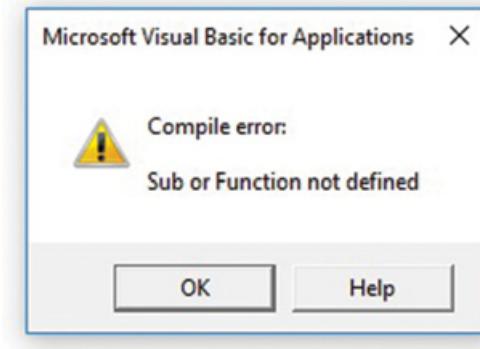


Figure 9.2 Message due to nonreferencing

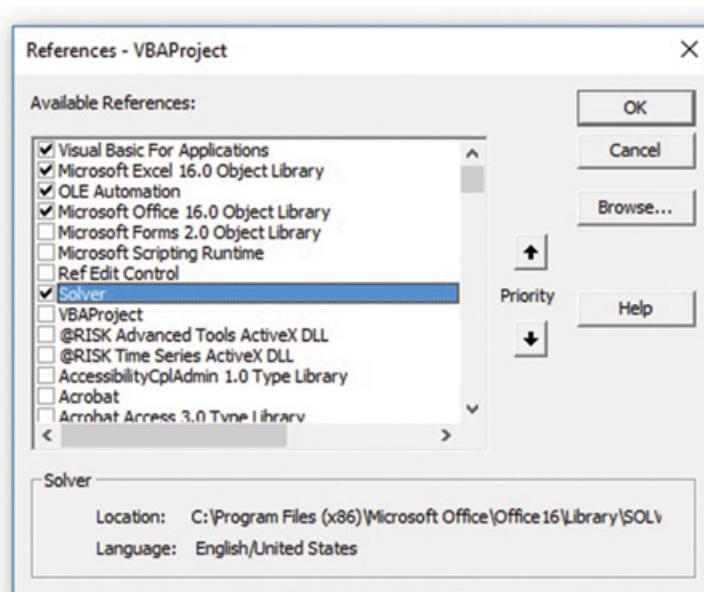


Figure 9.3 Referencing Solver in VBA Editor

Check the SOLVER box in the Available References list and then click OK. Solver is now formally a reference for the VBA Editor, acting as a kind of dictionary, or codex, for VBA to communicate commands to Solver. In other words, in making this specification in the VBA environment, we should be able to run subroutines that control Solver. And we have a lot of options in this regard.

The list of Solver-related commands that can be called through VBA is as extensive as the number of options made available by Solver. The following

is a subset of some of the more useful commands from a standard DSS standpoint:

- SolverSolve UserFinish:=True: This command allows Solver to accept the final solution it comes up with and save the associated values of the solution's utility variables in the associated cells. This way you don't have to click OK every time a subroutine running Solver comes up with a solution.
- SolverReset: This command clears all Solver contents (e.g., it deletes all constraints, objectives, and utility variable references that would typically be retained after each Solver run). This could be useful if you decide to create a program that uses different kinds of constraints in alternative iterations.
- SolverDelete "commands," for example, SolverDelete CellRef:="\$B\$2:\$C\$2", Relation:=4: This example of a selective deletion command removes a specific rule constraining two cells (\$B\$2 and \$C\$2) to be integers (i.e., that's what "Relation:=4" designates).
- SolverAdd "commands": The converse of selective deletion commands. In general, Solver allows for five kinds of relationships to be depicted by constraints as shown in Figure 9.4.

As an example of such code in use, the following two commands can be used to add individual constraints for the  $\leq$  (Relation:=1) or integer (Relation:=4) types.

```
SolverAdd CellRef:="$B$2", Relation:=1, FormulaText:="10"
SolverAdd CellRef:="$B$2:$C$2", Relation:=4, FormulaText:="integer"
```

As you can imagine, specifying a large and complicated optimization task would involve still more lines. The following code, for example, is the type of thing you might include to specify the nature of Solver's Evolutionary engine.

```
SolverOptions PopulationSize:=100, RandomSeed:=0,
    MutationRate:=0.075, Multistart:=False, RequireBounds:=True,
    MaxSubproblems:=0, MaxIntegerSols:=0, IntTolerance:=0,
    SolveWithout:=False, MaxTimeNoImp:=30
```

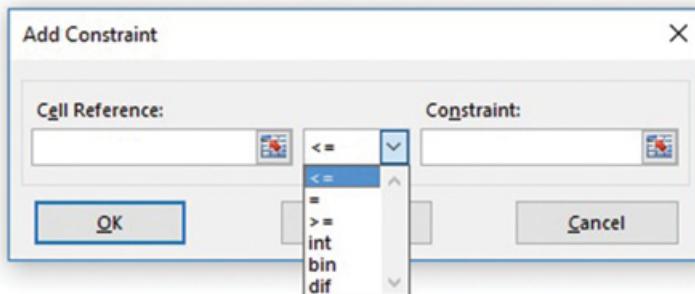


Figure 9.4 Menu interface for editing constraints

However, and this is critical, if all you want to do is call Solver to provide a solution to a prespecified problem (one that might have been specified manually within Solver at some point, or programmatically), all you really need is the following two lines.

```
SolverOk
SolverSolve UserFinish:=True
```

In fact these two lines are sufficiently generic such that a subroutine calling these could be used in any sheet containing distinct Solver optimization specifications (as provided in Chp9\_Examples\_wVBA).

And what about creating a record of Solver's progress as it examines solutions in a problem terrain? The Shadow tool in the Blackbelt Ribbon is capable of handling this, as demonstrated in Chapter 6. The code the tool requires you to put into action, which was in fact put in play in the Chp6\_JavaShadowing workbook, is shown in Figure 9.5, excluding the declaration of the system time variables used:

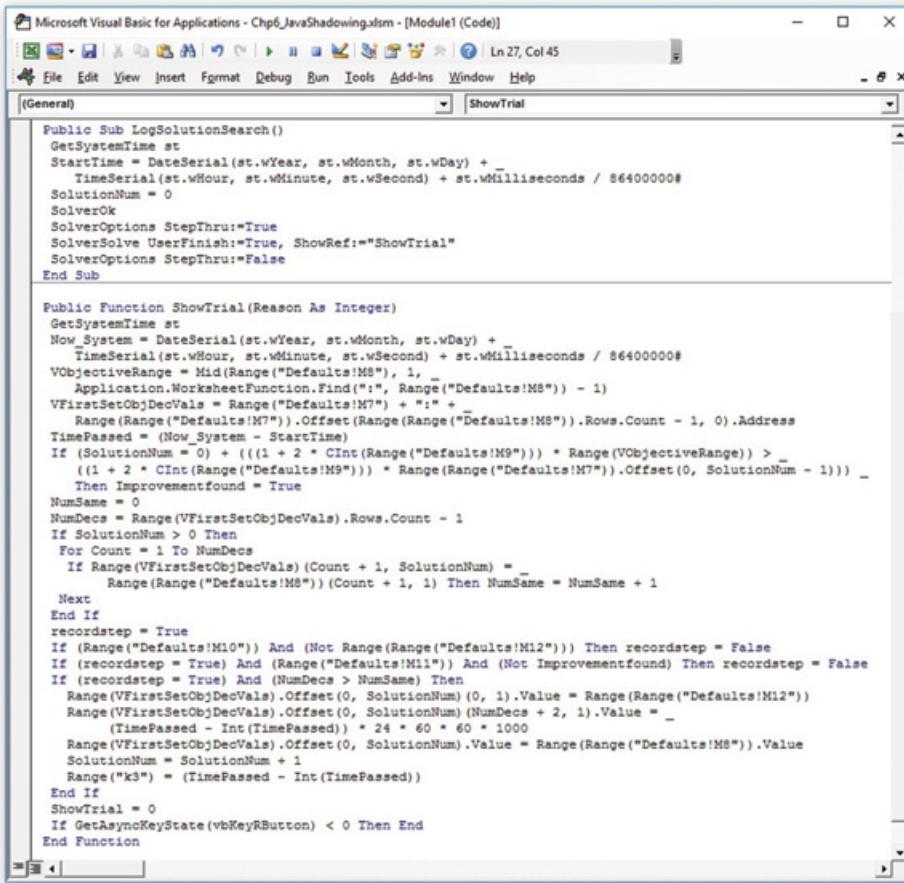
The key modifications to a generic Solver call are as follows:

```
SolverOk
SolverOptions StepThru:=True
SolverSolve UserFinish:=True, ShowRef:="ShowTrial"
SolverOptions StepThru:=False
```

The SolverOptions StepThru:=True or False are lines that turn on and off the incremental pauses between new solution discovery (something you can also interact with manually in the Solver forms). The addition of ShowRef:="ShowTrial" specifies that a function called ShowTrial should be evaluated at each step – a function that in this case is the majority of the remaining code and the means by which to record the details of the utilities, the objective, and constraints over time.

### **9.1.2 Calls to RISK Optimizer**

For problems with more complex structures, or those that involve forms of uncertainty that don't lend themselves to the kind of closed-form analysis with which basic Solver's hill climbing algorithm works well, we've seen that other approaches may be necessary. The use of genetic algorithms, as made available through additional tools such as RISK Optimizer, has already been discussed as an option. Luckily, tools within the @RISK ribbon can also be called from behind the scenes using VBA code, again perhaps as one of many steps involved in a larger analysis around which a decision support tool is designed. Let's consider a couple of the past examples discussed in the book and how we might more seamlessly automate their interface with Excel.



The screenshot shows the Microsoft Visual Basic for Applications (VBA) Editor window. The title bar reads "Microsoft Visual Basic for Applications - Chp6\_JavaShadowing.xlsm - [Module1 (Code)]". The menu bar includes File, Edit, View, Insert, Format, Debug, Run, Tools, Add-Ins, Window, and Help. A toolbar with various icons is visible above the menu. The code editor pane contains the following VBA code:

```

Public Sub LogSolutionSearch()
    GetSystemTime st
    StartTime = DateSerial(st.wYear, st.wMonth, st.wDay) +
        TimeSerial(st.wHour, st.wMinute, st.wSecond) + st.wMilliseconds / 86400000#
    SolutionNum = 0
    SolverOk
    SolverOptions StepThru:=True
    SolverSolve UserFinish:=True, ShowRef:="ShowTrial"
    SolverOptions StepThru:=False
End Sub

Public Function ShowTrial(Reason As Integer)
    GetSystemTime st
    NowSystem = DateSerial(st.wYear, st.wMonth, st.wDay) +
        TimeSerial(st.wHour, st.wMinute, st.wSecond) + st.wMilliseconds / 86400000#
    VObjectiveRange = Mid(Range("Defaults!M8"), 1, _
        Application.WorksheetFunction.Find(":", Range("Defaults!M8")) - 1)
    VFirstSetObjDecVals = Range("Defaults!M7") + ":" +
        Range("Defaults!M7").Offset(Range("Defaults!M8")).Rows.Count - 1, 0).Address
    TimePassed = (NowSystem - StartTime)
    If (SolutionNum = 0) + (((1 + 2 * CInt(Range("Defaults!M9"))) * Range(VObjectiveRange)) >_
        ((1 + 2 * CInt(Range("Defaults!M9")))) * Range(Range("Defaults!M7").Offset(0, SolutionNum - 1))) _ 
        Then Improvementfound = True
    NumSame = 0
    NumDecs = Range(VFirstSetObjDecVals).Rows.Count - 1
    If SolutionNum > 0 Then
        For Count = 1 To NumDecs
            If Range(VFirstSetObjDecVals)(Count + 1, SolutionNum) =
                Range(Range("Defaults!M8"))(Count + 1, 1) Then NumSame = NumSame + 1
        Next
    End If
    recordstep = True
    If (Range("Defaults!M10")) And (Not Range(Range("Defaults!M12"))) Then recordstep = False
    If (recordstep = True) And (Range("Defaults!M11")) And (Not Improvementfound) Then recordstep = False
    If (recordstep = True) And (NumDecs > NumSame) Then
        Range(VFirstSetObjDecVals).Offset(0, SolutionNum)(0, 1).Value = Range(Range("Defaults!M12"))
        Range(VFirstSetObjDecVals).Offset(0, SolutionNum)(NumDecs + 2, 1).Value = _
            (TimePassed - Int(TimePassed)) * 24 * 60 * 60 * 1000
        Range(VFirstSetObjDecVals).Offset(0, SolutionNum).Value = Range(Range("Defaults!M8")).Value
        SolutionNum = SolutionNum + 1
        Range("K3") = (TimePassed - Int(TimePassed))
    End If
    ShowTrial = 0
    If GetAsyncKeyState(vbKeyRButton) < 0 Then End
End Function

```

Figure 9.5 Example code used for shadowing Solver's search

### 9.1.2.1 Work-Group Selection Revisited

To begin with, consider the work-group selection problem last formally discussed in Chapter 7. We showed how RISK Optimizer could be used to derive group constituencies subject to criteria that might be outside the bounds of a black-boxed clustering approach. To call RISK Optimizer to run its routine behind the scenes, we'll want to have RISK Optimizer formally loaded (much in the same way that we might preload Solver, if not already present, before running subroutines that call it). In addition, as with Solver, a formal reference to RISK Optimizer needs to be made through the VBA Editor before the Editor can recognize the code being referenced in any subroutine calling the application. Here, you're looking for something like RiskXLA to be included as a reference, as well as the Palisade Object Library and Developer Kit (Figure 9.6).

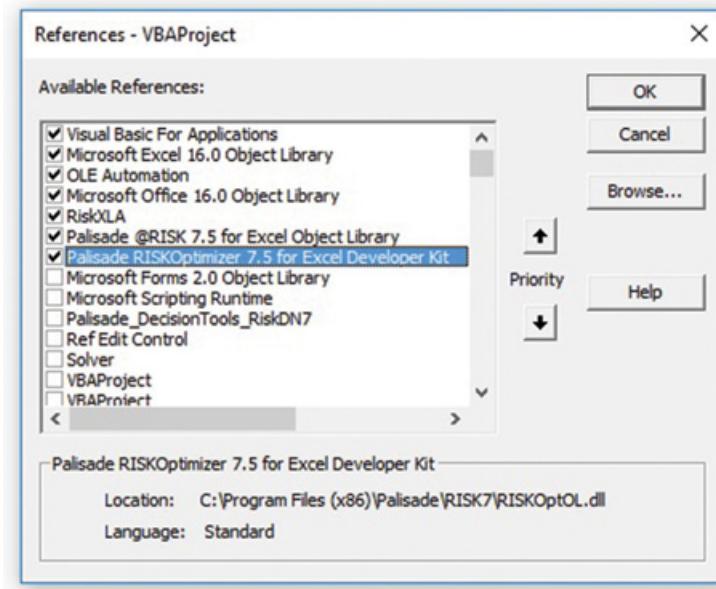


Figure 9.6 Referencing @RISK and RISK Optimizer

Once referenced, we can start to consider the kind of code required to get RISK Optimizer to do its thing. With @RISK installed, the Help button at the right end of the associated ribbon in fact provides access to a wealth of resources in this regard, including an automation guide, a detailed plan of the programmatic options with @RISK, and example workbooks with VBA automation of RISK Optimizer.

Getting back to the present example, let's take a look at how we could make use of these automation options. Recall that the worker assignment task involved placing eighty individuals each into one of four projects. Each individual is characterized by various levels of motivation and ability, levels specific to each of those four projects. We want assignments of workers where the multiple of these two factors is high, but we need to limit group sizes (18–22 people each) and we want to contain variation in the ability of team members ( $\text{spread} \leq 6$ ). Right now the search for good solutions is free to reassign any of the 80 individuals. But what if we saw an assignment or two that we really wanted to keep unchanged? What if we always wanted Elijah, Kaylee, and Dominic to be on project 1?

We could manually add a constraint that forced that assignment, or we could manually change the set of utilities we are allowing RISK Optimizer to modify. Both would require us to manually get into the RISK Optimizer model specification and make these changes. If we discovered other assignments we wanted to mandate or changed our minds on these, this back-and-

forth manual work could get tedious. Instead, what if we were simply able to designate assignments we wanted to keep constant by highlighting or bolding (Ctrl-B, changing the font to bold face) the names of those specific workers? And had a quick way of rerunning the search that guaranteed these rules were in place?

This is exactly the sort of thing that VBA automation of RO can provide, and what has been implemented in the Chp9\_WorkGroupSelection\_wROVBA workbook. The code below presents the automation, which can be activated by Ctrl-g or the green Search button presented in the workbook.

```
Sub AssignWorkers()
    Dim Assignment As Integer
    With RISKOptimizer.ModelWorkbook(ActiveWorkbook)
        With .Goal
            Set .CellToOptimize = Range("TotalAssessment")
            .StatisticToOptimize = OptStatisticValue
            .GoalType = OptGoalMaximize
        End With
        .AdjustableCellGroups.RemoveAll 'just to reset / avoid duplication
        .AdjustableCellGroups.AddWithGroupingSolvingMethod _
            .AdjustableCellRanges.AddForGroupingSolvingMethod _
                Range("WorkerAssignments") 'method and range
        .Constraints.RemoveAll 'just to reset / avoiding duplication
        .Constraints.AddHardInSimpleMode OptEvaluationEachIteration, _
            Range("MinSizes"), OptLessOrEqual, Range("ActualSizes"), _
            OptStatisticValue, 0, OptLessOrEqual, Range("MaxSizes"), _
            "Group Size Restrictions"
        For Count = 1 To Range("WorkerAssignments").Rows.Count
            If Range("NameHeader").Offset(Count, 0).Font.Bold Then
                Assignment = Range("AssignHeader").Offset(Count, 0).Value
                .Constraints.AddHardInSimpleMode OptEvaluationEachIteration, _
                    Assignment, OptLessOrEqual, Range("AssignHeader").Offset
                    (Count, _
                    0), OptStatisticValue, 0, OptLessOrEqual, Assignment, _
                    "Fixed/Constant assignment"
            End If
        Next
        Risk.Simulation.Settings.NumIterations = 1 'ensure no repeat eval
        RISKOptimizer.Optimize 'initiate the search
    End With
    Range("TotalAssessment").Select
End Sub
```

The following additional code is made available separately, to allow solutions to be reset to a trivial starting solution (first twenty 1s, next twenty 2s, etc.), with the exception again of those individuals bolded.

```

Sub ResetMost()
    For Count = 1 To Range("WorkerAssignments").Rows.Count
        If Not Range("NameHeader").Offset(Count, 0).Font.Bold Then
            Range("AssignHeader").Offset(Count, 0).Value = _
                1 + Int((Count - 0.001) / 20)
        End If
    Next
End Sub

```

Now, in the automation of RO, there's nothing stopping a developer from further specifying whether the best solution found should overwrite the original, based on inputs provided by their users in a customized interface, or all solutions encountered should be saved in the logbook (even poor ones or those that might fundamentally violate assumed constraints). Here, a few references to designated cells in a workbook interface (or a Userform along with the use of IF-THEN structures within the function) would do the trick.

As with Solver automation, developers also have the option of modifying the approach RO takes in its search; including search-stopping criteria and the dynamics of simulation, if relevant. We'll consider some of these in the next example.

#### *9.1.2.2 Inventory System Simulation Revisited*

Because RISK Optimizer is clearly useful in simulation optimization settings, it is worth reviewing how the code for building and executing an example of such optimization might be structured. For illustration, let's again draw on an example we've seen before: the Lobos inventory optimization example from Chapter 7. What would the code look like for fully automating that search? Perhaps adding some direct control over the number of periods of evaluation (e.g., rather than always having 200) and more immediate access to the bounds of the search and the number of repeated evaluations of each policy? Perhaps also adding automatic generation of a simple comparison plot of solutions encountered based on RO logs? A copy of the workbook containing such code is provided in Chp9\_LobosInventory\_wROVBA.

Let's start by examining a workbook setup subroutine that this book uses, guaranteeing that the correct iteration mode is being used, as well as looking at a redesign of the subroutine that evaluates the system over multiple periods (as recorded originally in Chapter 7). Note in particular the use of With statements and Loop structures to streamline syntax. In place of the constant "200" we now have reference to a spreadsheet cell that contains the number of periods to be evaluated (Range ("NumPeriods")).

```

Sub SetupWorkbookCalcs()
    Application.ScreenUpdating = False
    Range("Messageout") = ""
    With Application
        .Calculation = xlManual 'xlAutomatic
        .Iteration = True
        .MaxIterations = 1
    End With
End Sub
Sub MyFirstMacro() 'greatly shortened
    Range("Restart").Value = "FALSE"
    Calculate
    Range("Restart").Value = "TRUE"
    For Count = 1 To Range("NumPeriods")
        Calculate
    Next
    Range("Summaries").Copy
    Range("RecordsOut").Offset(1, 0).PasteSpecial _
        Paste:=xlPasteValues, Operation:=xlNone, _
        SkipBlanks:=False, Transpose:=True
    Range("RecordsOut").Offset(1, 0).EntireRow.Insert
End Sub

```

Now, let's look at the RISK Optimizer automation:

```

Sub runSearch()
    Dim MessageBoxtext As String
    Dim rptPlace As RiskReportPlacement
    SetupWorkbookCalcs
    With RISKOptimizer.ModelWorkbook(ActiveWorkbook)
        With .Goal
            Set .CellToOptimize = Range("NetImpliedCost ")
            .StatisticToOptimize = OptStatisticMean
            .GoalType = OptGoalMinimize
        End With
        .AdjustableCellGroups.RemoveAll 'just to reset/avoid duplication
        .AdjustableCellGroups.AddWithRecipeSolvingMethod _
            .AdjustableCellRanges.AddForRecipeSolvingMethod _
                Range("ReOrderPt"), Range("LowerLimit").Value, _
                Range("UpperLimit").Value, True
        .Constraints.RemoveAll 'just to reset
        With Risk.Simulation.Settings
            .NumIterations = Range("NumRepeats")
            .AutomaticResultsDisplay = RiskNoAutomaticResults
        End With
        RISKOptimizer.Optimize
        With .OptimizationResults

```

```

MessageBoxtext = "Details of Search:" & vbCrLf & vbCrLf _
    & "Search Time: " & Format((.FinishTime - .StartTime) * 24 _
    * 60 * 60, "#,#00") & " seconds" & vbCrLf & _
    "Recommended Reorder Point Level: " & _
    Range("ReOrderPt").Value & vbCrLf & vbCrLf & _
    "Log of search saved."
End With
Range("MessageOut") = MessageBoxtext
Risk.ApplicationSettings.ReportPlacement = RiskActiveWorkbook
With .OptimizationResults
    ' .GenerateOptimizationSummary True 'if Summary desired
    .GenerateLog False, True 'True, True for Improvements Log
End With
Risk.ApplicationSettings.ReportPlacement = _
    Risk.ApplicationSettings.ReportPlacement
End With
Sheets("StockoutDemo").Select
Range("ReOrderPt").Select
addChart
End Sub

```

The specification of the objective (goal), utilities (adjustable cells), and constraints proceeds much in the same way as in the prior example, apart from the fact that the Recipe method is being used here rather than Group, and that the range of reorder point levels to evaluate is specified by the user (Range("UpperLimit"), etc.). Further specific to this search, we need to make sure that RO will be comparing “means” rather than “values” (since we have multiple evaluations per policy scenario). We also have to designate the total number of iterations per scenario, here again drawing on specifications made by the user directly in the spreadsheet environment (.NumIterations = Range("NumRepeats")).

However, that's just half of the above code. The code below the RISKOptimizer.Optimize command is all about outputs. It generates a message specific to the details of the best search result, for writing to a designated cell (Range("MessageOut")). It also asks for the complete log of the search to be added to the workbook in a separate sheet – all automated and without popup dialog generated. With that information available, a final subroutine (addChart) is called to generate a simple graphic of the search and recommendation. The code involved is similar to that used in Chp8\_QuarterlyData\_Macros and our associated discussion. Considerable use is made of the With statement here to streamline and reduce the risk of errors in syntax.

```
Sub addChart()
    Dim numrows As Integer
    Application.ScreenUpdating = True
    Application.Calculation = xlAutomatic
    numrows = Application.CountA(Range(" 'Optimization Log'!F:F"))
    ActiveSheet.Shapes.AddChart2(240, xlXYScatter).Select
    With ActiveChart
        .SeriesCollection.NewSeries
        .FullSeriesCollection(1).Name = "Results"
        .FullSeriesCollection(1).XValues =
            ="Optimization Log"!$J$8:$J$" + CStr(numrows + 5)
        .FullSeriesCollection(1).Values =
            ="Optimization Log"!$F$8:$F$" + CStr(numrows + 5)
        .SeriesCollection.NewSeries
        .FullSeriesCollection(2).XValues = "=StockoutDemo!$C$13"
        .FullSeriesCollection(2).Values = "=StockoutDemo!$C$14"
        .ChartTitle.Text = "Search Results"
        With .FullSeriesCollection(2) ' selection
            .MarkerStyle = 8
            .MarkerSize = 13
            With .Format.Line
                .Visible = msoTrue
                .ForeColor.RGB = RGB(0, 176, 240)
                .Weight = 3
            End With
            With .Format.Fill
                .Visible = msoTrue
                .ForeColor.ObjectThemeColor = msoThemeColorText2
                .ForeColor.TintAndShade = 0.1
                .Transparency = 0
                .Solid
            End With
            .ApplyDataLabels
            .HasLeaderLines = True
            .DataLabels.Format.TextFrame2.TextRange. _
                InsertChartField msoChartFieldRange, _
                "=StockoutDemo!$B$37", 0
            .DataLabels.Select
            Selection>ShowRange = True
            Selection>ShowValue = False
            .Points(1).DataLabel.Select
            Selection.Left = 175
            Selection.Top = 120
        End With
        .Axes(xlValue).MajorGridlines.Delete
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Caption =
    End Sub
```

```

    "=StockoutDemo!R13C2"
.Axes(xlValue, xlPrimary).HasTitle = True
.Axes(xlValue, xlPrimary).AxisTitle.Caption = _
    "=StockoutDemo!R36C2"
.Parent.Left = Range("h14").Left
.Parent.Top = Range("h14").Top
End With
Range("ReOrderPt").Select
End Sub

```

The result of all of this in action, again provided @RISK is active and the appropriate references from the last section have been made, is shown in the text and graphical output of the Chp9\_LobosInventory\_wROVBA worksheet front end (Figure 9.7).

Since we now have a tantalizing glimpse into the many things we can accomplish with RISK Optimizer and @RISK automation, it is worth stating a general caveat. Application syntax does change. The syntax that was used ten years ago to deliver this automation is not the same as it is today. It is possible that the syntax relevant today will become obsolete as upgrades to the Palisade suite develop further down the road as well. Fortunately the general object structure remains fairly constant, as does the logic. As long as

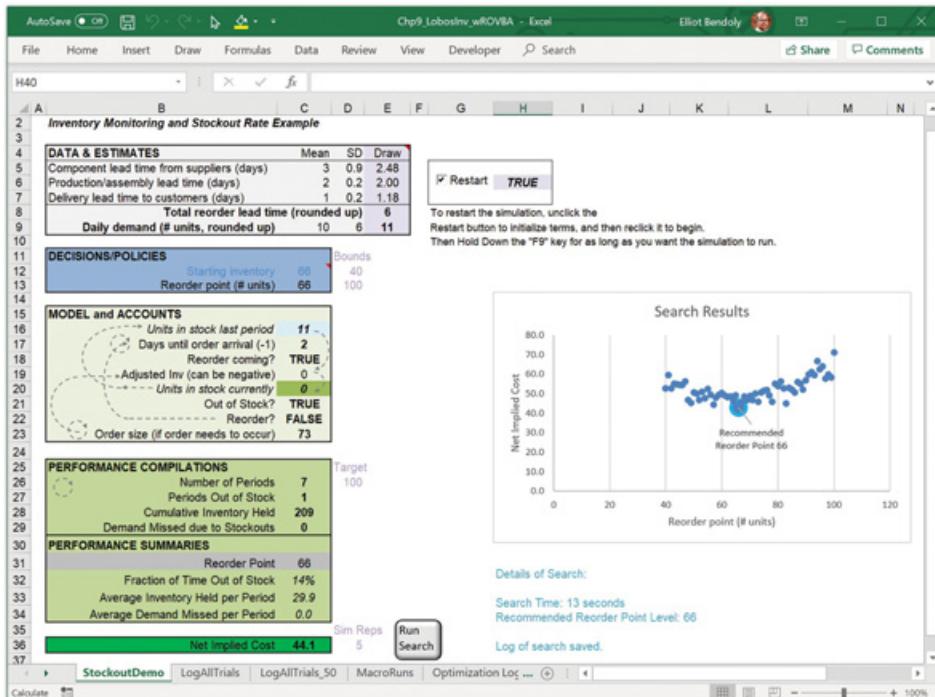


Figure 9.7 Workbook interface and visual output from RISK Optimizer automation

you can find working examples of modern RO automation syntax, per Palisade's most recent example files and documentation, you should have no issues upgrading your own applications if changes occur.

### 9.1.3 Communicating with Google Sheets

There's another spreadsheet environment that we haven't spent much time on, but which is nevertheless incredibly convenient for certain tasks. While I don't personally go to Google Docs to do much analytical work, I do rely on its ubiquity to help with sharing content and assuring that I have access to certain content regardless of where I am and what platform I happen to be on. The Blackbelt site is hosted on Google, as are the chapter example files. The Heat Mapper tool references a compendium of all shape sets from a Google Sheet, which in turn is used to generate its selection lists. And generally speaking, the data in any publically viewable Google Sheet can be queried much in the same way that data (e.g., stock prices) can be pulled from other well-structured pages.

All of this highlights the ease with which content can be viewed when posted to Google applications, allowing us to retrieve content for use with Excel, for example. But what about actually posting content *from* an Excel workbook? There are multiple options to consider here. Figure 9.8 outlines one perspective of the various ways in which Google Sheets might be communicated with via other applications. You'll notice some convenient parallels and repetition (intentional) in the various methods alluded to by this outline.

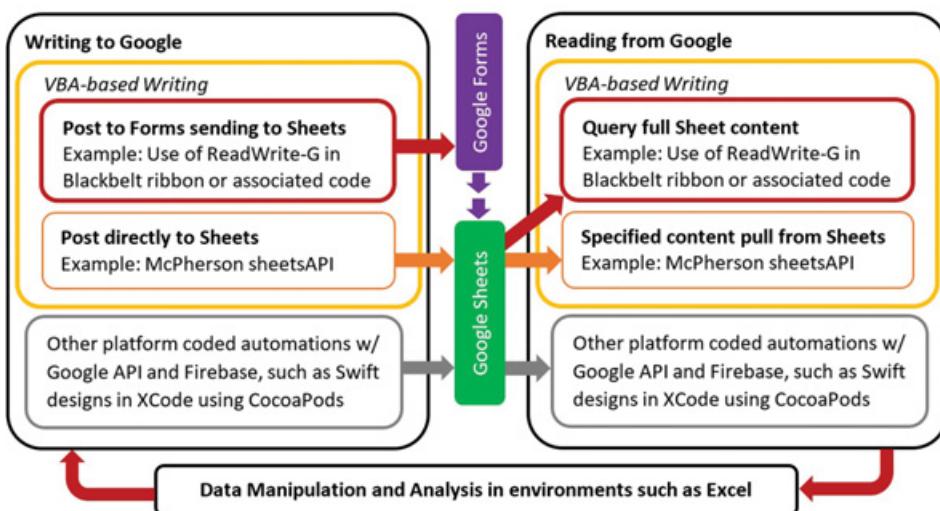


Figure 9.8 Communicating information and analysis across applications and platform

To keep things moderately simple, let's just focus on the task of using VBA to write to something that's meant for data entry: a Google Form. You've probably encountered a Google Form in the past, perhaps as a respondent to a survey. Conveniently, data entered into a Google Form is sent for storage as subsequent entries in Google Sheets, making cursory data examination by collectors extremely straightforward. Just as individuals can manually enter responses into a Google Form, however, we can programmatically enter content into such forms via VBA.

For demonstration, in Chrome I've created a Google Form and its associated storage Sheet. I've chosen to include just three fields in my Form, and therefore the associated Sheet houses four field columns, including the default time stamp in the first column (see Figure 9.9).

I've made both the Sheet and the Form "public" so I'll be able to both read from the Sheet and as well as write to the Form (and hence the Sheet in turn). The blue "Share" button on the Google Sheet provides access to a public

	A	B	C	D	E
1	Timestamp	Round Number	Personal ID Code	Quality-Level Investment	
2					
3					
4					
5					
6					
7					
8					

Figure 9.9 Example Google Sheet fed into by a Google Form (blinded URLs)

link. Although I'm blinding most of the identifying content here, the link should begin with something like "<https://docs.google.com/spreadsheets/d/>" and end with something like "/edit?usp=sharing." The white "Send" button on the Google Form likewise provides its URL, beginning with "<https://docs.google.com/forms/d/e/>" and ending with something like "/viewform?usp=s-f\_link." The long alphanumeric strings between the last and second last forward slashes, strings I'm not showing here (though you can get a sense of them from Figure 9.9), are the identifiers for these two objects. We'll be using these keys in the VBA code we will construct for reading and writing in these environments. So if you are following along with your own Form and Sheet, it's worth copying these down at this point.

We are also going to need to figure out how input specifications in the fields of the Google Form translate into data that appears on the related Sheet. To do this, we will leverage a special feature in Chrome in particular. We'll start by opening the Form as if we were going to provide a response (as a manual user), entering the public URL for the Form we just copied and pasting it into the URL field of our Chrome browser. Once the Form is loaded, we're going to pause for some inspection of the Form. Although we will soon be submitting test content into the Form, before we submit anything, we are going to right-click on that page and select the "Inspect" option that appears. This will give us access to nascent traffic records for that Form. The associated view that we now have access to will look a bit like Figure 9.10 (again with specific section content blinded).

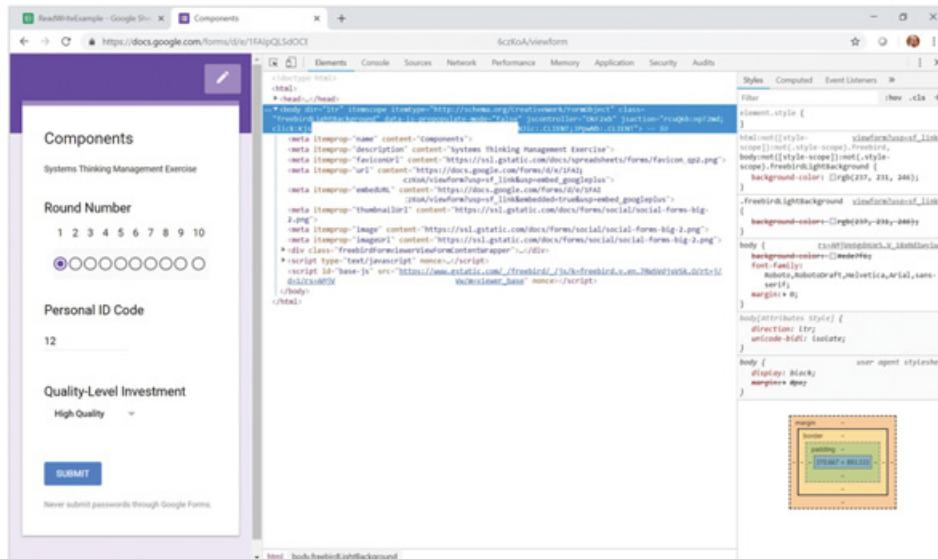


Figure 9.10 Inspection view for a sample Google Form (blinded details)

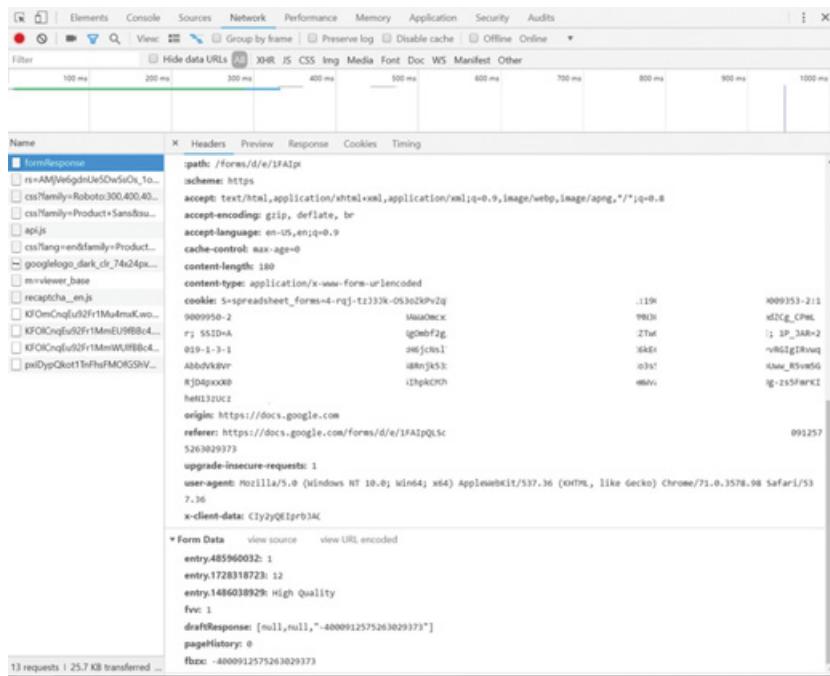


Figure 9.11 Network form formResponse view of Form content and entry code

From here we can select Network among the tabs at the top of this log. Hitting Submit will send any entered Form data to the Google Sheet but will also create a new entry, “formResponse,” at the top of the network listings. Clicking on formResponse will open further details, as shown with the Headers tab selected (as in Figure 9.11).

Finally, at the bottom of this listing, you will find a section called Form Data, in which the entries just submitted will be listed. Selecting “view source” allows you to see the fully encoded entry string, which we will use in communicating content to the Form (and Sheet). You get something like this:

```
entry.485960032=1&entry.1728318723=12&entry.1486038929=High
+Quality&fvv=1&draftResponse=%Bnull%2Cnull%2C%22-
4000912575263029373%22%5D%0D%0A&pageHistory=0&fbzx=-
4000912575263029373
```

Those lengthy numbers following each instance of “entry” are unique keys to the form fields. Just like the Sheet and Form keys, they will differ across documents but should be stable on reuse. We will use these in our code as well, which itself borrows from syntax we’ve seen before in automating Web Query modifications (Chapter 8). In this case we will use a much more

temporary form of query since we don't really want to pull data, but rather to communicate a submission of data. Fortunately for us, the same kind of actions that communicate requests for data from a source can involve instructions such as the posting of data to that source. The code used by the Read Write-G component of the Blackbelt Ribbon uses this approach. Below is a simplified version of that code:

```
Sub WriteViaForm() 'Simple routine for inputting to Google Form
    Dim SourceURL As String
    Dim Inputs As String
    SourceURL = _
        "https://docs.google.com/forms/d/e/YOURFORMKEY/formResponse"
        ' Above viewable as URL of the FormResponse
    FieldInput1 = Range("C3") 'Assuming source cell
    FieldInput2 = Range("D3") 'Assuming source cell
    FieldInput3 = Range("E3") 'Assuming source cell
    Inputs = "entry.YOURFIRSTFIELDKEY=" & FieldInput1 & _
        "&entry.YOURSECONDFIELDKEY=" & FieldInput2 & _
        "&entry.YOURTHIRDFIELDKEY =" & FieldInput3 & "&fvv=1"
        ' Above are elements of the view Form Data encoding
        ' Note: "+" for spaces, non-number/non-letters must be encoded
    SourceURL = SourceURL & "?" & Inputs 'First add inputs to the URL
    With ActiveSheet.QueryTables.add(Connection:= "URL;" & _
        SourceURL, Destination:=Range("A1"))
        .BackgroundQuery = False
        .PreserveFormatting = PlainText
        .RefreshStyle = xlOverwriteCells
        .Refresh
        .Delete
    End With
End Sub
```

A functional three-field form writing subroutine will yield results that are visible in the linked Google Sheet (as in Figure 9.12). The neat thing is that the very same Form can be shared with users for manual entry, permitting both automated and manual interaction with the Form and data entry in the Sheet essentially at the same time. Further, if the Sheet itself is shared and the URL provided, both automated and manual Form submissions can be viewed by any audience of any size, with any associated analytics that you might want to construct in the Google Sheet, or queried by such an audience possessing an Excel workbook designed for numerical, text, or visual analysis. The best of all worlds can be connected.

Now, if you are interested in replacing the entire content of a Google Sheet, or in entering data at a more complex tabular level, or in editing in a way that isn't limited to Form row entry, there are other approaches that

The screenshot shows a Google Sheets interface with the following data in the 'Timestamp' sheet:

	A	B	C	D	E
1	Timestamp	Round Number	Personal ID Code	Quality-Level Investment	
2	1/3/2019 19:34:45	2	12	Low Quality	
3	1/3/2019 19:21:00	1	12	High Quality	
4					
5					
6					
7					
8					

Below the sheet tabs, it says 'Form Responses 1'.

Figure 9.12 Final Sheet view of content written from VBA via to Google Form

you may want to consider. Rather than devoting time to these exercises, which are programmatically more involved albeit far more versatile than Form writing, I'm going to recommend the advice of some experts that work with the Google application program interface (API) on a regular basis. Some of these are listed in the left-side bar of the Blackbelt site, but in particular I would recommend looking into the sheetsAPI examples and related work by Bruce McPherson. Bruce has even written a book on VBA and Google app script.

## 9.2 Guided Design and Protection in Workbooks

Interfacing between applications is one thing. Interfacing with users or, more appropriately, designing interfaces that others can intuitively navigate, gain value from, and hence be interested in returning to – well, that's something else. Being highly skilled in technical integration isn't the same thing as appreciating the cognitive nuances of potential users. And there are many out there who have convinced themselves that they sufficiently understand the variations of human perception, but in reality understand neither people nor systems. Precious few have studied human behavior as well as complex analytical systems, both in the field and in the lab, at the enterprise level and at the level of individual learning.

Fortunately, you don't have to publish 50-plus articles or have an h-index of 30 to be familiar with some fundamental best practices when designing your own tools. There are many principles of effective interface design that prove highly relevant and simple to implement when assembling tools for others; these same principles benefit us in the design of tools we build for

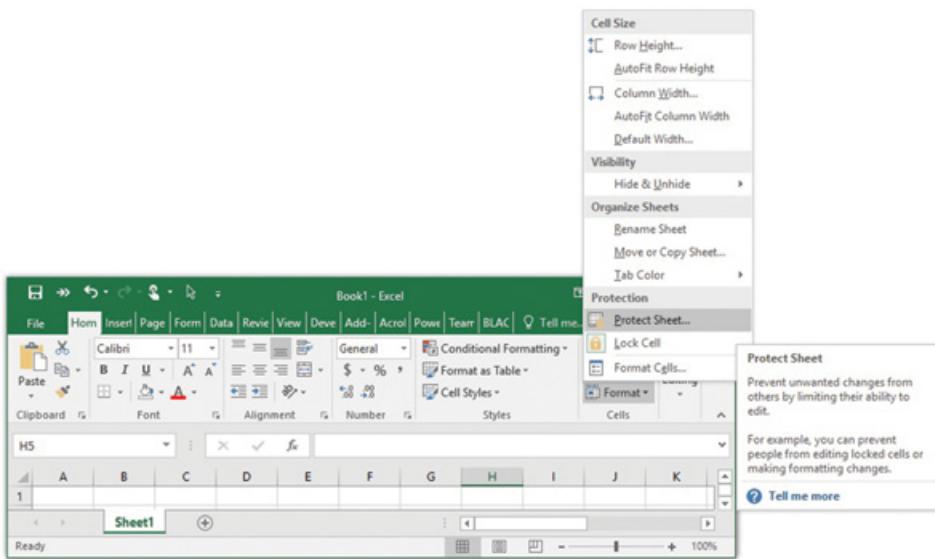


Figure 9.13 Access to interface locking and protecting mechanisms

ourselves. Bendoly and Clark (2017) discuss some of these in Sections 2.1 and 2.2 of their text (“Gestalt Laws of Grouping” and “The Virtue of Good Fences”). Explicit guidance, reinforced focus, and multivariate/multiperspective messaging, and simply putting rails on the way in which individuals might otherwise deviate from effective use are all things that can be built into proof-of-concept designs, even in the workbook environment.

We can start with the simplest of example tactics here to appreciate what’s available to us in this setting before we get a bit more sophisticated. For example, under the Home tab of the main Excel menu you’ll find the general Format drop-down menu (Figure 9.13) that includes various options that are meaningful to security measures you might want to capitalize on in your DSS design.

One of the elements subject to consideration when adding protection to workbooks is which cells, if any, should be locked. Locking cells can have multiple implications when the spreadsheets and workbooks within which they reside are protected. One consequence is the inability to make modifications to cells that are locked, either manually or through VBA, without first unprotecting (or unlocking) through the code. At the same time, calculations and functions present within locked cells can still be updated automatically when sheets are protected, as can graphs based on these calculations. Cells can be toggled between locked and unlocked states through either right-clicking and modifying their properties, or through the Format drop-down menu shown in Figure 9.13. The same drop-down menu

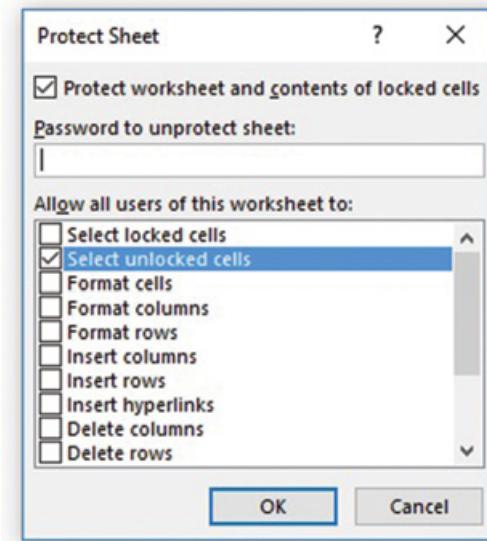


Figure 9.14 Customized specifications for sheet protection

allows for the protection of a spreadsheet. The Protect Sheet dialog box generated for detailing sheet protection is shown in Figure 9.14.

Note the variety of options that are contained in this dialog box. In general, the option to protect (i.e., prevent changes) to locked cells should seem a natural one; however, developers are also given the option to assign a password to the protection mechanism so that the sheet cannot be unprotected by those who don't know the password. Developers are also given a slate of check boxes to customize the nature of the protection. These are basically aimed at lessening the severity of the protection, such as allowing unlocked cells to be selected, or allowing objects to be edited while blocking most other activities.

Let's take a look at an example in which such locking might prove helpful in directing usage. In the workbook Chp9\_TickerPFunction\_pulldown we have a version of the stock data pull example from Chapter 8. The big difference is the presence of an ActiveX control permitting the selection of a stock ticker from a list. Upon selection, a query is triggered and the stock price presented. There are a few elements in the Summary sheet of this workbook, including the cell the Combo Box is linked to, the list that feeds into the Combo Box (in light gray), and the price of the stock. But there's no need for a user to try to edit the price directly, and there may be no desire to let the user modify the base list of tickers. In fact we could keep all cells locked apart from the Combo Box's linked cell (positioned behind that object) and protect things until they need to be modified (e.g., a new Price

presented). And this is precisely what this example and its associated code does.

The action of locking cells and protecting worksheets can in fact be handled programmatically, as available in this example as well. Since locking is simply an action that modifies a cell, we should be able to access and modify that attribute just like any other. The following code would do the trick for locking cell A2:

```
Range("A2").Locked = True
```

Unlocking a cell, if a sheet is unprotected, simply involves swapping True for False. Unprotecting a sheet is fairly straightforward as well (ActiveSheet.Unprotect), though the various options associated with sheet protection make its code a bit more complicated (and versatile):

```
ActiveSheet.Protect DrawingObjects:=False, _
    Contents:=True, Scenarios:=False
```

As another example and follow-up to our previous discussion of the benefits and caveats to the DoEvents syntax in VBA, consider how one might similarly protect the Chp8\_BasicClock example, while still permitting exits from the countdown.

Another mechanism by which to indirectly protect sheets of your DSS from tampering is simply hiding them, as shown in Figure 9.15.

The hide and unhide options for rows, columns, and entire sheets are found in the Format drop-down menu. After a sheet is protected, all rows

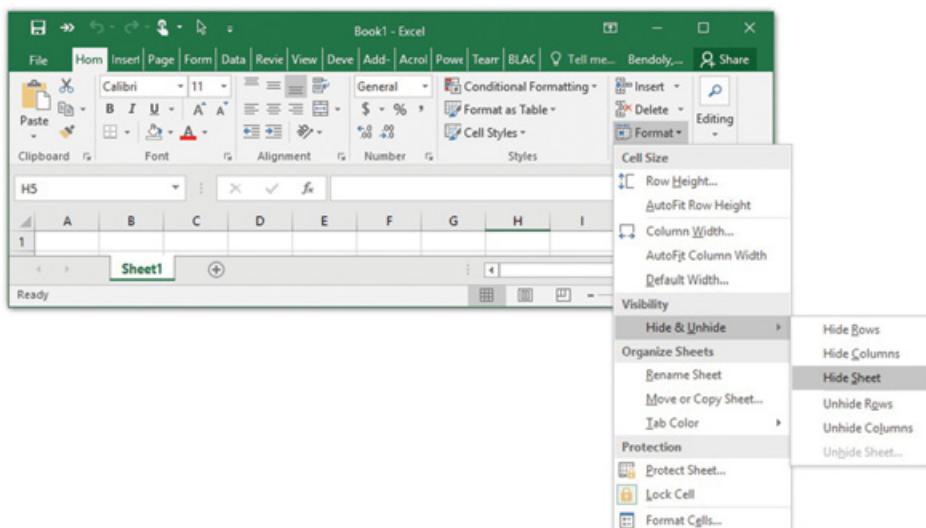


Figure 9.15 Hiding and unhiding interface

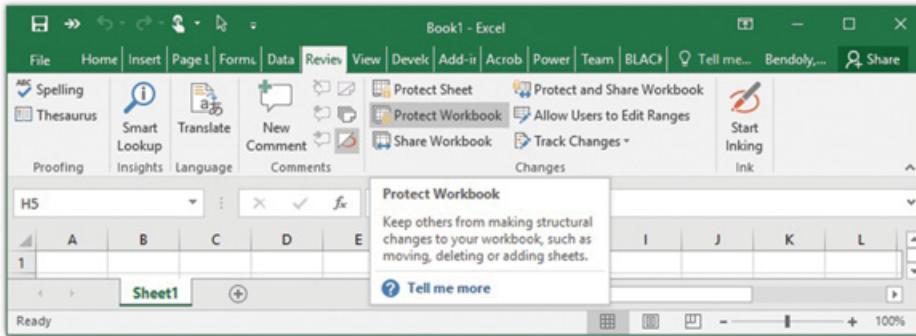


Figure 9.16 Workbook protection options

or columns that had been hidden before (but nevertheless might hold vital data and calculations) cannot be unhidden or expanded. It's a way to quickly hide spreadsheet structure without going overboard on interface refinement activities.

If developers are seriously interested in ensuring that hidden sheets remain hidden, an additional level of protection needs to be applied. This can be found under Review>Protect Workbook, as seen in Figure 9.16. Protecting the structure of the workbook with or without a password prevents hidden sheets from being manually unhidden. Just keep in mind, password protection can be cracked, if someone is interested enough to do so. So be cautious of what you share. Further, the fact is that most casual users of Excel won't actually bother to try (or know how) to unprotect books and sheets that *lack* passwords. Thus protections sans passwords can still provide useful rails for, at least, proof-of-concept builds.

### 9.3 GUIs beyond the Worksheet: Userforms

The ability to call up forms and controls on an as-needed basis can also be highly effective in the structuring of a DSS interface. Such capability ensures that these controls don't get in the way of more critical elements of your interface when they are not needed. Also, the ability to withdraw access to these controls can be critical in foolproofing your DSS against inadvertent changes that these controls could generate. To start the development of a Userform, enter the VBA Editor and select Insert>UserForm. This will generate a new (empty) Userform. The form toolbox may also appear; if it doesn't, select View>Toolbox from the main header (see Figure 9.17). You'll use this toolbox for a variety of tasks.

The tools in the toolbox should look familiar. We've seen them in the ActiveX object options of Chapter 4. Additional tools are also available here

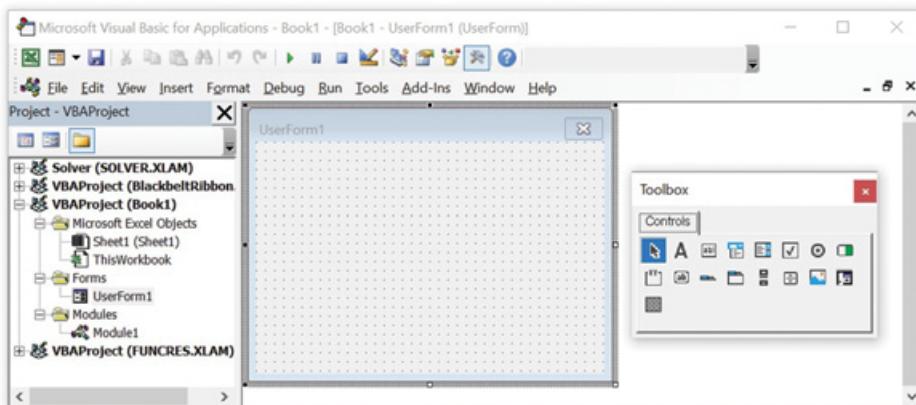


Figure 9.17 Userforms and Control options toolbox

that are often less functional on the spreadsheet front end. VBA-controlled pop-ups, for example, can make Web Browser presentations available, beyond just video access (see the example in Chapter 8). A couple examples of how we might do this are made available in Chp9\_PopupPagesImages, if you're interested.

### ***9.3.1 Constructing and Linking a Userform: An Example***

Let's walk through the steps involved in developing a fully integrated Userform for a specific analytical task. In Chp9\_FunctionPopup, I provide an example where I've designed a Userform using a number of basic objects that we've seen in the ActiveX realm. Figure 9.18 outlines and summarizes how most of the various controls were dragged and dropped onto a newly generated Userform, as well as how incidental changes such as name-property changes were made to fit the existing terminology of the QueueEstimator function developed in Chapter 8.

With the Userform form showing in the VBA Editor, double-click on any form component of choice to get access to specifying code for that particular item. Specifically, double-clicking on the Calc button would create a new private subroutine for you to edit to your needs (its privacy refers to its designed use specifically in reference to the Userform interface).

```
Private Sub Calc_button_Click()
End Sub
```

Thankfully, these subroutines are coded the same as anything else in VBA. The following is what I've developed as a subroutine for the Calc button (the functioning code is available in Chp9\_FunctionPopup):

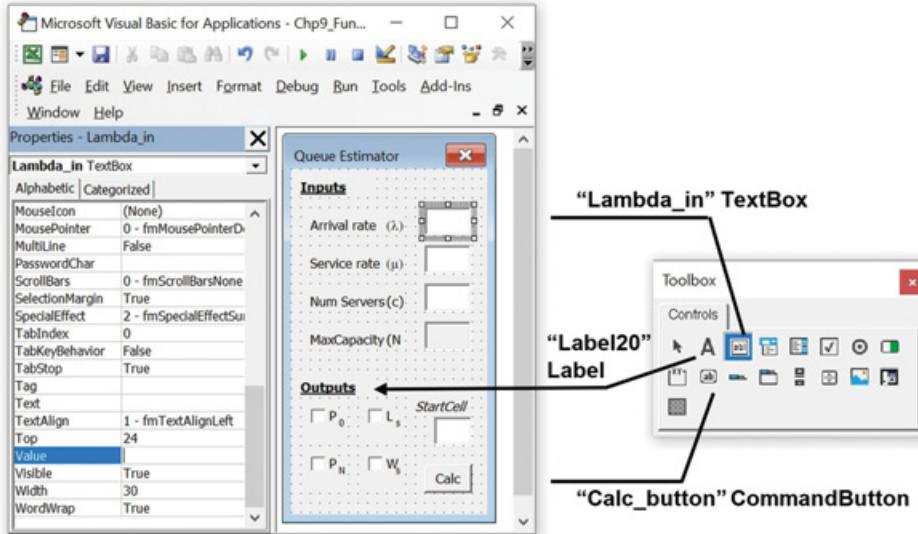


Figure 9.18 In-form control applications and labels used for VBA reference

```

Private Sub Calc_button_Click()
    Dim rho, Pnot_out, Pn_out, Ls_out, Ws_out As Double
    Dim Cellposition As String
    If Lambda_in.Value <> 0 And Mu_in.Value <> 0 Then
        rho = Lambda_in.Value / Mu_in.Value
        If rho / c_in.Value < 1 Then
            Pnot_out = P0(c_in.Value, rho, N_in.Value)
            Pn_out = Pn(c_in.Value, rho, N_in.Value, N_in.Value, Pnot_out)
            Ls_out = Ls(c_in.Value, rho, N_in.Value, Pnot_out, Pn_out)
            Ws_out = Ws(Lambda_in.Value, Mu_in.Value, rho, Pn_out, Ls_out)
            P0_val.Caption = WorksheetFunction.Round(Pnot_out, 3)
            PN_val.Caption = WorksheetFunction.Round(Pn_out, 3)
            Ls_val.Caption = WorksheetFunction.Round(Ls_out, 3)
            Ws_val.Caption = WorksheetFunction.Round(Ws_out, 3)
        Else
            P0_val.Caption = "NA"
            PN_val.Caption = "NA"
            Ls_val.Caption = "NA"
            Ws_val.Caption = "NA"
        End If
        Cellposition = Start_cell.Value
        If Cellposition <> "" Then
            If Give_P0.Value Then
                Range(Cellposition) = Pnot_out
                Cellposition = Range(Cellposition).Offset(1, 0).Address
            End If
        End If
    End If
End Sub

```

```

If Give_PN.Value Then
    Range(Cellposition) = Pn_out
    Cellposition = Range(Cellposition).Offset(1, 0).Address
End If
If Give_Ls.Value Then
    Range(Cellposition) = Ls_out
    Cellposition = Range(Cellposition).Offset(1, 0).Address
End If
If Give_Ws.Value Then
    Range(Cellposition) = Ws_out
End If
End If
End Sub

```

Close inspection reveals that I've intended to use this single button to activate the full array of calculations specified in the original QueueEstimator (from Chapter 8). However, in this case, the tool outputs only the calculations requested by the user, and only specifically where the user has indicated with the StartCell entry field.

To activate this form (i.e., to actually get it to display on request from someone working through the Excel interface), I'll need something to call it into existence. Specifically, I'll need another subroutine defined under Modules, structured as follows:

```

Sub Call_Estimator()
    Queue_Estimator.Show VBAModeless
End Sub

```

If I assign that subroutine to a simple image I've created or imported (to serve as an activation button), the clean version of the interactive Userform will appear. Entering the appropriate parameters into the text fields and clicking Calc should give me exactly the kind of output I designed for. And it does. I'll get some summaries of results built into this version of the Userform and their values written to the spreadsheet depending on what outputs I asked to be copied.

One note on syntax: The VBAModeless term used in the Userform call is an addition that allows a little more mobility by the user while the Userform is showing. For example, it allows the user to view other sheets and interact to some extent through the main Excel toolbars. Another interesting feature of VBAModeless is that I can move the entire Excel interface out of site while still interacting with the interface. Figure 9.19 shows a clip of the Userform on my computer desktop. To this extent, Userforms can serve as handy and compact specialty calculators when you may want to deal with numerous applications at the same time (aside from Excel).

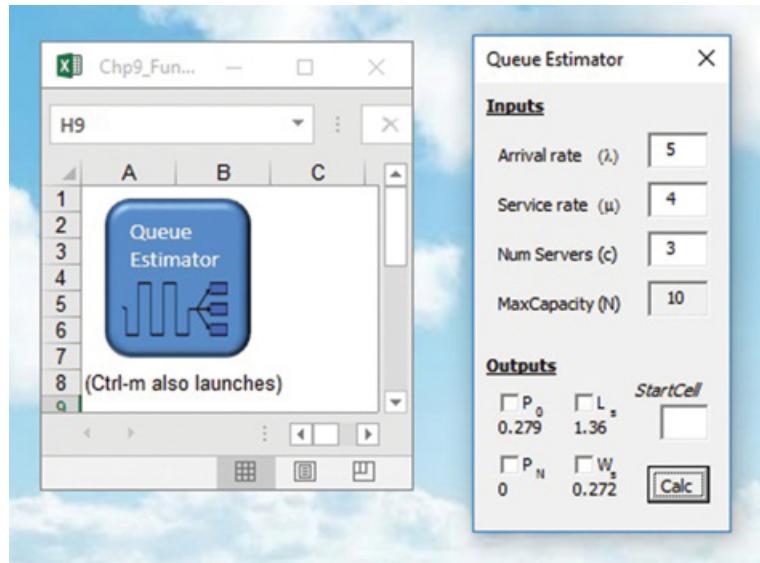


Figure 9.19 Userform and its functionality called from worksheet

### 9.3.2 More Sophisticated Userform Options

Aside from the standard TextBoxes, CheckBoxes, and fixed structure ListBoxes that we've encountered here and in our discussion of sheet-embedded controls, in the realm of Userforms we have many more options, particularly when paired with the setup capabilities of VBA. For example, ListBoxes in a Userform setting can leverage additional properties allowing multiple items in the ListBox to be meaningfully selected. This is the “MultiSelectMulti” setting of ListBoxes in Userforms and can be accessed under the list form’s properties in the VBA Developer.

An image of this specification is presented in Figure 9.20, alongside the appearance and functionality of the object in use. During the design process, once this property specification is selected, you might not see any immediate changes in the appearance of the ListBox under development (if you haven’t specified the actual list content yet). As in the case of embedded ListBoxes we’ve seen in the Chapter 4 Supplement, as well as in the use of the data validation tool, the lists that fed these controls have to come from somewhere – a range of cells or a specification through VBA. The same options exist here by way of the RowSource property.

Through VBA the specification of whether items are checked on or not in the multiselect scenario, as well as what the items in the list in fact are, can be easily manipulated programmatically as needed. This is what in fact occurs in the Heat Mapper tool of the Blackbelt Ribbon add-in, where the contents of

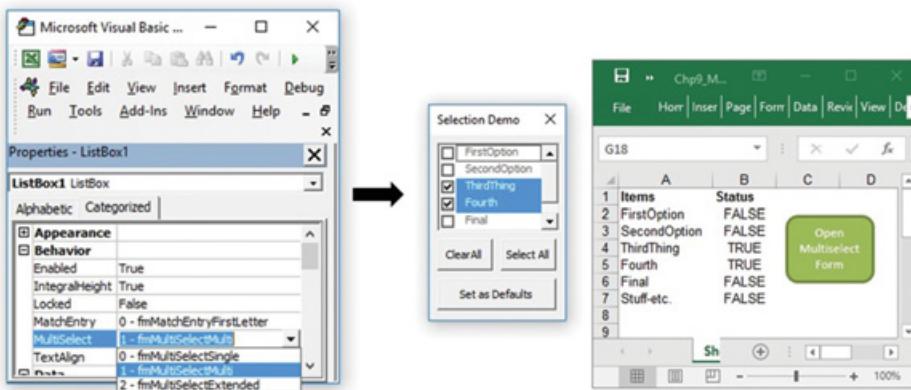


Figure 9.20 The MultiSelect option of Userform listboxes

both a single-select and a multiselect ListBox are populated in VBA based on earlier selections. In this particular case we begin by considering what would make sense as an initial presentation of the form to a user (i.e., how we might determine the contents of the list and whether those contents should be shown as selected or not by default). The following code in the Userform's private subroutines (accessed by double-clicking on the form) demonstrates how a Do While loop might be used to determine list length, assuming the existence of the list in a specified workbook location, for the construction of a string input to the RowSource property. A For Next loop is then used, based on the length of that defined list, to draw in data regarding default selections (again assuming specific locations of that data in the workbook).

```
Dim count As Integer

Private Sub UserForm_Activate() 'what happens when form is activated
    count = 1
    Do While Range("ItemsHeader").Offset(count, 0) <> ""
        count = count + 1
    Loop
    ListBox1.RowSource = Range("ItemsHeader").Offset(1, 0).Address +
        ":" + Range("ItemsHeader").Offset(count - 1, 0).Address
    For count = 0 To (ListBox1.ListCount - 1)
        'Note: if 5 items, they would be indexed 0,1,2,3,4
        ListBox1.Selected(count) = Range("StatusHeader").Offset(count + 1)
    Next
End Sub
```

Once again, we see the criticality and versatility of loop structures in getting this setup accomplished. Similar use of loop structures can then be applied if you're interested in allowing a user to select or deselect all items, or

in allowing a user to save their form selections as new defaults in the workbook:

```

Private Sub SetDefaults_Click() 'Saving defaults to associated range
    For count = 0 To (ListBox1.ListCount - 1)
        Range("StatusHeader").Offset(count + 1) = ListBox1.Selected(count)
    Next
End Sub

Private Sub SelectAll_Click()
    For count = 0 To (ListBox1.ListCount - 1)
        ListBox1.Selected(count) = True
    Next
End Sub

Private Sub ClearAll_Click()
    For count = 0 To (ListBox1.ListCount - 1)
        ListBox1.Selected(count) = False
    Next
End Sub

```

This use of spreadsheet content (perhaps on a hidden worksheet) as a default establishing and resetting mechanism is a nice, easy way to increase usability. It's the same approach, more or less, that the Blackbelt Ribbon uses in storing desired defaults to specific workbooks.

Further additions that can increase the ease of use in complex Userforms include the leveraging of MultiPage and TabStrip controls, as well as control locking and unlocking mechanisms. MultiPage and TabStrip controls essentially multiply the space available on Userforms and allow users to perform tab-based navigation to access specific options made possible by those forms. An open example of this is provided in the Relative Impact and System Ring tools of the Blackbelt Ribbon. In some cases you don't want users to be able to skip ahead. Sometimes you need them to make initial specifications before attempting to make subsequent ones. To that end, VBA manipulation of the “enabled” property of objects comes in particularly handy. For example, if you don't want an individual to be able to access one of the pages in your multipage form, simply change the form's enabled property to FALSE. Change it back to TRUE when you're ready to give them access.

### **9.3.3 Managing Shortcut Defaults**

The discussion of a multiselect ListBox in a Userform brings to a head a related question. What if our users, or we ourselves, have specific preferences regarding the keys set aside for shortcuts that trigger things such as Userform activation – preferences that are specific to the work we are tasked with, and hence the workbook environments we might be working in?

Furthermore, what if a range of different users make use of different tools and would like to have control-key access to these tools in a way that best fits their own needs, whatever those might be? Is there an easy way to redefine a broad set of shortcut key assignments?

Certainly. We could in fact use a tactic similar to that of the default discussion to allow users the ability to set and save default shortcuts to various subroutines they might want to call during the course of using a workbook. The Blackbelt Ribbon's default setting page resource includes the ability to define not only default data and option specification but also the control-key shortcuts to its Blackbelt tools. The code for assigning (or de facto modifying) a control-key activation to any subroutine is fairly straightforward. The following code, for, say, assigning the Ctrl-Shift-z (capital Z) to call an existing subroutine like showMultiSelect would do the trick:

```
Application.OnKey "+^{Z}", "showMultiSelect"
```

The letter “Z” can be replaced by an alternative letter; just keep in mind that some lowercase shortcuts may always have precedence. The additional use of loop structures and text concatenation in VBA using operations like “+” can allow this tactic to be applied across a range of preferences in one fell swoop, perhaps as part of a broader customization protocol – all with the aim of making the user experience as comfortable as possible.

## 9.4 Customizing Excel Ribbon and Menu Interfaces

Before we talk about modifying the ribbon presentations of Excel for showcasing your own tools, let's review the simplest (albeit inelegant) way to create a button for calling subroutines (or opening Userforms, etc.) in Excel. You could start by just drawing your own button. And it doesn't need to be an ActiveX control. For example, you could choose Insert>Illustrations>Shapes (see Figure 9.21) to open the Shapes drop-down menu. For any object, such as a drawn circle or an inserted picture, right-click on that object to see a number of property options including Assign Macro (shown in Figure 9.22). Selecting that option then allows you to specify which subroutine is to be associated with the object's interaction.

After a subroutine is assigned, place the cursor over the object. You will notice a change in the cursor icon from that of an arrow to that of a pointing finger, indicating that the object can now be clicked to run the assigned subroutine. There's really no limit to the number of subroutine activation buttons you can place within a workbook, though if you are going to use this approach, make sure to label them clearly so that it is absolutely obvious what their functionality is. I used a few along these lines in the Chp8\_BasicClock example.

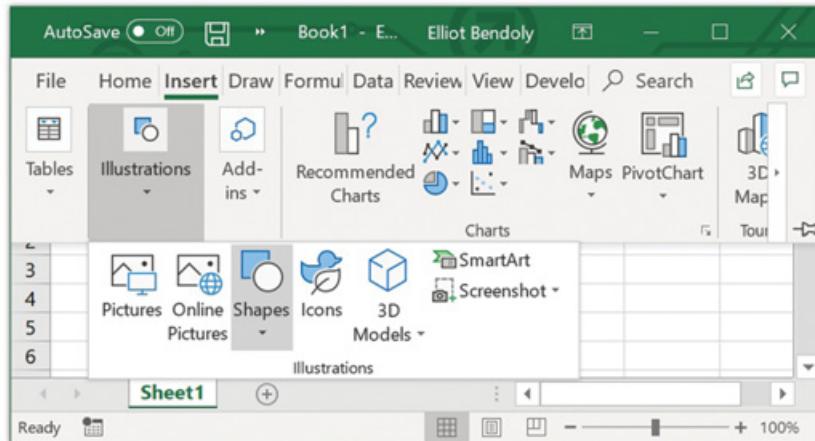


Figure 9.21 Accessing drawing capabilities to generate a subroutine activation button

Along with being able to generate pop-up control panels (aka Userforms) on an as-needed basis, there is also the potential for actually customizing the primary interfaces with which Excel users otherwise normally interact. In other words, instead of having a button (which activates a subroutine or Userform) residing somewhere among the cells of a spreadsheet, you could place that button where all other buttons already built into Excel reside, such as within the command bar, Quick Access bar, or among the drop-down menus (see Figure 9.23).

It's yet another tactic for keeping work and other items out of the way of the premium space available in the spreadsheet environment. And, frankly, we can create these kinds of customized additions to the Excel command button and menu environment fairly quickly through both manual and automated approaches.

#### **9.4.1 Manual Additions**

If the primary development environment is Excel 2007 or later, there is a fairly straightforward approach to adding customized buttons. You can use the Office button to access the Customize screen where any existing tool (including user-developed subroutines) can be added to the Quick Access Toolbar. The Modify Button dialog box, accessed by selecting the “Modify ...” option shown in Figure 9.24, enables you to select a button picture that suits your needs (see Figure 9.25).

Similar additions can be made to the existing ribbons in Excel.

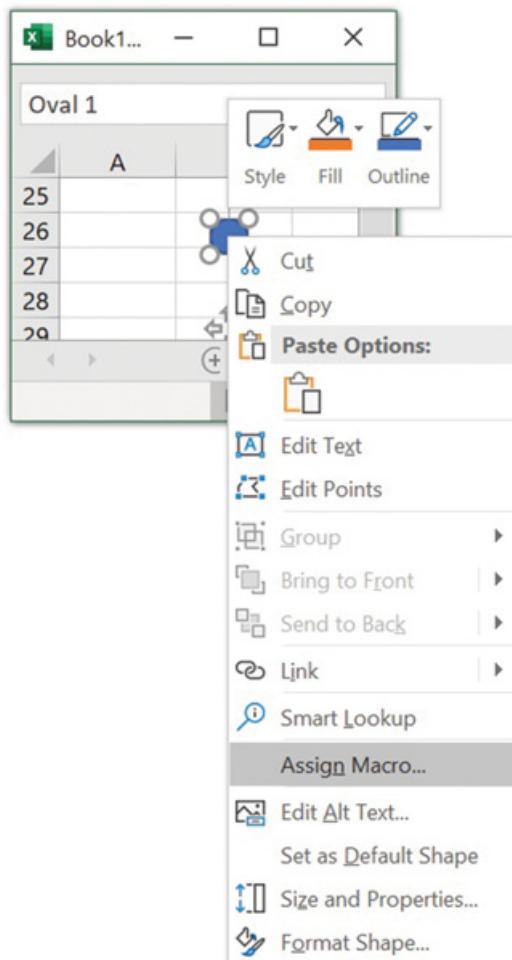


Figure 9.22 Assigning a subroutine to a drawn button



Figure 9.23 Embedded subroutine button versus integration with control/menu interface

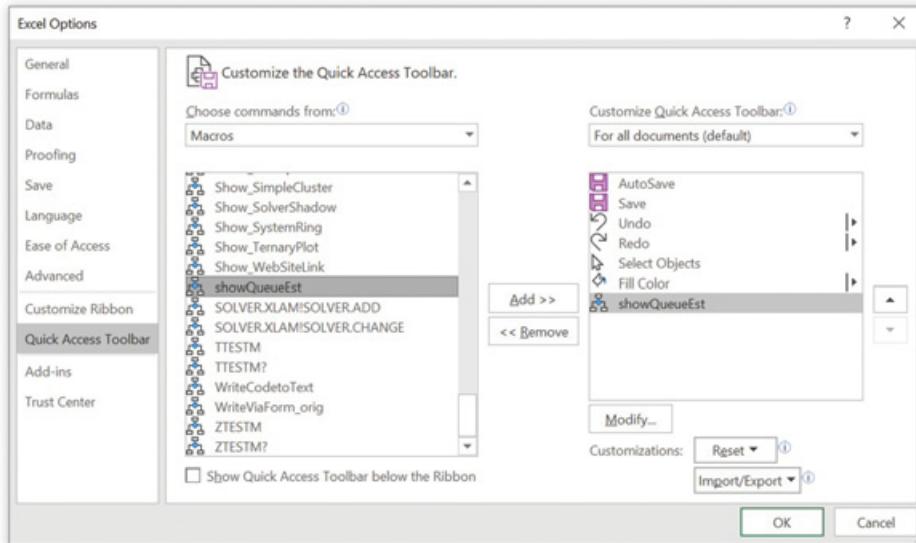


Figure 9.24 Adding subroutine calls to the Quick Access Toolbar

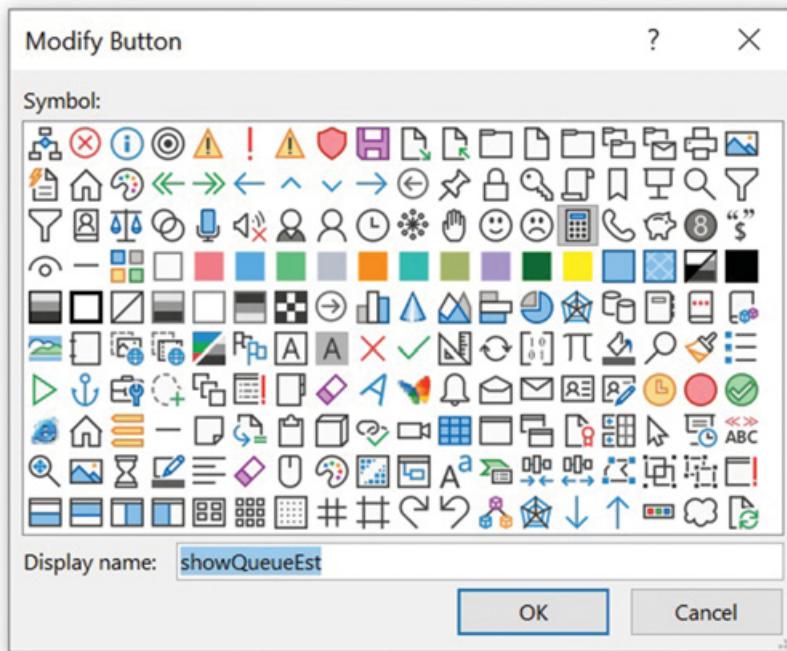


Figure 9.25 Selecting button icons for user-defined buttons in Excel

#### 9.4.2 Fully Automating Button and Menu Setups

Aside from their presence in the menu bar, user-designed buttons have a particular nuance that distinguishes them from buttons that would otherwise be embedded in a spreadsheet. When created, these command bar buttons become registered in a file called Excel.xlb (or something similar, depending on your system configuration), which is usually in your “Roaming” directory. This allows the buttons to appear in your command bar no matter what workbook you open in the future as long as that workbook is on the same computer. For example, if I reboot my computer and then open a fresh workbook in Excel, any command button that I’ve recently added to the menu, ribbon structure, or Quick Access should be there. It should have all the properties I assigned it when I last edited it. It’ll even know enough to find the subroutine to which it was assigned, unless the workbook containing the code for the subroutine has been moved.

The downside is that because the button is defined in that Excel.xlb file and not in a workbook, simply transferring the workbook file (e.g., via e-mail) won’t bring that button along for the ride. There are at least two ways around this. You could insist that clients use the specific Excel.xlb file that you use (which seems an unlikely option in most situations), or you could write a subroutine that creates a set of menu command buttons for your specific DSS (and maybe handles updates in cases where code files are moved around). For illustration, we’ll look to a version of the workbook containing the queuing calculation Userform (discussed earlier). In this version (Chp9\_FunctionPopup\_menu) we have an additional module with code that “installs” a button for calling up that form within a new “Add-ins” ribbon. The code in that module is as follows:

```
Sub AddNewMenuItem()
    Dim CmdBar As CommandBar
    Dim CmdBarMenu As CommandBarControl
    Dim CmdBarMenuItem As CommandBarControl
    On Error Resume Next
    Set CmdBar = Application.CommandBars("Worksheet Menu Bar")
    Set CmdBarMenu = CmdBar.Controls("Tools")
        ' Point to the Tools menu on the menu bar
    CmdBarMenu.Controls("Queue Estimator").Delete
    CmdBar.Name = "Tools"
    'CmdBarMenu.BeginGroup = True
    Set CmdBarMenuItem = CmdBarMenu.Controls.Add
        ' Add a new menu item to the Tools menu
    With CmdBarMenuItem
        .Caption = "Queue Estimator"
        ' Set the properties for the new control
    End With
End Sub
```

```

.OnAction = "" & ThisWorkbook.Name & "'!showQueueEst"
.FaceId = 9145 'many FaceIDs in fact exist as options
.Height = barHeight * 2
.Width = 50
End With
End Sub

```

Running this subroutine (e.g., from the “Add Menu Item” drawn button available in the spreadsheet) will make the new Add-ins ribbon appear. The first icon depicted, in the listing within the Menu Commands group of this ribbon, calls our Userform. You can imagine adding multiple button loads to this installation, with results similar to that depicted in Figure 9.26. As with Quick Access items, these installed buttons will open the files they are tied to in order to run their linked subroutines (if those books are not already open, and again provided they remain where they can be found by the application).

#### 9.4.3 Fully Automating Button and Menu Cleanup

Customized buttons and menu items can also easily be removed manually by right-clicking and selecting removal options (e.g., Delete Custom Command). Alternatively, if you are interested in providing an automatic cleanup of added customized items and buttons, something similar to the following code could be used.

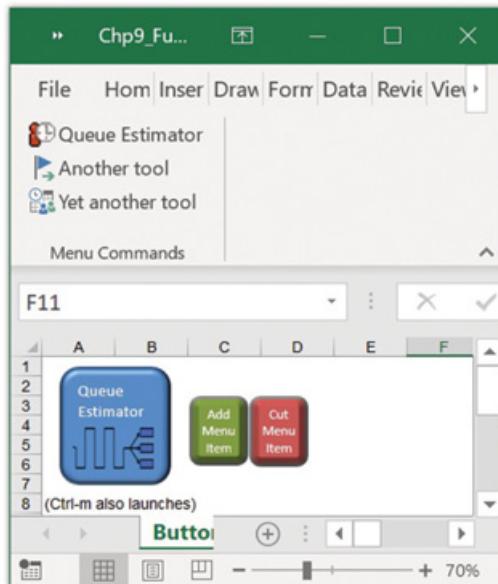


Figure 9.26 Appearance of programmatically installed menu buttons

```
Sub RemoveMenuItem()
    Dim CmdBar As CommandBar
    Dim CmdBarMenu As CommandBarControl
    Set CmdBar = Application.CommandBars("Worksheet Menu Bar")
    Set CmdBarMenu = CmdBar.Controls("Tools")
    On Error Resume Next
    CmdBarMenu.Controls("Queue Estimator").Delete
End Sub
```

Here, again, we can imagine an uninstall procedure that removes a range of buttons that have been added, as desired. It might even be nice to include an Uninstall button and associated Userform as one of the buttons your installation develops. If your prospective users are like most, they would rather avoid the frustration of having to manually uninstall items one at a time, and would much prefer to avoid a search for a separate uninstall resource. Part of making things easy to get to, and easy to use, includes making things easy to get rid of.

## 9.5 Final Thoughts on Packaging

As a final note, all these methods can become part of the installation protocol of a larger add-in you might save and pass on to others for use. However, there are also some very elegant solutions to developing ribbon controls to accompany add-in saves. The most powerful and straightforward approaches are the variations on the RibbonX editor provided by a number of professional developers in this space (for links to associated resources see the bottom listings on the [www.excel-blackbelt.com](http://www.excel-blackbelt.com) site's left bar). In fact, the ribbon structure and appearance of the Blackbelt Ribbon add-in was developed using Andy Pope's editing tool. I won't rehash Andy's instructions on use here, since they are already maintained on associated sites. Suffice it to say that this is an add-in that anyone interested in packaging their own custom add-ins should check out.

There is one thing that I would like to emphasize in all of this, however: Don't give up on the spreadsheet. Why? If we can build interfaces that boil user interaction down to a series of guided button clicks, why would we create elements such as User Defined Functions (UDFs) that are callable through the spreadsheet? If we could get the same result with a Userform, isn't that preferable?

DSS design, like many things, is a balancing act – balancing guidance against restrictiveness, automation against black-boxing. The spreadsheet landscape is still a great arena for laying out multiple calculations that can be updated in real time as other parameters and calculations change, and whose updates can be visualized (i.e., changes depicted graphically) in real time. Replicating that kind of an environment strictly through use of a Userform is

both cumbersome and unnecessary for developers. Similarly, given the fantastic integration between live spreadsheet updates and live graphical updates in Excel, the ability to make abundant use of functions (as opposed to strictly Userform and button-activated VBA-based calculations) should not be underappreciated.

As developers, it's increasingly beneficial for us to be able to offer the best of both worlds in interface designs – allowing routine calculations to be menu driven while also providing a function infrastructure that permits a wide range of user creativity and experimentation in the spreadsheet environment and elsewhere.

### **Supplement: Simple Interface Parallels in iOS Swift**

Interface development in other platforms proceeds using much of the fundamental logic discussed in this chapter. Variables are declared, assigned values, and modified as interactions occur. Objects defined possess traits, and when interacted with can trigger calls to code, which in turn can leverage functions and still other code actions to yield changes in other interface objects. We saw some simple examples of Swift code in Playground Book settings in the Supplement of Chapter 8. Here we can consider what goes into a simple interactive app that leverages conditional statements in the course of a card trick. The app itself, developed in XCode on a Macbook Air, is implemented for iPhone access. A clip of a very early version of this app in action is shown in Figure 9.27.

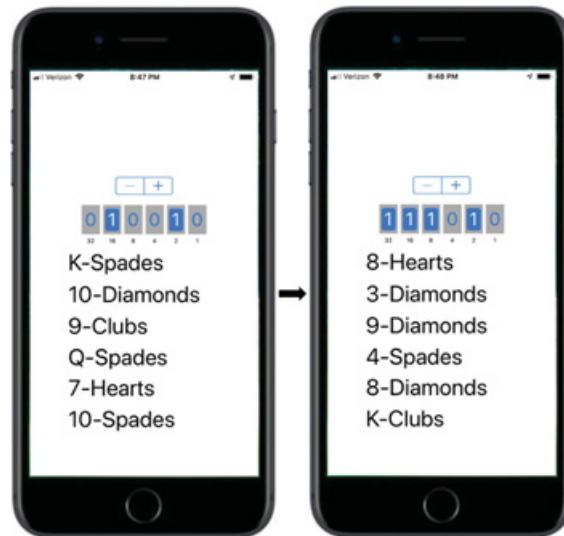


Figure 9.27 A Swift-developed interface for a simple interaction and conditional output

After interface initialization, as buttons are pressed in this interface, the buttons toggle from On to Off (or back On) and a string of 0–1 values is constructed as a kind of unique reference key. This code is then processed for output purposes, revealing critical information to the user (the person performing the trick in this case). Aspects of the code driving the app functionality are provided below. A magician never reveals all secrets, but you'll find that the flow of logic and the nature of this syntax are fairly straightforward and reminiscent of code we've seen in our VBA examples.

```
// ViewController.swift
import UIKit
class ViewController: UIViewController {
// Declarations
    @IBOutlet var Card1: UILabel!
    @IBOutlet var Card2: UILabel!
    //... additional four declarations omitted here
    @IBOutlet var buttons: [UIButton]!
// 0. Initial displayed value should be 0
    override func viewDidLoad() {
        super.viewDidLoad()
        tvalue="000000"
    }
// 1. The following is called each time a button is pressed
    @IBAction func toggle(_ sender: UIButton) {
        sender.isSelected.toggle()
        tvalue = binaryValueOfButtons()
    }
// 2. After a press, simply construct a string of binaries in serial
    func binaryValueOfButtons() -> String {
        var value = ""
        for (index, button) in buttons.enumerated().reversed() {
            if index < 100 {
                if button.isSelected {
                    value += "1"
                } else {
                    value += "0"
                }
            }
        }
        return value
    }
// 3. Called each time tvalue is modified (follow button presses)
    var tvalue = "" {
        didSet {
            var CardsString : String =

```

```

// 52 IF Else statements to test for string value
if tvalue == "100111" {
    CardsString = "9-H 3-S J-C 3-H 8-H 3-D"
// ... next 50 statements omitted here ...
} else {
    CardsString = "7-C 9-H 3-S J-C 3-H 8-H"
}
let CardsStringArr : [String] =
    CardsString.components(separatedBy: " ")
Card1.text = CardsStringArr[0]
Card2.text = CardsStringArr[1]
// ... additional four outputs omitted here ...
}
}
}
}

```

## PRACTICE PROBLEMS

### *Practice 9.1*

Recall the Columbus Professional Training example we first introduced in Chapter 6. We used Solver to find an optimal solution to this problem, given a single set of parameters such as revenue numbers and hours required per student per curriculum for both the geek and beatnik types.

Now, imagine that CPT was reconsidering the number of hours the geek students should be required to take in the two categories of Etiquette and Analytics. The director would like to see how much his revenue-maximizing enrollment would vary under nine different scenarios. Each scenario is represented in Table 9.1.

Table 9.1 *Scenarios for automated analysis and summary*

Revenue Maximizing Enrollments	Total # of Communication Training Hours for “Geeks”		
Total # of Analytic Hours for “Geeks”	7	11	13
2	{#Geeks} {#Beatniks} {\$Revenue}		
6			
10			

Set up a table similar to this in Excel. It doesn't need to be exact; I've used the Merge cell option under Home>Alignment to combine some cells and make the table look neat.

Now, create a subroutine that does the following:

- 1) Copy a new set of "geek" curriculum parameters (hours needed) outlined by one of the row and column combinations in the table (e.g., six hours of analytics and eleven hours of communication) into the template we originally used when solving this problem.
- 2) Run Solver to generate the revenue maximizing enrollment (assuming integer constraints on the number of students).
- 3) Record the revenue maximizing enrollment (i.e., the number of geek and beatnik students) and the associated revenue level generated by Solver for that set of parameters in the associated cells of your table.
- 4) Design your subroutine so that it does this for all nine scenarios, one after another, so that you generate a completely filled table using only one activation (e.g., one click) of the subroutine.

### Practice 9.2

Population distributions are often assumed to be normal, or bell-curved in their shape, with most members of the populations represented at the center of the curve. In statistical terms this location of greatest frequency is referred to as the *mode*. In management scenarios we are often faced with multiple populations that have some overlap (e.g., clusters of demographically related customers that geographically overlap to some degree). Managers interested in dealing with multiple populations simultaneously may be interested in the associated points of greatest frequency (i.e., where the greatest number of people in the combined population live). However, several forms can arise when combining two normally distributed populations. Figure 9.28 shows a couple of simple examples generated in the Chp9\_CompoundDistrib.xls workbook. The first shows a bimodal structure (with one peak representing the global mode); the second shows a single composite mode structure. Numerically, the two pictures differ only in the location of the standard deviation of the second distribution: 1.6 versus 3, respectively.

Build your own function for determining the mode of a distribution resulting from the merger of two normal distributions. Your inputs should be the means and standard deviations for the two distributions, as well as a cell location to dump any resulting output. The output should be the global mode of the composite distribution, and whether a second local peak exists (just give a yes or no response). Hint: Feel free to use some easily parsed concatenation of potential outputs here.

Create a Userform that prompts users to enter the input information into dialog text boxes, and a button that makes VBA go through some kind of loop to determine the outputs required. Use a fresh workbook (not a version of Chp9\_CompoundDistrib.xls) for your work. Do not use any portion of the workbook's spreadsheet to store numbers or do the calculations. This exercise should all be done in VBA.

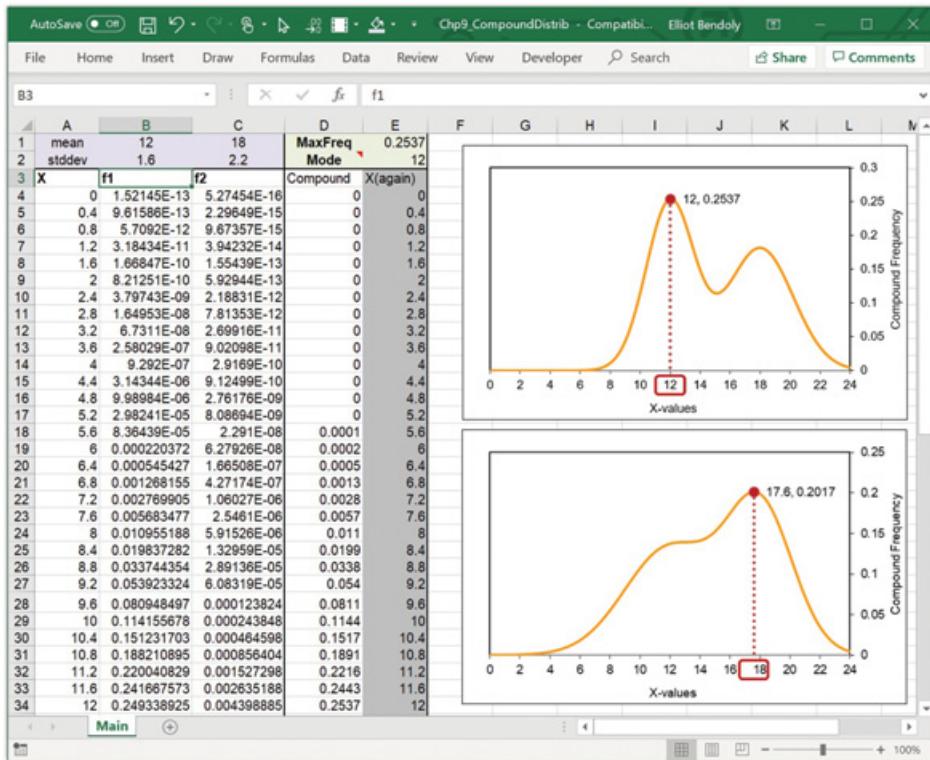


Figure 9.28 Examples of two compound distributions and associated modes

To help you, the following is the code needed to calculate the frequency of a single distribution given a mean and standard deviation:  
`f1 = WorksheetFunction.NormDist(x, mean1, stdDev1, 0)`

Remember, you'll want to calculate a frequency for each and then add the frequencies together. You can conduct your search by constructing each composite frequency in turn (on 0.2 intervals of  $x$ ; 0, 0.2, 0.6, and so on up to 24.0) in code and checking whether each new composite has a greater composite frequency than the maximum previously encountered.

## References

- Bendoly, E., Clark, S. 2017. *Visual analytics for management: translational science and applications in practice*. Taylor & Francis / Routledge: London (ISBN: 978-1138190726).

## Glossary of Key Terms

**Absolute Referencing** [2.5]: The use of \$ ahead of row and/or column designations in references to cell ranges in cell formulations. The use of absolute, or hard, referencing assures that the copying of formula content continues to reference the row and/or column information that is fixed, while adjusting any relative cell references (soft references) as alternate rows and columns are copied into. *[Also see Names]*

**ActiveX Controls** *[See Controls]*

**Add-in** [3.2.2]: An additional application compatible with Excel, designed to extend the capabilities of the spreadsheet environment. Examples include Solver, the Data Analysis tools, and the Blackbelt Ribbon ([www.excel-blackbelt.com](http://www.excel-blackbelt.com)), among others. The VBA developer content of any workbook (forms, subroutines, functions) can be saved as an add-in and made available to any workbook opened on a computer in which the add-in is installed.

**AdjustColors** *[Blackbelt Ribbon function, See Appendix B.4]*

**Best-Fit Line** *[See Regression]*

**Binding Constraints** *[See Connections]*

**Bivariate** *[Blackbelt Ribbon function, See Appendix B.4]*

**Bubble Chart** *[See Scatters]*

**Click Plot** [3. Supplement]: A tool within the Blackbelt Ribbon add-in that allows point-and-click mapping of coordinates associated with imported images. Each point clicked upon in the generated transparent graph (over-laying the image) is recorded in relative coordinates in the spreadsheet for later use.

**Cluster Analysis** [3.6.4]: A computational technique used to identify a set of fairly distinct groups of observations (entities, records) in a data set. It

operates by attempting to place individual observations in groups whose attribute values are most similar, while simultaneously attempting to ensure that the general characters of all formed groups are significantly different from one another. [See also *Simple Cluster*]

**CompSearch\_TSP** [*Blackbelt Ribbon function*, See Appendix B.4]

**Conditional Formatting** [2.2.2]: A mechanism enabled by Excel by which cells in a spreadsheet take on various visual formats based solely on the nature of their content and ostensibly the content of the range of cells that they were coformatted with.

**Conditional Logic** [2.3.2]: The use of criteria to determine which subsequent actions (e.g., calculations) should take place. IF statements and compound functions such as SUMIF in the spreadsheet and VBA back end are examples of conditional logic functions, as are Stdevif and Percentileif in the Blackbelt Ribbon. [Also see *Decision Trees*]

**Confidence Ellipses** [5.3.3]: A tool within the Blackbelt Ribbon add-in that provides a two-dimensional visualization of the area within which a certain percentage of observations is expected to be observed. It assumes that the two variables in question can be represented by a bivariate normal distribution around a central pair of average coordinates.

**Connections** [3.5.2]: Relationships that intertwine utility variables (impactful decisions) and the pursuit of objectives. Often these relationships explicitly prevent utility variables from taking on certain values either in isolation or in tandem with other decisions, but they can also have more nuanced effects. For example, they can outline nonlinear relationships and levels of risk connecting utility variables and the objective. Binding constraint connections (aka Bottlenecks) in particular are those that work to prevent certain kinds of changes to the best discovered options (i.e., objective-optimizing utilities) that might otherwise lead to still greater levels of achievement in the objective. Nonbinding constraint connections demonstrate slack, and changes to these generally do not benefit objective pursuit.

**Constraints** [See *Connections*]

**Controls** [4. Supplement]: A range of mechanisms in Excel through which to develop visually appealing and user-friendly object interfaces for modifying and/or displaying content within a workbook. Controls (and Forms) can reside entirely as embedded objects in a workbook or can appear as needed through the use of pop-up forms callable through Visual Basic code. In many cases they are also capable of and ideally used in referenced calls to user-developed macros.

**Copy Prompt** [2.1]: A small square in the lower right of a selected cell range, the selection and dragging of which, horizontally or vertically, triggers efforts by Excel to copy content, formatting, and formulations. When applied to fixed content, Excel will attempt to extrapolate that fixed data series, mathematically if numerical, or by using list recognition or repetition if nonnumeric. [*Also see Custom List and Format Painter*]

**Custom List** [2.1]: An ordered sequence of text entries recognized by Excel and available for series extension in worksheets.

**Dashboard** [9]: From a general decision-making perspective, a computer interface that allows individual users to simultaneously view various depictions of data and information as well as various subsets of data relevant to a particular task and user context.

**Data Tables** [4.4]: An automated mechanism for generating multiple evaluations of a computational model. The top row and/or left column of a data table can contain inputs to the model. One of the following can contain the output reference(s) from the manipulated model for populating the bulk of the data table: upper leftmost cell, the top row, or left column. Data tables are typically used to describe joint effects of inputs on final or intermediate outputs. They are also very useful in simulation assessments and the calculation of distributions of effects in the presence of noise. [*See Design of Experiments*]

**Data Validation** [4. Supplement]: A user control mechanism that capitalizes on the presence of a list of options, and the attributes of cell objects, to allow for the setup of a simple drop-down menu of options to select from. More customizable drop-down selection structures are available through Form Controls and ActiveX controls. [*See Controls*]

**Decision Support System (DSS)** [1.1]: An application designed to support, not replace, decision making. Often characterized as providing eased access, facilitated analysis, and rich communication – all of which is greatly augmented through intelligent and effective use of visualization.

**Decision Trees** [2. Supplement]: Structures that outline sequential systems of compound logic. Useful in mapping the course of decision-making processes, the course of questions to be asked in order to determine identity or state, or the course of likely events in order to assess the value/cost of early decisions.

**Descriptive Analytics** [1]: A general term referring to any of a variety of approaches to depicting the nature of real-world and modeled observations,

their central tendencies, and risk profiles. [See *Predictive Analytics, Prescriptive Analytics, Risk Profiles*]

**Design of Experiments (DOE)** [4.4.2]: An approach to leveraging data table capabilities and working around their inherent limitations. In a DOE approach, a codex table is created in which a reference number is used to refer to the combination of levels of multiple inputs in a model. Those reference numbers (also occupying the left column or top row of a data table) serve as the primary input that the data table will send to the model. Translation of the reference number into its separate variable components (e.g., using VLOOKUP on the original codex table) is done prior to model evaluation.

**DoEvents** [8.4.3.2]: A command in VBA that allows items such as forms and controls on the spreadsheet to respond to actions taken by users while running subroutines that contain this line of code. It also permits direct changes to cell contents during this time, which in turn will automatically shut down the running of subroutines (without shutting down Excel). [Also see *Loops*]

**Error Handling** [8.6]: VBA code available for identifying and handling errors as they occur during run-time. Generally useful in safeguarding the effectiveness of complex applications developed in workbooks where macros and User Defined Functions are active.

**Evolutionary Search** [See *Genetic Algorithm*]

**External Data** [3]: Any variety of data from external sources of content, including flat text files, databases, and web sites hosted on the Internet. This data can be accessed by tools such as the Get External Data resources, Power BI resources, or add-in tools like Read Write-G and Heat Mapper in the Blackbelt Ribbon.

**ExtractURL** [Blackbelt Ribbon function, See Appendix B.4]

**Filter** [2.4]: A built-in tool within the Excel worksheet environment that permits selective hiding of rows in existing tables when column content does not meet the needs of presentation. Hidden rows remain active in the spreadsheet and in functions referencing those cells, regardless of the rows not being visible. [See *Pivot Table*]

**Forecasting** [See *Predictive Analytics, Regression*]

**Form Controls** [See *Controls*]

**Format Painter** [2.2.2]: A tool designed to replicate the format (only) of cells across other cell ranges.

**Formulae** [See *Functions*]

**Functions** [2.3]: Built-in formulae that can be executed within cells of spreadsheets. Under most scenarios functions (similar to most graphs built in Excel) are automatically updated when changes take place in the content of workbooks or when there are forced updates: for example, through the use of F9 in the spreadsheet or “Calculate” in VBA code. These functions can also be automatically updated in conjunction with Web query and iterative calculation mode functionality. [Also see *User Defined Functions*]

**FurthestNext\_TSP** [Blackbelt Ribbon function, See Appendix B.4]

**Genetic Algorithm** [7.2]: Genetic algorithms, as examples of evolutionary search approaches, are based on at least two principles that are fundamental precepts in the biological sciences: (1) Entities, scenarios, and solutions that do well in a context have traits favored for replication in these contexts (“survival and progeneration of the fittest”). (2) For performance improvements to develop there must be opportunity to capitalize on diversity. Such diversity is made possible through the intermingling of sufficiently large populations and/or the result of perturbations (mutations) that introduce novel changes. These same principles can be used to help analysts develop high-performing solutions for professional practice, which might otherwise be difficult for other approaches, such as hill climbing, to achieve. [See *Hill Climbing, Solver, RISK Optimizer*]

**Hard Referencing** [See *Absolute Referencing*]

**Heat Mapper** [5.3.4.1]: A tool within the Blackbelt Ribbon add-in that allows for the creation of customized heat maps with features that might not be otherwise available by standard packages for the visualization of data. An Internet connection is needed here as well since the library of image sets is hosted online. Used in conjunction with the AdjustColors function, this provides the potential for dynamic and interactive visualization of heat mapped data. [See *AdjustColors* in Appendix B.4]

**Heuristic** [6.2]: Codified approaches to developing ideas, decisions, and/or solutions to problems. Fast and frugal heuristics employ a minimum of time, knowledge, and computation to make adaptive choices in real environments. Heuristics in general typically do not guarantee optimal solutions but can be extremely capable in developing high-performing solutions (relative to random decision making).

**Hill Climbing** [6.3]: A fast and frugal (greedy) heuristic that starts with an initial solution (aka an initial set of utility variable values), considers incremental changes in each of these variables, and pursues changes that yield the

greatest return to the objective. When no further change in the direction you're following yields superior and feasible improvements to the objective (due to the presence of constraints, for example), alternate directions are pursued or the search ends. [See *Heuristic, Solver, Genetic Algorithm*]

**HistoricalInvBB** [*Blackbelt Ribbon function, See Appendix B.4*]

**Image Extract** [5.3.4.1]: A tool within the Blackbelt Ribbon add-in that allows for the extraction of a set of drawn polygons into ordinal listings of relative coordinates for use with tools such as Heat Mapper. [See also *PolyPtsExtract and PolyPtsBuild in Appendix B.4*]

**Iterative Calculation Mode** [3.3]: A calculation mode in Excel that allows for stepwise calculation of functions within a spreadsheet based on the results of previous calculations, even if the calculation referenced is stored in the same cell that the called function resides in (i.e., circular loop calculations). In Excel, the sequence of cell calculations under iterative calculation mode is very specific and needs to be understood before it can be accurately used.

**Link Data Wizard** [*See MapPoint*]

**Living Data Records** [3.3]: Data records that typically consist of a range of cells containing data that are updated on an iterative basis. These records usually represent sequences of calculations or observations and hence typically have some temporal meaning associated with them.

**Locking** [*See Protecting*]

**Loops** [8.4.2]: VBA code structures designed to allow for iteration as a part of a subroutine's (macro's) run. Two commonly used loop structures include fixed-finite repetition loops (For loops) and condition-dependent repetition loops (While loops). Given that loop structures are often associated with lengthy run times, the availability of flexibility and auto-exit mechanisms such as DoEvents scripts can become highly useful to DSS developers.

**Macros** [4.4]: Subroutines coded in Visual Basic for Applications (VBA), often generated through the recording of changes within the spreadsheet environment that enable specific changes to the workbook content, structure, and associated elements (e.g., pop-ups) to take place. Only a limited set of activities can actually be directly recorded and translated into VBA code; however, a wide range of actions can be coded for directly through use of the VBA Editor. External calls to applications and add-ins are also possible through the use of macros/subroutines. Activation of subroutines typically takes place through the use of embedded buttons, customized menu items, or

control-key shortcuts but can also be triggered through calls by other subroutines or applications (e.g., @RISK). [See VBA Editor]

**Names** [2.2.3]: Text that can be associated with cells, cell ranges, worksheets, or objects embedded within an Excel workbook for use in referencing these items in an intuitively meaningful fashion. Names are also useful in locating items in workbooks with considerable added content as well as referencing these items from behind the scenes through VBA code. The Name Manager resource facilitates edits to assigned names.

**NearestNext\_TSP** [Blackbelt Ribbon function, See Appendix B.4]

**Nonbinding Constraints** [See Connections]

**Objective** [3.5]: The main pursuit of a decision-making process. Entirely context- and orientation-specific, and often unique in form, depending on the nature of the problem faced by the decision maker. Also entirely dependent upon the range of decisions available for consideration as well as the constraints placed upon the decision-making process. Typically viewed as compound calculation (e.g., expected total satisfaction, average total risk, and so on) to be either maximized, minimized, or set as close to a particular value as possible.

**Objects** [2.2.4]: While this term can refer to the broad variety of entities within an Excel workbook (sheets, cells, etc.), it can also refer to imported images, controls, and embedded structures and graphics. All objects have a unique set of attributes (properties and methods) that may differ largely from those available to other kinds of objects. These typically include unique references and presentation attributes, among many other functional traits. Many of these attributes are available for manual manipulation and/or activation through the Excel GUI but can also be accessed in VBA code.

**Optimization** [6.3]: The process by which values of decision variables are altered computationally (or derived mathematically) in order to obtain the best possible results in an objective function, subject to constraints on both decision variables and the objective (as well as on any computational method used to drive the process).

**OUtCoMES Cycle** [6.1]: A process of decision making and refinement, involving stepwise documentation and repeated assessment. The OUtCoMES stages include outlining three properties, Objectives, Utilities, and Connections, and executing on three methods, Manifest, Explicate, and Scrutinize. The process can be expressed in a systems-oriented A3

framework along with associated workbook (or other platform) documentation and modeling efforts.

**Path-Directed Flow** [5. Supplement]: The flow of elements through a system based on a structured network of paths and interpath flow dynamics (logic and stochastic mechanisms). Graphically, the mechanism through which entities travel along meaningful attribute space in their transitions between interpretable states (points on a graph).

**PathLength** [Blackbelt Ribbon function, See Appendix B.4]

**Percentileif** [Blackbelt Ribbon function, See Appendix B.4]

**Pivot Charts** [5.2.1.4]: A limited family of simple graphics tethered to Pivot Table summary data. Flexible to manual manipulation in tandem with Pivot Tables. [Also see *Pivot Table, Slicer*]

**Pivot Table** [3.6.3]: A compound filtering mechanism available in Excel that enables the summarizations of multiple related records based on similarities across specific attributes. These attributes can be organized along rows and columns, and specific values of attributes associated with specific blocks of data can be filtered to allow focus and comparison on key subsets of data. As with other tools in Excel, Pivot Tables do not automatically update themselves when input data changes (manual or coded refreshing is needed). More recently the advent of Power Pivot has been made available to connected multitable sources (databases such as Access and Azure cloud repositories) as inputs to Pivot Table organizational structures. This has augmented the scale and flexibility of data accessible to Excel. [Also see *Pivot Charts, Slicer*]

**PoissonInvBB** [Blackbelt Ribbon function, See Appendix B.4]

**PolyPtsBuild** [Blackbelt Ribbon function, See Appendix B.4]

**PolyPtsExtract** [Blackbelt Ribbon function, See Appendix B.4]

**Pop-Ups** [See User Forms]

**Power BI Desktop** [5.3.4.3]: An application that can leverage Excel workbook data to provide a wide range of both standard and exotic graphical capabilities along with the key capacity to create interactive graphics that can be shared on mobile devices.

**Power Map** [5.3.4]: A Microsoft application that provides multilayer geographic visualization capabilities, drawing on data from the Bing engine. Aspects of Power Map or aspects akin to its functionality are found in Filled and 3D Maps as well as the Power BI Desktop.

**Power Pivot** [See *Pivot Table*]

**Power Query** [3.2.3]: A structured approach to Web queries that makes targeted attempts to provide connection and extraction of identified tabular content. [See also *Web Queries, External Data, Read Write-G*]

**Predictive Analytics** [1]: A term generally describing the family of analytical tactics (including regression) that attempts to account for variation in a key variable (Y) by considering existing information on that variable and other associated variables (X) that may serve as predictors of that outcome. The objective of predictive analytics is in part the development of strong predictive models (and their predictions) but also in part the development of models of risk that remain around that outcome. [See *Regression, Simulation*]

**Prescriptive Analytics** [1]: A term generally applied to describe the family of tactics focused on extracting recommended policy/decisions in the form of utility variables that are likely to optimize a target objective, subject to constraining connections. [See *Optimization, Solver, RISK Optimizer*]

**Principal Components Analysis (PCA)** [3.5.2]: A statistical technique that attempts to create a reduced subset of attributes based on a larger set of potentially highly related (and perhaps redundant) attributes. The result is typically a condensed set of higher-level concepts that can ideally be used to more efficiently distinguish the nature of observations in a data set.

**Protecting** [9.2]: In general, a means by which to prevent certain kinds of changes from taking place in a developed spreadsheet or workbook as a whole. A critical part of the protection mechanism involves specifying the extent to which actions are limited under protection and whether or not certain cells are “locked” from modification. Password protection is an option in both spreadsheet and workbook protection cases.

**Read Write-G** [3.2.2]: A tool within the Blackbelt Ribbon add-in that makes available the ability to both write to Google Forms as well as read from Google Sheets or any other online source of data, including web pages housing traded stock, weather, and sports-related data, to list just a few examples. [Also see *External Data, Web Query*]

**Regression** [4.1.2]: A tactic, in the family of predictive analytic approaches, which attempts to assign coefficient values ( $\beta$ ) to predictive variables (X) with the intention of minimizing the sum of squared differences between all of the observed values of Y and the corresponding summed product of the predictors and those coefficients ( $\beta X$ ).

**Relative Impact (RI) Fishbone** [5.4.2.1]: A tool within the Blackbelt Ribbon add-in that generates an Ishikawa diagram with the superimposition of trends, lags, and risk levels regarding presumed cause and effect.

**Relative Referencing** [*See Absolute Referencing*]

**Risk Profiles** [4.2]: Measurable outcomes tend to follow distributions, often with distinct central tendencies (means, medians, modes, etc.) but also the significant likelihood of observations on either side of these centers, and in some cases far from these centers (e.g., long-tailed phenomena). There can also be multiple, notable, highly probable values distinct from an estimated center (e.g., multimodal phenomena). The overall landscape of possible values, including any distinct center, describes the risk profile of outcomes (Y) that decision makers must contend with. The breadth of these profiles can be mitigated through predictive analytics and prescriptions that modify impactful utility variables (e.g., X's we have control over).

**RISK Optimizer** [7.3]: An application developed to search out solutions to highly complex problems, often characterized by features such as nonlinearity and discontinuity in the objective function (as well as within relational constraints that involve decision variables that are under consideration). Moreover, the tool is designed to enable searches across simulated scenarios of complex problems where various attributes of these problems are subject to uncertainty. In order to tackle these problems, the application makes abundant use of genetic algorithms as a robust globalized search method (see Chapter 7 Supplement), thereby avoiding some of the difficulties that hill climbing procedures might encounter. Callable both through menu interaction and/or through VBA code.

**Scatters** [5.2.2]: Graphical depictions of data used to describe general closeness or proximity among comparable observations as well as to tee up quick examinations of implied association between two more continuous variables. Bubble chart options provide a variation that permits the visualization of a third dimension by point size. Line connections, judiciously applied, can also show time and spatial direction in these associations. Trend lines can further be superimposed to identify univariate statistical relationships. [*Also see Ternary Plot*]

**Simple Cluster** [3.6.4]: A tool within the Blackbelt Ribbon add-in that clusters data, represented in as many dimensions as specified, into as many groups as requested. The clustering process will continue to run until a user-specified limit is reached.

**Simulated Variants** [8.1.1]: Generally a set of structured problems or decision-making scenarios that are equivalent in structure but differ in the

actual values of the parameters used to describe them. Extremely useful in “what-if” analysis, where the conditions of a problem-solving context (and hence the value of decisions constructed) are highly dependent on certain elements of uncertainty.

**Simulation** [4.2]: A family of approaches that involve the generation of random draws from the distribution of inputs not under our control as well as the generation of subsequent outcomes dependent on these inputs and on the utility variable that we do have control over. Outcomes may build over time and can involve feedback loops and connecting constraints, as common in system models. Simulations provide a way to estimate risk profiles of modeled outcomes that are not easily estimated otherwise, and therefore permit comparisons based on alternative decisions (utility variable values). *[Also see Risk Profiles]*

**Slack** *[See Connections]*

**Slicer** [3.6.3.2]: A convenient interactive mechanism for selective aggregation and filtering modifications to Pivot Table summaries and associated Pivot Chart graphics. *[Also see Pivot Table]*

**Soft Referencing** *[See Absolute Referencing]*

**Solver** [6.3]: The standard mechanism provided through Excel to conduct a search for optimal solutions (values of decision variables) to an objective function, subject to constraints. Solver is technically another add-in tool and is therefore subject to the same strengths and limitations as many other tools in Excel. In its most typical use it pursues optimization by means of a hill climbing mechanism (local sensitivity search). It provides rich details regarding the nature of its final solution with respect to its view of the search through elements such as Answer Reports.

**Solver Shadow** [6.3.3]: A tool within the Blackbelt Ribbon add-in that allows you to take an optimization problem that you've designed for use with Solver and document its various steps as it proceeds to seek out a solution.

**Sparklines** [5.2.1.3]: Simple graphical elements that allow any single series of data, divided into discrete bins, to be visually depicted directly within cells of the spreadsheet.

**Stdevif** *[Blackbelt Ribbon function, See Appendix B.4]*

**Subroutines** *[See Macros]*

**System Ring** [5.4.2.1]: A tool within the Blackbelt Ribbon add-in that allows you to take data in the form of a symmetrical matrix of forward and

backward flow, and diagonal of current system-stock levels, and plot the associated data and labels to emphasize codependencies and feedback mechanisms across various factors in the system.

**Ternary Plot** [5.2.2.1]: A tool within the Blackbelt Ribbon add-in that plots three interdependent variables in a ternary plot graphic. The three variables should add up to a meaningful sum, such as a budget, or are each percentages of a common whole.

**Text Import** [*See External Data*]

**Tools** [2.6]: Additional mechanisms that are often built in to Excel to facilitate the organization and analysis of information in workbooks. These include mechanisms by which to sort information and conduct complex statistical computations, among others. However, unlike other features of Excel (such as most functions and graphs) the functionality of these tools is typically not characterized by live updating – that is, changes in the content of workbooks, including the data that they are intended to operate on, typically do not alter their results automatically. Whereas other means by which to automate updates of their results exist, this is a necessary caveat to consider when relying on their use.

**Trend Lines** [*See Regression*]

**Types** [*See VBA Editor*]

**Userforms**/[9.3]: Compound graphic user interface (GUI) structures that present numerous input and output options on an as-needed basis, appearing only when called for by the user or as a portion of a subroutine run sequence. These new form windows are not embedded in the spreadsheets in the same way that many fixed and standard controls may be, and can be designed as modal, to float out of the way of the spreadsheet or exist visually apart from the spreadsheet entirely, while simultaneously allowing for continuous interaction with the Excel workbook that generates them.

**User Defined Functions (UDF)** [8.5]: VBA code that allows inputs to be specified, complex calculations made, or information derived, and individual results to be posted to cells or other macros that make calls to the code. In spreadsheet usage these can operate more or less like any other function built into Excel, including providing automatic updates to results as data relevant to the calculations are modified. [*Also see Functions*]

**Utility Variables** [3.5.2]: The elements of a decision-making process over which a decision maker has either direct or indirect control (i.e., which change as a consequence of other decisions made). These variables in turn impact the value taken on by the “objective function” and hence the

relationship between these variables and that function mitigates the pursuit of the overall objective of the decision-making process, which is also subject to the presence of additional connecting constraints on these variables.

**VBA Editor [8]:** The fundamental interface that allows developers to view, edit, and generate code (e.g., macros and User Defined Functions) for use in Excel-based DSS. A critical key to leveraging the various capabilities of Excel and linked applications in a seamless and integrated manner, the VBA Editor provides multiple mechanisms through which to facilitate development including debugging tools, help mechanisms, and step-through execution (as needed). Key to the leveraging of complex code structures is an understanding of the role of user-defined variables and variable types (structures) that can make the best use of data resources and calculations toward a particular programming goal.

**Web Query [3.2]:** An external data acquisition approach that attempts to connect and import available content (regardless of tabular or nontabular structure) from an online source. [*Also see Power Query and Read Write-G*]



## Appendix A Workbook Shortcut (Hot Key) Reference

### Popular Function Keys (F#)

<b>F1 Key</b>	Pop-up Help	<b>F7 Key</b>	Spell Check
<b>F2 Key</b>	Edit cell content	<b>F9 Key</b>	Recalculate
<b>F4 Key</b>	Toggles hard references (\$)	<b>F12 Key</b>	Save File as . . .
<b>F5 Key</b>	Pop-up Goto menu		
<b>F6 Key</b>	Hit twice – Activates key-driven menu (provides guide to F6-# shortcut commands; e.g., F6-f opens the Office Button menu)		

**Standard Popular CTRL-# shortcuts** (*assuming that these are not overwritten by user-defined macro shortcuts*)

<b>CTRL-a</b>	Select entire worksheet	<b>CTRL-1</b>	Displays Format Cells dialog Box
<b>CTRL-b</b>	Toggles Bold Text (CTRL-2 does same)	<b>CTRL-2</b>	Toggles Bold Text (CTRL-B does same)
<b>CTRL-c</b>	Copies the item or items selected to the Clipboard (can be pasted elsewhere using CTRL-V)	<b>CTRL-3</b>	Toggles Italics (CTRL-I does same)
<b>CTRL-d</b>	Copies Down the topmost cell in a range of cells to the rest of the selected range	<b>CTRL-4</b>	Toggles Underline (CTRL-U does same)
<b>CTRL-f</b>	Displays the Find dialog box	<b>CTRL-5</b>	Toggles Strikethrough Text
<b>CTRL-g</b>	Pop-up Goto menu (as does F5)	<b>CTRL-6</b>	Toggles display of objects
<b>CTRL-h</b>	Displays the Replace dialog box	<b>CTRL-9</b>	Hides the selected rows
<b>CTRL-i</b>	Toggles Italic Text	<b>CTRL-0</b>	Hides the selected columns
<b>CTRL-k</b>	Insert/Edit Hyperlink dialog		
<b>CTRL-l</b>	Displays the Create Table dialog box		
<b>CTRL-n</b>	New File		
<b>CTRL-o</b>	Open File		

<b>CTRL-p</b>	Print		
<b>CTRL-r</b>	Copies Right the leftmost cell in a range of cells to the rest of range	<b>CTRL-End</b>	Go to Cell of last Row and last Column containing data
<b>CTRL-s</b>	Save File	<b>CTRL-Home</b>	Go to Cell A1
<b>CTRL-u</b>	Toggles Underlined Text	<b>CTRL-Up</b>	Go to Top Row of data/sheet
<b>CTRL-v</b>	Paste the contents of the clipboard	<b>CTRL-Right</b>	Go to Rightmost Column
<b>CTRL-w</b>	Closes selected workbook window	<b>CTRL-Left</b>	Go to Leftmost Column
<b>CTRL-x</b>	Cut the selected item	<b>CTRL-Down</b>	Go to Bottom Row of Column
<b>CTRL-y</b>	Redo the last undone action	<i>Adding "Shift" to the above six selects range</i>	
<b>CTRL-z</b>	Undoes the last action	<b>CTRL-PageDown</b>	Go to Next Worksheet
		<b>CTRL-Spacebar</b>	Select the entire Column
<b>CTRL-;</b>	Insert Current Date	<b>CTRL-&amp;</b>	Applies the outline border
<b>CTRL-:</b>	Insert Current Time	<b>CTRL-_</b>	Removes the outline border
<b>CTRL-~</b>	Applies the General number format	<b>CTRL-^</b>	Exponential format (two decimals)
<b>CTRL-\$</b>	Currency format (two decimals)	<b>CTRL-#</b>	Date format
<b>CTRL-%</b>	Percentage format (no decimals)	<b>CTRL-@</b>	Time format
<b>CTRL-!</b>	Number format (two decimals)	<b>CTRL-'</b>	Copies formula from cell above into active cell
<b>CTRL-`</b>	Toggles display of cell values vs. display of formulas in the worksheet	<b>CTRL-"</b>	Copies the value from cell above into active cell
<b>Some Popular ALT-# shortcuts</b>			
<b>ALT</b>	Grants keyboard access to ribbon	<b>ALT-F8</b>	Opens macros
<b>ALT-=</b>	Generates sums of rows and columns	<b>ALT-F11</b>	Opens Visual Basic Editor
<b>Critical Function Keys in VBA Developer</b>			
<b>F8</b>	Step into (line-by-line execution)	<b>F9</b>	Toggle Breakpoint

## Appendix B Blackbelt Ribbon Add-in Tools and Functions

Blackbelt Ribbon Tools			
Information	<b>Blackbelt Website</b>  Chapter 3		<a href="#">Update Shortcuts</a> Chapter 3  <a href="#">Initialize Defaults</a> Chapter 3
Data Acquisition	<b>Read Write-G</b>  Chapter 3		<b>Click Plot</b>  Chapter 3
Data Visualization	<b>Heat Mapper</b>  Chapter 5		<b>Image Extractor</b>  Chapter 5
	<b>Ternary Plot</b>  Chapter 5		<b>Confidence Ellipses</b>  Chapter 5
	<b>Relative Impact</b>  Chapter 5		<b>System Ring</b>  Chapter 5
	<b>Simple Cluster</b>  Chapter 3		<b>Solver Shadow</b>  Chapter 6

### APP-B.1 Blackbelt Ribbon add-in Tools and Functions



APP-B.2 Preset defaults for Blackbelt Ribbon shortcut keys

	A	B	D	E	F	G	H	I	J	L	M
2		Blackbelt Website		Update Shortcuts	M						
3	Ribbon Tool	Read Write-G	Click Plot	Heat Mapper	Image Extract	Confidence Ellipses	Ternary Plot	Relative Impact	System Ring	Single Cluster	Solver Shadow
4	Macrostem	ReadWriteG	ClickPlot	HeatMapper	ImageExtract	ConfEllipses	TernaryPlot	RelImpact	SystemRing	SingleCluster	W
5	ShowForm Shortcut	A	P	H	L	C	Y	F	S	K	W
6	Rerun Shortcuts	B	G	X	T	D	A	Q	Z		
7	Parameter	https://predashb...gfn/v14	0	Geography-National	C:\Users\ElliottDe...sktop\	16763904	TestData!\$A21:\$E28	TestData!\$A3:\$J16:\$K59	TestData!\$L\$3:\$F73	TestData!B47:\$T3	TestData!I33
8	Parameter	t2chRdat...MwKofz	0	Brasil	NewLibrary.html	8388736	FALSE	TRUE	TRUE	2	TestData!F33:\$F41
9	Parameter	FALSE	10	1.11111e+26	13382655	FALSE	FALSE	FALSE	5		FALSE
10	Parameter	FALSE	10	3	1675103	TRUE	FALSE	FALSE			FALSE
11	Parameter	TestData!\$I\$2	6737515	200	5224	7829367					
12	Parameter	TestData!\$O:\$N\$O	0	26777062	8421504	204					TestData!F42
13	Parameter		0	0	8421504	16764057					TestData!E34:\$E41
14	Parameter	489760393	308.5	FALSE	8421504	16777215					
15	Parameter	1728318723	467	FALSE	52224	FALSE					
16	Parameter	1486038929	TRUE	FALSE	TestData!\$K21:\$R22	TRUE					
17	Parameter	11.914	ShapeVault		value	FALSE					
18	Parameter	-1			value						
19	Parameter	296.329			text						
20	Parameter	460			TRUE						
21	Parameter	TRUE			FALSE						
22	Parameter	2152853			FALSE						
23	Parameter	TRUE			FALSE						
24	Parameter	204			FALSE						
25	Parameter	4			#1						
26	Parameter	FALSE			TRUE						
27	Parameter	894									
28	Parameter	43									
29	Parameter	49									
30	Parameter	578									

Below the table, there are three tabs: Defaults, Clicktest, and TestData.

APP-B.3 Sample Blackbelt Ribbon defaults page view

*Blackbelt Ribbon Spreadsheet Function*

<b>ExtractURL</b>	<See Chapter 3>	E.g., =ExtractURL(G5)
Provides the address of the first hyperlink associated with a cell.		Parameters: • Range of cell in which hyperlink exists
<b>PathLength</b>	<See Chapter 3>	E.g., =PathLength(P2:R21, TRUE)
Calculates the total path distance along a sequence of points of any number of dimensions.		Parameters: • Your full data range of observations without headers • TRUE if individual records are in each row, and dimensions are by column

(cont.)

---

<b>PolyPtsExtract</b>	<See Chapter 3>	E.g., =PolyPtsExtract("Shape4", "P1")
Extracts all the pairs of coordinates of a drawn polygon and stores these in a newly created text box.		Parameters: <ul style="list-style-type: none"><li>• The name of the drawn polygon for extract</li><li>• Name of the destination cell for later text box transfer</li></ul>
<b>PolyPtsBuild</b>	<See Chapter 3>	E.g., =PolyPtsBuild("P1",0.5)
Draws a new polygon using paired data starting in a cell, assuming two columns of data and multiple rows.		Parameters: <ul style="list-style-type: none"><li>• Name of the upper left cell in the paired data set</li><li>• Relative size of the polygon to be drawn; 1 = Original</li></ul>
<b>PoissonInvBB</b>	<See Chapter 4>	E.g., =PoissonInvBB(4)
Generates a Poisson distributed random number, given a mean. Optional input for percentage, or random.		Parameters: <ul style="list-style-type: none"><li>• Mean value of the Poisson distribution [Optional] Percentile of value desired from distribution</li></ul>
<b>HistoricalInvBB</b>	<See Chapter 4>	E.g., =HistoricalInvBB(c2:c8,d2:d8)
Generates a historically distributed random number, given an array of events and their probabilities.		Parameters: <ul style="list-style-type: none"><li>• Range of events for which you have probability data</li><li>• Range of probabilities for these events (summing to 1)</li><li>• [Optional] Percentile of value desired from distribution")</li></ul>
<b>Stdevif</b>	<See Chapter 5>	E.g., =Stdevif(j4:j25, "Low",m4:m25)
Delivers the standard deviation of a set of cells subject to criteria, consistent with AverageIF.		Parameters: <ul style="list-style-type: none"><li>• Range of data that will be examined by criteria</li><li>• Criteria to be checked against for data use</li><li>• Range of cells that might contain values to include in calculation of standard deviation</li></ul>
<b>Percentileif</b>	<See Chapter 5>	E.g., =Percentileif(j4:j25, "Low",m4:m25,0.25)
Delivers the value at the percentile of a set of cells, subject to criteria, consistent with AverageIF.		Parameters: <ul style="list-style-type: none"><li>• Range of data that will be examined by criteria</li><li>• Criteria to be checked against for data use</li><li>• Range that might contain values to include in extraction of percentile value</li><li>• The percentile level sought for the value to be returned; e.g., 0 = Min, 1 = Max, 0.5 = Median</li></ul>

(cont.)

---

<b>Bivariate</b>	<See Chapter 5>	E.g., =Bivariate(0.5,0.5,X12,X13,Y12, Y13,-0.2)
Delivers the bivariate distribution frequency at a coordinate pair, given both means, stdevs, and correlation.		Parameters: <ul style="list-style-type: none"><li>• X coordinate</li><li>• Y coordinate</li><li>• X Mean</li><li>• Y Mean</li><li>• X Standard Deviation</li><li>• Y Standard Deviation</li><li>• X-Y Correlation</li></ul>
<b>AdjustColors</b>	<See Chapter 5>	E.g., =Percentileif(A1,Rand(),"Auto-updating")
Adjusts the colors of a HeatMapper generated polygon set, based on header and left column cell colors and second column values.		Parameters: <ul style="list-style-type: none"><li>• Upper left cell of associated Heat Mapper data series</li><li>• Either a fixed value, reference, or Rand () depending on updating preferences</li><li>• What content/notes this cell should present to the "user"</li></ul>
<b>CompSearch_TSP</b>	<See Chapter 6>	E.g., =CompSearch_TSP(D\$4:E13)
Comprehensively examines all $N!$ sequences of $N$ X-Y points, and provides best, worst, and the option to view all other sequences encountered.		Parameters: <ul style="list-style-type: none"><li>• Range of X-Y coordinate pairs</li><li>• [Optional] Output cell for transfer of all sequences from textbox to spreadsheet</li><li>• [Optional] Whether the textbox is equipped with an auto-extract macro</li></ul>
<b>NearestNext_TSP</b>	<See Chapter 6>	E.g., =NearestNext_TSP(D\$4:E13, Z1, TRUE)
Applies the Nearest-Next heuristic to examine $N*(N + 1)/2$ sequences of $N$ X-Y points, and provides best and worst encountered, and the option to view all other sequences encountered.		Parameters: <ul style="list-style-type: none"><li>• Range of X-Y coordinate pairs</li><li>• [Optional] Output cell for transfer of all sequences from textbox to spreadsheet</li><li>• [Optional] Whether the textbox is equipped with an auto-extract macro</li></ul>
<b>FurthestNext_TSP</b>	<See Chapter 6>	E.g., =FurthestNext_TSP(D\$4:E13, Z1, TRUE)
Applies the Furthest-Next heuristic (opposite of Nearest-Next) to examine $N*(N + 1)/2$ sequences of $N$ X-Y points, and provides best and worst encountered, and the option to view all other sequences encountered.		Parameters: <ul style="list-style-type: none"><li>• Range of X-Y coordinate pairs</li><li>• [Optional] Output cell for transfer of all sequences from textbox to spreadsheet</li><li>• [Optional] Whether the textbox is equipped with an auto-extract macro</li></ul>

---

# Index

- A3 frameworks, OUtCoMES cycle in, 209–210  
ActiveX controls, 136–139  
ActiveX Media Objects, 319–322  
add-ins, 356–358. *See also* Blackbelt Ribbon;  
    Solver  
    in Web Query interfaces, 47–51  
aggregation of data. *See* data aggregation  
alignment, in static formatting, 19  
answer reports, Solver, 241, 242, 249  
anthropomorphism, in DSS, 8  
attribute consolidation and grouping, 60–66  
    in data aggregation, 59, 60  
    Dodecha Solutions, Ltd., 62  
        management issues, 63  
    Palisade's StatTools ribbon, 63–66  
        PCA in, 64–65, 66  
    PCA  
        in Palisade's StatTools ribbon, 64–65, 66  
        in statistical attribute grouping, 61–62  
auto-reset toggles, 299–300, 301
- bar charts, 145–155  
    box-whisker plots in, 151–154  
    cell-embedded sparklines, 154, 155  
    data preparation in, 146–149  
    data validation, 147–149  
    graphics in, access to, 150  
    Pivot Charts, 154–155, 156  
    simple, 149–154
- BI tools. *See* Business Intelligence tools  
binding constraints, 243  
Binomial distribution, in simulation, 102  
Blackbelt Ribbon. *See also* specific tools  
    Confidence Ellipse tool, 169, 170  
    data generation, 82–85  
        Click Plot tool, 83–84  
        Pathlength function, 84–85  
        PolyPtsExtract function, 84–85  
    Heat Mapper tools, 173–174  
        libraries in, 174  
        packaging of, 399–400  
    RI Fishbone diagram, 186  
    Shadow tool interface, 236  
    Simple Cluster tool, 78–79  
    Ternary Plot tool, 164  
    visualizations with, 143–144
- conditional variation via, 146  
Web Query interfaces with, 48–51
- borders, in static formatting, 19
- box-whisker plots, 151–154, 168–169
- bubble charts, 161–164  
    comparisons between, 161  
    rotational, 163  
    Ternary Plot tool, 164
- Business Intelligence (BI) tools  
    Power BI Desktop, 177–180  
        mobile sharing, 179  
    for Web Query interfaces, 51–53
- button icons, for Excel ribbons, 396, 398
- Cartesian heat mapping, 170–172  
causal mapping, in systems structures visualization  
    conceptual, 185–188  
constraints in, 184–190  
in predictive modeling, 184–185  
scatter plots, 190, 192  
of system dynamics, 188–190  
System Ring tool, 187, 188
- Cause-and-Effect diagram. *See* Fishbone diagram
- cell-embedded sparklines, 154, 155
- cells, 17–25. *See also* formatting  
    fixed data in, 12–13  
    formulae in, 12–13  
    live data links in, 12–13  
    names in, 12–13  
    in worksheets, 12–13, 23
- charts, visualization with, 145–167. *See also* bar charts; scatter plots
- bubble, 161–164  
        comparisons between, 161  
        rotational, 163  
        Ternary Plot tool, 164
- data source editing interfaces, 151
- circular calculations, 54–56
- Click Plot tool, 83–84
- closed-form normative modeling, 105–106
- cluster development, evolutionary search on, 274–280. *See also* k-means approach;  
    multidimensional splits; Simple Cluster code. *See* Visual Basic for Applications Editor
- coefficients, in regression processes, 94
- COLUMN function, 335–336

- comments  
 application of, 21–23, 25  
 in cells, 12–13  
 complex areas, visualization of, 168–180. *See also*  
 polygonal heat mapping  
 Cartesian heat mapping, 170–172  
 regional definition of, 172–180  
 surface maps, 170–172  
 3D Surface plots, 171, 172  
 of uncertainty, 168–170  
 concept mapping, in OUtCoMES cycle, 206  
 conceptual structure causal mapping, 185–188  
 conditional formatting, 19–21  
 access to, 20  
 development of, 20  
 for multiple cells, 21  
 of other cells, 21  
 conditional statements, 32, 35–39. *See also*  
 IF statements  
 analytic transitions, 37–38  
 from conceptual to computational formulaic  
 logic, 37  
 tree structures, 36, 37  
 Confidence Ellipse tool, 169, 170  
 Connections specification, in OUtCoMES cycle,  
 204–205  
 constraints  
 binding, 243  
 in OUtCoMES Cycle, 204–205  
 in regression processes, 94  
 RISK Optimizer, 269  
 in system structures, visualization of, 181–184  
 causal mapping, 184–190  
 decision-making and, 184  
 scope of, 181–182  
 continuous random variables, in simulation, 97–99  
 dashboards  
 definition of, 362  
 development of, 363  
 hiding interfaces, 385  
 Userforms, 386–393  
 construction of, 387–389  
 control applications, 388  
 linkage of, 387–389  
 options, 390–392  
 shortcut defaults, management of, 392–393  
 workbooks, 382–386  
 hiding interfaces, 385  
 locking and protecting mechanisms, 383  
 protection options, 386  
 RISKOptimizer interfaces, 376  
 data aggregation, 58–60. *See also* cluster  
 development; k-means approach; Principal  
 Components Analysis; Simple Cluster  
 attribute consolidation, 59, 60  
 data entry, 13–16  
 cell selection in, 13–15  
 copying across in, 13–14  
 copying down in, 14  
 extrapolation in, 16  
 through pattern recognition, 16, 17  
 initial, 14, 15  
 switching sequences in, 15  
 data generation hacks, 82–85
- Click Plot tool, 83–84  
 Pathlength function, 84–85  
 PolyPtsExtract function, 84–85  
 data links, live, 12–13  
 data management, 33–34  
 filter application tool, 33  
 for hidden rows, 34  
 data tables, 110–117. *See also* simulation  
 simulation optimization, 286–287  
 databases, functions for, 27  
 date/time functions, 26, 335  
 decision logic, utilities with, 249–251  
 decision support systems (DSS)  
 anthropomorphism in, 8  
 approaches to, 3–5  
 analytic, 4–5  
 common elements of, 4  
 development stages for, 5–8  
 heuristics in, 219–220  
 intentions for, 3–5  
 overview of, 8–11  
 resources for, 8–11  
 scope of, 8  
 visualization in, 5–7  
 causality in, 7  
 comparisons in, enforcement of, 6–7  
 content factors in, 7  
 content integration as element of, 7  
 multivariate displays as element, 7  
 Design-of-Experiments (DOE) methods, 120, 121,  
 122, 123  
 Developer tab, 308  
 development environment, front-end elements, 13  
 discrete random variables, in simulation, 100–104  
 Dodecha Solutions, Ltd., 62  
 management issues, 63  
 DOE methods. *See* Design-of-Experiments  
 methods  
 Do-While loops, 344  
 DSS. *See* decision support systems
- Edit Custom Lists, pattern recognition and, 16, 17  
 engineering applications, functions for, 27  
 error handling, in VBA Editor, 353–356  
 code flow influenced by, 355  
 evolutionary search. *See also* RISK Optimizer  
 on cluster development, 274–280  
 on group development, 274–280  
 improvements in, 279  
 on schedule development, 280–282  
 Excel ribbons. *See also* Blackbelt Ribbon  
 button icons, 396, 398  
 customization of, 393–399  
 drawing capabilities, access to, 394  
 fully automating buttons, 397–399  
 menu cleanup, 398–399  
 manual additions, 394  
 menu setup, 397–399  
 packaging of, 399–400  
 subroutines, 395, 396  
 Explicate stage, in OUtCoMES  
 cycle, 208  
 extrapolation, 16  
 through pattern recognition, 16  
 Edit Custom Lists, 16, 17

- fill, in static formatting, 19  
filled maps, 174–177  
filter application tool, in data management, 33  
  for hidden rows, 34  
filtering, selective, 58–60  
  in multidimensional bins, 74–75  
  in pivoting, 74  
  slicer, 75  
  timeline, 75  
financial data, functions for, 26  
Fishbone diagram, 185  
  RI, 186  
fixed data, in cells, 12–13  
fixed-finite loops, 340–342  
fonts, in static formatting, 19  
form controls, in simulation, 136–140  
  ActiveX controls, 136–139  
  list-box options, 139–140  
formatting, of cells, 17–25  
  comments, application of, 21–23, 25  
  conditional, 19–21  
    access to, 20  
    development of, 20  
    for multiple cells, 21  
    of other cells, 21  
hyperlinks, 25  
multicell ranges in, 21–23  
  conditional formatting and, 21  
naming in, 21–23  
  Name Manager, 22–23  
  for objects, 23, 24  
  for referencing, 22  
  in worksheets, 23, 24  
notes, application of, 21–23, 25  
static, 18–19  
  alignment in, 19  
  borders in, 19  
  fill in, 19  
  fonts in, 19  
  numbers in, 18  
  protections in, 19  
formulae, in cells, 12–13  
For-Next loops, 342  
fully automating buttons, Excel ribbons, 397–399  
  menu cleanup, 398–399  
functions, 25–33  
  for databases, 27  
  for date and time, 26  
  for engineering applications, 27  
  for financial data, 26  
  for informational data, 27  
  logic, 27, 32–33  
    conditional, 32  
    IF statements, 32–33  
  lookup, 27, 28–32  
    INDEX, 29–31  
    MATCH, 29–31  
    OFFSET, 29–31  
    VLOOKUP, 28–29, 103, 104  
  for mathematical calculations, 26  
reference, 27, 28–32  
selection of, 26  
  by category, 26  
for statistical data, 27  
for words and text, 27  
genetic algorithms, 265  
  for evolutionary search, 264–267  
  iterations by, 266  
  RISK Optimizer options, 282–284  
Google Sheets, 377–382  
  cross-application communication, 377  
  customized specifications, 384  
Form Data, 380  
  VBA Editor, 382  
group development, evolutionary search on, 274–280  
Heat Mapper tools, 173–174  
  libraries in, 174  
heat maps. *See* polygonal heat mapping  
heuristics, in solution development, 210–220  
  DSS designs, 219–220  
  for mazes, 211–213  
  MiniSlack heuristic, 219  
  Nearest-Next, 255, 274  
  objectives in, documentation of, 215  
  recognition heuristics, 213–214  
  sequencing heuristics, 214–219  
  SPT heuristic, 219  
hill climbing algorithms, 255–264  
  alternative approaches to, 263  
  complex nonlinearity, 257–264  
  discontinuity in, 257–264  
  simple linearity, 256–257  
hovering windows, in VBA Editor, 332  
hyperlinks, 25  
IF statements, 32–33, 336–340  
Image Extract tool, 173–174  
images, integration of, in visualization processes, 7  
Immediate windows, in VBA Editor, 332  
INDEX function, 29–31  
informational data, functions for, 27  
interfaces. *See specific interfaces*  
iOS Swift, 358–360, 400–402  
Ishikawa diagram, 185, 186  
iterative calculation mode, 54–56, 130–132  
k-means approach, 80. *See also* cluster development; multidimensional splits  
LINEST function, 94–97  
  sample outputs, 96  
list-box options, as form control, 139–140  
live data links, in cells, 12–13  
living data records, 53–58  
  circular calculations, 54–56  
  iterative calculations, 54–56  
  WebQuery histories, 56–58  
  structures of, 57  
logic functions, 27, 32–33  
  conditional, 32  
  IF statements, 32–33  
logic matrix, OUTCOMES cycle, 207  
lookup functions, 27, 28–32  
  INDEX, 29–31  
  MATCH, 29–31  
  OFFSET, 29–31  
  VLOOKUP, 28–29  
  in simulation, 103, 104

- loops, as iteration structures, 340–345  
 Do-While, 344  
 fixed-finite, 340–342  
 For-Next, 342  
 open-ended, 343–345
- macros, 130–132. *See also* Subroutines  
 iterative calculation mode macros, 130–132  
 multi-iterative calculation mode, 132–135  
 reset macros, 298
- Manifest stage, of OUtCoMES cycle, 205–207
- mapping. *See also* causal mapping; polygonal heat mapping  
 concept, in OUtCoMES cycle, 206
- MATCH function, 29–31
- mathematical calculations, functions for, 26
- MiniSlack heuristic, 219
- mobile sharing, 179
- multicell ranges, formatting of, 21–23  
 conditional formatting, 21
- multidimensional bins, 70–76  
 filtering by rows and columns, 74–75  
 quartile cross-binning, 71, 76
- multidimensional splits, cluster analysis for, 76–82  
 k-means approach, 80  
 from reduced component set, 81  
 by Simple Cluster tool, 78–79
- multiple-iterative calculation mode, 294–300  
 auto-reset toggles, 299–300, 301  
 evaluation in, 300  
 reset Macros, 298  
 selection in, 300
- multivariate displays, in visualization, 7
- Name Manager, 22–23
- names, in cells, 12–13, 21–23  
 Name Manager, 22–23  
 for objects, 23, 24  
 for referencing, 22  
 in worksheets, 23, 24
- Nearest-Next heuristics, 255, 274
- notes, application of, 21–23, 25
- numbers  
 integration of, in visualization processes, 7  
 in static formatting, 18  
 in VBA Editor, 334
- Objective specification, in OUtCoMES cycle, 202, 203, 206
- objects, naming of, 23, 24
- OFFSET function, 29–31, 335–336
- online data acquisition, 44–53  
 through generalized web-data pulls, 44–47  
 data range properties, 47  
 legacy mechanisms, 44–46  
 for spreadsheet content, 45  
 with Web Query interfaces, 45, 46, 47
- Web Query interfaces, 47–53  
 add-ins in, 47–51  
 BI tools, 51–53  
 with Blackbelt Ribbon, 48–51  
 data connections in, 53  
 generalized web-data pulls with, 45, 46, 47  
 Read Write-G, 50
- open-ended loops, 343–345
- optimization, 220–239. *See also* RISK Optimizer; Solver
- OUtCoMES cycle, 201–209  
 in A3 frameworks, 209–210  
 concept mapping in, 206  
 Connections specification, 204–205  
 constraints in, 204–205  
 Explicate stage, 208  
 logic matrix, 207  
 Manifest stage of, 205–207  
 Objective specification, 202, 203, 206  
 in PDCA processes, 209  
 Scrutinize stage, 208–209  
 Utilities specification, 203, 204, 206
- Palisade's @RISK Ribbon, 63–66  
 PCA in, 64–65, 66
- Palisade's StatTools ribbon, 63–66  
 PCA in, 64–65, 66
- Pathlength function, 84–85
- pattern recognition, 16  
 Edit Custom Lists, 16, 17
- PCA. *See* Principal Components Analysis
- PDCA processes. *See* Plan-Do-Act-Check processes
- picture import, in VBA Editor, 316
- Pivot Charts, 154–155, 156
- Pivot Tables  
 in record grouping, 71–73, 76  
 scatter plots and, 159–161  
 pivoting, in record grouping, 71–73  
 filtering by rows and columns, 74  
 mechanisms in, 71–73  
 Pivot Tables, 71–73, 76  
 Power Pivot, 75–76  
 slicer and timeline filters, 75
- Plan-Do-Act-Check (PDCA) processes, 209
- point-structures, in visualization, 193
- Poisson distribution, in simulation, 102
- polygonal heat mapping, 172–180  
 definition of, 172  
 filled maps, 174–177  
 Heat Mapper tool, 173–174  
 libraries in, 174  
 Image Extract tool, 173–174  
 in Power BI Desktop, 177–180  
 mobile sharing, 179  
 3D maps, 174–177  
 timeline playback, 178
- PolyPtsExtract function, 84–85
- Power BI Desktop, 177–180  
 mobile sharing, 179
- Power Pivot, 75–76
- predictive modeling, for systems structures visualization, 184–185
- Principal Components Analysis (PCA)  
 in Palisade's StatTools ribbon, 64–65, 66  
 in statistical attribute grouping, 61–62
- progress graphs, 273
- purchasing problem solutions, 241
- quartile cross-binning, 71, 76
- Read Write-G interface, 50
- recognition heuristics, 213–214

- record grouping, 67–82  
multidimensional bins, 70–76  
filtering by rows and columns, 74–75  
quartile cross-binning, 71, 76  
multidimensional splits, cluster analysis for, 76–82  
k-means approach, 80  
from reduced component set, 81  
by Simple Cluster tool, 78–79  
percentile-based categorization, 67–69  
by p-levels, 69–70  
rank and percentile analysis, 67, 68  
by z-scores, 69–70  
pivoting, 71–73  
filtering by rows and columns, 74  
mechanisms in, 71–73  
Pivot Tables, 71–73, 76  
Power Pivot, 75–76  
slicer and timeline filters, 75  
reference functions, 27, 28–32  
regression processes, 90–97  
coefficients, 94  
constraints, 94  
with data analysis tools, 91–93  
inputs based on spreadsheet structure, 92  
sample output from, 93  
LINEST function, 94–97  
sample outputs, 96  
objectives of, 94  
purpose and function of, 93–94  
in VBA Editor, 318–319  
Relative-Impact (RI) Fishbone diagram, 186  
reorder point problem, 293  
reorder point systems performances, 124–135  
design structure in, 126  
dynamics of, 129  
iterative calculation mode macros, 130–132  
module structure, 127  
multi-iterative calculation mode, 132–135  
summaries of, 129  
RI Fishbone diagram. *See Relative-Impact Fishbone diagram*  
ribbons. *See Blackbelt Ribbon; Excel ribbons*  
risk. *See also regression processes*  
characteristic values of, 90  
profile construction for, 97  
theoretical approach to, 89–90  
RISK Optimizer, 268  
evolutionary search, 267–284  
connecting constraints, 269  
iteration conditions, 271  
objectives, 269  
progress graphs, 273  
solutions available through, 270, 272  
solving methods, 269  
utilities, 269  
genetic algorithm options in, 282–284  
for population size, 283  
inventory system simulation, 372–377  
in multiple-iterative calculation mode, 294–300  
auto-reset toggles, 299–300, 301  
evaluation in, 300  
reset macros, 298  
selection in, 300  
sample output reports, 278  
search-stopping conditions in, 277  
simulation optimization, 284–289  
with data tables, 286–287  
without data tables, 288–289  
descriptive statistics on, 289  
random number generation conditions, 286  
specifications in, 285–286, 287, 288  
stopping conditions, 285  
in single-iterative calculation mode, 291–298  
data tables in, 298  
nonlinearities in, 295  
reorder point problem, 293  
resets in, 292–293  
for system simulations, 289–300  
flow tactics in, 291  
workbook interfaces, 376  
work-group selection, 369–372  
rotational bubble charts, 163  
ROW function, 335–336  
rudimentary attribute grouping, 60–61  
sample output reports, 278  
scatter plots, 156–167  
in causal mapping, 190, 192  
connections in, 164–166  
consistency in association in, 160  
default generations prior to editing, 158  
dimensional options in, 159–164  
editing of, 159  
flow in, 166–167  
source-destination arcs in, 167  
mechanisms for, 157–159  
Pivot Tables and, 159–161  
time-window filtered, 165  
schedule development, evolutionary search on, 280–282  
Scrutinize stage, in OUtCoMES cycle, 208–209  
search methods, 255–264  
*evolutionary search. See also evolutionary search*  
genetic algorithms for, 264–267  
RISK Optimizer options, 282–284  
hill climbing algorithms, 255–264  
alternative approaches to, 263  
complex nonlinearity, 257–264  
discontinuity in, 257–264  
simple linearity, 256–257  
in Solver, 221  
Nearest-Next heuristic, 255, 274  
search-stopping conditions, 277  
selective filtering. *See filtering*  
sequencing heuristics, 214–219  
Shadow tool interface, 236, 237  
Short Processing Time (SPT) heuristic, 219  
simple bar charts, 149–154  
Simple Cluster tool, Blackbelt Ribbon, 78–79  
simulation, 97–104  
Design-of-Experiments methods, 120, 121, 122, 123  
form controls in, 136–140  
ActiveX controls, 136–139  
list-box options, 139–140  
Pivot Tables in, 124  
random variables in, 97–104  
binomial distribution in, 102

- simulation (cont.)
  - continuous, 97–99
  - discrete, 100–104
  - Poisson distribution in, 102
- for reorder point systems performances, 124–135
  - design structure in, 126
  - dynamics of, 129
  - iterative calculation mode macros, 130–132
  - module structure, 127
  - multi-iterative calculation mode, 132–135
  - summaries of, 129
- for revenue management, 108–117
  - structure of, 110
  - with “what if” models, 117
- VLOOKUP function in, 103, 104
- single-iterative calculation mode, 291–298
  - data tables in, 298
  - nonlinearities in, 295
  - reorder point problem, 293
  - resets in, 292–293
- slicer filtering, 75
- Solver, 221–222
  - answer reports, 241, 242, 249
  - Connections in, documentation of, 223
  - constraints, 248
    - binding, 243
    - in production environments, 231
    - specifications for, 224, 227
    - on utilities, 228, 229
  - decision-making
    - for enrollment issues, 225
    - specifications for, 224
  - examples, 222–239, 252
  - model adjustments to, 302
  - nonreferencing messages, 366
  - objectives in
    - documentation of, 223
    - specifications of, 224
  - problem structures in, 226
  - production environments
    - connecting constraints on, 231
    - cost and revenue details, 230
    - decision-making structure for, 231
    - objectives of, 231
    - optimal solutions for, 232
    - requirement specifications, 229
    - utilities in, 231
  - purchasing problem solutions, 241
  - Shadow tool interface, 236, 237
  - spreadsheet construction in, 234
    - layout examples, 246, 247
    - solutions in, 235
  - utilities in
    - documentation of, 223
    - integer constraints on, 228, 229
    - in production environments, 231
  - in VBA Editor, 365–368
  - visualization, 238
  - source-destination arcs, in scatter plots, 167
  - SPT heuristic. *See* Short Processing Time heuristic
  - static formatting, 18–19
    - alignment in, 19
    - borders in, 19
    - fill in, 19
  - fonts in, 19
  - numbers in, 18
  - protections in, 19
  - stopping conditions, 285
  - storage, in VBA Editor
    - changes in information, 330–332
    - comparison of data, 329
    - spreadsheets, 322–333
    - variables, 323–330
  - subroutines, 395, 396. *See also* macros; User Defined Functions
  - surface maps, 170–172
  - system structures, visualization of, 180–190
    - causal mapping
      - conceptual, 185–188
      - constraints in, 184–190
      - in predictive modeling, 184–185
      - scatter plots, 190, 192
      - of system dynamics, 188–190
      - System Ring tool, 187, 188
    - constraints, 181–184
      - causal mapping, 184–190
      - decision-making and, 184
      - scope of, 181–182
    - decision-making and, 180–181
      - constraints and, 184
      - flexible structures in, 183
      - plan development in, elements of, 182
    - RI Fishbone diagram, 185, 186
    - System Ring tool, 187, 188
  - table transfers, 41–43
  - Ternary Plot tool, 164
  - text file imports, 41–43
    - FIND function, 42–43
    - Get & Transform Data, 43
    - MID function, 42–43
    - SUBSTITUTE function, 42–43
    - Text Import Wizard interfaces, 41–42
  - 3D maps, 174–177
    - timeline playback, 178
  - 3D Surface plots, 171, 172
  - time. *See* date/time functions
  - timeline filtering, 75
  - time-window filtered scatter plots, 165
  - tree, structures, for conditional statements, 36, 37
  - uncertainty. *See also* risk
    - of complex areas, 168–170
  - User Defined Functions, 345–353
    - add-ins, 356–358
    - for online stock data, 352–353, 354
    - for queuing equations, 349–352
    - recognition of, 348
  - user interfaces. *See* dashboards; Solver
  - Userforms, 386–393
    - construction of, 387–389
    - control applications, 388
    - linkage of, 387–389
    - options, 390–392
    - shortcut defaults, management of, 392–393
  - Utilities
    - decision logic with, 249–251
    - in OUtCoMES cycle, 203, 204, 206
    - in RISK Optimizer, 269

- in Solver
  - documentation of, 223
  - integer constraints on, 228, 229
  - in production environments, 231
- Visual Basic for Applications (VBA) Editor, 307–356. *See also* macros; subroutines; User Defined Functions; Userforms
  - ActiveX Media Objects, 319–322
  - chart properties, 318–319
    - trend line modifications, 320
  - data tools, modifications to, 316–317
    - slicer selections, 317
  - Developer tab, 308
  - embedded visuals in
    - code management of, 322
    - manipulation of, 315–316
  - environmental elements, 309
  - error handling, 353–356
    - code flow influenced by, 355
  - Google Sheets, 382
  - iOS Swift compared to, 358–360
  - loops, as iteration structures, 340–345
    - Do-While, 344
    - fixed-finite, 340–342
    - For-Next, 342
    - open-ended, 343–345
  - picture import in, 316
  - regression tools, 318–319
  - Solver in, 365–368
  - spreadsheet-related functions, 322–340
    - COLUMN function, 335–336
    - date/time function, 335
    - IF statements, 336–340
    - OFFSET function, 335–336
    - random numbers, 334
    - ROW function, 335–336
  - storage and, 322–333
    - changes in information, 330–332
    - comparison of data, 329
      - in spreadsheet cells, 330
      - variables, 323–330
  - Watch windows, 332
  - Windows Media Player and, 320–321
  - visual design, purpose of, 142–143
  - visualizations
    - with Blackbelt Ribbon, 143–144
      - conditional variation via, 146
    - box-whisker plots, 168–169
    - of complex areas, 168–180. *See also* polygonal heat mapping
  - Cartesian heat mapping, 170–172
  - of confidence, 168–170
  - regional definition of, 172–180
  - surface maps, 170–172
  - 3D Surface plots, 171, 172
  - of uncertainty, 168–170
  - in DSS, 5–7
    - causality in, 7
    - comparisons in, enforcement of, 6–7
    - content factors in, 7
    - content integration as element of, 7
    - multivariate displays as element, 7
    - dynamic paths, 192–194
  - with Google platforms, 191
  - heat maps, 143–144
  - Image Extract tool, 173–174
  - image integration in, 7
  - multivariate displays as element, 7
  - numbers integration in, 7
  - with Palisade's suite, 192
  - point-structures in, 193
  - Power Map, 143–144
  - with R resources program, 191
  - of system structures. *See* system structures
  - with Tableau program, 191
  - word integration in, 7
  - VLOOKUP function, 28–29
    - in simulation, 103, 104
  - Watch windows, 332
  - Web Query interfaces, 47–53
    - add-ins in, 47–51
    - BI tools, 51–53
    - with Blackbelt Ribbon, 48–51
    - data connections in, 53
    - generalized web-data pulls with, 45, 46, 47
    - living data records on, 56–58
      - structures of WebQuery histories, 57
    - Read Write-G, 50
  - web-data pulls, generalized, 44–47
    - data range properties, 47
    - legacy mechanisms, 44–46
    - for spreadsheet content, 45
    - with Web Query interfaces, 45, 46, 47
  - Windows Media Player, 320–321
  - workbooks, 12–13, 382–386
    - locking and protecting mechanisms, 383
    - protection options, 386
  - worksheets, 12–13
    - cells in, 12–13, 23
    - formatting of names in, 23, 24

