Use R !

Six Sigma with R

Statistical Engineering for Process Improvement

Springer

Use R!

Series Editors:
Robert Gentleman • Kurt Hornik • Giovanni Parmigiani

Series Editors:

Robert Gentleman
Program in Computational Biology
Division of Public Health Sciences
Fred Hutchinson Cancer Research Center
1100 Fairview Ave. N, M2-B876
Seattle, Washington 98109-1024
USA

Kurt Hornik
Department für Statistik und Mathematik
Wirtschaftsuniversität Wien Augasse 2-6
A-1090 Wien
Austria

Giovanni Parmigiani
The Sidney Kimmel Comprehensive Cancer
Center at Johns Hopkins University
550 North Broadway
Baltimore, MD 21205-2011
USA

Emilio L. Cano • Javier M. Moguerza
Andrés Redchuk

Six Sigma with R

Statistical Engineering for Process
Improvement

Springer

Emilio L. Cano
Department of Statistics and Operations Research
Rey Juan Carlos University
Madrid, Spain

Javier M. Moguerza
Department of Statistics and Operations Research
Rey Juan Carlos University
Madrid, Spain

Andrés Redchuk
Department of Statistics and Operations Research
Rey Juan Carlos University
Madrid, Spain

ISBN 978-1-4614-3651-5 ISBN 978-1-4614-3652-2 (eBook)
DOI 10.1007/978-1-4614-3652-2
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2012939055

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To our families

Foreword

Many books on Six Sigma have appeared in recent decades. Most of them follow a standard structure: a description of Six Sigma plus some examples of its successful implementation. And many of them have the same motivation: through Six Sigma you will achieve huge cost reductions in your organization, quantified as being between 10 and 30%. It is an appealing claim, especially in the current tough economic climate, but it is an empirical one; as such, it is hard to prove and, of course, a gateway to criticism. You will not find that kind of argument in this book. Its motivation is purely scientific: Six Sigma is "a quality paradigm that translates the involved scientific methodology into a simple way to apply the scientific method." And it really does. When selling a concept within an organization, make sure that the concept is understood. For decades, scientists have been trying to sell great ideas to industry with less than stellar success. In my opinion, this is because, too frequently, we were speaking two different languages. This is a hard obstacle to overcome. Six Sigma has emerged as one of the solutions to this lack of understanding.

This solution did not arise from the scientific camp: it came directly from Motorola and General Electric, two leading industrial organizations. Yet there is a lesson in this for scientists: If you want to be understood by industry, use the language of Six Sigma (or a similar one). And, of course, take Six Sigma as an example of how scientists should work to transfer knowledge and technology to industry.

This book deals with the use of R within Six Sigma. R is an open platform widely used in the academic and scientific communities. Its success is based on two main features: (1) R is robust, rigorous, and efficient and (2) R is free. The latter is a real example of how to reduce costs: if your company uses expensive software tools, then this book is an opportunity to reduce them. It's that simple.

But the book is addressed not just to Six Sigma practitioners. If you do not know what Six Sigma is, then this could be the book for you. The Six Sigma philosophy is introduced from the beginning, making the book a self-contained work. In fact, I strongly recommend the first chapter as a simple and complete description of Six Sigma. All the facts are there.

The authors have combined their scientific, technological, industrial, and pedagogical skills in this one volume. I know them well.

Prof. Javier Moguerza is an accomplished multidisciplinary scientist. He holds active membership in the Global Young Academy. As part of his scientific backpack, he is a specialist in project management. I have seen him work with heterogeneous groups, always successfully. Applying Six Sigma is a natural way of thinking for him. He is “cause-and-effect born” or, if you prefer, “Six Sigma born.”

Dr. Andres Redchuk is a Six Sigma Master Black Belt (read the book’s first chapter to find out what this means). For almost 20 years, he has been working as a Six Sigma practitioner in the private sector, with more than a hundred projects under his belt. He informs the book’s industry point of view.

Emilio Lopez is one of the best Ph.D. students we have seen in the Department of Statistics and Operations Research at Universidad Rey Juan Carlos. He brings broad industry experience and is currently applying Six Sigma as a tool for academic improvement.

The result of these authors’ blending is a perfect conciliation of the scientific and industry points of view. For scientists, the book will describe the needs of the industrial sector. Industrial practitioners, on the other hand, will find an innovative, rigorous, and economical way to apply Six Sigma.

Writing a book about Six Sigma is not an easy task. Writing a useful and original book on Six Sigma is an almost Herculean undertaking. I hope you will benefit from the fruits of the author’s labor.

Royal Academy of Sciences, Spain David Rios

Preface

Why Six Sigma with R

Six Sigma has grown over the last two decades as a breakthrough Total Quality Management methodology. With Six Sigma, problems are solved and processes are improved, taking as a basis one of the most powerful tools of human development: the scientific method. For the analysis of data, Six Sigma requires the use of statistical software, with R an open source option that fulfills this requirement. R is a software system that includes a programming language widely used in academic and research departments. It is currently becoming a real alternative within corporate environments.

Many publications deal with R or Six Sigma separately, but none addresses the use of statistical techniques with R while at the same time focusing on the Six Sigma methodology. Thus, we were encouraged to write this book and develop an R package as a contribution to the R Project. The aim of this book is to show how R can be used as a software tool in the development of Six Sigma projects.

Who Is This Book For?

This book is not intended as a very advanced or technical manual. It is aimed at addressing the interests of a wide range of readers, providing something interesting to everybody. To achieve this objective, we have tried to include as few equations and formulas as possible. When we find it necessary to use them, we follow them up with simple numerical examples to make them understandable to someone with basic math skills. The examples clarify the tools presented using the Six Sigma language and trying to convey the Six Sigma philosophy.

As far as the software is concerned, we have not used complicated programming structures. Most examples follow the structure `function(arguments)` $\rightarrow$ results. In this regard, the book is self-contained because it includes all the necessary background concepts. Nevertheless, we provide plenty of references to both generic and specific R books.

Six Sigma practitioners will find a roadmap at the beginning of the main parts of the book indicating the stage of Six Sigma about to be treated. Moreover, the book contains a chapter with an overview of the R system and a basic reference guide as an appendix. Although you may know about Six Sigma, it is advisable to read the first chapter and check some of the references therein.

Statistical software users and programmers working in organizations using Six Sigma and related methodologies will find in this book a useful alternative. Similarly, analysts and advisers of consulting firms will learn new approaches to their businesses.

Statistics teachers have in a single book the essentials of both disciplines (Six Sigma and R). Thus, the book can be used as a textbook or reference book for intermediate-level courses in engineering statistics, quality control, or related topics.

Finally, business managers who wish to understand and acquire the proper background to encourage their teams to improve their business through Six Sigma can read selected chapters or sections of the book, focusing on the examples.

How to Read This Book

In this book, we present the main tools used in the Six Sigma methodology and explain how to implement them using R. Our intention is to show the relevance of the Six Sigma work flow, the so-called DMAIC cycle (design, measure, analyze, improve, control). This is why, in some parts of the book, concepts that will be defined in subsequent chapters are intuitively used in advance. Detailed description of the concepts can be consulted using the subject index at the end of the book.

The book is organized into seven parts. Part I contains two chapters with the necessary background information for both Six Sigma and R. Parts II–IV constitute the path for our intuitive roadmap. This roadmap goes over the DMAIC cycle throughout its five phases, relating each part to one phase and the main tasks and tools within it. Finally, Part VII comprises a short description of other important tools that the reader should know to have a global view of the advanced possibilities of both Six Sigma and R. An appendix, organized in tables with the main functions used throughout the book, completes the contents.

The chapters have a common structure with an introduction to the given topic, followed by an explanation illustrated with straightforward and reproducible examples. The material used in these examples (data and code) as well as the results (output and graphics) are included sequentially as the concepts are explained. All figures include a brief explanation to enhance the understanding of the interpretation. The last section of each chapter includes a summary and references useful to extend the chapter contents. Finally, some exercises are proposed. The reader is invited to apply the chapter contents on the helicopter case study presented in the first chapter. It is a common example used in quality courses and literature.

A brief guidance is given, leaving freedom to the reader to deal with it in depth at his/her convenience. Finally, we propose some exercises with solutions at the end of the book.

Though the book may be read from beginning to end for a straightforward comprehension of the DMAIC cycle, it can be used also as a reference manual. You can go directly to a specific chapter to learn (or remember) how to perform certain tasks. Thus, you may encounter the same topic explained in more than one way within the book.

We are aware that the book's contents are only a part of the large number of existing tools, methods, models, and approaches. It was not our intention to write the "Bible of Six Sigma with R" as this would deserve several volumes. The book paves the way to encouraging readers to delve into Six Sigma and R in depth, perhaps igniting an enthusiasm for both topics that is as strong in the readers as it is in the authors. To this end, a number of references are provided in each chapter. With the background acquired in the R system and the references provided (Internet-based articles and books), it will be easy for Six Sigma practitioners to extend the use of R to other tools not treated in the book.

Conventions

We use a homogeneous typeset throughout the book so that elements can be easily identified by the reader. Text in a sans serif font relates to software (e.g., R, Minitab). Text in `teletype` font within paragraphs is used for R components (packages, functions, arguments, objects, commands, variables, etc.).

The commands and scripts are also in `teletype` font, preceded by a ">" symbol and enclosed in a gray box with a left darker line:

```
> #This is an input code example
> my.var <- rnorm(10, 2, 0.5)
> summary(my.var)
```

The outputs appear just below the command that produces it, enclosed in a gray box (lighter than the input) and a thin border:

```
   Min.  1st Qu.  Median    Mean  3rd Qu.    Max.
 0.8231  1.6790  2.1000  1.9640  2.3820  2.5600
```

The book contains quite a few examples. They start with the string *Example (Brief title for the example).* and end with a square (□) below the last line of the example. In the subsequent continuation of the examples within the chapters, the string *(cont.)* is added to the example title.

In the book we will refer to a *Black Belt* as a person in charge of a Six Sigma project. Black Belt and other roles are defined in Chap. 1. Perhaps there will be tasks in the book (especially in the examples) that are not carried out by a Black

Belt; however, we will always use this name for the sake of simplicity. Similarly, what we say about products will very often be suitable for services, and we use in a general manner the term *customer* when referring to customers or clients.

Production

The book was written in `.Rnw` files. The Eclipse IDE (Integrated Development Environment) with the plug-in StatET was used as both editor and interface with R. If you have a different version of R or an updated version of the packages, you might not obtain exactly the same outputs. The session info of the machine where the code has been run is as follows:

- R version 2.14.1 (2011-12-22), `i686-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=es_ES.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `LC_MONETARY=es_ES.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=es_ES.UTF-8`, `LC_NAME=es_ES.UTF-8`, `LC_ADDRESS=es_ES.UTF-8`, `LC_TELEPHONE=es_ES.UTF-8`, `LC_MEASUREMENT=es_ES.UTF-8`, `LC_IDENTIFICATION=es_ES.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, utils
- Other packages: ggplot2 0.8.9, lattice 0.20-0, MASS 7.3-16, nortest 1.0, plyr 1.7, proto 0.3-9.2, qcc 2.2, qualityTools 1.50, reshape 0.8.4, rj 1.0.3-7, SixSigma 0.5.0, xtable 1.6-0
- Loaded via a namespace (and not attached): rj.gd 1.0.3-3, tools 2.14.1

Resources

The code and figures included in the book are available at the book's Web site: http://www.SixSigmaWithR.com. The `SixSigma` package and the data sets can also be downloaded. Links and materials will be updated regularly.

About the Authors

Emilio L. Cano is Adjunct Lecturer in the Department of Mathematics at the University of Castilla-La Mancha and Research Assistant Professor in the Department of Statistics and Operations Research at Universidad Rey Juan Carlos. He has more than 14 years of experience in the private sector as statistician.

Javier M. Moguerza is Associate Professor in Statistics and Operations Research at Universidad Rey Juan Carlos. He publishes mainly in the fields of mathematical programming and machine learning. Currently, he is leading national and international research ICT projects funded by public and private organizations. He has held membership in the Global Young Academy since 2010.

Andres Redchuk is Master Black Belt and Research Assistant Professor in the Department of Statistics and Operations Research at Universidad Rey Juan Carlos. He has worked for more than 18 years as a senior consultant in the private sector.

Madrid, Spain

Emilio L. Cano
Javier M. Moguerza
Andres Redchuk

Acknowledgements

We wish to thank David Rios Insua for his kind foreword and for the time he devoted to reading the manuscript. Mariano Prieto also provided valuable comments on our work, and his publications and courses applying Six Sigma have been an inspiration for us. We appreciate the gentle review of Virgilio Gomez-Rubio as an experienced R author and contributor. Our colleague Javier Cano has generously supported us with some book production issues. We thank the Springer staff (Kurt Hornik, Mark Strauss, Hannah Bracken), who from the beginning have lent their enthusiastic support to our proposal and have seen the project through to completion. A debt of gratitude must be paid to R contributors, particularly to the R core group (http://www.r-project.org/contributors.html), for their huge work in developing and maintaining the R project. Most of the quotes included throughout the book can be found at http://www.wikiquote.org/. We also acknowledge projects VRTUOSI (LLP, code 502869-LLP-1-2009-ES-ERASMUS-EVC) and RIESGOS-CM (code S2009/ESP-1685), in which the methodology described in this book has been applied.

Last but not least, we will be eternally grateful to our families for their patience and for forgiving us for the stolen time. Thank you Alicia, Angela, Erika, Lucia, Pablo, Rita, Sonia, Sophie, and Vicky.

Contents

List of Figures

List of Tables

Acronyms

6 Ms	Six “ems”: Machines, Methods, Materials, Measurements, Mother Nature (environment), Manpower (people).
ANCOVA	Analysis of Covariance
ANOVA	Analysis of Variance
BB	Black Belt (role)
cdf	cumulative distribution function
CI	Confidence Interval
CL	Center Line
COQ	Cost of Quality
Cp	Process Capability Index
Cpk	Adjusted Process Capability Index
CRAN	Comprehensive R Archive Network
CTQ	Critical to Quality
DMAIC	Define, Measure, Analyze, Improve, Control
DFSS	Design for Six Sigma
DMADOV	Define, Measure, Analyze, Design, Optimize, Verify
DMADV	Define, Measure, Analyze, Design, Verify
DMEDI	Define, Measure, Explore, Develop, Implement
DoE	Design of Experiments
DPMO	Defects Per Million Opportunities
DPPM	Defective Parts Per Million
DPO	Defects Per Opportunity
DPU	Defects per Unit
EWMA	Exponentially Weighted Moving Average
FMECA	Failure Mode, Effect and Criticality Analysis
FTY	First Time Yield
GAM	Generalized Additive Model
GB	Green Belt (role)
GLM	Generalized Linear Model
GNU	GNU’s Not Unix! (a form of licencing software)
GUI	Graphical User Interface

HTML	HyperText Markup Language
IDOV	Identify, Design, Optimize, Validate
IQR	Interquartile Range
ISO	International Organization for Standardization
IT	Information Technologies
LCL	Lower Control Limit
LSL	Lower Specification Limits
LT	Long Term
MANOVA	Multivariate Analysis of Variance
MBB	Master Black Belt (role)
MDI	Multiple Document Interface
MSA	Measurement System Analysis
MSD	Mean Squared Deviation
NaN	Not a Number
NA	Not Available
NN	Neural Network
OC	Operating Characteristic
ODBC	Open DataBase Connection
PLS	Partial Least Squares
Q–Q	Quantile–Quantile
QC	Quality Control
R&R	Repeatability & Reproducibility
RNG	Random Number Generation
RPN	Risk Priority Number
RFT	Right First Time
RTY	Rolled Throughput Yield
RUMBA	Reasonable, Understandable, Measurable, Believable, Achievable
SD	Standard Deviation
SDI	Simple Document Interface
SDMAIC	Select, Define, Measure, Analyze, Improve, Control
SIPOC	Suppliers, Inputs, Process, Outputs, Customers
SMART	Specific, Measurable, Achievable, Relevant, Timed (SMART objectives)
SPC	Statistical Process Control
ST	Short Term
SVM	Support Vector Machine
UCL	Upper Control Limit
USL	Upper Specification Limit
VOC	Voice of the Customer
VOP	Voice of the Process
VSM	Value Stream Mapping
Z	Sigma score

Part I
Basics

The first part of the book aims at giving the reader the necessary background to understand both R and *Six Sigma*. The book is intended to cover a wide range of readers, from beginners to experts. If you are in the latter group, you may go directly to Part II.

Part I contains two chapters: *Six Sigma in a Nutshell* and *R from the Beginning*. The former is a brief and concise introduction to the Six Sigma methodology. The latter explains how to deploy the R system for newcomers and covers basic usage of the R language and software.

Though these two chapters include material sufficient for reading this book, more advanced references can be consulted in the *Summary and Further Reading* sections.

Chapter 1
Six Sigma in a Nutshell

Science is organised knowledge.
Herbert Spencer

1.1 Introduction

Many total quality management methodologies have been introduced in recent decades, and Six Sigma has emerged as a breakthrough methodology. Essentially, the Six Sigma methodology is a quality paradigm that translates the involved scientific methodology into a simple way to apply the scientific method within every organization. In fact, according to the International Organization for Standardization (ISO), "Six Sigma speaks the language of business" [44].

The basis of the Six Sigma methodology is the DMAIC cycle. It consists of five stages: define, measure, analyze, improve, and control.

Another important issue in the Six Sigma methodology is the strongly defined roles within the organization. Using martial arts comparisons, the people involved in Six Sigma projects are classified into champions, master black belts, black belts, green belts, and even yellow belts, all of them engaged with the Six Sigma philosophy.

This chapter provides the necessary background to understand Six Sigma as a management philosophy. It gives a review of the history of Six Sigma in Sect. 1.2. An overview of the Six Sigma methodology is explained in Sect. 1.3. The two main concepts in Six Sigma are tackled in Sects. 1.4 and 1.5: the DMAIC cycle and organization roles, respectively.

1.2 Brief History

The beginning of the Six Sigma methodology dates to the mid-1980s. At that time, Mikel Harry, known as the "godfather" of Six Sigma, was working for Motorola.

Together with Bill Smith, he developed a methodology for the solution of problems following a disciplined approach. Based on this methodology, on January 15, 1987, Motorola launched a quality program called “The Six Sigma Quality Program.”

But the methodology achieved its highest success in the mid-1990s, when Jack Welch, Chairman and CEO of General Electric, adopted it as the main business strategy for that company. It was at that moment that Six Sigma became a management philosophy based on scientific decision making.

1.3 What Is Six Sigma?

Many definitions of the Six Sigma methodology can be found in the literature. In our opinion, essentially, the Six Sigma methodology consists of the application of the scientific method to process improvement. Therefore, it seems convenient to clarify what we mean by process. A process can be referred to as a group of interdependent actions directed toward reaching some goal or end. These actions may correspond to smaller processes. Six Sigma can also be used for the creation of new processes. In this case, because there is no current process to measure, a set of tools known as “Design for Six Sigma” can be used.

Thus, when we use Six Sigma for the improvement or creation of a new process, we apply the scientific method to obtain high-quality processes. The reason for this is clear: high-quality processes automatically lead to high-quality final outputs, usually referred as “products.” A simple example will illustrate what we mean by a high-quality process.

Imagine the administrative process of automatically generating certificates within a given organization. As a part of this process, a database exists with the first and last names of the people who might ask for a certificate, for instance, employees. Imagine that your last name, for instance, Smith, is wrongly stored in the database as, say, Smoth.

This means that whenever you ask for a certificate, your name will appear in its misspelled form. Of course, if you file a complaint, the certificate will be corrected. The simplest way to do this would be as follows: once the certificate has been generated, a secretary can open it with a word processor and change your last name from Smoth to Smith. In this way, the certificate (the final output) will be corrected. However, the next time you ask for a certificate, your last name will again be misspelled.

A more sophisticated option is to generate an automatic procedure that, once a complaint about a name is received, sends an alert reporting the problem. Automatically, the database administrator will correct the wrong last name in the database, and the certificate will be generated again. In this way, the process will be improved and the outputs arising from this process will have the desired quality.

In both cases, the final product is good, but in the second case, future products, future certificates for the same person, will also be good thanks to the process improvement and the fact that we now have a higher quality process. To obtain the second solution of the previous problem, the sophisticated one, we have implicitly taken the following steps:

1. Ask a question (why is the certificate wrong?).
2. Do some background research (names come from a database).
3. Construct a hypothesis (if names are correct in the database, then names will be correct in the certificates).
4. Test the hypothesis with an experiment (generate the certificate a second time).

Two additional steps would be helpful:

5. Analyze the data and draw conclusions (for instance, check that previous errors with names have not been corrected in the database, which is why users keep complaining).
6. Communicate the results (make a report describing the new procedure to address complaints).

These steps, which seem a natural way of thinking, correspond exactly to the steps of the scientific method and, as we will see in what follows, are the key points of the Six Sigma methodology. Within the Six Sigma lexicon, these steps are reduced to a five-stage cycle: define, measure, analyze, improve, and control, also known as the DMAIC cycle.

A new question now arises: What is so new about Six Sigma? The answer is simple: Six Sigma translates scientific language into an understandable way to apply the scientific method at any organization. This is the key to Six Sigma success. We are solving problems and improving processes using as our basis one of the most powerful tools of human development: the scientific method.

Although in this book we will focus on the use of the R statistical software package within Six Sigma, it is important to remark that Six Sigma is far from being just a set of statistical tools. Statistical methods are scientific tools useful for analyzing data and, therefore, play a very important role within the Six Sigma methodology. But, as you can deduce from the previous paragraphs, Six Sigma is more than that: it is a complete methodology for process improvement and, hence, for the success of organizations.

We do not focus on manufacturing processes. This methodology is general enough to be applied to any process (e.g., administrative, academic, scientific) inasmuch as "cost" and "quality" are common final goals of any given improvement initiative.

1.4 DMAIC Cycle

When Six Sigma is implemented, it must be applied on well-defined projects. Once the projects or processes have been chosen, the strategy to solve them follows the above-mentioned DMAIC cycle. This is the basis of the Six Sigma methodology. An improvement group (team) must be created to allow process improvements. This issue will be discussed in Sect. 1.5. Firstly, we give a brief description of all the stages of the cycle.

1.4.1 Define Phase

With the Define phase, the journey to the problem at hand begins. Initially it must be determined if the Six Sigma methodology is suitable for the solution of the problem. The key deliverable for this phase of the DMAIC cycle is the project charter. A project charter is a statement of the scope, objectives, and participants in a project. It provides a delineation of roles and responsibilities, outlines the project objectives, identifies the main stakeholders, and defines the authority of the project.

Important aspects of the project charter are the business case (that is, a brief description of the business problem), the problem statement, the goal statement, and the project scope. Also in the define phase, the team develops a list of critical to quality (CTQ) characteristics.

1.4.2 Measure Phase

The Measure phase is the second DMAIC stage. The objective of this phase is to glean from the current process as much information as possible. It must be accurately determined how the process operates.

The key tasks in the Measure phase are the creation of a detailed process map, the collection of baseline data, and, finally, a summary of the collected data. In most projects, a basic process map is developed in the Define phase.

The process map provides a visual representation of the process under investigation. It can also provide additional awareness of process inefficiencies, such as cycle times or bottlenecks, or identify process requirements that do not add value to the process. The process map may also give new information about data collection.

1.4.3 Analyze Phase

The third phase of the DMAIC process is the Analyze phase, where the team sets out to identify the root causes of the problem under study. Unlike with other simpler problem-solving strategies, within the Six Sigma methodology the root causes must be validated by data, leading to what we call “fact-based decisions.”

The process map, the collected data, and any other knowledge accumulated during the Define and Measure phases should be used to determine the root causes.

The power of the Analyze phase is provided by the statistical analysis that is conducted. This higher level of analysis sets Six Sigma apart from other problem-solving strategies. The statistical techniques commonly used to validate potential root causes include, among others, analysis of variance (ANOVA), correlation analysis, scatterplot, or chi-square analysis.

1.4.4 Improve Phase

The objective of the Improve phase is to determine a solution to the problem at hand. Brainstorming is commonly used to generate a set of potential solutions. It is important in this phase to involve people who will perform the process regularly. Their input can be invaluable. In some cases, they even provide the best potential solution ideas because of their process knowledge. In other words, the combination of experience and scientific analysis is a guarantee of success.

In addition, you must keep in mind that the term "best" does not mean the same thing to all people. What the team should strive to find is the best overall solution. A solution criteria list is another good tool to assist in selecting the best solution.

Prior to implementation, the team must be sure that the proposed solution actually works. Pilot programs, computer simulations, and segmented implementation are all possibilities at this point.

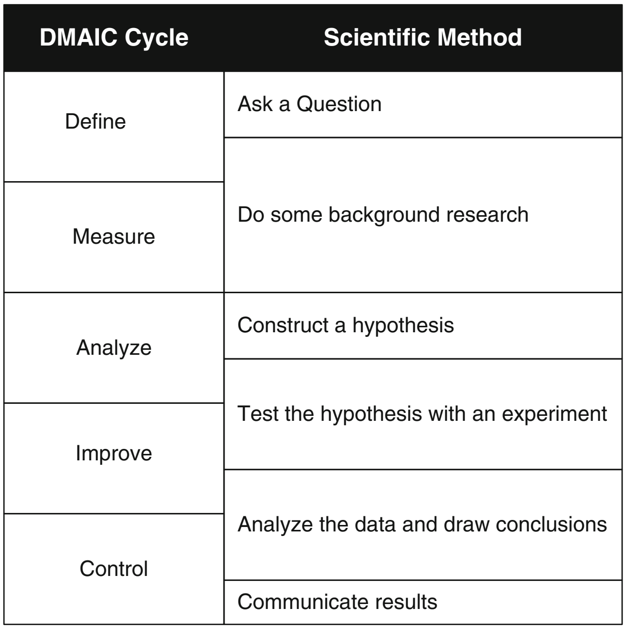
The team should also create a future state process map as part of the Improve phase. This must be done so that the process can be performed as many times as necessary to ensure that the correct implementation of the solution is accomplished.

1.4.5 Control Phase

The final phase is the Control phase; its objective, simply put, is to sustain the gains that were achieved as a result of the Improve phase. A plan detailing the steps to be taken during the Control phase should be developed and any new potential ideas discussed. The idea of control in Six Sigma differs from traditional operations. The way of assuring quality in the CTQ characteristics is through the control of key input variables, which differs from the traditional (and usually non-value-adding) final inspection procedure.

Once success is achieved, a celebration may take place. The scale of the celebration is up to each individual organization, but to create a sustainable improvement environment, at the very least, the efforts of the various participants should be recognized.

To finish with this outline of the DMAIC cycle, we show in Fig. 1.1 its connection with the scientific method. Figure 1.2 demonstrates the interactions among the different phases.



DMAIC Cycle	Scientific Method
Define	Ask a Question
Measure	Do some background research
Analyze	Construct a hypothesis
Improve	Test the hypothesis with an experiment
Control	Analyze the data and draw conclusions
	Communicate results

Fig. 1.1 Relationship between DMAIC cycle and scientific method. There is a sort of correspondence between the phases of the DMAIC cycle and the steps in the scientific method

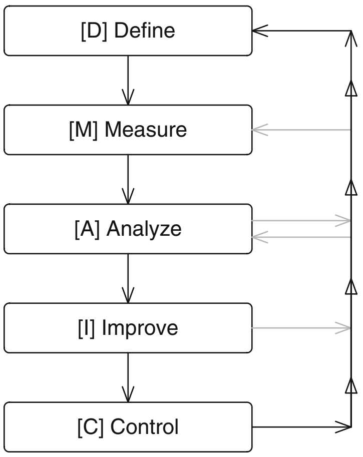


Fig. 1.2 Roadmap of the DMAIC cycle. Interaction among phases of cycle. There is feedback between the distinct phases

1.5 Six Sigma Operational Structure

Six Sigma does not require special changes in the hierarchical structure of an organization; it only requires operational adjustments. The typical Six Sigma work structure is given by the following roles: champion, master black belt (MBB), black belt (BB), process owner, and green belt (GB). This terminology has its origin in Mikel Harry's interest in martial arts.

The Champion is the senior most person responsible for an area of an organization, usually the person in charge of a department. The main areas of action must have a Champion. These areas include human resources, engineering, finance, etc. She will lead the improvement program within the area she is responsible for and, therefore, will select the projects with the best improvement opportunities. She must coordinate her work with the other Champions in the organization.

The Process Owner is the person in charge of a concrete project. He will report directly to the Champion. As the link to those responsible for the global improvement strategy, and in particular Champion, he is tasked with controlling the process he is in charge of. If possible, the Process Owner must be the person with the highest knowledge of the project or process at hand.

The MBB must possess a thorough knowledge of statistical techniques, planning techniques, and management. Hierarchically, he will report directly to the organization's management board. He will apprise the management board on the general strategy of the improvement program and the Champions on the selection of the projects to improve. Moreover, he will be responsible for the Six Sigma training program. The MBB must be a respected person within the organization, and his dedication to the improvement program should be full time.

The BB will be responsible for the improvement group created by the Champion within a given project. He should be sufficiently autonomous to lead a project. In addition, the BB must have a good knowledge of statistical techniques (one level below that of the MBB), leaving the specific technical details to the MBB. He will also see to it that the DMAIC cycle is followed with discipline.

The GB is someone with elementary statistical knowledge who occasionally may lead a simple improvement project. GB is the ideal level for Project Owners and intermediate managers.

The hierarchical relation between these roles is described in Figs. 1.3 and 1.4.

Notice that this operational structure is equivalent to an internal consultancy within the organization. Figures 1.3 and 1.4 could be merged into a single figure, but for the sake of space, we divided the structure into these two figures, which correspond to the strategic and the operational levels, respectively.

1.6 Summary and Further Reading

This chapter presented an overview of the Six Sigma methodology from its origin. Following a brief history of the methodology, we introduced the focal point behind it: the DMAIC cycle, which serves as the book's roadmap. Finally, we outlined our approach to the Six Sigma roles in an organization. The case study used throughout the book is explained below. We recommend practicing while you read the book to reinforce what you have learned.

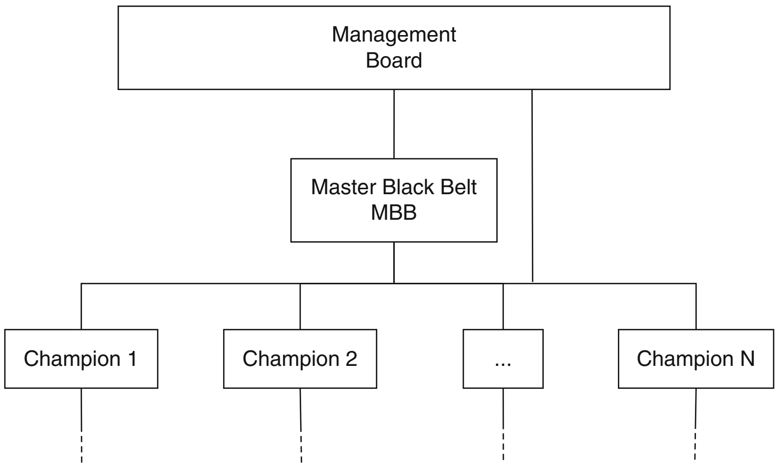


Fig. 1.3 Strategic Six Sigma roles. The Champion is the link between the strategic and operational roles. He creates the improvement groups

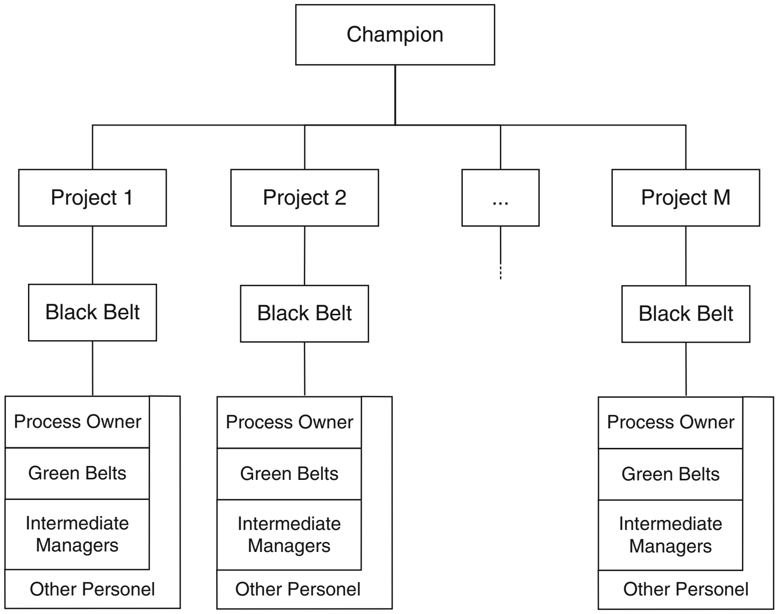


Fig. 1.4 Operational Six Sigma roles. The Black Belt is the core of the Six Sigma structure; process owners are crucial for improvement

Regarding the literature on Six Sigma, it is too large to be exhaustively cited here. In the book, many references will be cited, including many that report success stories using the Six Sigma methodology. A seminal work showing Mikel Harry's point of view on Six Sigma and its success is [36]. A more recent work is [97].

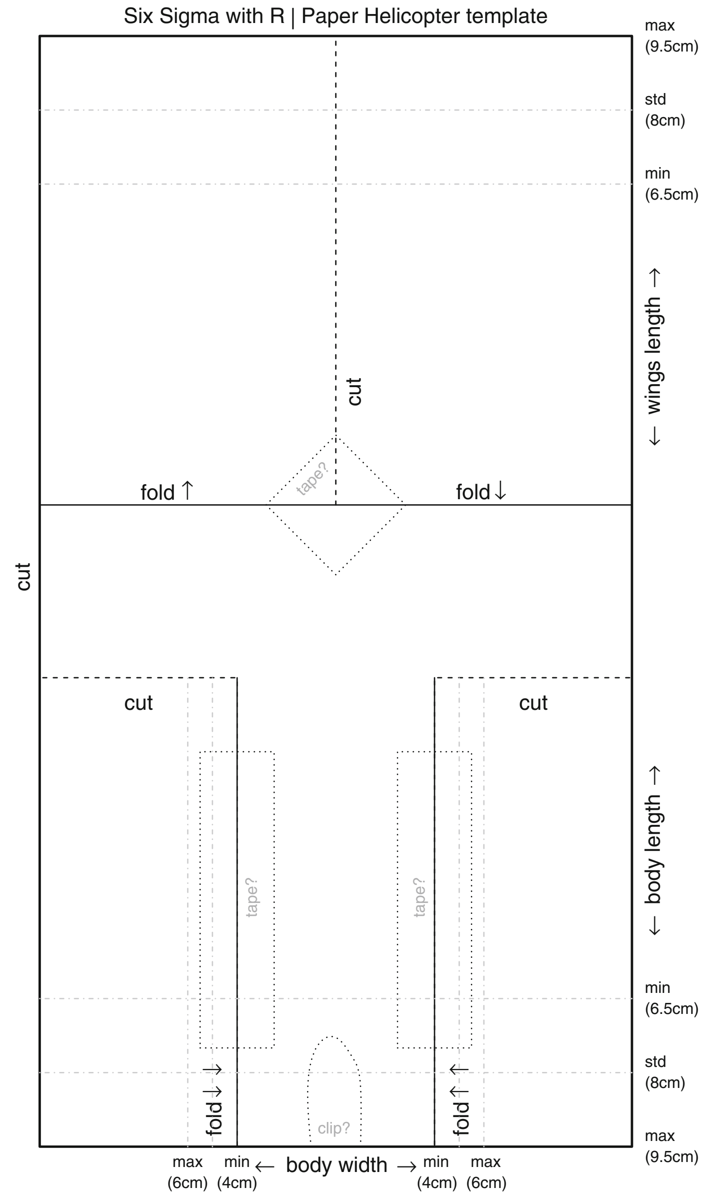


Fig. 1.5 Paper helicopter template. Designed to be printed on an A4 sheet. Cut and fold according to lines and arrows. Add paper clip or adhesive tape to make different prototypes

For more details on the evolution of Six Sigma, we recommend the books [76] and [82]. Specific books on Six Sigma applied to different scenarios such as product design or small and medium enterprises are [4, 13, 65, 81, 107].

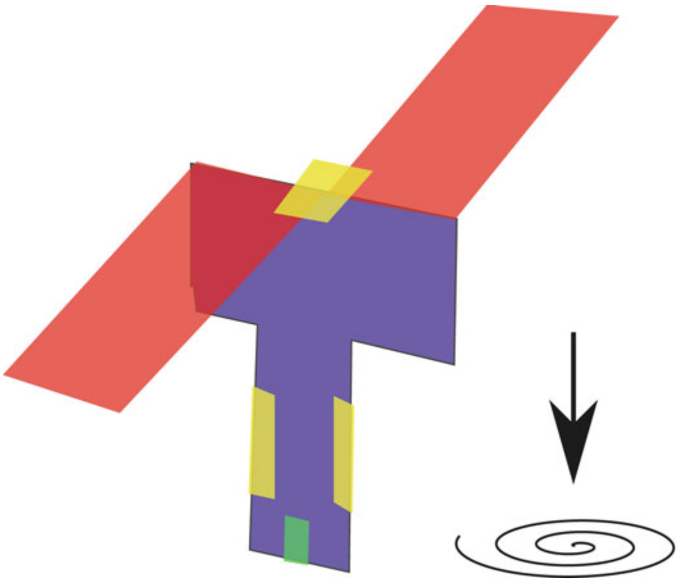


Fig. 1.6 Paper helicopter mounting scheme. The mounted paper helicopter should have this shape. When descending from a given height, it twists around until it reaches the floor

For references describing and discussing the scientific method, we refer the interested reader to the classic works [79] and [80].

Case Study

Throughout the book we present a case study widely used in the quality literature. It first appeared in a paper [9] using an original idea of Kip Rogers of Digital Equipment. It consists in the construction of a paper helicopter whose design can vary in its wings and body length, body width, the use of adhesive tape stuck to the structure, or the inclusion of a paper clip at the bottom.

A template to construct the paper helicopter can be obtained using the function `ss.heli` within the `SixSigma` package. The result is a PDF file in the working directory with the design shown in Fig. 1.5.

Figure 1.6 shows a scheme for mounting the paper helicopter. The light-gray strips are for adhesive tape, the strip at the bottom is for a paper clip. These materials are optional.

There is a vignette in the Six Sigma package with the mounting instructions of the paper helicopter. Type `vignette("HelicopterInstructions")` to open the PDF.

Throughout the book we will ask you to describe processes, take measurements, analyze data, improve processes, and control processes related to the paper helicopter.

The response characteristic of our product is the flight time. The flying test is performed by dropping the helicopter in the vertical position from a height of 2 m and measuring the time it takes to reach the floor. This test is performed with the aid of a stopwatch, capable of measuring 1/100 s.

Retrieve the template and mount some paper helicopters. Practice measuring the flight time.

Chapter 2
R from the Beginning

Software is like sex; it's better when it's free.

Linus Torvalds

2.1 Introduction

R is a system for statistical computing and graphing. It consists of a language and a software environment. R has been widely used for academic and research purposes and is increasingly being deployed in corporate environments. As an example, companies such as Google and Pfizer have been using R for a long (in technological terms) time.[1] R has its origin in the S language, developed at Bell Laboratories (see [102]), and it was initially written by Ross Ihaka and Robert Gentleman in the Department of Statistics of the University of Auckland, New Zealand [84]. It is a freely available software, under a GNU license, and is supported by the R Development Core Team. A major strength of R is its extensibility, through the packages developed by the community of R users, as the `SixSigma` package we have developed, available through the CRAN repository, where support is also given. Furthermore, it is available for a wide range of platforms, including Windows, Mac, and Linux.

As far as the language is concerned, it is interpreted language easy for nonprogrammers to learn. Nevertheless, it has huge possibilities for information technology (IT) professionals, for instance, through its interaction with other languages such as C or Fortran.

[1]An article in the New York Times in January 2009 surprised many professionals and was a milestone in R's surging popularity (http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html).

Although many companies are still reluctant to use freely available software, the advantages are worth considering. Hundreds of people work every day to improve the system and share their experiences, problems, and solutions. Thanks to R's community of users and developers, issues related to R are resolved faster than are issues with paid software. Furthermore, more and more professionals and consulting companies provide technical support with R solutions. You simply pay for their services, not for the license.

In this chapter, we provide basic background information to accustom readers to R and to anticipate subsequent chapters. Following this brief introduction, the first steps are explained. Then we address data issues, functions, graphics, and statistics in successive sections. A wider collection of functions is listed in Appendix A as a short reference guide. Section 2.8 contains additional information helpful for advancing one's knowledge of R.

2.2 First Steps

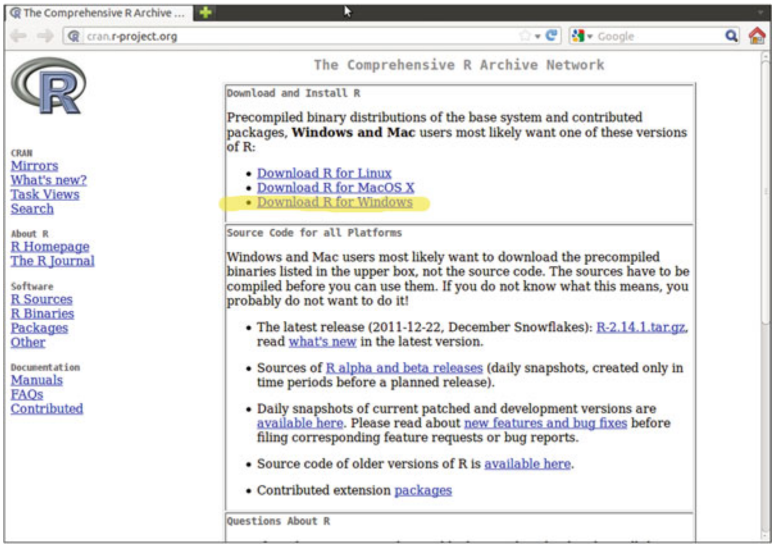
2.2.1 Get and Install

As was mentioned in the introduction, R is freely available software. There is some confusion about what this means. Indeed, several different concepts merge and there are several interpretations. In the case of R, free entails both free of charge and open source. The latter is even more important than the former, especially when trying to improve systems, such as in the Six Sigma methodology.

Due to its GNU license and the wide support provided by the community, obtaining R is easy. We simply have to download from CRAN the files corresponding to our operating system. The steps to follow are follows:

1. Go to http://cran.r-project.org (Fig. 2.1).
2. Select your operating system. (Henceforth we will assume a Windows system. If you want R for MacOS or Linux, select the correct link on the CRAN Web page and follow the instructions.) A complete manual for R installation and administration is available on the R Project Web site.[2]
3. Click on "`base`" (Fig. 2.2).
4. Click on `Download R x.xx.x` for Windows (Fig. 2.3).
5. Run the downloaded file and follow the wizard to install (Fig. 2.4).

[2]http://cran.r-project.org/doc/manuals/R-admin.html.


The Comprehensive R Archive ...

cran.r-project.org

The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for MacOS X
- Download R for Windows

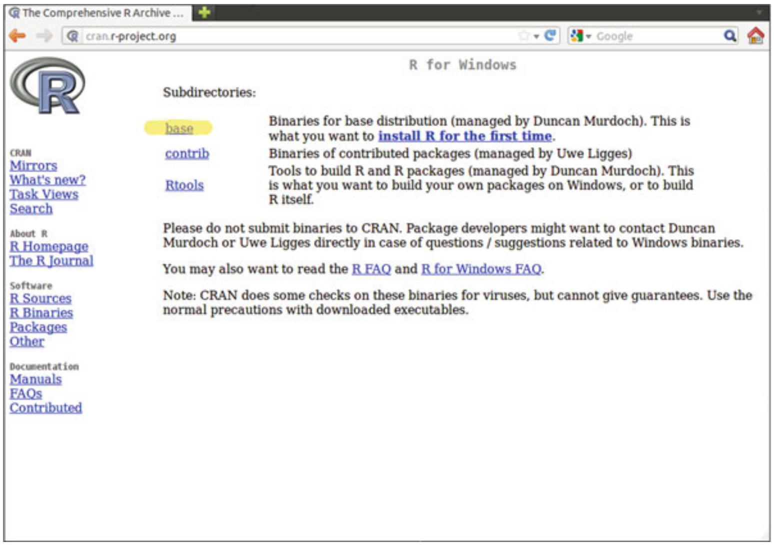
Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2011-12-22, December Snowflakes): R-2.14.1.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

Fig. 2.1 CRAN Web page. Select the download link suitable for your operating system. A new page is opened


The Comprehensive R Archive ...

cran.r-project.org

R for Windows

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Subdirectories:

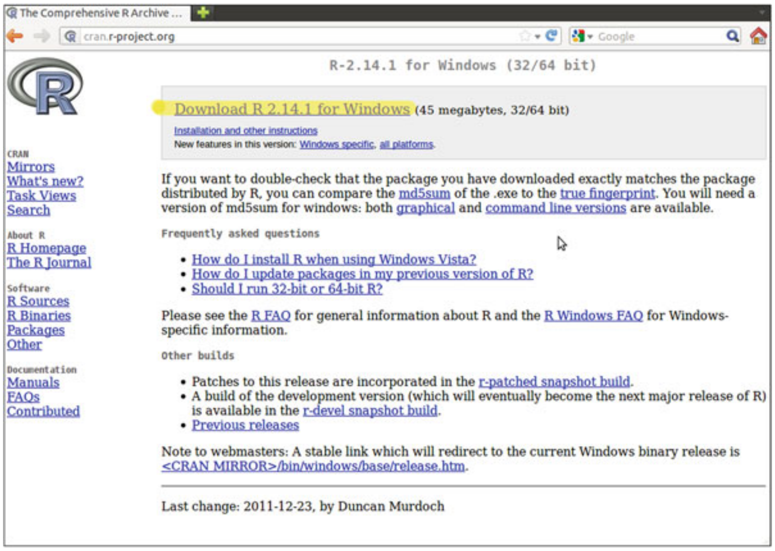
base	Binaries for base distribution (managed by Duncan Murdoch). This is what you want to **install R for the first time**.
contrib	Binaries of contributed packages (managed by Uwe Ligges)
Rtools	Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch or Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Fig. 2.2 CRAN Web page. Selection of subdirectory. Select “base” to download and install the base installation. Links for contributed packages and building tools are also available



R-2.14.1 for Windows (32/64 bit)

Download R 2.14.1 for Windows (45 megabytes, 32/64 bit)

Installation and other instructions

New features in this version: Windows specific, all platforms.

If you want to double-check that the package you have downloaded exactly matches the package distributed by R, you can compare the md5sum of the .exe to the true fingerprint. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- How do I install R when using Windows Vista?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

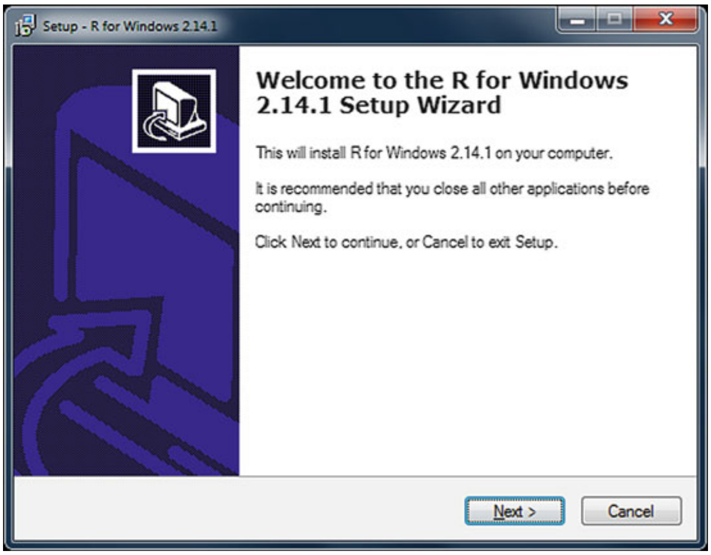
Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.htm.

Last change: 2011-12-23, by Duncan Murdoch

Fig. 2.3 CRAN Web page. Download link for Windows. There may be a newer version by the time this book is published



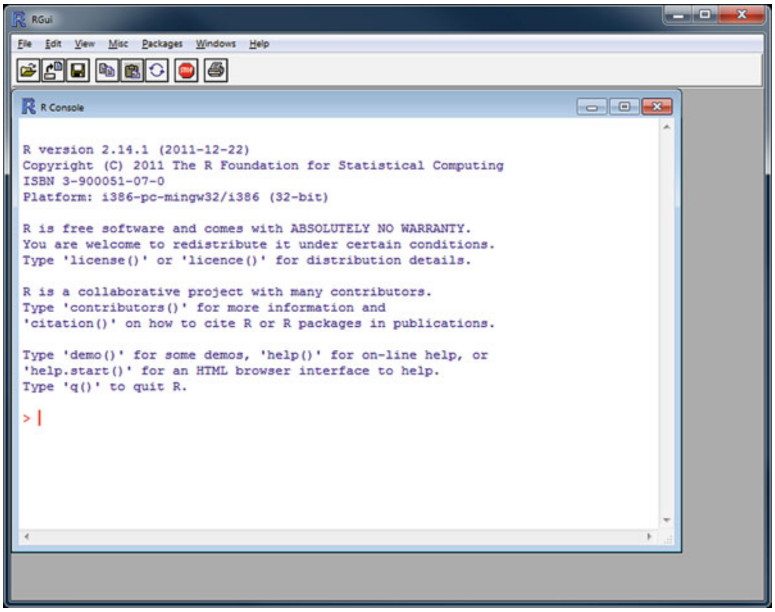
Welcome to the R for Windows 2.14.1 Setup Wizard

This will install R for Windows 2.14.1 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Fig. 2.4 R installation wizard. A welcome message is shown at the beginning. Click “Next” to proceed



```
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Fig. 2.5 R's graphical user interface. The R Console is the window where you interact with R. Menu and command bars are available

2.2.2 Run and Interact

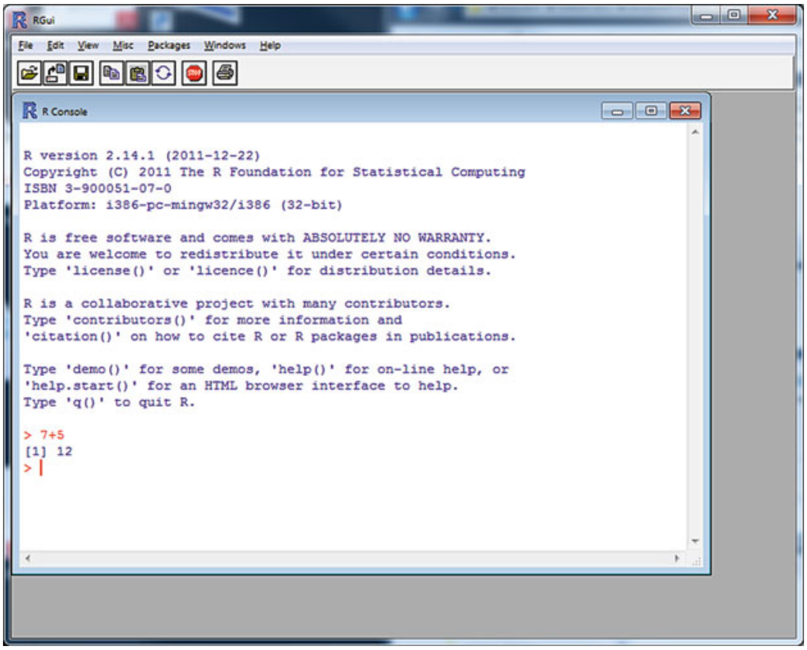
Once the software is installed, you will have an icon on your desktop. Double click it to get the R graphical user interface (GUI) shown in Fig. 2.5.[3] Although the GUI has graphical elements like any Windows program (menu bar, tool icons, windows), the main window to interact with is the R Console.[4]

Commands

When we start R, some messages are printed in the R Console, and then the cursor blinks next to the prompt symbol (>), waiting for a command. A command is an

[3]If you choose SDI (simple document interface) in the custom installation, you only get the R Console with the menu bar. You can run the SDI or the MDI (multiple document interface) by adding the option `--sdi` or `--mdi`, respectively, to the command line in the shortcut icon properties, e.g., `C:\R\R-2.14.1\bin\i386\Rgui.exe -sdi`.

[4]There are some easy-to-use graphical alternatives to some R functions (Sect. 2.8). They can be useful when migrating from other systems to R, but we recommend using the R Console and scripting facilities as much as possible to exploit R's possibilities.



```
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 7+5
[1] 12
>
```

Fig. 2.6 R text output. A basic text output for an arithmetic operation. It appears after the `INTRO` key is pressed

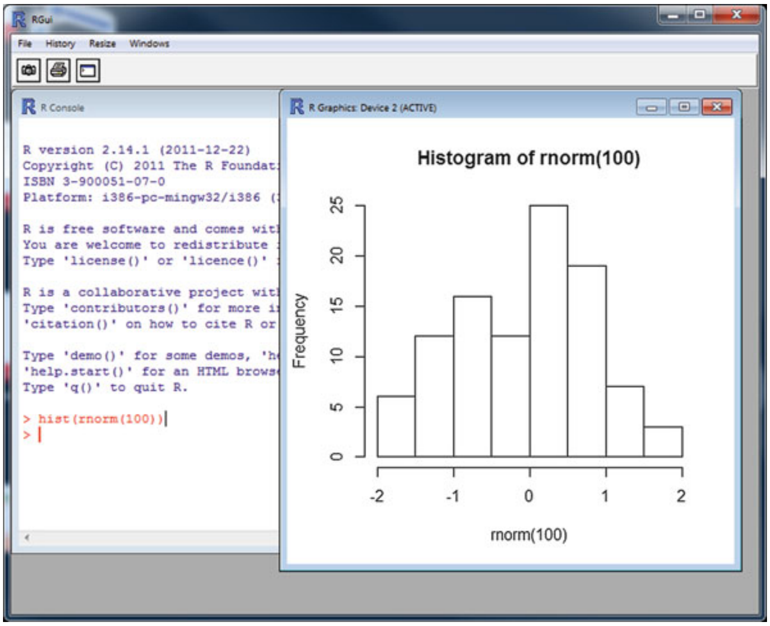
Table 2.1 R Console shortcuts

Shortcut	Result
`Arrow up/down`	Navigate through command history
`CTRL+L`	Clean console
`ESC`	Stop current run
`TAB`	Complete function name

expression that will be evaluated by R when the `INTRO` key is pressed.[5] Once the command is evaluated, an output may or may not be obtained. The output will be text, a graphic, or both. The text output will be displayed below the command just run. For example, if you type a very simple arithmetic operation as a command, you will obtain the result as output. Type `7+5` at the prompt symbol, and press `INTRO`. You should see the result shown in Fig. 2.6. The length of the output varies depending on the command. You can scroll up and down with the vertical scroll bar to see the whole output.

Table 2.1 presents some useful shortcuts using the R Console.

[5]When `INTRO` is pressed before a command is completed (for example, if a closing bracket '`)`' is expected), then the prompt symbol changes to `+`. This is sometimes annoying when learning R and usually indicates a mistake. Simply press the `Esc` key to return to the prompt symbol.



```
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundat
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (

R is free software and comes wit
You are welcome to redistribute
Type 'license()' or 'licence()'

R is a collaborative project wit
Type 'contributors()' for more i
'citation()' on how to cite R or

Type 'demo()' for some demos, 'h
'help.start()' for an HTML brows
Type 'q()' to quit R.

> hist(rnorm(100))
>
```

Fig. 2.7 R graphic output A new window is opened when a graphic output is generated by R

The graphic output opens a new graphic device (usually a window inside the R GUI) displaying the plot you asked for. Type `hist(rnorm(100))`.[6] This command makes a histogram (Chap. 8) with the data of a 100-sized random sample from a standard normal random variable (Chap. 9) and outputs the graphic in an R Graphics window inside the R GUI. Figure 2.7 shows what you must obtain.[7]

The last command is slightly different from the first one. We have made a call to a *function* with an argument. This argument is in brackets, and it can be any R *object*. Do not run away; we will explain how to use functions and objects in Sect. 2.4.

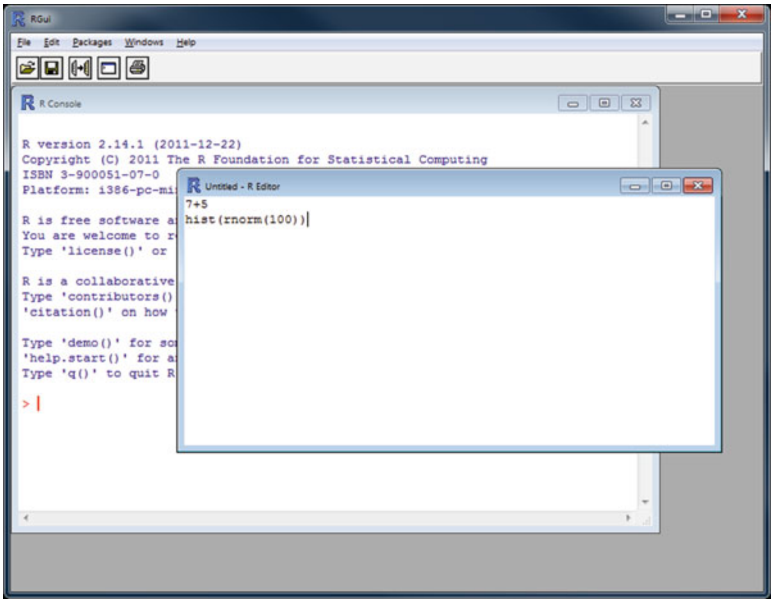
The image can be copied and pasted into another application such as a text editor (Microsoft Word, OpenOffice, ...). You can also save it on your computer in a wide range of formats or even print it.

Scripting

Running commands in the R Console is the fastest way to obtain a result, but the most efficient way to use R is through scripts. A script is a sequence of commands in a text file that can be run in R. You can run all the commands in a script, a group of them, or just one.

[6]Do not worry about what it means for the moment, just type it.

[7]The shape of the histogram may be slightly different due to the randomness of the data.



```
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mi

R is free software a
You are welcome to r
Type 'license()' or

R is a collaborative
Type 'contributors()
'citation()' on how

Type 'demo()' for so
'help.start()' for a
Type 'q()' to quit R

>
```

```
7+5
hist(rnorm(100))
```

Fig. 2.8 R script editor. The commands executed from the R Editor are run in the console, eventually getting the text or output results

In Chap. 13, we detail some advanced editors for scripting. For the moment, we will use the basic editor provided by the R GUI. To create a script, navigate to `File` → `New Script`. The R editor is opened, and you can start typing commands. In Fig. 2.8 the two commands mentioned above are typed (type them yourself in your editor). You can perform commonly used operations with the text: copy, paste, select, find, undo, … .

To run a command in the script, set the cursor in the line you want to run and press `Ctrl+R`. You can select a group of lines or even the whole text and run it. You can also run all the script through the menu bar (`Edit` → `Run all`).

An important issue in R is the location where objects and files are stored. This is the *Working Directory*. The default working directory is the folder `My Documents`. You can see your default working directory with the command `getwd()`. You can change your working directory with the command `setwd("My directory")`, where `My directory` is the path to the folder you want to work in. You can also use the menu bar and navigate to `File` → `Change dir ...` .

We advise you to save scripts as often as possible. Otherwise, you may lose changes. Navigate to `File` → `Save` in the menu with the editor window active. The first time you do this with a script, you will be asked to enter a name for the file. Type a descriptive name for your script and the extension `.R`. By default, the Save script dialog points to the Working Directory, but you can navigate to any other.[8] Save the script, using the two commands previously mentioned, with the name `sixsigma.R`.

Now we can exit R and continue later. You can close the R GUI window using the mouse or type in the R Console the command `q()`. A dialog box asking `Save workspace image?` will appear. Select `No`. We will explain what the workspace is in Sect. 2.3.

When you resume your work in R, you can open the script to edit or run the commands. Run R again and open the script saved previously. Navigate to `File` → `Open Script...` and select the file named `sixsigma.R`. The script is then opened in the R editor. In this way, you can continue with your work at the same step where you stopped previously.

Another way to use a script is running it entirely without opening it. Just type `source("sixsigma.R")` in the R Console, and all the commands in the script file will be executed.

There are two special characters in a script. One of them is `#`. No line begun with the character `#` will be executed. This is very useful for commenting the script for future revisions or for simply omitting some commands temporarily. The other special character is `;`. It is used to separate commands in the same line and can be used either in the R Console or inside a script.

Installing Packages/Libraries

Once we have installed the base installation of R and know something about the GUI and the R Console, we can proceed to the use of the contributed *packages*. A package or library is an additional module for specific purposes. For example, the `SixSigma` package is a module that aims at carrying out Six Sigma projects. A package may contain functions, data, and documentation and can be developed by anyone. A package may be contributed to the project, making it available for others. You can build a package customizing the functionality of R to your needs.

Installing a new package is quite simple. If you know the name of the package (say `SixSigma`), just type what follows in the R Console:

```
install.packages("SixSigma", dependencies=TRUE).
```

The dialog box will ask for a mirror.[9] Select one close to your location; R does the rest. Some windows may appear indicating the installation progress until the package is finally installed.

To use an add-on package, you need to load it. Type `library(SixSigma)` to load the `SixSigma` package. Now you can use the functions and the data of the package. Type `example(SixSigma)` in the R Console to see a complete example of the `Six Sigma` package.

[8]The working directory will not change.

[9]A server on the Internet where you can download the package from.

These tasks can also be performed through the menu bar `Packages` and the suitable submenu.

2.2.3 Ask for Help

One of the strengths of R is the amount of documentation available, not to mention the specialized webs, blogs, and forums. In the `Help` menu of the R Console, you can find R manuals in PDF format and links to other documentation. There is also a set of functions to obtain information from the R Console.

As was mentioned above, a command is a function that may accept arguments. If you need help about a specific function, type `?thefunction`, where `thefunction` is the name of the function you need help with. The browser with the documentation on the function is opened, and you can see the syntax, arguments, value, and other information. If you cannot remember the exact name, use the function `apropos("fun")`, where `fun` is a portion of the name of the function. You get a list of functions that contain `fun` in their name. The command `??something`, where `something` is a text string, looks for help files containing this string in the alias, concept, or title. If you want to see an example of a function, type `example(namefun)`, where `namefun` is the name of the function. Practice with the functions of the `SixSigma` package, for example `example(ss.rr)`.

Some packages also have demos; type `demo("graphics")` to see a demonstration of the `graphics` topic. You can request a list of available demos by typing `demo()`.

Some authors include a special type of documentation for the package: vignettes. A vignette is a PDF document independent of the R documentation where the authors explain in detail the topic, usually including examples or methodological information. You can browse available vignettes in an Internet browser by typing `browseVignettes()` or open it directly if you know the topic, for example, `vignette("grid")`.

If you want to navigate through the full help functionality, type `help.start()`, and your default browser[10] will open the documentation in HTML format.

There is also information available online. On the R Project's Web site (http:\www.r-project.org) you can find a documentation section in the left sidebar, with links to Manuals, FAQs, The R Journal, Wiki, Books, Certification, and Other. Click on those links if you want to get involved in the "R philosophy." An interesting resource is the CRAN Task Views (http://cran.r-project.org/web/views/). In every Task View, packages, links and tools related to a specific topic are gathered and briefly explained.

Henceforth, we explain the principal functions used in R through simple examples and their output. Appendix A represents a basic reference for these functions.

[10]Though the browser opens, the documentation is in the computer, not on the Internet.

2.3 Coping with Data

2.3.1 Data Types

The simplest type of data we find in R is a single value, which can be a number, a character string, or a logical value (`TRUE`/`FALSE`). Data are saved in variables[11] using the assign function `(<-)`. For example, if we want to save the value 24 in the variable *machine.age*, we use the following command:

```
> machine.age <- 24
```

Table 2.2 presents some special data values. The data can be arranged in varied types of collections. The main data types are listed in Table 2.3. We will show examples of these kinds of objects in the following sections.

2.3.2 Creating Data Objects

The values of a vector may be assigned with the function `c` (combine):

```
> my.vector <- c(10, 20, 30, 40)
```

Another way to insert data into a vector is with the function `scan()`. The R Console accepts data until `INTRO` is pressed with no data. To invoke a data object, just type its name in the R Console:

```
> my.vector
```

Table 2.2 Special data values in R

Value	Meaning
`NA`	Not available
`NaN`	Not a number $\left(\text{e.g. } \frac{0}{0}\right)$
`Inf`	Infinite
`NULL`	No value

Table 2.3 Main R data types

Data Type	Description
`vector`	A collection of values of the same type
`matrix`	A structure of values with rows and columns of the same type
`array`	Same as matrix but with more than two dimensions
`list`	An ordered collection of objects (components) that may be of different types and lengths
`factor`	For classifying categorical data; has `levels` and `labels`
`data.frame`	A collection of variables and records (or observations)

[11] A temporary space to save information and assign a name.

```
[1] 10 20 30 40
```

To create a matrix , we use the function `matrix`:

```
> my.matrix <- matrix(c(10, 20, 30, 40, 12, 26, 34, 39),
    nrow = 4, ncol = 2)
> my.matrix
```

```
     [,1] [,2]
[1,]   10   12
[2,]   20   26
[3,]   30   34
[4,]   40   39
```

We can set names for columns and rows:

```
> colnames(my.matrix) <- c("myFirstCol", "mySecondCol")
> rownames(my.matrix) <- c("Case1", "Case2", "Case3",
    "Case4")
> my.matrix
```

```
      myFirstCol mySecondCol
Case1         10          12
Case2         20          26
Case3         30          34
Case4         40          39
```

Arrays are similar to matrices but with more dimensions. The equivalent arguments `dim` and `dimnames` may be used.

A list may contain different types of objects:

```
> my.list <- list(lvector = my.vector,
    lmatrix = my.matrix)
> my.list
```

```
$lvector
[1] 10 20 30 40

$lmatrix
      myFirstCol mySecondCol
Case1         10          12
Case2         20          26
Case3         30          34
Case4         40          39
```

Factors aim at creating levels of categorical data:

```
> my.factor <- factor(c(1,0,1,1),levels = c(0, 1),
    labels = c("Incorrect", "Correct"))
> my.factor
```

```
[1] Correct   Incorrect Correct   Correct
Levels: Incorrect Correct
```

Data frames are the best way to manage data in R. A data frame is a list of variables. Each variable contains data, of any type. All columns must have the same length. To create a data frame, we specify every column:

```
> my.data.frame <- data.frame(CTQ = my.vector,
    state = my.factor, aux = my.matrix)
> my.data.frame
```

```
      CTQ     state aux.myFirstCol aux.mySecondCol
Case1  10   Correct             10              12
Case2  20 Incorrect             20              26
Case3  30   Correct             30              34
Case4  40   Correct             40              39
```

We can add and remove columns (variables) from a data frame:[12]

```
> my.data.frame$machine <- as.factor(c(1, 1, 2, 2))
> my.data.frame$aux.myFirstCol <- NULL
> my.data.frame
```

```
      CTQ     state aux.mySecondCol machine
Case1  10   Correct              12       1
Case2  20 Incorrect              26       1
Case3  30   Correct              34       2
Case4  40   Correct              39       2
```

Sometimes we just want to know about the structure of a data frame. There are two very useful functions to do that. The function `str` prints the class of an object and its structure (number of observations and variables), and the data frame has as many lines as columns, showing the type and the first values of each column:

```
> str(my.data.frame)
```

```
`data.frame':   4 obs. of  4 variables:
 $ CTQ            : num  10 20 30 40
 $ state          : Factor w/ 2 levels "Incorrect","Correct":
      2 1 2 2
 $ aux.mySecondCol: num  12 26 34 39
 $ machine        : Factor w/ 2 levels "1","2": 1 1 2 2
```

The `head` function returns the first part of the object included as the first argument, corresponding to the number of rows indicated in the second argument (very useful when you have a very large data set):

```
> head(my.data.frame, 2)
```

```
      CTQ     state aux.mySecondCol machine
Case1  10   Correct              12       1
Case2  20 Incorrect              26       1
```

[12]See next section to find out what the $ symbol is for.

2.3.3 Accessing Data

We already know that by calling a data object, we get all its values. But what if we want a specific subset or just a single value? Some functions extract, summarize, or group data. We will see them in the next section. For now we will talk about indices and subscripts. A data object has a number of elements. Each element is identified by an index, that is, the position it occupies inside the object. If we want to get the third element in the `my.vector` object, we type:

```
> my.vector[3]
```

```
[1] 30
```

Thus, to filter the elements of a data object, we include a *subscript* between square brackets. Hence, a subscript is an expression indicating which elements of a data object will be returned. This subscript can be as complex as you want, so you get exactly what you need.

For an object with more than one dimension, the subscript may be cut through commas (`,`). If you do not do this, you will search in the whole object, not just in the row or column you need. To understand the difference, you need to realize what the length of a data object is. The following examples illustrate the difference between various objects:

```
> length(my.vector)
```

```
[1] 4
```

```
> length(my.matrix)
```

```
[1] 8
```

```
> dim(my.matrix)
```

```
[1] 4 2
```

```
> length(my.matrix[, 1])
```

```
[1] 4
```

```
> dim(my.data.frame)
```

```
[1] 4 4
```

```
> length(my.data.frame[1,])
```

```
[1] 4
```

For example, let us obtain the element in the third row, second column of our matrix:

```
> my.matrix[3, 2]
```

```
[1] 34
```

We can obtain the entire row or column by leaving blank the other dimension (an empty dimension means *get everything*). The following command retrieves "all the rows of the second column":

```
> my.matrix[, 2]
```

```
Case1 Case2 Case3 Case4
   12    26    34    39
```

The elements of a list can be extracted through subscripts or through their names using the operator `$`. The single square brackets return a list with one element.

```
> my.list[1]
```

```
$lvector
[1] 10 20 30 40
```

The double square brackets and the named item return an object whose class is that of the element extracted.

```
> my.list[[1]]
```

```
[1] 10 20 30 40
```

```
> my.list$lvector
```

```
[1] 10 20 30 40
```

Although both elements seem the same, the element obtained with the `my.list[1]` command is a list and should preferably not be used as a numeric vector, whereas the element obtained with the `my.list[[1]]` command is in fact a numeric vector and can be straightforwardly used within algebraic operations. You can realize the difference by practicing with the `class()` function (Sect. 2.4):

```
> class(my.list[1])
```

```
[1] "list"
```

```
> class(my.list[[1]])
```

```
[1] "numeric"
```

```
> class(my.list$lvector)
```

```
[1] "numeric"
```

In a data frame, we can also use both subscript strategies, with indices or with names:

```
> my.data.frame[, 2]
```

```
[1] Correct   Incorrect Correct   Correct
Levels: Incorrect Correct
```

```
> my.data.frame$machine
```

```
[1] 1 1 2 2
Levels: 1 2
```

Both strategies can be merged, and inside a subscript we can insert logical conditions to accurately find the desired element:

```
> my.data.frame$machine[my.data.frame$state=="Incorrect"]
```

```
[1] 1
Levels: 1 2
```

As you have guessed, with the previous command we have obtained the machine that produced the incorrect item in our very simple data set.

2.3.4 Importing and Exporting Data

In Sect. 2.4, we explain how to save and load a data object in R. This is the most effective way to work with data in R. On the other hand, R has lots of data sets ready to load in the workspace. You can see those included in the `base` package by typing `data()` in the R Console. Other packages (including the `SixSigma` package) have their own data sets (type `data(package="Six Sigma")`.

To load one of these data sets in the workspace, for example `cpus` from the `MASS` package, type:

```
> data(cpus, package = "MASS")
```

It is very common in statistical analysis, and especially in Six Sigma projects, to use data sets with different formats. The best way to deal with this is to convert the data from that source to a text file and then import it into R. Suppose you have a Microsoft Excel spreadsheet with the results of the performance of a process. First, save the spreadsheet as a `.csv` file (e.g., `test.csv`). Then, save the data in a data frame as follows:

```
> data.test <- read.csv("test.csv",
    header = TRUE, sep = ";")
> str(data.test)
```

```
`data.frame':   19 obs. of  3 variables:
 $ Id         : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Laboratory: int  1 1 1 1 1 1 1 1 1 2 ...
 $ Length    : num  9.7 7.65 8.96 6.32 11.51 ...
```

An even faster way to import data from a spreadsheet is by copying the data in the clipboard and then reading them from there in R. Once you have copied the range you want to import, type in the R Console:

```
> copied.data <- read.table("clipboard", header = TRUE)
```

R allows for importing data from different formats through the base installation as well as through add-on packages. The packages `RODBC` and `foreign` deal with ODBC sources and other statistical software, respectively.

The package `XLConnect` allows one to import data from Microsoft Excel files easily.[13] Using this package, files can also be manipulated (see the package documentation if you have an interest in other features). To import the file named `test.xlsx` directly, type:

```
> library(XLConnect)
```

```
XLConnect 0.1--7 by Mirai Solutions GmbH <xlconnect@mirai-
    solutions.com>
http://www.mirai-solutions.com ,
http://miraisolutions.wordpress.com
```

```
> wb <- loadWorkbook("test.xlsx")
> data.test <- readWorksheet(wb, sheet = 1)
> str(data.test)
```

```
`data.frame':   20 obs. of  4 variables:
 $ Id         : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Laboratory: chr  "A" "A" "A" "A" ...
 $ Length    : num  9.7 7.65 8.96 6.32 11.51 ...
 $ Diameter  : num  3.14 2.93 5.83 3.3 5.66 ...
```

To import a Minitab portable worksheet (`.mpt`) and many other file formats (like SPSS) we can use the functions in the `foreign` package:

```
> #SPSS file
> data.spss <- read.spss(test.sav)
> #Minitab file
> data.minitab <- read.mtp(test.mtp)
```

As far as exporting data is concerned, our advice is to apply the same main rule: save the data in a standard text format, and then import them from this file to your destination program:

```
> write.csv(my.data.frame,file = "mydata.csv")
```

[13]It requires Java in the Operating System, and package `RJava` in R.

2.4 Objects and Functions

In the previous section, we used objects and functions. In R, as is explained in [16], everything is an object, and every object has a class.

2.4.1 *Objects*

Let us find out the class of some of the objects we created in the last section.

```
> class(my.vector)
```

```
[1] "numeric"
```

```
> class(my.data.frame)
```

```
[1] "data.frame"
```

Everything means "everything." Even a function is an object:

```
> class(c)
```

```
[1] "function"
```

Data objects may be coerced from one type to another (if possible):

```
> as.character(my.vector)
```

```
[1] "10" "20" "30" "40"
```

```
> as.data.frame(my.matrix)
```

```
      myFirstCol mySecondCol
Case1         10          12
Case2         20          26
Case3         30          34
Case4         40          39
```

The objects we create are stored in the *workspace*, and all of them are available in the current R session. We can check all the objects of the session:

```
> objects()  #Equivalent function: ls()
```

```
 [1] "addObject"     "ccNormal"      "ccPoint"
 [4] "cc.tree"       "chapters"      "cpus"
 [7] "data.test"     "gencc"         "gencc2"
[10] "gencc3"        "gen.run"       "i"
[13] "machine.age"   "make.roadmap"  "my.data.frame"
[16] "my.factor"     "my.list"       "my.matrix"
```

```
[19] "my.var"         "my.vector"      "newDataset"
[22] "smsSets"        "ss.bookTable"   "s.y2"
[25] "s.y3"           "s.y4"           "s.y5"
[28] "test"           "testfiels"     "testfiels"
[31] "testlist"       "wb"             "wdBook"
[34] "x"
```

We can remove objects or even get rid of them all:

```
> rm(machine.age)
> # rm(list=ls()) removes all the objects
> objects()
```

```
 [1] "addObject"      "ccNormal"       "ccPoint"
 [4] "cc.tree"        "chapters"       "cpus"
 [7] "data.test"      "gencc"          "gencc2"
[10] "gencc3"         "gen.run"        "i"
[13] "make.roadmap"   "my.data.frame"  "my.factor"
[16] "my.list"        "my.matrix"      "my.var"
[19] "my.vector"      "newDataset"     "smsSets"
[22] "ss.bookTable"   "s.y2"           "s.y3"
[25] "s.y4"           "s.y5"           "test"
[28] "testfiels"     "testfiels"      "testlist"
[31] "wb"             "wdBook"         "x"
```

Unless you select yes when you are asked about saving the *workspace image* just before exiting R, the data and other objects will be erased. You can save some or all of the session objects and load them later when you run another R session.

```
> save(my.data.frame, file = "mydata.Rdata")
> #Only an object is saved
> save.image(file = "myimage.Rdata")
> rm(list = ls())
> objects()
```

```
character(0)
```

```
> load("myimage.Rdata")
> objects()
```

```
 [1] "addObject"      "ccNormal"       "ccPoint"
 [4] "cc.tree"        "chapters"       "cpus"
 [7] "data.test"      "gencc"          "gencc2"
[10] "gencc3"         "gen.run"        "i"
[13] "make.roadmap"   "my.data.frame"  "my.factor"
[16] "my.list"        "my.matrix"      "my.var"
[19] "my.vector"      "newDataset"     "smsSets"
[22] "ss.bookTable"   "s.y2"           "s.y3"
[25] "s.y4"           "s.y5"           "test"
[28] "testfiels"     "testfiels"      "testlist"
[31] "wb"             "wdBook"         "x"
```

As you have guessed, we saved the workspace image, then we removed all the objects, and finally we restored all the objects from the file system.

Table 2.4 Arguments for `matrix` function

Argument	Default value	Description
`data`	`NA`	Optional data vector
`nrow`	1	Desired number of rows
`ncol`	1	Desired number of columns
`byrow`	`FALSE`	Way of filling a matrix
`dimnames`	`NULL`	List of length 2 for names of rows and columns

2.4.2 Functions

Using the functions, we create objects or do something with them, usually passing them as arguments of the function. The arguments of a function are thoroughly explained in the documentation of the function, and we can pass them explicitly, by naming them, or implicitly, by putting them in the expected position.

For example, you already know the function `matrix`. It accepts the arguments in Table 2.4.[14]

The following two commands return exactly the same object:

```
> matrix(1:4, 2, 2)
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
> matrix(nrow = 2, ncol = 2, data = 1:4)
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

When an argument is not passed to the function, the function assumes the default value. There is a special argument, symbolized by "...". It means that further arguments may be passed, depending on the specific function.

Given that a function is an object, we must always call the function with brackets, even if it has no arguments or we want the default ones. Otherwise we will get the function as an object, instead of the returning value of the function. Next, we show the difference:

```
> objects()
```

```
 [1] "addObject"     "ccNormal"      "ccPoint"
 [4] "cc.tree"       "chapters"      "cpus"
 [7] "data.test"     "gencc"         "gencc2"
```

[14]Type `?matrix` to see the documentation.

```
[10] "gencc3"         "gen.run"         "i"
[13] "make.roadmap"   "my.data.frame"  "my.factor"
[16] "my.list"        "my.matrix"      "my.var"
[19] "my.vector"      "newDataset"     "smsSets"
[22] "ss.bookTable"   "s.y2"           "s.y3"
[25] "s.y4"           "s.y5"           "test"
[28] "testfiels"     "testfiels"     "testlist"
[31] "wb"             "wdBook"         "x"
```

```
> objects
```

```
function (name, pos = -1, envir = as.environment(pos), all.
    names = FALSE,
    pattern)
{
    if (!missing(name)) {
        nameValue <- try(name, silent = TRUE)
        if (identical(class(nameValue), "try-error")) {
            name <- substitute(name)
            if (!is.character(name))
                name <- deparse(name)
            warning(sQuote(name), "converted to character
                string")
            pos <- name
        }
        else pos <- nameValue
    }
    all.names <- .Internal(ls(envir, all.names))
    if (!missing(pattern)) {
        if ((ll <- length(grep("[", pattern, fixed = TRUE)))
            &&
            ll != length(grep("]", pattern, fixed = TRUE))) {
            if (pattern == "[") {
                pattern <- "\\["
                warning("replaced regular expression pattern
                    `[' by  '\\\\[''")
            }
            else if (length(grep("[^\\\\]\\[<-", pattern))) {
                pattern <- sub("\\[<-", "\\\\\\[<-", pattern)
                warning("replaced `[<-' by `\\\\[<-' in
                    regular expression pattern")
            }
        }
        grep(pattern, all.names, value = TRUE)
    }
    else all.names
}
<bytecode: 0x900c5b4>
<environment: namespace:base>
```

2.5 Operators and Functions Commonly Used

Now we present some examples of operators and functions commonly used in R. More functions are listed in Appendix A, with a short description. See the R manuals referred to in Sect. 2.2.3. The result of a function depends on the type of data passed as argument. Thus, if you pass a vector to a function that operates over a single value (e.g., square root), then the output will be a vector of the same length and the square root of each component. Other functions return a single value as the result of applying the function to the whole data object (e.g., the mean for a set of values).

2.5.1 *Operators*

Arithmetic and logical operations can be performed over data:

```
> 3.14 * my.vector
```

```
[1]  31.4  62.8  94.2 125.6
```

```
> sqrt(my.vector[3])
```

```
[1] 5.477226
```

```
> my.vector[3] >= my.vector[2]
```

```
[1] TRUE
```

There is a special operator for the product of matrices:

```
> my.vector%*%my.matrix
```

```
     myFirstCol mySecondCol
[1,]       3000        3220
```

See Table A.10 in Appendix A for additional operators.

2.5.2 *Mathematical Functions*

We can compute mathematical operations over data:

```
> cos(pi)
```

```
[1] -1
```

```
> exp(-2)
```

```
[1] 0.1353353
```

```
> round(data.test$Length)
```

```
 [1] 10  8  9  6 12  8  9 11 10 14  9  9 13  9  9  7 11
[18]  9 11 10
```

See Table A.12 in Appendix A for additional mathematical functions.

2.5.3 Functions for Vectors

Some functions produce a result over all the values of a vector, for example, if we want to know the mean of our vector:

```
> mean(my.vector)
```

```
[1] 25
```

Set operations can also be performed with vectors:

```
> A <- union(my.vector, -3)
> is.element(-3, A)
```

```
[1] TRUE
```

See Table A.13 in Appendix A for additional vector functions and Table A.11 for more set functions.

2.5.4 Loop and Summary Functions

A special group of functions allows us to summarize the information of a data object. For example, if we want to count the number of data taking each different value of the `state` variable in our data frame, we just type:

```
> table(my.data.frame$state)
```

```
Incorrect   Correct
        1         3
```

We can apply a function to all the elements of a data object. For example, we can obtain the mean for every element of our list/vector/matrix:

```
> lapply(my.list, mean)
```

```
$lvector
[1] 25

$lmatrix
[1] 26.375
```

```
> sapply(my.list, mean)
```

```
lvector lmatrix
 25.000  26.375
```

In a matrix, we can apply a function to make operations on rows (1 as the second argument) or columns (2 as the second argument) with the function `apply`:

```
> apply(my.matrix, 1, mean)
```

```
Case1 Case2 Case3 Case4
 11.0  23.0  32.0  39.5
```

```
> apply(my.matrix, 2, mean)
```

```
 myFirstCol mySecondCol
      25.00       27.75
```

A special function to summarize data in R is `summary`. This function is a generic R function. By generic, we mean that you can use it with arguments of different classes. For instance, when applying it to a data object, we obtain some statistics:

```
> summary(my.data.frame)
```

```
      CTQ               state    aux.mySecondCol machine
 Min.   :10.0   Incorrect:1   Min.   :12.00   1:2
 1st Qu.:17.5   Correct  :3   1st Qu.:22.50   2:2
 Median :25.0                 Median :30.00
 Mean   :25.0                 Mean   :27.75
 3rd Qu.:32.5                 3rd Qu.:35.25
 Max.   :40.0                 Max.   :39.00
```

Appendix A contains more functions for summarizing data. See Tables A.13, A.14, and A.7.

2.6 Graphics in R

One of the strengths of R is its powerful and versatile graphics system. The `graphics` package of the base installation can perform many types of graphics. Run the demos for `graphics`, `persp`, `image`, and `plotmath` to see some examples.

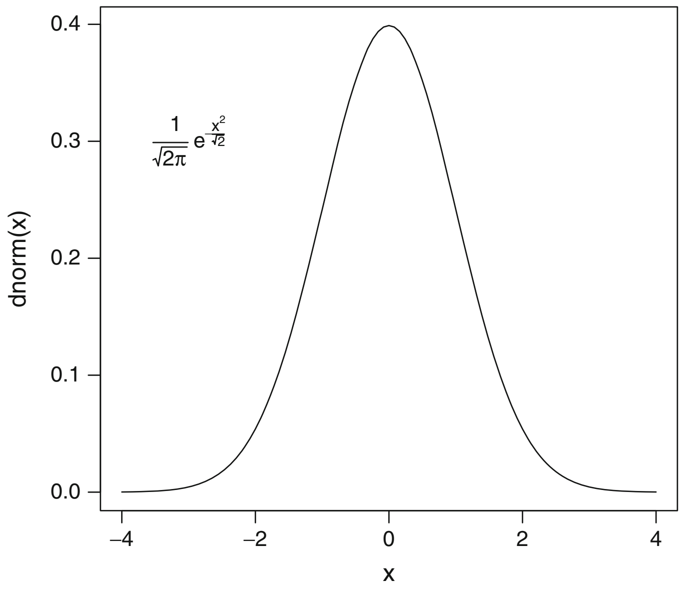


Fig. 2.9 Plotting curves and math expressions. The first command plots the density function of the normal probability distribution; the second command plots the mathematical expression of the function

The base graphics are perfect for learning R, but once you become a regular R user (we expect you will achieve that soon with the aid of this book), we recommend using more advanced packages to plot advanced graphics. The `grid`, `lattice`, `ggplot2`, and `sp` packages produce stunning graphics. In Sect. 2.8 we provide references, including some Web sites, to learn more about R graphics.

2.6.1 Plotting Functions

Using R we can plot any function and graph any mathematical expression (Fig. 2.9).

```
> curve(dnorm(x), -4, 4)
> text(-3, 0.3,
    expression(frac(1, sqrt(2 * pi)) * " " * e^{-frac(x^2,
        sqrt(2))}))
```

The function `plot` produces a different figure depending on the data object used as argument.

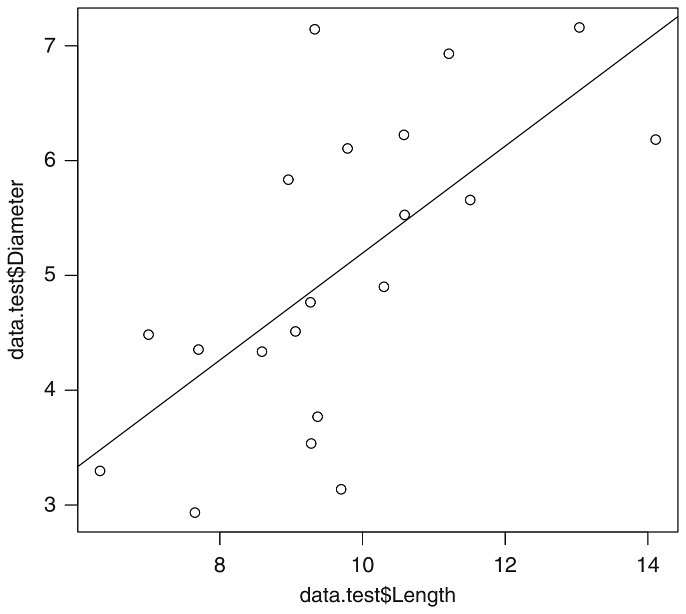


Fig. 2.10 A basic scatterplot. A cloud of points represents the values of the two numeric variables. A regression line was added to the plot

2.6.2 *Bivariate Plots*

When we want to represent the relation between two variables, we can produce several types of plots, depending on the nature of the explanatory variable. When both variables are continuous, we can use a scatterplot (Fig. 2.10).

```
> plot(data.test$Length, data.test$Diameter)
> abline(lm(Diameter~Length, data = data.test))
```

We have superposed a regression line with the function `abline`. It is the first time we have used the $\sim$ symbol. This tilde is intended to separate the two terms of a formula. The left side of the $\sim$ symbol is for the dependent variable/s, and the right side for the independent variable/s. We will also use formulas in Part IV for data modeling.

There are many options for plotting, even in the `graphics` package. The preceding plot can be customized as much as desired using the following code (Fig. 2.11).

```
> par(bg = c("#FCFCFC"))
> plot(data.test$Length, data.test$Diameter,
    main = "My first plot",
    sub = "This is a plot by Six Sigma with R book",
    pch = 16, las = 1,
    col = "orange",
    ylab = "Diameter", xlab = "Length")
```

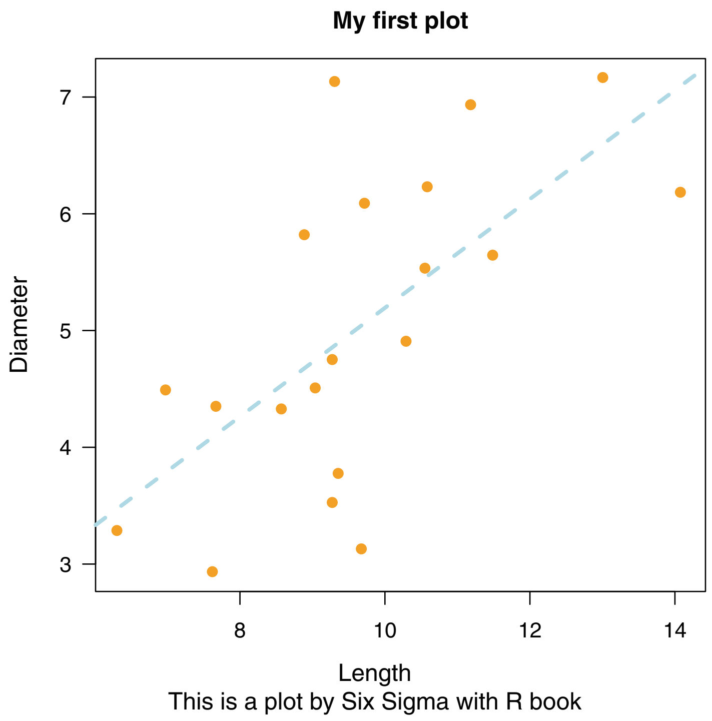


Fig. 2.11 Improved scatterplot. In this improved chart, we have changed the colors and symbols (dots and lines), in addition to the foreground color. Title, subtitle, and axis labels have been customized

```
> abline(l\,m(Diameter ~ Length, data = data.test),
    lty = 2, lwd = 3,  col = "lightblue")
```

When the explanatory variable is a factor, the plot function produces the so-called box plot (Fig. 2.12).

```
> plot(as.factor(data.test$Laboratory), data.test$Diameter)
```

For factors and discrete variables, bar plots (Fig. 2.13) are used.

```
> someData <- factor(rep(1:4, 14:11))
> plot(someData)
```

Univariate continuous data are usually represented with a histogram, which is a sort of bar plot. We can superpose a density line, as in Fig. 2.14.

```
> hist(data.test$Diameter, freq = FALSE, col = "gray")
> lines(density(data.test$Diameter), lty = 2, lwd = 2)
```

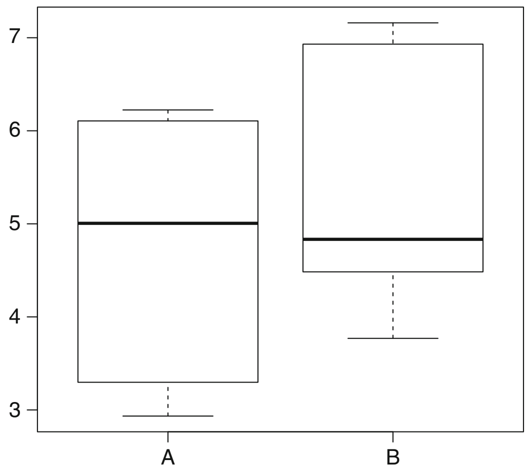


Fig. 2.12 Box plot chart. When the independent variable is a factor, the `plot` function generates a box plot

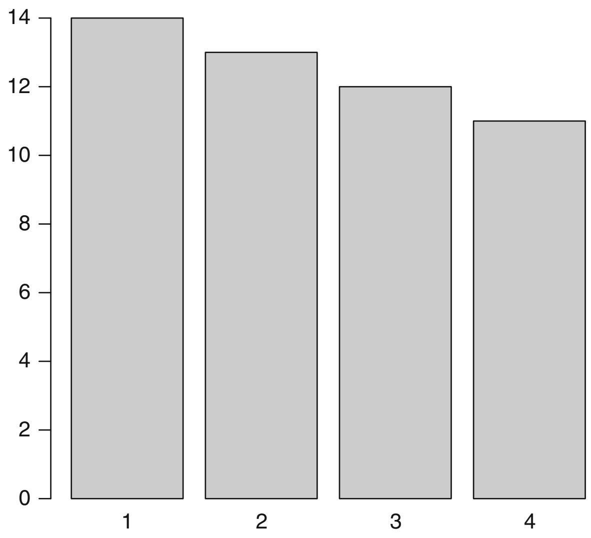


Fig. 2.13 Bar plot. The bar plot is the appropriate chart to represent counts on categorical variables

2.6.3 Pie Plots

For categorical data, pie plots (Fig. 2.15) can be used.

```
> pie(table(someData))
```

More plot functions are listed in Table A.19 in Appendix A. Useful options and arguments for graphic functions can be found in Table A.17. We will explain in detail the plots and charts used within Six Sigma and how to use them with R in Parts II–VI.

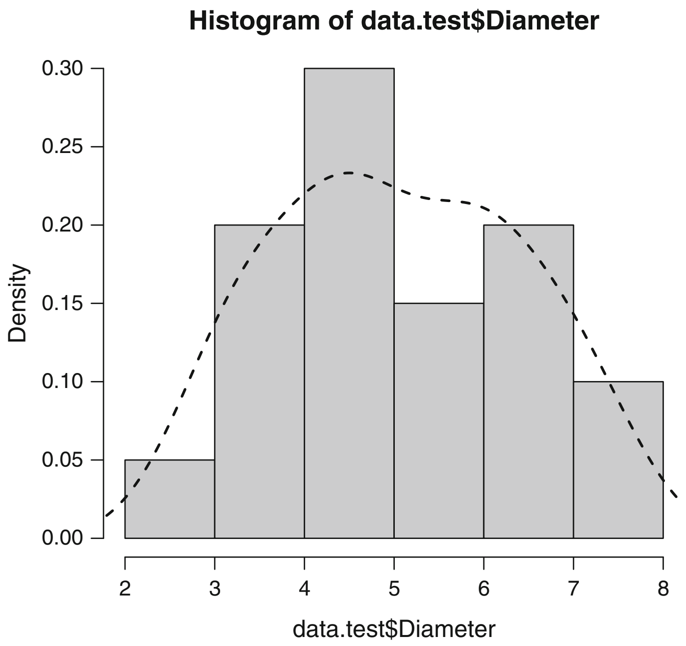
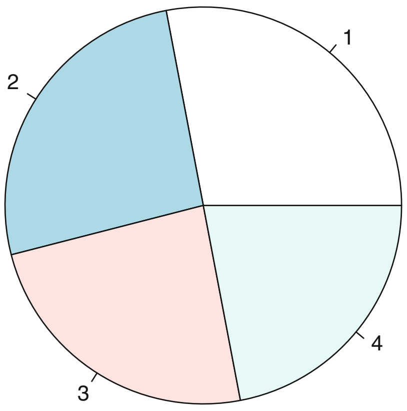


Fig. 2.14 Histogram with density line. The width of the bids corresponds to the width of the classes into which the continuous variable has been divided

Fig. 2.15 Pie plot. The angle of each sector is proportional to the counts of the category represented



2.7 Statistics

2.7.1 Samples and Combinatorial Computations

We can extract a random sample from a vector:

```
> sample(my.vector,2)
```

```
[1] 20 10
```

Other combinatorial computations can be carried out:

```
> factorial(5)
```

```
[1] 120
```

```
> choose(5, 2)
```

```
[1] 10
```

See Table A.16 in Appendix A for additional sampling functions.

2.7.2 Random Variables

We can easily generate random values for a probability distribution. For example, if we want to obtain a sample of size 5 from a normal process with mean 5.2 and standard deviation 0.5, the following commands should be used.

```
> rnorm(5, 5.2, 0.5)
```

```
[1] 5.759547 4.809685 5.650381 5.283937 5.758157
```

If we want to find out the probability that this distribution will be lower than 5.6, we use:

```
> pnorm(5.6, 5.2, 0.5)
```

```
[1] 0.7881446
```

We can also obtain the quantile or density values for a given value of the probability and variable, respectively:

```
> qnorm(0.9, 5.2, 0.5)
```

```
[1] 5.840776
```

```
> dnorm(5.6, 5.2, 0.5)
```

```
[1] 0.5793831
```

See Table A.15 in Appendix A for additional probability distributions.

We will explain in detail the statistical tools used in Six Sigma and how to use them with R in Parts II–VI.

2.8 Summary and Further Reading

This chapter provided basic background information on R that will be useful in following the rest of the book. We reviewed a wide array of functions of the R system, and we learned how to install, run, and interact with that system. Commands, functions, and objects should no longer be something alien for you after having read this chapter. We also imported, saved, and edited several types of data and then performed some operations with those data. Finally, some of the graphics and statistical tools to be explained thoroughly in subsequent chapters were introduced.

Nevertheless, some specialized books are available to the reader interested in learning how to exploit R for a successful Six Sigma project. [112] and [18] may serve as complete guides to the R system. [102] and [16] go one step beyond; we highly recommend them to improve your R knowledge. For improving and customizing graphics, [91], [105], and [74] are excellent guides.

There are abundant electronic resources on the Internet for learning and practicing with R. Some of our favorites are listed below:

- http://en.wikipedia.org/wiki/R_(programming_language)
- http://rwiki.sciviews.org/doku.php
- http://rgm2.lab.nig.ac.jp/RGM2
- http://en.wikibooks.org/wiki/R_Programming
- http://journal.r-project.org/
- http://rseek.org/
- https://r-forge.r-project.org/
- http://www.r-bloggers.com/
- http://www.statmethods.net/

The R manuals are free, online, and updated resources that you can consult whenever you need:

- An introduction to R: http://cran.r-project.org/doc/manuals/R-intro.pdf;
- To learn the R language definition: http://cran.r-project.org/doc/manuals/R-lang.pdf;
- To write R extensions: http://cran.r-project.org/doc/manuals/R-exts.pdf;
- To learn about R data import/export: http://cran.r-project.org/doc/manuals/R-data.pdf;
- To perform R installation and administration: http://cran.r-project.org/doc/manuals/R-admin.pdf;
- To understand R internals: http://cran.r-project.org/doc/manuals/R-ints.pdf;
- To consult the R reference index: http://cran.r-project.org/doc/manuals/fullrefman.pdf.

Entire books related to R are also freely available. These include [25,45,64], and [51]. [73] is available online under a Creative Commons license and in a bound print version from Chapman & Hall. Search the Web to find other books.

As far as the interface is concerned, several alternatives exist to the R Console. However, we recommend getting used to the R Console and then deciding which of the other interfaces fits your needs better. RCommander[15] is a good option for beginners. It is available for all platforms. RKWard[16] is a pretty good looking interface for Linux. JGR [17] and SciViews-R[18] are other suitable options.

For those who need to go further and make programs with R, two projects outshine the rest. RStudio[19] provides a clean and easy-to-install interface. Furthermore, you can produce scientific and professional documentation through its Sweave/LaTeX editor by literate programming. This latest feature is also available in Eclipse through the plugins StatET[20] and TexLipse (see [104]). This is the more complete environment in which to program, though it is not very easy to set up.[21] Nevertheless, there is a straightforward implementation available called Architect.[22] See [60] to find out more about literate programming.

Case Study

Choose any paper helicopter prototype defined in Chap. 1 and take 10 measurements of the flight time. Select your favorite method to enter data in R (directly as vectors and data frames, importing from a spreadsheet, or something else). Save the data in an R data file (`.RData`).

Use the R Console as well as the R editor to create and save a script with your commands.

Obtain a summary of the data and construct some plots. We will extend the case study thoughout the book to perform more analyses and construct more plots.

Practice

2.1. Install the `SixSigma` package yourself.

2.2. Save the data in Table 2.5 into a data frame. Get a summary of its variables. Save the data frame in a `.csv` file.

[15]http://www.rcommander.com/.

[16]http://rkward.sourceforge.net/.

[17]http://www.rforge.net/JGR/.

[18]http://www.sciviews.org/SciViews-R/.

[19]http://www.rstudio.org/.

[20]http://www.walware.de/goto/statet.

[21]There is a very good post explaining all the processes at http://www.r-bloggers.com/getting-started-with-sweave-r-latex-eclipse-statet-texlipse/.

[22]http://www.openanalytics.eu.

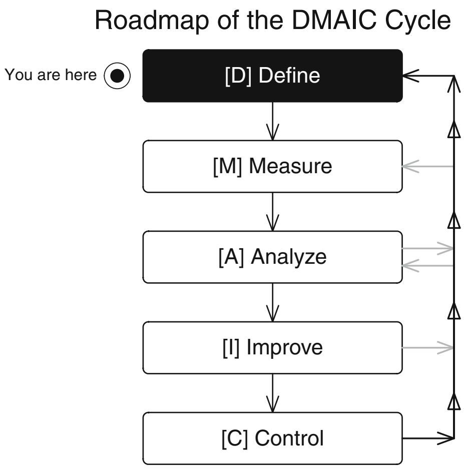
Table 2.5 Data for practice 1, Chap. 2

	Failure time	Temp	Factory
1	0.29	63.89	A
2	0.32	63.38	B
3	1.21	65.05	C
4	0.95	62.31	C
5	0.14	68.04	B
6	2.00	59.12	B
7	0.81	62.80	A
8	0.88	61.89	B

2.3. Make some plots using the data frame from the previous problem. Choose the correct plot for each variable.

2.4. Produce a table showing the number of items in each factory. Create a vector with temperatures corresponding to failure times lower than 1.

Part II
R Tools for the Define Phase



In this part of the book, tools useful during the Define phase are introduced.

In this phase, customer needs must be stated and the processes and products to be improved must be identified.

We will introduce two of the most representative tools: process mapping and loss function analysis.

Chapter 3
Process Mapping with R

A problem well stated is a problem half solved.

Charles Franklin Kettering

3.1 Introduction

Process mapping is a tool to retrieve information about a process. This information will be used in the phases of the Six Sigma project to be discussed later, and many of the measurements, analyses, and conclusions will be based on this information. The result of the process mapping is a map of a process. This map stems from the Project Charter and should be modified during the development of the project.

Process mapping begins with a top-level map, identifying the inputs and outputs of the process. Then the process is broken down into simpler steps, where parameters and features are identified as well. The study and classification of parameters will guide the posterior analysis of the relationship between the parameters and the critical to quality (CTQ) characteristics.

In this chapter, we will give an outline of process mapping. A short introduction is given in Sect. 3.2. Sections 3.3–3.5 show a method for building process maps and describe how to draw them with R. Section 3.6 contains some thoughts about diagrams. Finally, Sect. 3.7, Summary and Further Reading, provides references to find out more about process mapping and diagrams in R. You can practice using this tool with the indications in the Case Study and Practice sections at the end of the chapter.

3.2 Process Mapping as a Problem-solving Method

Process mapping can be considered a technique for solving problems. A process map helps in the posterior application of analytic methods to solve problems.

A basic problem-solving flow is the one in Fig. 3.1. The process map describes the current situation of a process, and it is fundamental to identifying root causes. In

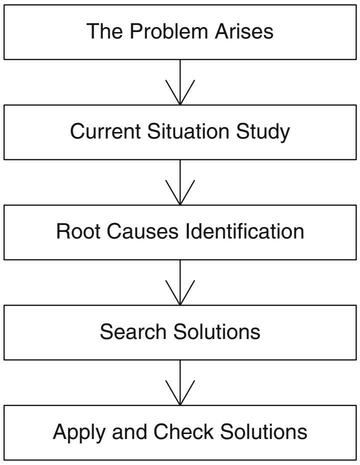


Fig. 3.1 A problem-solving flow. A systematic method helps to get rid of the problems' causes. The process mapping includes mainly the first three stages and, in some cases, part of the fourth stage

other words, the process map is an eye-opener. It identifies the stakeholders of the process (owner, customer, supplier, ...), bottlenecks, lean, and time waste.

While building the process map, we may also identify where measurements should be taken for the forthcoming analyses, and some possible solutions could arise.

The guidelines to construct a process map are provided in the Define phase by the Project Charter (Sect. 1.4 in Chap. 1). However, the process map should be *alive* and, thus, revised and updated when necessary. To obtain the data for the process map, several techniques can be used. The direct observation and review of documentation (manuals, articles, ...) is one example, but other collaborative tools are very useful. These include brainstorming, meetings, surveys, and workshops. Usually information provided by the expert (owner, experienced staff, ...) is the most interesting and valuable material.

3.3 Strategies for Process Mapping

Any chart that represents the flow of a process is a process map. A Six Sigma project may have different process maps. Thus, in the Define phase, we usually have a *top-level map* ("as is") which we will describe in Sect. 3.3.1, whereas in the Improve phase we have a *final process map*. This *final process map* should be similar to the *ideal state map* ("to be") developed in the Improve phase.

Mainly two specific types of process maps are used in Six Sigma. One is the SIPOC, which represents the flow of the process from supplier (S) to customer (C). The inner letters are from inputs, process, and outputs (Fig. 3.2). The other is the VSM (value stream map). This type of process map not only represents the flow of the process but also identifies which steps add value to the final product/service.

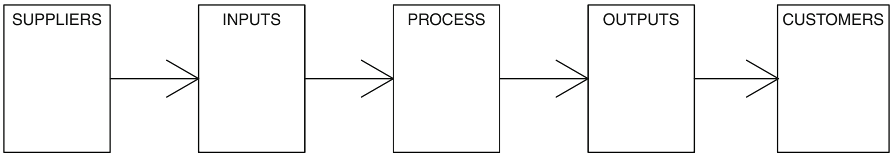


Fig. 3.2 SIPOC flow. The SIPOC flow chart represents the natural flow of a process or service from supplier to customer

A more sophisticated outline of the process can be made by arranging the boxes into rows representing the stakeholders involved in the process, and the columns correspond to the time in the flow of the process, with arrows connecting the different steps.

We will not go into the details of SIPOC or VSM. We will simply describe a simple two-stage process mapping strategy that is common to all process maps.

3.3.1 Two-stage Process Mapping

The most common way to construct process maps is through two stages. In the first stage, a top-level map is defined. In the second stage, the process is broken down into simpler steps. We begin with the top-level map, which contains only the inputs and the outputs of the process, whereas the process is a *black box* (at this stage). In this way, we focus on finding the most important output, that is, the key characteristics of the process (product or service), known as the CTQ characteristics. The outputs are represented by a Y, and we may refer to them as "the Ys" of the process.

While identifying the inputs, we should take into account the six Ms: machines, methods, materials, measurements, mother nature (environment), and manpower (people). The goal is to deduce which inputs are influencing the CTQ characteristics. The inputs are represented by an X, and we may refer to them as "the Xs" of the process.

Example 3.1 (Pizza restaurant). The manager of a restaurant wants to study the process of making and serving a pizza. First, he needs to make a top-level process map. After collecting some data from his staff, he determines that the inputs are the ingredients, the cook, the oven, and the plates; the outputs are temperature, taste, tenderness, weight, radius, and time to be served. In this case, these outputs are the main features of the pizza, that is, the CTQ characteristics.

A simple diagram of the top-level map (Fig. 3.3) can be drawn with this code in R (see [72] for more details about drawing diagrams with R):

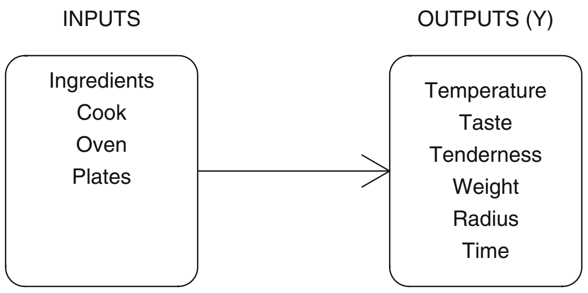


Fig. 3.3 Top-level process map for pizza example. The inputs and outputs are identified outside of the process. The process is a black box that will be analyzed later

```
> grid.roundrect(width = .25,
    height = unit(1.8, "inches"),
    x = 0.25)
> grid.text("INPUTS\n\nIngredients\nCook\nOven\nPlates",
    x = 0.25,
    y = 1,
    just = "top")
> grid.roundrect(width = .25,
    height = unit(1.8, "inches"),
    x = 0.75)
> grid.text("OUTPUTS (Y)\n\nTemperature\nTaste\nTenderness\
    nWeight\nRadius\nTime",
    x = 0.75,
    y = 1,
    just = "top")
> grid.lines(x = c(0.375, 0.625),
    y = c(0.5, 0.5),
    arrow = arrow())
```

You can apply formatting to texts and lines. The position and alignment of the elements are not difficult to understand as they are based on coordinates. In Sect. 3.7, you will find resources on grid graphics. □

3.3.2 *Drilling Down into the Process Steps*

As we have already mentioned, the second stage in process mapping is the breakdown of the process into simpler steps.

This breakdown of the top-level process map may look like an easy task. However, many obstacles must be overcome. The stakeholders of the process usually go on the defensive. They may feel threatened, and usually they are reluctant to change. So the first challenge for the Black Belt is to choose the best strategy to motivate the staff to obtain more precise information about the process.

The breakdown can be done sequentially as well, that is, by splitting the process into simpler steps. Each step (task, activity, subprocess) is named and described.

The steps have their own inputs and outputs. The inputs of the first step are the Xs. The outputs of the last step are the Ys. For intermediate steps, the outputs of a given step are the inputs of what follows.

Once we have broken down the process, we can focus on each step. We must identify the *parameters* of the process in that specific step. The parameters are called the xs (not equal to the Xs defined previously), and they represent all factors or variables that may have an influence over the features of the product at the output of the process. The *features* of the step are those characteristics that define its quality. They are called the ys of the process (not equal to the Ys defined previously). To distinguish between these two, notice that the capital letters correspond to the inputs and outputs of the process, whereas the small letters correspond to the inputs and outputs of the intermediate steps within the process.

The relationship $y = f(x)$ (how the parameters influence the features) is the basis of the next steps in the DMAIC cycle. Once the xs and ys have been measured, we will plot and analyze them to design experiments, test hypotheses, and obtain results for improvements. Thus, this is a very important task in a Six Sigma project.

3.4 Step-by-Step Process Mapping

The procedure to define a process map may be structured into the five steps described in the following sections.

3.4.1 Identifying Inputs and Outputs

As was already mentioned, the inputs are the Xs of the process. In addition to the six Ms, we can identify additional ones such as energy, regulations, or any others. Regarding the output, it will be a product or a service. The CTQ characteristics of the output are the Ys of the process. At this stage, these CTQ characteristics must have already been identified.

3.4.2 Listing the Project Steps

This task may be executed sequentially. The more detailed our breakdown of the process, the more accurate will be the posterior analysis. The steps may be classified into two groups: those that add value to the product and those that do not add value (e.g., transport, storing, inspection, ...).

3.4.3 *Identifying the Outputs of Each Step*

An output is produced at every step. It may be an in-process product, materials, data, etc., or it may be features of the final product, which may change from one step to another. As previously described, these in-process output/features are the *y*s of the process.

3.4.4 *Identifying the Parameters of Each Step*

The parameters affecting the process at a specific step are known as the *x*s of the step. These parameters are the ones that clearly influence the output/features of the product. This influence can be with respect to quality, cost, or other important aspects of the company. To detect the *x*s of a given step, we may use the techniques in Sect. 3.2.

3.4.5 *Classifying the Parameters*

While detecting the parameters in a step of the process, we must assess their influence in the features and how this influence is produced. Thus, the parameters must be classified in one of the following groups:

N	Noise: Noncontrollable factors
C	Controllable factors: May be varied during the process
P	Procedure: Controllable factors through a standard procedure
Cr	Critical: Those with more influence on the process

Example 3.2 (Pizza restaurant (cont.)). The process of making and serving a pizza can be broken down into the following steps:

- Prepare the dough.
- Spread the toppings.
- Bake the pizza.
- Deliver the pizza to the customer.

We assume that the inputs of each step are the output of the previous step. The inputs for the first stage are the *x*s defined previously (ingredients, cook, oven, and plates). Next, we describe in detail the parameters and outputs corresponding to each step (with the classification of the parameters in brackets).

First step: Dough:

- Parameters: cook (C), brand of flour (C), proportion of water (P).
- Outputs: dough (density, toughness, and thickness).

Second step: Toppings:

- Parameters: cook (C), brand of ingredients (Cr), amount of ingredients (P), preparation time (Cr).
- Outputs: raw pizza (diameter, weight, thickness).

Third step: Baking:

- Parameters: cook (C), queue (N), baking time (Cr).
- Outputs: baked pizza (temperature, tenderness, taste).

Fourth step: Deliver:

- Parameters: waiter (C), queue (N).
- Outputs: pizza on table (temperature, taste, tenderness, weight, radius, time).

Next, we save the names of the inputs, outputs, and steps of the process into R objects:

```
> inputs <-c ("Ingredients", "Cook", "Oven", "Plates")
> outputs <- c("temperature", "taste", "tenderness",
        "weight", "radius", "time")
> steps <- c("DOUGH", "TOPPINGS", "BAKE", "DELIVER")
```

Then we save the names of the outputs of each step in lists (creating first the list object). The outputs of a step are the inputs of what follows:

```
> io <- list()
> io[[1]] <- list("X's")
> io[[2]] <- list("Dough", "ingredients", "Cooker")
> io[[3]] <- list("Raw Pizza", "Cooker", "Oven Plate")
> io[[4]] <- list("Baked Pizza", "Plate")
```

Finally, we save the names, parameter types, and features:

```
> param <- list()
> param[[1]] <- list(c("Cook", "C"),
        c("flour brand", "C"),
        c("prop Water", "P"))
> param[[2]] <- list(c("Cook", "C"),
        c("Ing.Brand", "Cr"),
        c("amount", "P"),
        c("prep.Time", "Cr"))
> param[[3]] <- list(c("Cook","C"),
        c("queue", "N"),
        c("BakeTime", "Cr"))
> param[[4]] <- list(c("Waiter","C"),
        c("queue", "N"))
```

```
> feat <- list()
> feat[[1]] <- list("Density", "toughness", "thickness")
> feat[[2]] <- list("Diameter", "Weight", "thickness")
> feat[[3]] <- list("temperature", "tenderness", "taste")
> feat[[4]] <- list("temperature", "taste", "tenderness",
        "weight", "radius", "time")
```

Now we have all the data stored in variables. □

Table 3.1 Arguments of `ss.pMap` function

Argument	Description
steps	Vector of characters with names of "n" steps
inputs.overall	Vector of characters with names of overall inputs
outputs.overall	Vector of characters with names of overall outputs
input.output	Vector of lists with names of inputs of *i*th step, which will be the outputs of the $(i-1)$th step
x.parameters	Vector of lists with list of x parameters of process; the parameter is a vector with two values: name and type
y.features	Vector of lists with list of y features of step; the feature is a vector with two values: name and type
main	Main title of process map
sub	Subtitle of process map
ss.col	List of colors for lines of process map

3.5 Drawing a Process Map with the Six Sigma Package

We have provided the `SixSigma` package with a function that makes use of the `grid` package capabilities to draw process maps. The name of the function is `ss.pMap`, and it accepts the arguments in Table 3.1.

Example 3.3 (Pizza restaurant (cont.)). Once we have all the data in variables, we proceed to draw the process map for the pizza example. We call the `ss.pMap` function to draw the process map in Fig. 3.4.

```
> ss.pMap(steps, inputs, outputs,
      io, param, feat,
      sub = "Pizza Process Project")
```

□

3.6 Why Should I Use R for Drawing Diagrams?

Now we wish to motivate the advantages of using R to make diagrams instead of other so-called user-friendly programs. In principle, it may seem easier to draw boxes and arrows using the mouse with any drawing program (e.g., Dia ([59]), Microsoft Visio). Under the GUI paradigm (using a drawing program), you need to open the drawing program, modify it, check the aspect, export the image, import it into the report and check against the outline. Repeat these steps every time you change or generate a new diagram. The main advantage of doing it with R is that the diagram is reproducible. Every time you generate it, it will have the same aspect.

This feature is especially useful when you are using *reproducible research* techniques (Chap. 13). Thus, if you have the code inside the report of your project,

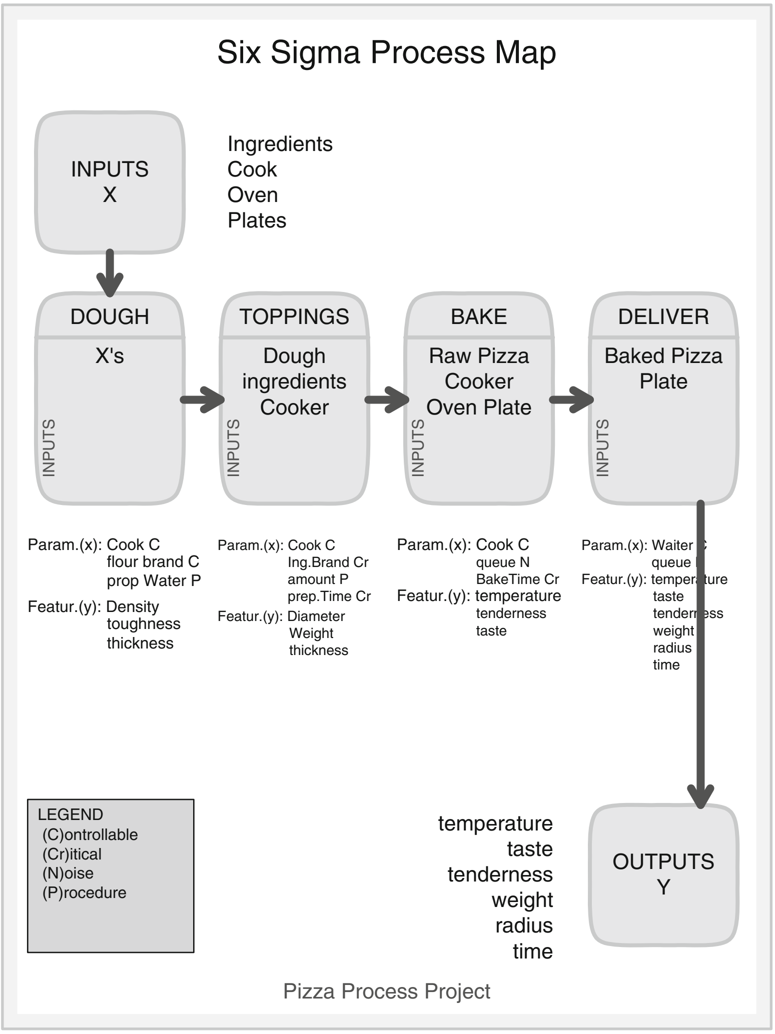

Six Sigma Process Map
INPUTS
X
Ingredients
Cook
Oven
Plates
DOUGH
X's
TOPPINGS
Dough ingredients Cooker
BAKE
Raw Pizza Cooker Oven Plate
DELIVER
Baked Pizza Plate
INPUTS
Param.(x): Cook C flour brand C prop Water P
Featur.(y): Density toughness thickness
Param.(x): Cook C Ing.Brand Cr amount P prep.Time Cr
Featur.(y): Diameter Weight thickness
Param.(x): Cook C queue N BakeTime Cr
Featur.(y): temperature tenderness taste
Param.(x): Waiter C queue N
Featur.(y): temperature taste tenderness weight radius time
LEGEND
(C)ontrollable
(Cr)itical
(N)oise
(P)rocedure
temperature
taste
tenderness
weight
radius
time
OUTPUTS
Y
Pizza Process Project

Fig. 3.4 Process map for pizza example. The whole process is visualized using the `ss.pMap` function

when you make a change in the code, the diagram will be rendered again and automatically updated in the report. Therefore, in the long term it is worthwhile to learn how to generate diagrams with R. As an example, most diagrams in this book have been created with R code.

3.7 Summary and Further Reading

Process mapping is a very important tool in the Define phase of the DMAIC cycle. The source for its construction is the Project Charter, and it consists in describing the steps that a process entails and the identification of inputs, outputs, parameters, and features. The relationship between the parameters and the features of the process will be the main thread in subsequent phases of a Six Sigma project. Hence, collecting this information is not a trivial matter.

In this chapter, we provided a brief introduction to process mapping. Some process mapping strategies were mentioned (SIPOC, VSM). A two-stage method to construct process maps was outlined through the breakdown of the process into simpler steps. We showed how to classify parameters according to their influence and nature using an intuitive example. This example was used to draw both top-level and detailed process maps.

The `grid` package has a significant amount of documentation. See the help (`help(package = "grid")`) for a list of functions for drawing diagrams or any annotation in an R graph. There are also several vignettes with complete explanations and assorted examples. Type `vignette(package = "grid")` to see a list of vignettes in the package. Then type `vignette("foo")` to obtain a PDF document on the topic `foo` (e.g., `vignette("moveline")`). If you plan to utilize the graphic features of R, [74] is a must-read. You can find more resources about graphics with R at the Web site of this book's author.[1]

Not all books about Six Sigma explain the process mapping tool. [2] presents value stream mapping as an extension of process mapping. [82], [92], and [22] have brief introductions to the topic. A book entirely aimed at building process charts is [30].

Case Study

Make a process map for the paper helicopter project. First, make a top-level map, identifying the inputs and the outputs. Then, make a breakdown of the process from your point of view, following the steps described in Sect. 3.4. Identify the parameters and the features in each step, and classify the parameters.

Try to draw the top-level map using the same code as in the pizza example, changing the names. Draw a process map with the `ss.pMap` function in the `SixSigma` package.

Hint: You can use the `example("ss.pMap")` command to see an example of the process map for the paper helicopter project, and then try to generate it following the instructions given in this chapter.

[1] http://www.stat.auckland.ac.nz/~paul/.

Practice

3.1. Draw a top-level map for the process of making an invoice in an administrative department of an office. You need the inputs (data, computer, operator) and the outputs (invoice). You can print some features of the invoice that may be important, e.g., number of pages, color, paper size, etc.

3.2. Draw a process map for the process of doing the laundry, using three steps: prepare, wash, and hang. Image the inputs and the outputs and include some parameter and feature examples.

Chapter 4
Loss Function Analysis with R

Defects are not free. Somebody makes them, and gets paid for making them.

W. Edwards Deming

4.1 Introduction

Most features defining a product are not usually important to the customer. Only a few of them are critical to quality (CTQ), in particular, those defining what the customer expects. To meet these expectations, the processes involved in the development of the final product should be correct. This is the Six Sigma way: high-quality processes lead automatically to high-quality products.

This relates to the concept of *cost of quality* (COQ), which is the cost of having a low-quality product (from the customer's perspective). Some managers still think that this concept is equivalent to total quality cost, which corresponds to the amount of money expended in implementing quality methodologies and improving processes. To avoid misunderstandings, we will refer to COQ as the cost of poor quality. The cost of poor quality will result in a quantifiable loss for the organization and for society in general. This loss can be modeled by a function. In Six Sigma, this function is based on the variability of the process.

In this chapter we will analyze the quality loss function introduced by Taguchi and explain how to use it to calculate the average loss of a process. In Sect. 4.5, we link this tool with other phases of the design, measure, analyze, improve, and control cycle. Use of the `SixSigma` package for a simple loss function analysis is explained in Sect. 4.6. This simple model is known as "nominal-the-better." More complex models are briefly summarized in Sect. 4.7.

4.2 Cost of Poor Quality

Throughout the history of quality management, many definitions of quality have been advanced. In most of them, the concept of *adequateness to customer needs* remains, regardless of the formulation. If we look for a formal definition, the principal source is the ISO Standards (ISO 9000:2008 Standard), where we find the following definition of quality:

"Degree to which a set of inherent characteristics fulfills requirements."

The requirements, or customer needs, must be a characteristic of the product or service that can be measured. Thus, requirements are usually a *target* value and a *tolerance* around the target, and they can be expressed as the interval between the lower specification limit (target minus tolerance) and the upper specification limit (target plus tolerance).

The characteristic used to measure quality by assessing the requirements is called CTQ (acronym for critical to quality), already defined in previous chapters. In the real world, there is variation in a process, and what we have is a sample of measures within an interval (time, batch, . . .). This sample has a probability distribution. If the process is normally distributed, the actual values of the CTQ characteristic surround the mean of the process with some variation (Fig. 4.1).

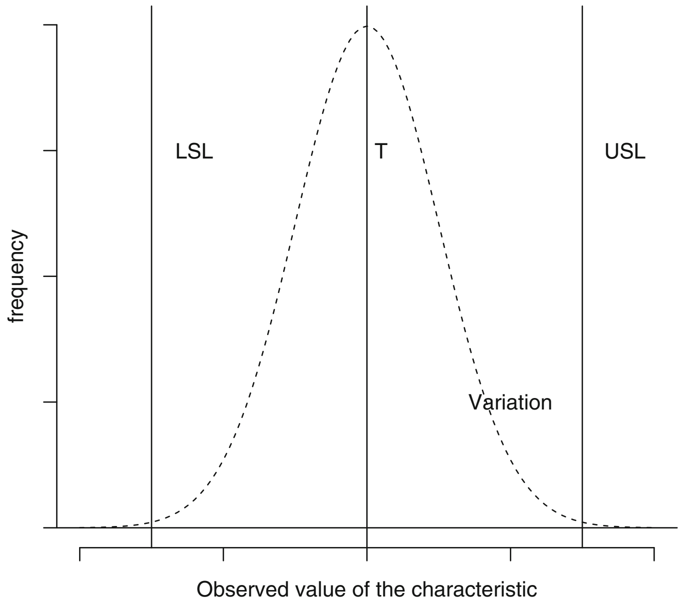


Fig. 4.1 Frequency distribution of observed values and requirement limits. In a normal distributed process, the observed values surround the target. Far from the mean are the smallest frequencies

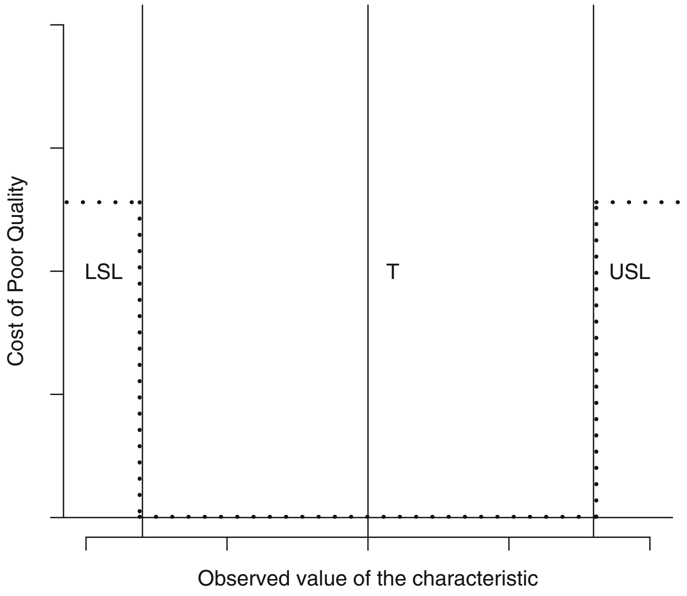


Fig. 4.2 Cost of poor quality under classical approach. The *dotted line* represents the cost of poor quality when the observed value of the CTQ characteristic varies. In this paradigm, the cost only is produced when the CTQ characteristic is outside the specification limits

When the CTQ characteristic lies outside of the specification limits, that is, quality is poor, the company incurs a cost that obviously must be calculated. This cost may be produced by both measurable (scrapping, rework, waste, forfeit, ...) and unmeasurable (bad reputation, dissatisfied staff, ...) losses. The former can be calculated directly, but the latter must be estimated somehow.

Under a classical approach to quality (Fig. 4.2), we can multiply the number of defective items by the cost of poor quality in a period and take this fact as the cost of poor quality. This is far from scientific, and so under the Six Sigma approach to quality, we need a better tool. This tool is the Taguchi loss function described in the following sections.

4.3 Modeling the Loss Function

4.3.1 Some Notation

Let us define some of the symbols to be used. The CTQ characteristic will be represented by Y. The loss function will provide a number indicating the value of the cost in monetary units ($, €, £, ...). This cost depends directly on the value of the CTQ. Thus, we say that *the loss is a function of the observed value* and represent it by $L(Y)$. This entails that for every value of the CTQ characteristic, we only have one value of the loss (cost).

The target value of the CTQ characteristic is denoted by Y_0 and the tolerance by Δ. Let the cost of poor quality at $Y = Y_0 + \Delta$ be L_0. That is:

$$L_0 = L(Y_0 + \Delta). \tag{4.1}$$

4.3.2 Taguchi Loss Function

As was mentioned in the previous section, it is not enough to have a CTQ characteristic inside the specification limits. In fact, a definition of quality under the Six Sigma approach should *approximate the target with as little variation as possible*. Then the loss function should be related to the distance from the target. [99] defined the loss function (known as the Taguchi loss function) as

$$L(Y) = k(Y - Y_0)^2. \tag{4.2}$$

This function has the following properties:

- The loss when the observed value is equal to the target is zero.
- The loss increases when the observed value moves away from the target.
- The constant k is indicative of the risk of having more variation (Fig. 4.3).

Once we have evaluated the cost of poor quality for an individual item as described in Sect. 4.2, we can calculate the constant k, replacing the target value Y_0 by its value in (4.2) and taking into account the cost at L_0 defined in (4.1).

Example 4.1 (Bolts). A factory makes bolts whose CTQ characteristic is the diameter of the bolts. Suppose that the Master Black Belt wants to assess the process of making 10-mm bolts. The following data are known:

- The customer will accept the bolts if the diameter is between 9.5 and 10.5 mm.
- When a bolt fails to meet the requirements, it is scrapped, and the cost estimation is 0.001 monetary units.

Thus the target is

$$Y_0 = 10,$$

the tolerance of the process is

$$\Delta = 0.5\,\text{mm},$$

and the cost at $Y_0 + \Delta$ is

$$L_0 = L(Y_0 + \Delta) = 0.001.$$

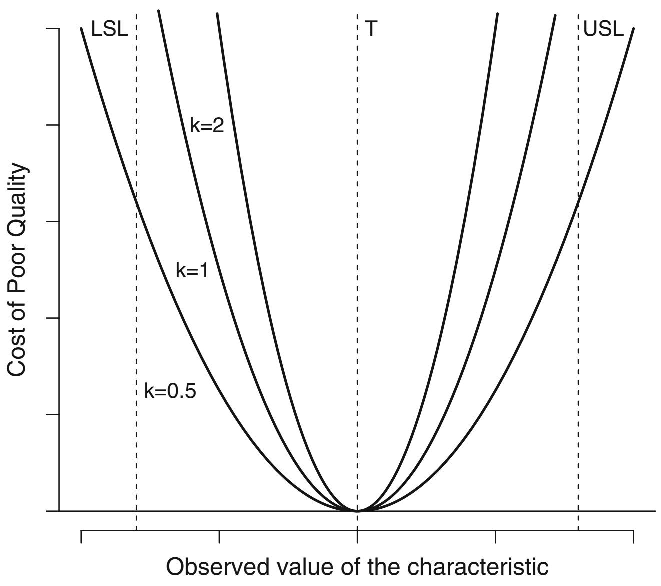


Fig. 4.3 Taguchi loss function. When the observed value of the CTQ characteristic is exactly the target, there is no cost. The further we move away from the target, the higher the cost of poor quality. The constant k indicates how fast the cost rises as the distance to the target increases

As was mentioned, we can easily compute the value of the constant k with the formula of the loss function:

$$\begin{aligned} L(Y) &= k(Y - Y_0)^2, \\ L(Y_0 + \Delta) &= k((Y_0 + \Delta) - Y_0) = L_0, \\ k \times \Delta &= L_0, \\ k &= \frac{L_0}{\Delta}. \end{aligned}$$

This formula is always the same. Hence, we can model the loss function knowing the tolerance (Δ) and the cost of poor quality of an individual item (L_0).

In the case at hand, k is equal to

```
> 0.001/0.5
```

```
[1] 0.002
```

So the loss function for the process of making bolts is

$$L(Y) = 0.002(Y - 10)^2,$$

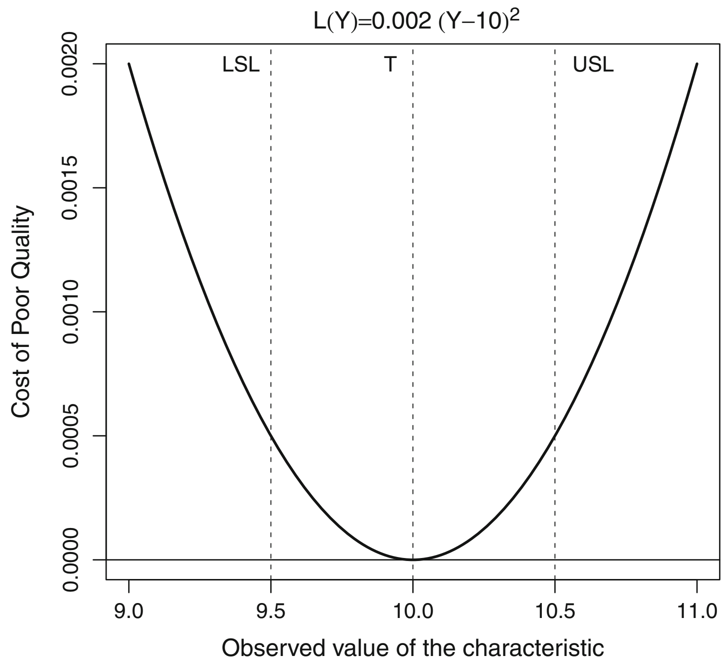


Fig. 4.4 Taguchi loss function for bolt example. This is the graphical representation of Taguchi's loss function for $k = 0.0002$

and we can plot the function in R with the following code (see the result in Fig. 4.4):

```
> curve(0.002 * (x - 10)^2, 9, 11,
    lty = 1,
    lwd = 2,
    ylab = "Cost of Poor Quality",
    xlab = "Observed value of the characteristic",
    main = expression(L(Y) == 0.002 ~ (Y - 10)^2))
> abline(v = 9.5, lty = 2)
> abline(v = 10.5, lty = 2)
> abline(v = 10, lty = 2)
> abline(h = 0)
> text(10, 0.002, "T", adj = 2)
> text(9.5, 0.002, "LSL", adj = 1)
> text(10.5, 0.002, "USL", adj = -0.1)
```

□

Table 4.1 Data for bolt example

10.4042	10.0578	9.7491	10.0475	10.1301
10.2584	10.1098	10.0436	9.9468	9.6706
9.9478	10.1013	10.2102	10.1539	9.7827
10.2678	9.8838	9.6940	10.1691	9.8784
10.1144	9.8491	10.2111	10.0913	9.9463
9.9830	10.1462	10.0127	10.0299	10.0379
10.3977	9.9481	10.3018	9.6780	10.1120
10.2947	10.1443	10.0357	10.0244	10.1306
10.2469	9.8660	9.9973	9.7712	9.7274
10.0433	9.8711	9.9762	9.8510	10.1934

4.4 Average Loss Function

In the previous section we obtained the quality loss function for a single item. The aim of the loss function analysis is to calculate the cost of poor quality for a process over a period of time.

Therefore, if we have n elements in a period or set of items, the average loss per unit (L) is obtained by averaging the individual losses. Thus:

$$L = \frac{\sum_{i=0}^{n} k\left(Y_i - Y_0\right)^2}{n} = k \times \frac{\sum_{i=0}^{n} \left(Y_i - Y_0\right)^2}{n}.$$

Notice that the second factor is the mean squared deviation (MSD). Hence, the average loss per unit in a given sample (period, batch, ...) is simply expressed by

$$L = k(\text{MSD}). \tag{4.3}$$

Example 4.2 (Bolts (cont.)). To calculate the average loss function for the bolt factory, we need a sample of bolts. Let us suppose that a random sample of 50 bolts has been correctly obtained. Their diameters are listed in Table 4.1. The data are available as a data frame in the `SixSigma` package. Type `ss.data.bolts` to see them. The unitary average loss can be obtained with the following code:

```
> 0.002*(sum((ss.data.bolts$diameter-10)^2))/length(ss.data.
    bolts$diameter)
```

```
[1] 6.74413e-05
```

□

4.5 Use of Loss Function Within DMAIC Cycle

In the Measure phase, loss function analysis is used to obtain the expected loss (in average) of a group of items (e.g., within a period, a batch, a region, etc.). This is simply the multiplication of the number of items in the group by the average loss per item.

Example 4.3 (Bolts (cont.)). The factory has a production of 100,000 bolts per month. Thus the expected loss in a month is

```
> 100000 * 0.002 * (sum((ss.data.bolts$diameter - 10)^2))/
    length(ss.data.bolts$diameter)
```

```
[1] 6.74413
```

□

Loss function analysis is not an isolated tool. For example, it can be used to set the tolerance limits for customers and suppliers on a scientific basis using knowledge about customer perceptions of specifications. This eventually leads to designing the acceptance sampling plan.

In the Improve phase, the quality loss function is used to ascertain the economic impact of a new method, material, etc. In other words, it represents a way to quantify improvements.

4.6 Loss Function Analysis with Six Sigma Package

A specific function in the `SixSigma` package performs loss function analysis. To show its performance, we will use the data described in the previous sections. The function is called `ss.lfa`, and it accepts the arguments in Table 4.2. The function returns a list with the elements in Table 4.3.

Table 4.2 Arguments for `ss.lfa` function

Argument	Description
lfa.data	Data frame with data sample
lfa.ctq	Name of field in data frame containing data
lfa.Delta	Process tolerance
lfa.Y0	Process target
lfa.L0	Cost of poor quality at tolerance limit
lfa.size	Number of items to calculate total loss in a group
lfa.output	String with type of output: "text," "plot," "both")
lfa.sub	Subtitle of graphic output

Table 4.3 Values returned by `ss.lfa` function

Value	Description
lfa.k	Constant *k* for loss function
lfa.lf	Expression with loss function
lfa.MSD	Mean squared differences from target
lfa.avLoss	Average loss per unit of the process
lfa.Loss	Total loss of process (if size is provided)

If "both" or "plot" is passed as the `lfa.output` argument, then a graphic is generated with the algebraic expression of the function and its graphic, in addition to the data and the computations.

Another function evaluates the loss function at a specific value of Y (the CTQ characteristic). Type `help("ss.lf")` to see how to use it.

Example 4.4 (Bolts (cont.)). To obtain the output of the loss function analysis for the bolt project, we use the following command:

```
> ss.lfa(ss.data.bolts, "diameter", 0.5, 10, 0.001,
    lfa.sub = "10\,mm. Bolts Project",
    lfa.size = 100000, lfa.output = "both")
```

```
$lfa.k
[1] 0.002

$lfa.lf
expression(bold(L == 0.002 %.% (Y - 10)^2))

$lfa.MSD
[1] 0.03372065

$lfa.avLoss
[1] 6.74413e-05

$lfa.Loss
[1] 6.74413
```

When passing the string "both" or "text" as an argument of the `lfa.output` function we obtain a list of the elements mentioned above. Thus, for example, the total loss is 6.7441.

The input and output values and the expression of the function and its graphical representation are depicted in Fig. 4.5. □

4.7 Other Models

In the preceding sections, we assumed that the CTQ characteristic had a target value with a surrounding tolerance. This model is called "nominal-the-better."

Not all the processes fit this model. Specifically, there are processes with only one specification limit (upper or lower). Next, we describe briefly these two possibilities.

Smaller-the-Better Characteristic

A smaller-the-better characteristic has only an upper specification limit (USL). Above this value, the product does not accomplish the customer specification. Thus,

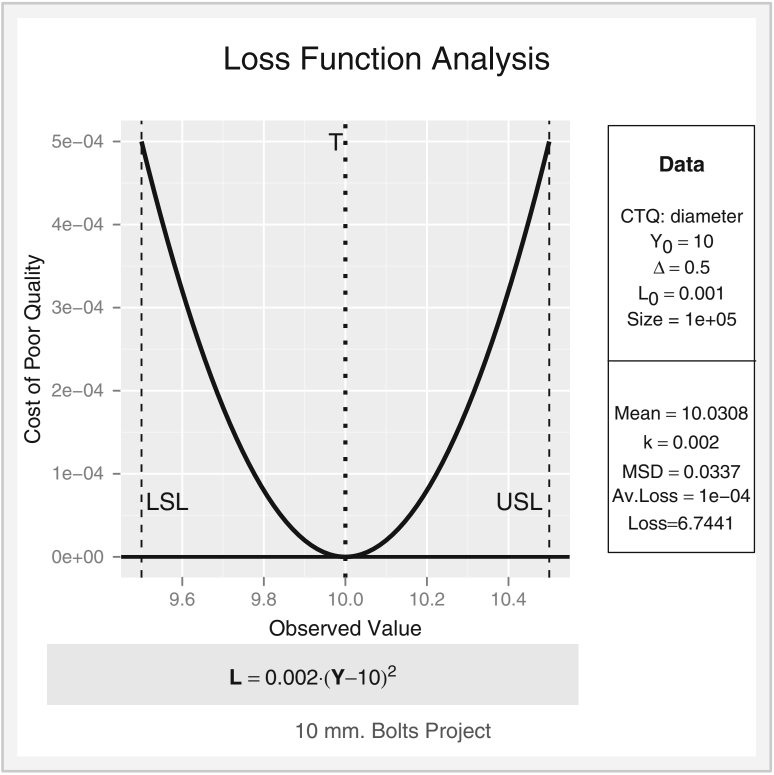


Fig. 4.5 Loss function analysis for bolt example. The Taguchi loss function for the bolt example is plotted in the graphic. The specification limits and the target are also represented as *vertical lines*. Below the graphic is the algebraic expression of the function. On the right-hand side of the output, we have the input data (variable, target, tolerance, cost at tolerance limit, and size of production set) and the computed output data (mean, constant k, MSD, average loss, and total loss)

the ideal target value is defined as zero. For example, the CTQ characteristic of a heat exchanger could be heat losses. Another example is any service in which the CTQ characteristic is the time to serve a client (process, customer, telephone call, order, . . .). The equation corresponding to the loss function for this kind of processes is as follows:

$$L = kY^2.$$

The graphical representation of the function has the shape in Fig. 4.6.

Larger-the-Better Characteristic

The larger-the-better characteristic corresponds to the case of only a lower specification limit (LSL). In this case, the initial target value is infinity. Typical examples

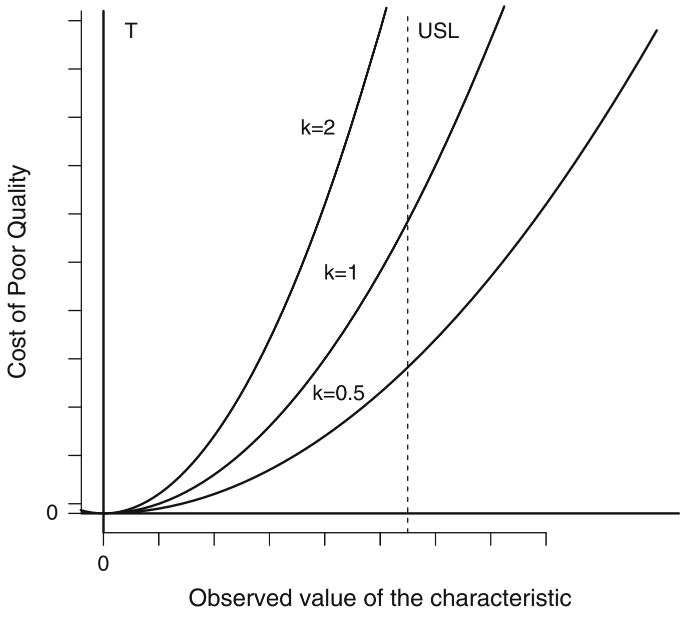


Fig. 4.6 Smaller-the-better loss function. The loss is minimized as the observed value of the characteristic decreases. The higher the observed value, the larger the cost of poor quality amounts

are the resistance of a material or the strength of a weld. The equation of the loss function for this model is as follows:

$$L = \frac{k}{Y^2}.$$

In this case, the graphical representation is shown in Fig. 4.7.

The computations for the analysis of loss functions for smaller-the-better and larger-the-better characteristics may be done in a similar way to that described for the nominal-the-better model. In [99] you can check some other applications of the loss function analysis.

4.8 Summary and Further Reading

This chapter gave a brief introduction to quality loss function analysis. In the framework of quality, this loss function was introduced by Genichi Taguchi and is thus also known as the Taguchi loss function.

First, the concept of COQ was discussed. Then, we modeled the cost function and performed calculations for the "nominal-the-better" model. We used an example and functions from the `base` installation of R and from the `SixSigma` package.

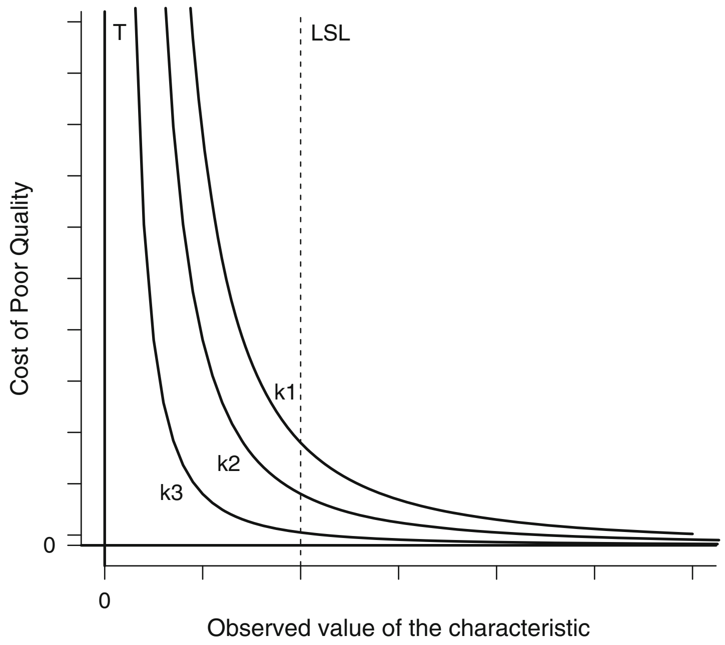


Fig. 4.7 Larger-the-better loss function. The loss is minimized as the observed value of the characteristic increases. The lower the observed value, the larger the cost of poor quality amounts

A straightforward explanation of the Taguchi quality loss function can be found in the "Michigan Chemical Process Dynamics and Controls Open Textbook" (see [53]). Other good references for the Taguchi methods are those by [89] and [99].

Case Study

Analyze the loss function for the paper helicopter project. Estimate the cost of poor quality when the flight time does not meet customer specifications (e.g., cost of the sheet, cost of the operator building, etc.). Set an arbitrary target and tolerance based on your experience from previous chapters.

Model the function and run the `ss.lfa` function from the `SixSigma` package to find out your average loss. Estimate the number of helicopters you can build in a month and get the total loss from the output of the function.

Practice

4.1. Write the loss function for a process whose target value is 15, tolerance is 2, and cost of poor quality at the tolerance limit is 350.

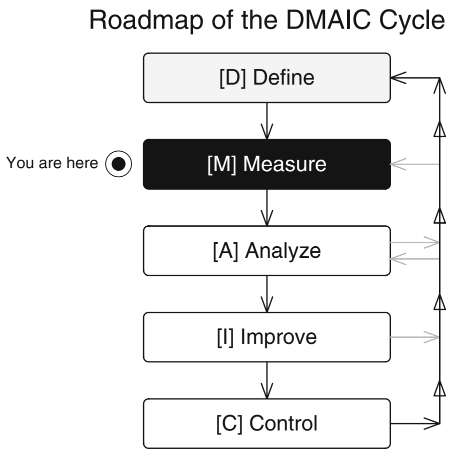
4.2. Consider a process where bottles are to be filled with wine. The target of the bottle-filling process is 750 cl, and the specification limits are 740 and 760 cl. Suppose that the mean cost of poor quality of a bottle is \$1.25.

What is the algebraic expression of the loss function? Try to plot it with R as in the bolt example.

4.3. Use the data of the process in `ss.data.ca` to perform a loss function analysis using the `ss.lfa` function in the `SixSigma` package.

What is the average loss? If the number of bottles filled in a month is 12,500, what is the total loss?

Part III
R Tools for the Measure Phase



In this part of the book, tools useful during the Measure phase are introduced.

In this phase, the baseline and target performance of the process must be determined, the input/output variables of the process defined, and the measurement systems validated.

We will introduce three of the most representative tools: measurement system analysis, Pareto analysis, and process capability analysis.

Chapter 5
Measurement System Analysis with R

If you cannot measure it, you cannot improve it.

Lord Kelvin

5.1 Introduction

A measurement system analysis (MSA), also known as a gage R&R study, identifies and quantifies the sources of variation that influence the measurement system. R&R stands for repeatability and reproducibility. It is a very important matter in Six Sigma because if the variability of the measurement system is not controlled, then the process cannot be improved. To perform a gage R&R study, several of the individual tools described in other chapters of the book may be used, such as control charts, analysis of variance (ANOVA), and plots.

The principal types of studies are crossed studies and nested studies. This chapter shows how to use these tools individually with R and provides an interpretation of the outputs from the SixSigma package for crossed studies.

First, we provide some definitions according to international standards. Then we explain how to collect data for an MSA with an example that is used throughout the chapter. Finally, a numerical and graphical analysis is performed and the results are explained.

5.1.1 Definitions

The most important activity in the Measure phase is the MSA. MSA is used to quantify the amount of variation coming from the measurement system. Commonly, MSA is known as gage R&R, where, as we have already mentioned, R&R stands for repeatability and reproducibility.

So it is very important to have an accurate measurement system. This is not the same as "exact": there is always some variability in a process. A good measurement system has only random variability due to the inherent variation of the item being measured. It must have no significant variability produced by appraisers (operators, machines, etc.), parts, time, or another factor.

Regarding the concepts involved in MSA, some of the principal definitions used throughout this chapter can be found in ISO 3534 (*Statistics*):

Accuracy The closeness of agreement between a test result and the accepted reference value.

Trueness The closeness of agreement between the average value obtained from a large series of test results and an accepted reference value.

Precision The closeness of agreement between independent test results obtained under stipulated conditions.

Repeatability Precision under repeatability conditions (where independent test results are obtained using the same method on identical test items in the same laboratory by the same operator using the same equipment within short intervals of time).

Reproducibility Precision under reproducibility conditions (where test results are obtained using the same method on identical test items in different laboratories with different operators using different equipment).

In summary, repeatability may be defined as the inherent variability of the measurement system (under similar conditions) and reproducibility as the variability under different conditions (groups), for instance, operators, machines, or day of the week.

5.2 Data Analysis

5.2.1 Data Collection

As in all statistical research, randomization is very important in Six Sigma. Thus, we must randomly select the parts (prototype, piece, mechanical items, service activities) and assign them to the appraisers (machines, meters, operators, ...), again randomly.

In what follows, we will consider n measures of each part ($n \geq 2$) within each appraiser. Let a and b be the number of parts and the number of appraisers, respectively. Let us save the data as a data frame with one numeric variable (the feature to measure) and two factors (parts and appraisers). Thus, we have $n \times a \times b$ observations.

Table 5.1 Battery voltage measures

	Voltmeter	Battery	Run	Voltage
1	1	1	1	1.4727
2	1	1	2	1.4206
3	1	1	3	1.4754
4	1	2	1	1.5083
5	1	2	2	1.5739
6	1	2	3	1.4341
7	1	3	1	1.5517
8	1	3	2	1.5483
9	1	3	3	1.4614
10	2	1	1	1.3337
11	2	1	2	1.6078
12	2	1	3	1.4767
13	2	2	1	1.4066
14	2	2	2	1.5951
15	2	2	3	1.8419
16	2	3	1	1.7087
17	2	3	2	1.8259
18	2	3	3	1.5444

Example 5.1 (Voltage in batteries). A battery manufacturer makes several types of batteries for domestic use. The Black Belt of the company must start a Six Sigma project in order to improve the 1.5-volt battery production line. He has identified the voltage output as the CTQ characteristic in the Define phase. Now the measurement system must be evaluated before starting other analyses.

There are two voltmeters available, and three different batteries are randomly picked up from the end of the production line. The voltage in each battery is measured with each voltmeter three times.[1] The results are listed in Table 5.1.

In this example, the parts are the batteries ($a = 3$) and the appraisers are the voltmeters ($b = 2$). As the measurement is taken three times ($n = 3$), there are $3 \times 2 \times 3 = 18$ measurements. Let us save the data in a data frame using the following commands:

```
> voltmeter <- factor(rep(1:2, each = 9))
> battery <- factor(rep(rep(1:3, each = 3), 2))
> run <- factor(rep(1:3, 6))
> voltage <- c(1.4727, 1.4206, 1.4754, 1.5083, 1.5739,
        1.4341, 1.5517, 1.5483, 1.4614, 1.3337,
        1.6078, 1.4767, 1.4066, 1.5951, 1.8419,
        1.7087, 1.8259, 1.5444)
> batteries <- data.frame(voltmeter, battery,
        run, voltage)
```

You can type `batteries` to see and check the data. □

[1]It is supposed to be done randomly.

$$\text{Total Variability}\begin{cases}\text{Gage R\&R}\begin{cases}\text{Repeatability}\\ \text{Reproducibility}\begin{cases}\text{Appraisers}\\ \text{Interaction}\end{cases}\end{cases}\\ \\ \text{Part-to-Part}\end{cases}$$

Fig. 5.1 Variability decomposition. The total variability can be split into gage R&R variability (from the inherent randomness of the process) and part-to-part variability (not due to the measurement system)

5.2.2 *First Approach to Analysis of Variance (ANOVA)*

As was already mentioned, the key point is the identification of the variability sources within the measurement system. The statistical technique known as analysis of variance (ANOVA) allows us to find out who is responsible for that variation.

A reasonable measurement system should have random variability due to the inherent variation of the item that is been measured and no variability arising by appraisers operations, parts, etc. To perform ANOVA, the statistical functions `lm` and `anova` from the `stats` package can be used. This package is included in the R base installation. To assess the measurement system, the variability must be split into the elements shown in Fig. 5.1. The Gage R&R variability is due to the inherent randomness of the process, and it can be split into repeatability and reproducibility. The part-to-part variability is the variability element that is not due to the measurement system.

An ANOVA table provides all the information we need to compute all of the components mentioned in Fig. 5.1. In Chap. 9, a more thorough explanation of ANOVA will be given.

Let us consider an ANOVA table with two factors, A and B, corresponding respectively to parts and appraisers. The ANOVA table provides variability information for both factors A and B, for the interaction and for the so-called residuals of the model. This variability information is divided into sum of squares, mean of squares, and information regarding statistical tests.

Example 5.2 (Batteries (cont.)). With the following command we obtain an ANOVA table for the battery measurements:

```
> anova(lm(voltage ~ battery + voltmeter +
        battery * voltmeter,
    data = batteries))
```

```
Analysis of Variance Table

Response: voltage
                  Df   Sum Sq  Mean Sq F value Pr(>F)
battery            2 0.063082 0.031541  1.9939 0.1788
voltmeter          1 0.044442 0.044442  2.8095 0.1195
```

```
battery:voltmeter  2 0.018472 0.009236  0.5839 0.5728
Residuals         12 0.189821 0.015818
```

□

Repeatability can be obtained directly from the table as the residual mean square. The rest of the components of the total variability in Fig. 5.1 can be calculated as follows:

$$
\begin{aligned}
\sigma^2_{Appraiser} &= \frac{MSB - MSAB}{a \times n},\\
\sigma^2_{Interaction} &= \frac{MSAB - MSE}{n},\\
\sigma^2_{Reproducibility} &= \sigma^2_{Appraiser} + \sigma^2_{Interaction},\\
\sigma^2_{Gage\ R\&R} &= \sigma^2_{Repeatability} + \sigma^2_{Reproducibility},\\
\sigma^2_{Part\ to\ Part} &= \frac{MSA - MSAB}{b \times n},\\
\sigma^2_{Total} &= \sigma^2_{Gage\ R\&R} + \sigma^2_{Part\ to\ Part},
\end{aligned}
$$

where a and b are the number of levels of each factor, n is the number of replicated measures, and MSA, MSB, MSAB, and MSE correspond respectively to the mean of squares of A and B, the interaction between factors (AB), and the error. If any component is negative, it must be taken as zero ([94]).

Example 5.3 (Battery voltage (cont.)). Repeatability can be measured directly from the residual mean squares in the ANOVA table (third column, fourth row), so that $\sigma^2_{Repeatability}$ is

```
> anova(lm(voltage ~ battery + voltmeter +
        battery * voltmeter,
    data=batteries))[3][4,1]
```

```
[1] 0.01581842
```

The appraiser variability (in this case, due to the voltmeter) $\sigma^2_{Appraiser}$ is

```
> (0.0444 - 0.0092) / (3*3)
```

```
[1] 0.003911111
```

The variability due to the interaction between the battery and the voltmeter $\sigma^2_{Interaction}$ is

```
> (0.0092 - 0.0158) / 3
```

```
[1] -0.0022
```

This value should not be negative, and therefore we take it as zero. Thus the reproducibility $\sigma^2_{Reproducibility}$ amounts to:

```
> 0.0039 + 0
```

```
[1] 0.0039
```

The gage R&R variation $\sigma^2_{Gage\ R\&R}$ takes the value

```
> 0.0158 + 0.0039
```

```
[1] 0.0197
```

The part-to-part variation $\sigma^2_{Part\ to\ Part}$ will be

```
> (0.0315 - 0.0092) / (2 * 3)
```

```
[1] 0.003716667
```

Finally, the total variation σ^2_{Total} is

```
> 0.0037 + 0.0197
```

```
[1] 0.0234
```

□

5.2.3 *Assessing the Measurement System*

For a measurement system to be accurate, the contribution of the gage R&R variability to the total variability should be lower than 10%. A value between 10 and 30% may be acceptable. A value larger than 30% represents a bad measurement system.

To evaluate this fact, we use the square roots of the variabilities, that is, the standard deviations. We will describe this concept in detail in Chap. 9.

The contribution of each source of variation is then computed over the total Study Var:[2]

$$\%Gage\ R\&R\ Variation = \frac{\sigma_{Gage\ R\&R}}{\sigma_{Total}}.$$

Another useful metric is the *number of distinct categories*. It is calculated using the formula

$$N^o\ Cat = \frac{\sigma_{Part\ to\ Part}}{\sigma_{Gage\ R\&R}} \times 1.41$$

[2]The study var is defined as the standard deviation of each source of variation, multiplied by 5.15 (5.15 standard deviations cover 99% of data).

and rounding to the nearest lower integer (1 if lower than 1). The number of distinct categories should be greater than or equal to four ([69]). This value measures the relationship between the variability due to the measurement system and the inherent variability. If it is lower than 4, then the gage R&R variability is large compared to the inherent variability. Otherwise, the relationship between both variabilities can be considered adequate.

Example 5.4 (Battery voltage (cont.)). The contribution of the Gage R&R variability to the overall variation of the process *%Gage R&R Variation* is as follows:

```
> (sqrt(0.0197) / sqrt(0.0234)) * 100
```

```
[1] 91.75404
```

This means that most of the variation of the process is due to the gage R&R variability. The number of distinct categories *N^o Cat* is

```
> (0.0037 / sqrt(0.0197)) * 1.41
```

```
[1] 0.03716959
```

As was mentioned above, we take this value to be 1. These results confirm the inadequacy of the current measurement system. □

5.3 Using the `SixSigma` Package

The graphics and the numeric computations are collected in the `ss.rr` function of the `SixSigma` package. The syntax of the function is

```
> ss.rr(var, part, appr, data, main, sub)
```

It accepts the arguments in Table 5.2. After running it, you obtain a text output with an ANOVA table, the details of the sources of variance, and some graphics, in particular:

- Bar chart for contribution percentage of each variation source,
- Measured values by appraiser plot,

Table 5.2 Arguments for `ss.rr` function

Parameter	Description
`var`	Measured variate
`part`	Factor for parts
`appr`	Factor for appraisals (operators, machines, ...)
`data`	Data frame containing variates
`main`	Main title of graphic output
`sub`	Subtitle of graphic output (the recommended project name)

- Measured values by part plot,
- Mean measured values by part and appraiser,
- Mean control chart for R&R study,
- Range control chart for R&R study.

Example 5.5 (Battery voltage (cont.)). The text and graphic output for the battery data can be obtained and saved in an R object using the following command:

```
> my.rr <- ss.rr(var = voltage, part = battery,
        appr = voltmeter,
        data = batteries,
        main = "Six Sigma Gage R&R Measure",
        sub = "Batteries Project MSA")
```

```
Analysis of Variance Table

Response: var
              Df   Sum Sq  Mean Sq F value Pr(>F)
part           2 0.063082 0.031541  1.9939 0.1788
appr           1 0.044442 0.044442  2.8095 0.1195
part:appr      2 0.018472 0.009236  0.5839 0.5728
Repeatability 12 0.189821 0.015818

Gage R&R
                    VarComp %Contrib
Total Gage R&R    0.0197301    84.15
  Repeatability   0.0158184    67.46
  Reproducibility 0.0039117    16.68
    appr          0.0039117    16.68
    part:appr     0.0000000     0.00
Part-To-Part      0.0037174    15.85
Total Variation   0.0234476   100.00

                      StdDev   5.15*SD %StudyVar
Total Gage R&R    0.14046387 0.7233890     91.73
  Repeatability   0.12577122 0.6477218     82.14
  Reproducibility 0.06254358 0.3220995     40.84
    appr          0.06254358 0.3220995     40.84
    part:appr     0.00000000 0.0000000      0.00
Part-To-Part      0.06097048 0.3139980     39.82
Total Variation   0.15312609 0.7885994    100.00

Number of Distinct Categories= 1
```

The results of the MSA show that the %StudyVar due to R&R (second table, first row, third column) is 91.73%, larger than 30%. In addition, the number of distinct categories is equal to 1. This small number of categories, together with such a large percentage of variability, is the worst possible result for a measurement system.

To find out where the problem with the measurement system is, we can use some graphical tools. These tools will be explained in detail in Chap. 8. Figure 5.2 shows the charts that will help us to identify the causes of the problem. □

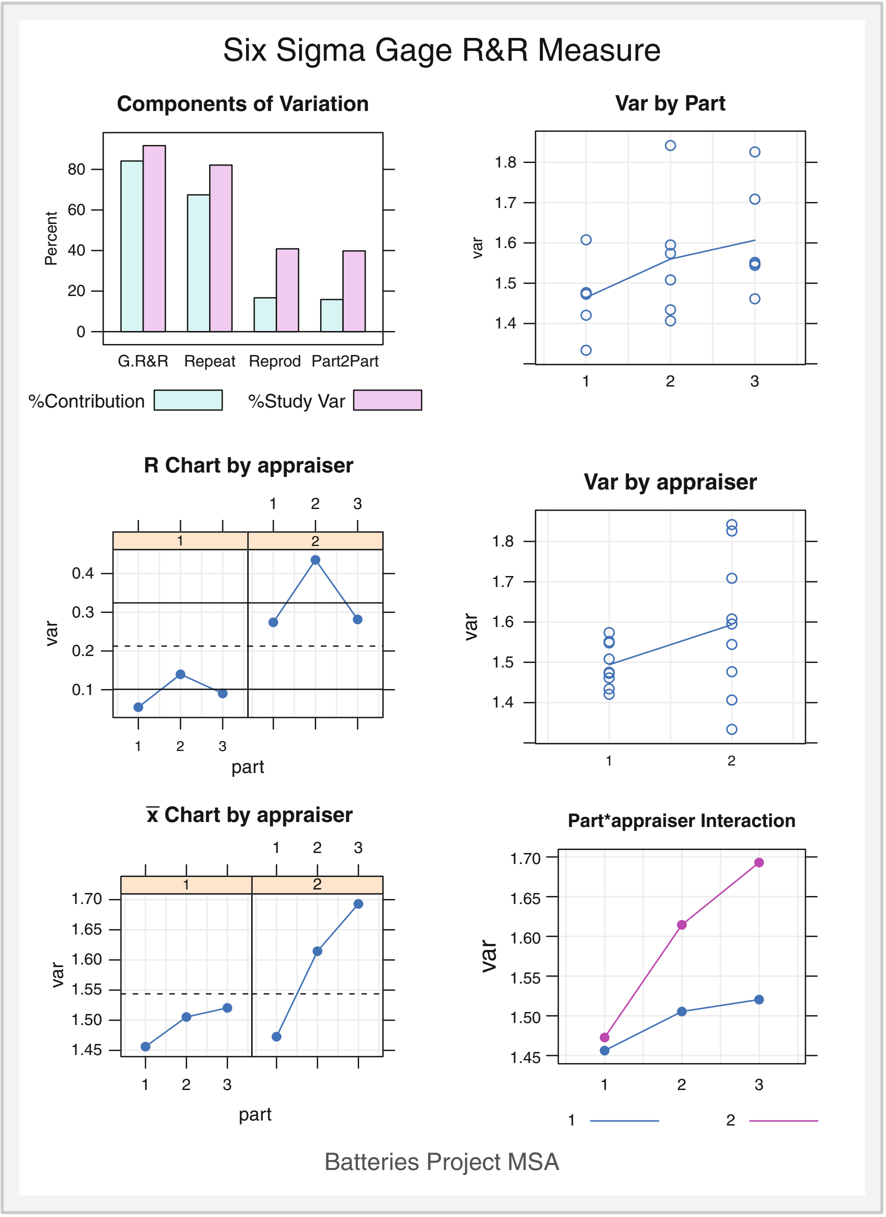


Fig. 5.2 Charts for MSA of battery example. In the top left plot we see graphically that our measurement system is in trouble

5.3.1 Interpreting the Charts

A **bar plot** can be generated to see the contribution of each component to the total variance. Thus we will be able to detect at a glance if the measurement problems come from the repeatability or reproducibility.

Example 5.6 (Battery voltage (cont.)). The plot in the top left in Fig. 5.2 is a bar chart representing the contribution of each component to the total variance. It is intended to detect at a glance if the measurement problems come from the repeatability or the reproducibility. In this case, it is clear that the process has problems with the repeatability and the reproducibility values, as their contribution is in both cases larger than 10%, with repeatability being approximately twice reproducibility. □

Using **strip plots** and **line plots** representing the effects, we can see if the difference between appraisers is the problem or if the interaction between appraisers and parts is important.

Example 5.7 (Battery voltage (cont.)). The first two plots in the right column in Fig. 5.2 show every measure as a point in the graph. In the top plot, the x-axis represents the batteries, and a line has been plotted linking the means of each prototype. In the middle plot, the x-axis represents the voltmeters. The bottom plot shows the interaction between the two factors operators and prototypes. The means of the pairs battery $\times$ voltmeter are represented by points and linked by lines.

With these plots we can detect if there is any interaction between operators and prototypes (which would be a problem) or differences between operators. In the case at hand, we detect that the first battery seems to have a lower voltage than the rest, but its variability is similar to that of the others, so this is not a problem for the measurement system. However, the plot by appraiser indicates apparent differences between the voltmeters. Means and variabilities are different, and this is a problem for our measurement system. The lines in the interaction plot do not cross each other, so interaction between factors is irrelevant. □

Finally, **control charts** can be plotted by group and with the control limits adapted for R&R studies. For the *mean control chart*, the center line and the limits are:

$$\begin{aligned}
\mathit{Center\ Line} &= \bar{\bar{x}},\\
\mathit{Upper\ Limit} &= \bar{\bar{x}} + \frac{3}{d_2\sqrt{n}}\overline{R},\\
\mathit{Lower\ Limit} &= \bar{\bar{x}} - \frac{3}{d_2\sqrt{n}}\overline{R};
\end{aligned}$$

and for the *range control chart*:

$$
\begin{aligned}
\textit{Center Line} &= \overline{R},\\
\textit{Upper Limit} &= \overline{R} \times \left(1 + \frac{d_3}{d_2}\right),\\
\textit{Upper Limit} &= \overline{R} \times \left(1 - \frac{d_3}{d_2}\right),
\end{aligned}
$$

where d_2 and d_3 are the famous Shewart's constants to construct control charts ([69]), $\overline{\overline{x}}$ is the overall average, and $\overline{R}$ is the average range. Control charts will be tackled in Chap. 12.

Within a gage R&R study, most points in the $\overline{x}$ chart should be outside the control limits. This is due to the fact that the plot represents part-to-part variation (same operator, same prototype), while the limits apply to the overall data. Otherwise, the measurement system would be considered inadequate. However, in the range control chart, all of the points should be inside the control limits.

Example 5.8 (Battery voltage (cont.)). In Fig. 5.2, all the points in the $\overline{x}$ control chart are within the limits. When the limits are not drawn, it is because they are outside the chart's margins. In addition, several points in the range control chart lie outside the limits. These two facts confirm that the measurement system is not working properly.

It is apparent that our measurement system is not working properly. The main problem is the voltmeters (appraisers). Thus, before proceeding with the Six Sigma project, the Black Belt should calibrate the voltmeters, or just replace them and repeat the MSA, until the measurement system becomes correct. □

5.4 Summary and Further Reading

Measurement is not a trivial question in quality. If we are making decisions about data that have been wrongly measured, we may obtain inadequate results. So it is very important to conduct an MSA as a preliminary stage in any Six Sigma project. Once the measurement system is working properly, we will be able to analyze the data. In this chapter, we explained how to perform an MSA using several statistical tools. We used functions both in the base installation of R and in the `SixSigma` package and explained how to obtain the facts and figures through a battery factory example. The `qualityTools` package, [88], also contains functions for MSA.

Readers interested in the standards and rules for MSA may consult [110] and [42]. [43] is a more generic standard. For an in-depth discussion of MSA, [2] is a practical guide. For foundations and more academic explanations, see [69]. [3] dedicates an easy-to-follow chapter to MSA. A book dedicated to gage R&R studies is the one by [12]. In [101], some practical extensions to gage R&R studies are proposed.

Table 5.3 Practice: Compound in a pastry batch

	Lab	Batch	Run	Comp
1	1	1	1	0.2616
2	1	1	2	0.2511
3	1	2	1	0.2821
4	1	2	2	0.2790
5	1	3	1	0.2384
6	1	3	2	0.2368
7	2	1	1	0.2462
8	2	1	2	0.2561
9	2	2	1	0.2791
10	2	2	2	0.2802
11	2	3	1	0.2343
12	2	3	2	0.2351
13	3	1	1	0.2484
14	3	1	2	0.2435
15	3	2	1	0.2753
16	3	2	2	0.2809
17	3	3	1	0.2385
18	3	3	2	0.2407

Case Study

Make four different prototypes of the paper helicopter. Use the `ss.heli` function to retrieve the template. Ask three friends or colleagues to help act as operators (appraisers). Take three measurements of the flight time with each prototype and each operator. Remember to randomize the way you take the measurements. Save the data in a data frame, and run the `ss.rr` function to obtain the plots and facts of the MSA. Is it correct? Repeat the measurements until the MSA reflects only significant variation due to the parts.

Practice

5.1. The data in Table 5.3 are from a bakery that makes several types of pastries.[3] Customer satisfaction is related to the amount of a given compound in the chocolate used. To evaluate the measurement system, three laboratories are used to determine the amount of the compound in the pastries. Identify the appraisers, the parts, and their lengths. How many measurements were taken from each batch?

5.2. Using the bakery data in Table 5.3, conduct an MSA and record your conclusions.

[3]It is available in the SixSigma package as a data object called `ss.data.pastries`. You can save the data in a data frame yourself.

Chapter 6
Pareto Analysis with R

Causa latet: vis est notissima. [The cause is hidden, but the result is known.]

Ovid

6.1 Introduction

Pareto analysis is a technique that can be used in several stages of a Six Sigma project. In the Measure phase of the design, measure, analyze, improve, and control (DMAIC) cycle, we use it to prioritize the possible causes of defects and then focus on the important ones.

The basis of Pareto analysis is the Pareto principle, which applies to many processes in real life. Roughly speaking, the Pareto principle states that most effort/benefit (approximately 80%) is due to a limited number of key actions (approximately 20%). It is also known as the 80/20 rule. A search for these key actions is usually made using a Pareto chart, a tool that allows us to see at a glance the results of a Pareto analysis.

In this chapter, we review the foundations of Pareto analysis in Sects. 6.2 and 6.2.1. In Sect. 6.3, we apply the Pareto principle to detect important improvement opportunities in a Six Sigma project. We use R to plot Pareto charts in Sect. 6.4. Finally, in Sect. 6.5 we introduce other uses of the Pareto principle within Six Sigma projects.

6.2 Pareto Principle

Vilfredo Pareto (1848–1923) was an Italian economist whose most famous contribution was the principle known by his name. He was also a philosopher, engineer, sociologist, and political scientist. The Pareto principle was a result of Pareto's observations about the distribution of wealth in the 19th century.[1] He observed that 80% of the wealth was owned by 20% of the population.[2] Hence, the Pareto principle is also known as the 80/20 rule. It appears in many real-life situations, and therefore it is sometimes considered a natural principle. For example:

- 80% of the profits comes from 20% of the customers.
- 20% of the employees do 80% of the work.
- 20% of patients use 80% of care resources.
- 80% of cost of quality is produced by 20% of the sources of error.

The last example corresponds to the main application of the Pareto principle in Six Sigma. We will explain how to use it in the following sections.

6.2.1 Pareto Principle as a Problem-solving Technique

The Pareto principle is used to prioritize the actions to take from a given list of options. In this sense, the Pareto chart is an eye-opener. You can see at a glance where the best results are to be achieved if you focus on "the vital few rather than on the *trivial* many." This expression was coined by Joseph M. Juran and remains in the literature and jargon on quality. However, Juran himself confessed that those "many" may be nontrivial, and he preferred the expression "the vital few and the *useful* many" [47]. Thus, we should focus on the trivial few but not dismiss the useful many in subsequent approaches to the problem.

In problem-solving methods, identification of the root causes of a problem is critical to finding robust solutions (see, for instance, Fig. 3.1 in Chap. 3). To identify the possible causes, we can use several techniques (e.g., brainstorming) and tools (cause-and-effect diagrams, affinity diagrams, etc.).

Once the possible causes have been identified, selection of the critical ones can be done in various ways. Following the Pareto Principle, the natural result should be an 80/20 distribution of the causes. If we do not have an 80/20 distribution, we should rearrange our data by grouping or splitting the causes depending on the distribution we have reached (Fig. 6.1).

This is actually the challenge of Pareto analysis: selection of the main characteristics that lead to measuring the relevance of a problem. In the next section, we will show different methods of conducting a Pareto analysis.

[1]In those days, wealth was measured as the land an individual owned.

[2]Nowadays this relationship is nearly 99% and 1%, respectively.

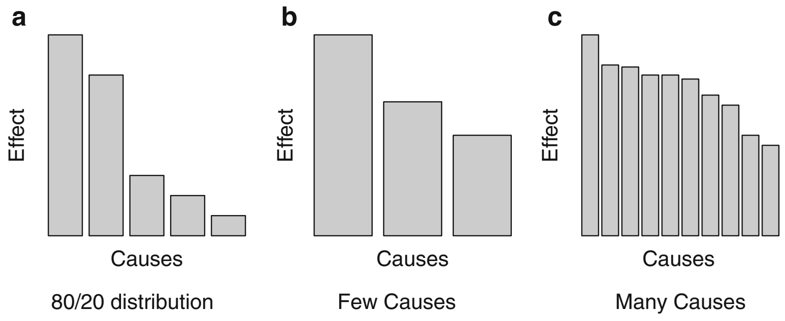


Fig. 6.1 Distribution of causes in a Pareto analysis. The distribution on the left approximates an 80/20 distribution. The center distribution needs to split the causes to focus on the important ones. The distribution on the right needs to group causes

6.3 Pareto Analysis in Six Sigma

The Pareto chart is considered one of the seven basic tools for quality control:

1. Histogram
2. Check sheet
3. Pareto chart
4. Cause-and-effect diagram
5. Defect concentration diagram
6. Scatter diagram
7. Control chart

They are also known as the *magnificent seven* tools of statistical process control (SPC) or the seven QC (quality control) tools. Quality was mainly developed during the 20th century in Japan, where number seven has a special meaning ([99]). For Pareto analysis we will use another of the magnificent seven jointly with the Pareto chart: the cause-and-effect diagram.

6.3.1 Identifying Causes

If we want to improve a process, the first thing we must do is identify where we can improve it. Eventually the process will be improved by reducing the variability. As a result, we will obtain a process with a reduced number of defects and reduced costs.

The current situation within the DMAIC cycle is as follows. We have described the process with the process map, and we have identified the parameters (x) that influence the features of the process (y). Thus these parameters can engender problems with the process. Now, with the support of the process map and other

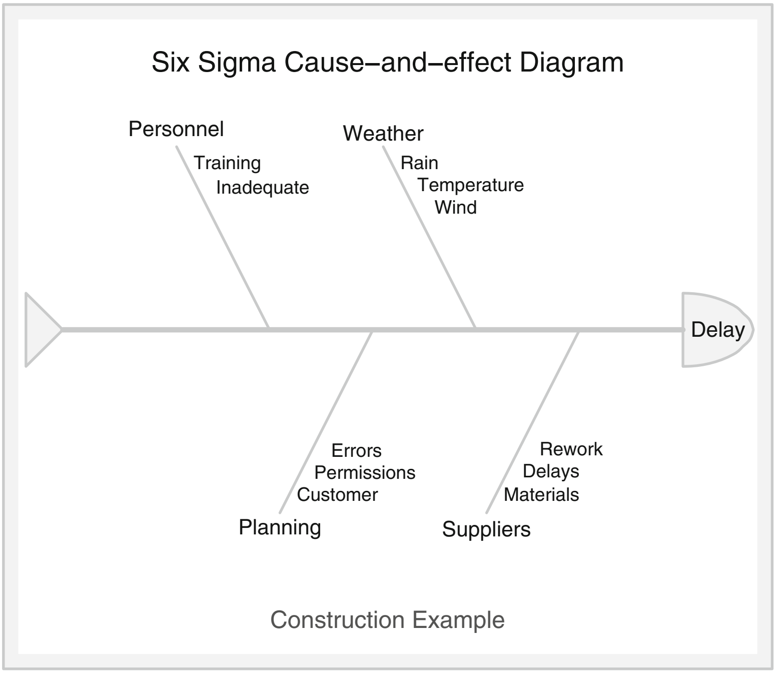


Fig. 6.2 Cause-and-effect diagram for construction example. In the head of the fish we have the effect we want to investigate. Each main fishbone represents a group. An individual cause stems from the main fishbone

suitable techniques (brainstorming, six Ms, five whys, affinity diagrams, etc.), we can identify the causes of errors that may arise in the process. As a result, a list of causes is created. We can group the causes into categories on several levels. The causes are then arranged in a *fishbone* shape, with the causes recorded in the fish bones and the effect (usually the CTQ characteristic) in the head of the fish. This will be the so-called cause-and-effect diagram also known as the fishbone diagram or the Ishikawa diagram, after its originator, Kaoru Ishikawa.

Example 6.1 (Construction project). In the construction of a building, a CTQ characteristic might be the fulfillment of a deadline, as unfulfillment can lead to failure.

Using the techniques such as those described previously, the Six Sigma team identified the following events that can cause a delay in the schedule: weather, errors in planning, delay of suppliers, inadequate operators, customer specifications/delays, defects in materials, and permissions.

A cause-and-effect diagram allows us to arrange the information in such a way as to make it easier to interpret. We can create a cause-and-effect diagram with the `ss.ceDiag` function of R's `SixSigma` package (Fig. 6.2):

```
> b.effect <- "Delay"
> b.groups <- c("Personnel", "Weather",
        "Suppliers", "Planning")
> b.causes <- vector(mode = "list",
        length = length(b.groups))
> b.causes[1] <- list(c("Training", "Inadequate"))
> b.causes[2] <- list(c("Rain", "Temperature", "Wind"))
> b.causes[3] <- list(c("Materials", "Delays",
                "Rework"))
> b.causes[4] <- list(c("Customer", "Permissions",
                "Errors"))
> ss.ceDiag(b.effect, b.groups, b.causes, sub = "Construction
    Example")
```

Type `?ss.ceDiag` to learn more about the `ss.ceDiag` function. □

6.3.2 *Measuring the Effect*

Now that we have identified the possible causes of the problem, we need to measure the effect.

A first approach is to consider two easy and intuitive dimensions of the problem: the number of errors and the cost of the errors. Depending on the process being used, the data can derive from historical data, sampling, or even simulation.

Counting the number of errors arising from a cause is easy, but it is not enough. We may have a large number of defects that are unimportant to customers or do not cause a significant increase in the cost of the process. Thus, the selection of the measurement units, and the measurement itself, is a critical task in Pareto analysis.

Example 6.2 (Construction project (cont.)). The Black Belt in the construction company has investigated why a sampling of deadlines on projects developed in the last 2 years went unfulfilled.

He has also estimated the cost of these delays for the company (larger labor force, extra payments, etc.).

We will save the data in a data frame with a factor whose levels are the possible causes (we use the `b.causes` list created previously to draw the cause-and-effect diagram), and with two variables, (namely: number of unfulfilled deadlines and estimated cost).

```
> b.data <- data.frame(cause=factor(unlist(b.causes)),
        count = c(5,1,3,1,2,18,20,4,15,2,4),
        cost = c(50,150,50,10,20,180,200,10,5,20,150))
```

□

6.3.3 Building a Pareto Chart

The Pareto chart is a bar chart representing the causes on the *x*-axis and the effects on the *y*-axis, where the bars are sorted in descending order by magnitude of the effect. To make the chart easier to interpret, some enhancements are usually included in such charts:

1. A secondary *y*-axis on the left from 0 to 100, representing the cumulative percentage of the effect measurement.
2. A line chart linking the cumulative percentages of each cause.
3. Auxiliary lines from the axis identifying the 80/20 rule, that is, which causes are responsible for 80% of the effect.

To summarize, let us create a step-by-step algorithm to create our Pareto chart:

1. Identify the causes.
2. Choose the appropriate measurement units.
3. Obtain the data.
4. Sort by importance.
5. Figure the cumulative percentage.
6. Plot a bar chart for the measurements.
7. Plot a line chart for the cumulative percentages.
8. Find the causes responsible for 80% of the effect.

In the next section, we will demonstrate how to build a Pareto chart using R. Then we will focus on the selected causes and show how to use it for the improvement of the process to reduce errors.

6.4 Pareto Charts in R

As a Pareto chart is essentially a sorted bar chart, it is easy to plot it with R using the standard `graphics` package.

A straightforward way to create a Pareto chart for Six Sigma projects is by using the specific functions in the `qcc` and `qualityTools` packages.

Example 6.3 (Construction project (cont.)). Let us plot our Pareto chart with data from the construction example using R. First, we can plot a single bar plot (Fig. 6.3) and include the cumulative percentages as an annotation. The `barplot` function returns a vector with the position of the bars on the *x*-axis. We use this vector to print the values of the cumulative percentage in the appropriate spot on the chart.

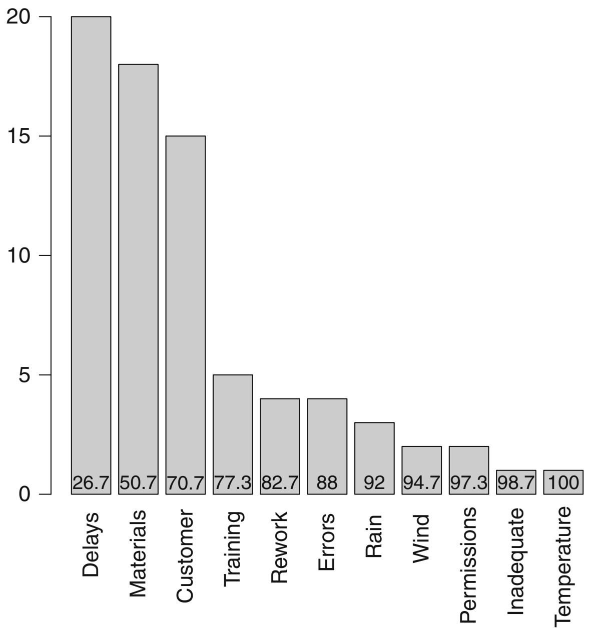


Fig. 6.3 Pareto chart with base graphics. Ordering the values in the data frame makes it easy to plot a Pareto chart with the `barplot` function. We can annotate the plot with the cumulative percentages using the `text` function

```
> pChart <- barplot(rev(sort(b.data$count)),
        names.arg = b.data$cause[order(b.data$count,
                        decreasing = TRUE)],
        las = 2)
> text(pChart,
    rep(0.5,11),
    sort(round(cumsum(100 * (b.data$count/sum(b.data$count))[
        order(b.data$count, decreasing = TRUE)]), 1)))
```

The `qcc` package includes the `pareto.chart` function (Fig. 6.4). It provides a text output with the data.

```
> library(qcc)
> b.vector <- b.data$count
> names(b.vector) <- b.data$cause
> pareto.chart(b.vector, cumperc = c(80))
```

The `qualityTools` package includes the `paretoChart` function to plot Pareto charts (Fig. 6.5).

```
> require(qualityTools)
> paretoChart(b.vector,
    las = 2,
    percentVec = c(0, 0.5, 0.80, 1))
```

```
Pareto Analysis for b.vector
---

Frequency              20     18     15      5      4      4
```

```
Cum. frequency      20    38    53    58    62    66
Percentage       26.7% 24.0% 20.0%  6.7%  5.3%  5.3%
Cum. percentage  26.7% 50.7% 70.7% 77.3% 82.7% 88.0%

Frequency            3     2     2     1      1
Cum. frequency      69    71    73    74     75
Percentage        4.0%  2.7%  2.7%  1.3%   1.3%
Cum. percentage  92.0% 94.7% 97.3% 98.7% 100.0%


Frequency        20.00000 18.00000 15.00000  5.000000
Cum. frequency   20.00000 38.00000 53.00000 58.000000
Percentage       26.66667 24.00000 20.00000  6.666667
Cum. percentage  26.66667 50.66667 70.66667 77.333333

Frequency         4.000000  4.000000  3  2.000000
Cum. frequency   62.000000 66.000000 69 71.000000
Percentage        5.333333  5.333333  4  2.666667
Cum. percentage  82.666667 88.000000 92 94.666667

Frequency         2.000000  1.000000   1.000000
Cum. frequency   73.000000 74.000000  75.000000
Percentage        2.666667  1.333333   1.333333
Cum. percentage  97.333333 98.666667 100.000000
```

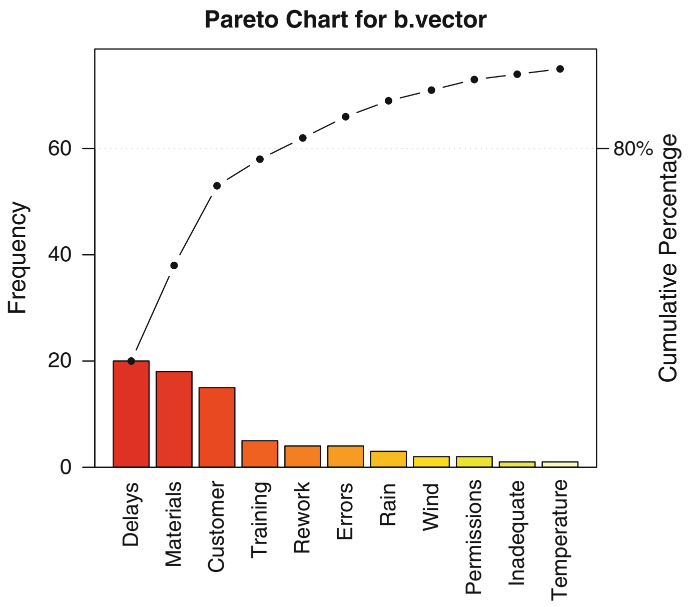
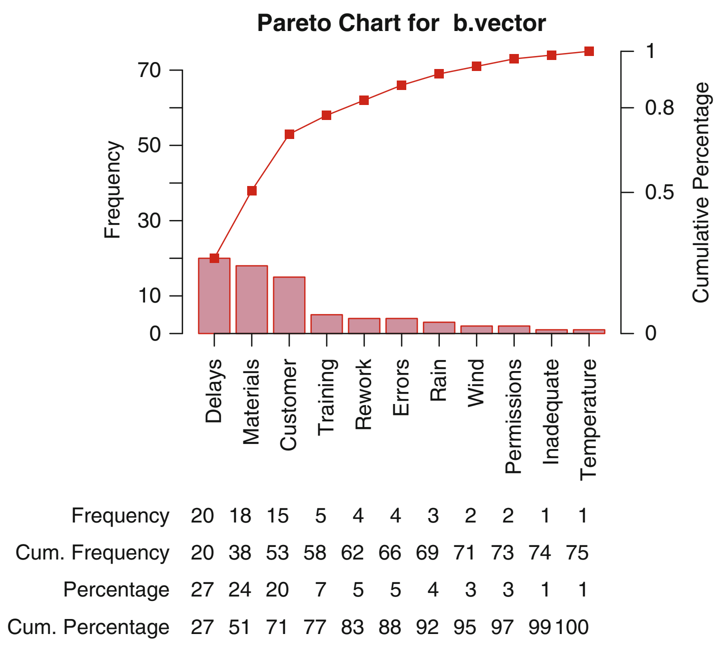


Fig. 6.4 Pareto chart in `qcc` package. You can choose a palette of colors and the *horizontal lines* for the cumulative percentage



Frequency	20	18	15	5	4	4	3	2	2	1	1
Cum. Frequency	20	38	53	58	62	66	69	71	73	74	75
Percentage	27	24	20	7	5	5	4	3	3	1	1
Cum. Percentage	27	51	71	77	83	88	92	95	97	99	100

Fig. 6.5 Pareto chart in `qualityTools` package. You can choose the ticks for the right axis (cumulative percentages). It plots a table below the chart with the values and the percentages

Both functions require a numeric vector with labeled components, instead of a data frame. See the documentation for the packages to find out more about these functions. □

6.5 Other Uses of the Pareto Chart

As was mentioned in Sect. 6.2.1, the Pareto principle is a tool for solving problems. Thus, we can use the Pareto chart anywhere in our project to improve our processes.

In the initial stages of a Six Sigma implementation, we may have to select a project from a list of available projects. Some authors even transform the DMAIC cycle into the SDMAIC strategy, where S stands for Select. Therefore, selecting a project is an important task for a Master Black Belt. To make this selection, we can use a Pareto chart, choosing the appropriate measurement units, e.g., expected cost reduction, and focusing on the vital few projects that will allow us to achieve 80% of the savings.

Another stage where we will probably use a Pareto chart will be during the Improve phase. In the initial steps of the design of an experiment, we usually measure many possible effects, and a Pareto chart is one of the techniques used to select the important ones when we do not have enough data to conduct, for instance, a hypothesis test.

6.6 Summary and Further Reading

In this chapter, we explained the foundations of a simple and useful tool in Six Sigma: the Pareto chart. It can be used when we have to prioritize a list of elements, such as the causes of failure in our process, in order to differentiate between the "vital few" (20%) and the "useful many" (80%). In conjunction with the Pareto chart, we used the cause-and-effect diagram to identify the possible causes of error.

We plotted Pareto charts with R through base and contributed packages. Other uses of the Pareto chart were also presented.

More advanced options beyond the scope of this book include the use of `lattice` ([91]) and `ggplot2` ([105]) graphics, which allow for the customization of Pareto charts using annotations, multiple charts, etc.

Examples of the 80/20 rule in real life can be found in the philosophical guide [55]. The works by Ishikawa can be consulted in [41]. The magnificent seven are mentioned in many quality-control books, e.g., [69] or [99]. [47] is the sixth edition of a classic text by the founder of the Pareto principle for quality control.

Case Study

Think about the helicopter project, and try to list the factors that can cause defects in the paper helicopter, leading to worse flight times.

Group the causes and create a cause-and-effect diagram. You can use just paper and pencil, but once you have finished with it, try to plot it with R.

If you have time, try to measure the errors making lots of helicopters with different conditions (different operators, paper, wind, etc.). You can also invent or simulate the data.

Save the data into an R data frame and plot a Pareto chart. Analyze the results, and figure out where you will have to focus your attention to produce optimal improvements.

Practice

6.1. In the construction example, we plotted a Pareto chart by number of defects. Plot the chart for cost of error.

6.2. Select the causes you must focus on to improve the construction process. Can you select them from only one of the charts?

Chapter 7
Process Capability Analysis with R

One cannot develop taste from what is of average quality but only from the very best.

Johann Wolfgang von Goethe

7.1 Introduction

Capability analysis is a very important part of the Measure phase in a Six Sigma project. It is also part of classical statistical process control. Through capability analysis we can measure how the process performance fits the customer/client requirements. These requirements must be translated into the specification limits for the characteristic of interest, and they can be one-sided (only upper or lower limit) or two-sided (upper and lower limits). Over the years, many advances have been made, especially with respect to nonnormality, nonlinearity, processes with multiple characteristics, and many others. Variation (short-term and long-term), yield, DPU, DPO, DPMO, DPPM, RTY, and Z (the sigma score of the process) are used to measure process performance.

In this chapter, we will go over specifications (Sect. 7.2), process performance (Sect. 7.3), their relationship (Sect. 7.4), and the capability indices (Sect. 7.5). We will illustrate the concepts through a practical example.

7.2 Specifications

The specifications are the voice of the customer.[1] Under a Six Sigma paradigm, the process must fulfill the customer requirements. These requirements must be

[1]Note that the customer can be external or internal, a person, or another process within the organization.

quantified in some way to be attainable. Sometimes the specifications are defined by the customer, and sometimes not. In this case, the specifications are estimated through marketing studies, know-how, or some other means.

The specification limits are the numerical expressions of the customer requirements. For example, we must produce bulbs that last at least 10,000 h, supply an order in less than 2 days, or answer a phone call before 4 rings.

The numbers mentioned in the previous paragraph are the *target* of the process. Because that is an ideal (unreachable because of the natural variations in the process), usually the specifications are a range with upper and lower bounds. For example, the length of a nail can be between 9.5 and 10.5 mm, or the service time at a counter cannot be more than 5 min. The upper and lower limits are abbreviated as *USL* and *LSL*, respectively (upper specification limit, lower specification limit). Thus, the USL is a value above which the process performance is unacceptable, and the LSL is a value below which the process performance is unacceptable.

The specifications must be realistic. A well-known method to evaluate the validity of the specifications in Six Sigma is RUMBA, which stands for:

R	Reasonable
U	Understandable
M	Measurable
B	Believable
A	Achievable

Figure 7.1 represents the variation of a characteristic against its specification limits.

7.3 Process Performance

Once we have realized what our process must achieve, we need to know what our process is doing. This is the so-called voice of the process (VOP). A simplification could be (and actually it has been for a long time) that the process is right when the current value of the characteristic is within the specification limits.

In the Six Sigma methodology, this approach is not enough. The process is right when we are approximating the *target*, with as little *variation* as possible. This can be easily understood under the concept of the Taguchi loss function (Chap. 4). As we see in Fig. 7.2, the further we move away from the target, the higher the cost, and so the worse the performance of our process. Thus, the position of the characteristic value within the limits is important. The dotted line is the cost of poor quality under the traditional approach. The dashed line is the cost of poor quality in the Six Sigma approach.

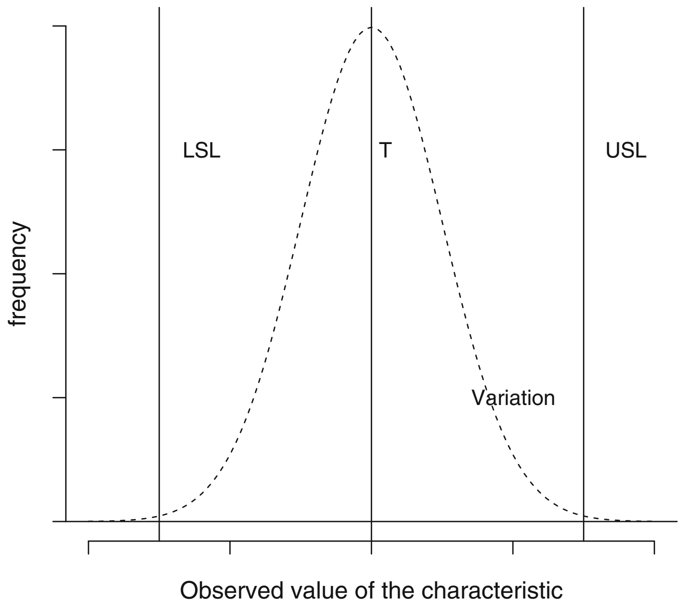


Fig. 7.1 Specification limits. The real value of a characteristic varies according to its probability distribution. The upper limit (USL), lower limit (LSL), and target (T) are the references to assess the capability of the process

Another important difference between the classical and the Six Sigma approach is the hidden factory concept. The *yield* (Y) of a process is the amount of "good stuff" produced by the process. It can be assessed once the process is finished, counting the items that fit the specifications:

$$Y = \frac{total - defects}{total}.$$

In Six Sigma, we must also take into account the rework in the middle of the process. Thus, regardless of the number of correct items at the end of the process, we count the correct items as "first time" correct items and calculate the first-time yield (FTY):

$$FTY = \frac{total - rework - defects}{total}.$$

The difference between Y and FTY is what the hidden factory is "producing".

Example 7.1 (Winery). In a winery, the key characteristic of the product is the volume in a bottle of wine. The target is 750 cl. The specification limits for the process are 740 (LSL) and 760 (USL) cl.

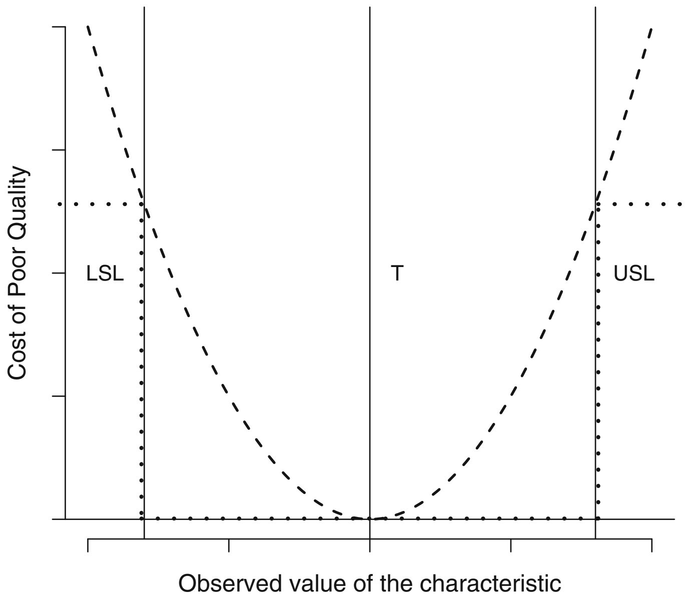


Fig. 7.2 Taguchi's loss function. The cost of poor quality rises as the observed value of the characteristic moves away from the target

Suppose that on a given day, the winery has bottled 1,915 bottles of wine, five of which were rejected just before packaging for being outside the specifications. The *yield* of the process, traditionally, is

```
> (1915 - 5) / 1915
```

```
[1] 0.997389
```

But after a review, the Six Sigma Black Belt finds out that during insertion of the corks, 12 bottles had to be processed again due to excess wine. So the *first-time yield* of the process is

```
> (1915 - 5 - 12) / 1915
```

```
[1] 0.9911227
```

Consequently, the first-time yield is lower than the *final* yield. □

When a process is formed by several linked processes, we calculate first the FTY of each individual process and then the *rolled throughput yield* (RTY) of the overall process. It is calculated by multiplying the FTY of every chained process:

$$RTY = \prod_{i=1}^{n} FTY_i = FTY_1 \times FTY_2 \times \cdots \times FTY_{n-1} \times FTY_n,$$

where n is the number of individual processes and the Greek letter pi (Π) stands for "multiply."

Example 7.2 (Winery (cont.)). In our winery, we have three processes in the filling line:

1. Fill bottle.
2. Insert cork.
3. Label bottle.

Let 3, 12, and 5 be the number of bottles outside of the specification limits in each subprocess. Then the *RTY* is:

```
> ((1915-3)/1915) * ((1915-12)/1915) * ((1915-5)/1915)
```

```
[1] 0.9895864
```

□

Defects are the complementary of yield. The simplest measurement for defects is *DPU* (defects per unit):

$$DPU = 1 - FTY = \frac{defects}{total}.$$

We assume that "unit" can refer to whatever we do: products, services, processes, operations, transactions, etc.

To be able to make comparisons, *DPO* (defects per opportunity) and its multiple *DPMO* (defects per million opportunities) are used. Thus, we can compare the defect rate of processes regardless of their complexity, for example, between a big server with 100 parts and a PC with 50:

$$DPO = \frac{number\ of\ defects}{number\ of\ opportunities},$$

$$DPMO = DPO \times 10^6.$$

Example 7.3 (Winery (cont.)). In the case of the winery, a possible unit to consider is the batch. If we consider the previous data as a batch, the DPU would be

```
> 3 + 12 + 5
```

```
[1] 20
```

As every batch may have a different size, we should use DPMO to make comparisons:

```
> ((3 + 12 + 5) / 1915) * 10^6
```

```
[1] 10443.86
```

Table 7.1 Arguments for `ss.ca.yield` function

Argument	Description
`defects`	Numeric vector with number of defects in each item/batch
`rework`	Numeric vector with number of items reworked in each item/batch
`opportunities`	Numeric value with number of opportunities

□

You can obtain all the performance measures mentioned above using the function `ss.ca.yield` included in the `SixSigma` package. It accepts the arguments in Table 7.1.

Example 7.4 (Winery (cont.)). We use the same data as was previously used for the arithmetic computation, adding a vector with the number of reworked bottles in each batch (1, 2, and 4).

```
> ss.ca.yield(c(3, 5, 12), c(1, 2, 4), 1915)
```

```
      Yield       FTY       RTY DPU     DPMO
1 0.9895561 0.9859008 0.9859563  20 10443.86
```

□

7.4 Process vs. Specifications

The *sigma score* of a process (Z), or simply the sigma, is the most famous metric for Six Sigma practitioners. So do not be surprised if you are asked for *the sigma* of your process. Actually, this simple number conveys how your process fits your customer specifications. A process with a sigma equal to 6 may be considered an "almost perfectly" designed process. This sigma value implies that less than 3.4 DPMO will be attained.

The sigma is the number of standard deviations that fit between the specification limit and the mean of the process. To calculate it, we need the specification limits (Sect. 7.2) and a *sample* of the key characteristic we are evaluating to estimate the mean, $\bar{x}$, and the standard deviation, σ. $\bar{x}$ and σ will be explained thoroughly in Chap. 9).

$$Z = \min\left\{\frac{(USL - \bar{x})}{\sigma}, \quad \frac{(\bar{x} - LSL)}{\sigma}\right\}.$$

Depending on which kind of standard deviation we use, we obtain the short-term (ST) or long-term (LT) sigma score (see Chap. 9 to understand the difference between short-term and long-term variation). We usually get the short-term and then estimate the long-term score as

Table 7.2 DPMO through sigma scores

Z_{LT}	DPMO
1	690,000
2	308,000
3	66,800
4	6,210
5	233
6	3.4

Table 7.3 Arguments for `ss.ca.z` function

Argument	Description
`x`	Vector with data of key process characteristic
`LSL`	Lower specification limit of process
`USL`	Upper specification limit of process
`LT`	Indicates if data are from long-term performance
`f.na.rm`	Indicates if NAs must be ignored

$$Z_{LT} = Z_{ST} - 1.5.$$

This long-term sigma score can be translated immediately in terms of DPMO using Table 7.2. The 1.5 value is a source of criticism in the literature because of its arbitrary (empirical) nature. In the original Six Sigma conception, it was assumed that even when the Six Sigma quality level is satisfied by a process, its mean may still suffer disturbances up to 1.5 standard deviations from the target value. It is just a way of stating that processes are not stable forever, and this behavior should be modeled somehow. This choice "has proven to be a useful way to think about process performance" [69]. Do not be shortsighted in this regard. As mentioned in Chap. 1, the importance of Six Sigma is that its stages of application (the DMAIC cycle) are supported by the scientific method.

Example 7.5 (Winery (cont.)). In the winery, the actual volume in a 20-bottle sample was measured after the filling process. The data are in Table 7.4. What is the sigma of this process?

The function `ss.ca.z` calculates the Z of the process. The function uses the parameters in Table 7.3. First we save our data in a vector (see Chap. 2 for other options for input data):

```
> x<-c(755.81, 750.54, 751.05, 749.52, 749.21, 748.38,
     748.11, 753.07, 749.56, 750.08, 747.16, 747.53,
     749.22, 746.76, 747.64, 750.46, 749.27, 750.33,
     750.26, 751.29)
> ss.ca.z(x,740,760)
```

```
[1] 3.139539
```

So our process has a sigma of 3.1395 (less than 66,800 DPMO). □

Table 7.4 Data for winery example

	Volume
1	755.81
2	750.54
3	751.05
4	749.52
5	749.21
6	748.38
7	748.11
8	753.07
9	749.56
10	750.08
11	747.16
12	747.53
13	749.22
14	746.76
15	747.64
16	750.46
17	749.27
18	750.33
19	750.26
20	751.29

7.5 Capability Indices

Capability indices directly compare the customer specifications with the performance of the process. They are based on the fact that the *natural limits* or *effective limits* of a process are those between the mean and ± 3 standard deviations. It can be shown that 99.7% of data are contained within these limits.

Figure 7.3 shows graphically the effective width of the process. Thus, the capability of a process (C_p) is calculated using the formula

$$C_p = \frac{USL - LSL}{6\sigma_{ST}}.$$

This formula does not allow us to check whether the process is centered in the mean (which is desirable). To deal with this issue, we use the adjusted capability index (C_{pk}):

$$C_{pk} = \min\left\{\frac{USL - \bar{x}}{3\sigma_{ST}}, \quad \frac{\bar{x} - LSL}{3\sigma_{ST}}\right\}.$$

Sometimes long-term indices are used instead of short-term ones. They are denoted by P_p and P_{pk}. We can compare these indices with the previous ones to identify improvement opportunities. For example, if $C_p = P_p$ and $C_{pk} = P_{pk}$, then the process is likely too biased.

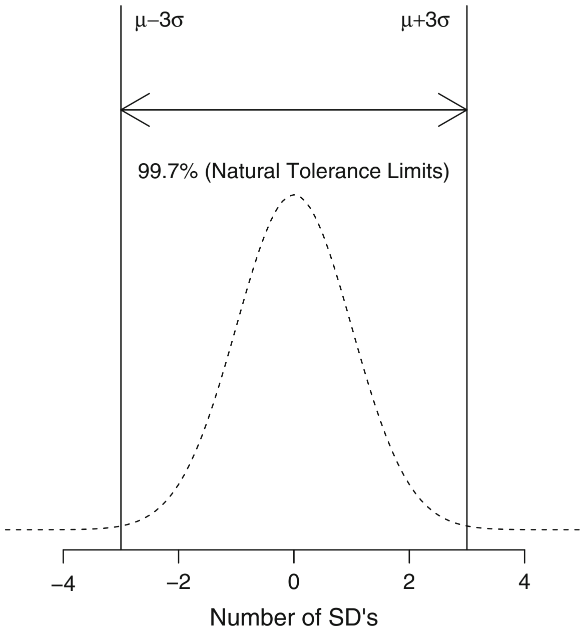


Fig. 7.3 Natural tolerance limits or effective limits. Between the mean and three standard deviations lie 99.7% of data

Like the sigma score, capability indices help us to determine how well a process is meeting customer specifications. In general, a C_{pk} of 1.33 is acceptable, but 1.67 is usually the goal.

The indices that we are dealing with are estimations based on the samples available. As such, they have a statistical variation that is highly influenced by the sample size. So it is considered a good practice to obtain confidence intervals for the indices. The formulas and bases for this can be found in [69].

Example 7.6 (Winery (cont.)). We are going to calculate the C_p and C_{pk} indices, and their confidence intervals, for the filling process. The `ss.ca.cp` and `ss.ca.cpk` functions perform the calculations to obtain the capability indices:

```
> ss.ca.cp(x,740, 760)
```

```
[1] 1.584136
```

```
> ss.ca.cpk(x,740, 760)
```

```
[1] 1.546513
```

So the process capability is acceptable, but it can be improved to reach the desired 1.67 value for C_{pk}. The confidence intervals are obtained when we add the parameter `ci` to the functions:

```
> ss.ca.cp(x, 740, 760, ci = TRUE)
```

```
[1] 1.084600 2.083046
```

```
> ss.ca.cpk(x, 740, 760, ci = TRUE)
```

```
[1] 1.033560 2.059466
```

□

7.6 Capability Study with `SixSigma` Package

Dissemination of the Six Sigma results is very important for practitioners, especially Master Black Belts, in order to spread the Six Sigma thinking throughout the organization. A summary of a capability analysis can be obtained in R with the function `ss.study` in the `SixSigma` package. It accepts the arguments in Table 7.5.

Example 7.7 (Winery (cont.)). Let us run the capability study for the winery example.

```
> ss.study.ca(x, LSL = 740, USL = 760,
    Target = 750, alpha = 0.5,
    f.su = "Winery Project")
```

Figure 7.4 is the graphical output of the function. The top chart is a histogram of the sample data, which includes the target and the specification limits. Density lines are plotted for the empirical and theoretical density functions.

The bottom chart is a quantile–quantile chart (Q–Q plot) to verify if the data are normally distributed. When they are, the points are approximately on a straight line. In addition, the most common numerical tests are shown as well. Normality is accepted when the p-value of the hypothesis test is larger than 0.05. We will describe hypothesis tests in detail in Chap. 10.

On the right-hand side of the figure, we have the facts of the analysis. The specification limits are those that we have entered in the function. The process

Table 7.5 Arguments for `ss.ca.study` function

Argument	Description
xST	Data of sample (short-term)
xLT	Data of long-term process performance
LSL	Lower specification limit
USL	Upper specification limit
T	Target
alpha	Type I error for confidence intervals
f.na.rm	Indicates if NAs must be ignored
f.main	Output title
f.sub	Output subtitle

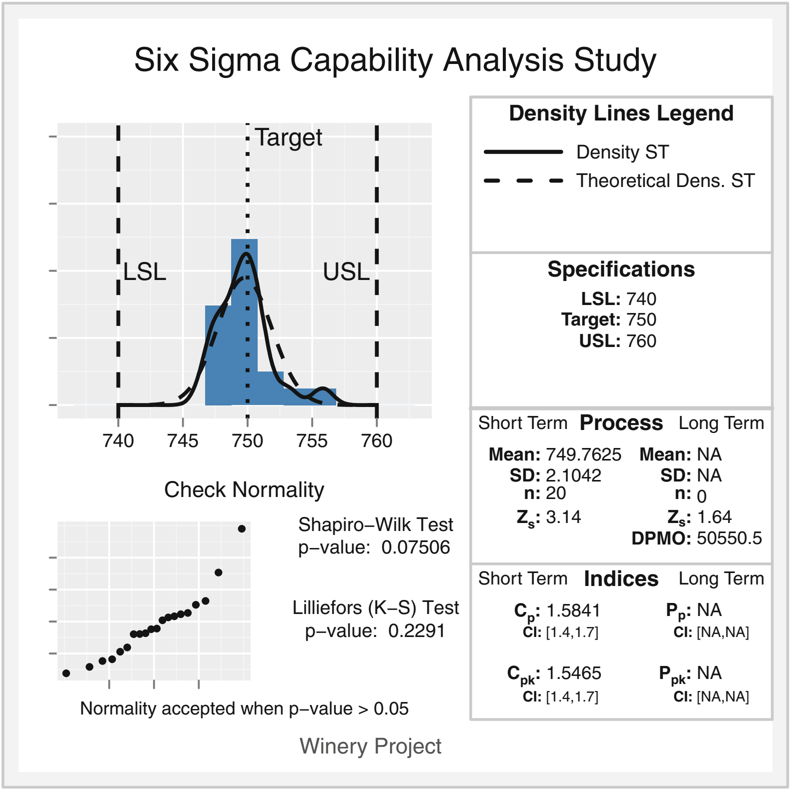


Fig. 7.4 Capability study output

performance and the indices are calculated with the short-term and long-term data provided. In this example, we have no long-term data. A confidence interval (CI) is calculated for the capability indices. We can see that the capability index (1.547) is quite acceptable, though it can be improved to reach the desired 1.67 value. Furthermore, the estimation of the long-term sigma indicates that we could have problems if the long-term variation turns out to be much higher than the short-term one. □

7.7 Summary and Further Reading

In this chapter, we explained the relationship between the specifications of a process and its performance. First, the specifications were defined. Then, the process performance was measured and some metrics calculated (yield, FTY, RTY, DPU, DPMO). And, finally, we put both together through the sigma of the process (Z) and the capability indices (C_p, C_{pk})

The capability analysis tool was illustrated with an example involving a winery. We obtained the sigma and the capability indices for a bottle-filling process as well as global output with charts and facts. We used the functions `ss.ca.z`, `ss.ca.cp`, `ss.ca.cpk`, and `ss.study.ca`, which are available in the `SixSigma` package. Other R packages containing capability analysis functions are `qcc` [93]; `qAnalyst` [98]; and `qualityTools` [88].

Now you can proceed with the capability analysis for the paper helicopter project and the proposed practice. You can also study some topics in more depth. Explanations about statistical issues and formulas can be found in [69]. A good starting point, with straightforward explanations, is [34]. A good visual reference is [46]. A detailed compilation of the various capability indices can be found in [77].

Case Study

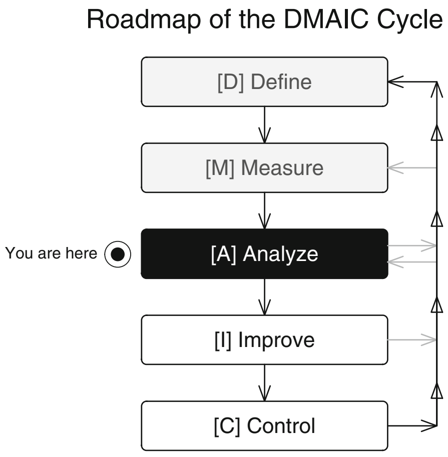
The most important key characteristic of the paper helicopter is the flight time. Now that you have experience measuring the flight time, decide your specification limits (LSL and USL). Take 20 measurements of one prototype and save the data in a vector. Obtain the sigma and the indices with R. Run the capability study and comment on the output. Is your process capable?

Practice

7.1. Consider a process whose specification limits are LSL $= 4$ and USL $= 14$. We also know that the mean is 10 and standard deviation is 2. Calculate the sigma of the process.

7.2. If you successfully improve the process described in Practice 7.1 and lower the standard deviation to 1, by approximately how much will you reduce the DPMO?

Part IV
R Tools for the Analyze Phase



In this part of the book, tools useful during the Analyze phase are introduced.

In this phase, data are used to establish the key process inputs and their relation to the process outputs.

We will introduce the most representative tools, including charts and statistical tools for the analysis of data. Probability and inference are explained in a comprehensive way.

Chapter 8
Charts with R

The greatest value of a picture is when it forces us to notice what we never expected to see.

John Tukey

8.1 Introduction

Charts are particularly important in Six Sigma projects. The aim of a chart is usually to support the interpretation of data. Hence, providing an adequate explanation of data through charts is crucial. Two-dimensional charts can be used as the basis for more complex representations, and multidimensional data can be shown through extensions of these charts.

Before describing the main charts used in Six Sigma projects, in this section we are going to explain some important concepts regarding charts in general. Sections 8.2–8.7 detail the most important types of charts. In Sects. 8.8 and 8.8.4 we provide some advanced features for charting.

8.1.1 Use of Charts

When talking about charts, the adage *a picture is worth a thousand words* comes to mind. A graphical representation of data helps us to have a better understanding of our process. But we must realize that charts are just a way to interpret data; they are not the data or the conclusions themselves. Charts are useful in every step of a Six Sigma project, and in fact they are used not only in the Analyze phase of the design, measure, analyze, improve, and control cycle, but in the whole cycle.

Charts are very useful in the early stages of analysis, when we need a description of the process to make conjectures, figure out relationships, and plan further research. Thus, descriptive analysis is the most important mission of charts. In the final step of our analysis, when we present our results, charts will probably be the

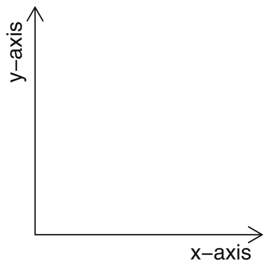


Fig. 8.1 Two-dimensional chart foundation. The *horizontal axis* is called the x-axis, and the *vertical axis* is called the y-axis. Most charts are based on these concepts, representing one variable (independent) on the x-axis, and another (dependent) on the y-axis

business card of our conclusions. Hence, choosing the appropriate type of chart to communicate our advances or proposals is crucial. A poor chart in a report may ruin good work performed during the project. In the middle, we will probably have used a lot of different charts to interpret the information provided by the data until we reach the final conclusions.

8.1.2 Background Concepts

Most charts we use in statistics (and thus in Six Sigma) are two-dimensional charts. These two *dimensions* are represented as two axes, namely the x-axis (horizontal) and the y-axis (vertical) (Fig. 8.1). Each axis represents a different variable. If we want to display the relationship between two variables, the x-axis is usually assigned to the *independent* variable and the y-axis to the *dependent* variable.

The nature of the variables we are measuring in each dimension determines the scale of the chart. Most times the chart will be right for us, but sometimes we will need to adjust the scale of the chart to enhance its ability to *talk* about the data. We may choose different scales. Thus, we can use a logarithmic scale for one of the dimensions or for both. We can also set an aspect ratio between the two dimensions (e.g., y-axis doubles x-axis). The scale must be clearly shown in the chart to avoid wrong interpretations or misunderstandings.

Another decision about our chart is fixing its limits. Space is infinite, but the piece of paper or the screen where we are going to display our chart is not. Therefore, we must choose the limits of our axes, and these limits must allow a clear visualization of the data without hiding any relevant information. It is advisable to append some figures to the chart, or even a table with the data that are being plotted.

Example 8.1 (Printer cartridge). Table 8.1 shows the data of a sample of 24 printer cartridges of a given brand. They are from three different fillers, and there are two types of cartridge: color and black. The volume and density in each cartridge were measured. The data set `ss.data.pc` in the `SixSigma` package contains these data. As an example of axis scaling, we have plotted a scatterplot of the two continuous variables (volume and density) with different settings for the axes. See how the chart changes in Fig. 8.2.

Table 8.1 Data for printer cartridge example

	pc.col	pc.filler	pc.volume	pc.density	pc.batch	pc.op
1	C	1	16.75	1.25	1	A
2	C	2	18.01	1.11	1	B
3	C	3	15.64	1.14	1	C
4	C	1	18.03	1.09	1	D
5	C	2	13.78	1.15	2	A
6	C	3	16.76	1.12	2	B
7	C	1	14.69	1.35	2	C
8	C	2	15.20	1.18	2	D
9	C	3	14.21	1.46	3	A
10	C	1	15.96	1.19	3	B
11	C	2	18.15	1.28	3	C
12	C	3	14.23	1.28	3	D
13	B	1	16.86	1.30	4	A
14	B	2	14.28	1.11	4	B
15	B	3	16.13	1.27	4	C
16	B	1	15.92	1.34	4	D
17	B	2	16.86	1.20	5	A
18	B	3	16.34	1.20	5	B
19	B	1	15.42	1.54	5	C
20	B	2	16.79	1.31	5	D
21	B	3	15.31	1.14	6	A
22	B	1	14.82	1.22	6	B
23	B	2	17.27	1.26	6	C
24	B	3	15.69	1.33	6	D

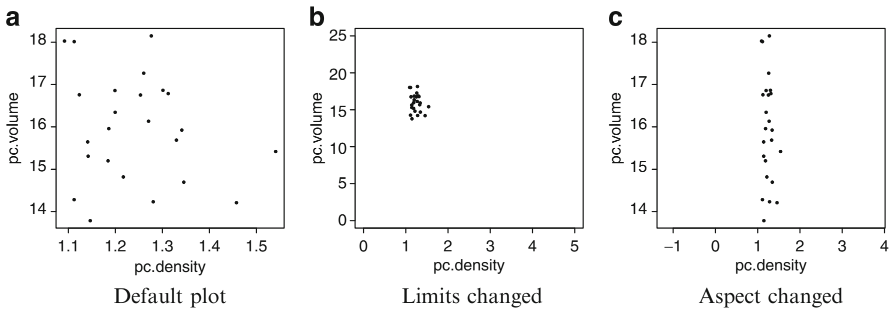


Fig. 8.2 Different visualizations for the same data. (**a**) was plotted using the default settings (scales adjusted automatically). (**b**) was plotted with the axes enlarged. (**c**) was plotted with the same scales on the two axes

Figure 8.2a is plotted with the default settings of the `plot` function:

```
> plot(pc.volume ~ pc.density,
    data = ss.data.pc,
    pch = 16)
```

Figure 8.2b is plotted with the default limits changed:

```
> plot(pc.volume ~ pc.density,
    data = ss.data.pc,
    xlim = c(0,5),
    ylim = c(0,25), pch=16)
```

Figure 8.2c is plotted applying the same scale on both axes:

```
> plot(pc.volume ~ pc.density,
    data = ss.data.pc,
    asp = 1,
    pch = 16)
```

We will describe scatterplots in detail in Sect. 8.4. □

Last but not least: what are we going to plot in the chart? The two main elements to choose from are dots or lines. We may use different types of dots (filled circle, unfilled circle, triangle, etc.) and different types of lines (solid, dotted, dashed, etc.) to distinguish different blocks of information (e.g., the data for different countries in a given time series). Moreover, we can merge both symbols or use more elaborate ways to identify data (see below).

Often two dimensions are not enough to represent reality. We have several options to visualize multidimensional data, but they are extensions of two dimensions, so the concepts introduced previously are applicable.

Here are some options that will be used throughout this chapter:

- Colors and symbols to distinguish other variables;
- Annotations (you may print text inside a plot to identify the value of another variable);
- Multiple charts; this is the most extended way to represent multidimensional data. We can plot a matrix of charts where each chart plots two of the variables. Usually, all charts in a matrix are plotted using the same scale.

Once you have measured the variables you want to analyze, you may follow this guideline to generate the best chart:

- Choose the variables you want to visualize.
- Find out what types of variable you have (factor, discrete, continuous).
- Decide what characteristic of the data you need: variation, relationship, distribution, location, amount, etc.
- Choose the chart that will best show what you need to know.
- Carefully choose the scale of your chart.
- Plot the chart several times until you have what you really need. Maybe you will have to transform the data, or even acquire different or more data.

In the following sections, we will explain with examples some charts that are widely used in Six Sigma projects. We will use functions in the `graphics` package, included in the `base` installation of R, with different arguments in the commands, to show varied visualizations. As we explained in Chap. 2, you can retrieve the documentation of the functions by typing `?somefunction`, where `somefunction` is the name of the function we are using (e.g., `barplot`). You also have a quick reference guide in Appendix A.

8.2 Bar Chart

A bar chart is a very simple chart where some quantities are shown as the height of bars. Each bar represents a factor where the variable under study is being measured. A bar chart is usually the best graphical representation for counts.

Example 8.2 (Printer cartridge (cont.)). The printer cartridge manufacturer distributes its product to five regions. An unexpected amount of defective cartridges has been returned in the last month. The bar plot in Fig. 8.3 is a straightforward way to see the data in Table 8.2 (available in the `SixSigma` package within the data set `ss.data.pc.r`). We use the following code in R:

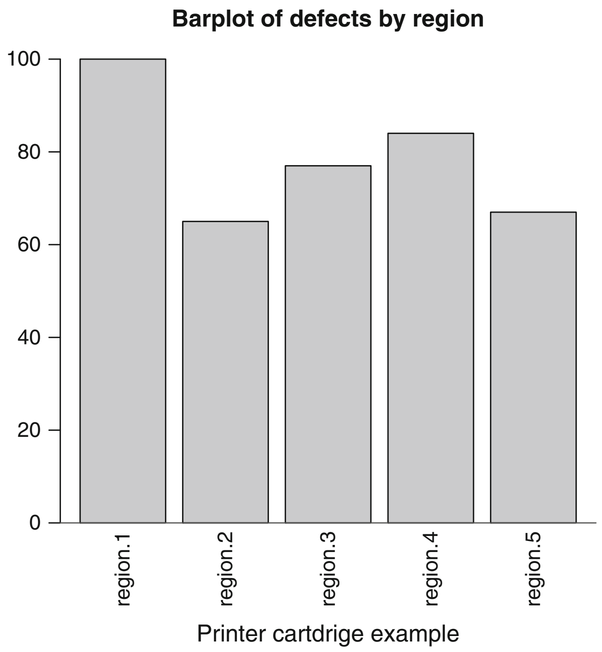


Fig. 8.3 Bar plot of defects by region (printer cartridge example). We can make some variations easily: sort the data to obtain a Pareto chart, set colors of the bars, or flip horizontally passing the argument `horiz = TRUE`

Table 8.2 Defects by region for printer cartridge example

	pc.regions	pc.def.a	pc.def.b	pc.def
1	region.1	37.00	63.00	100.00
2	region.2	17.00	48.00	65.00
3	region.3	38.00	39.00	77.00
4	region.4	41.00	43.00	84.00
5	region.5	13.00	54.00	67.00

```
> with(ss.data.pc.r,
    barplot(pc.def,
        names.arg = pc.regions,
        las = 2,
        main = "Barplot of defects by region",
        sub = "Printer cartridge example"))
> abline(h = 0,
    col = "#666666")
```

□

Some variations exist of bar charts. A Pareto chart (Chap. 6) is a sorted bar chart (highest to lowest bars). A horizontal bar chart is a bar chart with the factors on the y-axis. The histogram described in the next section is a special bar chart for continuous variables. We can plot several bars (adjacent or stacked) when we have more than one category to display.

Example 8.3 (Printer cartridge (cont.)). We can plot the bar chart for each type of cartridge within the same plot, even with a legend, using the following command (Fig. 8.4):

```
> barplot(as.matrix(ss.data.pc.r[,2:3]),
    las = 1,
    beside = TRUE,
    legend = ss.data.pc.r[,1],
    args.legend = list(x=3.5,y=60),
    main = "Barplot of defects by region and type",
    sub = "Printer Cartridge Example")
> abline(h = 0,
    col = "#666666")
```

□

8.3 Histogram

A histogram is a bar chart for continuous variables. This bar chart shows the distribution of the measurements of variables. On the x-axis, each bar represents an interval of the possible values of a variable. The height of the bars (that is, the y-axis) depends on the frequency (relative or absolute) of the measures within each interval. The rule is that the area of the bars should be proportional to the frequencies.

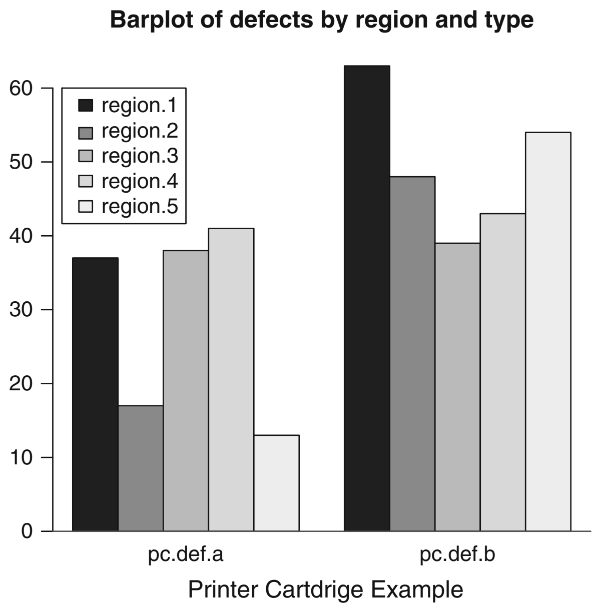


Fig. 8.4 Multiple bar plot for printer cartridge example. The regions are plotted twice: one for each type of cartridge. This way we can see at a glance all the information in the table

The histogram is used to find the distribution of a variable, that is:

- Is the variable centered or biased?
- What is the variation like? Are the observations close to the central values, or is it a spread distribution?
- Is there any pattern that would prompt further analysis?
- Is it a *normal* distribution?

Figure 8.5 has some patterns that we can identify in a histogram.

To make a histogram of our data, we first determine the number of bins (bars) that we are going to plot. Then, we decide on the width of the intervals (usually the same for all intervals) and count the number of measures within each interval. Finally, we plot the bars. For intervals with equal widths, the height of the bars will be equal to the frequencies. R will do all the calculations automatically, but we can always change the default settings in order to obtain the best visualization for our histogram.

Example 8.4 (Printer cartridges (cont.)). Figure 8.6 has the histograms for the variables *volume* and *density* in the `ss.data.pc` data set (Table 8.1). We generate them using the following code:

```
> hist(ss.data.pc$pc.volume,
    main="Printer Cartridge Volume",
    xlab="Volume",
    col="#DDDDDD")
```

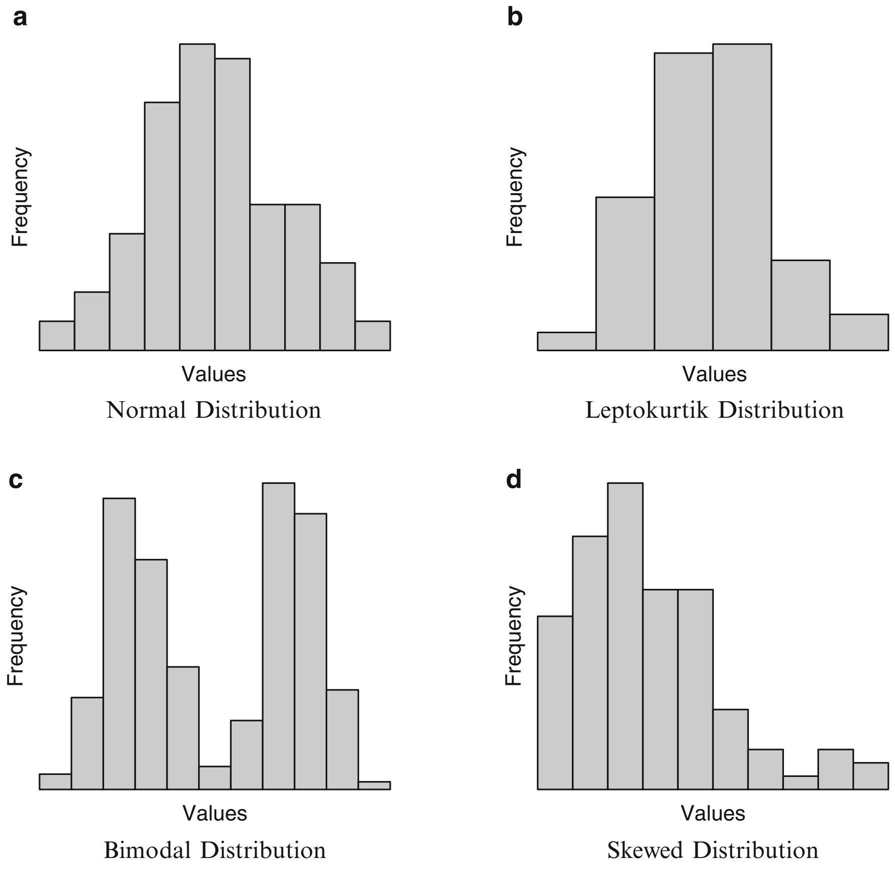


Fig. 8.5 Various histogram patterns. (**a**) represents a normal distribution, with most of the values around the mean and centered (mean is equal to the median and mode). (**b**) is another normal distribution, but with higher peakedness (leptokurtik) than a standard normal. (**c**) suggests that we may have two different groups of data, with distinct means. (**d**) is a right-skewed distribution in which high values are less frequent than low values

```
> hist(ss.data.pc$pc.density,
    breaks = "FD",
    main = "Printer Cartridge Density",
    xlab = "Volume",
    col = "#DDDDDD")
```

The `breaks` argument allows us to set the number of bins. We could pass a fixed number or use the Sturges, Scott, or FD method. We could also use a customized function to compute the number of breaks (intervals).

The following code generates an improved histogram for the variable *volume* superposing a density line for a theoretical normal distribution whose mean is 16 and standard deviation is 1. An annotation beside the line shows the parameters.

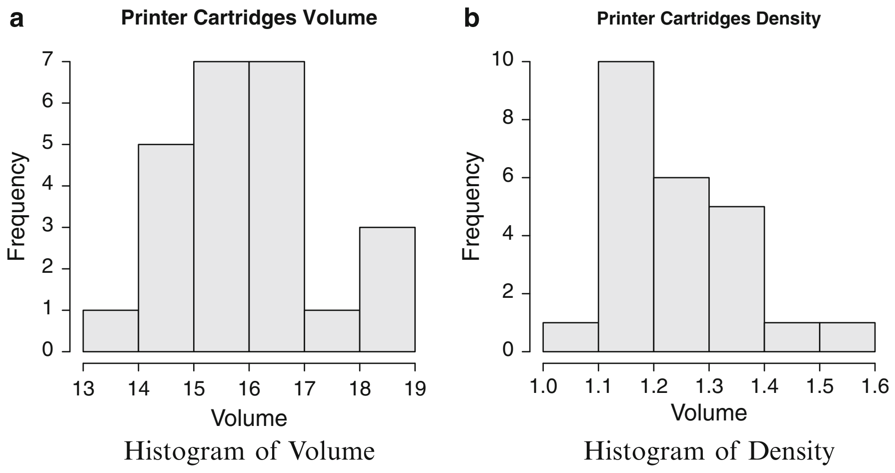


Fig. 8.6 Histograms for printer cartridge example. The data in (**a**) (volume) look like the stem from a normal distribution around 16. The data in (**b**) (density) are right skewed, with most values near 1

A density line of the distribution and grid lines are also plotted. The result is shown in Fig. 8.7.

```
> hist(ss.data.pc$pc.volume,
    main = "Printer Cartridge Volume",
    xlab = "Volume",
    col = "#BBBBBB",
    border = "white",
    bg = "red",
    freq = FALSE,
    ylim = c(0,0.4))
> curve(dnorm(x,16,1),
    add = TRUE,
    lty = 2,
    lwd = 2)
> lines(density(ss.data.pc$pc.volume),
    lwd = 2)
> text(label = expression(paste(mu==16,
                              "; ",
                              sigma==1,
                              sep = "")),
    x = 16.5,
    y = 0.4,
    adj = c(0,1))
> grid()
> box()
```

□

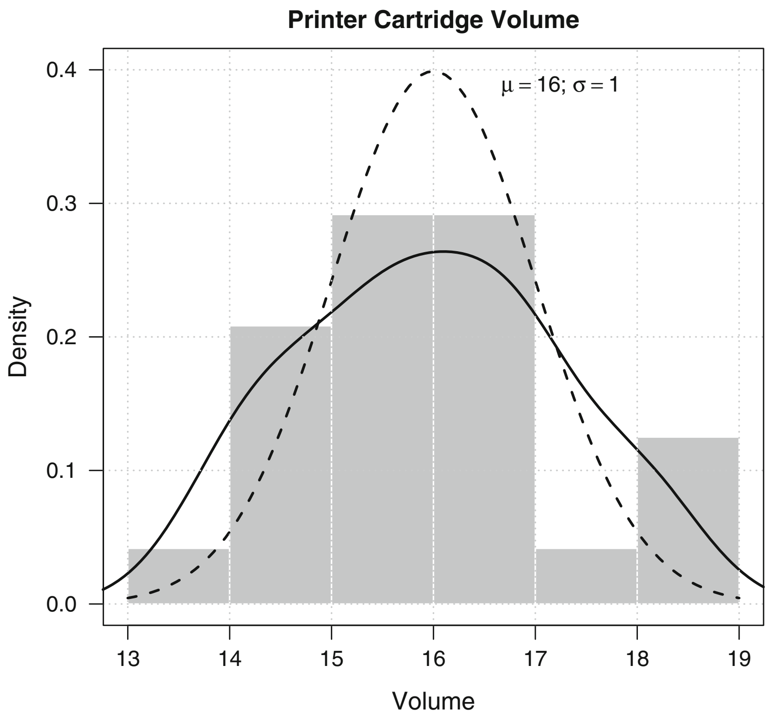


Fig. 8.7 Histogram with density lines (cartridge example). In this histogram, we have changed the color of the bins. The *solid line* is a density line for the data. The *dashed line* is the theoretical density line for a normal distribution. The parameters of this theoretical distribution have been annotated within the chart

8.4 Scatterplot

A scatterplot is an important tool to reveal relationships between two variables. In statistical language, these relationships can be subsumed into the concept of *correlation*. Thus, we may have three types of correlation between two variables:

- Positive correlation: high values of one of the variables lead to high values of the other one.
- Negative correlation: high values of one of the variables lead to low values of the other one.
- No correlation: the variables are independent.

Within a Six Sigma project, discovering the relationship between the Ys and Xs of the process is an important task. This relationship will be measured in terms of the correlation among the variables. Once we have discovered the relationship, we will have to prove that the independent variable is causing the variation of the dependent variable. The starting point to this analysis is the scatterplot. We will describe in depth the concept of *correlation* in Chap. 9.

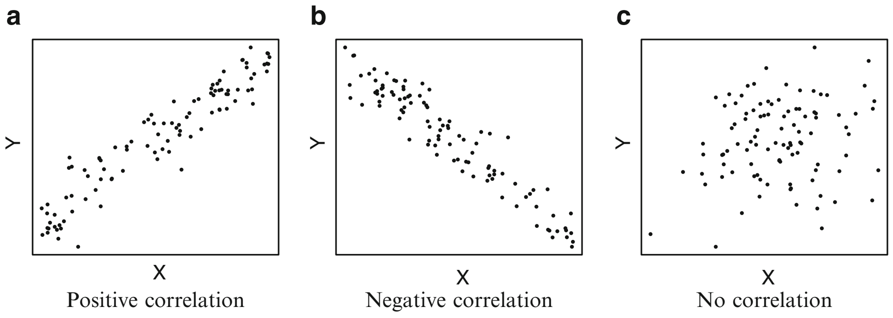


Fig. 8.8 Scatterplot patterns. When the relationship between two variables is clear, we get one of these patterns. (**a**) shows a positive correlation, (**b**) a negative correlation. In (**c**), there is no correlation: the variables are independent

Correlation describes how variables vary, but this variation is not necessarily due to a cause-and-effect relation.

In the scatterplot, the x-axis represents the independent variable and the y-axis the dependent variable. In this way, plot each pair (x, y) obtaining a *cloud of points*. We can obtain the three patterns shown in Fig. 8.8 corresponding to the relations mentioned above.

The scatterplot is mainly used when we have two continuous variables. However, it can be used also when the independent variable is a factor. The difference will be that we will have the dots aligned over the position of the factor value.

To make a scatterplot with R, we use the function `plot`, using the vectors that contain the data as arguments. We can use symbolic expressions (also known as model *formulae*) to call the function. Type `?formula` to learn more about model formulae with R.

Example 8.5 (Printer cartridge (cont.)). In the `ss.data.pc` data set, we have two continuous variables: `pc.volume` and `pc.density`. If we want to check whether the density and the volume are related, the first thing we have to do is generate a scatterplot to find patterns for this relation. We use the following code to produce the scatterplot in Fig. 8.9.

```
> plot(pc.volume ~ pc.density,
    data = ss.data.pc,
    main = "Searching correlation between Density
            and Volume",
    col = "#666666",
    pch = 16,
    sub = "Printer Cartridge Example",
    xlab = "Volume of Ink",
    ylab = "Density")
> grid()
```

□

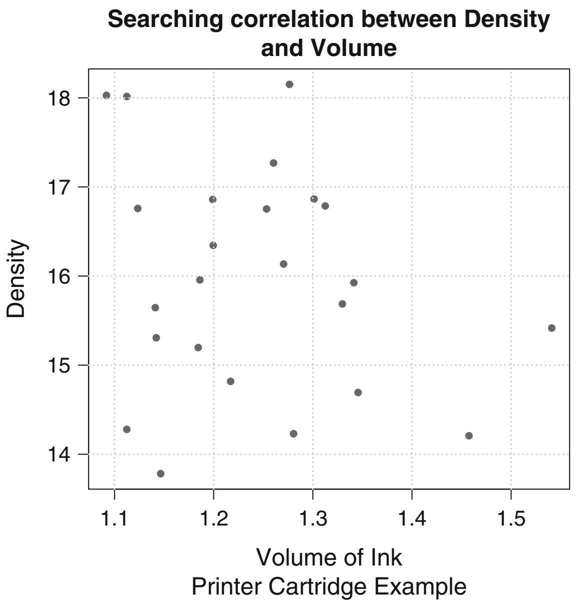


Fig. 8.9 Scatterplot for printer cartridge example. The density is plotted on the x-axis, the volume on the y-axis. We can see that the cloud of points does not follow any pattern, so the variables are independent

We can reveal other types of relationships such as nonlinear ones. This topic is beyond the scope of this book. Nevertheless, you can find out more about statistical modeling in the *Summary and Further Reading* sections of this and subsequent chapters.

8.5 Run Chart

A run chart is a bidimensional chart where the x-axis represents a time line and on the y-axis is plotted a variable that we want to monitor. This variable may be a critical to quality (CTQ) characteristic of our process or a parameter that affects it. These types of charts are also called time-series charts when we have a time scale on the x-axis (for example, the number of orders received every day). The scale of the x-axis may not necessarily be temporal (for example, the volume of some recipients whose production is sequential).

Thus, we will have a number of subgroups where a characteristic is measured, and we have the order of the subgroups (notice that a subgroup may contain only one element). Usually a centered line is plotted in a run chart. It may represent a target, the mean of the data, or any other value.

Run charts allow us to detect patterns that can be indicative of changes in a process. Changes entail variability and, thus less quality. In particular, if we detect cycles, trends, or shifts, we should review our process (Fig. 8.10).

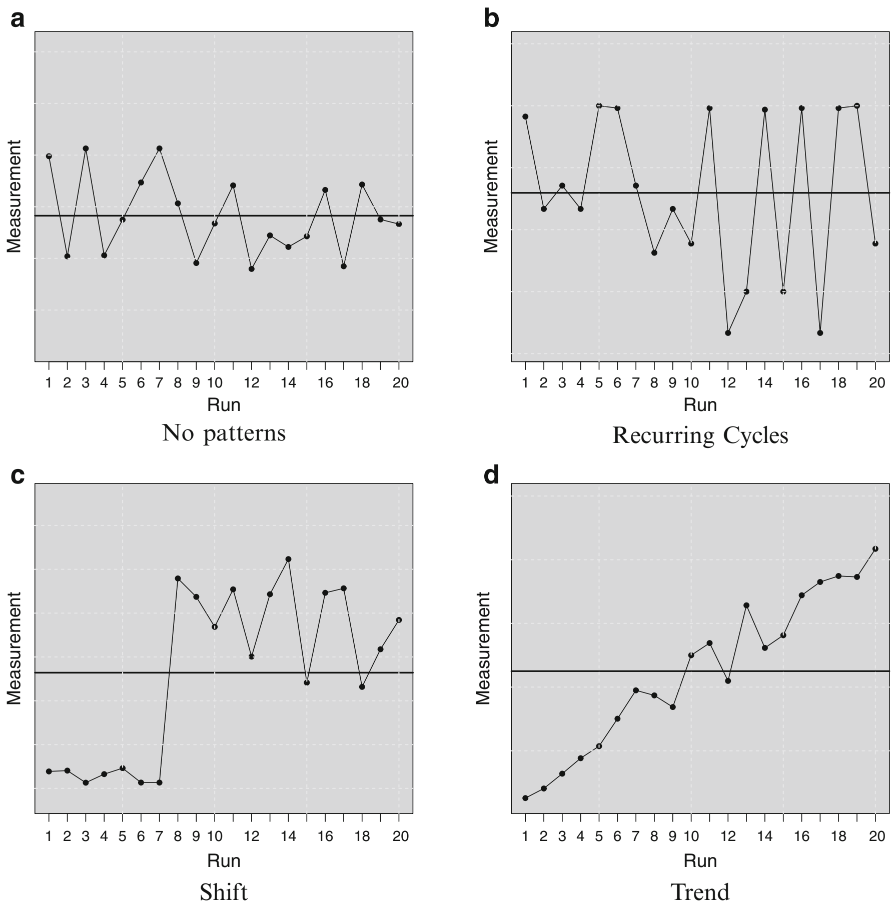


Fig. 8.10 Patterns in a run chart. We can identify patterns in a run chart. (**a**) has no pattern. (**b**) looks like a wave (ups and downs). (**c**) presents a clear shift from the eighth run. Finally, (**d**) presents an ascending trend

To create a run chart with R, we must plot the variable we want to study on the *y*-axis and the number of runs on the *x*-axis. If we have the data sorted by run, we can automatically plot the values using the vector of response data.

Example 8.6 (Printer cartridge (cont.)). We want to monitor the volume in the cartridges, available in the dataset `ss.data.pc`. Suppose that the target value for the volume is 16. Using the following code we get the Run Chart in Fig. 8.11.

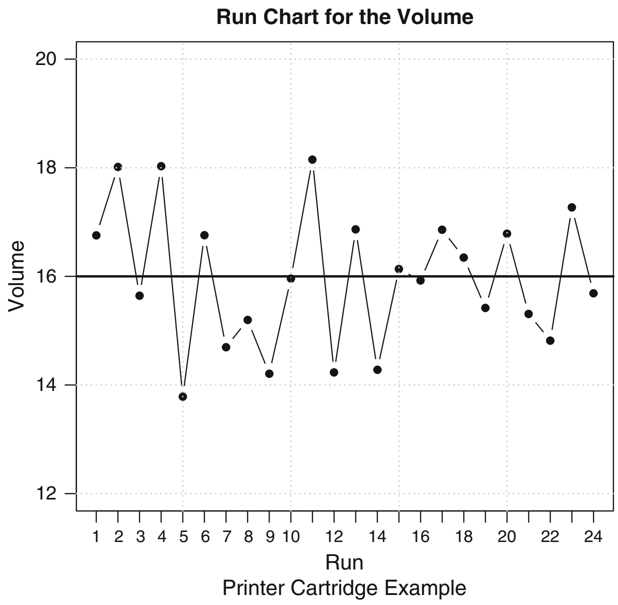


Fig. 8.11 Run chart for printer cartridge example. Here we have 24 sorted observations. No pattern (trend, cycle, or shift) is apparent in the process evolution

```
> plot(ss.data.pc$pc.volume,
    type = "b",
    pch = 16,
    ylim = c(12,20),
    axes = FALSE,
    main = "Run Chart for the Volume",
    sub = "Printer Cartridge Example",
    xlab = "Run",
    ylab = "Volume")
> axis(1,
    at = 1:24,
    cex.axis = 0.7)
> axis(2)
> box()
> grid()
> abline(h = 16,
    lwd = 2)
```

To obtain a better visualization of the data, we made the following improvements from the default settings of the `plot` function:

- We added title, subtitle, and customized *x*-axis and *y*-axis labels.
- We dropped the axes in the `plot` function and then added them with the `axis` function. In this way, we were able to display all the ID numbers of the runs.
- We added a grid and a box to the plot.

□

8.6 Tier Chart

A tier chart is similar to a run chart. We use tier charts when we have more than one observation in each run (e.g., batches, days, etc.). With the tier chart we can see short-term variation and long-term variation jointly in a single chart. Short-term variation is the variation within each subgroup, whereas long-term variation is the variation among all the groups.

To create a tier chart, we plot vertical lines at the position of each run from the higher to the lower value. Then, the single values are plotted as a point or as a vertical segment.

Example 8.7 (Printer cartridge (cont.)). Let us assume that the sample of 24 cartridges comes from 6 different batches and that they are sequentially measured. That is, cartridges 1 to 4 are from batch 1, 5 to 8 are from batch 2, and so on. The information is in the variable `pc.batch`.

We can create the plot using the `stripchart` function and adding one line for each subgroup (Fig. 8.12):

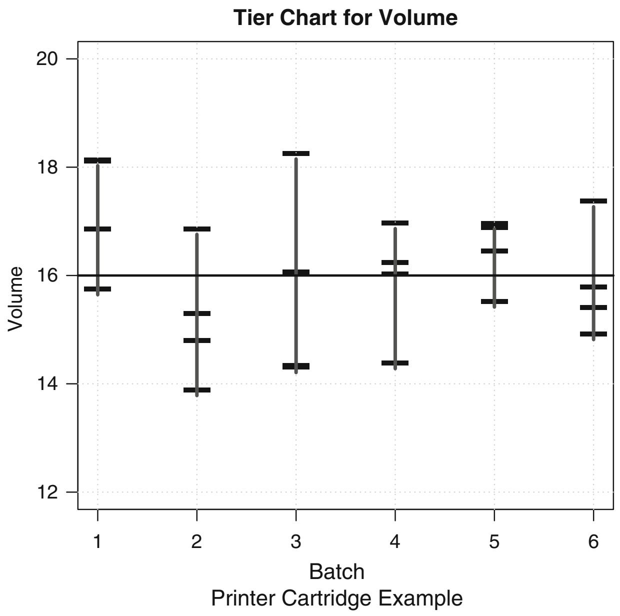


Fig. 8.12 Tier chart for printer cartridge example. First we plotted a strip chart and then added lines for every subgroup (batch)

```
> stripchart(pc.volume ~ pc.batch,
    data = ss.data.pc,
    pch = "-",
    cex = 3,
    xlab = "Batch",
    ylab = "Volume",
    ylim = c(12,20),
    vertical = TRUE,
    main = "Tier Chart for Volume",
    sub = "Printer Cartridge Example")
> grid()
> for (i in 1:6){
 lines(x = rep(i,2),
    lwd = 3,
    col = "#666666",
    y = c(max(ss.data.pc$pc.volume[
                            ss.data.pc$pc.batch==i]),
        min(ss.data.pc$pc.volume[
                        ss.data.pc$pc.batch==i])))
 }
> abline(h = 16,
    lwd = 2)
```

□

8.7 Box–Whisker Chart

The box–whisker chart is also known as the *box plot*. It graphically summarizes the distribution of a continuous variable. The sides of the box are the first and third quartiles (25th and 75th percentile, respectively). Thus, inside the box we have the middle 50% of the data. The median is plotted as a line that crosses the box. The extreme whisker values can be the maximum and minimum of the data or other limits beyond which the data are considered *outliers*. The limits are usually taken as

$$[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR],$$

where $Q1$ and $Q3$ are the first and third quartiles, respectively, and IQR is the interquartile range ($Q3$–$Q1$). We can replace 1.5 with any value in the `boxplot` function of R. The outliers are plotted beyond the whiskers as isolated points and can be labeled to identify the index of the outliers.

The box plot tells us if the distribution is centered or biased (the position of the median with respect to the rest of the data), if there are outliers (points outside the whiskers), or if the data are close to the center values (small whiskers or boxes). This chart is especially useful when we want to compare groups and check if there are differences among them.

Example 8.8 (Printer cartridge (cont.)). In a production line, we have three fillers for the printer cartridges. We want to determine if there are any differences among

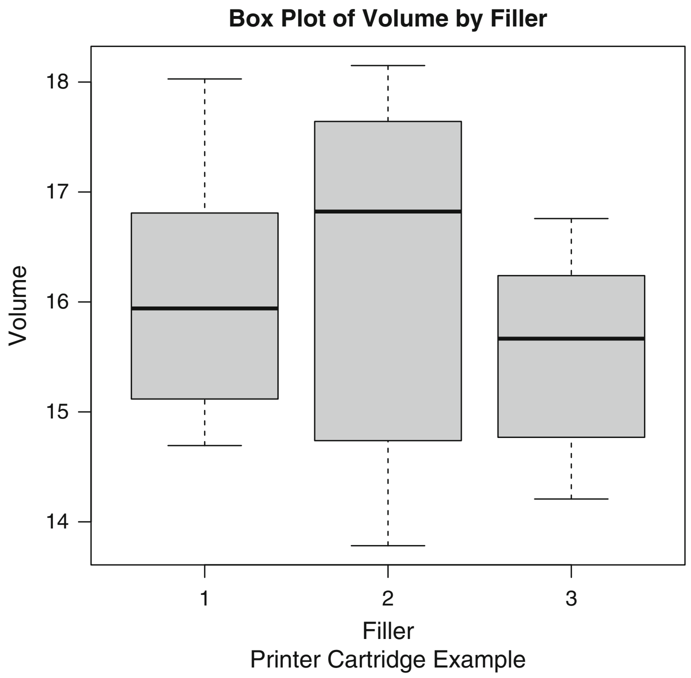


Fig. 8.13 Box–whisker charts for printer cartridge example. The volume of the cartridges filled filler 2 is more spread out than those filled by fillers 1 and 3. There are no outliers and the data are quite centered

the fillers and identify outliers (for instance, errors in data entry). We obtain the box plot in Fig. 8.13 for each group with the following code:

```
> boxplot(pc.volume ~ pc.filler,
    data = ss.data.pc,
    col = "#CCCCCC",
    main = "Box Plot of Volume by Filler",
    sub = "Printer Cartridge Example",
    xlab = "Filler",
    ylab = "Volume")
```

□

We can save the box plot in an object and have access to the data used to plot the box–whisker chart. This is very useful for identifying outliers or for conducting further analysis.

Example 8.9 (Printer cartridge (cont.)). Let us plot a box–whisker chart again for all the data with the following code. We also changed the range value for outliers to 0.7. We obtain one outlier, corresponding to the 11th row of the data set (Fig. 8.14).

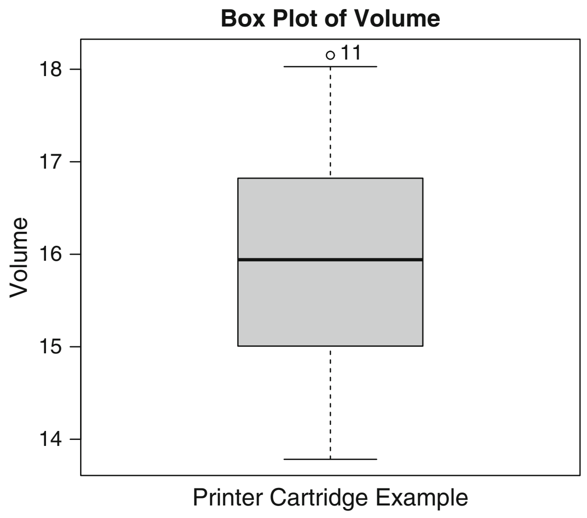


Fig. 8.14 Labeling outliers in a box plot. In this chart, we modified the default value of the range argument. We used a vector as input data instead of a formula because we do not want to analyze the groups separately

```
> my.bp <- boxplot(ss.data.pc$pc.volume,
    col = "#CCCCCC",
    main = "Box Plot of Volume",
    sub = "Printer Cartridge Example",
    ylab = "Volume",
    range = 0.7)
> text (x = rep(1, length(my.bp$out)),
    y = my.bp$out,
    labels = which(ss.data.pc$pc.volume==my.bp$out),
    pos = 4)
> str(my.bp)
```

□

8.8 Other Charts

Many types of charts may be plotted based on bidimensional charts like those described in previous sections. It is not pleasant to be constrained to the sometimes strict default settings of the software we are using. R overcomes this limitation. We can plot almost anything we need by adjusting some parameters in the graphical functions.

We will plot some charts commonly used in Six Sigma just using the appropriate arguments in the `base` R functions. The *multivariate chart* is an exception: we will use `lattice` graphics, which is a powerful package to display multivariate data.

8.8.1 Group Chart

In a group chart, we identify the points in the chart, showing the group they belong to. It is useful when the characteristic we are measuring is produced by different process streams or is measured in different locations. The higher and lower values of each group are usually linked by lines.

Example 8.10 (Printer cartridge (cont.)). Suppose that the four printer cartridges in each batch are labeled according to the operator that sells the cartridge (A, B, C, and D). The information is contained in the variable `pc.op`. With the following code we obtain the group chart in Fig. 8.15. At first we do not plot anything in the chart (`pch = ""`), and then we plot graphical elements (a gray rectangle, dots, labels, and lines).

```
> stripchart(pc.volume ~ pc.batch,
    vertical = TRUE,
    data = ss.data.pc,
    pch="",
    xlab = "Batch",
    ylab = "Volume",
    ylim = c(12,20),
    main = "Group (Operator) Chart for Volume",
    sub = "Printer Cartridge Example")
> rect(par("usr")[1],
    par("usr")[3],
    par("usr")[2],
    par("usr")[4],
    col = "#CCCCCC")
> box(col = "#CCCCCC")
> grid(col = "#EEEEEE")
> points(pc.volume ~ pc.batch,
        data = ss.data.pc,
        pch = 19)
> with(ss.data.pc,
    text(label = pc.op,
        x = pc.batch,
        y = pc.volume,
        pos = 4))
> lines(aggregate(pc.volume ~ pc.batch,
        data = ss.data.pc,
        max),
    lwd=2)
> lines(aggregate(pc.volume ~ pc.batch,
        data = ss.data.pc,
        min),
    lwd=2)
```

□

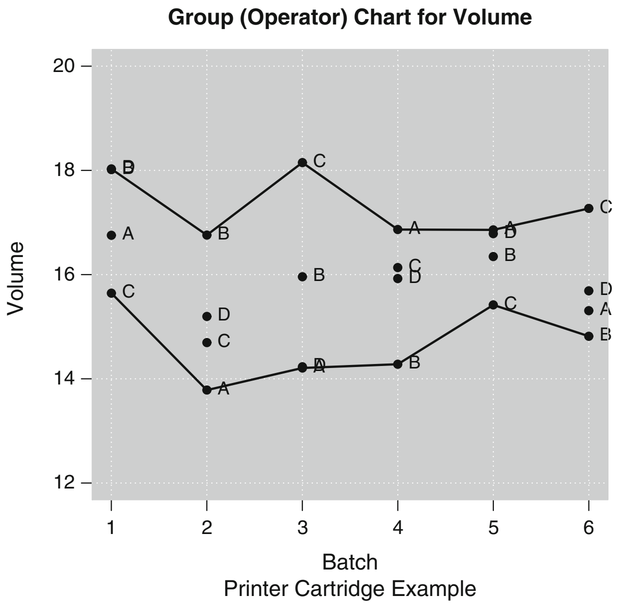


Fig. 8.15 Group chart for printer cartridge example. For measurements coming from different process streams or taken indifferent locations, we can use this type of chart to identify the group (stream or location) each measurement belongs to

8.8.2 Location Charts

Consider a characteristic that must be measured at several locations of a part. If we want to measure this characteristic in a set of different parts, then we can use location charts to represent it. For each part, the *x*-axis represents the locations and the *y*-axis corresponds to the value of the characteristic at each location. Different symbols or colors can be used to distinguish the response values corresponding to each part.

We can plot several types of location charts: location run chart, location tier chart, location box–whisker chart, or any other customized chart with the location always on the *x*-axis. This kind of chart can be plotted in a similar way to the representations described above, taking into account that their interpretation is different.

8.8.3 Multivariate Chart

Multivariate charts are used to detect graphically which factors can affect the CTQ characteristic. In exploratory analysis, we can plot historical data to discover which factors may be important to design experiments. The idea is to plot individual measurements jointly with the average values of the factors involved.

The `mvPlot` package function in the `qualityTools` package builds multivariate charts using the standard graphics of R. See the documentation on the package and the function to find out more about the use of this function.

We can build our customized multivariate chart using the `lattice` package [91]. We can arrange the layout of several charts in rows and columns and plot by group using colors, symbols, and annotations to display several variables.

Example 8.11 (Printer cartridge (cont.)). We are going to use an extended data set of the printer factory for the multivariate chart (`ss.data.pc.big`). We have all the combinations for the four factors mentioned previously [color (2), operator (3), filler (3), and batch (4), instead of the (6) we had before]. This data set has 72 observations ($2 \times 3 \times 3 \times 4$). You can see the structure of the data set with the `str` command:

```
> str(ss.data.pc.big)
```

```
'data.frame':   72 obs. of  5 variables:
 $ filler  : Factor w/ 3 levels "1","2","3": 1 2 3 1 2 3 1 2
    3 1 \ldots
 $ batch   : Factor w/ 4 levels "1","2","3","4": 1 1 1 2 2 2
    3 3 3 4 \ldots
 $ col     : Factor w/ 2 levels "B","C": 1 1 1 1 1 1 1 1 1 1
    \ldots
 $ operator: Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1
    1 1 \ldots
 $ volume  : num  16.8 18 15.6 18 13.8 \ldots
```

Graphics obtained with the `lattice` package may be plotted with the standard options using model *formulae* (symbolic expressions) to obtain the most frequently used charts. We can also use advanced features using *panel functions* to customize the multivariate charts. This is the code to get the multivariate chart in Fig. 8.16:

```
> require(lattice)
> #compute the overall mean
> o.mean <- mean(ss.data.pc.big$volume)
> mvChart <- xyplot (volume ~ filler | col * operator,
    main="Multi-vari chart for Volume by color and operator "
        ,
    xlab = "Filler",
    ylab ="Volume",
    data = ss.data.pc.big,
    groups = batch,
    panel = function (x, y, \ldots,
        groups, subscripts){
        #horizontal lines
        panel.grid(h = -1, v = 0,
            col = "#CCCCCC")
        #points and lines for the measurements
        panel.stripplot(x, y,
            groups = groups,
            subscripts = subscripts,
            type="b")
        #this line is for the mean of all the batches
        #for each color, operator and filler
        panel.linejoin(x, y,
            groups = groups,
            subscripts = subscripts,
            horizontal = FALSE,
            col = "#000000",
            type= "a",
            lwd= 4)
        #A square for the overall mean
        panel.points(mean(as.numeric(x)), o.mean,
                cex=2,
                pch=15,
                col = "#CCCCCC")
        #this is for the mean inside each sub-plot
        panel.points(mean(as.numeric(x)),mean(y),
                cex=1.5,
                pch=2)
    },
    #this is for the legend above the chart
    auto.key = list(title="batch",
            lines = TRUE,
            points = TRUE,
            columns = 4))
>   print(mvChart)
```

□

8.8.4 More About Charts

In addition to the methods described in previous sections, we can use other techniques to display multivariate data:

- Auxiliary axes for superposed plots.
- Perspective. We can visualize three-dimensional data by plotting a perspective of the three dimensions (space) rather than two dimensions (plane) just by adding a third axis to the plot.
- Bubbles. You may plot circles whose area is proportional to the value of another variable.

Most charts in this chapter were obtained using functions in the `graphics` package. The only exception was the multivariate chart, where we used `lattice` graphics. Many others advanced packages exist that generate sophisticated charts and graphs.

In addition to `lattice`, we can use the package `ggplot2`, which uses the *grammar of graphics* (see [106]). Interactive plots are possible with the `iplots`

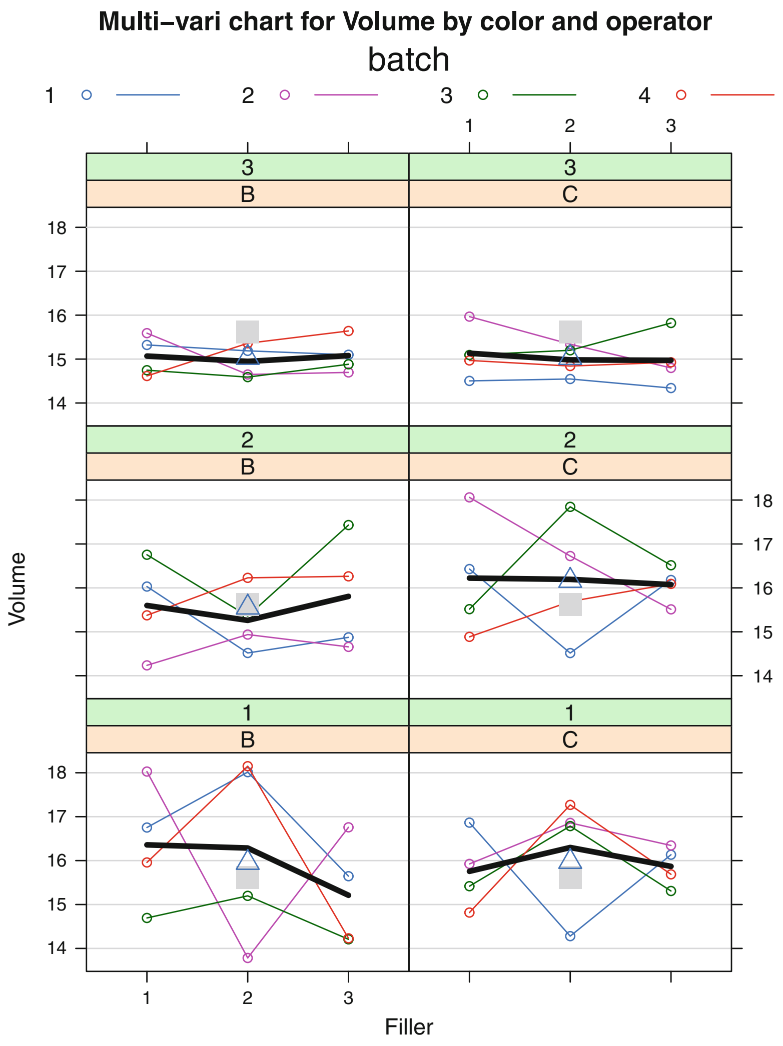


Fig. 8.16 Multivariate chart for printer cartridge big data frame. We plotted different lines joining measurements in the same batch. The *gray square* represents the overall mean, and the triangles are at the mean of each subplot. We can see that the plots for operator 3 (top) look different from the rest (lower values and less variation)

package. We recommend visiting the Graphics Task View at CRAN[1] to stay up-to-date on the topic.

[1]http://cran.r-project.org/web/views/Graphics.html.

We also have several ways to plot geographical data using the packages described in the spatial task view at textsfCRAN.[2] For example, you can color and annotate the regions in a part of the world according to the number of failures of your process using the package `rworldmap`.

Other important feature you may use to improve your charts is the *low-level* graphics manipulation. Use the `grid` package to access directly the your chart's *canvas* and the mathematical annotation to text-draw (type `?plotmath` at the command prompt).

8.9 Summary and Further Reading

In this chapter, we introduced some important concepts about charting. The Black Belt can use these concepts to build the most appropriate chart to explain the data at hand. A set of charts was created through simple R functions from the `graphics` package in the base installation. Enhancements such as annotations and superimposing were made, and a more complex multivariate chart was plotted using the `lattice` package.

The more you advance in your graphical analysis, the more you will realize that you need to customize your charts. Reference [18] devotes a chapter to R graphics. It is worthwhile studying the graphical capabilities of R. You can start with the book [74], which contains an in-depth description of the R graphics system. For the specific types of graphics in the `lattice` and `ggplot2` packages, see [91] and [105], respectively. For interactive and dynamic graphics, consult [17]. Regarding charts for spatial data, the book [8] is advisable.

A nice free resource is [64]. There is also a compilation of resources and applications in the task view at CRAN devoted to graphics (http://cran.at.r-project.org/web/views/Graphics.html).

Case Study

Plot graphics of your paper helicopter data. You can plot histograms and box plots for the flight time. Try to plot the flight time data aggregated and separated by group (e.g., operator, prototype, etc.). Draw bar plots with counts of nonconformance parts. Practice with your data plotting all the types of charts you have learned. You can measure or simulate new variables and situations (e.g. countries, factories, etc.).

[2]http://cran.r-project.org/web/views/Spatial.html.

Practice

8.1. Plot a bar chart for the total volume of cartridges by filler.

Hint: Use the `aggregate` function to appropriately transform the data of the `ss.data.pc` data frame. See `?agregate`.

8.2. Using the `ss.data.pc.big` data frame, plot a run chart for the cartridge volume in batch 1. Do you detect any pattern in the data?

Hint: Use indexing to filter data. See `?Extract`.

Chapter 9
Statistics and Probability with R

Statistics are like bikinis. What they reveal is suggestive, but what they conceal is vital.

Aaron Levenstein

9.1 Introduction

Most definitions of statistics entail several actions over *data*. These operations may vary depending on the field of applications where statistics are used. Some of the more frequently mentioned ones are collection, analysis, and interpretation. Other applications include classify, summarize, represent, or make an inference. We can summarize these definitions in the expression *data science*. Likewise, statisticians are becoming *data scientists*, sharing fields of knowledge with computer scientists, mathematicians, engineers, bioresearchers, and others.

On the other hand, a clear division between descriptive statistics and inferential statistics is usually made. *Descriptive statistics* deals with the description of data by organizing and summarizing them. Exploratory data analysis uses descriptive statistics to figure out the underlying process behind data to make better inferences in subsequent steps of data analysis. Data representation, described in Chap. 8, is often considered part of descriptive statistics. *Inferential statistics* tries to draw conclusions about the process at hand through the available data. Probability theory and its methods help to validate these conclusions scientifically.

There is another trend that has come out in two different approaches of statistics: the frequentist approach and the Bayesian approach. Although this book is focused on the frequentist approach, the authors are confident in the stunning applications of Bayesian inference to Six Sigma. Nevertheless, this would deserve a new book by itself.

This chapter deals with descriptive statistics and probability, including central tendency measurements, variability measurements, random variables, and probability distributions, with a special focus on the binomial and normal probability distributions.

9.1.1 Variables and Observations

Before explaining how to make calculations over data, we are going to review some basic concepts about statistics from an R perspective. The data we want to analyze are measurements or *observations* of a characteristic in a set of items. These items can be people, processes, parts, or some individual unit in a set where we can observe the characteristic under study.

The characteristic we are observing is a *variable* because it is expected not to be constant among the items.[1] In terms of probability theory, a characteristic is a *random variable* with a given probability distribution (Sect. 9.3). Importantly, statistical analyses are based on the underlying probability distribution of the data.

These *raw data* are usually organized in matrices, that is to say, a set of rows and columns representing the observations and the variables, respectively. Thus, row n corresponds to the nth item, and column m corresponds to the mth variable. Consequently, the value a_{nm} is the observed value of the mth variable in the nth item.

There are two main types of variables: quantitative variables and qualitative variables. When the observed characteristic can be measured using some scale, we have a quantitative variable. When the observed characteristic is a description or a categorization of an item, we have a qualitative variable. Furthermore, quantitative variables can be classified in continuous and discrete variables. Continuous variables can take any value within an interval, for example, temperature or length. Discrete variables can take a countable number of values (finite or infinite), for example, number of defects in a batch or number of correct parts until the first failure.

We can use several R object types to store data:

`vector` Unidimensional objects to store one variable.

`factor` Unidimensional objects to store one qualitative variable. The possible values of the variable are called *levels*.

`matrix` Multidimensional objects to store several variables of the same type.

`data.frame` Multidimensional objects to store several variables of any type. Each column of a data frame is a vector or a factor. This is a more appropriate object for data analysis.

With the following code, we create a vector, a factor, and a matrix, and then we put it all together in a data frame (see Sect. 2.7.2 in Chap. 2 for random variable generation with the `rnorm` function):

```
> my.vector <- c(12.5, 13.6, 12.7, 12.1, 13.0, 12.5)
> my.factor <- factor(c(rep("Red", 3), "Yellow", rep("Blue",
    2)))
> my.matrix <- matrix(rnorm(18),
    ncol = 3)
> my.data.frame <- data.frame(var1 = my.vector,
```

[1]In what follows we will use both "variable" and "characteristic."

```
    color = my.factor,
    my.matrix)
> my.data.frame
```

```
  var1  color         X1         X2         X3
1 12.5    Red -0.2241475  0.3537150  0.1992208
2 13.6    Red  1.1190943 -2.3537350 -0.6604397
3 12.7    Red -0.7806308 -1.2595080 -1.6183023
4 12.1 Yellow  0.9007623  0.2303545  0.5165678
5 13.0   Blue  0.1678742  1.6569767 -1.2069046
6 12.5   Blue  1.1163146 -0.3124438 -1.2862550
```

9.1.2 Summary Tables

Usually, *raw data* are difficult to interpret because the number of observations increases. Thus, many times the first operation we do with the data is to build a *frequency table*. For a discrete variable, the frequency of a value of the variable is the number of times this particular value appears. The relative frequency is the fraction of times the value appears. We use the R function `table` to get frequency tables:

```
> my.freqTable <- table(my.data.frame$color)
> my.freqTable
```

```
  Blue    Red Yellow
     2      3      1
```

```
> my.relTable <- my.freqTable / nrow(my.data.frame)
> my.relTable
```

```
     Blue       Red    Yellow
0.3333333 0.5000000 0.1666667
```

For continuous variables, we need to arrange the data into classes. For example, if we want to count the positive and negative values for the variable `X1` in the data frame `my.data.frame`, we first create a new factor variable. We can use the `cut` function or indexing techniques.

```
> my.data.frame$X1disc <- cut(my.data.frame$X1,
    c(-Inf, 0, Inf),
    labels = c("negative", "positive"))
> table(my.data.frame$X1disc)
```

```
negative positive
       2        4
```

We can summarize the information of two variables with a 2×2 frequency table:

```
> table(my.data.frame$X1disc, my.data.frame$color)
```

```
          Blue Red Yellow
negative     0   2      0
positive     2   1      1
```

For the sake of simplicity, this sample data frame is very short, but it is possible to have a data set with hundreds or thousands of observations.

9.1.3 Population and Sample

Let us go back to the inference concept. A statistical analysis tries to learn or demonstrate something about a characteristic of a population based on a sample of this population. This is due to the fact that we do not usually have access to all the items of the population, just a reduced sample of them. The reason we cannot access the entire population can be economic (expensive measurement methods), social, legal, or other.

The sampling method will determine the validation of the results. A randomized sample extraction is mandatory in order for the results to be scientifically acceptable. Random number generation has been a big challenge in statistics and computer science in recent decades. Type `?RNG` in the R Console to learn more about how R deals with it. If we have a list with the items of the population, we can extract a randomized sample of them. The *sample size* is the number of items we select from the population, and it will be a critical factor in subsequent analyses.

Example 9.1 (Guitar strings). The quality manager of a company that produces guitar strings wants to study the resistance to tension of the strings produced one day. Obviously, testing the resistance of the whole population is not possible. Every day, the company produces 1,000 strings of each type (E1, B2, G3, D4, A5, E6), and once manufactured, they are packaged and numbered.

To select a random sample of the strings, we use the R function `sample`. For a sample size of 20 items:

```
> sample(1:1000, 20)
```

```
 [1]  90 474 356 216 571   2 714 264 153 288  67 922
[13] 132 803 749 366 937  98 302 485
```

The numbers randomly selected are the number of the strings we have to pick up to make the tension resistance test. Every time you run the command, you get a different sample.

The process is repeated for the six types of strings, and the resistance is measured on a scale that ranges from 1 to 10. The `ss.data.strings` data frame contains the entire data set plus more variables. We will use it throughout this chapter and subsequent ones to illustrate the concepts explained. □

9.1.4 Special Data Values

There are several important issues we should consider regarding the individual values of a sample. One is the fact that sometimes not all data are available. We say that these unavailable data are *missing data*. Before doing the computations needed for the statistical analysis, we will have to decide what to do with these items in the data set. We can simply drop them for a specific calculation, or even delete them from the data frame. Or we can assign a value to these missing "cells" of the matrix. There are several techniques for this task that are outside the scope of this book (the simplest is to assign the mean to these missing values).

In R, a special value is used for missing data: `NA` (Not Available). In many functions, we can pass an argument indicating what to do with missing values. We can also check whether a value is missing using the `is.na` function:

```
> data.test <- c(2, 4, 2, NA, 5)
> is.na(data.test)
```

```
[1] FALSE FALSE FALSE  TRUE FALSE
```

```
> mean(data.test)
```

```
[1] NA
```

```
> mean(data.test, na.rm = TRUE)
```

```
[1] 3.25
```

Another special kind of data we have to be aware of are *outliers*. They are extreme data that are far from the rest of the data. We must analyze them to detect if these outlying data are wrong (e.g., due to typing errors, measurement errors, etc.). The unusual data should be monitored as they might show new trends or changes in our process and eventually become out of control.

A possible decision about outliers can be to disregard or remeasure them (when possible). Finally, sometimes extreme data are outliers for a specific model but not for other more sophisticated models.

9.2 Descriptive

9.2.1 Measures of Central Tendency

The central values of a data set are the simplest way to summarize the data. A central measure value is a number around which the data vary. Three important central tendency measures are mainly used in any statistical analysis:

1. *The sample mean* is the average value. This is the most widely used measure due to its mathematical properties. The main inconvenience is that it is sensitive to outliers (values far from the central values):

$$\bar{x} = \frac{\sum x_i}{n}.$$

2. *The median* is the value that divides the data into two halves: one containing the higher values, the other containing the lower values. It is not influenced by outliers. If we have an even number of data, the average value of the two central values is taken.
3. *The mode* is the most frequent value (or range of values in a continuous variable). In a frequency table, is the value that has the maximum frequency.

Other central values are useful for some disciplines such as the geometric mean and the harmonic mean. Furthermore, some variations of the mean are used for specific data structures, such as the weighted mean and the trimmed mean. References in Sect. 9.4 treat these concepts in detail.

Example 9.2 (Guitar strings (cont.)). The data in `ss.data.strings` data frame contain the resistance test level for each string sampled. To compute the average resistance of all the strings, we use the `mean` function:

```
> mean(ss.data.strings$res)
```

```
[1] 6.666667
```

We have also a function for the median:

```
> median(ss.data.strings$res)
```

```
[1] 7
```

To compute the mode, we first need the frequency table and then search the value that is more frequent:

```
> res.freq <- table(ss.data.strings$res)
> res.freq
```

```
 1  2  3  4  5  6  7  8  9 10
 1  3  9  3  7 23 22 44  5  3
```

```
> res.freq[which(res.freq == max(res.freq))]
```

```
 8
44
```

We obtain only one mean and one median, but we can obtain more than one value for the mode. A symmetric data distribution will have similar values for the mean, the median, and the mode. □

9.2.2 *Measures of Variability*

Variability is statistics' reason for being. With respect to process improvement, it entails a reduction in the variability. It is clearly not enough to study the mean of our process when we are trying to improve it. We need to know how data are clustered around the central values.

The variance is the most important measure of variability due to its mathematical properties. It is the average squared distance from the mean, and we will represent it by $\hat{\sigma}^2$:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}.$$

Another estimator for the variance of a population with better mathematical properties than $\hat{\sigma}^2$ is the sample variance, computed in a slightly different way and represented by s^2:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}.$$

The variance is in square units compared with the mean. Hence, the standard deviation is the most commonly used variability measure:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}.$$

The range (R) is the difference between the maximum value and the minimum value, but it is strongly influenced by extreme values. Nevertheless, when we have few data, it is used as a robust method to estimate the variability (see [69]):

$$R = max(x) - min(x).$$

If we want to measure the variability around the median, the appropriate measure is the median absolute deviation (MAD), that is:

$$MAD = Median(|x_i - Median(x)|).$$

Similarly to the median, we can compute the quartiles. These are the values that divide the data into four parts. Thus the median is the second quartile (Q_2). The first quartile (Q_1) is the value that has 25% of data below it, and the third quartile (Q_3) is the value above which 25% of data remain. The interquartile range (IQR) is a measure of variability that avoids the influence of outliers. This range contains the middle 50% of the data:

$$IQR = Q_3 - Q_1.$$

Example 9.3 (Guitar strings (cont.)). To compute the sample standard deviation of the string resistance, we use the `sd` function:

```
> sd(ss.data.strings$res)
```

```
[1] 1.857681
```

The `IQR` function computes the interquartile range:

```
> IQR(ss.data.strings$res)
```

```
[1] 2
```

We can calculate the quartiles (or even any quantile, e.g., 10%) with the `quantile` function :

```
> quantile(ss.data.strings$res, c(0.10, 0.25, 0.75))
```

```
10% 25% 75%
  3   6   8
```

The `range` function returns a vector with two values: the maximum and the minimum. So to compute the range (R), we apply the `diff` function to this vector:

```
> diff(range(ss.data.strings$res))
```

```
[1] 9
```

We can obtain a summary of the descriptive statistics of a variable by using the `summary` function over it:

```
>  summary(ss.data.strings[, 2:3])
```

```
 type          res
 A5:20   Min.   : 1.000
 B2:20   1st Qu.: 6.000
 D4:20   Median : 7.000
 E1:20   Mean   : 6.667
 E6:20   3rd Qu.: 8.000
 G3:20   Max.   :10.000
```

For a factor (qualitative variable), what you get is the frequency of each level.

□

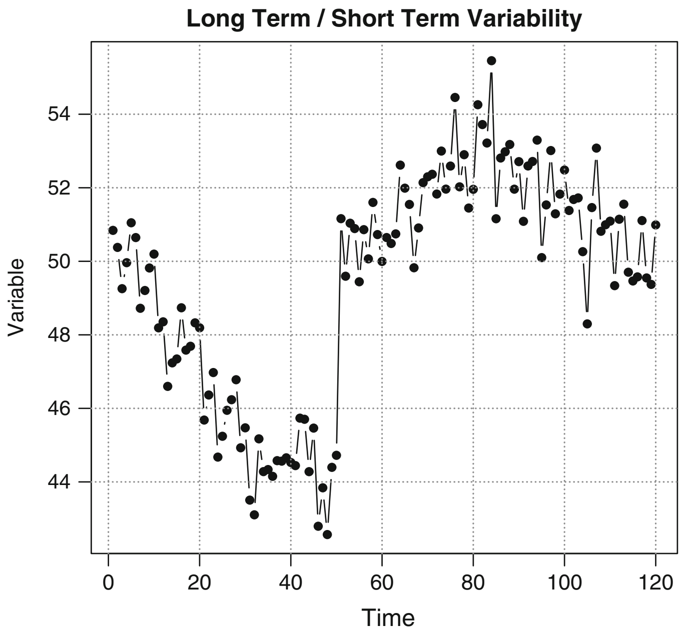


Fig. 9.1 Long-term and short-term variability. In the short term, the variability is constant over time. In the long term, there are ups and downs that reflect assignable causes that should be identified

Long-term and Short-term Variation

In process improvement, the sustainability of the results is an inherent objective. We will use this sustainability concept to support the following explanation.

When we are measuring a process, we take a sample limited to a specific time span (e.g., one day). This data sample will be affected by the so-called short-term variability. This short-term variability is due to common causes, uncontrollable and random.

In the long term, if we take samples at different moments, a long-term variability appears between the successive samples. This long-term variability is due to assignable causes that should be identified and eradicated. An illustrative example is presented in Fig. 9.1.

We can compute the long-term variability as the sample standard deviation of the whole data set. The short-term variability can be computed also with the complete data set using the average range($\bar{R}$):

$$\sigma_{ST} = \frac{\bar{R}}{d_2(2)},$$

where $\bar{R} = \frac{\sum_{i=1}^{n-1} R_i}{n-1}$ and d_2 is the mean of the random variable W (*relative range*) defined as $W = \frac{R}{\sigma}$ for two-sized samples (it only depends on the sample size; its value is 1.128). R_i is, in turn, the range between two subsequent measurements, that is, $R_i = |x_i - x_{i+1}|$.

9.3 Probability

There exist several definitions of probability. One is the relative frequency explained in Sect. 9.1.2, that is, the frequency as a fraction of the sample size. This can be called the *empirical probability*. Another is the classic definition suitable for trials where a finite number of equiprobable events can occur. Here, the probability of an event is the number of favorable cases for this event over the total number of outcomes possible.[2] This definition can be called the *theoretical probability*. The two approaches are linked by the *law of large numbers*, which states that as the sample size increases, the empirical probability is closer to the theoretical probability. Under the Bayesian approach, expert knowledge in the form of the a priori probability distribution is used to set probabilities through a likelihood function, resulting in a posterior probability distribution.

In any case, for discrete sample spaces (the sample space is the set of all possible outcomes of a trial), we have a function that returns the probability of an event:

$$f(x) = P(X = x).$$

For continuous sample spaces, the previous definitions break down, and we use the *cumulative distribution function (cdf)*. This is a function that returns the probability that the outcome of the trial will be less than or equal to a given value[3]:

$$F(x) = P(X \leq x).$$

The *probability density function* is a derivative of the cumulative distribution function, thus:

$$f(x) = \frac{dF(x)}{dx},$$

and we can obtain probabilities using integration:

$$P(a \leq X \leq b) = \int_a^b f(x)dx.$$

[2]For the simplest example of tossing a coin, the total outcomes possible are 2, and the number of favorable cases for the event *Heads* is 1. Thus the probability of getting heads is $\frac{1}{2} = 0.5$.

[3]The cdf is also suitable for discrete distributions.

All in all, probability is a quantitative representation of the likelihood that an event will happen. Thus, a probability is expressed as a number between 0 and 1. An event will never occur if its probability equals 0. An event must occur if its probability equals 1. Another important property is that the sum of the probability of an event (X) and the probability of its complement ($notX$, the event does not occur) equals 1. Then

$$P(not\, X) = 1 - P(X)$$

for discrete distributions, and

$$P(X \geq x) = 1 - P(X \leq x) = 1 - F(x)$$

for continuous distributions.

Example 9.4 (Guitar strings (cont.)). A string is considered defective if it resists less than 3 in the resistance test (that is, 1 or 2). Thus we can calculate the probability of a string's being defective using the empirical definition. The number of defective strings is easily found through the frequency table:

```
> table(ss.data.strings$res)
```

```
 1  2  3  4  5  6  7  8  9 10
 1  3  9  3  7 23 22 44  5  3
```

Or it can be directly computed using indexing and selection in the vector of data:

```
> sum(ss.data.strings$res < 3)
```

```
[1] 4
```

The size of the sample is:

```
> nrow(ss.data.strings)
```

```
[1] 120
```

So the probability of a string's being defective is

```
> sum(ss.data.strings$res < 3) / nrow(ss.data.strings)
```

```
[1] 0.03333333
```

and therefore the probability of its not being defective is

```
> 1 - (sum(ss.data.strings$res < 3) / nrow(ss.data.strings))
```

```
[1] 0.9666667
```

□

9.3.1 Random Variables

A phenomenon is said to be deterministic if its outcome can be precisely known before it occurs. Otherwise, we are talking about stochastic phenomena. Most real-life situations are stochastic, that is, we do not know exactly what will be the result of an event, although we probably are aware of which set of results can occur. When we can assign a value to any of the events that may occur, we have a *random variable*. Random variables can be discrete (the number of possible events are countable) or continuous (the variable can take any value within an interval).

For discrete random variables, the probability distribution assigns a probability to each possible value. For continuous random variables, the probability of a set of values is measured as the area under the curve of the probability density function over this set of values. With these two tools we can compute the counterparts of the mean and the variance of a population. The mean is called the *expectation* and is represented by μ, and the variance is represented by σ. The variance is usually

	Discrete	Continuous
μ	$E[X]=\sum_{i=1}^{n} x_i P(x_i)$	$E[X]=\int_{-\infty}^{\infty} x f(x) dx$
σ	$Var[X]=\sum_{i=1}^{n}(x_i-\mu)^2 P(x_i)$	$Var[X]=\int_{-\infty}^{\infty}(x-\mu)^2 f(x) dx$

calculated using the following equation,[4] suitable for both discrete and continuous distributions:

$$Var[X]=E[X^2]-\big(E[X]\big)^2.$$

Importantly, statistical inference consists in making inferences about a population by estimating its probability distribution, using a sample. Thus, the identification of a probability distribution will help us to make inferences about the population's properties. In the following sections we will explain two of the most important probability distributions (one discrete and one continuous). We will use them to make inferences in Chap. 10.

9.3.2 Binomial Distribution

The binomial distribution is the appropriate distribution to deal with proportions. It is defined as the total number of successes in n independent trials. By independent trial we mean the so-called Bernoulli trial, whose outcome can be success or failure, with p being the probability of success. The binomial distribution is absolutely determined by the parameters n and p, and its probability function is

[4]As a result of the fact that $Var[X]=E[(X-\mu)^2]$.

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x},$$

where $\binom{n}{x} = \frac{n!}{x!(n-x)!}$ and $n!$ (n-factorial) is calculated as

$$n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1.$$

It can be shown that the mean of a binomial distribution is

$$\mu = E[X] = np,$$

and the variance takes the value

$$\sigma^2 = Var[X] = np(1-p).$$

Example 9.5 (Guitar strings (cont.)). In our example, each tension test is a Bernoulli trial that can result in success (usually a defective outcome is considered a success in statistical terms) or failure (not defective). The historical proportion of defective strings in the factory is 1%. Then the random variable defined by "number of defects in a sample of 120 strings" follows a binomial distribution whose parameters are $n = 120$ and $p = 0.01$. The mean and variance of this random variable are $E[X] = n \times p = 1.2$ and $Var[X] = p \times (1-p) = 0.0099$, respectively.

With this information, we can compute the probability of getting exactly four defects with the `dbinom` function:

```
> dbinom(4, 120, 0.01)
```

```
[1] 0.02560162
```

To calculate the distribution function, that is, the probability of getting four or fewer defects, we use the `pbinom` function:

```
> pbinom(4, 120, 0.01)
```

```
[1] 0.9926167
```

For example, the probability of getting more than three defects is $1 - P(X \leq 3)$:

```
> 1 - pbinom(3, 120, 0.01)
```

```
[1] 0.03298491
```

□

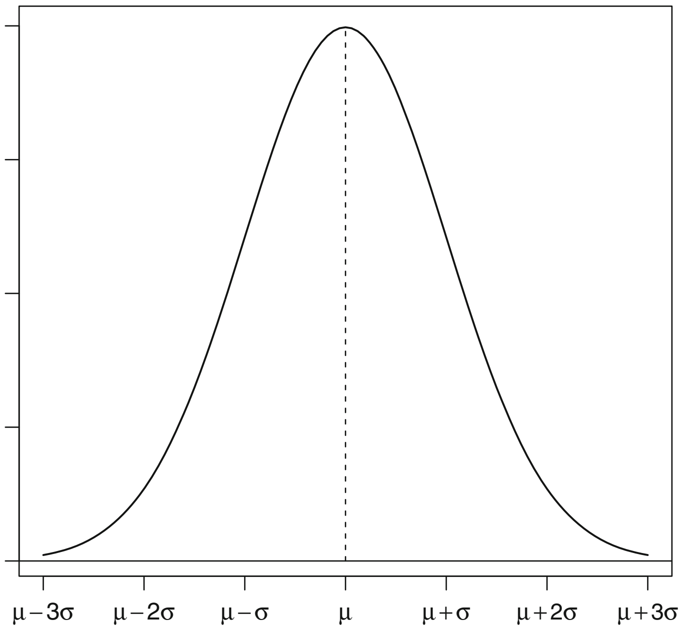


Fig. 9.2 Normal distribution. Between the mean and three standard deviations fall 99.7% of the data. Between the mean and two standard deviations fall 95.5% of the data. Between the mean and one standard deviation fall 68.3% of the data

9.3.3 Normal Distribution

The normal distribution, also known as the Gaussian distribution, is the most important probability distribution for continuous variables. It is determined by two parameters, the mean (which in this case coincides with the median and the mode) and the variance, and has the following probability density function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

whose shape is shown in Fig. 9.2.

The relevance of the normal distribution is due to the *central limit theorem*, which states that the sum of n random variables (regardless of its mean, variance, and distribution) approximates a normal distribution as n increases. Normally, a process is the result of many other subprocesses, and therefore the normal distribution appears in many real-world processes such as human measurements (e.g., height and weight of people) or industrial processes.

Example 9.6 (Guitar strings (cont.)). Before testing the resistance of the sampled strings, some measurements were taken. The variable `len` in the data frame

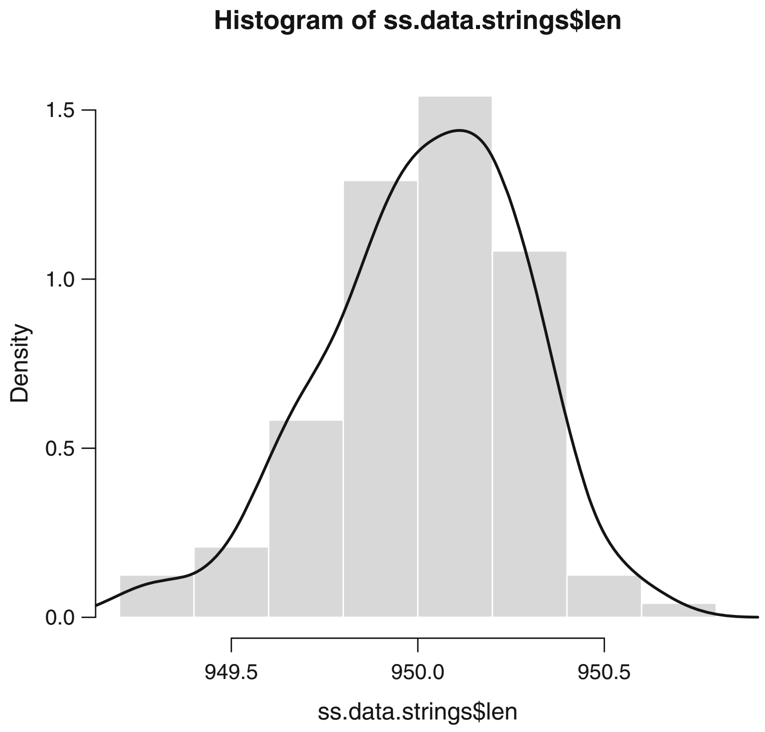


Fig. 9.3 Looking for the normal shape. With the histogram and the density line we can see if the data are normally distributed

`ss.data.strings` contains the length of each string. To see if this variable can be modeled as a normal distribution, we can plot a histogram and a density line (Fig. 9.3):

```
> hist(ss.data.strings$len,
       freq=FALSE,
       col="#CCCCCC",
       border="#FFFFFF")
> lines(density(ss.data.strings$len), lwd=2)
```

We can confirm normality using the quantile–quantile plot (Fig. 9.4), plotted using the following code:

```
> qqnorm(ss.data.strings$len, pch = 16)
> qqline(ss.data.strings$len)
```

If we assume that the length of the strings is normal with mean 950 mm and standard deviation 0.25,[5] the probability of finding a string shorter than 949.5 mm is

[5]We assume that these are the values defined in our manufacturing process.

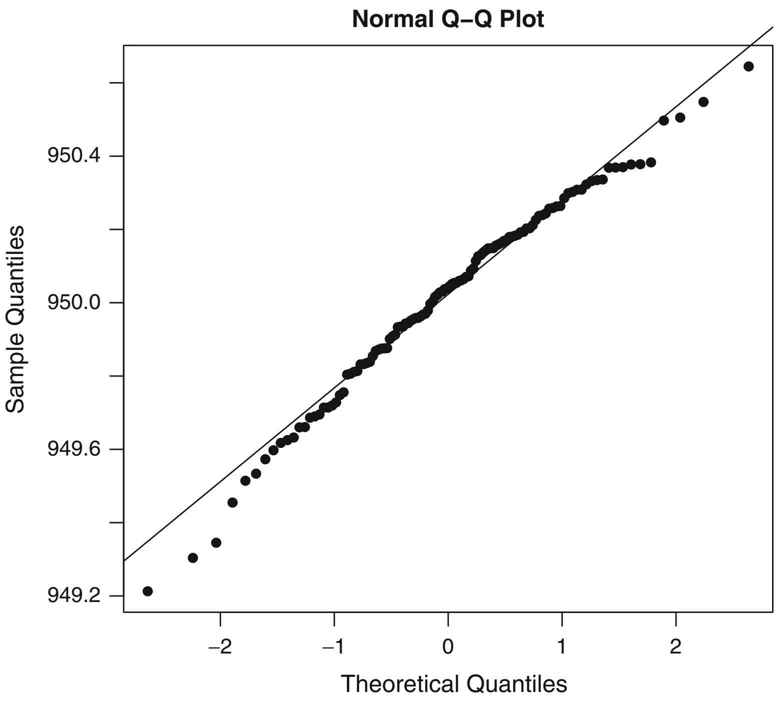


Fig. 9.4 Quantile–quantile plot. If the data are from a normal distribution, the points are approximately over a straight line

```
> pnorm(949.5, 950, 0.25)
```

```
[1] 0.02275013
```

If we want to know the length that is longer than 99% of the strings, we use the `qnorm` function:

```
> qnorm(0.99, 950, 0.25)
```

```
[1] 950.5816
```

Thus, 99% of the strings are shorter than 950.5816 mm. □

A useful operation for the normal distribution is standardization. A standard normal distribution is a normal distribution whose mean is 0 and variance is 1. We can convert any normal distribution with parameters μ and σ into a *Standard Normal Distribution* with the following transformation:

$$Z = \frac{X - \mu}{\sigma}.$$

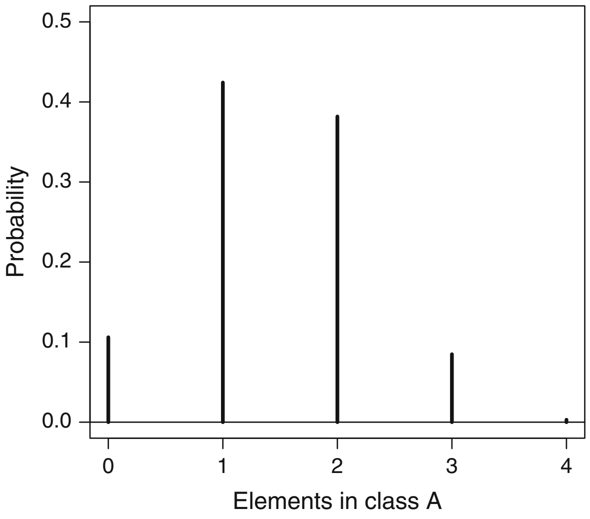


Fig. 9.5 Hypergeometric distribution. Number of elements of class A when extracting a sample of k items from a set of $m+n$ elements, where there are m elements of class A and n elements of class $\bar{A}$ (not A). This distribution is the counterpart of the binomial when the population is finite (small enough). We can use this distribution to approximate the binomial when the ratio sample/population is small (< 0.1). $p = \frac{n}{n+m}$ is the probability of success (class A). $\mu = kp$; $\sigma^2 = kp(1-p)$; $P(x) = \frac{\binom{(n+m)p}{x}\binom{(n+m)(1-p)}{k-x}}{\binom{n+m}{k}}$ R function: `phyper()`

Thus, if $X \sim N(\mu, \sigma)$, then $Z \sim N(0,1)$, which will be useful for inference in the following sections.

9.3.4 Other Useful Distributions

The binomial and normal distributions are the most important ones for process improvement, but there are many others that may describe a process. We include in Figs. 9.5–9.17 a brief summary of some of them, including a graphical example, their means, variances, and probability density functions, as well as the R functions necessary to obtain probabilities. We recommend changing the parameters of the probability density functions plotted to see how they vary. Type `?Distributions` for a complete list of distributions supported by R. See Sect. 9.4 for additional references.

Hypergeometric (discrete) distribution (Fig. 9.5):

```
> p.m = 4; p.n = 7; p.k = 4
> curve(dhyper(x, m = p.m, n = p.n, k = p.k),
    from = 0, to = p.k, type = "h", lwd = 4,
    ylab = "Probability", xlab = "Elements in class A",
    ylim = c(0,0.5), n = 5)
> abline(h = 0)
```

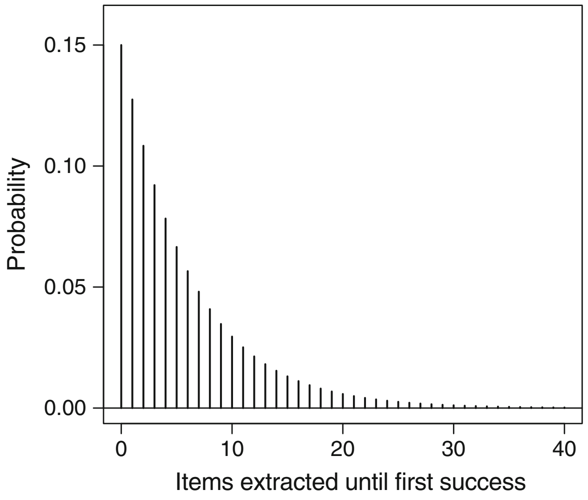


Fig. 9.6 Geometric distribution. Number of trials until first success occurs, where p is the probability of success. $\mu = 1/p$; $\sigma^2 = \frac{(1-p)}{p^2}$; $P(x) = p(1-p)^{x-1}$.R function: `pgeom()`

Geometric (discrete) distribution (Fig. 9.6):

```
> p.prob <- 0.15
> curve(dgeom(x, prob = p.prob),
    from = 0, to = 40, type = "h", lwd = 2,
    ylab = "Probability", xlab = "Items extracted until first
         success",
    ylim = c(0,0.16), n = 41)
> abline(h = 0)
```

Negative binomial (discrete) distribution (Fig. 9.7):

```
> p.size = 3; p.prob=0.2
> curve(dnbinom(x, size = p.size, prob = p.prob),
    from = 3, to = 40, type = "h", lwd = 2,
    ylab = "Probability", xlab = "Number of trials until 3
       events",
    ylim = c(0,0.08), n = 38)
> abline(h = 0)
```

Poisson (discrete) distribution (Fig. 9.8):

```
> p.lambda = 5
> curve(dpois(x, lambda = p.lambda),
    from = 0, to = 20, type = "h", lwd = 4,
    ylab = "Probability", xlab = "Number of successful
       experiments per unit",
    ylim = c(0,0.20), n = 21)
> abline(h = 0)
```

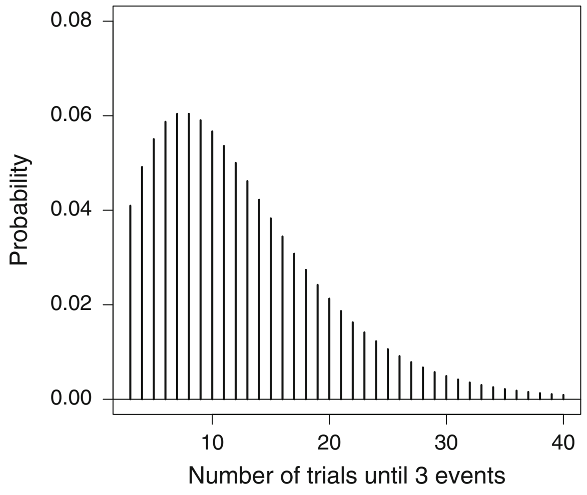


Fig. 9.7 Negative binomial distribution. Number of Bernoulli trials until r successes are achieved. The geometric distribution is a special case of the negative binomial when $r = 1$.
$\mu = \frac{r}{p}$; $\sigma^2 = \frac{r(1-p)}{p^2}$;
$P(x) = \binom{x-1}{r-1} p^r (1-p)^{x-r}$.
R function: `pnbinom()`

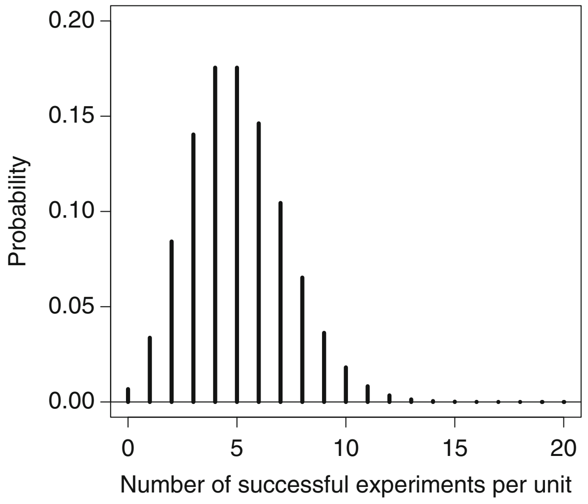


Fig. 9.8 Poisson distribution. The Poisson distribution is suitable when we are counting the number of events per unit (time, area, volume, etc.). The parameter λ is the average number of times the event occurs, e.g., the number of errors per hour in a service.
$\mu = \lambda$;$\sigma^2 = \lambda$;
$P(x) = \frac{e^{-\lambda}\lambda^x}{x!}$.
R function: `ppois()`

Exponential (continuous) distribution (Fig. 9.9):

```
> p.rate = 1
> curve(dexp(x, rate = p.rate),
    from = 0, to = 5, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,1))
> abline(h = 0)
```

lognormal (continuous) distribution (Fig. 9.10):

```
> p.meanlog = 0; p.sdlog = 1
> curve(dlnorm(x, meanlog = p.meanlog, sdlog = p.sdlog),
    from = 0, to = 6, type = "l", lwd = 2,
```

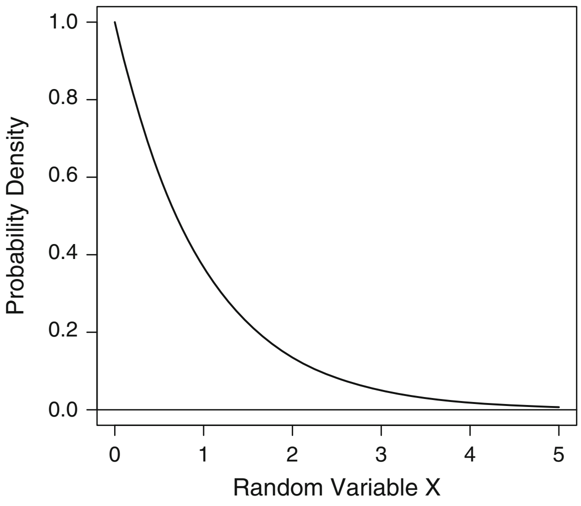


Fig. 9.9 Exponential distribution. The exponential distribution appears when we measure the time until an event occurs in a Poisson process (e.g., number of clients per hour for a service, number of lost packages per minute in a network). The rate is the λ parameter of the distribution (e.g., 20 clients per hour).
$\mu = \frac{1}{\lambda}$; $\sigma^2 = \frac{1}{\lambda^2}$;
$f(x) = \lambda e^{-\lambda x}$.
R function: `pexp()`

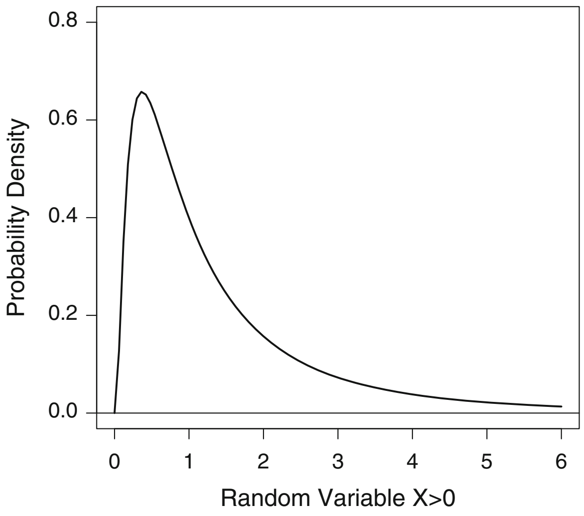


Fig. 9.10 Lognormal distribution. When the logarithm of a variable follows a normal distribution, the variable is lognormal. This fact appears in many natural processes as a result of the central limit theorem. Let μ_N and σ_N^2 be the mean and variance respectively of the variable $\log(X)$.
$\mu = e^{\mu_N + \frac{\sigma^2}{2}}$;
$\sigma^2 = e^{2\mu_N + \sigma_N^2}(e^{\sigma_N^2} - 1)$;
$f(x) = \frac{1}{x\sigma_N\sqrt{2\pi}} exp\left[-\frac{(ln(x)-\sigma_N)^2}{2\sigma_N^2}\right]$. R function: `plnorm()`

```
    ylab = "Probability Density", xlab = "Random Variable X>0
        ",
    ylim = c(0,0.8))
> abline(h = 0)
```

Uniform (continuous) distribution (Fig. 9.11):

```
> p.min = 0; p.max = 1
> curve(dunif(x, min = p.min, max = p.max),
    from = -0.5, to = 1.5, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,1.2))
> abline(h = 0)
```

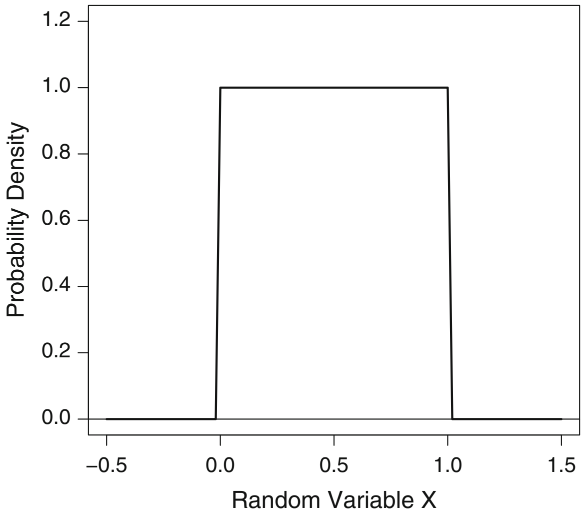


Fig. 9.11 Uniform distribution. In a uniform distribution between *a* and *b*, the probability density is constant in the interval [*a b*]. The uniform distribution is especially useful for generating random numbers. We use the R function `runif` for this purpose.
$\mu = \frac{b+a}{2}$;
$\sigma^2 = \frac{1}{12}(b-a)^2$;
$f(x) = \frac{1}{b-a}, a \leq x \leq b$.
R function: `punif()`

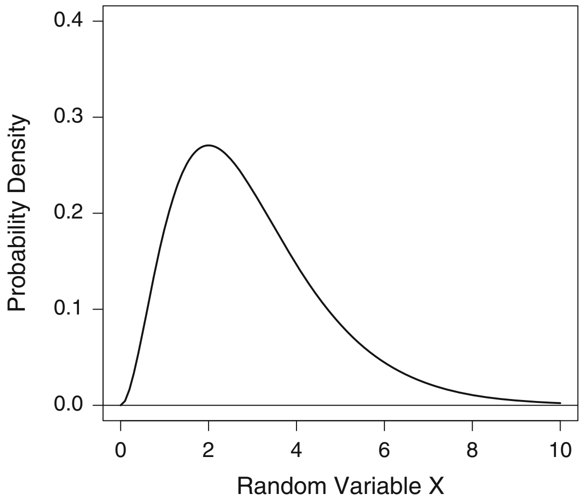


Fig. 9.12 Gamma distribution. The gamma distribution is defined by two parameters: shape (r) and rate (λ). By changing the parameters of a gamma distribution, we get different shapes.
$\mu = \frac{r}{\lambda}$;
$\sigma^2 = \frac{r}{\lambda^2}$;
$f(x) = \frac{1}{(r-1)!}\lambda^r x^{r-1} e^{-\lambda x}$.
R function: `pgamma()`

Gamma (continuous) distribution (Fig. 9.12):

```
> p.shape = 3; p.rate = 1
> curve(dgamma(x, shape = p.shape, rate = p.rate),
    from = 0, to = 10, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,0.4))
> abline(h = 0)
```

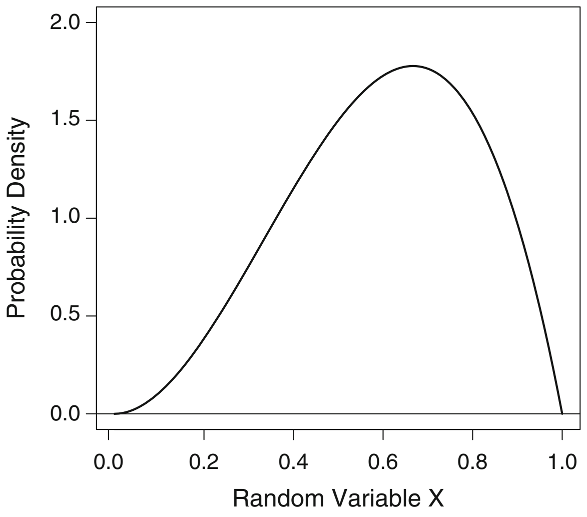


Fig. 9.13 Beta distribution. The beta distribution has two shape parameters (say, *a* and *b*). It is used in Bayesian inference, as it appears as the a priori distribution of other distributions. It is defined for values between 0 and 1.
$\mu = \frac{a}{a+b}$;
$\sigma^2 = \frac{ab}{(a+b)^2(a+b+1}$;
$f(x) = kx^{a-1}(1-x)^{b-1}$;
$k = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$.
R function: `pbeta()`

Beta (continuous) distribution (Fig. 9.13):

```
> p.shape1 = 3; p.shape2 = 2
> curve(dbeta(x, shape1 = p.shape1, shape2 = p.shape2),
    from = 0, to = 1, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,2))
> abline(h = 0)
```

Weibull (continuous) distribution (Fig. 9.14):

```
> p.shape = 2; p.scale = 2.5
> curve(dweibull(x, shape = p.shape, scale = p.scale),
    from = 0, to = 6, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,0.6))
> abline(h = 0)
```

Student's *t* (continuous) distribution (Fig. 9.15):

```
> p.df = 19
> curve(dt(x, df = p.df),
    from = -5, to = 5, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,0.5))
> abline(h = 0)
> lines(rep(qt(0.95, 19),2), c(0,dt(qt(0.95, 19),19)))
> text(qt(0.95, 19),0,paste(round(qt(0.95, 19),2)), adj=c
    (0.5,1.1))
> text(0,0.2,"95%")
> text(2,0,"5%", adj=c(0.1,-0.5))
```

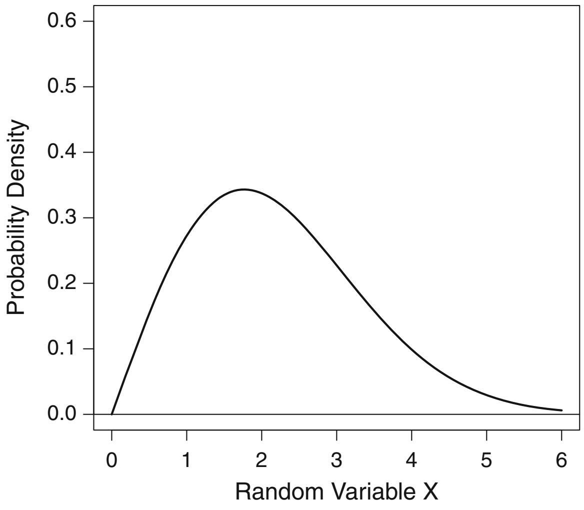


Fig. 9.14 Weibull distribution. The Weibull distribution can be modeled by two parameters: scale (λ) and shape (k). The Weibull distribution model has many *time-to-failure* random variables and is widely used in reliability analysis. $\mu = \lambda\Gamma\left(1+\frac{1}{k}\right)$; $\sigma^2 = \lambda^2\left[\Gamma\left(1+\frac{2}{k}\right) - \Gamma^2\left(1+\frac{1}{k}\right)\right]$; $\Gamma(n) = (n-1)!$; $f(x) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{(-x/\lambda)^k}$. R function: `pweibull()`

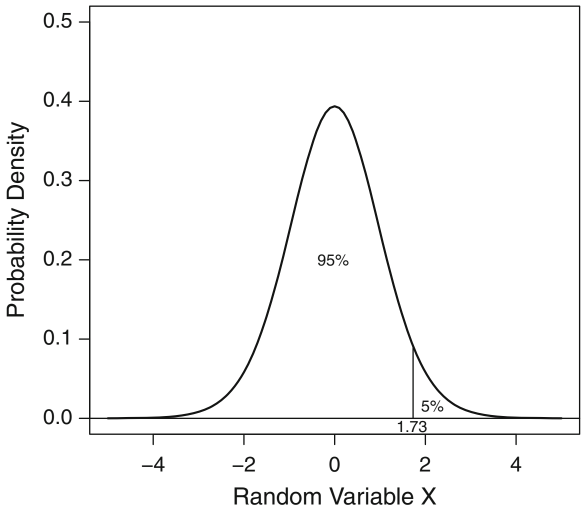


Fig. 9.15 Student's t distribution. Student's t distribution has only one parameter: degrees of freedom. This is a sampling distribution used in inference. Its main use is for finding quantiles for a given confidence level or significance level. We use the R function `qt`. For example, to find the value $t_{19,0.95}$ (19 degrees of freedom) with 0.95 probability below), type `qt(0.95, 19)`. R function: `pt()`

Chi-square (χ^2)(continuous) distribution (Fig. 9.16):

```
> p.df = 19
> curve(dchisq(x, df = p.df),
    from = 5, to = 50, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,0.08))
> abline(h = 0)
> lines(rep(qchisq(0.95, 19),2), c(0,dchisq(qchisq(0.95, 19)
    ,19)))
```

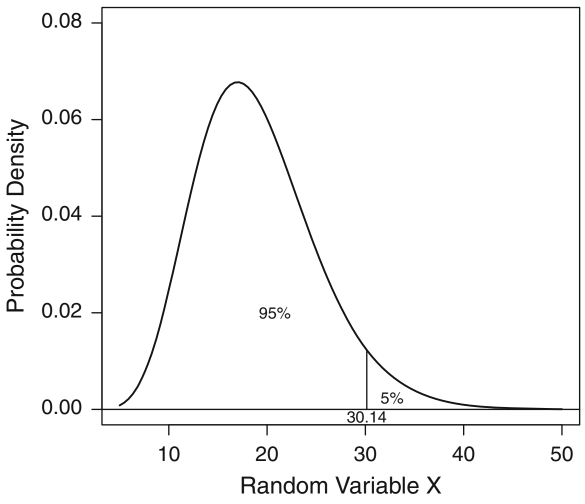


Fig. 9.16 Chi-square (χ^2) distribution. This is another sampling distribution used in inference, with degrees of freedom as parameter. Similarly to Student's *t* distribution, we can find quantiles for a given confidence level or significance level using R. To find the value $\chi^2_{19,0.95}$ (19 degrees of freedom) with 0.95 probability below, type `qchisq(0.95, 19)`. R function: `pchisq()`

```
> text(qchisq(0.95, 19),0,paste(round(qchisq(0.95, 19),2)),
    adj=c(0.5,1.1))
> text(20,0.02,"95%")
> text(32,0,"5%", adj=c(0.1,-0.5))
```

F (continuous) distribution (Fig. 9.17):

```
> p.df1 = 19; p.df2 = 15
> curve(df(x, df1 = p.df1, df2 = p.df2),
    from = 0, to = 4, type = "l", lwd = 2,
    ylab = "Probability Density", xlab = "Random Variable X",
    ylim = c(0,1))
> abline(h = 0)
> lines(rep(qf(0.95, 19, 15), 2), c(0, df(qf(0.95, 19, 15)
    ,19, 15)))
> text(qf(0.95, 19, 15),0,paste(round(qf(0.95, 19, 15), 2)),
    adj=c(0.5,1.1))
> text(1,0.4,"95%")
> text(2.5,0,"5%", adj=c(0.1,-0.5))
```

9.4 Summary and Further Reading

In this chapter we provided a brief review of the most important concepts of statistics and probability. Although this is the toughest part for a Six Sigma practitioner, it is very important to master these concepts in order to apply the scientific method and reach valid results. In the next chapter, statistical inference is tackled to complete the background needed for the Analyze phase of the DMAIC cycle.

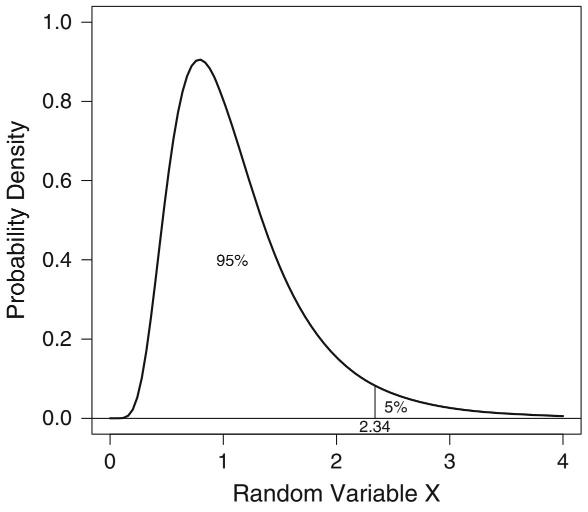


Fig. 9.17 *F* distribution. The *F* distribution is the last sampling distribution we will mention in this book. It has two parameters, both for degrees of freedom, as it is a quotient between two χ^2 variables over their degrees of freedom: $F_{n,m} = \frac{\chi^2_n/n}{\chi^2_m/m}$. To find the value $F_{19,15,0.95}$ (19 and 15 degrees of freedom) with 0.95 probability below, type `qf(0.95, 19)`. R function: `pf()`

To enhance your knowledge of probability and statistics, see [45], on both statistics and R, and [40], on probability. Reference [20] is another excellent book regarding statistics with R, and [69] contains thorough statistical explanations.

Case Study

Compute descriptive statistics for your paper helicopter data, including measurements of central tendency and variability. Decide on a threshold for your correct helicopter and make a table with the number of correct and incorrect items (you should have several of them).

Now set a percentage of defects you are allowed to manufacture, and compute the probability of having the actual number of defects (use the binomial distribution). Analyze the flight time, and find out which distribution the data follows. Is it normal? Compute the percentage of defective items you will have in the long term under the threshold defined in the previous paragraph.

Practice

9.1. Using the `ss.data.strings` data frame, save in a new data frame the data corresponding to type E1 strings. Compute descriptive statistics for the resistance. Do you think it is a symmetric distribution?

9.2. In an industrial process, the proportion of defects is 0.001. If we take a sample of 100 items, what is the probability of having less than 5 defective units?

Chapter 10
Statistical Inference with R

All models are wrong; some models are useful.
George E.P. Box

10.1 Introduction

Statistical inference is the branch of statistics whereby we arrive at conclusions about a population through a sample of the population. We can make inferences concerning several issues related to the data, for example, the parameters of the probability distribution, the parameters of a given model that explains the relationship between variables, goodness of fit to a probability distribution, and differences between groups (e.g., regarding the mean or the variance).

In Six Sigma projects, improvement is closely linked to the effect that some parameters of the process (input) have on features of the process (output). Statistical inference provides the necessary scientific basis to achieve the goals of the project and validate its results.

In this chapter, some basic statistical inference tools and techniques suitable for Six Sigma are reviewed. In Sect. 10.2, confidence intervals and point estimation are explained. Hypothesis-testing concepts are very important for every inference analysis. You can read about them in Sect. 10.3. Regression and analysis of variance (ANOVA) are the main techniques to study the relationship between variables. Sections 10.4 and 10.5 explain how to use them with R.

10.2 Confidence Intervals

10.2.1 Sampling Distribution and Point Estimation

Through point estimation, one or more parameters of a population's probability distribution can be inferred using a sample. A function over the values of the sample is called a *statistic*. For example, the sample mean is a statistic. When we make inferences about the parameters of a population's probability distribution, we use statistics. These statistics have, in turn, a probability distribution. That is, for every sample extracted from a given population, we have a value for the statistic. In this way, we may build a new population of values (of the statistic) that follows some probability distribution.

The probability distribution of a statistic is a *sampling distribution*, and the properties of this distribution allow us to know if the statistic is a good estimator of the parameter under study. Some important properties to study about a statistic to find out if it is a good estimator are unbiasedness, mean square error, consistency, and asymptotic distribution. An estimator is unbiased if its expectation equals the real value of the parameter.

To distinguish the actual value of a parameter from its estimator, a *hat* ($\hat{}$) is placed over the symbol of the estimator (e.g., $\hat{\sigma}^2$ is the estimator for the variance σ^2). We will not explain in detail the properties of the statistics or explain how to study sampling distributions. We will simply introduce some of the most important statistics for estimating proportions, means, and variances.

For binomial distributions, the sample proportion is an unbiased estimator:

$$\hat{p} = \frac{x}{n}.$$

That is, the number of events (x) in n Bernoulli experiments over the number of experiments. The mean and the variance of the statistic are[1]

$$\mu_{\hat{p}} = p,$$
$$\sigma^2_{\hat{p}} = \frac{p(1-p)}{n}.$$

The sample mean is an unbiased estimator of the population mean:

$$\hat{\mu} = \bar{x},$$
$$\mu_{\bar{x}} = \mu,$$
$$\sigma^2_{\bar{x}} = \frac{\sigma^2}{n}.$$

For the variance the unbiased estimator is the sample variance, defined as

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1},$$

whose expectation is the population variance ($E[\hat{s^2}] = \sigma^2$).

[1]For sampling distributions, we set the symbol of the statistic a subscript.

The sample standard deviation is not an unbiased estimator of the population standard deviation. In its place we can use the following unbiased estimator:

$$\hat{\sigma} = \frac{s}{c_4},$$

where c_4 is a constant that depends only on the sample size:

$$c_4 = \left(\frac{2}{n-1}\right)^{\frac{1}{2}} \frac{\Gamma(n/2)}{\Gamma\left[(n-1)/2\right]},$$

where $\Gamma(\cdot)$ can be evaluated using the `gamma` R function. You can compute this constant using the `ss.cc.getc4` function in the `SixSigma` package.

An unbiased estimator for the standard deviation σ of a normal distribution used in many applications of engineering statistics and quality control is

$$\hat{\sigma} = \frac{R}{d_2},$$

where R is the range of the data in the sample (that is, the difference between the maximum and the minimum) and d_2 is the mean of the random variable W (*relative range*) defined as

$$W = \frac{R}{\sigma}.$$

The mean of this random variable is constant and depends only on the sample size n. It can be computed numerically using integration. The `ss.cc.getd2` function[2] in the `SixSigma` package returns the constant d_2 for a given sample size:

```
> ss.cc.getd2(20)
```

```
      d2
3.734949
```

Example 10.1 (Guitar strings). Let us reproduce the example in Chap. 9. The quality manager of a company that produces guitar strings wants to study the resistance to tension of the strings produced one day. Obviously, testing the resistance of the whole population is not possible. Every day, the company produces 1,000 strings of each type (E1, B2, G3, D4, A5, E6), and once the strings are made, they are packaged and numbered.

We can do computations over the sample data to obtain estimators of the mean and variance[3] for tension resistance in the `ss.data.strings` data set:

[2]It in turn uses the `ptukey` function (see `?ptukey`).

[3]R provides the sample variance. If you need the population variance, just multiply it by $\frac{n-1}{n}$.

```
> mean(ss.data.strings$res)
```

```
[1] 6.666667
```

```
> var(ss.data.strings$res)
```

```
[1] 3.45098
```

For the proportion of defective strings, the estimator $\hat{p}$ is

```
> sum(ss.data.strings$res<3)/nrow(ss.data.strings)
```

```
[1] 0.03333333
```

The unbiased estimator for the standard deviation using the sample standard deviation can be computed as follows:

```
> c(Sigma.Est = sd(ss.data.strings$res)/ss.cc.getc4(nrow(ss.
    data.strings)))
```

```
Sigma.Est.c4
    1.861588
```

We can estimate the standard deviation through the range, using the constant d_2:

```
> c(Sigma.Est = diff(range(ss.data.strings$res))/ss.cc.getd2(
    nrow(ss.data.strings)))
```

```
Sigma.Est.d2
    1.749553
```

□

We have obtained an estimation for the parameter of interest for our process. However, any estimation is linked to some uncertainty, and therefore we will have to deal with some *error level*. To quantify this uncertainty, we use interval estimation. Interval estimation consists in giving bounds for our estimation (LL and UL, upper and lower limits). These limits are calculated so that we have confidence in the fact that the real value of the parameter is contained within them. This fact is stated as a *confidence level* and expressed as a percentage. The confidence level reflects the percentage of times that the real value of the parameter is assumed to be in the interval when repeating the sampling. Usually the confidence level is represented by $100 \times (1 - \alpha)\%$, with α the confidence coefficient. The confidence coefficient is a measure of the error in our estimation. Common values for the confidence level are 99, 95, or 90%, corresponding, respectively, to $\alpha = 0.01$, $\alpha = 0.05$, and $\alpha = 0.1$.

A confidence interval is expressed as an inequality.[4] If θ is a parameter, then $[LL, UL]$ means $LL \leq \theta \leq UL$.

[4]In Bayesian statistics, the *credible interval* is the counterpart of the confidence interval, which has a probabilistic meaning.

10.2.2 Proportion Confidence Interval

As we explained in Chap. 9, when we are dealing with proportions, the binomial distribution is the appropriate probability distribution to model a process. The typical application of this model is the calculation of the fraction of nonconforming items in a process.

Due to the central limit theorem (see Sect. 9.3.3 in Chap. 9), we can construct a confidence interval for the proportion using the following formula:

$$\hat{p} \pm z_{1-\frac{\alpha}{2}} \sqrt{\frac{\hat{p}(1-\hat{p})}{n}},$$

where $z_{1-\frac{\alpha}{2}}$ is the quantile of the standard normal distribution that leaves a probability of $\frac{\alpha}{2}$ on the right-hand side. This is the *classical* way to construct a confidence interval for the proportion when the sample size n is large and $\hat{p}$ is not small (under these circumstances, the normal distribution can be used to approximate the binomial distribution). The R function `binom.test` provides an exact confidence interval for the probability of success. However, in [1] it is shown that "approximate results are sometimes more useful than exact results, because of the inherent conservativeness of exact methods." In this regard, the approximation performed by the `prop.test` function may provide more accurate results. Likewise, the `binconf` function in the `Hmisc` package allows us to use different methods and compare the results.

Example 10.2 (Guitar strings (cont.)). To construct a 95% confidence interval for the proportion of defective strings using the normal approximation, we first obtain the value of the standard normal distribution for a probability of $\frac{\alpha}{2} = 0.025$:

```
>   def.z <- qnorm(0.975)
```

The point estimator for the proportion is

```
>   est.p <- sum(ss.data.strings$res<3)/nrow(ss.data.strings)
```

Therefore, our confidence interval is

```
>   est.p + (c(LL=-1,UL=1) * def.z * sqrt((est.p*(1-est.p))/
    nrow(ss.data.strings)))
```

```
         LL          UL
0.001216316 0.065450351
```

The previous normal approximation may not be adequate, given that p is small. A suitable alternative is to use the `prop.test` function, which provides the following interval:

```
> (prop.test(x = sum(ss.data.strings$res<3),
    n = nrow(ss.data.strings)))$conf.int
```

```
[1] 0.01073155 0.08825263
attr(,"conf.level")
[1] 0.95
```

We can compare different methods with the `binconf` function:

```
> require("Hmisc")
> binconf(x = sum(ss.data.strings$res<3),
    n = nrow(ss.data.strings), method = "all")
```

```
              PointEst       Lower      Upper
Exact       0.03333333 0.009155506 0.08314872
Wilson      0.03333333 0.013037551 0.08258034
Asymptotic  0.03333333 0.001216316 0.06545035
```

As a general rule, the result of the `prop.test` is the best option. □

10.2.3 Mean Confidence Interval

Thanks to the central limit theorem, for large[5] sample sizes we can construct confidence intervals for the mean of any distribution using the following formula:

$$\bar{x} \pm z_{\alpha/2}\frac{\sigma}{\sqrt{n}}.$$

Usually, σ is unknown. In this case, instead of σ and the normal quantile z, we must use the sample standard deviation (s) and Student's t quantile with $n-1$ degrees of freedom ($t_{\alpha/2,n-1}$). A thorough explanation of this important concept is outside the scope of this book. The degrees of freedom can be thought of as the number of data minus the number of constraints used to estimate the parameter under study. Therefore, the confidence interval takes the following form:

$$\bar{x} \pm t_{\alpha/2,n-1}\frac{s}{\sqrt{n}}.$$

Example 10.3 (Guitar strings (cont.)). We want to build a 95% confidence interval for the mean of the tension resistance of the strings in the data set `ss.data.strings`. Assuming that the population variance is unknown, we need the appropriate value of the t distribution:

```
> res.t <- qt(0.975, nrow(ss.data.strings) - 1)
> res.t
```

```
[1] 1.9801
```

[5]A sample size $n \geq 30$ is considered large.

This value will be multiplied by the standard deviation of the mean, that is:

```
> sd.mean <- sd(ss.data.strings$res)/sqrt(nrow(ss.data.
    strings))
> sd.mean
```

```
[1] 0.1695823
```

And finally we can give the confidence interval:

```
> mean(ss.data.strings$res) + c(LL = -1, UL = 1) * (res.t *
    sd.mean)
```

```
      LL       UL
6.330877 7.002457
```

If we know the actual value of the population variance, then we must use a different formula to construct the confidence interval. For example, if we know (through historical data or any other accepted procedure) that the variance of the resistance is 4, then we construct the confidence interval using the following code:

```
> res.z <- qnorm(0.975)
> sd.mean <- 2 / sqrt(nrow(ss.data.strings))
> mean(ss.data.strings$res) + c(LL = -1, UL = 1) * (res.z *
    sd.mean)
```

```
      LL       UL
6.308828 7.024505
```

□

10.2.4 *Variance Confidence Interval*

Sometimes we need to find out if the variance of a process is within a given range. Confidence intervals are a fast way to verify this issue. There are two important differences between mean and variance confidence intervals:

- Methods for variance are more sensitive to the normality assumption. Thus, a normality test is advisable for validating results.
- The statistic used to construct the confidence interval (χ^2) for the variance is not symmetric, unlike z or t. Therefore, the limits are not symmetric with respect to the point estimator.

The formulas to construct the confidence interval are as follows:

$$\frac{(n-1)s^2}{\chi^2_{1-\alpha/2,n-1}} \leq \sigma \leq \frac{(n-1)s^2}{\chi^2_{\alpha/2,n-1}}.$$

Example 10.4 (Guitar strings (cont.)). In the guitar string factory, the length of each string was measured before the resistance was tested. To compute a confidence interval for the variance of the length, we first compute the χ^2 quantiles:

```
> len.chi <- c(qchisq(0.975, nrow(ss.data.strings)),
        qchisq(0.025, nrow(ss.data.strings)))
> len.chi
```

```
[1] 152.21140  91.57264
```

Next, we calculate the bounds of the confidence interval:

```
> len.ci <- ((nrow(ss.data.strings)-1) * var(ss.data.strings$
    len)) / len.chi
> len.ci
```

```
[1] 0.05591480 0.09294119
```

We can obtain a confidence interval for the mean along with a graphical output and a normality test with the function `ss.ci` in the `SixSigma` package (Fig. 10.1):

```
> ss.ci(len, data = ss.data.strings, digits = 3)
```

```
        Mean = 950.016; sd = 0.267
        95% Confidence Interval= 949.967 to 950.064

      LL        UL
949.9674 950.0640
```

□

Confidence intervals can also be obtained using the functions for hypothesis testing available in R (we will talk about hypothesis tests in the next section). The hypothesis testing functions in R return an `htest` object. One of the components of these types of objects is `conf.int`, which contains a confidence interval for the data and the confidence level specified.

Example 10.5 (Guitar strings (cont.)). To obtain automatically a confidence interval for the mean of the length of the strings, we first save the object returned by the `t.test` function and then extract the `conf.int` component:

```
> my.test <- t.test (ss.data.strings$len)
> my.test$conf.int
```

```
[1] 949.9674 950.0640
attr(,"conf.level")
[1] 0.95
```

□

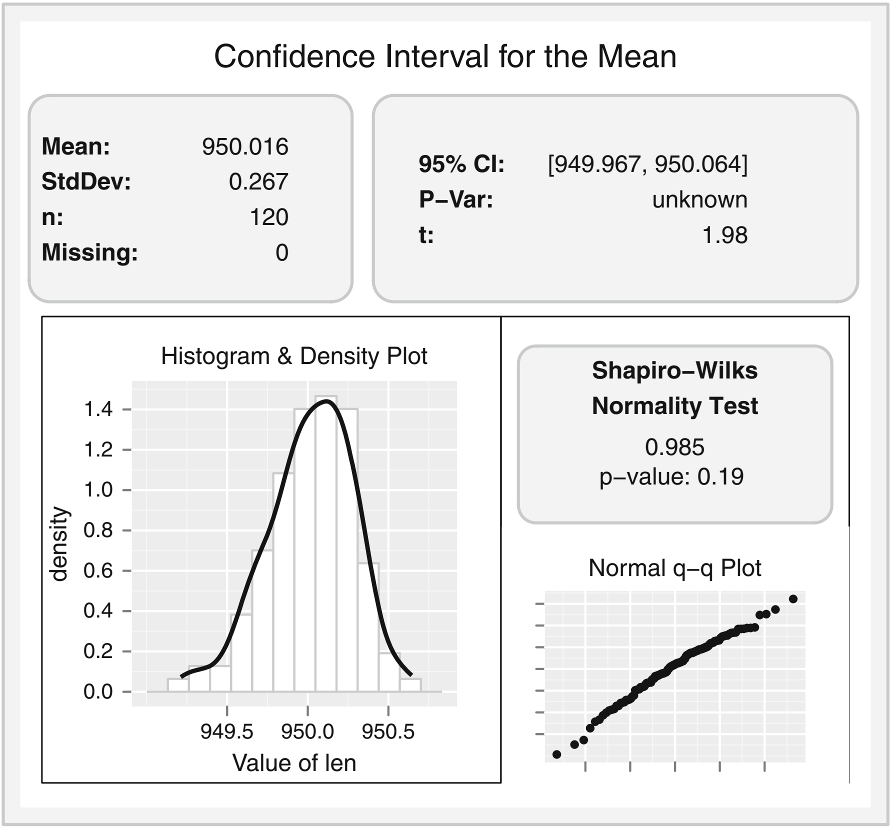


Fig. 10.1 Confidence interval for guitar string example. The output shows the confidence interval and the facts used to construct it. A normality test (including a quantile–quantile plot) and a histogram with a density plot are shown to validate the model assumptions

10.3 Hypothesis Testing

In statistical inference, hypothesis testing is intended to confirm or validate some conjectures about the process we are analyzing. Importantly, these hypotheses are related to the parameters of the probability distribution of the data. For example, if we have data from a process that are normally distributed and we want to verify if the mean of the process has changed with respect to the historical mean, we should make the following hypothesis test:

$$H_0 : \mu = \mu_0,$$

$$H_1 : \mu \neq \mu_0,$$

where H_0 denotes the *null hypothesis* and H_1 denotes the *alternative hypothesis*. Thus we are testing H_0 ("the mean has not changed") vs. H_1 ("the mean has changed").

Hypothesis testing can be performed in two ways: one-sided tests and two-sided tests. An example of the latter is when we want to know if the mean of a process has increased:

$$H_0 : \mu = \mu_0,$$
$$H_1 : \mu > \mu_0.$$

Hypothesis testing tries to find evidence about the refutability of the null hypothesis using probability theory.[6] We want to check if a new situation (represented by the alternative hypothesis) is arising. Subsequently, we will reject the null hypothesis if the data do not support it with "enough evidence." The threshold for enough evidence is decided by the analyst, and it is expressed as a significance level α (similarly to the confidence intervals explained in Sect. 10.2). A 5% significance level is a widely accepted value in most cases.

To verify whether the data support the alternative hypothesis, a statistic (related to the underlying probability distribution) is calculated. If the value of the statistic is within the rejection region, then the null hypothesis is rejected. If the statistic is outside the rejection region, then we say that we do not have enough evidence to accept the alternative hypothesis (perhaps it is true, but the data do not support it).

Usually the refutability of the null hypothesis is assessed through the p-value stemmed from the hypothesis test. If the p-value is larger than α, then H_0 should not be rejected, otherwise H_0 must be rejected. The p-value is sometimes interpreted as the probability that the null hypothesis is true. This interpretation is not correct. The p-value is the probability of finding a more extreme sample (in the sense of rejecting H_0) than the one that we are currently using to perform the hypothesis test. So if the p-value is small, the probability of finding a more extreme sample is small, and therefore the null hypothesis should be rejected. Otherwise, if the p-value is large, the null hypothesis should not be rejected. The concept of "large" is determined by the significance level (α). In practice, if $p < \alpha$, then H_0 is rejected. Otherwise, H_0 is not rejected. Thus, for instance, if the confidence level is 95% ($\alpha = 0.05$) and the p-value is smaller than 0.05, we do not accept the null hypothesis taking into account empirical evidence provided by the sample at hand.

There are some functions in R to perform hypothesis tests, for example, `t.test` for means, `prop.test` for proportions, `var.test` and `bartlett.test` for variances, `chisq.test` for contingency table tests and goodness-of-fit tests, `poisson.test` for Poisson distributions, `binom.test` for binomial distributions, `shapiro.test` for normality tests. Usually, these functions also provide a confidence interval for the parameter tested.

[6] We will not explain in detail the foundations of hypothesis testing. Some good references can be found in Sect. 10.6.

Example 10.6 (Guitar strings (cont.)). We can perform a hypothesis test to verify if the length of the strings is different from the target value of 950 mm:

$$H_0 : \mu = 950,$$
$$H_1 : \mu \neq 950.$$

```
> t.test(ss.data.strings$len,
    mu = 950,
    conf.level = 0.95)
```

```
        One Sample t-test

data:  ss.data.strings$len
t = 0.6433, df = 119, p-value = 0.5213
alternative hypothesis: true mean is not equal to 950
95 percent confidence interval:
 949.9674 950.0640
sample estimates:
mean of x
 950.0157
```

As the p-value is (much) greater than 0.05, we accept that the mean is not different from the target.

We can also compare two means, for example, for two types of strings. To compare the length of the string types E6 and E1, we use the following code:

```
>data.E1 <- ss.data.strings$len[ss.data.strings$type == "E1"]
>data.E6 <- ss.data.strings$len[ss.data.strings$type == "E6"]
>t.test(data.E1, data.E6)
```

```
        Welch Two Sample t-test

data:  data.E1 and data.E6
t = -0.3091, df = 36.423, p-value = 0.759
alternative hypothesis: true difference in means is not equal
     to 0
95 percent confidence interval:
 -0.1822016  0.1339911
sample estimates:
mean of x mean of y
 949.9756  949.9997
```

Again, we cannot accept the alternative hypothesis, and therefore we do not reject that there are no differences between the length of the two types of strings (that is, we do not reject the null hypothesis). If we want to compare the variances between the two types of strings, we can use the `var.test` function:

```
> var.test(data.E1, data.E6)
```

```
        F test to compare two variances

data:  data.E1 and data.E6
F = 1.5254, num df = 19, denom df = 19, p-value
= 0.3655
alternative hypothesis: true ratio of variances is not equal
    to 1
95 percent confidence interval:
 0.6037828 3.8539181
sample estimates:
ratio of variances
          1.525428
```

In this case, the statistic used is the ratio between variances, and the null hypothesis is "the ratio of variances is equal to 1," that is, the variances are equal.

We can also compare proportions using the `prop.test` function. Do we have the same defects for every string type? For example, with the following code we can compare types E1 and A5:

```
> defects <- data.frame(type = ss.data.strings$type, res = ss
    .data.strings$res < 3)
> defects <- aggregate(res ~ type, data = defects, sum)
> prop.test(defects$res, rep(20,6))
```

```
        6-sample test for equality of proportions
        without continuity correction

data:  defects$res out of rep(20, 6)
X-squared = 5.1724, df = 5, p-value = 0.3952
alternative hypothesis: two.sided
sample estimates:
prop 1 prop 2 prop 3 prop 4 prop 5 prop 6
  0.05   0.00   0.00   0.10   0.00   0.05
```

The *p*-value for the hypothesis test of equal proportions is 0.39 (larger than 0.05), so we cannot reject the null hypothesis, and therefore we do not reject that the proportions are equal.

A normality test to check if the data follow a normal distribution can be performed with the `shapiro.test()` function:

```
> shapiro.test(ss.data.strings$len)
```

```
        Shapiro--Wilk normality test

data:  ss.data.strings$len
W = 0.9846, p-value = 0.1902
```

The statistic used to perform this hypothesis test is the *Shapiro–Wilk* statistic. In this test, the hypotheses are as follows:

$$H_0 : \text{The data are normally distributed.}$$

$$H_1 : \text{The data are not normally distributed.}$$

The p-value is `0.19`. As it is larger than 0.05, we cannot reject the hypothesis of normality for the data, so we do not have enough evidence to reject normality. Other normality tests can be performed using the `nortest` package. □

Finally, regarding hypothesis testing, the concepts *error type I* and *error type II* should be introduced. A type I error occurs when we reject the null hypothesis and it is true. We commit a type II error when we do not reject the null hypothesis and it is false. The probability of the former is represented as α, and it is the significance level of the hypothesis test ($1-\alpha$ is the confidence level). The probability of the latter is represented as β, and the value $1-\beta$ is the statistical power of the test.

In statistics, committing a type I error is considered more severe than committing a type II error, and that is one reason why α is fixed before the test is performed. The statistical power of the test is usually used to find the appropriate sample size to conduct a hypothesis test. The R functions `power.prop.test` and `power.t.test` can be used to determine the power or the sample size.

Example 10.7 (Guitar strings (cont.)). Using the results of the initial study, the Black Belt plans to perform a new analysis to find out the sample size needed to estimate the mean length of the strings with a maximum error of $\delta = 0.1$ cm. He sets the significance level ($\alpha = 0.05$) and the power ($1-\beta = 0.90$). The sample size can be determined using the following command:

```
> power.t.test(delta = 0.1, power = 0.9, sig.level = 0.05,
        sd = sd (ss.data.strings$len))
```

```
     Two-sample t test power calculation

              n = 151.2648
          delta = 0.1
             sd = 0.2674321
      sig.level = 0.05
          power = 0.9
    alternative = two.sided

NOTE: n is number in *each* group
```

Therefore, for the new study he must select a sample of 152 strings. □

10.4 Regression

10.4.1 Model Identification

One of the most important inferences to be made in a Six Sigma project concerns the relationship between the critical to quality (CTQ) characteristics of our process (Ys) and the process parameters (Xs), that is, the variables that affect the process. This relationship is represented as a function where the value of the Ys can be inferred from knowledge of the Xs, plus some error (ε):

$$Y = f(X) + \varepsilon. \tag{10.1}$$

This function is a statistical model, with some parameters that must be estimated. Moreover, the error of the model must be assessed.

Regression is the statistical technique used to estimate the f function in (10.1). The easiest case is the simple linear regression, where f corresponds to a straight line. If we have only one independent variable (y) and one dependent variable (x), the regression model is

$$y = \beta_0 + \beta_1 x + \varepsilon,$$

and the parameters to be estimated are β_0 (the intercept of the straight line) and β_1 (the slope of the straight line). So the estimated model will be

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x.$$

The difference between the estimated values and the actual values in the sample, $\hat{y} - y$, are the *residuals*. The residuals are used to check and validate the model.

Before fitting the regression model, we need to find some evidence regarding the linear relation between the variables. The appropriate statistic to have a first approximation is the correlation coefficient, defined as

$$r = \frac{s_{xy}}{s_x s_y},$$

where $s_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n-1}$ is the covariance between the two samples and s_x and s_y are the sample standard deviations of x and y, respectively. This coefficient ranges from -1 to 1. There is no correlation when it equals 0, perfect positive correlation when it is 1, and perfect negative correlation when it is -1.

The graphical tool suitable for detecting at a glance the relationship between the two variables is the scatterplot (Chap. 8).

Example 10.8 (Guitar strings (cont.)). During the resistance test of the strings, a new measure was taken for those strings with a level of tension larger than 8. This measure is related to the sound volume produced by the string, which is considered a CTQ characteristic.

The Black Belt wonders whether the length of the strings is related to the sound volume. First, a new data frame is created to remove the missing data (those strings that broke down before achieving a level of tension equal to 8). Next, the correlation coefficient is calculated:

```
> data.sound <- na.omit(ss.data.strings)
> cor(data.sound$sound, data.sound$len)
```

```
[1] 0.7904747
```

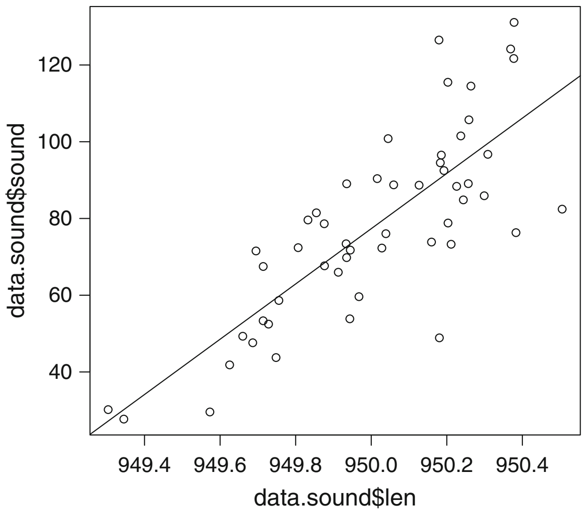


Fig. 10.2 Basic regression scatterplot. The regression line is plotted passing the adjusted model as an argument to the `abline` function

The value of the coefficient (0.79) shows that there is positive correlation between the two variables.

To confirm this relationship, we construct a scatterplot. Using the following code we can plot it using R standard graphics, including the regression line (Fig. 10.2):

```
> plot(data.sound$len, data.sound$sound)
> abline(lm(sound ~ len, data = ss.data.strings))
```

It is apparent that the longer a string is, the higher the sound it produces. We can plot a more sophisticated regression chart including a smooth line, confidence bands, and more information about the data (coloring the points according to string type) using the `ggplot2` package (Fig. 10.3) by typing the following code:

```
> qplot(len, sound, data = data.sound,
        geom = c("point", "smooth"),
        xlab = "String length",
        ylab = "String sound")  +
    geom_point(aes(col=type)) +
    opts(title = "Scatterplot for regression")
```

□

10.4.2 Model Fitting

Once we have identified the model, we estimate it, that is, we calculate $\hat{\beta}_0$ and $\hat{\beta}_1$. The function `lm` fits linear models (see Sect. 10.4.4 for additional models). The R functions that fit models accept certain specific arguments for the model to be fitted. There are two essential common arguments: `data` and `formula`. The former is the data frame where the data are stored, containing the variables we want to

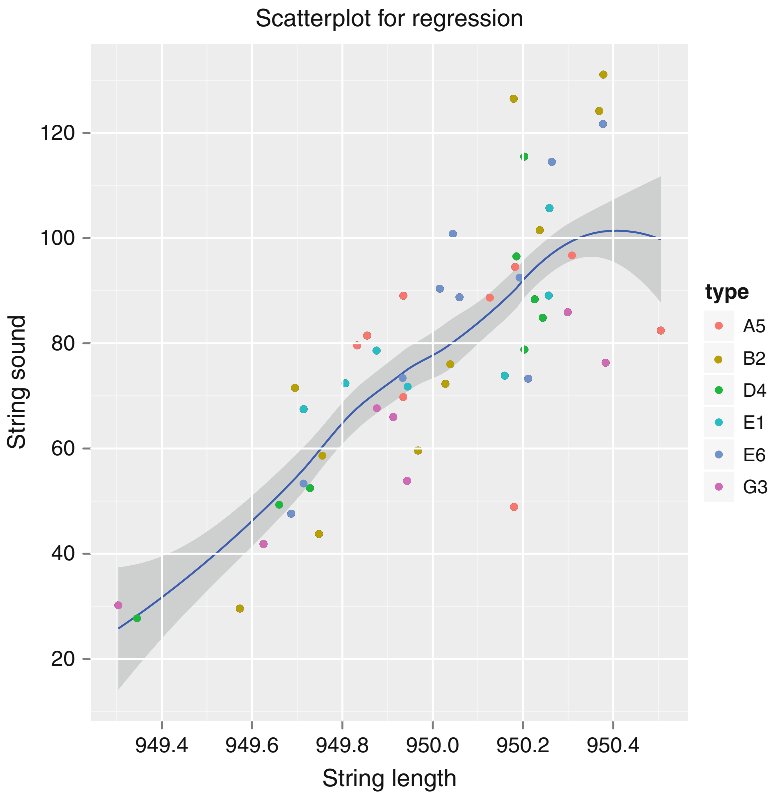


Fig. 10.3 Sophisticated scatterplot using `ggplot2` package. The line and bounds are computed using the `loess` function, which fits a polynomial regression. The smoothness can be adjusted by changing the `span` argument in the `qplot` function. It is also possible to include just the regression line. The points are colored according to string type to identify possible patterns

study. The latter is a special R expression to represent functions in symbolic form. It consists of two terms connected by the special *tilde* operator $\sim$. The term on the left is for the *response* or dependent variable. The term on the right is for the independent or explanatory variables. For example, for the simple linear regression model $y = \beta_0 + \beta_1 x$, the expression to pass as formula is `y ~ x` (assuming that we have two vectors named `x` and `y` in the R workspace).

The term on the right-hand side can consist of a single variable or a series of terms combined with some operators indicating the influence in the model (`+`, `-`, `:`, `*`, `^`, `%in%`). The following special symbols can be used to specify terms: `0` (avoid intercept), "`.`" (all but the response), and `1` (empty model). Type `?formula` in the R console to learn more about model formulae in R.

Example 10.9 (Guitar strings (cont.)). The linear model to estimate the relationship between the sound (response) and length of strings is

```
> lm(sound ~ len, data = data.sound)
```

```
Call:
lm(formula = sound ~ len, data = data.sound)

Coefficients:
(Intercept)          len
  -68346.72        72.03
```

By simply executing the `lm` function, we obtain the coefficients. These coefficients provide the formula for our model:

$$\text{sound} = -68346.72 + 72.03 \times \text{length}.$$

The usual way to proceed is to save the model in an object and access the elements in it. The elements stored in a `lm` model are listed below. The typical output for regression can be obtained with the `summary` generic function over a model object:

```
> my.model <- lm(sound ~ len, data = data.sound)
> names(my.model)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"
```

```
> summary(my.model)
```

```
Call:
lm(formula = sound ~ len, data = data.sound)

Residuals:
    Min      1Q  Median      3Q     Max
-41.438  -7.483  -1.345  10.369  36.252

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -68346.719   7497.948  -9.115 3.30e-12
len             72.025      7.892   9.126 3.18e-12

Residual standard error: 15.32 on 50 degrees of freedom
Multiple R-squared: 0.6249,     Adjusted R-squared: 0.6173
F-statistic: 83.28 on 1 and 50 DF,  p-value: 3.181e-12
```

In this output, we obtain, after the call of the function, some statistics about the residuals distribution. Next, the coefficients and the result of their hypothesis tests are shown. The null hypothesis in these tests is “the parameter equals 0.” Therefore, as the p-values are very small, we accept that the parameters are not zero. In the last part of the output, the R-squared statistic shows the proportion of the variability of the response that is explained by the linear model. The hypothesis test performed next is for testing the overall goodness of fit of the regression model. For simple linear models, the *p*-value is the same as that for the slope of the regression line.

The values returned for the parameters correspond to an estimation. We can compute a confidence interval for these estimators using the `confint` function over the model object:

```
> confint(my.model)
```

```
                    2.5 %       97.5 %
(Intercept) -83406.79017 -53286.64744
len             56.17277     87.87787
```

Other functions that can be run with a model as argument include `plot`, `residuals`, `predict`, and `anova`, among others. □

10.4.3 Model Validation

Once we have estimated the model, we need to validate the assumptions we made before fitting the model. For linear models, the main assumption is the normality and independence of the residuals. To verify the assumptions, we can use analytical tools such as a normality test for the residuals using the `shapiro.test` function or graphical tools such as those provided by the generic `plot` function over the model object.

Example 10.10 (Guitar strings (cont.)). Once we have fitted the model, we can perform a normality test over the residuals:

```
> shapiro.test(residuals(my.model))
```

```
        Shapiro-Wilk normality test

data:  residuals(my.model)
W = 0.9903, p-value = 0.9455
```

We cannot reject the null hypothesis (residuals are normal), so we accept normality for the residuals of the regression model.

The graphical output provided by the generic `plot` function is shown in Fig. 10.4. In the following code, the first line allows us to print in a single window the four graphics produced:

```
> par(mfrow=c(2,2))
> plot(my.model)
```

□

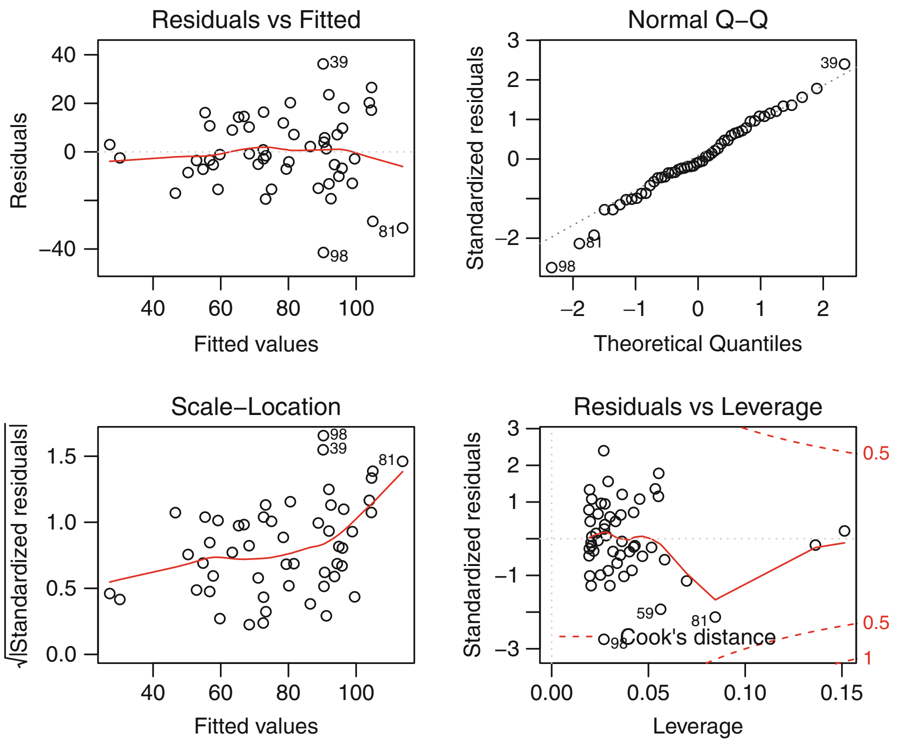


Fig. 10.4 Graphic output for linear regression model. The plot in the *top left* is a scatterplot of the residuals vs. the fitted values (the lack of patterns confirms the normality); the plot in the *top right* is a quantile–quantile plot for normal distributions (the points are approximately over a straight line, again confirming normality). The plot in the *bottom left* is a scatterplot for the $\sqrt{|\text{standardized residuals}|}$ vs. the fitted values. It is similar to the first one, and a triangular shape would be indicative of a lack of homoscedasticity (constant variance, which is one of the model assumptions). The last plot (*bottom right*) shows Cook's distance for all the points. Those points outside of the 0.5 line affect the parameter estimation and might be outliers

10.4.4 Other Models

So far, we have built step by step a simple linear regression model. In practice, many other models may describe our process. The first extension of the simple regression model is the multiple regression model, where we can include more than one independent variable, just adding up more terms on the right side of the formula. R can also fit models with different types of variables, including factors. When more than one variable is present, a variable selection technique should be considered. We can automatically select the significant variables with the function `step` over a model object. If we prefer to do it by ourselves, the function `update.formula` allows us to modify the model without rewriting the whole formula.

For multiple regression models, a new assumption is very important: the independent variables must be uncorrelated among each other. When this issue cannot be avoided, ridge regression is an alternative. The functions `lm.ridge` (`MASS` package), `ridge` (`survival` package), and `lpridge` (`lpridge` package) can fit regression models by ridge regression.

Other extension of the linear models are the *generalized linear model* (GLM), which allows us to deal with nonnormal data and adjust models of different *families* (binomial, Poisson, etc.). The R function `glm` is the appropriate one to fit these models. The *generalized additive model* (GAM) and *mixed models* are alternatives when we do not have linear relationships. The `gam` and `mgcv` packages contain functions for fitting these kinds of models.

State-of-the-art techniques to predict and estimate relationships between variables include neural networks (NN), partial least-squares (PLS) regression, and support vector machine (SVM). Packages for these techniques are available at CRAN: `AMORE`, `monmlp`, `nnet`, and `neuralnet` for NN; `pls`, `plsdof`, `plspm`, `plsRglm`, `plsgenomics`, `plsRbeta`, and `plsRcox` for PLS; and `e1071`, `kernlab`, and `RWeka` for SVM.

Example 10.11 (Guitar strings (cont.)). We can try to fit a multiple regression model by adding the variable `res` to our model to find out if tension also contributes to the sound. We can update the model with the following code:

```
> new.model <- update(my.model, . ~ . + res )
```

or just rewrite the formula:

```
> new.model <- lm(sound ~ len + res, data = data.sound)
```

Now we can see the model summary:

```
> summary(new.model)
```

```
Call:
lm(formula = sound ~ len + res, data = data.sound)

Residuals:
    Min      1Q  Median      3Q     Max
-41.194  -7.949  -1.641  10.702  36.496

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -68604.720   7604.325  -9.022 5.47e-12
len             72.285      8.001   9.035 5.24e-12
res              1.361      4.054   0.336    0.739

Residual standard error: 15.46 on 49 degrees of freedom
Multiple R-squared: 0.6257,     Adjusted R-squared: 0.6104
F-statistic: 40.96 on 2 and 49 DF,  p-value: 3.496e-11
```

The new estimate parameter for resistance (1.361) is not significantly different from zero (p-value = 0.739). Therefore, we do not need it in the model. We can automatically select the significant variables with the `step` function over the full model using the `backward` method[7]:

```
> step(new.model, direction = "backward")
```

```
Start:  AIC=287.67
sound ~ len + res

       Df Sum of Sq   RSS    AIC
- res   1      26.9 11735 285.79
<none>              11708 287.67
- len   1   19502.4 31210 336.66

Step:  AIC=285.79
sound ~ len

       Df Sum of Sq   RSS    AIC
<none>              11735 285.79
- len   1     19545 31280 334.77

Call:
lm(formula = sound ~ len, data = data.sound)

Coefficients:
(Intercept)          len
  -68346.72        72.03
```

As expected, the model obtained contains only the explanatory variable `len`. □

10.5 Analysis of Variance

Analysis of variance (ANOVA) is the appropriate statistical technique to analyze the relationship between variables when the explanatory variables are qualitative (also called factors). The possible values of the factors are called *levels*. When we have only one factor with two levels, we can perform a t-test to test the difference between means, as explained in Sect. 10.3. Otherwise, we use one-way ANOVA. One-way means the principal effects to be evaluated, that is, the effect of belonging to one of the groups determined by the levels. Two-way ANOVA is performed when we have more than one factor and we are interested in measuring the effect of second-order interactions (that is, if the response in one level of a factor depends on the level of another factor). Similarly, we can perform multiway ANOVA. However, it is very unlikely to improve our analysis with more than third-order interactions.

[7]We can choose one of the following methods in the `step` function: backward, forward, or both.

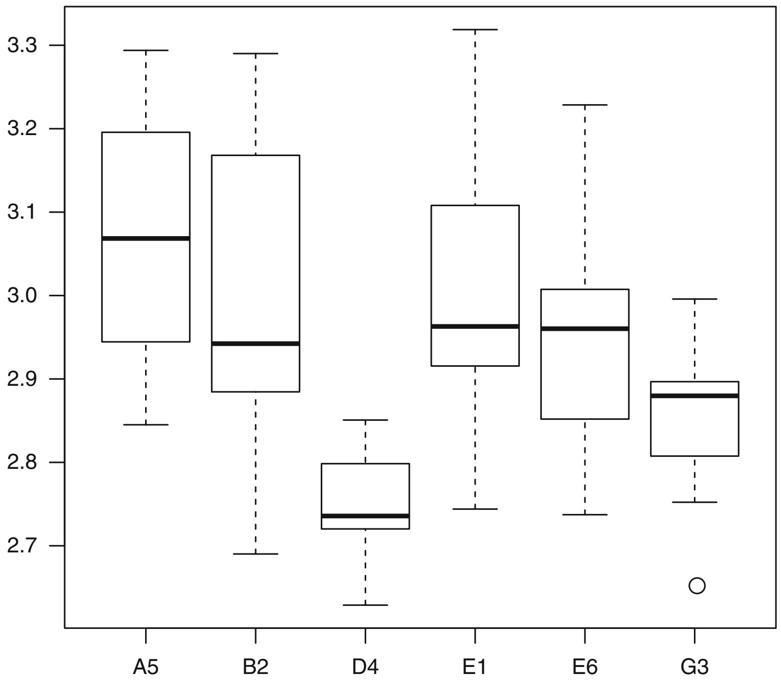


Fig. 10.5 Group box plots for guitar string example. The power needed for strings D4 and G3 seems to be different from the rest. We need to verify if this difference is significant

10.5.1 Model Identification

The counterpart of the scatterplot to reveal a possible relationship between the response and the factors is the box plot by groups. It consists in plotting in the same graphic box plots for each group defined by the factor levels.

Example 10.12 (Guitar strings (cont.)). A measurement of the power needed to pluck a string at a tension level of 8 was taken during the tension test. The Black Belt wants to know if there are differences between the various types of strings for this important characteristic. First, we should plot the box plots for the groups (Fig. 10.5):

```
> boxplot(power ~ type, data = data.sound)
```

We can see that strings D4 and G3 look different from the rest. □

10.5.2 Model Fitting and Validation

We can use the functions `aov` and `lm` to perform the ANOVA. When using `lm`, the ANOVA table is printed calling the function `anova` over the object. The effects of each factor level are the parameters to estimate in the model:

$$y_{ij} = \mu + \alpha_i + \varepsilon_{ij},$$

where α_i is the effect of level i. In practice, μ is replaced by the sample mean of the first level (reference level), and the effects are related to the reference level.

Example 10.13 (Guitar strings (cont.)). We fit the model with the `lm` function[8] and save the result in an object:

```
> model.power <- lm(power ~ type, data = data.sound)
> names(model.power)
```

```
 [1] "coefficients"  "residuals"     "effects"
 [4] "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "contrasts"
[10] "xlevels"       "call"          "terms"
[13] "model"
```

This object contains 13 components. By calling the generic `anova` function over the model, we get the following ANOVA table:

```
> anova(model.power)
```

```
Analysis of Variance Table

Response: power
          Df  Sum Sq  Mean Sq F value   Pr(>F)
type       5 0.57869 0.115739  4.8131 0.001273
Residuals 46 1.10615 0.024047
```

The ANOVA table shows a hypothesis test where the null hypothesis is "all the effects are equal to 0." As the p-value is lower than 0.05, we can reject the null hypothesis and accept that there are differences among the means of the groups.

The parameters are stored in the component `coefficients` of the model object, and they can be printed using the `summary` function. The number of parameters is equal to the number of levels in the factor. The intercept parameter is the mean of the response for the first level, and the remaining parameters represent the effect of this level on the mean of the first level. We can also obtain a confidence interval for each parameter:

```
> summary(model.power)
```

[8]The `aov` function can also be used. The difference is basically the presentation of the results.

```
Call:
lm(formula = power ~ type, data = data.sound)

Residuals:
      Min        1Q    Median        3Q       Max
-0.304940 -0.104136 -0.002825  0.076641  0.308359

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.07123    0.05169  59.416  < 2e-16
typeB2      -0.07609    0.06970  -1.092  0.28065
typeD4      -0.32267    0.07535  -4.282 9.32e-05
typeE1      -0.06081    0.07815  -0.778  0.44048
typeE6      -0.10966    0.07125  -1.539  0.13065
typeG3      -0.22355    0.07815  -2.861  0.00634

Residual standard error: 0.1551 on 46 degrees of freedom
Multiple R-squared: 0.3435,     Adjusted R-squared: 0.2721
F-statistic: 4.813 on 5 and 46 DF,  p-value: 0.001273
```

```
> confint(model.power)
```

```
                 2.5 %      97.5 %
(Intercept)  2.9671822  3.17527524
typeB2      -0.2163871  0.06420548
typeD4      -0.4743378 -0.17099267
typeE1      -0.2181127  0.09649442
typeE6      -0.2530739  0.03376260
typeG3      -0.3808542 -0.06624708
```

The intervals corresponding to types D4 and G3 do not contain a zero value and therefore differ from the intercept. However, types B2, E1, and D6 do not differ with respect to A5 (intercept).

We can make pairwise comparisons between all the groups using the function `pairwise.t.test`:

```
> pairwise.t.test(power, type,
    p.adj = "bonferroni",
    data = ss.data.strings)
```

```
        Pairwise comparisons using t tests with pooled SD

data:  power and type

   A5      B2      D4      E1      E6
B2 1.00000 -       -       -       -
D4 2.9e-05 0.00018 -       -       -
E1 1.00000 1.00000 0.00023 -       -
E6 1.00000 1.00000 0.00304 1.00000 -
G3 0.00409 0.01852 1.00000 0.01882 0.16540

P value adjustment method: bonferroni
```

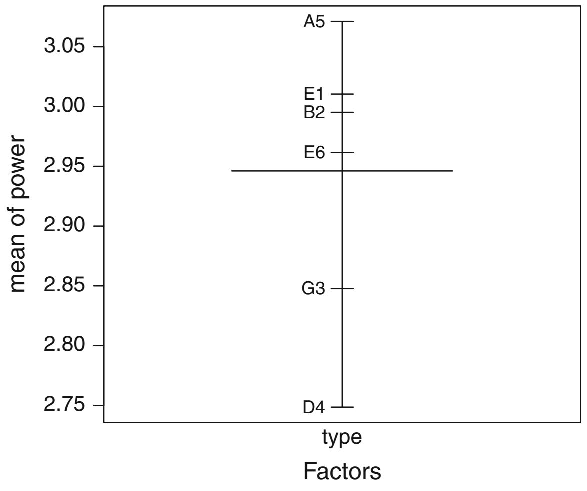


Fig. 10.6 Plot of effects in ANOVA for guitar string example. The mean of the power is lower for types G3 and D4

The output is a matrix with p-values corresponding to the individual hypothesis tests comparing the means. We can see that G3 and D4 are different from the rest of the string types, but there is no difference between them.

To visualize the effects, we can plot a simple chart using the `plot.design` function (Fig. 10.6) or a dot plot with the sample means linked by lines using the `ggplot2` package (Fig. 10.7).

```
> plot.design(power ~ type, data = data.sound)
```

```
> qplot(type, power, data = ss.data.strings) +
    stat_summary(fun.y = mean, geom = "line",
        aes(group = 1), col = "orangered") +
    stat_summary(fun.y = mean, geom = "point",
        shape = 17, size = 3, col = "red") +
    opts(title = "Effects of factor Type of string")
```

□

10.5.3 Additional Models and Related Tools

ANOVA is the suitable analytical tool to use after an experiment has been done. The correct design of that experiment is crucial for the results to be acceptable. Design of experiments (DoE) will be explained in some detail in Chap. 11.

When we are jointly analyzing factors and continuous variables as explanatory variables, analysis of covariance (ANCOVA) allows us to study interactions between both types of variables.

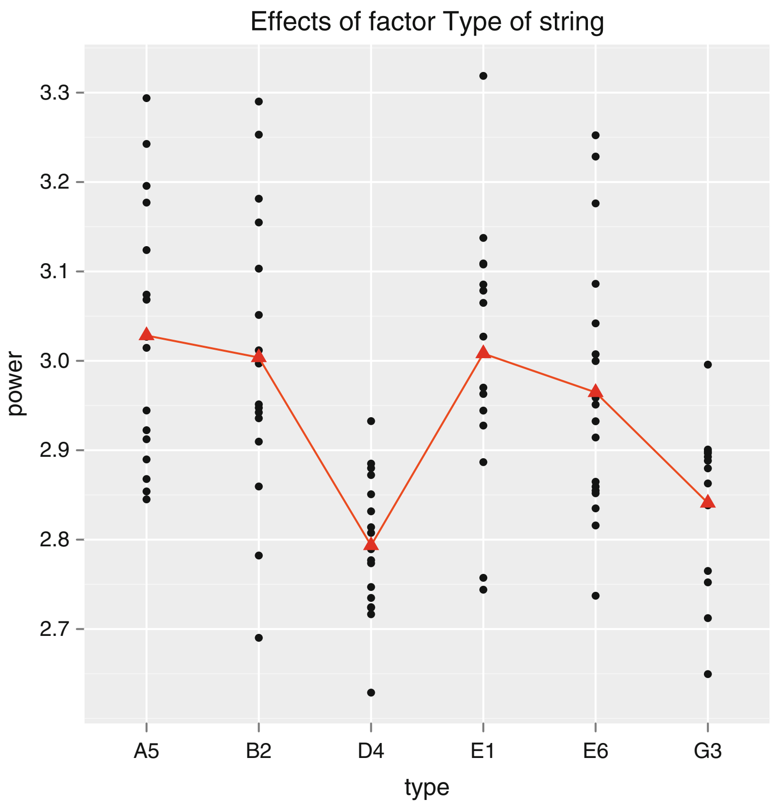


Fig. 10.7 Effects of string type. The means look quite different for types D4 and G3, with D4 having the lower mean value

Multivariate analysis of variance (MANOVA) is an extension that is used when more than one response variable may be influenced by some dependent variables. The R function `manova` accepts the same arguments as the `aov` function, expecting a matrix of continuous variables as the left-side component of the formula.

10.6 Summary and Further Reading

In this chapter we reviewed the basics of statistical inference. Point and interval estimation are the techniques used to make inferences about the parameters and probability distribution of a population. Hypothesis testing can be used throughout any inference analysis, with the interpretation results relying on the important concepts of p-value and null/alternative hypotheses.

Inference about the relationship of variables starts with linear models, such as regression or ANOVA. The main R functions and their outputs were explained. However, one of the most common statistical mistakes made in inference is to infer cause-and-effect correspondence from that relationship. It is not enough by itself, as there may be many other unstudied variables that are the real cause of some effect. DoE, described in the next chapter, will help to confirm those cause-and-effect relationships.

Extensions of the linear models can be read in [24]. The free resources [45] and [25] contain a number of R examples explaining the inference techniques reviewed in this chapter. The book [38], freely available on the book's Web site, is a nice reference on state-of-the-art prediction and estimation techniques. Regarding SVM, a complete review with applications is the work in [68]. References [18] and [20] discuss the application of statistics with R. Moreover, almost all books on Six Sigma contain entire chapters devoted to statistical inference.

Case Study

Estimate the parameters of the probability distribution of the flight time (mean and variance) and calculate a confidence interval. Test if the flight time follows a normal distribution. Obtain a confidence interval for the proportion of defects using any valid criterion for what constitutes a defect. Fit a regression model using as response variable the flight time and as independent variable any other measurable characteristic of the prototypes or the environment (e.g., temperature, wind speed). Determine if the estimated parameters are significant. Make an ANOVA using the flight time as the response and any categorical variable as the independent variable (e.g., a design characteristic, operator). Fit a model and determine if there are differences between the groups.

Practice

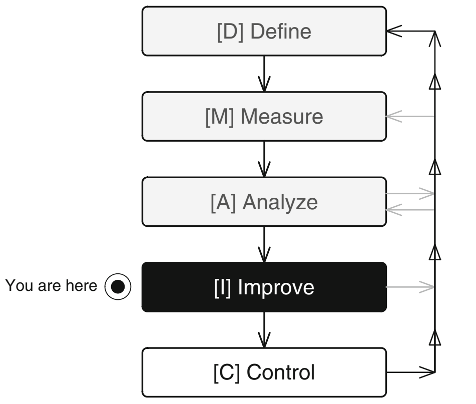
10.1. Test the normality hypothesis for the resistance of the strings.

10.2. Fit a linear model using the resistance of the strings as the response variable and the length of the strings as the independent variable. Is the linear model a good model to explain this relationship?

10.3. Determine whether the length of the strings is related to the type of string using ANOVA.

Part V
R Tools for the Improve Phase

Roadmap of the DMAIC Cycle



In this part of the book, tools useful during the Improve phase are introduced.

In this phase, the relationship between the input variables and the response is verified and the new process operating conditions are statistically validated.

We will describe the most representative tool: design of experiments (DoE).

Chapter 11
Design of Experiments with R

Sometimes the only thing you can do with a poorly designed experiment is to try to find out what it died of.
R. A. Fisher

11.1 Introduction

Design of experiments (DoE) is one of the most important tools in the Six Sigma methodology. According to [2] it is the "jewel of quality engineering." It is the essence of the Improve phase and the basis for the design of robust processes. However, DoE is not improvement itself. In fact, some authors include DoE within the Analyze phase of the design, measure, analyze, improve, and control (DMAIC) cycle; see, for instance, [82]. An adequate use of DoE will lead to the improvement of a process, but a bad design can result in wrong conclusions and engender the opposite effect: inefficiencies, higher costs, and less competitiveness.

In this chapter, we present the foundations of DoE and introduce the essential functions in R to perform it and analyze its results. We will describe 2^k factorial designs using a variation of the example described in [10]. This example is representative of how DoE should be used to achieve the improvement of a process in a Six Sigma way. The chapter is not intended as a thorough review of DoE. The idea is to introduce a simple model in an intuitive way. For more technical or advanced training, a number of references are given at the end of the chapter.

11.2 Importance of Experimenting

With the analytical tools presented in previous chapters (and in the following one), we can perform *observational studies*. In these kinds of studies we act as an external observer. In other words, we do not have any influence on the variables we are measuring. We just collect the data and analyze them using the appropriate statistical technique (e.g., regression, analysis of variance (ANOVA)).

DoE allows us to control the values of the Xs of the process and then measure the value of the Ys to discover what values of the independent variables will allow us to improve the performance of our process (the dependent variable or response). Moreover, we can manage the effects of other variables beyond our control using blocking techniques.

Experiments seem to be scientists' stuff. So do we need experiments for industrial and service activities? The answer is given by the following quote: *"An engineer who does not know experimental design is not an engineer."* This comment was made by an executive of the Toyota Motor Company to one of the authors of the book by [11], where it is included along with other quotations. In this regard, we have emphasized several times that Six Sigma consists in the application of the scientific method to process improvement, and experiments are an essential part of the scientific methodology of work.

Nevertheless, when we rely on the analysis of data gathered directly during the normal performance of a process, there are some risks pertaining to the conclusions. Next, we briefly list these risks, which should be taken into account.

11.2.1 Inconsistent Data

Noise factors such as aging, procedure modifications, repairs, and other circumstances that vary as time changes are generally not measured along with the variables of interest of a process. Thus, data automatically recorded may not be consistent, and conclusions based on their analysis may not be correct.

11.2.2 Variable Value Range

When measuring the process during its day-to-day performance, the variables are supposed to be under control, that is, within the control limits of the process. This fact may hide the real relationship between the variables because the independent variable (X) is inside the operation range and its performance outside this range is unknown. Figure 11.1 illustrates an example of this situation.

11.2.3 Correlated Variables

When two or more explanatory variables are correlated (Sect. 10.4, Chap. 10), the data gathered during the normal operation of a process (without experimentation) may lead to wrong conclusions. Thus we can obtain two different, misleading situations. The former is known as the *confusion of effects* and is produced when responses vary in the same sense of two correlated factors. Thus we do not know

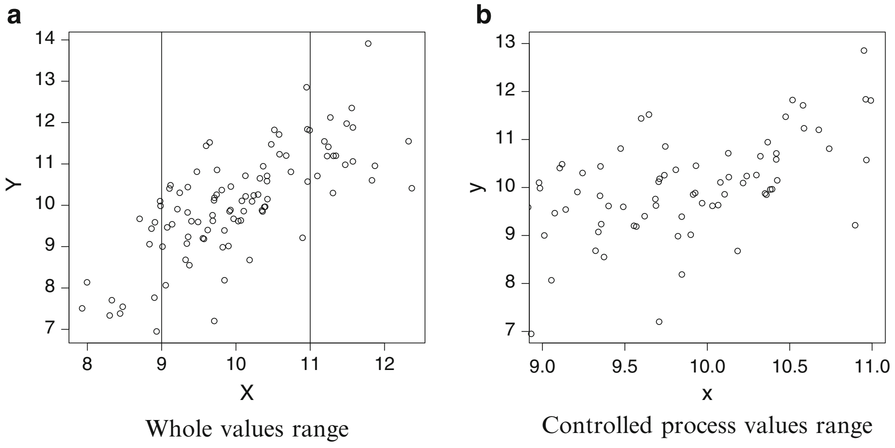


Fig. 11.1 Value range risk in observational studies. These are scatterplots of two related variables. The relationship is clear in plot (**a**), where the whole data set is represented. During the normal performance of the process, the values measured are those within the limits, but when we plot them (plot (**b**)), the relationship is not apparent

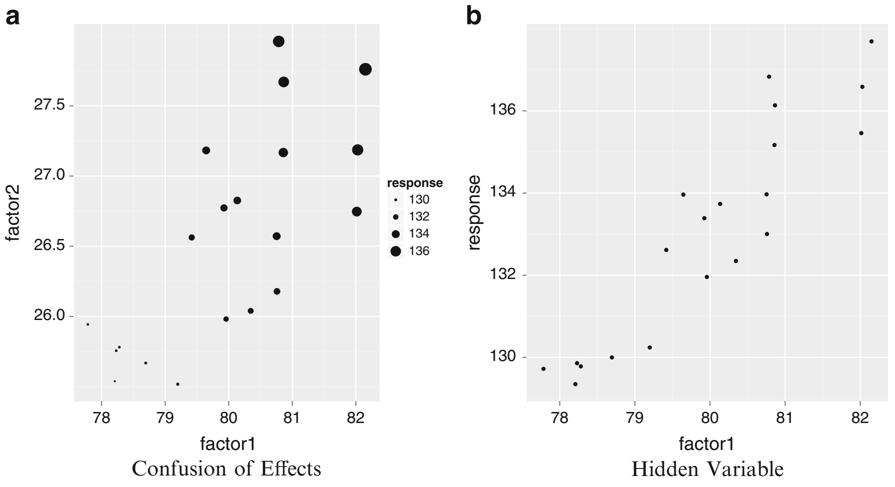


Fig. 11.2 Risks of correlated variables without experimentation. On the left-hand plot we have three variables: the two explanatory variables are represented, respectively, on the x-axis and on the y-axis. The response is represented as the area of the circle at each crossing factor1/factor2. Notice how the response increases as either of the two factors grows. Therefore, we do not know which one causes the variation in the response. On the right-hand plot, the x-axis is for one of the explanatory variables (factor1), and the y-axis is for the response. There seems to be a relationship, but perhaps the cause-and-effect relation comes from hidden variable factor2. Both charts are plotted using the same data set

which of the factors causes the variation in the response. The latter is called the *hidden variable* and is produced when one of the correlated factors has not been measured, and we may infer that the variation is caused by the measured one (when there is no cause-and-effect relation). Figure 11.2 illustrates an example of these situations.

11.3 Experimentation Strategies

We can deal with the risks explained in the previous section through experimentation. Cause-and-effect relationships can be determined by changing the levels of the factors and measuring the *effects*. Before explaining the computation of the effects using statistical techniques, let us have a look at the different strategies we can follow in the stages prior to experimentation.

11.3.1 Planning Strategies

Experimentation is usually expensive, and we have limited resources to carry out our experiments. So we must decide how to use them. The more trials we do, the more money we spend on the experiment. In Sect. 10.5, we described what we mean by factors and levels. In what follows, roughly speaking, we will refer to the explanatory variables as *factors* and the discretized values of these variables as *levels*. In fact, the number of trials increases exponentially as we add more factors or levels.

The decision on how to allocate our budget is made in the planning of the experiment. We can decide between several planning strategies:

No planning. Everybody agrees that this is a bad way of conducting an experiment. However, it happens often enough. This is the trial-and-error approach: we conducts one experiment after another until the budget is spent, following some criteria, often based on intuition or conjectures rather than on science or data analysis.

Plan everything at the beginning. Once the experiment has been defined (essentially, the factors whose effects in the response should be assessed), the entire budget is allocated to performing all the possible experiments, setting different levels for each factor within the range the analyst has decided on (for example, through observational studies, bibliography, or some other method). The down side of this planning strategy is that it does not take into account intermediate results, and we would probably spend money on experiments that contributed nothing to our knowledge of the process.

Sequential planning. The Six Sigma way of experimenting is to plan the experiments sequentially. In a first stage, a reduced number of trials will be conducted to make decisions about the next stage. This first stage should consume part of the budget (between 25 and 40%, depending on the researcher). Most of the budget should be spent in subsequent stages, taking into account previous results.

11.3.2 Factor Levels and Replications

In the sequential planning strategy described above, the factor levels can be modified in the direction of the experiment objective. For example, imagine a continuous explanatory variable (factor) that has been discretized into two levels (low, usually represented by the symbol "−," and high, represented by the symbol "+"). If we are interested in the increase in the response variable, and a first experiment shows that the effect of a given factor raises the response to a high level (+), in the next experiment we should try a higher level of the factor to find out if the response improves its performance even more and not spend money in the other direction, that is, the low level (−). Note that factors and levels are usually represented by a *design matrix* with one column for each factor, one row for each different experimental condition, and a symbol in the cells representing the factor level.

Example 11.1 (Pizza dough). Let us define now the experiment that will be used throughout the chapter. It is a variation of the example described in [10].

A food manufacturer is searching for the best recipe for its main product: a pizza dough sold in retailers across its market area. To discover this *magic formula*, the managers decided to perform an experiment to determine the optimal levels of the three main ingredients in the pizza dough: flour, salt, and baking powder. The other ingredients are fixed as they do not affect the flavor of the final cooked pizza. The flavor of the product will be determined by a panel of experts who will give a score to each recipe. Therefore, we have three factors that we will call `flour`, `salt`, and `bakPow`, with two levels each (− and +). We can then construct a data sheet to record the measured data using the following code:

```
> pizzaDesign <- expand.grid(flour = gl(2, 1, labels = c("-",
     "+")),
        salt = gl(2, 1, labels = c("-", "+")),
        bakPow = gl(2, 1, labels = c("-", "+")),
        score = NA)
> pizzaDesign
```

```
  flour salt bakPow score
1     -    -      -    NA
2     +    -      -    NA
3     -    +      -    NA
4     +    +      -    NA
5     -    -      +    NA
6     +    -      +    NA
7     -    +      +    NA
8     +    +      +    NA
```

Note that we have eight different experiments (recipes) including all the possible combinations of the three factors at two levels. □

When we have more than one factor, the sequential planning can be carried out changing one factor at a time, with the others kept fixed. This is the wrong approach as there may be interactions among the factors, that is, the combination of levels of different factors may affect the response. For instance, when there is an interaction between two factors, the effect of one of them on the response depends on the value of the other factor. Therefore, to discover the main effects and the interactions, we should vary more than one level at a time, performing experiments in all the possible combinations.

However, as we remarked above, experimentation is expensive, and the number of factor levels to test increases the number of experiments. This is the reason why two-level factor experiments (denoted by 2^k) are the most widely used.

Finally, to study the variation under the same experimental conditions, we will need to replicate the experiment, making more than one trial per factor combination. The number of replications depends on several aspects (e.g., budget).

11.3.3 Progressive Experimentation

In practice, the way to proceed with Six Sigma experimentation is a progressive (and sometimes iterative) method. First, performing *screening* experiments, we study a set of factors to discard those that do not affect the response and then select the important ones. A priori, we may think that some of them do not cause variation in the response, but the benefit of including them in the experiment is that we will have scientific evidence. Moreover, underlying interactions will be properly tested. Once the significant effects have been identified, *characterizing* experiments are conducted, estimating the effects and studying the residuals to model the $Y = f(X)$ function that governs the process. Finally, *optimization* experiments will help to find the operational optimum value for our process.

11.3.4 Model Assumptions

Remember that we are at the Improve phase of the DMAIC cycle, and we have arrived here after having defined the process and the project objectives. Moreover, we have achieved an appropriate Measurement system (Chap. 5), and we know the process capability (Chap. 7) and the cost of poor quality (Chap. 4).

To validate the results, we must reiterate the importance of randomization. Once you have fixed the factors and the levels of an experiment, measurements must be taken randomly. Thus, if we have an ordered list of all the combinations of factor levels, before starting measuring the response, we must set the random order in which they are going to be measured.

The suitable statistical technique to analyze experiments is analysis of variance (ANOVA), described in Chap. 10. There are two main assumptions in this type of model: (1) the normality and independence of residuals and (2) homoscedasticity (constant variance). A systematic strategy for randomization is useful for achieving the former, whereas the latter may be checked using hypothesis tests (Chap. 10).

Example 11.2 (Pizza dough (cont.)). Once an experiment has been designed, we will proceed with its randomization. We can add a column to the design matrix with the randomized order of the trials and sort the rows of the design matrix following a random order:

```
> pizzaDesign$ord <- sample(1:8, 8)
> pizzaDesign[order(pizzaDesign$ord),]
```

```
  flour salt bakPow score ord
5     -    -      +    NA   1
1     -    -      -    NA   2
7     -    +      +    NA   3
2     +    -      -    NA   4
4     +    +      -    NA   5
6     +    -      +    NA   6
8     +    +      +    NA   7
3     -    +      -    NA   8
```

Each time you repeat the command you get a different order due to randomization.

Finally, we can print the data frame in the appropriate media for the team member in charge of registering the value of the response (in this case, the scores given by the experts to each recipe). We will see how to generate automatic reports with R in Chap. 13.

Pizzas may be labeled with the number of the corresponding recipe, and then the scores are recorded in the form. We can have as many forms as experiment replications. □

11.4 2^k Factorial Designs

2^k factorial designs are those whose number of factors to be studied are k, all of them with 2 levels. The number of experiments we will have to carry out to obtain a complete replication is precisely the power 2^k (2 to the k). If we want n replications of the experiment, then the total number of experiments is $n \times 2^k$.

Using ANOVA, we will estimate the effect of each factor and interaction and assess which of these effects are significant. Let us represent the factors by uppercase latin letters (A, B, ...). The main effects are usually represented by greek letters (α, β, ...) corresponding to the latin letter of the factor, and the effects of interactions by the combination of the letters representing the factors whose effects interact. For example, for a 2^k experiment with three factors and n replications, the statistical model is

$$y_{ijkl} = \mu + \alpha_i + \beta_j + \gamma_k + (\alpha\beta)_{ij} + (\alpha\gamma)_{ik} + (\beta\gamma)_{kl} + (\alpha\beta\gamma)_{ijk} + \varepsilon_{ijkl}, \quad (11.1)$$
$$i = 1,2; \quad j = 1,2; \quad k = 1,2; \quad l = 1\dots n,$$
$$\varepsilon_{ijkl} \sim N(0,\sigma) \quad \text{independent},$$

where

μ is the global mean of the response,
α_i is the effect of factor A at level i;
β_j is the effect of factor B at level j;
γ_k is the effect of factor C at level k;
$(\alpha\beta)_{ij}$ is the effect of the interaction of factors A and B at levels i and j, respectively;
$(\alpha\gamma)_{ik}$ is the effect of the interaction of factors A and C at levels i and k, respectively;
$(\beta\gamma)_{jk}$ is the effect of the interaction of factors B and C at levels j and k, respectively;
$(\alpha\beta\gamma)_{ijk}$ is the effect of the interaction of factors A, B and C at levels i, j, and k, respectively;
ε_{ijkl} is the random error component of the model.

Example 11.3 (Pizza dough (cont.)). The experiment is carried out by preparing the pizzas at the factory following the package instructions, which had been tested previously as the better conditions to prepare the pizza, namely: "bake the pizza for 9 min in an oven at 180°C."

After a *blind trial*,[1] the scores given by the experts to each of the eight (2^3) recipes in each replication of the experiment are those in Table 11.1.

To perform the ANOVA in R, we keep the data in a data frame where each row corresponds to one of the individual experiments. You can do this by combining the design matrix created previously and assigning the response values using a vector or any other input data method available in R (Chap. 2). Using the following code you obtain an R `data.frame` object with the data in Table 11.2. The `ss.data.doe1` data set is available in the `SixSigma` package.

```
> ss.data.doe1 <- data.frame(repl = rep(1:2, each = 8),
        rbind(pizzaDesign[, -6], pizzaDesign[, -6]))
> ss.data.doe1$score <- c(5.33, 6.99, 4.23, 6.61,
        2.26, 5.75, 3.26, 6.24,
        5.7, 7.71, 5.13, 6.76,
        2.79, 4.57, 2.48, 6.18)
```

[1]The experts do not know the recipe used for each individual pizza.

Table 11.1 Pizza dough recipe scores

	Replication 1	Replication 2
Recipe 1	5.33	5.70
Recipe 2	6.99	7.71
Recipe 3	4.23	5.13
Recipe 4	6.61	6.76
Recipe 5	2.26	2.79
Recipe 6	5.75	4.57
Recipe 7	3.26	2.48
Recipe 8	6.24	6.18

Table 11.2 Data for pizza dough example

	Repl	Flour	Salt	BakPow	Score	Ord
1	1	−	−	−	5.33	2
2	1	+	−	−	6.99	4
3	1	−	+	−	4.23	8
4	1	+	+	−	6.61	5
5	1	−	−	+	2.26	1
6	1	+	−	+	5.75	6
7	1	−	+	+	3.26	3
8	1	+	+	+	6.24	7
9	2	−	−	−	5.70	2
10	2	+	−	−	7.71	4
11	2	−	+	−	5.13	8
12	2	+	+	−	6.76	5
13	2	−	−	+	2.79	1
14	2	+	−	+	4.57	6
15	2	−	+	+	2.48	3
16	2	+	+	+	6.18	7

We can get the average for each recipe:

```
> aggregate(score ~ flour + salt + bakPow,
    FUN = mean, data = ss.data.doe1)
```

```
  flour salt bakPow score
1     -    -      - 5.515
2     +    -      - 7.350
3     -    +      - 4.680
4     +    +      - 6.685
5     -    -      + 2.525
6     +    -      + 5.160
7     -    +      + 2.870
8     +    +      + 6.210
```

The best recipe seems to be the one with a high level of flour and a low level of salt and baking powder. We must fit a linear model and perform an ANOVA to find the significant effects. We use the `lm` function to fix the model:

```
> doe.model1 <- lm(score ~ flour + salt + bakPow +
                 flour * salt + flour * bakPow +
                 salt * bakPow + flour * salt * bakPow,
        data = ss.data.doe1)
```

Once we have saved the model into a `model` object, we obtain an ANOVA table using the `summary` function:

```
> summary(doe.model1)
```

```
Call:
lm(formula = score ~ flour + salt + bakPow + flour * salt +
    flour *
    bakPow + salt * bakPow + flour * salt * bakPow, data = ss
        .data.doe1)

Residuals:
    Min      1Q  Median      3Q     Max
-0.5900 -0.2888  0.0000  0.2888  0.5900

Coefficients:
                     Estimate Std. Error t value
(Intercept)            5.5150     0.3434  16.061
flour+                 1.8350     0.4856   3.779
salt+                 -0.8350     0.4856  -1.719
bakPow+               -2.9900     0.4856  -6.157
flour+:salt+           0.1700     0.6868   0.248
flour+:bakPow+         0.8000     0.6868   1.165
salt+:bakPow+          1.1800     0.6868   1.718
flour+:salt+:bakPow+   0.5350     0.9712   0.551
                     Pr(>|t|)
(Intercept)          2.27e-07
flour+               0.005398
salt+                0.123843
bakPow+              0.000272
flour+:salt+         0.810725
flour+:bakPow+       0.277620
salt+:bakPow+        0.124081
flour+:salt+:bakPow+ 0.596779

Residual standard error: 0.4856 on 8 degrees of freedom
Multiple R-squared: 0.9565,     Adjusted R-squared: 0.9185
F-statistic: 25.15 on 7 and 8 DF,  p-value: 7.666e-05
```

Looking at the *p*-values, we can figure out, with a high level of confidence, that the main effects of the ingredients flour and baking powder are significant, while the effect of the salt is not significant. Interactions among the ingredients are neither 2-way nor 3-way, making them insignificant.

Now, we can simplify the model, excluding the nonsignificant effects. Thus, the new model with the significant effects is

```
> doe.model2 <- lm(score ~ flour + bakPow,
    data = ss.data.doe1)
> summary(doe.model2)
```

```
Call:
lm(formula = score ~ flour + bakPow, data = ss.data.doe1)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.84812 -0.54344  0.06062  0.44406  0.86938 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.8306     0.2787  17.330 2.30e-10
flour+        2.4538     0.3219   7.624 3.78e-06
bakPow+      -1.8662     0.3219  -5.798 6.19e-05

Residual standard error: 0.6437 on 13 degrees of freedom
Multiple R-squared: 0.8759,     Adjusted R-squared: 0.8568 
F-statistic: 45.87 on 2 and 13 DF,  p-value: 1.288e-06
```

And we can obtain the coefficients for the model with the `coef` function:

```
> coef(doe.model2)
```

```
(Intercept)      flour+     bakPow+ 
   4.830625    2.453750   -1.866250 
```

Therefore, the statistical model for our experiment is

$$\widehat{score} = 4.8306 + 2.4538 \times flour - 1.8662 \times bakPow, \tag{11.2}$$

expressed in terms of the coefficients of the model. In the preceding equation, the possible values for the *flour* and *bakPow* factors are 0 for the low level ($-$), and 1 for the high level ($+$).

This codification of the factors is different from that in (11.1), in which the possible values for the factors are -1 for the low level and $+1$ for the high level. To calculate the effects, the coefficients must be divided by two, and the equivalent model to (11.1) will be

$$\widehat{score} = 5.1244 + 1.2269 \times flour - 0.9331 \times bakPow, \tag{11.3}$$

where 5.1244 is the overall average score. For instance, in this formulation of the model, passing flour from level -1 to level $+1$ implies an increase of 2.4538 in the average score of the recipe.

If we want to change from the model given by (11.3) to the model given by (11.2), then all we have to do is calculate the coefficients by duplicating the effects and calculating the intercept as

$$\bar{y} - \alpha_1 - \beta_1 = 5.124375 - 1.226875 - (-0.933125) = 4.8306. \tag{11.4}$$

```
> as.numeric(mean(ss.data.doe1$score) -
    coef(doe.model2)[2]/2 -
    coef(doe.model2)[3]/2)
```

```
[1] 4.830625
```

Thus, the recipe with a high level of flour and low level of baking powder will be the best one, regardless of the level of salt (high or low). The estimated score for this recipe is $4.8306 + 2.4538 \times 1 + (-1.8662) \times 0 = 7.284$:

```
> coef(doe.model2)[1] + coef(doe.model2)[2]
```

```
(Intercept)
   7.284375
```

We obtain the same result using the effect formula:

$$\hat{y} = 5.124375 + 1.2269 \times 1 + (-0.9331) \times (-1) = 7.2844.$$

```
> mean(ss.data.doe1$score) +
 coef(doe.model2)[2]/2 *(1) +
 coef(doe.model2)[3]/2 *(-1)
```

```
  flour+
7.284375
```

We can obtain the estimations for all the experimental conditions (including the replications) using the `predict` function:

```
> predict(doe.model2)
```

```
       1        2        3        4        5        6
4.830625 7.284375 4.830625 7.284375 2.964375 5.418125
       7        8        9       10       11       12
2.964375 5.418125 4.830625 7.284375 4.830625 7.284375
      13       14       15       16
2.964375 5.418125 2.964375 5.418125
```

The estimation for experimental conditions 2, 4, 10, and 12 are the same (7.2844) as the salt factor has been removed from the model.

We can also compute a confidence interval for each parameter:

```
> confint(doe.model2)
```

```
                2.5 %    97.5 %
(Intercept)  4.228435  5.432815
flour+       1.758401  3.149099
bakPow+     -2.561599 -1.170901
```

To graphically visualize the main effects, the `plot` function can be used. In this case, points and lines showing the effects will be plotted. For example, to represent the effects for the factor *flour* (Fig. 11.3) we type

```
> plot(c(-1, 1), ylim = range(ss.data.doe1$score),
       coef(doe.model1)[1] + c(-1, 1) * coef(doe.model1)[2],
       type="b", pch=16)
> abline(h=coef(doe.model1)[1])
```

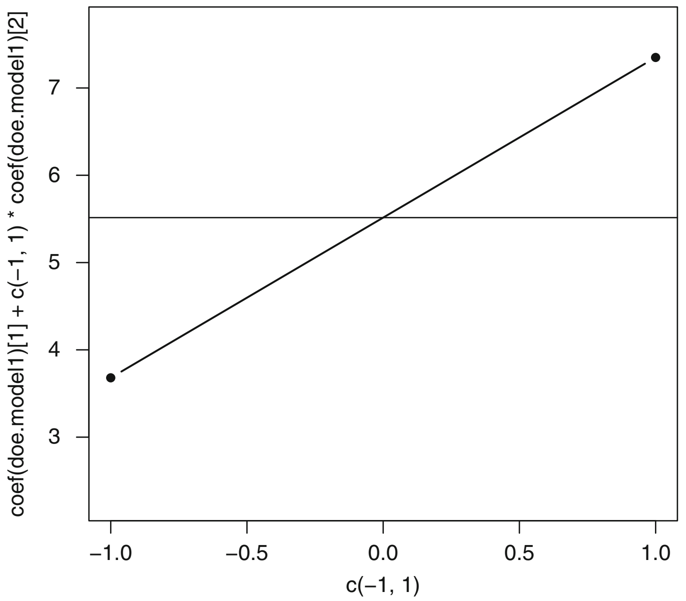


Fig. 11.3 Simple effect plot. We can simply plot lines and points representing the coefficients, a horizontal line for the intercept and linked points for the effects

We may also plot charts to visualize the effects with the `ggplot2` package. Using the following code we plot the main effects of the two factors in the same plot (Fig. 11.4):

```
> prinEf <- data.frame(Factor = rep(c("A_Flour",
                        "C_Baking Powder"), each = 2),
        Level = rep(c(-1, 1), 2),
        Score = c(aggregate(score ~ flour, FUN = mean, data =
             ss.data.doe1)[,2],
                aggregate(score ~ bakPow, FUN = mean, data =
                    ss.data.doe1)[,2]))
> p <- ggplot(prinEf,
        aes(x = Level, y = Score)) +
    geom_point() +
    geom_line() +
    scale_x_continuous(breaks = c(-1, 1)) +
    facet_grid(. ~ Factor) +
    stat_abline(intercept = mean(ss.data.doe1$score),
            slope = 0, linetype = "dashed") +
    opts(plot.background = theme_rect(colour = 'black'))
> print(p)
```

The effects of the interaction can be plotted by typing (Fig. 11.5)

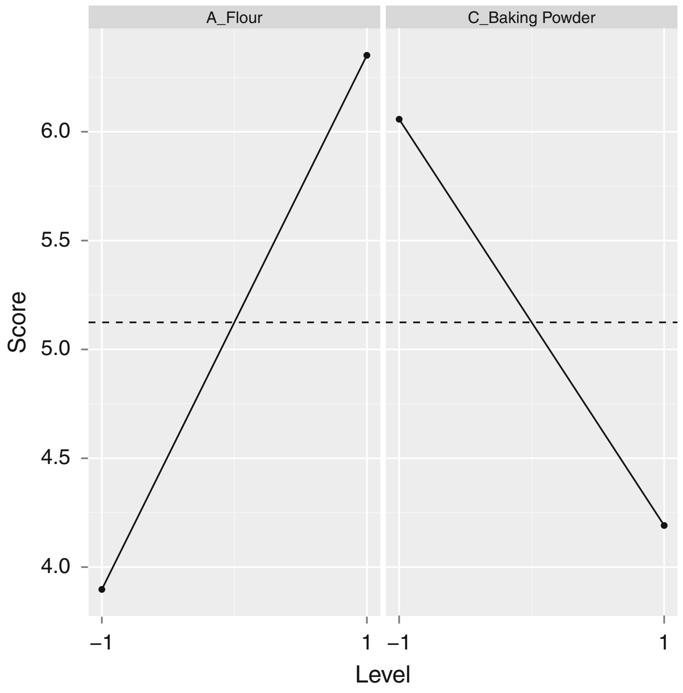


Fig. 11.4 Factor effects in pizza experiment. The effect of the flour is positive, whereas the effect of the baking powder is negative

```
> intEf <- aggregate(score ~ flour + bakPow,
                FUN = mean, data = ss.data.doe1)
> p <- ggplot(intEf, aes(x = flour, y = score, color = bakPow
    )) +
    geom_point() +
    geom_line(aes(group=bakPow)) +
    stat_abline(intercept = mean(ss.data.doe1$score),
        slope = 0,
        linetype = "dashed",
        color = "black")
> print(p)
```

Diagnostics of the model should be made by analyzing the residuals. We can plot the standard charts for linear models (Fig. 11.6):

```
> par(mfrow=c(2,2))
> plot(doe.model2)
> box("outer")
```

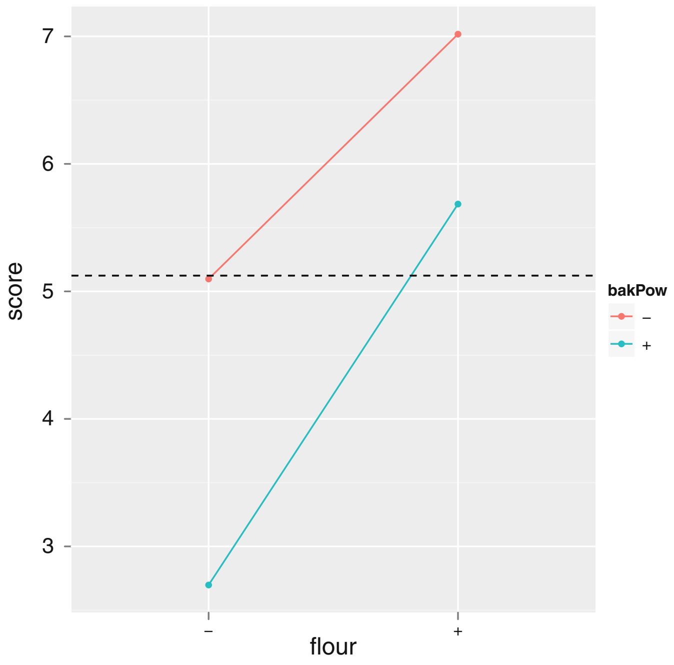


Fig. 11.5 Plot of interactions between factors. The *lines* do not cross, and therefore there is no interaction between the two factors plotted (flour and baking powder)

Finally, we should verify the normality of the residuals with a normality test. Let us try two of them:

```
> shapiro.test(residuals(doe.model2))
```

```
        Shapiro-Wilk normality test

data:  residuals(doe.model2)
W = 0.9065, p-value = 0.1023
```

```
> lillie.test(residuals(doe.model2))
```

```
        Lilliefors (Kolmogorov-Smirnov) normality test

data:  residuals(doe.model2)
D = 0.1891, p-value = 0.1313
```

Given that the p-values are large for both tests, we do not reject the normality of the residuals. □

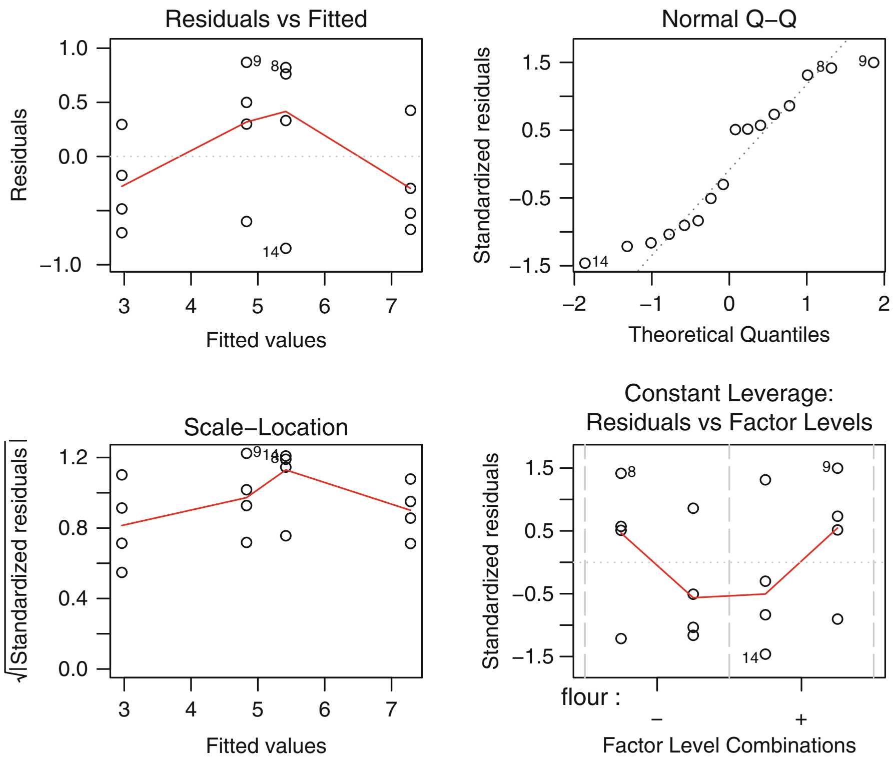


Fig. 11.6 Model diagnostics for pizza experiment. There are no clear patterns in the residuals. The normal q–q plot is not straight enough

11.5 Design of Experiments for Process Improvement

We have described a full DoE, but performing experiments may not be enough to improve a process. For instance, perhaps not all the X of the process have been identified, or some Xs may depend on external conditions and therefore are not under our control. Under these circumstances, one possibility is to design products that are robust to the environment (robust design). This consists of including *noise factors* within the experiment. These noise factors are those that may affect the CTQ characteristics, but they are not under our control. We must identify all the possible noise factors, and in this regard, the operational structure described in Chap. 1 becomes crucial. We can obtain valuable information about these external conditions from the process owner.

Example 11.4 (Pizza dough (cont.)). After a few weeks, the social media manager from the marketing department reported to the board of directors that some bad comments had been gathered from social media (mainly Facebook and Twitter) regarding a bad experience involving consumers of the pizza dough. The flavor of the pizzas made with the dough was not satisfactory.

To investigate the reasons, they decided to tackle the problem from a Six Sigma perspective. During the Measure phase, an accurate identification of the possible factors affecting the flavor of the pizza was made. A vital piece of information was provided by the process owner (the team member who came up with the preparation instructions). She pointed out that the exact baking conditions (temperature $= 180°$C and time $= 9$ min) were critical to achieve the flavor desired by customers.

Further research about how the customers were preparing pizzas showed that not everybody (actually, almost nobody) followed the preparation instructions in the box accurately. Therefore, a new design was proposed by the Black Belt to conduct new research about the pizza dough recipe. He decided to include two noise factors with two levels each: time (7 and 11 min, $t-$ and $t+$, respectively) and temperature (160°C and 200°C, $T-$ and $T+$, respectively). Now we have a 2^5 factorial design, with the following design matrix:

```
> pizzaDesign2 <- expand.grid(flour = gl(2, 1, labels = c("-"
    , "+")),
        salt = gl(2, 1, labels = c("-", "+")),
        bakPow = gl(2, 1, labels = c("-", "+")),
        temp = gl(2, 1, labels = c("-", "+")),
        time = gl(2, 1, labels = c("-", "+")),
        score = NA)
> pizzaDesign2
```

```
   flour salt bakPow temp time score
1      -    -      -    -    -    NA
2      +    -      -    -    -    NA
3      -    +      -    -    -    NA
4      +    +      -    -    -    NA
5      -    -      +    -    -    NA
6      +    -      +    -    -    NA
7      -    +      +    -    -    NA
8      +    +      +    -    -    NA
9      -    -      -    +    -    NA
10     +    -      -    +    -    NA
11     -    +      -    +    -    NA
12     +    +      -    +    -    NA
13     -    -      +    +    -    NA
14     +    -      +    +    -    NA
15     -    +      +    +    -    NA
16     +    +      +    +    -    NA
17     -    -      -    -    +    NA
18     +    -      -    -    +    NA
19     -    +      -    -    +    NA
20     +    +      -    -    +    NA
21     -    -      +    -    +    NA
22     +    -      +    -    +    NA
23     -    +      +    -    +    NA
24     +    +      +    -    +    NA
```

Table 11.3 Robust experimental data

	Flour	Salt	BakPow	T180t9	$T-t-$	$T+t-$	$T-t+$	$T+t+$	Mean	Sd
1	−	−	−	5.52	3.67	5.12	4.18	3.90	4.48	0.64
2	+	−	−	7.35	3.37	4.52	5.05	2.94	4.65	0.98
3	−	+	−	4.68	0.96	4.91	5.29	1.17	3.40	2.34
4	+	+	−	6.69	3.59	5.89	5.62	3.87	5.13	1.18
5	−	−	+	2.52	1.92	3.05	1.73	1.70	2.18	0.64
6	+	−	+	5.16	3.14	5.01	5.54	2.90	4.35	1.32
7	−	+	+	2.87	1.21	1.86	3.04	1.31	2.06	0.84
8	+	+	+	6.21	5.80	6.11	5.98	5.96	6.01	0.12

```
25        -     -       -     +     +     NA
26        +     -       -     +     +     NA
27        -     +       -     +     +     NA
28        +     +       -     +     +     NA
29        -     -       +     +     +     NA
30        +     -       +     +     +     NA
31        -     +       +     +     +     NA
32        +     +       +     +     +     NA
```

The Black Belt decided to make two replications. Given that we have to perform $2^5 = 32$ experiments per replication, a total of 64 observations were taken. Table 11.3 summarizes the data, showing the average result per recipe and noise factor combination. The whole set of data is contained in the `ss.data.doe2` data frame of the `SixSigma` package.

The average ("Mean" column) and standard deviation ("Sd" column) for each experiment are in Table 11.3. Notice that now the recipe with the best average score and the smallest variability is that with the three original factors at their high level. This recipe is the most convenient one taking into account the noise factors. This is an example of how Six Sigma can be used to improve processes through DoE. □

11.6 Summary and Further Reading

In this chapter, we provided a very brief introduction to the statistical technique known as design of experiments (DoE). The practical case presented is a very representative example of how DoE can be used within a Six Sigma project. The 2^3 design used throughout the chapter can be immediately extended to any 2^k factorial design.

A detailed description of DoE would entail writing a whole book. In this regard, a world-class reference on DoE is the book [11]. This reference includes more advanced ANOVA models such as, for instance, split-plot designs. A nice reference on two-level factorial experimentation is that by [66]. Other good works are the books [70] or [5]. These two books can be complemented with their respective R companions, [58] and [103], freely available on the Internet. These companion documents develop the examples in the books using R.

The books mentioned above include more complex models such as 2^{k-p} fractional factorial designs (useful when only a fraction of experimental trials is available) or response-surface designs (useful to find optimal process settings). A complete book on response-surface design is [75].

Regarding optimal experimental design (a technique to obtain optimum designs according to some statistical criterion), the article [63] is a comprehensive introduction. The book [85] uses R to tackle some areas of this topic.

Furthermore, Taguchi's work on robust experimental design, in [99], may be consulted.

There is a project devoted to industrial DoE with R. The information is available at [32]. The project includes two analysis packages (`DoE.base` and `FrF2`) and two interface packages (`DoE.wrapper` and `RcmdrPlugin.DoE`).

A complete collection of resources regarding DoE with R is the CRAN Task View on DoE & Analysis of Experimental Data [31]. In addition to the packages included in the above-mentioned project, the `AlgDesign`, `conf.design`, and `crossdes` packages are available for generic DoE. Other packages with DoE functionalities are `qualityTools` (containing methods associated with the DMAIC cycle), `agricolae` (with experimental designs applied to agriculture and plant breeding), and `rsm` (for response-surface designs). This task view is frequently updated, and we strongly recommend consulting it periodically.

Case Study

Based on your experience with helicopter production, try to determine the causes affecting the performance of the process and flight time. Design an experiment to find out the cause-and-effect relationships between the factors that you have identified (e.g., dimensions, accessories) and the response.

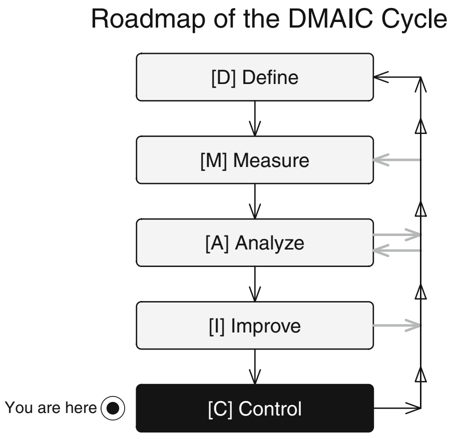
Randomize the experiment and take the measurements. Save the data and fit an ANOVA model using R. Estimate the effects and check to see which of the effects are significant. Write the formula of the underlying model.

Practice

11.1. Obtain the design matrix for a 2^4 experiment and create a randomized table to register the response.

11.2. Conduct an ANOVA for the complete experiment described in Sect. 11.5.

Part VI
R Tools for the Control Phase



In this part of the book, tools useful during the Control phase are introduced.

In this phase, the gains achieved in the previous phases must be monitored and the new process conditions documented. Moreover, sustainability of the new process structure needs to be guaranteed.

We will introduce mistake-proofing strategies and Control Charts, the most representative control tool.

Chapter 12
Process Control with R

Special causes of variation may be found and eliminated.
Walter A. Shewhart

12.1 Introduction

Engineers usually associate statistical process control (SPC) with a set of charts to monitor whether the outputs of a process are in or out of control. This is the classic approach to quality control (QC) and consists in adjusting processes only when their outputs are out of control. Under this approach, inspection is a standard way to proceed. One of the goals of modern QC is to reduce the need for inspection.

Six Sigma process control aims at sustaining the improvements achieved throughout the other stages of the DMAIC cycle. Under the Six Sigma paradigm, control is established over the variables affecting the critical to quality (CTQ) characteristics.

In this chapter, we first introduce some concepts of mistake-proofing strategies for process control. Then, control charts and their representation with R are explained. Finally, other topics related to SPC are touched upon along with the available R packages.

12.2 Mistake-proofing Strategies (Poka-Yoke)

In Six Sigma, the control issues start in the design stage of the product. This means that processes must be designed to be self-controlled, avoiding the influence of external factors such as, for example, operators, users, and tools. The old term used to define this approach of control is “fool-proofing” (becoming “idiot-proofing” in more colloquial parlance). The more polite expression “mistake-proofing” was finally adopted using the Japanese words “poka” (mistake)and “yoke” (avoid).

This is why these techniques are known as poka-yoke methods. A classic example of mistake-proofing design is the 3.5-in. diskette, which cannot be inserted upside down.

To achieve mistake-proofing processes, the objective must be zero defects. This implies that the design of the process should make it impossible to generate an error. In general, the more complex the poka-yoke method, the higher its cost.

Example 12.1 (Particle board factory). A particle board factory produces boards of several sizes. The key CTQ characteristics of the process are the dimensions, the thickness, and the density of the boards, in accordance with customer requirements.

Let us suppose that the process consists in obtaining the raw materials, making the particle boards, sawing them up into pieces as ordered by a customer, and packing and delivering them. An important task, and a possible source of errors, is the selection of the appropriate type of board (thickness and density) for each customer order.

An example of a poka-yoke method for this process is to store the boards in clearly separated and locked spaces. Before the operator takes the board, he must enter the board characteristics using his computer keyboard; then the space corresponding to the board specifications is automatically unlocked. In this way, it will be impossible for the operator to take a wrong board. Additional strategies could be implemented such as, for instance, designing and using specific wrappers that can only be used with boards corresponding to the width of the wrapper. □

12.3 What to Control

Under the Six Sigma approach, once we have identified the Xs of the process (the variables that cause the variation in the CTQ characteristics, i.e., Ys of the process) all we have to do is control these Xs. If the variability of the Xs is under control, then the process will perform correctly, leading to the required Ys and, therefore, high-quality products.

Example 12.2 (Particle board factory (cont.)). One of the key characteristics (Y) of the particle boards is their density. After some Six Sigma research, it is determined that the most important variable of the process affecting the density of the particle boards is the raw material humidity. Therefore, process control charts will be made according to a control plan measuring the humidity of the raw materials (X). □

We have already seen that under the Six Sigma paradigm the Xs of the process must be controlled. In some cases, the Ys of the process may also be monitored. In any case, monitoring only the Ys is not an option because this situation implies refusing to learn about the causes of the bad performance of the process.

12.4 Control Chart Basics

Control charts are the tool one should use to monitor the performance of variables involved in processes. Next we describe the basics of control charts and their interpretation.

12.4.1 The Chart

A control chart is a two-dimensional chart (Chap. 8) whose y-axis represents the variable we are monitoring. The values of the characteristic are plotted sequentially in the order in which they have arisen. Depending on the type of data we are analyzing, these values can be individual values or group means. Thus, the x-axis of the chart is an identification number of the set of items assessed (individual item or group). The values are plotted as points and linked with straight lines to identify patterns that show significant changes in the process performance. Along with the sequence of observations for the variable we are monitoring, three important lines are plotted:

1. *Center line (CL)*. This is the mean of the sampled variables. The monitored values vary around this mean.
2. *Lower control limit (LCL)*. This is the value below which it is very unlikely for the variable to occur.
3. *Upper control limit (UCL)*. This is the counterpart of the LCL on the up side of the variable. The LCL and UCL are symmetric if the probability distribution of the variable is symmetric (e.g., normal).

12.4.2 Chart Interpretation

The usual way to proceed with control charts follows a two-phase approach. In phase I, reliable control limits are estimated using a preliminary sample. From that point on, in phase II, the subsequent samples are plotted in a chart with the former control limits. For the sake of simplicity, in what follows we will plot the charts in a single phase, although in practice it should be done in two phases.

When the individual observations of the X are within the control limits, the process will be statistically under control. The control limits are completely different from the specification limits, that is, the limits beyond the process will not be accepted by the customer (Chap. 7). The control limits are computed as a confidence interval (Chap.10) that comprises a high proportion of the values. Typical control limits are those between the mean and three standard deviations ($\mu \pm 3\sigma$). For a normal probability distribution these limits include 99.7% of the data. Thus, there is only a probability of 0.3% for an individual observation to be outside the

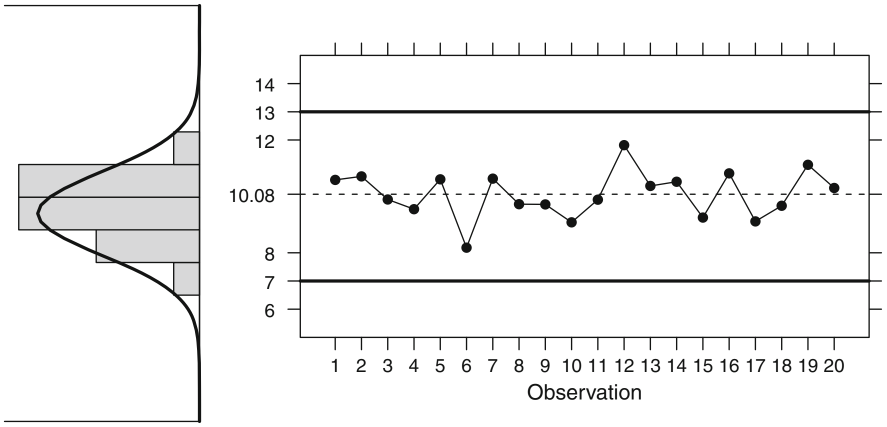


Fig. 12.1 Control charts vs. probability distribution. The control chart shows the sequence of the observations. The variation around the central line provides an idea of the probability distribution

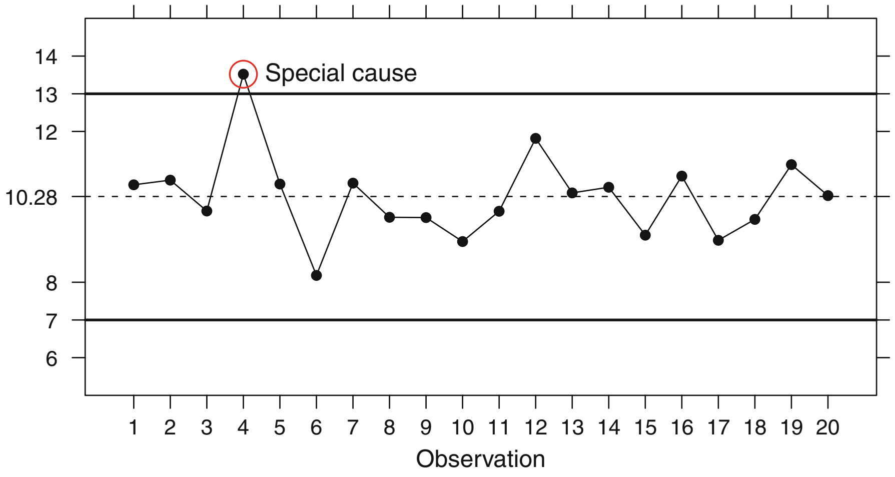


Fig. 12.2 Identifying special causes through individual points. When an individual point is outside the control limits, an investigation into the cause should be started to eliminate the root of the problem

specification limits. Moreover, a control chart adds information about the variation of the process. Figure 12.1 shows how both types of information are related.

Variation of the characteristic within the control limits is due to common causes, whereas variation outside the control limits is due to special causes. Common causes arise from randomness and cannot be eliminated. Special causes prompt variability that is not a consequence of randomness. Thus, when a point is outside the control limits, the (special) cause must be identified, analyzed, and eradicated (Fig. 12.2).

Special causes can also generate other persistent problems in a process. They can be identified through patterns in the chart. Three important patterns that we can detect are trends, shifts, and seasonality (Fig. 12.3). Other evidence to detect an out-of-control process are nine or more points to one side of the mean, or two out of three points beyond the center line plus or minus two standard deviations.

12.4.3 Sampling Strategy

Sometimes all the items produced are monitored, and the control charts are produced in run time. This is only possible for specific processes in which information related to the process performance can be automatically recorded by electronic means, or for highly reliable products that need to pass a verification step at the end of the production line before being delivered to the customer.

Most times control charts are plotted using data samples. In previous chapters we insisted on the importance of randomization. This is a must also when monitoring processes through control charts. Moreover, some particular concerns are to be taken into account with respect to control chart sampling:

- The larger the sample size, the higher the probability of detecting an error. On the down side, usually the cost associated with control increases with sample size.
- Samples are usually taken in subgroups (e.g., batches, days, departments), and these subgroups and their size should be rationally chosen.
- The subgroups must be independent of each other.
- A trade-off between independence and homogeneity should be attained in the measurements within the subgroups.
- Measurements must be recorded in chronological order, as they are produced.
- The sample size can be small when the process is under control and its capability exceeds the minimum requirements.

All in all, a general rule for rational control chart sampling could be "the sample represents the production under regular conditions." Think about this every time you plan to obtain a sample for controlling a process.

12.5 Plotting Process Control Charts

In this section, we describe the main types of control charts that can be used depending on the type of variable being monitored. For continuous variables, we can use individual and moving-range charts (I/MR charts) for complete monitoring and average and range charts ($\bar{x}-R$) or average and standard deviation charts ($\bar{x}-s$) for randomized monitoring through samples. For qualitative variables (attribute data)

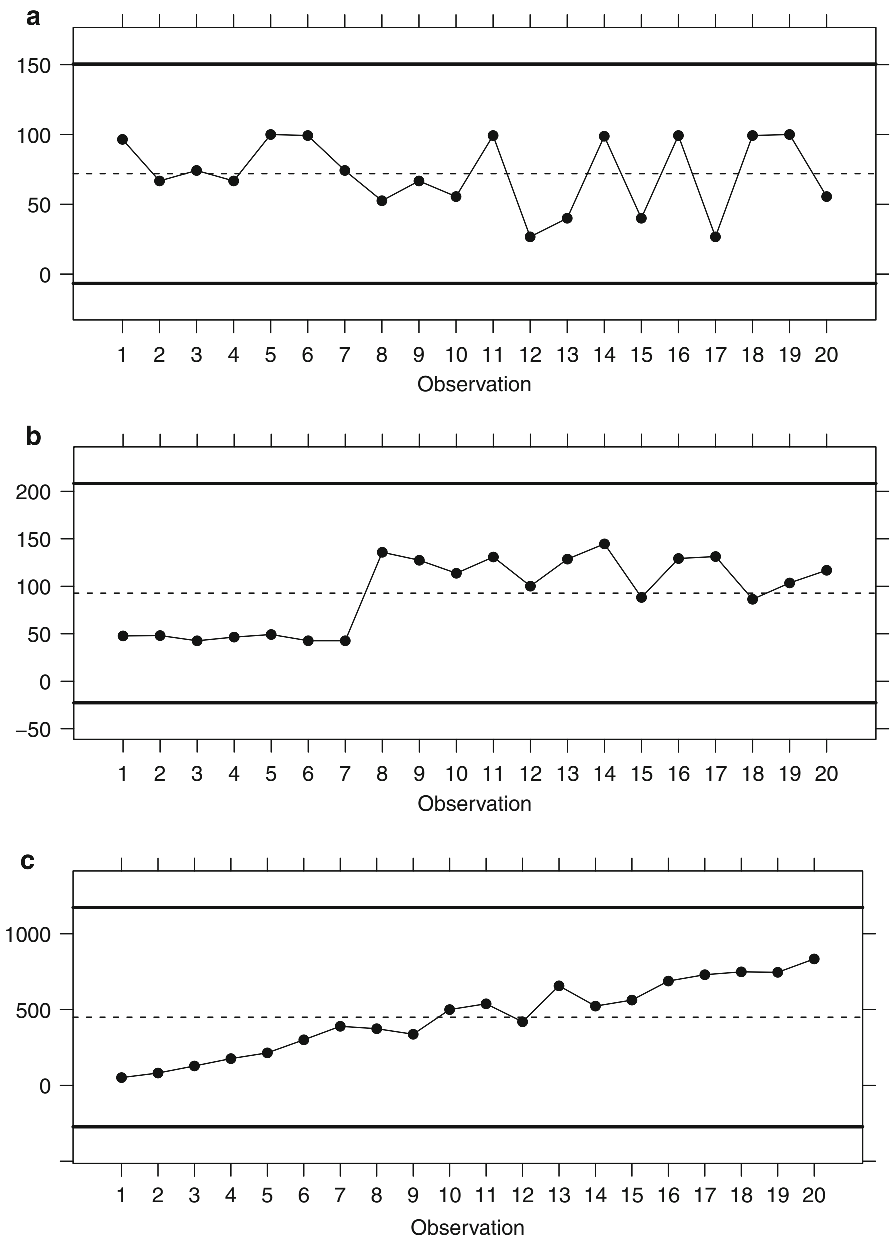


Fig. 12.3 Patterns in control charts. We can identify several patterns in a control chart: recurring cycles (**a**), shifts (**b**), and trends (**c**)

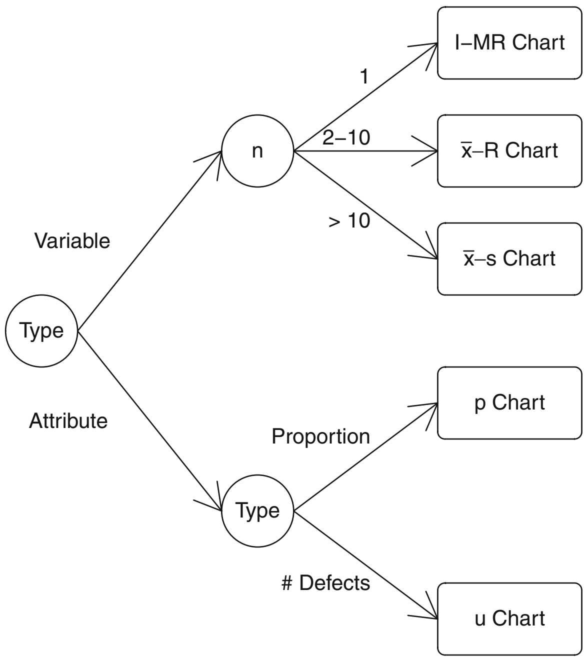


Fig. 12.4 Decision tree for basic process control charts. First, check your variable type. If it is a continuous variable, the chart depends on the group size. If it is an attribute variable, the chart depends on what you want to monitor (proportion of defects or number of defects per unit)

we can use *p* charts to control the proportion of defects and *u* charts to control the number of defects per unit. A decision tree summarizing the different options is shown in Fig. 12.4.

The R packages `qcc`, `IQCC`, and `qAnalyst` plot several types of control chart. We will use some functions in these packages for plotting control charts corresponding to the examples. Nevertheless, we provide the formulas for the control lines (center, upper limit, and lower limit) so as to allow you to plot your own control charts with the R graphical functions in the packages `graphics`, `lattice`, `ggplot2`, and many others, just plotting points and lines and adding control lines (CL, UCL, LCL). In this way, you can customize any feature of your control chart.

12.5.1 Notation for Computing Control Lines

We must compute the values for the center line (CL) and the upper and lower limit lines (UCL and LCL, respectively). We will use the following notation:

n	Number of elements sampled in each group (equal sample sizes);
k	Number of groups;
n_i	Number of elements sampled in the ith group (unequal sample sizes);
D_i	Number of defects in the ith group;

$\bar{\bar{x}}$	Overall sample mean;
$\bar{x}_i$	Sample mean of group i;
$\bar{s}$	Average standard deviation;
s_i	Standard deviation of group i;
d_2	Mean of random variable $W = \frac{R}{\sigma}$;
d_3	Standard deviation of random variable $W = \frac{R}{\sigma}$;
c_4	Constant for estimating σ as $\hat{\sigma} = \frac{s}{c_4}$.

Textbooks contain tables with "Shewhart's constants for constructing control charts." These tables are useful for teaching and assessing, but they make no sense when we are using computers. Actually, all of those constants can be computationally obtained using R. In particular, the `ss.cc.getc4`, `ss.cc.getd2`, and `ss.cc.getd3` functions return values of c_4, d_2, and d_3, respectively, passing as argument sample size n.

12.5.2 *Variable Control Charts*

The charts described in this section are suitable for monitoring continuous variables. For each group of control chart two plots are made, one for monitoring the central tendency and another for monitoring the variation.

Individual and Moving-range Charts (*I/MR*)

Sometimes we do not have groups of measurements. Reasons for this fact include that it simply makes no sense to have groups, or that every item is being automatically monitored. The center line and control limits for the individual control chart are computed as follows:

$$CL = \bar{\bar{x}} = \frac{\sum x_i}{n} \qquad i = 1 \ldots n,$$

$$UCL = \bar{\bar{x}} + 3\frac{\overline{MR}}{d_2},$$

$$LCL = \bar{\bar{x}} - 3\frac{\overline{MR}}{d_2},$$

where $\overline{MR}$ is the moving-range average, computed as

$$\overline{MR} = \frac{\sum_{i=2}^{k} MR_i}{k-1};\ MR_i = |x_{i+1} - x_i|.$$

The center line and the control limits for the moving-range control chart are

$$CL = \overline{MR},$$

$$UCL = \overline{MR} \times \left(1 + \frac{3d_3}{d_2}\right),$$

$$LCL = \overline{MR} \times \left(1 - \frac{3d_3}{d_2}\right).$$

If the LCL formula results in a negative number, then the LCL is taken as zero. The d_2 and d_3 constants are calculated for $n = 2$, as we are computing the range between two consecutive items.

Example 12.3 (Particle board factory (cont.)). A control plan has been designed to sustain the improvements achieved through previous Six Sigma projects. The Black Belt wants to asses complete lots of raw materials as they are received in the factory. Each batch has 30 items. As we want to control the individual values, the sample size is $n = 1$, and there are $k = 30$ groups. The data frame `ss.data.pb1` in the `SixSigma` package contains the values of humidity for each lot.

The `qcc` package flows to generate control charts, starts creating a `qcc.groups` object using the function identically named, with the vectors containing the measurements and the group identification as arguments. Next, a `qcc` object has to be created over the `qcc.groups` object, indicating the type of control chart. Once the `qcc` object is created, we can perform two actions over it, a summary and a plot, using the generic functions `summary` and `plot`, respectively. Firstly, we prepare data objects and obtain text output with the generic `summary` function:

```
> require(qcc)
> pb.groups.one <- with(ss.data.pb1, qcc.groups(pb.humidity,
    pb.group))
> pb.xbar.one <- qcc(pb.groups.one, type="xbar.one")
> summary(pb.xbar.one)
```

```
Call:
qcc(data = pb.groups.one, type = "xbar.one")

xbar.one chart for pb.groups.one


Summary of group statistics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  9.783  10.720  11.670  11.750  12.760  14.150

Group sample size:  30
```

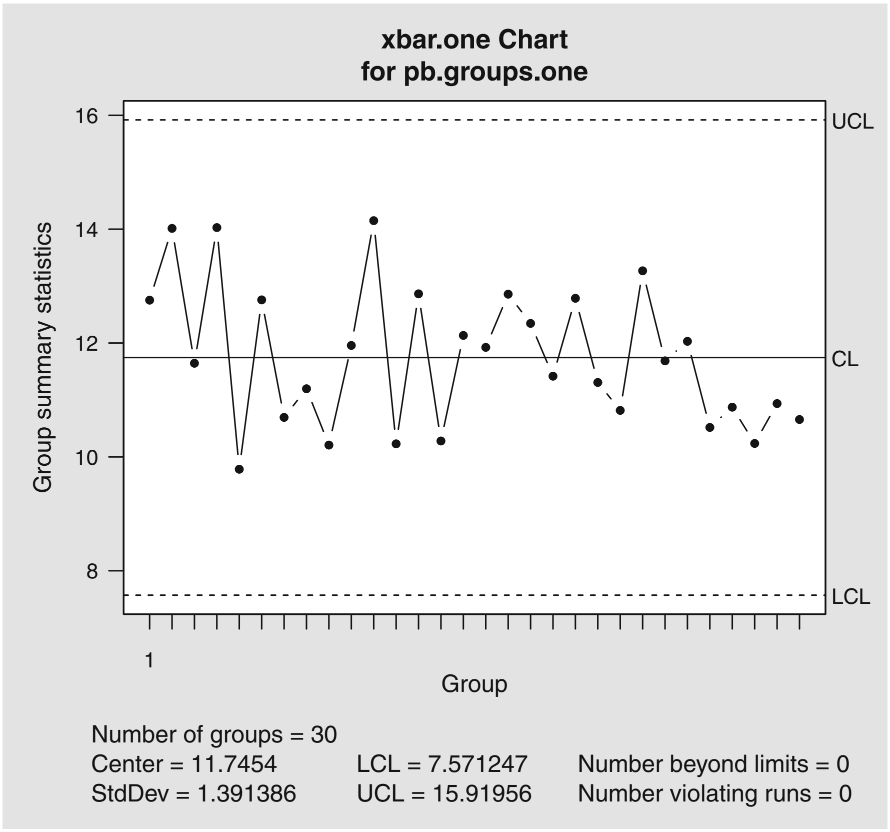


Fig. 12.5 Individual control chart. In this case, the process is statistically controlled. There are no individual points beyond the control limits or other suspicious patterns

```
Number of groups:  30
Center of group statistics:  11.7454
Standard deviation:  1.391386

Control limits:
      LCL      UCL
 7.571247 15.91956
```

The text output includes some summary statistics and the control limits. Then we can plot the individual control chart with the generic `plot` function (Fig. 12.5):

```
> plot(pb.xbar.one)
```

For phase II of control charting, you can pass the center line and the limits to the `qcc` function as arguments. They can be taken from the components `center` and `limits` of a `qcc` object previously saved, for example:

```
> plot(qcc(data = pb.groups.one,
    type = "xbar.one",
    center = pb.xbar.one$center,
    limits = pb.xbar.one$limits))
```

Notice that we have used the same data set as that used to calculate the control limits (in this case we obtain the same chart as that in Fig. 12.5). To plot a chart with the same control limits and new data arising from the process, the only thing we have to do is change the `data` argument.

As there is no option for moving-range charts in the `qcc` package, we will use the `qAnalyst` package. As with the `qcc` package, we must create an `spc` object and then call the generic functions:

```
> require(qAnalyst)
> pb.mr <- with(ss.data.pb1, spc(pb.humidity, sg = 2, type="
    mr"))
> print(pb.mr)
```

```
 mr chart of pb.humidity

 pb.humidity  main stats
 -----------------------------------------
                              value
Total observations    30.000000
complete observations 30.000000
missing observations   0.000000
number of groups       1.000000
Mean                  11.745404
min                    9.783126
max                   14.150043
total std. dev.        1.236948
average range          1.569483
```

The generic `print` function returns only some stats, but the generic `summary` function also returns information about the tests performed, probability, and the individual values of the plotted points (R_i):

```
> summary(pb.mr)
```

```
 mr chart of pb.humidity

 pb.humidity  main stats
 -----------------------------------------
                              value
Total observations    30.000000
complete observations 30.000000
missing observations   0.000000
number of groups       1.000000
Mean                  11.745404
min                    9.783126
max                   14.150043
total std. dev.        1.236948
```

```
average range          1.569483

 Control chart tests results
 ------------------------------------------

 Matrix of points failing required tests

 All tests successful

 Probability of having such a number of Test1 failing points:
0.9973002

 Probability of having a number of Test1 failing points
     greater or equal to:
1

 Control chart elements table
 ------------------------------------------
    points lcl3s   lcl2s  lcl1s center line  ucl1s
1       NA    NA      NA     NA          NA     NA
2  1.26104     0 0.52316 1.0463      1.5695 2.7563
3  2.36949     0 0.52316 1.0463      1.5695 2.7563
4  2.38330     0 0.52316 1.0463      1.5695 2.7563
5  4.24504     0 0.52316 1.0463      1.5695 2.7563
6  2.97527     0 0.52316 1.0463      1.5695 2.7563
7  2.06458     0 0.52316 1.0463      1.5695 2.7563
8  0.50367     0 0.52316 1.0463      1.5695 2.7563
9  0.98972     0 0.52316 1.0463      1.5695 2.7563
10 1.75021     0 0.52316 1.0463      1.5695 2.7563
11 2.19208     0 0.52316 1.0463      1.5695 2.7563
12 3.92027     0 0.52316 1.0463      1.5695 2.7563
13 2.63488     0 0.52316 1.0463      1.5695 2.7563
14 2.58481     0 0.52316 1.0463      1.5695 2.7563
15 1.85428     0 0.52316 1.0463      1.5695 2.7563
16 0.20995     0 0.52316 1.0463      1.5695 2.7563
17 0.93413     0 0.52316 1.0463      1.5695 2.7563
18 0.51340     0 0.52316 1.0463      1.5695 2.7563
19 0.92735     0 0.52316 1.0463      1.5695 2.7563
20 1.36862     0 0.52316 1.0463      1.5695 2.7563
21 1.47827     0 0.52316 1.0463      1.5695 2.7563
22 0.49094     0 0.52316 1.0463      1.5695 2.7563
23 2.45189     0 0.52316 1.0463      1.5695 2.7563
24 1.58140     0 0.52316 1.0463      1.5695 2.7563
25 0.34312     0 0.52316 1.0463      1.5695 2.7563
26 1.51285     0 0.52316 1.0463      1.5695 2.7563
27 0.35563     0 0.52316 1.0463      1.5695 2.7563
28 0.63719     0 0.52316 1.0463      1.5695 2.7563
29 0.70122     0 0.52316 1.0463      1.5695 2.7563
30 0.28038     0 0.52316 1.0463      1.5695 2.7563
    ucl2s ucl3s
1      NA    NA
2  3.9432  5.13
3  3.9432  5.13
```

```
4  3.9432  5.13
5  3.9432  5.13
6  3.9432  5.13
7  3.9432  5.13
8  3.9432  5.13
9  3.9432  5.13
10 3.9432  5.13
11 3.9432  5.13
12 3.9432  5.13
13 3.9432  5.13
14 3.9432  5.13
15 3.9432  5.13
16 3.9432  5.13
17 3.9432  5.13
18 3.9432  5.13
19 3.9432  5.13
20 3.9432  5.13
21 3.9432  5.13
22 3.9432  5.13
23 3.9432  5.13
24 3.9432  5.13
25 3.9432  5.13
26 3.9432  5.13
27 3.9432  5.13
28 3.9432  5.13
29 3.9432  5.13
30 3.9432  5.13
```

Finally, we can plot the control chart in Fig. 12.6 using the generic `plot` function:

```
> plot(pb.mr, cex=list(cexStrip=1.5, cexAxes=1))
```

Using the `cex` argument we can change the size of the labels and titles. □

$\bar{X}$ – R Chart

The center line and control limits for the $\bar{x}$ control chart are computed as follows:

$$CL = \bar{\bar{x}},$$

$$ULC = \bar{\bar{x}} + \overline{R} \times \frac{3}{d_2\sqrt{n}},$$

$$LCL = \bar{\bar{x}} - \overline{R} \times \frac{3}{d_2\sqrt{n}},$$

where $\overline{R}$ is the average range, computed as

$$\overline{R} = \frac{\sum_{i=1}^{k} R_i}{k}, \; R_i = \max\{x \in X_i\} - \min\{x \in X_i\}.$$

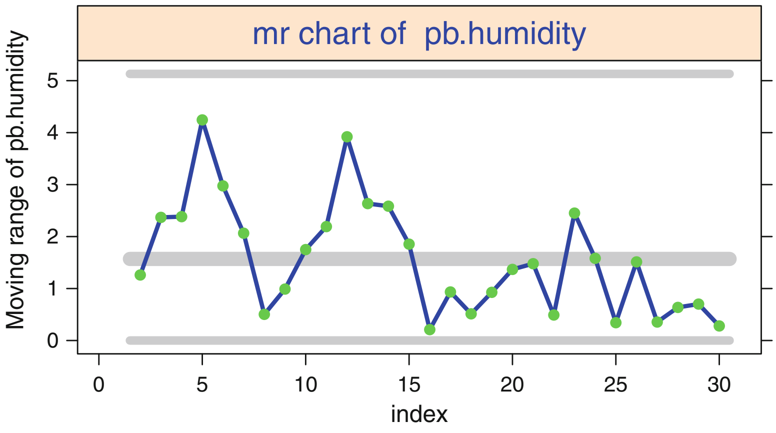


Fig. 12.6 Moving-range control chart. All the points are inside the control limits. The lower control limit is zero

The following formulas are for the R control chart:

$$CL = \overline{R},$$

$$ULC = \overline{R} \times \left(1 + \frac{3d_3}{d_2}\right),$$

$$LCL = \overline{R} \times \left(1 - \frac{3d_3}{d_2}\right)$$

LCL is taken as zero when the formula results in a negative number, and constants d_2 and d_3 are computed using the size of the groups.

Example 12.4 (Particle boards factory (cont.)). In the production line, the humidity of raw materials is measured again before starting the manufacture of the board. Five samples of each lot are obtained, and the total number of lots is 20. The data are in the `ss.data.pb2` data frame in the `SixSigma` package.

Using the following code we create the $\bar{x}$ chart for the 20 groups of size 5. Figure 12.7 is obtained by typing `plot(pb.xbar)`. To obtain the s chart, you only have to change the value of the `type` argument in the `qcc` function.

```
> pb.groups.xbar <- with(ss.data.pb2, qcc.groups(pb.humidity,
     pb.group))
> pb.xbar <- qcc(pb.groups.xbar, type="xbar")
> summary(pb.xbar)
```

```
Call:
qcc(data = pb.groups.xbar, type = "xbar")

xbar chart for pb.groups.xbar
```

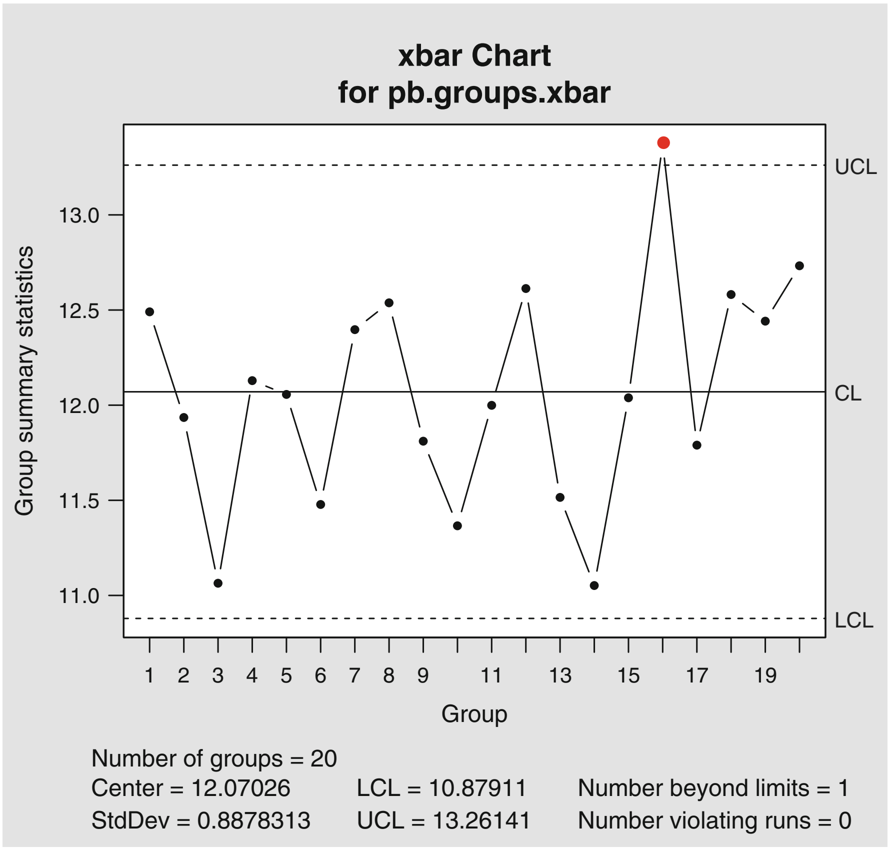


Fig. 12.7 $\bar{x}$ control chart. There is a point outside the control limits corresponding to group 16. The special cause must be identified

```
Summary of group statistics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  11.05   11.72   12.05   12.07   12.50   13.38

Group sample size:  5
Number of groups:  20
Center of group statistics:  12.07026
Standard deviation:  0.8878313

Control limits:
      LCL      UCL
 10.87911 13.26141
```

□

$\bar{X}$ – S Chart

The center line and control limits are computed as follows:

$$CL = \bar{s} = \frac{\sum_{i=1}^{k} s_i}{k},$$

$$ULC = \bar{s} \times \left(1 + 3\frac{\sqrt{1-c_4^2}}{c_4}\right),$$

$$LCLZ = \bar{s} \times \left(1 - 3\frac{\sqrt{1-c_4^2}}{c_4}\right), \tag{12.1}$$

where s_i is the sample standard deviation within each group.

12.5.3 Attribute Control Charts

The charts described in this section are suitable for monitoring qualitative (attributes) variables. The p chart is the appropriate one when the characteristic we want to monitor and control is a proportion. It is based on the binomial distribution. Otherwise, the u chart is suitable for monitoring and control the number of defects per unit, which is based on the Poisson distribution.

p Chart

A typical use of the p chart is to control the proportion of defects per time unit, e.g., the number of wrong invoices per week. In these cases, the sizes are not equal in all the groups, and we have stepwise control limits. Thus, equal group sizes represent a particular case for the following formulas. The center line and the control limits are computed as follows:

$$CL = \bar{p} = \frac{\sum_{i=1}^{k} D_i}{\sum_{i=1}^{k} n_i},$$

$$ULC_i = \bar{p} + 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n_i}},$$

$$LCL_i = \bar{p} - 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n_i}},$$

where D_i is the number of defects in the ith group.

Table 12.1 Data for *p* chart example

pb.group	pb.humidity	pb.group	pb.humidity	pb.group	pb.humidity	pb.group	pb.humidity
1	14.32	6	10.44	11	11.36	16	13.87
1	12.22	6	12.35	11	11.56	16	13.78
1	12.42	6	12.42	11	11.21	16	13.56
1	11.81	6	11.70	11	13.57	16	13.10
1	11.69	6	10.48	11	12.30	16	12.57
2	11.35	7	13.87	12	13.35	17	13.31
2	11.24	7	12.90	12	12.58	17	11.85
2	13.23	7	11.65	12	12.42	17	10.00
2	11.82	7	12.59	12	11.14	17	11.12
2	12.03	7	10.98	12	13.57	17	12.68
3	10.86	8	14.37	13	11.85	18	12.13
3	11.01	8	11.60	13	10.84	18	12.74
3	11.69	8	13.27	13	11.75	18	12.95
3	10.70	8	12.08	13	12.24	18	11.90
3	11.05	8	11.37	13	10.91	18	13.19
4	12.18	9	11.40	14	11.57	19	12.39
4	11.71	9	10.33	14	11.34	19	10.90
4	12.13	9	13.49	14	10.93	19	13.47
4	12.11	9	11.47	14	11.49	19	12.97
4	12.52	9	12.36	14	9.93	19	12.48
5	11.93	10	11.71	15	11.62	20	12.38
5	12.07	10	10.98	15	11.32	20	12.99
5	13.83	10	11.91	15	12.32	20	13.31
5	11.35	10	10.30	15	12.96	20	11.74
5	11.10	10	11.93	15	11.97	20	13.24

Example 12.5 (Particle board factory (cont.)). A CTQ characteristic for the service processes of the particle board factory is the proportion of on-time delivered orders (Y). In the framework of a Six Sigma project, it was discovered that one of the factors affecting the delivery time is the lack of appropriate raw materials when the order arrives at the factory. Therefore, a control plan is designed to monitor and control the proportion of out-of-stock orders. As the number of orders received every day is not constant, the control chart will present stepwise control limits. The `ss.data.bp3` data frame in the `SixSigma` package contains the daily stockouts and number of orders in a given month of 22 production days (Table 12.1).

The *p* chart in Fig. 12.8 is plotted using the following code with the `qcc` package:

```
> with(ss.data.pb3,
       plot(qcc(stockouts, orders, type ="p"))
  )
```

□

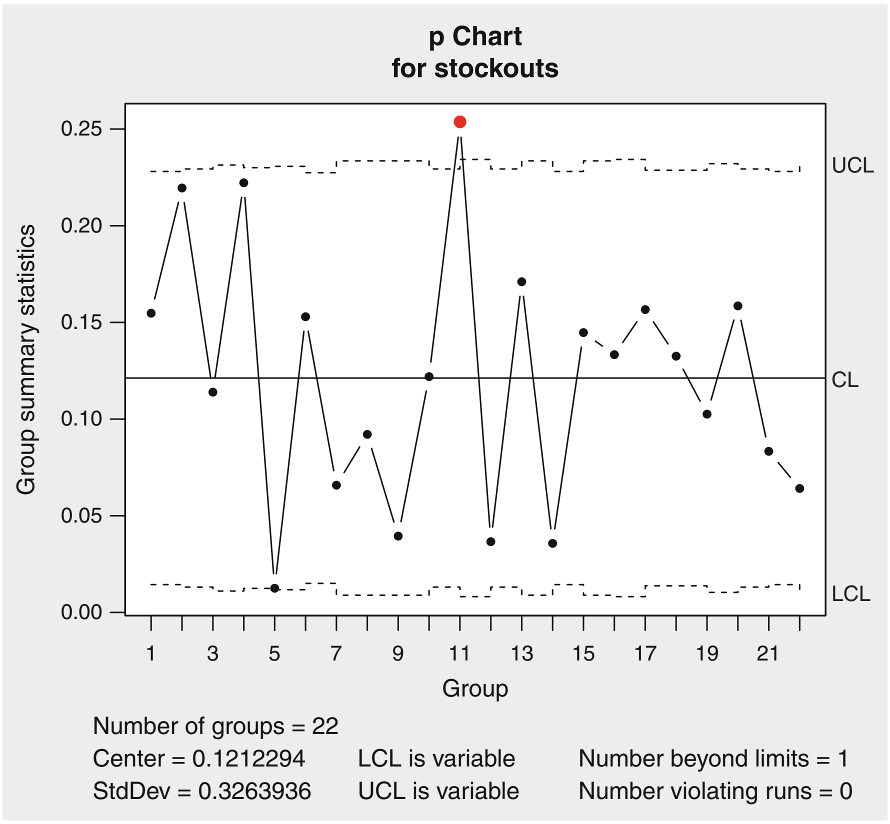


Fig. 12.8 *p* chart for particle board example. There is a point on the 11th day that is out of control

u Chart

The center line and the control limits are computed as follows:

$$CL = \bar{u} = \frac{\sum_{i=1}^{k} D_i}{\sum_{i=1}^{k} n_i},$$

$$ULC_i = \bar{u} + 3\sqrt{\frac{\bar{u}}{n_i}},$$

$$LCL_i = \bar{u} - 3\sqrt{\frac{\bar{u}}{n_i}},$$

where D_i is the number of defects in the ith group.

The command for plotting the control chart with the `qcc` package is the same, passing `"u"` as the `type` argument. Though the data structure is the same, the type of data is completely different: proportion of defects vs. number of defects per unit.

12.6 Summary and Further Reading

We have described so far the most representative control charts and the concepts needed for an overall understanding of the topic. In the literature there exist a variety of charts, for instance, *np* charts or *c* charts for attribute variables and exponentially weighted moving average charts (EWMA) for continuous variables. For multiple-characteristic processes there are also several possibilities for control charting such as three-way charts or Hotelling T2 multivariate charts. The book by [69] describes in detail these and other charts. The `qcc` package, [93], contains tools for some of these charts, and other useful functions for Six Sigma projects such as, for instance, operating characteristic (OC) curves or process capability analysis. Other packages containing useful functions for SPC are `IQCC`, [86]; `qAnalyst`, [98]; `spc`, [54]; and `spcadjust`, [28].

Other classical SPC tools that may be useful for Six Sigma projects are acceptance sampling and reliability analysis. Acceptance sampling is a technique used to accept or reject a material lot, using statistics and sampling. Several sampling strategies exist, including single, double, and multiple. The `AceptanceSampling` R package [52] deals with acceptance sampling plans and the related OC functions. Another interesting topic is reliability, that is, the probability that a system will survive after functioning for a period of time under certain conditions. There are several suitable probability models for system reliability; see, for instance, [96]. The `survival` R package contains useful functions for this topic; see [100].

Time-honored references in statistical quality control are [69] and [47]. [2] contains some case studies and advanced methods. [49] and [34] present straightforward explanations and guides for SPC.

Case Study

Construct control charts for the X variables you have identified while developing your Six Sigma project over the course of reading this book. Plot attribute control charts for the number of defects or any other qualitative variable affecting the CTQ characteristic. Plot variable control charts for the continuous Xs. Look for the special causes motivating the out-of-control points and arrange the necessary actions to eradicate them.

Practice

12.1. Plot the *s* control chart for the `ss.data.pb2` data frame.

12.2. In the particle board factory example described in the chapter, the CTQ characteristic defined as "number of customer complaints" (Y) is due to dents in the boards (X). To monitor this variable, the number of dents per square meter is automatically measured using precision cameras before packaging the pieces to be delivered to customers. The data corresponding to 80 orders on a given day are in the `ss.data.pb4` data frame. Plot the *u* control chart for the `ss.data.pb4` data frame.

Part VII
Further and Beyond

The last part of the book is a chapter of miscellanea where we name, list, or describe other tools and methodologies related to Six Sigma to provide the reader with a global view of the advanced possibilities of both Six Sigma and R.

Chapter 13
Other Tools and Methodologies

Instruction does much, but encouragement everything.
Johann Wolfgang von Goethe

13.1 Introduction

In this last part of the book, we outline some tools and methodologies that deserve at least a mention to encourage the reader to go further and beyond with Six Sigma and R. In this chapter, Sect. 13.2 briefly describes a useful tool for the Analyze and Improve phases: failure mode, effects, and criticality analysis (FMECA). Sections 13.3 and 13.4 summarize design for Six Sigma (DFSS) and lean quality approaches, respectively. Section 13.5 reviews the Gantt chart, a universally used tool for project planning. In Sect. 13.6, some advanced R topics are tackled to reveal the stunning possibilities of the system.

13.2 Failure Mode, Effects, and Criticality Analysis

FMECA is a tool to understand process risk. Action plans for managing risk are the results of this tool. FMECA was devised by NASA early in the U.S. Apollo space program in the 1960s. In the late 1970s, automotive companies, driven by liability cost issues, began to incorporate FMECA into the management of their processes. Many automotive, aeronautics, and electronics companies use FMECA principles for new product/process introduction.

FMECA can be used within processes (P-FMECA), designs (D-FMECA), or systems (S-FMECA). The purpose of a P-FMECA is to identify potential failure in terms of the process purpose (e.g., scrapped parts). D-FMECA is used to detect future potential component failure in terms of component function (e.g., forms not understood by a customer). S-FMECA evaluates failures within the system and in

Table 13.1 FMECA risk matrix

Component	Failure mode	Effect	Causes	P	S	D	RPN	Corrective or preventive action

interfaces with other systems (e.g., failures to meet regulatory standards). Basically, FMECA is based on risk matrices that use the following concepts:

- Potential failure modes, the probability (P) of failure;
- Effects of failure, severity (S) of failure;
- Chances of detection of failure, detectability (D) of failure;
- Prioritized list of risks, criticality or risk priority number (RPN) of failure, where $RPN = P \times S \times D$.

Table 13.1 displays an empty typical risk matrix for FMECA.

Such tables may be generated using R as a reporting tool. We will show how to generate reports within R in Sect. 13.6.3.

13.3 Design for Six Sigma

DFSS is a methodology to design new products and processes free of defects or to redesign existing ones. This is especially useful when new processes or products are to be designed or created, with no previous experimentation or data available.

Whereas there is a consensus on the DMAIC cycle as the best strategy to carry out Six Sigma projects, there are several distinct improvement cycles used for DFSS, namely:

DMADV	Define, Measure, Analyze, Design, and Verify;
DMADOV	Define, Measure, Analyze, Design, Optimize, and Verify;
IDOV	Identify, Design, Optimize, and Validate;
DMEDI	Define, Measure, Explore, Develop, and Implement.

In our view, DMADV is the more convenient approach. The first three stages are the same as in the DMAIC cycle and after them the redesign of the product is decided. Once the new design is approved, it is validated in the Verify phase.

DFSS is less extended than Six Sigma because the number of projects that need a complete design is smaller. Moreover, it is usually easier to improve something that already exists than to redesign it entirely.

13.4 Lean

Lean, or lean manufacturing, is a management philosophy derived from the Toyota Production System. Many lean techniques are simple and common sense. However, in practice only a few companies have achieved truly world-class operational efficiency. Lean techniques are often mistakenly associated with volume manufacturing. Nevertheless, the same principles are also applied in, for example, project management, engineering, supply chain, finance, human resources.

Efficient processes and operations are correlated with the elimination of non-value-adding activity (waste), in other words, producing only what is required, when it is required, and doing things right the first time. Lean concepts are aimed at identifying and eliminating waste. In this regard, waste entails the following inefficiencies:

- Overproduction,
- Inventory,
- Motion,
- Waiting,
- Transportation,
- Overprocessing,
- Defects (scrap/rework).

Lean manufacturing works in the following way:

1. Removing waste in terms of time, material, and cost from the value stream results in more capacity to deliver on time.
2. Waste removal usually contributes to process-variation reduction. The process becomes more robust and, therefore, on quality.
3. On-time and on-quality delivery at every step in the process will greatly support on-cost performance.

13.5 Gantt Chart

A Gantt chart is a type of horizontal bar chart useful for planning and scheduling projects. It shows project tasks and responsibilities mapped out over the project calendar time. Every bar in a Gantt chart represents a task. The x-axis is time scaled, and the bars are plotted from the task starting date to the task finishing date. Thus, such charts make clear not only the duration of tasks but also the relationships and interactions among them. A Gantt chart is a visual tool very helpful for the following purposes:

- Monitoring a project's progress;
- Illustrating the start and finish dates of project activities;

- Showing the current schedule status using the percentage of work finished within activities completed;
- Planning how long a project should take;
- Laying out the order in which tasks need to be carried out;
- Depicting project milestones, deadlines, and other significant events.

The R packages `plan`, [50], and `plotrix`, [61], provide functions for plotting Gantt charts. For instance, the following example for the `plotrix` package[1] returns the Gantt chart in Fig. 13.1:

```
> require(plotrix)
> Ymd.format <- "%Y/%m/%d"
> gantt.info <- list(labels =
    c("First task"," Second task", "Third task",
        "Fourth task", "Fifth task"),
    starts = as.POSIXct(strptime(
        c("2012/01/01", "2012/02/02", "2012/03/03",
            "2012/05/05", "2012/09/09"),
    format = Ymd.format)),
    ends = as.POSIXct(strptime(
        c("2012/03/03", "2012/05/05", "2012/05/05",
            "2012/08/08", "2012/12/12"),
    format = Ymd.format)),
    priorities = c(1, 2, 3, 4, 5))
> vgridpos <- as.POSIXct(strptime(c("2012/01/01",
        "2012/02/01", "2012/03/01", "2012/04/01",
        "2012/05/01", "2012/06/01", "2012/07/01",
        "2012/08/01", "2012/09/01","2012/10/01",
        "2012/11/01", "2012/12/01"),
    format = Ymd.format))
> vgridlab <- month.abb
> gantt.chart(gantt.info, main = "Calendar date Gantt chart
    (2012)",
    priority.legend = TRUE, vgridpos = vgridpos,
    vgridlab = vgridlab, hgrid = TRUE)
```

13.6 Some Advanced R Topics

13.6.1 Programming

One of the strengths of R is that it is not only a statistical software but also a programming language. The R language is a scripting programming language that is interpreted line by line. It allows control structures such as conditions and loops. Regarding loops, we can avoid most loops by operating directly over R data

[1]Type `example(plot.gantt)` to see a similar example for the `plan` package.

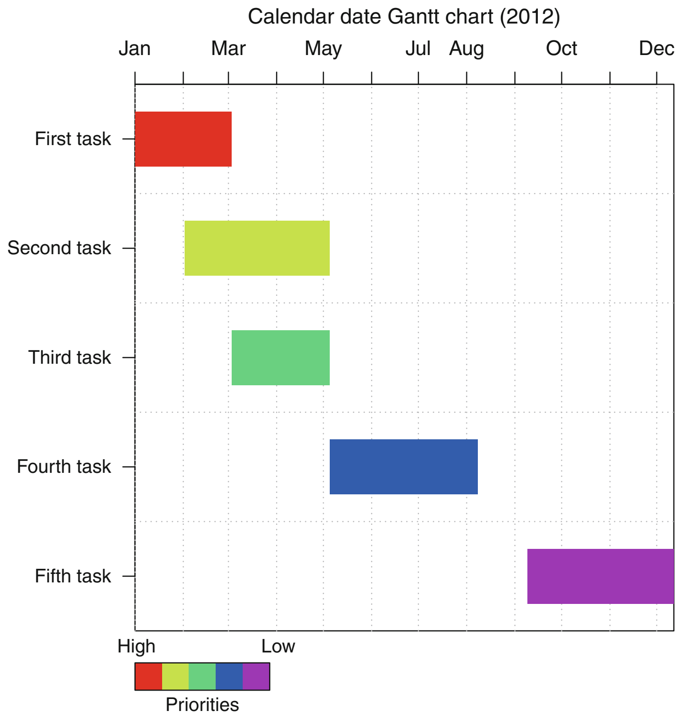


Fig. 13.1 Gantt chart. In a Gantt chart, each task is represented as a bar whose length is proportional to its time assigned within the project

structures such as vectors or matrices. Loops are in general computationally less efficient than the application of functions over data objects. For example, let us suppose that we have a vector with 10 values from a Poisson distribution. We want to create a new vector with random values from a standard normal distribution whose mean is the value in the original vector if it is greater than 3. We can perform this task with a typical loop:

```
> set.seed(666)
> my.vector <- rpois(10, 5)
> my.result <- numeric()
> for (i in 1:10){
    if (my.vector[i] > 3) {
        my.result <- c(my.result, rnorm(1, my.vector[i]))
    }
 }
> my.result
```

```
[1] 7.758396 8.693815 3.197480 4.207759 9.957968
[6] 7.150043
```

Note that the blocks inside the control structures are between braces "{" and "}," and the statements to evaluate the control structure are between parentheses "(" and ")." The `set.seed` function allows us to obtain reproducible results when generating random numbers.

Instead of doing all that work in a "looping" programming paradigm, the correct way to obtain the same result in R is by using a single command:

```
> set.seed(666)
> my.vector <- rpois(10, 5)
> sapply(my.vector[my.vector > 3], function(x) rnorm(1, x))
```

```
[1] 7.758396 8.693815 3.197480 4.207759 9.957968
[6] 7.150043
```

Functions are the heart of R programming. A function is a script that creates a `function` object, ready to be used in the system. A function object requires the creation of two elements: the body and the formal arguments. For example, if we want to create a function for the variance estimator $\hat{\sigma}^2$, we can create the function `popVar` with the following code:

```
> popVar <- function(x){
    var(x) * ((length(x) - 1) / length(x))
  }
```

The body function is between braces, and the formal arguments are between parentheses after the reserved word "function." Once the function is available in the R workspace, it can be called from the R Console or included within other functions or script files:

```
> popVar(my.vector)
```

```
[1] 8.36
```

The R language is very versatile and can be as complex as the user needs it to be. A user who knows nothing about programming can write its scripts just for repetitive tasks. An analyst can build programs for more complex analyses. Programmers can develop complete applications to customize R for their company's needs. Parallel computing is supported for high-performance computational needs. It is also possible to create customized menus and interfaces, and eventually to develop a package to encompass all customizations, regardless of whether it is shared with the R community or not.

13.6.2 R User Interfaces

The R Console is the first interface an R user must have a thorough knowledge of. As a user advances in the use of R, a better interface is advisable. There are several options, and there will probably be more in the future. The selection will depend on the user's preferences and reasons for using R. EMACS and ESS[2] are the favorites for many programmers. ECLIPSE is a complete IDE (integrated development environment) for any programming purposes, and the StatET plugin[3] allows you to create complete R projects and even debug code. RStudio[4] also provides an integrated interface, including an object explorer. All of them have enhanced editors to create scripts and be more productive.

13.6.3 Reporting

We can generate very sophisticated documents with R. The main concept in this regard is "literate programming," which consists in the generation of a report generated from a combination of content and data analysis code. A process will blend all the stuff (content and data analysis results), creating the final document. The main advantages of this system are that (1) it facilitates reproducible research, in the sense that under different circumstances (another analyst, at a future time), the results will be reproducible; and (2) any change in the analysis is automatically updated in the charts or parts of the document where the change should have an effect. Thus, we will be more productive and have fewer errors. Therefore, R is a perfect companion for Six Sigma projects.

The "native" tool for literate programming in R is the `Sweave` function.[5] This function converts a file containing R code and LaTeX code[6] into a dvi or pdf document with any content, accurately formatted and prepared for delivery or publication.

For example, the following code in an .Rnw file parsed with the `Sweave` function produces a table with the design matrix described in Chap. 11. We use the `xtable` function in the `xtable` package.

```
> require(xtable)
> my.table <- xtable(pizzaDesign[order(pizzaDesign$ord),1:4])
> align(my.table) <- "|c|ccc|c|"
> print(my.table,   caption.placement = "top",
    NA.string="____")
```

The complete content of the file (e.g., `foo.Rnw`) is

```

\documentclass[a4paper]{article}
\usepackage[OT1]{fontenc}
\usepackage{Sweave}
```

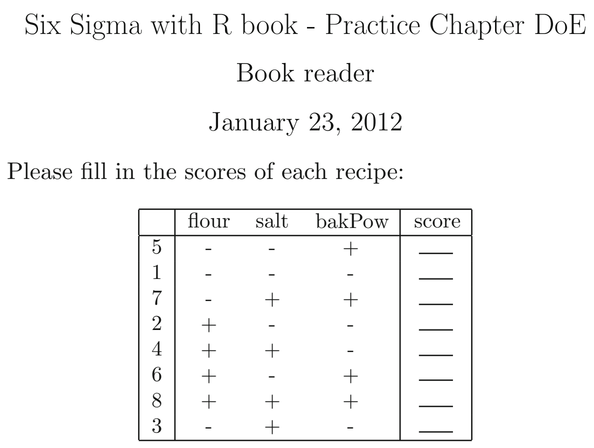
[2]http://ess.r-project.org/

[3]www.walware.de/goto/statet

[4]http://rstudio.org/

[5]Visit http://www.statistik.lmu.de/~leisch/Sweave/ to learn more about Sweave.

[6]Visit http://www.latex-project.org/ to learn more about LaTeX.



Six Sigma with R book - Practice Chapter DoE

Book reader

January 23, 2012

Please fill in the scores of each recipe:

	flour	salt	bakPow	score
5	-	-	+	____
1	-	-	-	____
7	-	+	+	____
2	+	-	-	____
4	+	+	-	____
6	+	-	+	____
8	+	+	+	____
3	-	+	-	____

Fig. 13.2 PDF file with design matrix. The document was generated merging text and R code using the `Sweave` function

```
\begin{document}

\title{Six Sigma with R book - Practice Chapter DoE}
\author{Book reader}

\maketitle

Please fill in the scores of each recipe:

 <<results = tex>>=
        require(xtable)
        my.table <- xtable(pizzaDesign[order(pizzaDesign$ord)
            , 1:4])
        align(my.table) <- "|c|ccc|c|"
        print(my.table, caption.placement = "top",
                NA.string = "____")
 @


\end{document}
```

Save the file to the work directory. Now you can generate the pdf file with the following commands:

```
> Sweave("foo.Rnw")
> require(tools)
> texi2dvi("foo.tex", pdf = TRUE)
```

The first command creates a LaTeX file, whereas the second command converts the LaTeX file into the pdf file in Fig. 13.2. You can include any plot, table, or output provided by the R code, along with any tests, figures, tables, pictures, etc. within the LaTeX code.

All the vignettes in your R installation are created using this technique. You can find a list of examples by typing `browseVignettes()`. The corresponding `.Rnw` files are in the doc folder of the package installation directory.

13.7 Summary and Further Reading

A tool set for DFSS and Lean can be found in [46]. Reference [23] is a good reference for DFSS focused on services. The book [71] can be consulted for a Six Sigma approach to Lean.

For mastering R programming we recommend the book [16], after reviewing the R language definition manual at CRAN.[7] The `knitr`, [111], and `pgfSweave` packages extend the Sweave functionality. `odfWeave` allows for merging of open document format documents with R code instead of LaTeX.

Case Study

Think about how you can apply these tools to the paper helicopter project. Determine how you can run the tools with R.

[7]http://cran.r-project.org/doc/manuals/R-lang.pdf

Appendix A
R Basic Reference Guide

This appendix is intended to provide a brief but broad collection of functions commonly used in R. See references and Sect. 2.8 in Chap. 2 to study R in depth. Type `help(foo)` in the R Console to see the documentation and the complete list of arguments for the function `foo`.

Remember that R is case sensitive, and `a` is not equal to `A`.

Table A.1 Help functions

Function	Action
`help()` or ?	Opens help about a topic
`help.search()` or ??	Returns list of topics containing some text
`apropos()`	Returns list of functions containing some text
`find()`	Tells what package something is in
`find()`	Shows which libraries and dataframes are attached
`example()`	Runs example for a topic
`demo()`	Runs demo about a topic
`args()`	Returns syntax of a function
`attach()`	Makes variables in a dataframe accessible by name
`detach()`	For a dataframe, stops accessibility by name; for a package, unloads it

Table A.2 Functions for packages and workspace

Function	Action
`library()`	Loads package or shows available package
`save.image()`	Saves all objects of R session
`save()`	Saves list of objects to file
`load()`	Loads objects from file system to R session
`rm()`	Removes objects
`source()`	Runs script in file
`sink()`	Sends output to text file instead of R Console
`savehistory()`	Saves command history of session
`loadhistory()`	Loads previously saved history
`q()`	Exits R

Table A.3 Constants and special values

Expression	Value
`letters`	26 lowercase letters
`LETTERS`	26 uppercase letters
`month.abb`	Three-letter abbreviation of months
`month.name`	English names of months
`pi`	π constant
`NA`	Not available
`NaN`	Not a number (e.g. $\frac{0}{0}$)
`Inf`	Infinite
`NULL`	No value
`1i`	Imaginary constant $\sqrt{-1}$

Table A.4 Operators for objects

Operator	Operation
`<-`	Assigns value
`[ ]`	Subscript vector
`[[ ]]`	Subscript list
`:`	Generates sequences
~	Model formula
`$`	List indexing

Table A.5 Functions for data types

Create	Verify	Convert	Description
`character()`	`is.character()`	`as.character()`	String type
`numeric()`	`is.numeric()`	`as.numeric()`	Numeric type
`logical()`	`is.logical()`	`as.logical()`	Logical data
`vector()`	`is.vector()`	`as.vector()`	One-dimensional data
`matrix()`	`is.matrix()`	`as.matrix()`	Two-dimensional data
`array()`	`is.array()`	`as.array()`	Any dimensional data
`factor()`	`is.factor()`	`as.factor()`	Categorical variable in a vector
`list()`	`is.list()`	`as.list()`	Collection of objects of different types
`data.frame()`	`is.data.frame()`	`as.data.frame()`	Data set as a list of vectors
`integer()`	`is.integer()`	`as.integer()`	Integers
`ts()`	`is.ts()`	`as.ts()`	Time-series objects
`complex()`	`is.complex()`	`as.complex()`	Complex vector (real and imaginary parts)
	`class()`		Returns the class of any object

Table A.6 Functions for importing and exporting data

Function	Action
`read.table()`	Imports text files
`read.csv(), read.csv2()`	Imports csv files
`read.delim()`	Imports delimited text files
`read.spss()`	Imports SPSS files
`read.mtp()`	Imports Minitab files
`read.arff()`	Imports Weka files (Data mining)
`read.dta()`	Imports Stata binary files
`write.table()`	Writes a text file
`write.csv()`	Writes a csv file
`writeClipboard()`	Saves data in Clipboard to paste in other applications

Table A.7 Functions for managing data

Function	Action
`c()`	Concatenates elements
`seq()`	Generates regular sequences
`rep()`	Replicates values
`length()`	Gets or sets length of an object
`sort()`	Sorts vectors
`rev()`	Reverses vectors
`order()`	Returns permutation of vector indices
`rank()`	Returns position of each vector element
`unique()`	Returns unique values of a vector
`duplicated()`	Returns indices that contain duplicated values
`which()`	Gives TRUE indices of logical object
`levels()`	Returns levels of a factor
`unlist()`	Converts a list into a vector
`cut()`	Converts a vector into a factor
`transform()`	Transforms a dataframe
`aggregate()`	Gets subtotals of variables in data set
`subset()`	Gets subset of data set

Table A.8 String functions

Function	Action
`nchar()`	Counts number of characters in a string
`substr()`	Returns a substring
`paste()`	Builds a string with substrings

Table A.9 Functions for exploring data

Function	Action
`data()`	Loads a data set
`head()`	Returns first rows of a data set
`tail()`	Returns last rows of a data set
`str()`	Returns structure of data set
`attributes()`	Accesses object attributes

Table A.10 Operators

Operator	Meaning
`+, -, *, /`	Arithmetic operators
`%/%`	Integer division
`%%`	Remainder of a division (modulo)
`%*%`	Matrix multiplication
`^`	Powers
`>, >=, <, <=, ==, !=`	Relational operators
`!, &, |`	Logical operators

Table A.11 Set operations

Function	Action
`union()`	Set union
`intersection()`	Set intersection
`setdiff()`	Set difference
`setequal()`	Set equality
`is.element()`	Set membership

Table A.12 Mathematical functions

Function	Meaning
`log(), exp()`	Logarithmic and exponential functions
`sqrt()`	Square root
`sin(), cos(), tan(), asin(), acos(), atan()`	Trigonometry functions
`cosh(), sinh(), tanh(), acosh(), asinh(), atanh()`	Hyperbolic functions
`round(), trunc(), ceiling(), floor()`	Different ways to round a number
`abs()`	Returns absolute value of a number

Table A.13 Vector functions

Function	Meaning
`max(), min()`	Maximum and minimum values in vector
`sum(), prod()`	Sum and product
`mean(), var(), sd()`	Sample mean, variance, and standard deviation
`median()`	Median value in vector
`range()`	Vector with a maximum and minimum
`cor()`	Correlation between vectors
`cumsum(), cumprod(), cummax(), cummin()`	Cumulative functions
`colMeans(), colSums(), rowMeans(), rowSums()`	Totals by rows or columns for dataframes and matrices
`ifelse()`	Returns a value for each component depending on a logical expression

Table A.14 Summarize functions

Function	Action
`apply`	Applies function to margins in matrix or vector
`lapply`	Applies function to all elements of list or vector
`tapply`	Applies function to ragged array
`sapply`	Friendly version of lapply
`summary`	Returns summary of object; for data objects, some statistics
`table`	Returns data frequency table

Table A.15 Probability distributions

Name	Probability	Random	Quantile	Density	Parameters	Default
Binomial	`pbinom()`	`rbinom()`	`qbinom`	`dbinom`	size, prob	
Normal	`pnorm()`	`rnorm()`	`qnorm()`	`dnorm()`	mean, sd	0,1
Uniform	`punif()`	`runif()`	`qunif()`	`dunif()`	min, max	0,1
Poisson	`ppois()`	`rpois()`	`qpois()`	`dpois()`	lambda	
Exponential	`pexp()`	`rexp()`	`qexp()`	`dexp()`	rate	1
t-Student	`pt()`	`rt()`	`qt()`	`dt()`	df	
Chi-square	`pchisq()`	`rchisq()`	`qchisq()`	`dchisq()`	df	
F	`pf()`	`rf()`	`qf()`	`df()`	df1, df2	
Beta	`pbeta()`	`rbeta()`	`qbeta()`	`dbeta()`	shape1, shape2	
Gamma	`pgamma()`	`rgamma()`	`qgamma()`	`dgamma()`	shape, scale	`NULL`, 1
Cauchy	`pcauchy()`	`rcauchy()`	`qcauchy`	`dcauchy`	location, scale	0,1
Weibull	`pweibull()`	`rweibull()`	`qweibull`	`dweibull`	shape, scale	`NULL`, 1
Tukey	`ptukey()`	—	`qtukey`	—	nmeans, nranges	`NULL`, 1

Table A.16 Sampling and combinatorial

Function	Action
`set.seed()`	Sets seed for replicable randomization
`sample()`	Extracts sample from a set
`choose()`	Computes binomial coefficient

Table A.17 Graphic functions

Function	Action
`plot()`	Produces different types of plots
`boxplot()`	Produces boxplot
`hist()`	Produces histogram
`coplot()`	Conditional plots
`dotchart()`	Produces dot chart
`stripchart()`	Produces 1-D scatterplot
`mosaicplot()`	Produces mosaic plot
`matplot()`	Plots matrix columns
`pairs()`	Matrix of scatterplots
`persp()`	Produces perspective plot (three dimensions)
`spineplot()`	Produces spine plot and spinogram
`text()`	Superposes text on a plot
`abline()`	Prints line in plot
`lines()`	Superposes line on a plot
`points()`	Superposes points on a plot
`polygon()`	Draws polygon
`xspline()`	Draws an X-spline
`identify()`	Identifies point in a plot with mouse

Table A.18 Plot arguments

Argument	Meaning
`type`	Type: p(points), l(lines), b(both), h(histogram), s(steps)
`main`	Main title
`sub`	Subtitle
`xlab`	Label for x axis
`ylab`	Label for y axis
`asp`	yx aspect ratio

Table A.19 Graphic parameters

Parameter	Description	Values
`col`	Color	Names, numbers, hex codes
`lwd`	Line width	Number
`lty`	Line type	Number
`pch`	Point symbol	Number
`las`	Style of axis labels	Number
`cex`	Magnifying ratio	Number (1 default)

Table A.20 Hypothesis tests and confidence intervals

Function	Task
`binom.test`	Exact test for binomial distributions
`bartlett.test`	Tests equal variances
`chisq.test`	Test for congingency tables and goodness-of-fit tests
`lillie.test`	Normality test
`pairwise.t.test`	Pairwise comparison with correction for multiple testing
`poisson.test`	Test for Poisson distribution parameter
`prop.test`	Test for proportions
`shapiro.test`	Normality test
`t.test`	Test for means
`var.test`	Test for comparing variances

Table A.21 Model fitting

Package	Function	Models
`lm`	`stats`	Linear model
`anova`	`stats`	Gets ANOVA table for a model
`manova`	`stats`	Performs multivariate ANOVA
`glm`	`stats`	Generalized linear model
`gam`	`gam`	Generalized additive model
`gam`	`mgcv`	Generalized additive model
`lm.ridge`	`MASS`	Linear models by ridge regression
`coef`	`stats`	Extracts coefficients from a model
`predict`	`stats`	Computes predictions for data input based on a model
`confint`	`stats`	Gets a confidence interval for model parameters
`residuals`	`stats`	Gets residuals of a model
`step`	`stats`	Selects variables for a model
`update`	`stats`	Updates and refits formula of a model

Table A.22 `SixSigma` package data sets

Function	Task
`ss.data.bolts`	Bolts example, Chap. 4
`ss.data.ca`	Winery example, Chap. 7
`ss.data.doe1`, `ss.data.doe2`	Pizza dough example, Chap. 11
`ss.data.pastries`	Pastries exercise, Chap. 5
`ss.data.pc`, `ss.data.pc.big`, `ss.data.pc.r`	Print cartriges example, Chap. 8
`ss.data.strings`	Guitar strings example, Chaps. 9 and 10
`ss.data.rr`	Data for a paper helicopter experiment

Table A.23 `SixSigma` package functions

Function	Task
`ss.ceDiag`	Cause-and-effect diagram
`ss.pMap`	Process map
`ss.lfa`	Loss function analysis
`ss.lfa`	Computes loss function value
`ss.rr`	Gage R&R study
`ss.study.ca`	Capability analysis study
`ss.ca.cp`	Capability index
`ss.ca.cpk`	Corrected capability index
`ss.ca.yield`	Computes yield of a process
`ss.ci`	Confidence interval for mean and normality test

Table A.24 Cited packages

Package	Use	Reference
`base`, `stats`, `utils`, `grid`, `graphics`, `tools`	Base packages	[84]
`AcceptanceSampling`	Acceptance sampling	[52]
`AMORE`	Neural network modeling	[62]
`e1071`	SVM, data mining, classification	[21]
`kernlab`	Data mining, classification, SVM	[48]
`foreign`	Data format conversion	[83]
`gam`	General additive model (inference)	[37]
`ggplot2`	Elegant graphics using grammar of graphics	[105]
`Hmisc`	Harrell miscellaneous (useful functions for many tasks by Frank E. Harrell)	[35]
`knitr`	Report generation	[111]
`lattice`	Lattice graphics	[91]
`lpridge`	Ridge regression	[95]
`MASS`	Support functions and data sets for Venables and Ripley's MASS	[102]
`mgcv`	Generalized additive and other models	[109]
`monmlp`	Neural network modeling	[14]
`neuralnet`	Neural network modeling	[27]
`nnet`	Neural network and multinomial log-linear models	[102]
`nortest`	Normality test	[33]
`odfWeave`	Reporting	[57]
`plan`	Gantt chart	[50]
`plotrix`	Gantt chart and other plots	[61]
`pls`	Partial least squares (PLS) regression	[67]
`plsdof`	Partial least squares (PLS) regression	[56]
`plspm`	Partial least squares (PLS) regression	[90]
`plsRbeta`	Partial least squares (PLS) regression	[26]
`plsRcox`	Partial least squares (PLS) regression	[6]
`plsRglm`	Partial least squares (PLS) regression	[7]
`qAnalyst`	Control chart and capability analysis	[98]
`qcc`	Quality control chart	[93]
`qualityTools`	Methods associated with DMAIC cycle	[88]
`RODBC`	Data source connection	[87]
`RWeka`	SVM and other data mining tools	[39, 108]
`SixSixgma`	Six Sigma tools and data sets for examples in this book	[15]
`sp`	Spatial data analysis	[78]
`spc`	Statistical process control computations	[54]
`spcadjust`	Calibration of control charts	
`survival`	Survival analysis (including ridge regression); reliability studies	[100]
`XLConnect`	Import/export, manipulate Microsoft Excel files	[29]
`xtable`	Elegant tables for reports and publications	[19]

References

1. Agresti, A., & Coull, B. A. (1998). Approximate is better than "exact" for interval estimation of binomial proportions. *American Statistician*, *52*, 119–126.
2. Allen, T. T. (2010). *Introduction to engineering statistics and lean Six Sigma—Statistical quality control and design of experiments and systems*. New York: Springer.
3. Arthur, J. (2006). *Lean six sigma demystified. Demystified series*. New York: McGraw-Hill.
4. Bentley, W., & Davis, P. (2009). *Lean Six Sigma secrets for the CIO: ITIL, COBIT, and beyond*. USA: CRC Press.
5. Berger, P., & Maurer, R. (2002). *Experimental design: With applications in management, engineering, and the sciences. Duxbury titles of related interest*. CA: Duxbury/Thomson Learning.
6. Bertrand, F., Maumy-Bertrand, M., & Meyer, N. (2011) *Partial least squares regression for Cox models and related techniques*. http://CRAN.R-project.org/package=plsRcox, r package version 0.2.0.
7. Bertrand, F., Meyer, N., & Maumy-Bertrand, M. (2011). *Partial least squares regression for generalized linear models*. http://CRAN.R-project.org/package=plsRglm, r package version 0.7.6.
8. Bivand, R., Pebesma, E., & Gómez-Rubio, V. (2008). *Applied spatial data analysis with R. Use R!*. New York: Springer.
9. Box G (1992). Teaching engineers experimental design with a paper helicopter. *Quality Engineering*, *4*(3), 453–459.
10. Box, G., & Jones, S. (1992). Designing products that are robust to the environment. *Total Quality Management*, *3*(3), 265–285.
11. Box, G., Hunter, J., & Hunter, W. (2005). *Statistics for experimenters: Design, innovation, and discovery. Wiley series in probability and statistics*. New York: Wiley.
12. Burdick, R., Borror, C., & Montgomery, D. (2005). *Design and analysis of gauge R&R studies: Making decisions with confidence intervals in random and mixed ANOVA models. ASA-SIAM series on statistics and applied probability*. Philadelphia: Society for Industrial Applied Mathematics.
13. Burton, T. T., & Sams, J. L. (2005). *Six Sigma for small and mid-sized organizations: Success through scaleable deployment*. Boca Rotan: J. Ross Publishing.
14. Cannon, A. J. (2011). *monmlp: Monotone multi-layer perceptron neural network*. http://CRAN.R-project.org/package=monmlp, r package version 1.1.
15. Cano, E. L., Moguerza, J. M., & Redchuk, A. (2011). *SixSigma: Six Sigma tools for quality improvement*. R package version 0.6.0.
16. Chambers, J. M. (2008). *Software for data analysis. Programming with R. Statistics and computing*. Berlin: Springer.

17. Cook, D., & Swayne, D. (2007). *Interactive and dynamic graphics for data analysis: With R and GGobi. Use R!*. New York: Springer.
18. Crawley, M. J. (2007). *The R Book*. UK: Wiley.
19. Dahl, D. B. (2009). *xtable: Export tables to LaTeX or HTML*. http://CRAN.R-project.org/package=xtable, r package version 1.5-6.
20. Dalgaard, P. (2008). *Introductory statistics with R. Statistics and computing*. New York: Springer.
21. Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D., & Weingessel, A. (2011). *e1071: Misc functions of the department of statistics (e1071), TU Wien*. http://CRAN.R-project.org/package=e1071, r package version 1.6.
22. Eckes, G. (2003). *Six sigma for everyone*. New Jersey: Wiley.
23. El-Haik, B., & Roy, D. (2005). *Service design for six sigma: A roadmap for excellence*. New York: Wiley.
24. Faraway, J. (2006). *Extending the linear model with R: Generalized linear, mixed effects and nonparametric regression models. Texts in statistical science*. Boca Raton: Chapman & Hall/CRC. http://books.google.es/books?id=ODcRsWpGji4C.
25. Faraway, J. J. (2002). *Practical regression and anova using r*. http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf. Accessed 01.08.2011.
26. Bertrand, F., & Myriam Maumy-Bertrand N. M. (2011). *plsRbeta: Partial least squares regression for beta regression models*. http://CRAN.R-project.org/package=plsRbeta, r package version 0.1.1.
27. Fritsch, S., Guenther, F., & following earlier work by Marc Suling (2010). *neuralnet: Training of neural networks*. http://CRAN.R-project.org/package=neuralnet, r package version 1.31.
28. Gandy, A., & Kvaloy, J. T. (2011). *spcadjust: Functions for calibrating control charts*. http://CRAN.R-project.org/package=spcadjust, r package version 0.1-1.
29. GmbH MS (2011). *XLConnect: Excel Connector for R*. http://CRAN.R-project.org/package=XLConnect, r package version 0.1-7.
30. Graham, B. (2004). *Detail process charting: Speaking the language of process*. New Jersey: Wiley.
31. Grömping, U. (2011). *Cran task view: Design of experiments (doe) & analysis of experimental data*. http://cran.r-project.org/web/views/ExperimentalDesign.html. Retrieved 24.01.2012.
32. Grömping, U. (2012). *Project: (industrial) doe in r*. http://prof.beuth-hochschule.de/groemping/software/design-of-experiments/project-industrial-doe-in-r/. Retrieved 24.01.2012.
33. Gross, J. (2006). *nortest: Tests for normality*. R package version 1.0.
34. Gygi, C., DeCarlo, N., & Williams, B. (2005). *Six sigma for dummies*. Hoboken: Wiley.
35. Harrell, F. E. (2011). *Hmisc: Harrell miscellaneous*. http://CRAN.R-project.org/package=Hmisc, r package version 3.9-0.
36. Harry, M., & Schroeder, R. (1999). *Six sigma: The breakthrough management strategy revolutionizing the world's top corporations*. New York: Doubleday Business.
37. Hastie, T. (2011). *gam: Generalized additive models*. http://CRAN.R-project.org/package=gam, r package version 1.06.2.
38. Hastie, T., Tibshirani, R., & Friedman, J. (2008). *The elements of statistical learning: Data mining, inference, and prediction. Springer series in statistics*. New York: Springer. http://www-stat.stanford.edu/~tibs/ElemStatLearn/.
39. Hornik, K., Buchta, C., & Zeileis, A. (2009) Open-source machine learning: R meets weka. *Computational Statistics*, *24*(2), 225–232.
40. Hsu, H. (2010). *Schaum's outline of probability, random variables, and random processes. Schaum's Outline Series* (2nd ed.). New York: McGraw-Hill.
41. Ishikawa, K. (1990). *Introduction to quality control*. Japan: 3A Corporation
42. ISO (1994). *ISO 5725-1: Accuracy (trueness and precision) of measurement methods and results—Part 1: General principles and definitions*. Geneva: International Organization for Standardization.

43. ISO (2009). *ISO/TS 16949: Quality management systems—Particular requirements for the application of ISO 9001:2008 for automotive production and relevant service part organizations*. Geneva: International Organization for Standardization.
44. ISO (2011). *Iso 13053-1:2011–Quantitative methods in process improvement—six sigma —part 1: DMAIC methodology*. Geneva: International Organization for Standardization.
45. Jakir, B. (2011). *Introduction to statistical thinking (with r, without calculus)*. http://pluto.huji.ac.il/~msby/StatThink/index.html. Accessed 01.08.2011.
46. John, A., Meran, R., Roenpage, O., Lunau, S., Staudter, C., & Schmitz, A. (2008). *Six Sigma+Lean Toolset*. Berlin: Springer.
47. Juran, J., & Defeo, J. (2010). *Juran's quality handbook: The complete guide to performance excellence*. New York: McGraw-Hill.
48. Karatzoglou, A., Smola, A., Hornik, K., & Zeileis, A. (2004). kernlab—An S4 package for kernel methods in R. *Journal of Statistical Software*, *11*(9), 1–20. http://www.jstatsoft.org/v11/i09/.
49. Keller, P., & Keller, P. (2011). *Six Sigma demystified. Demystified series*. New York: McGraw-Hill.
50. Kelley, D. (2009). *plan: Tools for project planning*. http://CRAN.R-project.org/package=plan, r package version 0.3-1.
51. Kerns, G. J. (2010). *Introduction to probability and statistics using r*. http://ipsur.r-forge.r-project.org/book/index.html. Accessed 01.08.2011.
52. Kiermeier, A. (2008). Visualizing and assessing acceptance sampling plans: The R package AcceptanceSampling. *Journal of Statistical Software*, *26*(6), 1–20. http://www.jstatsoft.org/v26/i06/.
53. Knight, E., Russell, M., Sawalka, D., & Yendell, S. (2007). *Taguchi quality loss function and specification tolerance design*. https://controls.engin.umich.edu/wiki/index.php/Taguchi_quality_loss_function_and_specification_tolerance_design in Michigan chemical process dynamics and controls open textbook. Accessed 20.09.2011.
54. Knoth, S. (2011). *spc: Statistical process control*. http://CRAN.R-project.org/package=spc, r package version 0.4.1.
55. Koch, R. (2005). *Living the 80/20 way: Work less, worry less, succeed more, enjoy more*. London: Nicholas Brealey Publishing.
56. Kraemer, N., & Braun, M. L. (2011). *plsdof: Degrees of freedom and statistical inference for partial least squares regression*. http://CRAN.R-project.org/package=plsdof, r package version 0.2-2.
57. Kuhn, M. (2011). *odfWeave: Sweave processing of open document format (ODF) files*. http://CRAN.R-project.org/package=odfWeave, r package version 0.7.17.
58. Lalanne, C. (2006). *R companion to montgomery's design and analysis of experiments*. http://www.aliquote.org/articles/tech/dae/. Retrieved 19.01.2012.
59. Larsson, A. (2008). *Dia software*. http://live.gnome.org/Dia.
60. Leisch, F. (2002). Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle & B. Rönz (Eds.), *Compstat 2002—Proceedings In Computational Statistics* (pp. 575–580). Heidelberg: Physica Verlag. http://www.stat.uni-muenchen.de/~leisch/Sweave, ISBN 3-7908-1517-9.
61. Lemon, J. (2006). Plotrix: A package in the red light district of r. *R-News*, *6*(4), 8–12.
62. Limas, M. C., Meré, J. B. O., Marcos, A. G., de Pisón Ascacibar, F. J. M., Espinoza, A. V. P., & Elías, F. A. (2010). *AMORE: A MORE flexible neural network package*. http://CRAN.R-project.org/package=AMORE, r package version 0.2-12.
63. Lopez-Fidalgo, J. (2009). A critical overview on optimal experimental designs. *Boletin de Estadística e Investigación Operativa*, *25*(1), 14–21. http://www.seio.es/BEIO/files/BEIOv25n1_ES_J.Lopez-Fidalgo.pdf. Retrieved 19.01.2012.
64. Maindonald, J. H. (2008). *Using r for data analysis and graphics. Introduction, code and commentary*. http://cran.r-project.org/doc/contrib/usingR.pdf. Accessed 01.08.2011

65. Martin, J. (2007). *Operational excellence: Using lean six sigma to translate customer value through global supply chains. Series on resource management*. PA: Auerbach Publications.
66. Mee, R. (2009). *A comprehensive guide to factorial two-level experimentation*. New York: Springer.
67. Mevik, B. H., Wehrens, R., & Liland, K. H. (2011). *pls: Partial least squares and principal component regression*. http://CRAN.R-project.org/package=pls, r package version 2.3-0.
68. Moguerza, J. M., Muñoz, A. (2006). Support vector machines with applications. *Statistical Science*, *21*(3), 322–336.
69. Montgomery, D. (2005). *Introduction to statistical quality control* (6th ed.). New York: Wiley.
70. Montgomery, D. (2008). *Design and analysis of experiments. Student solutions manual*. New York: Wiley.
71. Muir, A. (2005). *Lean Six sigma statistics: Calculating process efficiencies in transactional projects. Six sigma operational methods series*. New York: McGraw-Hill.
72. Murrell, P. (2009). Drawing Diagrams with R. *The R Journal*, *1*(1), 15–21. http://journal.r-project.org/archive/2009-1/RJournal_2009-1_Murrell.pdf.
73. Murrell, P. (2009). *Introduction to data technologies. Series in computer science and data analysis*. Boca Raton: CRC Press. http://www.stat.auckland.ac.nz/~paul/ItDT/.
74. Murrell, P. (2011). *R graphics* (2nd ed.). Boca Raton: Chapman & Hall/CRC.
75. Myers, R., Montgomery, D., & Anderson-Cook, C. (2009). *Response surface methodology: Process and product optimization using designed experiments. Wiley series in probability and statistics*. New York: Wiley.
76. Pande, P., Neuman, R., & Cavanagh, R. (2000). *The Six Sigma way: How GE, Motorola, and other top companies are honing their performance*. New York: McGraw-Hill.
77. Pearn, W., & Kotz, S. (2006). *Encyclopedia and handbook of process capability indices: A comprehensive exposition of quality control measures. Series on quality, reliability & engineering statistics*. New York: World Scientific.
78. Pebesma, E., & Bivand, R. (2005). Classes and methods for spatial data in r. *R News*, *5*(2), 9–13.
79. Poincaré, H. (1914). Science and method. London: Nelson. Originally published in French as Science et Méthode in 1908.
80. Popper, K. (1959). *The logic of scientific discovery*. London: Hutchinson. Originally published in German as Logik der Forschung in 1934.
81. Prieto, M. (2005). 6 sigma Qué es y cómo aplicarlo a la empresa española. Asociacion Española para la Calidad.
82. Pyzdek, T., & Keller, P. (2009). *The Six Sigma handbook: A complete guide for green belts, black belts, and managers at all levels*. New York: McGraw-Hill.
83. R-core (2011). *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase, ...*. http://CRAN.R-project.org/package=foreign, r package version 0.8-48.
84. R Development Core Team (2011). *R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing*, Vienna, Austria. http://www.R-project.org/, ISBN 3-900051-07-0.
85. Rasch, D., Pilz, J., & Simecek, P. (2010). *Optimal experimental design with R*. London: Taylor & Francis.
86. Recchia, D. R., Barbosa, E. P., & de Jesus Goncalves, E. (2010). *IQCC: Improved quality control charts*. http://CRAN.R-project.org/package=IQCC, r package version 0.5.
87. Ripley, B. (2012). *RODBC: ODBC Database Access*. R package version 1.3-4.
88. Roth, T. (2011). *qualityTools: Statistics in quality science*. R package version 1.50.
89. Roy, R. (2010). *A primer on the Taguchi Method* (2nd ed.). Michigan: Society of Manufacturing Engineers.
90. Sanchez, G., & Trinchera, L. (2012). *plspm: Partial least squares data analysis methods*. http://CRAN.R-project.org/package=plspm, r package version 0.2-2.
91. Sarkar, D. (2008). *Lattice: Multivariate data visualization with R*. New York: Springer. http://lmdvr.r-forge.r-project.org, ISBN 978-0-387-75968-5.

92. Schonberger, R. (2008). *Best practices in lean six sigma process improvement: A deeper look*. Safari Books Online. Chichester: Wiley.
93. Scrucca, L. (2004). qcc: An r package for quality control charting and statistical process control. *R News*, *4*(1), 11–17, http://CRAN.R-project.org/doc/Rnews/.
94. Searle, S., Casella, G., & McCulloch, C. (2006). Variance components. Wiley series in probability and statistics. New York: Wiley.
95. Seifert, B. (2011). lpridge: Local polynomial (ridge) regression. http://CRAN.R-project.org/package=lpridge, r package version 1.0-6; Packaged for R by Martin Maechler.
96. Shaffer, L., Young, T., Guess, F., Bensmail, H., & León, R. (2008). Using r software for reliability data analysis. *International Journal of Reliability and Application*, *91*(1), 53–70.
97. Snee, R. D. (2004). Six-sigma: The evolution of 100 years of business improvement methodology. *International Journal of Six Sigma and Competitive Advantage*, *1*(1), 4–20.
98. Spano', A. (2011). *qAnalyst: Control charts, capability and distribution identification*. http://CRAN.R-project.org/package=qAnalyst, r package version 0.6.4.
99. Taguchi, G., Chowdhury, S., & Wu, Y. (2005). *Taguchi's quality engineering handbook*. USA: Wiley.
100. Therneau, T. (2011). *survival: Survival analysis, including penalised likelihood*. http://CRAN.R-project.org/package=survival, r package version 2.36-10. Original Splus->R port by Thomas Lumley.
101. Vardeman, S. B., & VanValkenburg, E. S. (1999). Two-way random-effects analyses and gauge r&r studies. *Technometrics*, *41*(3), 202–211.
102. Venables, W., & Ripley, B. (2002). *Modern applied statistics with S. Statistics and computing*. New York: Springer.
103. Vikneswaran (2005). *An r companion to "experimental design"*. http://cran.r-project.org/doc/contrib/Vikneswaran-ED_companion.pdf. Retrieved 19.01.2012.
104. Wahlbrink, S. (2012). *Statet: Eclipse based ide for r*. http://www.walware.de/goto/statet. Retrieved 16.01.2012.
105. Wickham, H. (2009). *ggplot2: Elegant graphics for data analysis. Use R!*. New York: Springer.
106. Wilkinson, L., & Wills, G. (2005). *The grammar of graphics. Statistics and computing*. New York: Springer.
107. Wilson, G. (2005). Six sigma and the product development cycle. Butterworth-Heinemann: Elsevier.
108. Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.
109. Wood, S. (2011). Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society*, *73*(1), 3–36.
110. Workgroup, M. (2010). *MSA manual* (4th ed.). Automotive Industry Action Group (AIAG).
111. Xie, Y. (2012). *knitr: A general-purpose package for dynamic report generation in R*. http://CRAN.R-project.org/package=knitr, r package version 0.1.
112. Zuur, A., Ieno, E., & Meesters, E. (2009). *A beginner's guide to R. Use R!*. New York: Springer.

Solutions

Practice for Chap. 2

2.1

```
> install.packages("SixSigma", dependencies = TRUE)
```

Select a mirror close to your location when prompted.

2.2

```
> FailureTime <- c(0.29, 0.32, 1.21, 0.95, 0.14, 2, 0.81,
    0.88)
> Temp <- c(63.89, 63.38, 65.05, 62.31, 68.04, 59.12, 62.80,
    61.89)
> Factory <- c("A", "B", "C", "C", "B", "B", "A", "B")
> sol1a <- data.frame(FailureTime,
         Temp, Factory = as.factor(Factory))
> summary(sol1a)
> write.csv(sol1a, file = "sol1a.csv")
```

2.3

```
> plot(sol1a$Temp, sol1a$FailureTime)
> boxplot(sol1a$FailureTime ~ sol1a$Factory)
> hist(sol1a$Temp)
```

2.4

```
> table(sol1a$Factory)
> sol3 <- sol1a$Temp[sol1a$FailureTime < 1]
```

Practice for Chap. 3

3.1

```
> grid.roundrect(width = .25,
    height = unit(1.8, "inches"),
    x = 0.25)
> grid.text("INPUTS\n\nData\nComputer\nOperator",
    x = 0.25,
    y = 0.66,
    just = "top")
> grid.roundrect(width = .25,
    height = unit(1.8, "inches"),
    x = 0.75)
> grid.text("OUTPUTS (Y)\n\nInvoice\n(pages,\nsize,\ncolor)",
    x = 0.75,
    y = 0.66,
    just = "top")
> grid.lines(x = c(0.375, 0.625),
    y = c(0.5, 0.5),
    arrow = arrow())
```

3.2

```
>   inputs <-c ("Clothes", "Machine", "Powder")
>   outputs <- c("dryness", "cleanness", "time", "creases")
>   steps <- c("PREPARE", "WASH", "HANG OUT")
>   io <- list()
>   io[[1]] <- list("Clothes", "Machine", "Powder")
>   io[[2]] <- list("Clothes", "MachineState")
>   io[[3]] <- list("dryness", "cleanness", "time")
>   param <- list()
>   param[[1]] <- list(c("type", "P"), c("amount", "N"))
>   param[[2]] <- list(c("time", "Cr"), c("Powder.Brand", "Cr
    "),
                    c("weight", "P"), c("MachineAge", "Cr"))
>   param[[3]] <- list(c("WindSpeed", "N"), c("temperature",
    "N"),
            c("location", "C"))
>   feat <- list()
>   feat[[1]] <- list("cleanness")
>   feat[[2]] <- list("cleanness", "time", "dryness")
>   feat[[3]] <- list("cleanness", "time", "creases")
>   ss.pMap(steps, inputs, outputs,
         io, param, feat,
         sub = "Laundry Process")
```

Practice for Chap. 4

4.1

$L(Y) = 175(Y - 15)^2$

4.2

Formula:

$$L(Y) = 0.125 \cdot (Y - 750)^2$$

Graphic of function:

```
> curve(0.125*(x-750)^2,
    from = 735, to = 765)
```

4.3 Loss function analysis:

```
> ss.lfa(ss.data.ca, "Volume", 10, 750, 1.25, 850)
```

The average cost is \$0.53 and the total loss is \$452.9.

Practice for Chap. 5

5.1

The variable `lab` (laboratory) is the appraiser factor (three levels), and the variable `batch` is the part factor, also with three levels. Two measures of each batch were taken in each laboratory.

5.2

The command to run the MSA is as follows:

```
> ss.rr(var = comp, part = batch, appr = lab,
        data = ss.data.pastries)
```

This is a good measurement system. The number of distinct categories is greater than 4 (6) and the %Study Var for R&R is lower than 30%. However, this value (20.27%) is not lower than 10%, and the measurement system may be improved. The plots confirm these results.

Practice for Chap. 6

6.1

First we create the data from the example:

```
> b.effect <- "Delay"
> b.groups <- c("Personnel", "Weather", "Suppliers", "
    Planning")
```

```
> b.causes <- vector(mode="list", length=length(b.groups))
> b.causes[1] <- list(c("Training", "Inadequate"))
> b.causes[2] <- list(c("Rain", "Temperature", "Wind"))
> b.causes[3] <- list(c("Materials", "Delays", "Rework"))
> b.causes[4] <- list(c("Customer", "Permissions", "Errors"))
>
```

We can plot a cause-and-effect diagram:

```
> ss.ceDiag(b.effect, b.groups, b.causes, sub="Construction
    Example")
> b.data <- data.frame(cause = factor(unlist(b.causes)),
    count = c(5, 1, 3, 1, 2, 18, 20, 4, 15, 2, 4),
    cost = c(50, 150, 50, 10, 20, 180, 200, 10, 5, 20, 150))
```

Next we plot the Pareto chart:

```
> require(qcc)
> b.vector <- b.data$cost
> names(b.vector) <- b.data$cause
> pareto.chart(b.vector, cumperc = c(80))
```

6.2 We need the two charts in order to choose the adequate causes to focus on.

If we only use the Pareto chart for the count of errors, then we will select "Customer" as one of the "vital few." However, in the cost chart, it is apparent that delays due to customer needs do not generate a high cost. It is due to the fact that a new specification usually involves a revision of the contract and, therefore, an extended deadline without extra costs.

Using the cost chart, the factors to focus on are the delay in the receipt of materials, quality of materials, inadequateness of the personnel, and planning errors.

Practice for Chap. 7

7.1

With $\sigma = 2$:

$$Z = \min\left\{\frac{(14-10)}{2}, \frac{(10-4)}{2}\right\} = \min 2, 3 = 2,$$

$$Z_{LT} = Z_{ST} - 1.5 = 2 \times 1.5 = 0.5.$$

With $\sigma = 1$:

$$Z = \min\left\{\frac{(14-10)}{1}, \frac{(10-4)}{1}\right\} = \min 4, 6 = 4,$$

$$Z_{LT} = Z_{ST} - 1.5 = 4 \times 1.5 = 2.5,$$

7.2 Now we have less than 308,000 DPMO (previously we had more than 690,000 DPMO).

Practice for Chap. 8

8.1

```
> newData <- aggregate(pc.volume ~ pc.filler,
    sum, data = ss.data.pc)
> barplot(newData$pc.volume,
    names.arg = newData$pc.filler,
    main = "Total volume by filler")
```

8.2

```
> newData <- ss.data.pc.big[ss.data.pc.big$batch == 1,5]
> plot(newData, type = "b")
```

There seems to be a descendant trend over time.

Practice for Chap. 9

9.1 To save the subset:

```
> data.stats.prob1 <- ss.data.strings[ss.data.strings$type ==
     "E1",]
```

```
> summary(data.stats.prob1$res)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    5.75    6.50    6.20    8.00    9.00
```

```
> sd(data.stats.prob1$res)
```

```
[1] 2.067289
```

```
> IQR(data.stats.prob1$res)
```

```
[1] 2.25
```

The mean and median are not equal, so the distribution might not be symmetric. We can generate a boxplot to see the data from the summary.

```
> boxplot(data.stats.prob1$res)
```

9.2

```
> pbinom(4, 100, 0.01)
```

```
[1] 0.9965677
```

Practice for Chap. 10

10.1

Using the `shapiro.test` function:

```
> shapiro.test(ss.data.strings$res)
```

Using the `ss.ci` function from the `SixSigma` package:

```
> ss.ci(res, data = ss.data.strings)
```

These data are nonnormal. We need other methods for inference.

10.2

```
> prob2.model <- lm(res ~ len, data = ss.data.strings)
> summary(prob2.model)
```

The linear model is not good for these data. The p-value for the goodness-of-fit hypothesis test is very high (0.7149). Furthermore, the R-squared statistic is very low.

10.3 Solution using `lm` function:

```
> prob3.model1 <- lm(len ~ type, data = ss.data.strings)
> summary(prob3.model1)
```

Solution using `aov` function:

```
> prob3.model2 <- aov(len ~ type, data = ss.data.strings)
> summary(prob3.model2)
```

The length of the strings is independent of the type of string (p-value > 0.05). Plotting the effects:

```
> qplot(type, len, data = ss.data.strings) +
    stat_summary(fun.y = mean, geom = "line",
        aes(group = 1), col = "orangered") +
    stat_summary(fun.y = mean, geom = "point",
        shape = 17, size = 3, col = "red") +
    opts(title = "Effects of factor Type of string on Length"
        )
```

Practice for Chap. 11

11.1

```
> myDesign <- expand.grid(factor1 = gl(2, 1, labels = c("-",
    "+")),
        factor2 = gl(2, 1, labels = c("-", "+")),
        factor3 = gl(2, 1, labels = c("-", "+")),
        factor4 = gl(2, 1, labels = c("-", "+")),
        response = NA)
```

```
> myDesign$ord <- sample(1:16, 16)
> myDesign[order(myDesign$ord),]
```

11.2

The formula in the following code allows one to check up to three-way interactions.

```
> model.prob1 <- lm(score ~ (.-repl)^3  , data = ss.data.doe2
    )
> summary(model.prob1)
```

There are some important interactions; therefore, we keep them in the model jointly with the significant main effects.

```
> selectionvar <- step(model.prob1, method="backwards")
> summary(selectionvar)
> coef(selectionvar)
```

Practice for Chap. 12

12.1

```
> pb.groups.s <- with(ss.data.pb2,
    qcc.groups(pb.humidity, pb.group))
> pb.s <- qcc(pb.groups.s, type = "S")
> summary(pb.s)
> plot(pb.s)
```

12.2

As the number of errors is registered by surface unit, the sample size for each group is 1, and we will have straight control limit lines. The *u* chart is plotted with the following command:

```
> with(ss.data.pb4,
        plot(qcc(defects, sizes = rep(1, 80), type = "u")))
```

R Packages and Functions Used in the Book

Subject Index