

RICE UNIVERSITY

Tools and theory to improve data analysis

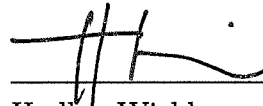
by

Garrett Grolemond

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:



Hadley Wickham, *Chair*
Assistant Professor of Statistics



David M. Lane
Associate Professor of Psychology, Statistics,
and Management



David W. Scott
Noah Harding Professor of Statistics

HOUSTON, TEXAS
DECEMBER 2012

ABSTRACT

Tools and theory to improve data analysis

by

Garrett Grolemund

This thesis proposes a scientific model to explain the data analysis process. I argue that data analysis is primarily a procedure to build understanding and as such, it dovetails with the cognitive processes of the human mind. Data analysis tasks closely resemble the cognitive process known as sensemaking. I demonstrate how data analysis is a sensemaking task adapted to use quantitative data. This identification highlights a universal structure within data analysis activities and provides a foundation for a theory of data analysis. The model identifies two competing challenges within data analysis: the need to make sense of information that we cannot know and the need to make sense of information that we cannot attend to. Classical statistics provides solutions to the first challenge, but has little to say about the second. However, managing attention is the primary obstacle when analyzing big data. I introduce three tools for managing attention during data analysis. Each tool is built upon a different method for managing attention. `ggsubplot` creates embedded plots, which transform data into a format that can be easily processed by the human mind. `lubridate` helps users automate sensemaking outside of the mind by improving the way computers handle date-time data. **Visual Inference Tools** develop expertise in young statisticians that

can later be used to efficiently direct attention. The insights of this thesis are especially helpful for consultants, applied statisticians, and teachers of data analysis.

ACKNOWLEDGEMENTS

First and foremost, I'd like to thank my thesis advisor, Hadley Wickham. I may not have pursued this important area of research if not for Hadley's encouragement, advice, and example of success pioneering similar topics. He has been an indispensable source of wisdom and support, and has invested unceasingly in my skills as a researcher and a teacher. Hadley's many accomplishments would seem even more remarkable if people knew how much time he freely lavishes on novice students like myself.

I'd also like to thank the faculty and students of Rice University's Department of Statistics. The faculty have made statistics a real and exciting wonderland to be explored. I feel privileged to have attended so many thought provoking lectures and to have done so in the company of so many friendly and talented students. Together the students and faculty of Rice have provided me a world class education. I'd especially like to thank Jaime Ramos, Stephanie Hicks, David Kahle, Eric Chi, and Terrance Savitsky for their support and their enthusiasm for discussing the ideas contained in this dissertation.

Finally, I'd like to thank my partner, Kristin Kitchen. Kristin took care of me while I took care of problem sets and journal articles. Without her, I would not today be so fed, healthy, and sane. Kristin, you have made the past five years a pleasure that I would repeat with you again and again.

Dedicated to my parents, Eric and Bonnie Grolemond

Contents

Abstract	ii
Acknowledgements	iv
1 Introduction	1
2 A Cognitive Interpretation of Data Analysis	7
2.1 Introduction	8
2.2 A theory of data analysis	9
2.3 The role of cognition in data analysis	13
2.4 Making sense of measured data	21
2.5 A conceptual model of data analysis	26
2.6 Implications for data analysis practice	32
2.7 Conclusion	39
3 Visualizing complex data with embedded plots	43
3.1 Introduction	44
3.2 Case Study: Analyzing complex data	48
3.3 Benefits of embedded plots	52
3.4 Implementing embedded plots with the grammar of graphics	59
3.5 Conclusion	73
4 Dates and times made easy with lubridate	76
4.1 Introduction	76
4.2 Motivation	78

4.3	Parsing date-times	81
4.4	Manipulating date-times	82
4.5	Arithmetic with date-times	85
4.6	Rounding dates	95
4.7	Time zones	96
4.8	Daylight savings time	98
4.9	Case study 1	99
4.10	Case study 2	102
4.11	Conclusion	108
5	How and why to teach statistical inference with simulations in R	110
5.1	Introduction	111
5.2	Background	113
5.3	Why teach with visual simulations	115
5.4	Why program visual simulations in R	124
5.5	How to implement visual simulations in R	127
5.6	How to use simulations in the classroom	137
5.7	Conclusion	144
6	Conclusion	147
6.1	Original contributions	150
6.2	Future Work	152
6.3	Final thoughts	155
	References	156

List of Figures

2.1	A simplified summary of the sensemaking process. Schemas are compared to observed information. If any discrepancies (i.e, insights) are noticed, the schema is updated or the information is disregarded as untrustworthy.	16
2.2	Exploratory and confirmatory data analysis both follow the sensemaking process. Exploratory analysis begins with a received data set. A confirmatory analysis begins with a received schema (often a hypothesis or model).	18
2.3	Data analysis parallels sensemaking. Analysts deduce a precise hypothesis (model) from the schema, which they compare to the data or a transformation of the data. Analysts must attempt to distinguish discrepancies between schema and data from differences that result from variance and bias. Analysts must also match each accepted model back to a schema to provide interpretation in real world concepts.	27
2.4	The seven flights examined by NASA and Morton Thoikol managers (above). The 24 flights for which information was available (below). Recreated from Presidential Commission (1986) (p. 146).	38

-
- 3.1 Three examples of graphs that use embedded subplots. **A.** (*upper left*) A subcycle plot of CO_2 measurements taken on Mauna Lau, Hawaii between 1959 and 1990. Recreated from Cleveland (1994), page 187. Observations are grouped by month. **B.** (*upper right*) A glyphmap of temperature fluctuations in the western hemisphere over a six year period. Each glyph is a polar chart with $r = temperature$ and $\theta = date$ these charts are organized on a cartesian plane with $x = longitude$ and $y = latitude$. **C.** (*lower left*) A binned plot of the diamonds data set from the `ggplot2` software package. Subplots are used to show patterns in diamond colors without overplotting. When this data is presented in its raw form, the accumulation of points hides patterns in the data (*lower right*). 45
- 3.2 **A.** (*upper left*) Relative rates of casualties by area in Afghanistan between 2004 and 2010. Raw data can not be visualized due to overplotting. **B.** (*upper right*) A heat map shows casualty counts, but not relative rates by group. **C.** (*lower left*) Embedded bar charts reveal that there have been more civilian than combatant casualties around Kabul, the capital of Afghanistan. Marginal bar charts reveal similar information as a heat map, but also display rates by group. **D.** (*lower right*) Conditional bar charts make regional rates the more obvious; they show that inordinate civilian casualties is not unique to the capital city. 50
- 3.3 Casualty frequencies between 2004 and 2010 by region. The embedded graphics show that the heaviest fighting has been confined to the southern and eastern regions of Afghanistan. The most casualties have occurred around Kandahar. Many regions seem peaceful since 2008. However, casualties have increased recently throughout south-east Afghanistan. 52
- 3.4 Users can control the amount of summarization that occurs in an embedded plot. When scatterplots are used for subplots, no summarization occurs (*left*). Line graphs provide partial summarization (*center*). Heat maps provide complete summarization, within each bin data is reduced to a single number (*right*). This may not always be desirable. 53

3.5	Cleveland's subcycle plot can be decomposed into twelve subplots arranged as a scatterplot. The subplots behave as a rectangle geom with an internal drawing aesthetic.	62
3.6	Every individual geom is a self contained plot when paired with a set of axes. Such plots may be not be very interesting, as is the case with point geoms.	63
3.7	<code>ply_aes</code> offers a new strategy for overplotting. Groups of geoms are combined into single geoms that display summary information. This approach reveals that <i>mean(ozone)</i> has a different linear relationship with temperature in the southern hemisphere than it does in the north.	67
3.8	The position of a subplot is often related to which points the subplot shows. <code>Position = merge</code> and <code>geom_subplot2d</code> provide two ways to avoid overlapping subplots without disrupting this relationship.	69
3.9	Reference objects allow comparison across subplots and can be more easily read at small scales than coordinate axes. In <code>ggsubplot</code> , users can add one of three types of reference objects to subplots by adding <code>ref = ref_box()</code> , <code>ref = ref_hline()</code> , or <code>ref = ref_vline()</code> to <code>geom_subplot</code> and <code>geom_subplot2d</code> calls.	70
4.1	Home games and away games appear to occur in clusters.	103
4.2	More games occur on Tuesdays than any other day of the week.	104
4.3	The graph on the left displays seconds on the x axis. The graph on the right uses a more intuitive division with minutes.	106
4.4	Wait times between shot attempts rarely lasted more than 30 seconds.	107
4.5	The lead changed between the Lakers and Celtics numerous times during the game.	108
5.1	Statistical inference simulations are often organized around three fields: a field that displays the population data (top), a field that displays the sample data (middle), and a field that displays the sample statistic data (bottom).	129
5.2	The three field format can be used to illustrate inference related concepts such as the Central Limit Theorem (left), confidence intervals (middle), and the variance of regression lines regression lines (right)	129

-
- 5.3 The user can interact with VIT through the GUI provided in the panel on the left. This panel is managed by the “gui environment” described in our software blueprint. The panel on the right provides a canvas to animate the simulation created by VIT. This panel is managed by the “canvas environment.” 131
- 5.4 A general inference simulation tool can be built in R by organizing information into a GUI environment that manages the user interface and a canvas environment that displays the visual simulation. The canvas only needs to handle five actions, the details of these can be supplied at run time. To extend the program, a user only needs to supply additional methods for these five tasks. 136
- 5.5 Previous sample means remain visible as “ghosts” in the middle field. These transparent blue bars are distinguishable from the current sample mean, which is larger and solid. Transparency also helps students see where multiple sample means have accumulated. 141
- 6.1 A cognitive view of data analysis can help organize a statistics curriculum. Data analysis uses sensemaking to develop mental models that accurately described observed data. Analysts can either simplify data in a way that allows mental sensemaking to occur (left hand side). Or they can transform schemas into hypotheses and attempt automated sensemaking, with a statistical or computerized technique. Statistical methods can be grouped by how they facilitate this process. 154

List of Tables

4.1	<code>lubridate</code> provides a simple way to parse a date into R, extract the month value and change it to February.	79
4.2	<code>lubridate</code> easily displays a date one day earlier and in the GMT time zone.	79
4.3	<code>lubridate</code> provides a simple alternative for many date and time related operations. Table adapted from Grothendieck and Petzoldt (2004).	80
4.4	Parse function names are based on the order that years, months, and days appear within the dates to be parsed.	82
4.5	Each date-time element can be extracted with its own accessor function.	84
4.6	Adding two date-time objects will create the above type of object.	95

Introduction

This thesis proposes a new way to look at data analysis: data analysis is a cognitive task shaped by the structure of the human mind. I examine support for this view, discuss how this view can be used to improve the practice of data analysis, and introduce three new data analysis tools that are informed by a cognitive view of data analysis. The insights of this thesis are especially helpful for consultants, applied statisticians, and teachers of data analysis.

Data analysis is the investigatory process used to extract knowledge, information, and insights about reality by examining data. Although I submit this thesis as partial requirement for fulfilling a PhD in Statistics, data analysis is not synonymous with statistics. The act of data analysis begins before a statistical technique is chosen and continues after statistical calculations have yielded results. Yet data analysis is not entirely separate from statistics either. Analyzing data relies on statistical techniques. Data analysis depends on both the results of statistical methods and on the theory that lets these results be interpreted, such as probability theory and statistical inference. Statistics, in turn, depends on data analysis. Statistical methods are meant to analyze data. The needs of data analysis determine which statistical techniques are used in practice and which statistical theories are developed in academia. Data

analysis has been called applied statistics and statistical engineering. I refrain from using these terms for the simple reason that data analysis also relies on techniques from other fields, such as computer science and philosophy. I also believe that data analysis relies heavily on the field of cognitive science, although this connection has largely escaped notice.

This thesis examines the connection between data analysis and cognitive science. I contend that data analysis is a cognitive task, known elsewhere as sensemaking. I marshal evidence to support this view and discuss how a cognitive interpretation can be used to improve the practice of data analysis, which is also the practice of statistics. The implications of this view also reach beyond practice. They extend to how statistics is taught and to how data analysis tools are devised. This is a new approach to data analysis. Data analysis has not been studied as a cognitive task, and much of the cognitive science research that underpins this dissertation has only been conducted in the last decade.

This thesis is organized into four self-contained articles, which are in various stages of publication.

Chapter 2, *A cognitive interpretation of data analysis*, proposes that data analysis is guided by the mind's attempts to encode quantitative data into mental representations of knowledge. I evaluate this proposition with three lines of argument. First, I assume as a "first principle" that to create understanding for a human audience, data analysis must interact with the processes that create understanding in the human mind. I argue from this principle that data analysis is a cognitive task, known as sensemaking, that has been adapted to quantitative data. Second, I compare the adapted sensemaking task to descriptions of data analysis that have been published by expert data analysts. The sensemaking model aligns with these descriptions and explains why the descriptions take the specific form that they do. Third, I demon-

strate that the sensemaking model can usefully critique failures in data analysis that are not explained by statistical theory. These failures have a cognitive origin.

Chapter 2 proposes a theory based model of data analysis and discusses shortcomings in the previous theories and conceptions of data analysis. The article has been accepted for publication in *International Statistical Review* and will appear in early 2013 along with discussion of the article by noted statisticians.

The insights of Chapter 2 present data analysis in a new light: data analysis is a fairly simple task made complicated by two constraints. One is epistemological, one is cognitive. Epistemologically, we can not know all of the information relevant to an analysis. Cognitively, we can not attend to all of the information that we do have at the same time. Classical statistics has long recognized the first constraint. Statisticians have developed many tools, like Probability theory, that help circumvent the limits of what we can know. Chapter 2 discusses how epistemological constraints affect the data analysis process.

In contrast, the cognitive constraint has gone unrecognized and under-appreciated by data analysts. Data analysis attempts to find and understand the relationships between data points. At the mental level, the human mind can not recognize relationships that involve the interaction of many points until it simultaneously considers each point involved (Sweller, 2010). However, studies suggest that the average person can only hold about four pieces of new information, such as data points, in their attention¹ at once (Cowan, 2000). The implications are unambiguous: to find complex relationships in data, data analysts need to invoke strategies for managing their limited attentional resources. These strategies will gain increasing importance as the profession of statistics embraces big data, the area of analysis where attentional limits

¹Within the introduction, I use the word attention to also refer to the related idea of working memory capacity. I do this because attention is more easily understood by a general audience. This follows the example of Cowan (2000). I discuss the limits of attention and the working memory in more detail in Chapters 2, 3, and 5.

are the most strained.

Cognitive scientists have identified a number of mechanisms that the mind uses to manage attention and learn complex material. These mechanisms have become the basis for education practices that help students master difficult content.¹ I contend that these mechanisms can form a basis for data analysis tools that improve an analyst's ability to discover insights in large, complex data. I support this contention and further verify the cognitive model of data analysis by introducing three new tools that improve the practice of data analysis, Chapters 3–5. Each tool builds upon a separate mechanism for managing attention during data analysis. These mechanisms include preprocessing data to aid mental processing, avoiding mental processing by delegating sensemaking tasks to computers, and building expertise that can more efficiently direct attention management.

Chapter 3, *Visualizing complex data with embedded plots*, demonstrates how attention can be more efficiently utilized by preprocessing and transforming information. The chapter introduces embedded plots, a class of data visualizations, which organize a collection of graphs into a larger graphic. This arrangement allows users to visualize multi-dimensional relationships in 2D graphs. Embedded plots preprocess large amounts of information in a way that can easily be attended to by the human mind. They reduce the cognitive load associated with reading a plot by relying on visualization, isolation, and automation.

From a practical perspective, embedded plots provide many benefits. They offer new ways to avoid overplotting, to control the amount of information lost when summarizing data, and to visualize interactions. Chapter 3 also refines our understanding of the grammar of graphics by showing that graphs are hierarchical, or recursive, in nature. The grammar is built around the concept of a geom, a visual element used

¹See Mayer (2009) and Sweller et al. (2011) for overviews.

to represent data. Embedded plots reveal that every geom is a graph and every graph can be a geom. The chapter also demonstrates new ways of understanding and utilizing other parts of the grammar, such as stats and position adjustments. *Visualizing complex data with embedded plots* is built upon the `ggsubplot` package, which creates embedded subplots in R. `ggsubplot` was written by Garrett Grolemund and will be made available from <http://cran.r-project.org>. The article *Visualizing complex data with embedded plots* has been submitted for publication to the *Journal of Computational and Graphical Statistics*.

Chapter 4, *Dates and times made easy with lubridate*, provides a way to help data analysts externalize cognition when working with large data. External cognition is introduced in Chapter 2 and provides a way to circumvent attention constraints by having computers process information for the mind. However, computer manipulations often break down with date-time data. Date-times follow idiosyncratic rules and must be treated differently than other quantities. Yet date-times come in a variety of formats, which means computers cannot easily recognize them. Human analysts must often semi-manually parse date-times into a computer program. When computers manipulate dates, the results are often untrustworthy due to capricious date-time rules such as leap years and daylight savings time, which requires further user inspection. As a result date-times create an information processing bottleneck that reduces the benefits of using a computer to help with sensemaking.

Chapter 4 introduces a software package, `lubridate`, and a set of mental models that make computing with date-times easier. `lubridate` simplifies a variety of date-time tasks, such as parsing dates and creates a series of commands that allow users to specify how dates should be manipulated. These commands are associated with precise date-time concepts that let users and computers share common expectations for how date-time rules should be handled. `lubridate` was written by Garrett Grolemund

and Hadley Wickham and is available from <http://cran.r-project.org>. *Dates and times made easy with lubridate* was published in 2011 by the *Journal of Statistical Software* (Grolemund and Wickham, 2011).

Chapter 5 addresses building expertise, which is a third way to manage attention. Experts are able to focus their attention on pieces of information that are more likely to be suggestive, important, and helpful for understanding. Chapter 5, *How and why to teach statistical inference with simulations in R*, introduces a software program built by Garrett Grolemund, **Visual Inference Tools (VIT)**, that can be used to teach statistical inference. The chapter reviews cognitive science literature to argue that visual simulations can improve student understanding of statistical inference. It evaluates R as a language for building such tools and provides a software design that can be used to make a general purpose simulation tool in R. *How and why to teach statistical inference with simulations in R* concludes by providing four principles to guide the presentation of instructional simulations.

Chapter 5 is part of a larger research program being conducted by the Government of New Zealand and a team of researchers partly based out of the University of Auckland. I developed the infrastructure of VIT as a tool to be used in this program. VIT is now being maintained and extended by researchers at the University of Auckland and is being tested within New Zealand classrooms for incorporation into the New Zealand K-13 mathematics curriculum. *How and why to teach statistical inference with simulations in R* will be adapted to form the basis of a paper that describes the results of these tests. It will be co-authored by Garrett Grolemund, Chris Wild, Maxine Pfannkuch, and Ross Parsonage.

A Cognitive Interpretation of Data Analysis

This chapter will appear in the International Statistical Review in 2013 as ‘A cognitive interpretation of data analysis’ by Garrett Grolmund and Hadley Wickham. The article will be accompanied by discussion about the article from invited statisticians.

ABSTRACT — This paper proposes a scientific model to explain the data analysis process. We argue that data analysis is primarily a procedure to build understanding and as such, it dovetails with the cognitive processes of the human mind. Data analysis tasks closely resemble the cognitive process known as sensemaking. We demonstrate how data analysis is a sensemaking task adapted to use quantitative data. This identification highlights a universal structure within data analysis activities and provides a foundation for a theory of data analysis. The competing tensions of cognitive compatibility and scientific rigor create a series of problems that characterize the data analysis process. These problems form a useful organizing model for the data analysis task while allowing methods to remain flexible and situation dependent. The insights of this model are especially helpful for consultants, applied statisticians, and teachers of data analysis.

2.1 Introduction

This paper proposes a scientific model to explain the data analysis process, which attempts to create understanding from data. Data analysis tasks closely resemble the cognitive process known as sensemaking. We demonstrate how data analysis is a sensemaking task adapted to use quantitative data. This identification highlights a universal structure within data analysis activities and provides a foundation for a theory of data analysis. The proposed view extends existing models of data analysis, particularly those that describe data analysis as a sequential process (Tukey, 1962; Tukey and Wilk, 1966; Box, 1976; Wild, 1994; Chatfield, 1995; Wild and Pfannkuch, 1999; Cook and Swayne, 2007). The paper follows the suggestion of Mallows and Walley (1980) to build on insights from psychology and the examples of Lakoff and Núñez (1997) and Lakoff and Núñez (2000), who documented the influence of cognitive mechanisms on mathematics. The paper was motivated by the authors' need to find criteria on which to compare and optimize the usefulness of data analysis tools; however, the paper's discussion is relevant to all users of data analysis techniques, such as consultants, applied statisticians, and teachers of data analysis.

The paper is organized as follows. Section 2.2 defines data analysis and explains the shortcomings of the current treatment of data analysis in statistics. Section 2.3 examines the relationship between cognitive science and data analysis. It outlines areas of cognitive science research that are relevant to the data analysis process, such as mental representations of knowledge, the sensemaking process, and the use of external cognitive tools to complete sensemaking tasks. Section 2.4 identifies how the use of precise, measured data disrupts the sensemaking process. It then describes the adaptations to general sensemaking that measured data require. Section 2.5 proposes that data analysis is a sensemaking task adapted to the use of measured data. This provides a theoretic model of data analysis that explains existing descriptions of the

data analysis process. In Section 2.6 we examine a prediction of this model: data analysis inherits the known shortcomings of sensemaking. We examine two of these shortcomings with case studies of well known data analyses. These shortcomings include the tendency to retain false schemas and the inability of sensemaking to prove its conclusions. We conclude by discussing the usefulness of the cognitive model of data analysis as a guiding theory for data analysis.

2.2 A theory of data analysis

Data analysis is the investigative process used to extract knowledge, information, and insights about reality by examining data. Common data analysis activities include specifying a hypothesis, collecting data relevant to a problem, modelling data with quantitative methods, and interpreting quantitative findings. This process relies on statistics, a field with useful methods for specific data analysis tasks, but has an applied focus; data analysts focus less on the properties of a method and more on the connections between the data, the method, its results, and reality. Data analysis is sometimes referred to as “applied statistics” (Mallows, 1998) or the “wider view” of statistics (Wild, 1994), but we prefer the term data analysis because it does not suggest that statistics is the only tool to be applied when analyzing data.

Data analysis is a widely used technique that is relevant to many fields. This relevance has increased sharply in the past decades as data has become more ubiquitous, more complex, and more voluminous. Large data sets, such as online customer review ratings, social network connections, and mappings of the human genome, promise rewarding insights but overwhelm past methods of analysis. The result is a “data deluge” (Hey and Trefethen, 2003) where current data sets can far exceed scientists’ capacity to understand them. Despite this difficulty, the rewards of understanding data are so promising that data analysis has been labelled the sexiest field of the next

decade (Varian, 2009).

Future advancements in data analysis will be welcomed by the scientific community, but progress may be limited by the currently sparse theoretical foundations. Little theory exists to explain the mechanisms of data analysis. By theory, we mean a conceptual model that synthesizes relevant information, makes predictions, and provides a framework for understanding data analysis. This definition is more pragmatic than formal: a useful theory of data analysis would help analysts understand what data analysis is, what its goals are, how it achieves these goals, and why it fails when it falls short. It should go beyond description to explain how the different parts of a data analysis, such as experimental design, visualization, hypothesis testing, and computing relate to each other. Finally, a theory of data analysis should allow analysts to predict the success or failure of possible data analysis methods.

It is hard to prove such a theory does not exist, but Unwin (2001) points out that there are few texts and little theory to guide a data analysis. Similar concerns have been expressed by Mallows and Walley (1980), Breiman (1985), Wild (1994), Huber (1997), Velleman (1997), Mallows (1998), Wild and Pfannkuch (1999), Viertl (2002), Mallows (2006), Cobb (2007), Huber (2011) and in the discussion of Breiman (2001). Huber (1997) identifies one reason for the lack of data analysis theory: techniques are developed by researchers who work with data in many different fields. Often knowledge of the technique remains localized to that field. As a result, data analysis ideas have been balkanized across the fields of statistics, computer science, economics, psychology, chemistry, and other fields that proceed by collecting and interpreting data. The subject matter of data analysis is also hard to generalize. The methods of each analysis must be flexible enough to address the situation in which it is applied. This malleability resists a top-down description and led Unwin (2001) to suggest a bottom-up pattern language to stand in for data analysis theory.

A well defined theory of data analysis would provide many benefits. First, it would facilitate the development of better techniques. In many fields, advancements accrue through the extension and development of theories (Unwin, 2001). Advancements in data analysis techniques may lead to many potential rewards. The areas of applications for data analysis have developed more in recent decades than they have during any previous period in the history of statistics (Breiman, 2001). Despite this, many statistics courses still teach methods typical of the first half of the 20th century, an era characterized by smaller data sets and no computers (Cobb, 2007). The development of theory could hasten the speed with which data fields adapt to emerging challenges. A theory of data analysis may also curtail the development of bad techniques. Technology and large data sets do not guarantee useful results. Freedman (2009) argues that “many new techniques constitute not progress but regress” because they rely on technical sophistication instead of realistic assumptions. A better understanding of data analysis will help ground future innovations to sound practice.

A theory of data analysis will also improve the education of future analysts. Statistics curricula have been criticized for teaching data analysis techniques without teaching how or why statisticians should use them (Velleman, 1997). This undermines students’ attempts to learn. As Wild and Pfannkuch (1999) explain “the cornerstone of teaching in any area is the development of a theoretical structure with which to make sense of experience, to learn from it and to transfer insights to others.” The lack of data analysis theory means that little structure exists with which to teach statistical thinking. As a result, some graduates from statistics programs have been poorly trained for their profession; they know the technical details of statistical methods but must undertake an on-the-job apprenticeship to learn how to apply them (Breiman, 1985; Mallows, 1998). The focus on technique also fails non-statisticians, who are the primary consumers of introductory statistics courses. Without a grasp of sta-

tistical thinking, non-statisticians are less likely to recognize the need for a trained statistician and therefore less likely to hire one (Wild, 1994).

A theory of data analysis may also benefit the field of statistics by providing unity and direction. At the end of his 1997 assessment of statistics, Huber predicted that statistics would dissolve as a field unless statisticians replaced their focus on techniques with a focus on “meta-methods” and “meta-statistics” (Huber, 1997). Three years later in 2000, a panel on data analysis called for statistics to evolve into a data science organized by a general theory of data analysis (Viertl, 2002). These conclusions echo Tukey’s argument that statistics should be “defined in terms of a set of problems (as are most fields) rather than a set of tools, namely those problems that pertain to data” (Friedman (1998) summarizing Tukey (1962)). A theory of data analysis would offer a unifying theme for statistics and its applications. It would also offer a common language that would promote collaboration by analysts in various fields.

Finally, a theory of data analysis would improve data analysis practice. A theory would aid practitioners because theoretical concerns guide practice (Gelman and Shalizi, 2010). Theory also improves practice; people problem solve more successfully when they “have suitably structured frameworks” to draw upon (Pea, 1987; Resnick, 1988).

Where should we look for such a theory? Many published papers involve a data analysis. But as Mallows and Walley (1980), Cox (2001), and Mallows (2006) point out, most studies do not provide a detailed description of the analysis involved. Instead, they focus on results and implications. We could narrow our focus to statistics and computer science; both fields develop tools to analyze data. However, statistics articles usually focus on the mathematical properties of individual techniques, while computer science articles focus on algorithmic efficiency. As a result, little research

deals explicitly with the data analysis process. We propose an alternative source for data analysis insights: cognitive science.

2.3 The role of cognition in data analysis

Cognitive science offers a way to understand data analysis at a theoretic level. Concerns of cognitive science may seem far from the field of statistics, but they have precedent in the early literature of exploratory data analysis. Tukey and Wilk (1966) highlight the role of cognitive processes in their initial descriptions of EDA (emphasis added): “The basic general intent of data analysis is simply stated: to seek through a body of data for interesting relationships and information and to exhibit the results *in such a way as to make them recognizable to the data analyzer*” (emphasis added). And again, “...at all stages of data analysis the nature and detail of output, both actual and potential, *need to be matched to the capabilities of the people who use and want it*” (emphasis added.) Cognitive concerns also appear in recommendations for improving data analysis. Tukey (1962) suggested that “those who try may even find new [data analysis] techniques evolving ... from studies of the nature of ‘intuitive generalization.’” Mallows and Walley (1980) list psychology as one of four areas likely to support a theory of data analysis.

Cognitive science also addresses a commonality of all data analyses. Data analyses rely on the mind’s ability to learn, analyze, and understand. Each analysis attempts to educate an observer about some aspect of reality. Usually, this requires data to be manipulated and preprocessed, but the end result of these efforts must be a knowledge product that can be interpreted by the human mind. An analysis cannot be useful if it fails to provide this. Even “black box” analyses, which may rely on methods that are incomprehensible to the analyst, must produce a result that the analyst can assign meaning to. If they do not, they will not be useful. This last step of assigning

meaning is not a statistical or computational step, but a cognitive one. In this way, each data analysis is part of a larger, cognitive task. The success of each data analysis depends on its ability to interact with this cognitive process.

This alone is good reason for data analysts to learn about cognition. However, cognitive processes also shed insights on the preprocessing stages of a data analysis; mental processes closely parallel the preprocessing stages of data analyses. Moreover, untrained analysts can and do “analyze” data with only their natural mental abilities. The mind performs its own data analysis-like process to create detailed understandings of reality from bits of sensory input. In this section, we examine these mental processes. In Sections 2.4 and 2.5 we argue that data analysis is a specific extension of a mental process known as sensemaking.

2.3.1 Schemas and sensemaking

Studies suggest that the average person can only hold two to six pieces of information in their attention at once (Cowan, 2000). Yet people are able to use this finite power to develop detailed understandings of reality, which is infinitely complex. The mind builds this understanding in a process that is similar to many descriptions of data analysis. The mind creates and manages internal cognitive structures that represent aspects of external reality. These structures consists of mental models and their relationships (Rumelhart and Ortony, 1976; Carley and Palmquist, 1992; Jonassen and Henning, 1996). Mental models have been studied under a number of different names. Examples include frames (Goffman, 1974; Minsky, 1975; Rudolph, 2003; Smith et al., 1986; Klein et al., 2003), scripts (Schank and Abelson, 1977), prototypes (Rosch and Mervis, 1975; Rosch, 1977; Smith, 1978) and schemas (Bartlett, 1932; Neisser, 1976; Piaget and Cook, 1952). A schema is a mental model that contains a breadth of information about a specific type of object or concept. Schemas are organized into

semantic networks based on their relationships to other schemas (Wertheimer, 1938; Rumelhart and Ortony, 1976). This arrangement helps the brain process its experiences: instead of storing every sensory observation, the brain only needs to maintain its schemas, which are sufficient summaries of all previous observations. Some “memories” may even be complete recreations built with a schema (Bartlett, 1932; Klein et al., 2003). Once the brain associates an event with a schema, it can use the schema to access unobserved information related to the event. The mind uses this information to assign meaning to sensory inputs and predict the relationships between data points (Klein et al., 2003). In this way, the mind uses schemas and semantic networks to construct our perception of reality from limited sensory input (Neisser, 1967).

People maintain their schemas in a process known as *sensemaking*. Russell et al. (1993); Klein et al. (2003); Pirolli and Card (2005) and Zhang (2010) have each proposed a description of the sensemaking process. These models vary in their details, but they all contain the same basic components, shown in Figure 2.1. Variations of this basic model have been utilized by scientists in the fields of cognitive science (Lundberg, 2000; Klein et al., 2003; Helsdingen and Van den Bosch, 2009); organizational studies (Weick, 1995; Weick et al., 2005); computer science (Attfield and Blandford, 2009; Russell et al., 1993); knowledge management (Dervin, 1998); intelligence analysis (Pirolli and Card, 2005); InfoVis (Yi et al., 2008); and Visual Analytics (Wu et al., 2010).

The sensemaking process revolves around noticing discrepancies between schemas and reality. To understand an event, the brain selects a relevant schema. This selection may be guided by context clues or a few initial observations that serve as anchor points (Klein and Crandall, 1995). The brain then uses this schema to scan the environment for relevant sensory inputs. The schema helps the brain build information from the inputs by assigning meaning to them. This is similar to Moore’s

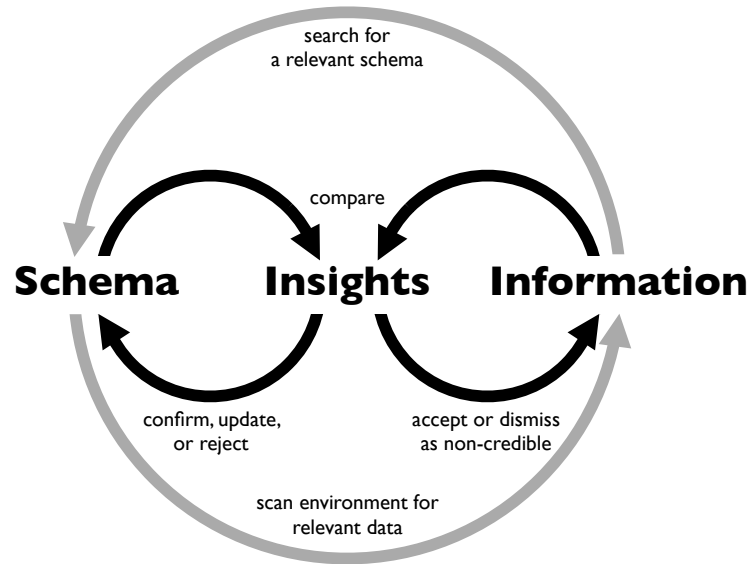


Figure 2.1: A simplified summary of the sensemaking process. Schemas are compared to observed information. If any discrepancies (i.e, insights) are noticed, the schema is updated or the information is disregarded as untrustworthy.

definition of data as numbers *that have been given a context* (Moore, 1990). As new information is constructed, the brain tries to fit it into the schema. If a piece of relevant information does not fit, the schema may be flawed. The sensemaking literature calls these unique, non-fitting pieces of information *insights* (Pirolli and Card, 2005). If new information contains no insights, the brain retains the schema as it is. If insights are present, the brain either updates the schema to account for them, dismisses the information as non-credible, or abandons the schema entirely. In the last outcome, the insights and information would then guide the selection of a new schema. This process repeats itself whenever further information becomes available.

Data analysis is a sensemaking task. It has the same goals as sensemaking: to create reliable ideas of reality from observed data. It is performed by the same agents: human beings equipped with the cognitive mechanisms of the human mind. It uses the same methods. Experts in data analysis such as John Tukey and George Box have offered descriptions of the data analysis process. These descriptions show that data

analysis proceeds like sensemaking by comparing theory to fact, searching for discrepancies, and modifying theory accordingly. According to Box (1976), “matters of fact can lead to a tentative theory. Deductions from this tentative theory may be found to be discrepant with certain known or specially acquired facts. These discrepancies can then induce a modified, or in some cases, a different, theory. Deductions made from the modified theory now may or may not be in conflict with fact and so on.” Tukey’s view of data analysis also stresses comparison, discrepancy, and iteration: “Data analysis is a process of first summarizing [the data] according to the hypothesized model [theory] and then exposing what remains [discrepancies], in a cogent way, as a basis for judging the model or the precision of this summary, or both” (Tukey and Wilk, 1966). Both Tukey and Box also emphasize the iterative nature of data analysis and the importance of successive approximations of the truth.

Sensemaking also explains both exploratory data analysis and confirmatory data analysis. Many researchers separate data analysis tasks into exploratory and confirmatory parts (for example, Mulaik (1985), Chatfield (1995)). As Mulaik (1985) explains, “exploratory statistics are usually applied to observational data collected without well-defined hypotheses for the purpose of generating hypotheses. Confirmatory statistics, on the other hand, are concerned with testing hypotheses.” In other words, confirmatory analyses focus on a hypothesis (the schema) and seek to validate the schema against data. Exploratory analyses focus on the data and seek to find schemas that explain the data. Many sensemaking descriptions begin with a schema and then proceed to collecting data as in a confirmatory analysis. However, sensemaking can also begin with data and then seek a plausible schema as in exploratory analysis. Qu and Furnas (2008) demonstrates the data to schema direction to sensemaking. In pilot studies of information search tools, Qu and Furnas found that people use sensemaking to develop schemas that explain available data. Early definitions of

sensemaking also reflect its bi-directional nature. For example, Russell et al. (1993) define sensemaking as “a process of searching for a representation and encoding data in that representation to answer task-specific questions.” To summarize, sensemaking is an integrated, iterative process with multiple points of entry. Exploratory data analysis follows a sensemaking loop that begins with data. Confirmatory data analysis follows a sensemaking loop that begins with a schema (in the form of a hypothesis), Figure 2.2.

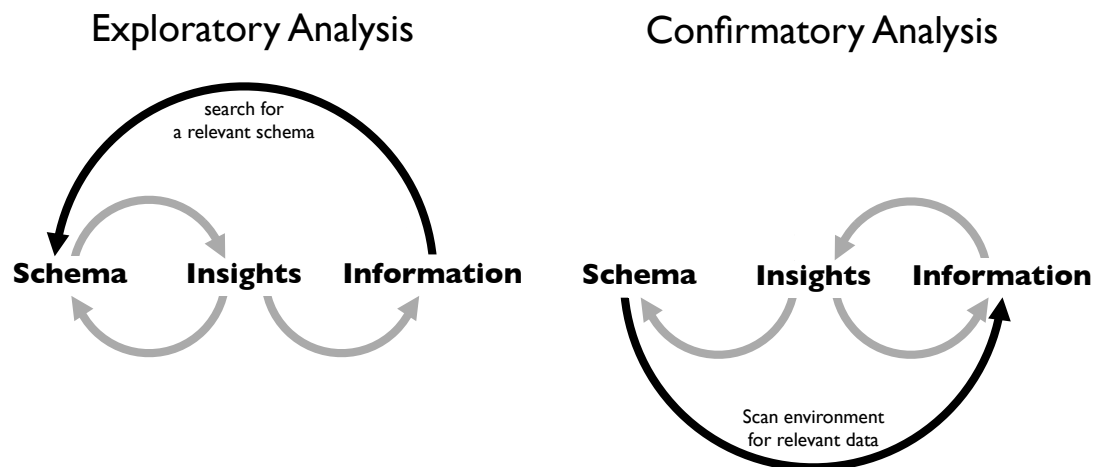


Figure 2.2: Exploratory and confirmatory data analysis both follow the sensemaking process. Exploratory analysis begins with a received data set. A confirmatory analysis begins with a received schema (often a hypothesis or model).

While the general structure of data analysis aligns with sensemaking, its results differ. The results of unguided sensemaking are too unreliable to meet the goals of science. Science requires objective results that can be recreated under consistent conditions. Sensemaking creates subjective results that can vary from person to person and from time to time. It is common experience that different people come to different conclusions when presented with the same information. This subjectivity occurs because people have and use different sets of schemas when analyzing information. Unguided sensemaking also has other flaws that increase subjectivity.

Tversky and Kahneman (1974) showed that people express predictable biases when they try to make sense of uncertain information. Tversky and Kahneman (1981) showed that meaningless changes in the way information is presented can result in complete changes in the conclusions people draw. These are only two of the most well known biases in human thinking, many more exist. Fortunately, sensemaking can be augmented in ways that reduce these biases and foster objective results. Data analysis is shaped by these augmentations.

2.3.2 External tools of cognition

We can augment sensemaking with external methods of cognition. The human mind has evolved to rely on external, artificial tools to aid thought (Donald, 1991). These external tools allow us to perform cognitive feats that would not be possible otherwise. For example, a child may use a paper and pencil to perform math that they could not do in their head, or an adult may rely on a written list when grocery shopping (Zhang, 2000). External cognition tools can also be used to reduce the subjectivity of sensemaking. Data analysis relies on two external tools: data, which is an external representation of knowledge; and logic, particularly mathematics, which is an external system for processing information.

External representations of knowledge are information that is stored in the environment. This information can be stored as physical symbols (e.g, written numbers), as relations embedded in physical configurations (e.g, the beads of an abacus or lines on a map), as systems of rules and constraints (e.g, the laws of algebra), or in other ways (Zhang, 1997). External representations play an important role in many cognitive tasks. They can extend a person's working memory, permanently archive large amounts of data, make abstract information directly accessible, suggest solutions by reducing the number of options, and change the nature of a cognitive task

to something easier (Zhang, 2000). Well chosen external representations can even provide access to knowledge and skills unavailable from internal representations. For example, the invention of arabic numerals enabled the development of algebra, something that was not possible with roman numerals or purely internal representations of counts. External representations of knowledge guide a sensemaker's attention and give schemas and observations a form that can be shared among sensemakers.

Data analysis relies heavily upon an external representation of knowledge: measured and recorded data. Data provides many benefits that reduce the subjectivity of sensemaking. Recorded data allows large quantities of information to be stored outside of the memory. Here it can be quickly and easily accessed to support cognition. Recorded data can also be manipulated outside of the working memory (e.g. with computers) and shared with other sensemakers. Data is usually collected in a prescribed manner, which reduces the role that schemas play in attending to and interpreting observations. Measurement also allows data to be defined with more consistency and precision than the human senses can supply. Finally, precise measurement facilitates the use of other external cognitive tools such as math and logic.

Systems of rules and constraints can also be external cognitive tools. These systems automate the extraction and transformation of knowledge. As a result, information can be processed outside of the working memory. This allows more data to be processed at once, more complex operations to be performed, and fewer errors to occur during processing. Data analysis relies heavily on math and logic, which are external systems of information processing. Logic and math reduce the subjectivity of data analysis by mandating which conclusions can be drawn from which facts. As Norman (1993) summarizes, "logic is reliable: provide the same information and it will always reach the same conclusion." This is not true of unguided sensemaking. Logic also allows sensemakers to work with entire data sets instead of just the col-

lection of data points they can mentally attend to. As mentioned at the start of this section, the working memory seems to only be able to hold two to six pieces of information at once Cowan (2000). Although, the mind uses various strategies to augment this ability (see for example, Sweller et al. (1998)), the average modern data set exceeds the capacity limits of the working memory. Finally, math and logic allow us to perform our reasoning externally, where we can examine it for errors and biases.

Data analysis can be distinguished from general sensemaking by its reliance on measured data and math and logic. Data and logic reduce the subjectivity of sensemaking. The use of these external cognitive tools makes sensemaking more fit for science, which prefers objective results. Unmodified, our internal knowledge building processes are too subjective to provide these results. Data, in particular, resists the internal forces that create subjectivity. Data reduces the tendency of schemas to screen out observations. Data expands our storage and processing powers. Data can be manipulated and examined externally, which allows us to police our reasoning during sensemaking. But using data introduces new problems: how do we compare abstract schemas to specific, often quantitative, data? How do we identify discrepancies between schema and data when data contains its own type of variation?

2.4 Making sense of measured data

The sensemaking process must be adapted to accommodate measured data. First, schemas must be made precise to allow comparison against precisely measured data. Schemas must be made quantitative to be easily compared against quantitative data. Second, a test must be developed to identify discrepancies between schema and data in the presence of variation. If data analysis is a sensemaking process, as we propose, each instance of data analysis will exhibit these accommodations. We discuss these accommodations below.

2.4.1 Abstract schema, quantified data

Sensemaking proceeds by identifying discrepancies between schemas and reality. These two objects must have similar forms to allow accurate comparison. However, schemas do not usually resemble measured data. A typical schema may be as simple as an idea that can be expressed in a sentence or as well developed as what Kuhn (1962) calls a paradigm, a world view that not only contains a theory, but also defines what questions are acceptable, what assumptions are permissible, what phenomena deserve attention, and more. How should an analyst compare schemas against data? The common solution is to deduce a prediction from a schema that can be tested against the data. The predictions can be simple or complex, but they must take the same precise or quantitative form as the data. A linear regression model of the form $Y = \alpha + \beta X + \epsilon$ is one example of a quantified prediction deduced from a schema. A set of data simulated from $Y = \alpha + \beta X + \epsilon$ would be a further prediction from the schema. The underlying schema includes additional non-quantitative information, such as model assumptions, contextual information, and any other beliefs about the subject matter, data sources, and their relationships. The direction of causal relationships and the assumption that there are no lurking variables are two examples of information contained in a schema but not the quantitative hypothesis deduced from the schema.

Data analysis proceeds by testing these quantitative predictions against data in the usual sensemaking fashion. We should not confuse these predictions with the actual underlying schema. They are only deductions that must be true if the schema is true. The validation of a prediction does not validate the underlying schema because the same prediction may also be associated with other competing schemas. This ambiguity is most clear in exploratory data analyses. Exploratory analyses begin with data and then attempt to fit a model to the data. Often more than one model

can be fit, which presents one layer of ambiguity. Then the analyst must grapple with a second layer of ambiguity: which explanation of reality (i.e., schema) does the fitted model support? If smoking is correlated with lung cancer, does this suggest that smoking causes lung cancer (schema 1), that lung cancer causes smoking (schema 2), or that a third variable causes both (schema 3)? Analysts can reduce ambiguity by using multiple lines of argument, collecting more data, iterating between confirmatory and exploratory analyses, and deducing and testing as many predictions as can be had from each schema.

Transforming schemas is not the only way to facilitate comparison. Often it is also useful to transform the data to resemble a schema or model. Schemas parallel the way humans think, which rarely involves sets of measured numbers. More often a schema will only describe a characteristic of the data, such as the mean, maximum, or variance. In other occasions, a schema may focus on a variable that must be derived from the data, such as a rate (*count/time*) or density (*mass/volume*). Mathematical calculations can transform the data into the appropriate quantity prior to comparison. Exploratory analysis can be made simpler by transforming data to resemble familiar situations. For example, “curved” scatterplots can be unbent with a log transformation to resemble linear scatterplots. This aids schema search: humans have more schemas to explain familiar situations than they do to explain unfamiliar ones. It also facilitates comparison: humans are better at perceiving differences on a linear scale than a curved one. Visualization is another way to transform data that allows analysts to use their strongest perceptual abilities.

In summary, human cognitive processes are unaccustomed to sets of measured data. To use such data, a sensemaker must transform his or her schemas to resemble data. This can be done by deducing precise predictions from the schema (such as the models commonly used by statisticians). Often it can be helpful to transform the data

as well. The need to navigate between schema and prediction/model characterizes all data analyses and distinguishes data analyses from general sensemaking.

2.4.2 Omnipresent variation

Variation creates a second distinction between general sensemaking and data analysis. Variation in quantitative data is an omnipresent and demonstrable reality (Wild and Pfannkuch, 1999). In usual sensemaking tasks, this variation goes unnoticed. Observers assign observations to general categories (Rosch and Mervis, 1975). Variation is only noticed when it is large enough to place an observation in an unexpected category. Measurement, however, reveals even small variations. These variations disrupt the sensemaking process. A model will appear discrepant with data if it does not account for all of the sources of variation that affect the data. This is not a failure of sensemaking. Afterall, a schema can not be a very accurate model of reality if it does not account for variation that exists in the real world. However, it is unlikely that any model used in data analysis will describe all of the relevant sources of variation. Cognitive, computational, and financial constraints will intervene before every associated variable can be identified and measured. Moreover, many sources of variation will have little to do with the purpose of the analysis. To summarize, the omnipresence of variation in quantitative data derails the sensemaking process. Discrepancy ceases to be an informative signal; unobserved sources of variation will create minute discrepancies between predictions and observations even if a schema correctly describes the relationships between observed variables.

Data analysis proceeds by examining schemas and models that predict a *pattern* of outcomes. This pattern can then be compared against the pattern of the data. Models that predict a pattern do not need to be very complex. Probability theory provides a concise, expressive, and mathematical toolbox for describing patterns. A

deterministic model can be transformed into a model that predicts a pattern by adding a probability term. This term acts as a “catch all” that describes the combined effects of all sources of variation that are not already explicitly accounted for in the model.

Comparing patterns changes the task of identifying discrepancies in an important way. To accurately diagnose a discrepancy between two patterns, an analyst must observe the entirety of both patterns, which is rarely an option. The entire patterns may contain a large or even infinite number of points. Research budgets will intervene before the observation can be completed. However, comparing subsets of two patterns can be misleading; a subset of a pattern may look very different than the overall pattern. The data analyst must decide whether or not an observed discrepancy between sub-patterns implies a genuine difference between the entire patterns. This introduces a new step into confirmatory analyses: the analyst must decide whether observed differences between the hypothesis and data imply actual differences between the hypothesis and reality. In exploratory analyses, an analyst must decide how closely to fit a model to the data. At what point does the model begin to fit the sub-pattern of the observed data more closely than the implied pattern of the unobserved data? These variance related judgements provide a second characterization of data analysis.

These judgements become harder when data is contaminated with measurement bias and sampling bias. Both types of bias obscure the true pattern of unobserved reality, which invalidates the results of sensemaking. Bias can be minimized by ensuring that the observed data accurately represent reality and that measurements are made accurately. This may require identifying (but not measuring) all of the data points contained in the pattern, which is sometimes referred to as the population of interest, as well as identifying the relationships between unobserved points and the observed points. These considerations make data collection a more salient part of

data analysis than information collection is in sensemaking. Obviously, data analysts can not always control how their data is collected. However, data analysts should always seek out and consider evidence of bias when making variance related judgements. Avoiding and considering bias may be considered a third characteristic of data analysis that distinguishes it from general sensemaking.

2.5 A conceptual model of data analysis

Data analysis combines sensemaking with two data related considerations: how can we compare abstract schemas to precise data? And, how can discrepancy between schema and data be distinguished from variance? These considerations combine with the general sensemaking structure to create a conceptual model of the data analysis process, see Figure 2.3. Data analyses proceed as a series of iterations through sub-loops of this process. Individual analyses will vary by the paths they take and the methods they use to achieve each step.

A generalized exploratory task proceeds as follows:

1. Fit a tentative model to available data
2. Identify differences between the model and data
3. Judge whether the differences suggest that the model is misfit, overfit, or underfit (discrepancies)
4. Retain or refine the model as necessary
5. Select a plausible schema that interprets the model in the context of the research

A generalized confirmatory task proceeds in the opposite direction:

1. Select an appropriate schema to guide data collection.

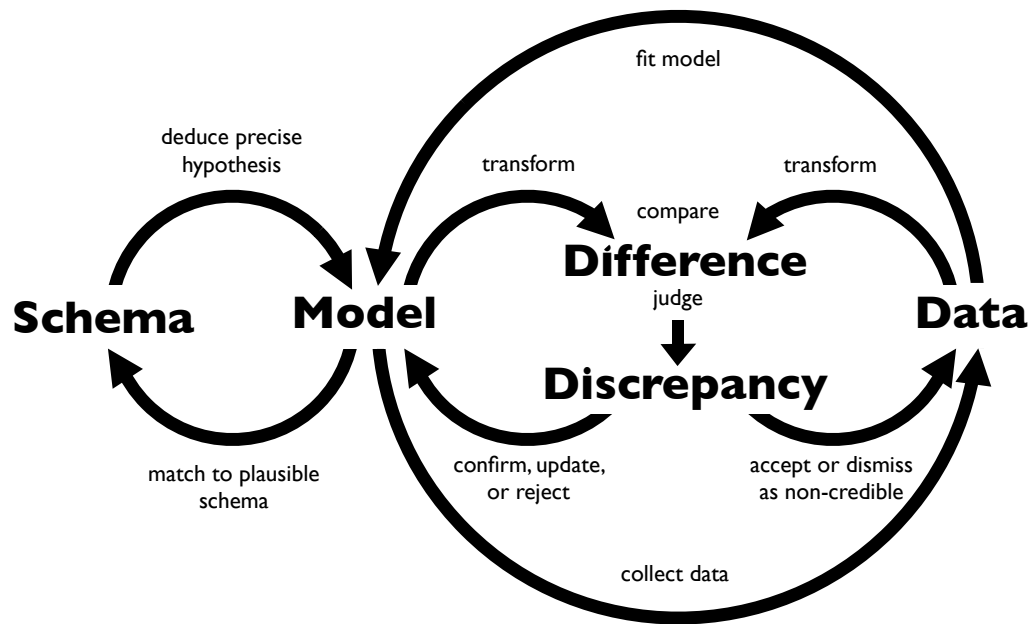


Figure 2.3: Data analysis parallels sensemaking. Analysts deduce a precise hypothesis (model) from the schema, which they compare to the data or a transformation of the data. Analysts must attempt to distinguish discrepancies between schema and data from differences that result from variance and bias. Analysts must also match each accepted model back to a schema to provide interpretation in real world concepts.

2. Deduce a precise hypothesis from the schema. Multiple hypotheses may be developed to test multiple aspects of the schema.
3. Identify the set of data that would be relevant for testing the hypothesis
4. Collect a representative subset of the data.
5. Identify differences between data and hypothesis
6. Judge whether the discrepancies imply a meaningful difference between the hypothesis and reality or result from random variation or faulty data
7. Confirm, update, or reject the hypothesized model (and its associated schema)

This model parallels the descriptions of data analysis offered by Chatfield (1995), Wild and Pfannkuch (1999), MacKay and Oldford (2000), Cox (2007), and Huber (2011) as well as the description of data analysis offered by Tukey and Wilk (1966), and Box (1976) which we discussed before. The model also lends these descriptions explanatory power: data analysis follows consistent stages because it is a sensemaking process that has been adapted to accommodate data. We briefly discuss the alignment of these descriptions with the cognitive model of data analysis below.

2.5.1 Chatfield (1995)

Chatfield (1995) divides an idealized statistical investigation into seven stages. As with the proposed model, the methods used in each stage will vary from situation to situation. The seven stages loosely follow our proposed model:

1. Understand the problem and clarify objectives (*begin with a schema*)
2. Collect data in an appropriate way (*collect data*)
3. Assess the structure and quality of the data, i.e, clean the data

4. Examine and describe the data (*transform data into words, visuals, etc.*)
5. Select and carryout appropriate statistical analyses
 - (a) Look at data (*transform into visuals*)
 - (b) Formulate a sensible model (*make schema precise*)
 - (c) Fit the model to the data (*fit model*)
 - (d) Check the fit of the model (*identify discrepancies*)
 - (e) Utilize the model and present conclusions
6. Compare findings with further information, such as new data or previous findings (*iterate*)
7. Interpret and communicate the results

Many of Chatfield's stages directly map to steps in the cognitive model (shown in italics above). Other stages align with sub-loops of the cognitive model, such as step 3, which requires comparing the data to a schema of "clean" data and then updating the data set. Chatfield's final stage does not match the cognitive model. We agree that communication is an important part of the data analyst's job; however, it occurs after sensemaking has finished. As such, it deals with a different set of cognitive concerns and we refrain from examining it in this article.

2.5.2 Wild and Pfannkuch (1999)

Wild and Pfannkuch (1999) develop a model of the "thought processes involved in statistical problem solving." This model has four dimensions, but the first dimension is a description of the phases of a data analysis: problem, plan, data, analysis, conclusions (PPDAC). These phases were developed by Mackay and Oldford and later published

in MacKay and Oldford (2000). The problem stage involves defining the problem and understanding the context. In these respects, it resembles selecting an initial schema. The plan and data stages involve collecting data relevant to the problem. The analysis stage includes data exploration and both planned and unplanned analyses. These activities search for relevant models and identify discrepancies between the model and the data, when they exist. The final stage, conclusions, encapsulates communicating and using the understanding developed by the analysis. Wild and Pfannkuch (1999) develop connections between data analysis and cognition in other ways as well. They conceptualize applied statistics as “part of the information gathering and learning process.” Wild and Pfannkuch also argue that scientists utilize statistical modeling because we are incapable of handling the enormous complexity of real world systems, which include variation in innumerable components. Modeling provides data reduction, which allows understanding. Schemas play the same role in sensemaking by distilling data and assigning meaning. Wild and Pfannkuch further argue that statistical models become the basis of our mental models, where understanding accumulates, an observation supported by the cognitive model of data analysis.

2.5.3 Cox (2007)

Cox (2007) discusses the main phases of applied statistics with a focus on technical considerations. Like Chatfield (1995) and Wild and Pfannkuch (1999), Cox divides data analysis into general phases that parallel the sensemaking model: formulation of objectives, design, measurement, analysis of data, and interpretation. The formulation phase parallels selecting a schema. The design and measurement phases focus on acquiring relevant data. The data is analyzed by searching for discrepancies with the model. Cox’s interpretation phase focuses on parsing the results of analysis into new

understanding. Our model describes this in cognitive terms as matching the accepted model to a schema.

2.5.4 Huber (2011)

Huber (2011) divides data analysis into the following activities.

1. Planning and conducting the data collection (*collect data*)
2. Inspection (*transform data*)
3. Error checking
4. Modification (*transform data*)
5. Comparison (*identify discrepancies*)
6. Modelling and model fitting (*model fitting*)
7. Simulation (*make schema precise*)
8. What if analyses
9. Interpretation (*match model to a schema*)
10. Presentation of conclusions

Most of these activities directly appear in the cognitive model of data analysis. Other activities, such as error checking, play a general support role to the distinct phases of data analysis that appear in the cognitive model. Like Chatfield (1995), Huber also highlights the important role of communication, which is not covered by the cognitive model. Huber parts with the cognitive model by asserting that “ordering the [above] pieces is impossible.” However, Huber’s explanation of this agrees with the cognitive model: “one naturally and repeatedly cycles between different actions.”

The model of data analysis proposed in this section synthesizes insights provided by prominent descriptions of data analysis. The model explains why these descriptions take the form that they do, and the model provides a framework for understanding data analysis: data analysis is a sensemaking process adapted to measured data. The cognitive model of data analysis also offers an immediate implication for the practice of data analysis, which we discuss in the next section.

2.6 Implications for data analysis practice

The cognitive model of data analysis predicts a set of problems that may undermine data analysis practice. The mind uses sensemaking to build knowledge of the world, but the process has known flaws. If data analysis is built upon sensemaking as we propose, it will inherit these flaws. Each flaw poses challenges for a data analyst. In this section, we discuss two of these flaws and illustrate each with a case study of a notable data analysis failure.

2.6.1 Data analysis is biased towards accepted schemas

The nature of cognition tends to undermine the sensemaking mechanism for detecting faulty schema. People only attend to a small portion of the information in their environment, and schemas direct where this attention is placed (Klein et al., 2003). To understand an event, the brain selects a relevant schema. This selection may be guided by context clues or a few initial observations that serve as anchor points (Klein and Crandall, 1995). The brain then uses this schema to scan the environment for additional relevant sensory inputs. The schema then helps the brain build information from the inputs by assigning meaning to them. In other words, schemas determine where attention will be placed and how observations will be interpreted (Klein et al.,

2003). Information that contradicts a schema is less likely to be noticed (Klein et al., 2003), correctly interpreted (DeGroot, 1965), or recalled later (Woodworth, 1971; Miller, 1962). As a result, the mind is prone to retain incorrect schemas. This tendency has been well documented in educational research. Students are more likely to misinterpret new information than update their misconceptions. For example, when children are told that the world is round, they are more likely to picture a pancake than a sphere (Vosniadou and Brewer, 1989). High school students are likely to retain an Aristotelian worldview even after completing a year long curriculum in Newtonian physics (Macabebe et al., 2010). Statisticians are not immune to this schema inertia either. The “hot hand” effect in basketball (Gilovich et al., 1985) and the Monty Hall problem (Tierney, 1991) are two well known examples where students (and sometimes professors) have been unable to update their schemas despite statistical training.

The mind tends to discredit observations before beliefs whenever it is easy to do so. A direct experience that requires minimal interpretation is often necessary to impugn an accepted schema. In the classroom, schema change can be initiated by having the student examine their beliefs and then creating an experience that directly contradicts the faulty schema (Bransford et al., 2000). In naturalistic settings, schema change usually does not occur until experience violates expectation, creating a shock or surprise (Klein et al., 2003).

The discovery of the hole in the ozone layer illustrates the inertia that incorrect schemas can have in an analysis. In 1974, Molina and Rowland (1974) predicted that industrial use of chlorofluorocarbons (CFCs) could deplete levels of atmospheric ozone, which could have dangerous environmental effects. According to Jones (2008), NASA’s Nimbus-7 satellite began to record seasonal drops in ozone concentrations over Antarctica just two years later. These drops went unnoticed for eight years until the

British Antarctic Survey spotted the decrease in ozone through its own measurements in 1984 (Farman et al., 1985). Why did analysis of the Nimbus-7 data fail to reveal such a dramatic phenomenon for eight years?

The Nimbus-7 delay demonstrates the need to address low level schemas during a data analysis. Analysts normally focus on quantifiable models, which are deductions from low level schemas. But it is the cognitive schema that will dictate where analysts direct their attention and how they will explain their findings. These cognitive schemas are particularly dangerous because they often persist in the presence of contradictory information.

NASA programmed the Nimbus-7 to flag observations of low ozone as unreliable, which accords with an initial belief that ozone values should fall in a known range. When NASA scientists encountered these values, the flag made it easy to explain away the data and preserve their schema. Moreover, the unreliability hypothesis was easy to believe because the Nimbus-7 observations depended upon numerous mechanical, electrical, and communication systems. In other words, the observations were collected through a process too complex for the analysts to cognitively comprehend or mentally check. This could explain why the data did not raise any alarm bells; evidence suggests that observations that seem less certain than direct experience will be ineffective for removing faulty schemas.

The BAS team had two advantages on the NASA team. First, the BAS team did not receive a pre-emptive flag of unreliability with their low ozone measurements. Second, the BAS team measured ozone in person in Antarctica and used much simpler equipment than the NASA team. This imbued their observations with the jolt of direct experience, which facilitates schema change. The lack of complexity in the measurement process allowed the BAS team to assign the same confidence to the measurements that they assign to their everyday sensory experiences.

Analysts can not always collect their data in person with simple tools. However, analysts can guard against faulty schemas by addressing the mechanisms that allow them to persist: mis-attention and premature data rejection. Analysts should consider whether or not they have sought out the type of data that would be likely to disprove their basic beliefs should they be wrong. Analysts can further avoid mis-attention by focusing on all plausible schemas. Tukey (1960) advocates for this approach. According to Tukey, science examines “a bundle of alternative working hypotheses.” Conclusion procedures reduce the bundle to only those hypotheses “regarded as still consistent with the observations.” Considering all plausible schemas helps prevent the adoption of a faulty schema, which may then mis-direct an analyst’s attention.

Once data has been collected, analysts should be circumspect about rejecting data. Individuals are prone to reject data as erroneous when it violates their basic ideas about what data should say. However, this prevents the analyst from discovering that their basic ideas are wrong. Data cleaning is a useful and often necessary part of an analysis, but analysts should be wary of using part of a schema under consideration to filter their data. Instead, data points should only be rejected when a source of error can be found in the data collection or generation mechanism.

Finally, we suspect that analysts can approximate the jolt of direct experience by visualizing their data. NASA’s flagged ozone observations were highly structured. They occurred in a temporal pattern (ozone dips low each Antarctic spring and then recovers). They also occurred in the same geographical location in the Southern Hemisphere. We speculate that had the NASA team noticed this by visualizing their data, the pattern would have been as striking as direct experience and prompted a schema change.

2.6.2 Data analysis does not prove its conclusions

Data analysis inherits a second flaw from sensemaking; it relies on an unsound logical connection between premise and conclusion. As a result, data analysis does not prove its conclusions with logical certainty, and hence, does not completely remove the subjectivity of sensemaking. The reasoning an analyst uses to adopt a schema on the basis of data is as follows:

If schema P is true, data should look like Q

The data looks like Q

Therefore schema P is true

This type of reasoning is not rare, nor is it useless. It is so common in science that it has been given a name: abduction. Abduction was introduced and broadly explored by Peirce (1932). It has been discussed more recently in a statistical context by Rozeboom (1997). Abduction does not prove its conclusions unless there is a one to one mapping between P and Q. More often, alternative schemas R, S, etc. exist that also predict that the data should look like Q. If the data looks like Q, this increases the likelihood that P is true, but it does not rule out the possibility that P is false and R or S is instead true. Yet this is how the human mind functions when sensemaking, and it is how data analysts must function as well.

Data analysts can improve the success of abduction with statistical techniques. Many statistical techniques perform an optimized abduction within a constrained set of models. For example, maximum likelihood estimation chooses the model of the form $P(X = x) \sim f(x|\theta)$ that is most likely to explain the data. However, maximum likelihood does not guarantee that the true explanation is in the set of models of the form $P(X = x) \sim f(x|\theta)$ to begin with. Many statistical methods, such as the method of moments, statistical learning methods, and bayesian estimation methods are all also ways to guide abductive selection. Statistical methods provide a significant

advantage over unguided sensemaking. Humans are extremely prone to be biased by emotionally salient information when reasoning about likelihoods (Tversky and Kahneman, 1974). Statisticians frequently use models as tools without assuming the models are true. This mitigates reliance on abduction. Nevertheless, the abductive nature of data analysis requires caution and corroboration before data is used to make weighty decisions.

The space shuttle *Challenger* accident demonstrates the need to strengthen abductive arguments with further analysis. NASA decided to launch the space shuttle in 31°F weather despite worries that the shuttle's O-rings would leak at that temperature. The O-rings failed, killing all aboard. Prior to launch, engineers from NASA and Morton Thoikol, the manufacturer of the space shuttle, examined data on the relationship between O-ring failure and temperature. They concluded that no relationship existed. This analysis has been widely scrutinized and criticized (see for example, Dalal et al. (1989), Tufte (1997), Presidential Commission on the Space Shuttle *Challenger* Accident (1986), etc.). However, the data that NASA examined *could* be construed to support the belief that temperature does not affect O-ring performance. The seven data points that NASA examined could be seen as random cloud that does not vary over temperature, Figure 2.4 (top). Alternatively, they could be seen as a parabola that suggests increasing danger at extreme temperatures. This is the nature of abduction, it does not rule out competing plausible explanations.

To strengthen its conclusions, NASA should have sought to corroborate its view with a second line of evidence. NASA had access to 17 additional data points from shuttle launches that it could have examined. These points would have cast doubt on NASA's conclusion; a trend between O-ring failure and temperature appears when the additional data points are considered, Figure 2.4 (bottom).

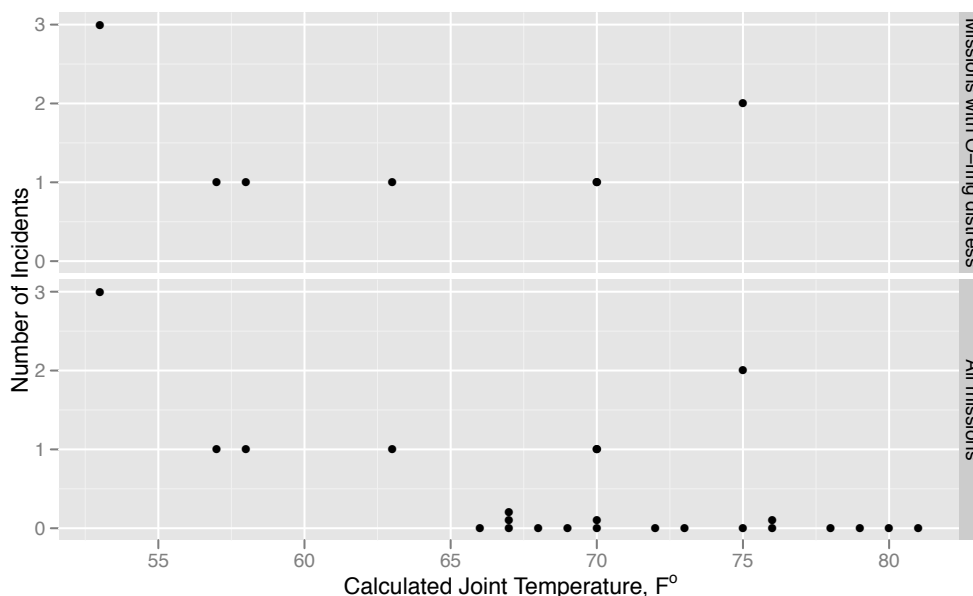


Figure 2.4: The seven flights examined by NASA and Morton Thoikol managers (above). The 24 flights for which information was available (below). Recreated from Presidential Commission (1986) (p. 146).

Even more analysis, however, may have been needed to avert disaster. Lavine (1991) points out that any attempt to predict performance at 31°F from the data would be an extrapolation since the observed data all occurred between 53°F and 81°F. In other words, multiple models could be fit to the existing data and each would predict the performance at 31°F differently. In fact, a careful statistical analysis that considered the leverage of each available data point could potentially be seen as evidence that 31°F would increase the risk of O-ring erosion, but not by enough to pose a severe danger (Lavine, 1991).

In summary, abduction even assisted by statistics could not differentiate between an eventless launch and catastrophe based on the available data. Data analysis could be used to support either argument, although the argument for danger would have appeared stronger. To validate one of the arguments, NASA and Morton Thoikol would have had to collect new data near 31°F that could distinguish between competing

models. Such data was collected during the investigation of the *Challenger* disaster. A controlled experiment of O-ring performance at different temperatures reported by Presidential Commission on the Space Shuttle *Challenger* Accident (1986) (p. 61–62) demonstrated conclusively that O-rings would not perform safely at 31°F.

In general, analysts can avoid trouble by acknowledging the abductive nature of data analysis. Researchers should view an analysis as an argument for, but not proof of its conclusions. An analyst can strengthen this argument by judging the strengths and weaknesses of the argument *during the analysis* and adjusting for them. An analyst can also continue an analysis — often by collecting new data — until one schema appears much more plausible than all others.

Controlled experiments, expertise, and corroboration can also be used to strengthen the abductive step of data analysis. An experiment can be designed to limit the amount of plausible schemas that can be abduced from, which increases the likelihood of success. Subject matter expertise provides the analyst with more relevant schemas to select from, which allows a better mental approximation of reality. Expertise also helps ensure that the analyst will know of a correct schema, which is a prerequisite for selecting one during abduction. Expertise also broadens the amount of previous information and data that the analyst can utilize when matching a schema. Finally, an independent line of argument can corroborate the results of an abduction if it comes to the same conclusion.

2.7 Conclusion

This paper identifies data analysis as an extension of the internal cognitive processes that build knowledge. In particular, we propose that data analysis is a sensemaking process that has been modified to use precisely measured data. This improves the performance of sensemaking, but creates a new set of problems that exist in every

data analysis. Every data analysis must choose a way to express abstract concepts precisely (often quantitatively). Every data analysis must also find a way to identify discrepancies between a schema and reality in the presence of variation. These problems characterize data analyses and give them a recognizable pattern. Moreover, data analysis inherits weaknesses from the sensemaking processes upon which it is built. In this paper, we identify two such weaknesses: the unusual persistence of false schemas and the unavoidable subjectivity of model and schema selection.

We began this paper by pointing to the need for a formal theory of data analysis. Does the cognitive model of data analysis qualify as a formal theory of data analysis? Perhaps not. Philosophers of science have offered multiple definitions of a scientific theory. These range from the axiomatic to the semantic and usually require a degree of mathematical precision that our conceptual model does not offer. However, the cognitive model of data analysis meets our pragmatic view of a theory. It offers an explanatory framework for data analysis that synthesizes available information and makes predictions about data analysis tasks.

The cognitive model of data analysis may not change the way data analysis is practiced by experienced statisticians. We believe that the prescription offered by the model is very similar to current expert practices. The value of the model lies instead in its portability. Current methods of statistical training have been criticized because novices must acquire years of experience before they settle into expert data analysis practices. In contrast, the cognitive model can be taught to novice statisticians to guide data analysis practices from the get go. It is easy to understand that a data analysis seeks to minimize discrepancies between theory and reality. It is easy to accept that the mind goes about this in an innate way. It is also easy to see that this task can be hindered by cognitive, logistical, and epistemological obstacles. The details of data analysis emerge as these problems arise and are overcome.

The cognitive model also provides a way to unify the field of statistics, as advocated by Huber (1997); Viertl (2002) and others. The model focuses on cognition, but it does not ignore the contributions of statistics to data analysis. Instead it organizes them. Statistical pursuits can be associated with the steps of data analysis that they perform or support. Individual techniques of data analysis, such as design of experiments, data visualization, etc., can be categorized and criticized by identifying which problems they solve. This arrangement highlights how different areas of statistics interact with each other. It also provides a global framework for students trying to master the field of statistics.

The cognitive model also offers guidance for adapting data analysis to new contexts. Small sample statistical methods may become less applicable as the size and nature of data sets change, but the general structure and challenges of data analysis will remain. The cognitive model identifies these challenges: analysts will need methods that facilitate comparisons between data and schema and allow judgements of dissimilarity in the presence of variation. Analysts will need ways to develop abstract schemas into precise models that describe patterns of observation, and they will need guidance for transforming the best fitting models into real world explanations.

Finally, a cognitive interpretation of data analysis also offers a way to improve current data analyses. A cognitive view suggests that cognitive phenomena may adversely affect data analysis – often in unnoticed ways. We examined two such effects in Section 2.6. Other cognitive phenomena with other effects should also be looked for. Each would provide new opportunities to improve data analysis. This focus on the human analyst distinguishes the cognitive model of data analysis from other models of science, which it may appear similar to. A focus on the human analyst is necessary. When errors in analysis occur, they will do harm because they violate Aristotelian logic or Sir Karl Popper’s principles of falsification. But the cause of these

errors will be ingrained human tendencies. To prevent such errors, data analysts must understand and watch for these tendencies.

Visualizing complex data with embedded plots

This chapter has been submitted for publication to the Journal of Computational and Graphical Statistics as ‘Visualizing complex data with embedded plots’ by Garrett Grolemund and Hadley Wickham

ABSTRACT — We describe a class of graphs, embedded plots, that are particularly useful for analyzing large and complex data sets. Embedded plots organize a collection of graphs into a larger graphic. This arrangement allows more complex relationships to be visualized with static graphs than would be otherwise possible. Embedded plots provide additional axes, prevent overplotting, provide multiple levels of summarization, and facilitate understanding. Complex data overwhelms the human cognitive system, which prevents comprehension. Embedded plots preprocess complex data into a form more suitable for the human cognitive system through visualization, isolation, and automation. We illustrate the usefulness of embedded plots with a case study, discuss the practical and cognitive advantages of embedded plots, and demonstrate how to implement embedded plots as a general class within visualization software, something currently unavailable.

3.1 Introduction

Analyzing large, complex data is difficult. Complex data strains the human cognitive system, which can prevent comprehension. Visualizations can help, but it is difficult to visualize more than two or three dimensions at once in a static graph. We present a class of graphs, embedded plots, that are ideal for visualizing complex data.

Embedded plots can be generalized as graphics that embed subplots within a set of axes. Figure 3.1 shows three graphs that represent this type of plot: William Cleveland’s subcycle plots, glyphmaps, and the binned graphics that are emerging from big data visualization efforts. When viewed on its own, each subplot is a self contained plot (or would be if it contained the appropriate axis, labels, and legend). The axes of the subplot do not have to be the same as the axes that the subplot is positioned on. In fact, the subplot can use an entirely different coordinate system than the higher level plot. For example, Figure 3.1.b. embeds polar graphs in a cartesian coordinate system.

Embedded plots have a rich pedigree and a growing future. Subcycle plots were devised by William Cleveland (Cleveland and Terpenning, 1982), one of the leading innovators in computer based graphics. Glyphs and other plots have been embedded in maps since Charles Minard (Minard, 1862). Such maps figure prominently in Bertin’s *Semiologie of Graphics* (1983), a seminal work in the academic study of visualization. Embedded maps comprise 21 pages of the text. More recently, glyphmaps have been developed as a tool for tracking climate and climate change data (Wickham et al., Submitted; Hobbs et al., 2010). The binned graphics of Figure 3.1.c are a promising candidate for solving the problem of overplotting when visualizing big data. Other types of embedded plots are widely used as well. Glyphs (Anderson, 1957), trees and castles (Kleiner and Hartigan, 1981), chernoff faces (Chernoff, 1973), stardiates (Lanzenberger et al., 2003), icons (Pickett and Grinstein, 1988) and oth-

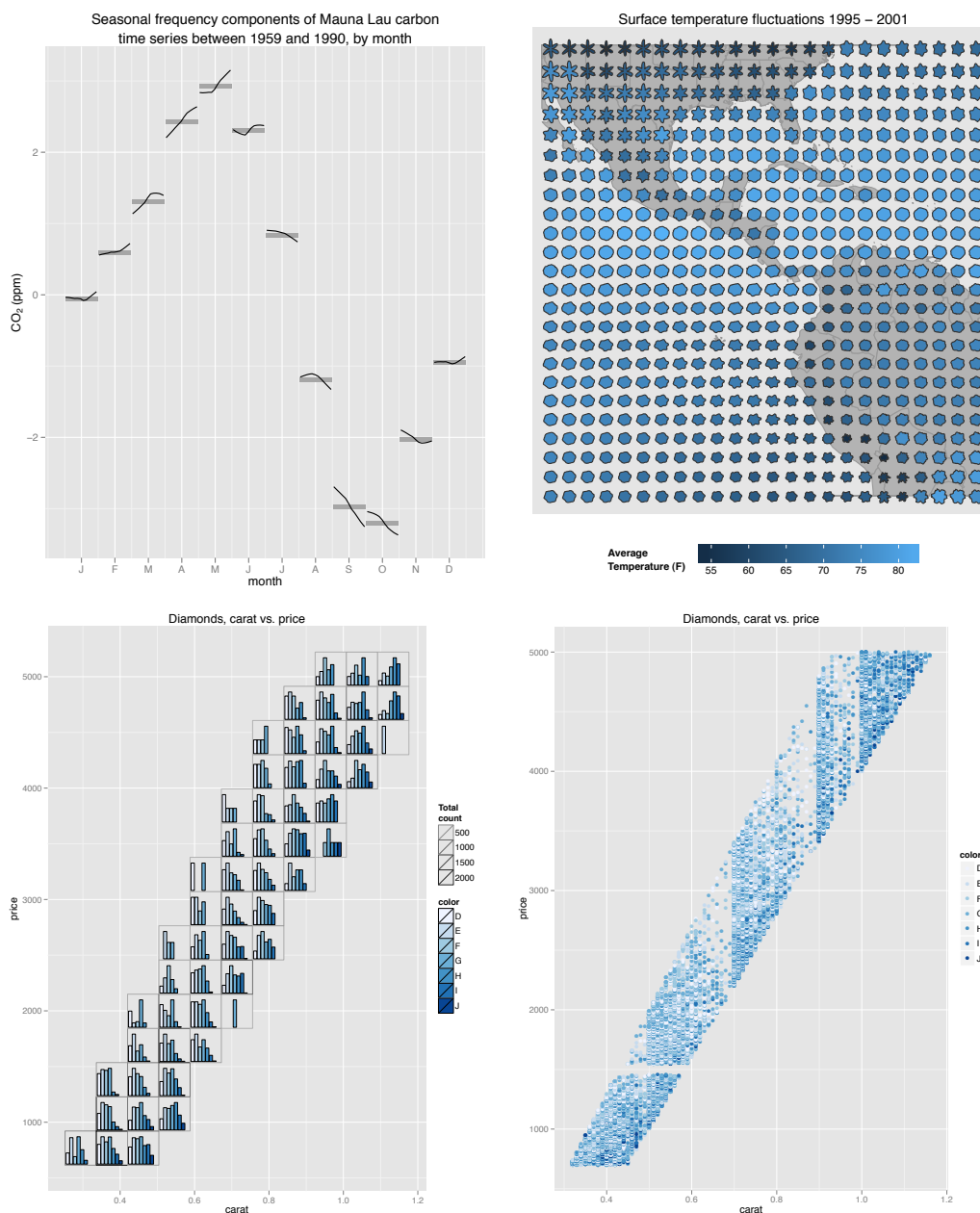


Figure 3.1: Three examples of graphs that use embedded subplots. **A.** (*upper left*) A subcycle plot of CO_2 measurements taken on Mauna Lau, Hawaii between 1959 and 1990. Recreated from Cleveland (1994), page 187. Observations are grouped by month. **B.** (*upper right*) A glyphmap of temperature fluctuations in the western hemisphere over a six year period. Each glyph is a polar chart with $r = \text{temperature}$ and $\theta = \text{date}$ these charts are organized on a cartesian plane with $x = \text{longitude}$ and $y = \text{latitude}$. **C.** (*lower left*) A binned plot of the diamonds data set from the `ggplot2` software package. Subplots are used to show patterns in diamond colors without overplotting. When this data is presented in its raw form, the accumulation of points hides patterns in the data (*lower right*).

ers have been developed as types of subplots that can be compared to each other. Scatterplot matrices (Chambers, 1983), trellises (Sarkar, 2008) and facets (Wilkinson and Wills, 2005) are popular types of embedded graphics that arrange subplots into a table. We generalise all of these graphs into a larger class of plots, embedded plots, because they all share a two tier structure. The first tier is the overall graph or visual itself, the second tier is the collection of subplots that appear within the graph.

The two tiered structure of embedded graphs makes them well suited for solving a number of data analysis problems. The examples in Figure 3.1 illustrate three areas where embedded graphics are particularly useful. First, embedded graphics make it easy to visualize interaction effects. For example, Figure 3.1.a shows that the direction of long term change in CO_2 levels at Mauna Lau observatory in Hawaii varies by month in relation to seasonal patterns in CO_2 concentrations. Embedded graphics also provide an intuitive way to organize spatio-temporal data. Visualizing spatio-temporal data usually requires four or more dimensions: two for spatial coordinates, a third for the passage of time, and a fourth for the quantity of interest. The glyphmap in Figure 3.1.b organizes these dimensions in a way that is easily interpreted and that makes both spatial and temporal patterns obvious. Finally, embedded graphics solve the problem of overplotting. Figure 3.1.c. represents almost 20,000 observations. When this data is plotted as a colored scatterplot, the accumulation of points obscures the underlying relationship between carat, color, and price. The use of binned subplots makes the relationship visible again. Yet embedded plots provide more than just practical advantages.

Embedded plots amplify the abilities of the human cognitive system by presenting complex information in a way that is particularly easy to process. Complex data is data that includes multiple simultaneous relationships between its elements. At the cognitive level, complex data overwhelms the capacity of the working memory Sweller

(1994). Repeated studies have shown that it is difficult to comprehend, use, and teach complex data.¹ Moreover, success in understanding complex data depends heavily on how the data is presented Mayer (2009). Embedded plots present data in a way that exploits several known mechanisms for facilitating the processing of complex data. As a result, embedded plots may allow viewers to comprehend information that they would not grasp in other formats.

As useful as embedded plots are, it is difficult to make them. Currently, programs that can make embedded plots focus on a specific type of subplot, such as glyphs (Gribov et al., 2006) or scatterplot matrices (Sarkar, 2008). This limits the customizability and usefulness of embedded plots. We discuss the advantages of embedded plots and describe how embedded plots can be implemented as a general class of graphs in data analysis software.

The remainder of this paper proceeds as follows:

Section 3.2 begins with a case study that presents the usefulness of embedded plots. We explore the Afghan War Diary data, made available by the WikiLeaks organization. The data set is large and complex: 76,000+ observations organized by location and time. The case study shows how embedded plots can be used in practice to reveal patterns that can not be seen in single level graphs.

Section 3.3 examines why embedded plots are useful tools for finding and communicating information found in large data sets. At the practical level, embedded plots have two advantages: they provide two extra axes and a high degree of customizability. More importantly, however, embedded plots exploit several cognitive mechanisms for attending to and processing information. This allows embedded plots to present complex information without becoming muddled or indecipherable.

Section 5.5 discusses how generalized embedded plots can be implemented in data

¹See Sweller et al. (2011) for an overview.

analysis software. We present a very customizable implementation of embedded plots that uses the layered grammar of graphics (Wickham, 2010) and the `ggplot2` package (Wickham, 2009) in R. Incorporating embedded plots into the grammar of graphics yields a new insight about graphics: they have an inherently hierarchical structure.

Section 3.5 concludes by offering general principles to guide the use of embedded plots.

3.2 Case Study: Analyzing complex data

The Afghan War Diary data, made available by the WikiLeaks organization at http://www.wikileaks.org/wiki/Afghan_War_Diary,_2004-2010, is large, complex and intriguing, because it provides insights into an ongoing military conflict. The data set was collected by the US military and contains information about military events that occurred in or around Afghanistan between 2004 and 2010. Among other variables, the data set records the number of injuries and deaths that resulted from each event. These casualty statistics are collected for four groups: enemy forces (enemies), coalition forces (friendly), Afghanistan police and security forces (host), and civilians (civilians). The data set is large enough (76,000 observations) that overplotting becomes a concern when visualizing the data. The data set is complex in that it contains a spatio-temporal component: each observation is labelled by longitude, latitude, and date. Our analysis will focus on two topics: the ratio of civilian casualties to combatant casualties and the escalation (or de-escalation) of hostilities since 2004 as measured by total casualties. We will calculate total casualties based only on the number of wounded and killed in each group. The Afghan War Diary does not have complete information on the number of people captured or missing across all four groups.

3.2.1 Civilian casualties

Operation Enduring Freedom, the US led military engagement in Afghanistan, has received international criticism for the high number of civilian casualties associated with the war. The Afghan War Diary seems to justify this criticism. Civilians comprise almost a quarter of all casualties recorded in the diary, and civilians have suffered more casualties (12,871) than coalition (8,397) and Afghan (12,184) forces. Civilians have nearly half as many casualties as enemy forces (24,233). We wish to see if these ratios vary by location. Are civilian casualties noticeably high everywhere the war has been fought, or just for certain locations, such as urban centers, where military action occurs in close proximity to a large number of civilians?

The size of the Afghan War Diary makes it difficult to visualize this information. When plotted as a point map, individual casualties obscure one another, a phenomenon known as overplotting, Figure 3.2.a. A heat map avoids overplotting, but can not show casualties by type, Figure 3.2.b. We only see that the majority of casualties occur in the southern region of Afghanistan between Kabul and Kandahar. To examine casualties by type, we would have to create four separate heat maps, each with a different subset of the data. We turn to embedded plots for a simpler solution. In Figure 3.2.c, we replace each tile in the heat map with a bar graph of casualties by type. This embedded plot reveals similar information as the heat map, but it also displays the ratio of casualties for each area. We can further adjust the embedded plot to show the conditional distribution of casualties for each region, Figure 3.2.d. This technique makes regional patterns more clear and would not make sense for a heat map or contour plot.

The plots show that civilian casualties often surpass coalition and host casualties, and sometimes enemy casualties. Near Kabul, civilian casualties seem to surpass all other types of casualties put together. The visualizations suggests that alarmingly

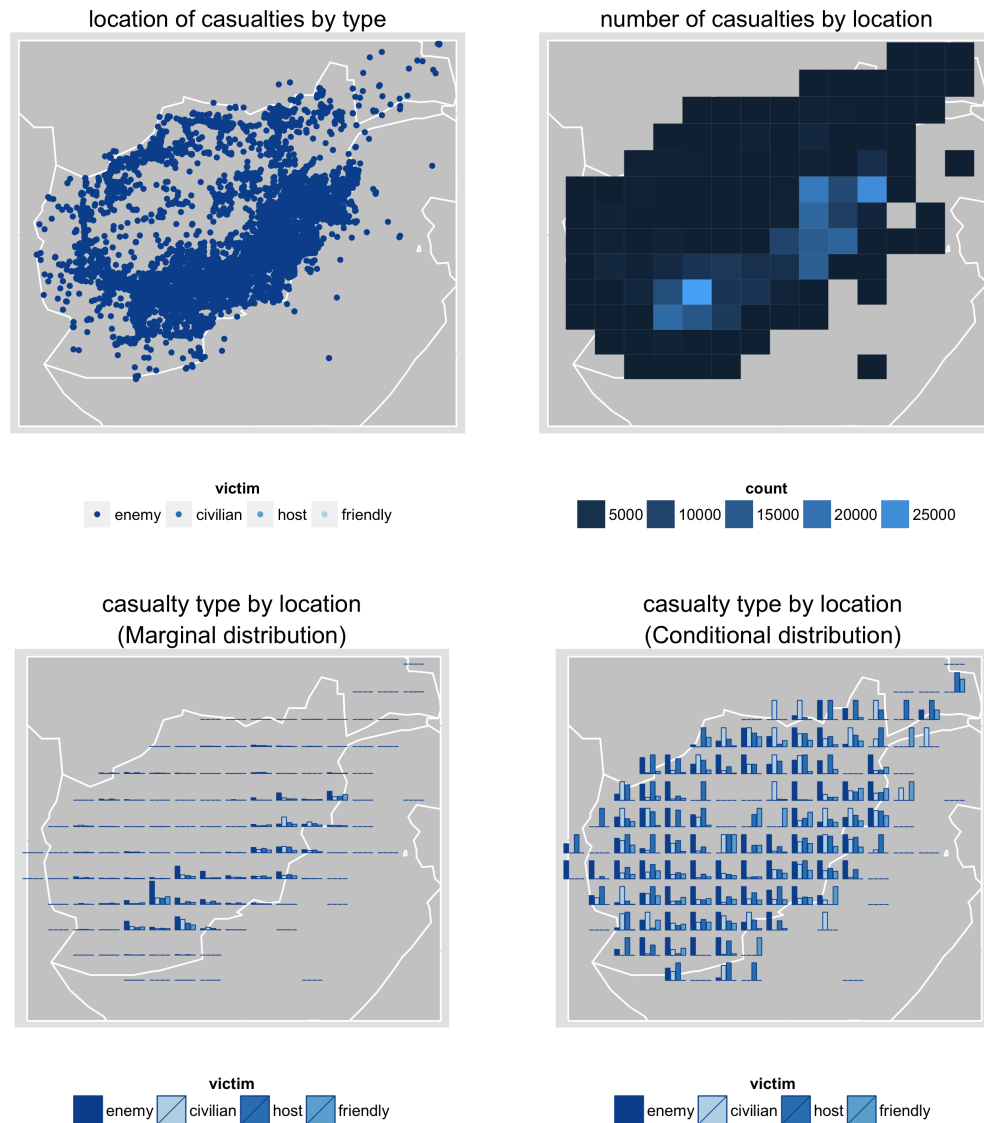


Figure 3.2: **A.** (*upper left*) Relative rates of casualties by area in Afghanistan between 2004 and 2010. Raw data can not be visualized due to overplotting. **B.** (*upper right*) A heat map shows casualty counts, but not relative rates by group. **C.** (*lower left*) Embedded bar charts reveal that there have been more civilian than combatant casualties around Kabul, the capital of Afghanistan. Marginal bar charts reveal similar information as a heat map, but also display rates by group. **D.** (*lower right*) Conditional bar charts make regional rates the more obvious; they show that inordinate civilian casualties is not unique to the capital city.

high civilian casualty rates occur throughout Afghanistan and not just near population centers like Kabul, although high civilian casualty rates also occur there as well.

3.2.2 Frequency of hostilities

Operation Enduring Freedom has also been criticised for lasting longer than any previous American war without showing signs of abatement. We would like to look for signs of abatement in the total number of casualties by region. If the total number of casualties in a region has decreased over time, this may suggest that the region has become pacified, a sign of progress.

Events in the Afghan War Diary are labelled according to the region in which they occurred: the capital, the north, the east, the west, or the south and unknown locations, which mostly have latitude and longitude positions in Pakistan. These labels allow us to visualize how the war has progressed in different areas over time, Figure 3.3.a. However, we can only see the change in time with this plot. Embedded line plots allow us to see variation in space and time simultaneously, Figure 3.3.b. We again plot the conditional distributions to better see the pattern in each region, Figure 3.3.c. We can also use the background color of each subplot to display the total number of casualties per region. This is the information we would normally lose by looking at conditional distributions instead of marginal distributions. We see that casualties peaked in most locations around 2007, but have been on the rise again in the most recent years.

Although the embedded plot “increases” the complexity of Figure 3.3.a by adding two new dimensions (latitude and longitude) and over 100 new lines, it actually makes it easier to see the spatio-temporal relationship. The viewer no longer has to expend mental energy thinking about which line in Figure 3.3.a corresponds to which

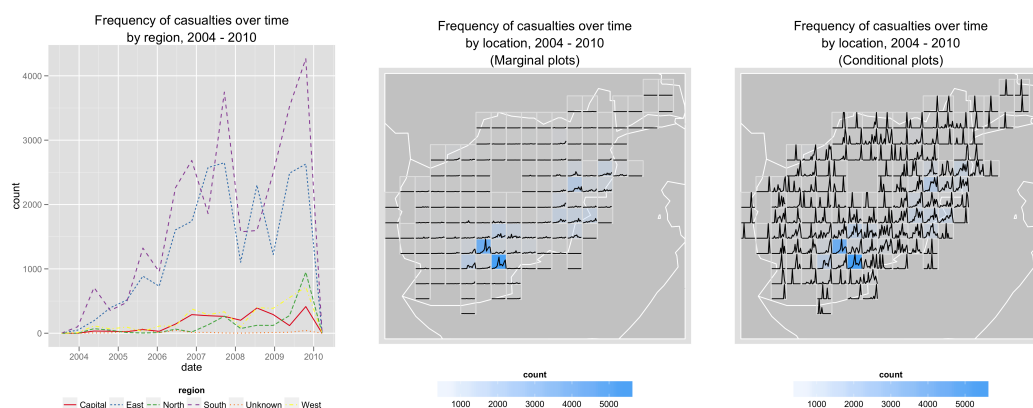


Figure 3.3: Casualty frequencies between 2004 and 2010 by region. The embedded graphics show that the heaviest fighting has been confined to the southern and eastern regions of Afghanistan. The most casualties have occurred around Kandahar. Many regions seem peaceful since 2008. However, casualties have increased recently throughout southeast Afghanistan.

part of the country.

3.3 Benefits of embedded plots

Embedded subplots expand the power of static graphics. Adding a second tier of information in the form of subplots creates practical advantages not available with non-embedded plots. This second tier may at first seem counterproductive: embedded subplots increase the complexity of the graph, which can obstruct comprehension. However, embedded subplots present information in a way that minimizes the cognitive load a viewer must expend to understand the graph. This makes embedded subplots unusually comprehensible. Below, we review the practical advantages of embedded subplots as well as the cognitive science findings that suggest that embedded subplots can be simple and easy to understand.

3.3.1 Practical advantages of embedded subplots

Embedded graphics provide two advantages over non-embedded graphics: they allow customizable summarization and provide additional x and y axes. Each of these advantages can be used in a variety of ways.

Common strategies for overplotting, such as heat maps and contour maps, summarize data into a single number and then attempt to visualize that number. In contrast, subplots summarize information into an image, which can carry more information than a lone number. For example, the bar charts in Figure 3.2.c display multiple measurements in the same space as a heatmap tile, which only displays one. By summarizing with an image, subplots allow users to choose between no summarization, partial summarization and complete summarization, Figure 3.4. Distracting data can be removed, but enough information can be retained to display complex relationships.

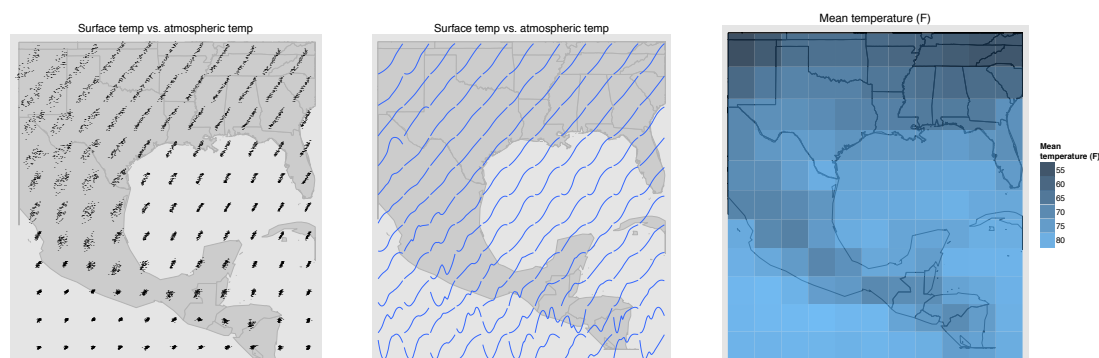


Figure 3.4: Users can control the amount of summarization that occurs in an embedded plot. When scatterplots are used for subplots, no summarization occurs (*left*). Line graphs provide partial summarization (*center*). Heat maps provide complete summarization, within each bin data is reduced to a single number (*right*). This may not always be desirable.

The choice of a subplot also allows the user to control effects of overplotting. For example, Figure 3.1.c summarizes more than 20,000 data points. When this data is viewed as a colored scatterplot, points occlude each other and underlying patterns

are hidden, Figure 3.1.d. The use of embedded subplots avoids overplotting and shows a relationship between price, carat, and color: for any value of carat, better colored diamonds occur more often in the higher price ranges than the low ones. The embedded subplots in Figure 3.1.c would not suffer from overplotting even if the data set was enlarged to 100,000, a million, or even a trillion points.

Embedded subplots also provide a second practical advantage: they supply an additional set of axes to plot data on, the minor x and y axes of the subplots. These two new dimensions allow complex relationships to be visualized. Four separate variables can be assigned between the major x, major y, minor x, and minor y axes. Additional variables can be included with colors, shapes, sizes, etc. The usefulness of this approach is most easily seen with spatio-temporal data. Spatio-temporal data usually requires at least four dimensions to be visualized: two dimensions for spatial coordinates, a third dimension for time, and a fourth for the quantity of interest. Four variables can quickly clutter a non-embedded plot, but an embedded plot can visualize them with just the major and minor x and y axes. The extra axes can be used in a similar way to visualize any high dimensional relationship, such as interaction effects and conditional effects. The two level system of axes can also be used to organize data first by groupwise characteristics, then by individual characteristics.

3.3.2 Cognitive advantages of embedded subplots

These practical advantages come at the expense of making a graph more complicated. Embedded plots add an extra layer of information that a viewer must process before comprehensions can occur. However, when used appropriately, embedded plots may not be much more difficult to understand than non-embedded plots. Furthermore, embedded plots may allow users to understand information that would be incomprehensible in other formats. This is because embedded plots organize information in a

way that lowers the cognitive load required to process the information.

Cognitive load is the mental energy required to convert information into knowledge, understanding, and insights within the working memory. Cognitive science has long known that the working memory has a fairly small processing capacity (Miller, 1956; Cowan, 2000). In 1988, John Sweller demonstrated that learning fails to occur when the cognitive load of a task exceeds the capacity available in the working memory (Sweller, 1988). This insight, the basis of Cognitive Load Theory, has led to a series of successful education principles that work by reducing the cognitive load required during learning tasks (such as reading a graph).¹

Cognitive load theory explains why it is easy to make a graph confusing by including distracting information or multiple variables. Each variable increases the cognitive load required to interpret the graph by adding new information that the viewer must process. Current estimates suggest that the average person has trouble processing more than four pieces of novel information at once (Cowan, 2000). Complex data affects the working memory in the same way as a complicated graph. Each relationship and interaction increases cognitive load and threatens to overwhelm the working memory (Sweller, 1994). When this happens, comprehension will not occur. However, mechanisms exist that can decrease the cognitive load associated with learning tasks. These mechanisms allow more information to be processed than would otherwise be possible. Embedded plots automatically employ three such mechanisms: visualization, isolation, and automation.

Visualization is an extremely powerful information processing tool. All graphics rely on it, but embedded plots use it in a specific way to present information with a decreased cognitive load. Mounting evidence suggests that thinking is a primarily visual activity. The mind uses visual simulations to process verbal information, such

¹See Sweller (2003) for an extensive list of these principles and the supporting literature

as the orientation of words in a sentence (Stanfield and Zwaan, 2001) and the relationship between words (Zwaan and Yaxley, 2003). The mind also relies on visual simulations to compare numbers (Moyer and Landauer, 1967; Dehaene, 1997), to perform approximate arithmetic (Dehaene et al., 1999; Walsh, 2003), and to make logical deductions (Lakoff and Núñez, 2000). The human adaptation to vision is reflected in our working memory system, which treats visual and verbal information differently (Baddeley and Hitch, 1974). The working memory can only handle four novel objects, whether verbal or visual. However, each piece of visual information can be an image that contains multiple features. A study by Luck and Vogel (1997) demonstrated that four visual objects that each contain four pieces of information can be processed by the working memory as easily as four visual objects that each contain only one piece of information. This ability gives the working memory a higher bandwidth for visual information than for verbal information. If we compare one tile of a heat map to one subplot, we see that embedded plots exploit this bandwidth more effectively than other graphs. Both the tile and the subplot are a single visual object. The tile has one feature (a color). The subplot has four (four bar lengths). Luck and Vogel (1997)'s study suggests that the subplot should require little (if any) more cognitive load to be processed by the working memory than the tile. The subplot, however, conveys four times as much information.

Embedded plots also organize information in a way that further decreases cognitive load. The working memory must expend considerable cognitive energy to process new information, but almost no energy to recall and use previously acquired information (Sweller, 2003). When the complexity of a data set exceeds the capacity of the working memory, the mind can proceed by dividing the data set into small pieces and processing each separately. This is akin to rote learning. It does not create full understanding; connections between the separate pieces go unnoticed and unexamined.

However, once each piece is processed, it becomes part of the long term memory where it can be recalled at little to no cognitive cost. Further processing can then occur until full understanding is attained. Sweller et al. (2011) call this the isolating elements effect. The mind can build a deep understanding of highly interactive (i.e., complex) data by iterating between processing small subsets of data and then recalling these subsets from the LTM to compare against each other and new information.

Embedded plots usually display information that can not be understood without an approach that isolates elements; these plots usually deal with at least four interacting dimensions (major x, major y, minor x, minor y). Such data will always demand a heavy cognitive load for comprehension. However, embedded plots make this load manageable by dividing the data into isolated elements (subplots) and visualizing the interactions between these elements (the overall graph). This arrangement allows the mind to use its strongest information processing channel, visualization, to perform the isolating elements processing algorithm. As a result, embedded plots are a particularly efficient way to present information that has four to six interacting dimensions in a static graph.

Embedded plots also benefit from a third cognitive mechanism: automation. To process new information, the mind uses a cognitive structure known as a schema. The schema directs attention during information processing and identifies relationships between data points and previous knowledge.¹ When the mind frequently uses a particular schema, it becomes *automated* (Schneider and Shiffrin, 1977; Shiffrin and Schneider, 1977). When this happens, information related to the schema can be processed with less and less conscious effort. Kotovsky and Simon (1985) demonstrated that automated processing decreases cognitive load to such an extent that information can be processed 16 times faster than with non-automated schemas. A

¹Literature on schemas are extensive. See Neisser (1976) Rumelhart (1980), etc. for highlights

common example of automated processing is reading written text. For young children, reading is a laborious process that involves identifying letters, assigning sounds to them, associating these sounds with words and then meanings. However, by the time children become adults, these tasks are done unconsciously and reading proceeds automatically. Reading graphs is a second example of automated processing.

Embedded plots rely on graph reading skills to convey information: each subplot is a new graph. For analysts familiar with reading graphs, embedded plots allow information to be processed automatically, which results in quicker processing and reduced cognitive load. Embedded plots provide twice the opportunity for automation when subplots are embedded in a map. Reading data off a map and associating it with spatial coordinates is an activity commonly practiced by analysts and non-analysts alike. Information may be automatically read off these graphs at both the plot level and the subplot level. Embedded plots will not offer the benefits of automation to everyone, though. Occasionally, we hear anecdotal reports of people who can not easily read graphs. Until a person learns to read statistical graphs, conscious effort will be required to interpret embedded plots, but this will be true for other types of graphs as well.

In summary, embedded plots display more information than other static graphs, but remain easily interpretable. They present information visually, with an intuitive organization and a familiar presentation. As a result, they minimize the cognitive load needed to comprehend and interpret graphs. This is an attractive feature: it allows embedded plots to display complex relationships that would not otherwise appear in static graphs. More fundamentally, embedded plots may allow users to comprehend complex relationships that would remain incomprehensible in other formats. Embedded plots are not a panacea for all big data situations: it is possible to abuse embedded plots, as we describe in Section 3.5. Also embedded plots can not

effectively visualize relationships that involve more than six dimensions. However, embedded plots provide a way to present one or two additional dimensions in a static graphic; this creates increased opportunities for exploring and understanding large, complex data.

3.4 Implementing embedded plots with the grammar of graphics

Embedded graphs are useful, but difficult to make. Particular types of software exist to make particular types of embedded plots. For example, interactive glyph plots can be made with `gaugain` (Gribov et al., 2006). Facetted graphs can be made with the `ggplot2` (Wickham, 2009) and `lattice` (Sarkar, 2008) packages in R. Scatterplot matrices can be created with the `GGally` package (Schloerke et al., 2011) as well as with base R (R Development Core Team, 2010). However, these programs do not allow users to customize which type of subplot to use in an embedded plot. This customization is one of the chief advantages of embedded plots. Different types of subplots reveal different types of relationships and provide different levels of summarization. In this section, we describe how to create software that can produce any type of embedded plot. Our implementation is built on the layered grammar of graphics and reveals a conceptual insight about graphics: graphs are hierarchical, or recursive, in structure. The implementation of embedded subplots described in this section is available for the R programming language through `ggsubplot`. `ggsubplot` is a software package written by the authors that implements embedded plots within the grammar of graphics paradigm. `ggsubplot` is written in the R programming language and extends the `ggplot2` package. The `ggsubplot` package is available from <http://github.com/garrettgman/ggsubplot>.

Embedded plots can be easily implemented in software built on the layered grammar of graphics, a conceptual framework for understanding and creating visual graphics. The grammar was proposed by Wickham (2010) and builds on ideas from Wilkinson and Wills (2005) and Bertin (1983). The layered grammar organizes each graph into a collection of visual elements and a set of rules that describe how the appearance of these elements should be mapped to a data set. The grammar enables a deeper understanding of how graphics function and relate to one another and allows more concise, elegant programming. This approach to graphics has become widely popular: `ggplot2`, an implementation of the grammar of graphics in R, has been cited over 200 times in scholarly journals and supports an online community of 2500 members. The grammar creates efficiencies and insights by replacing a descriptive taxonomy of charts with a set of general rules that can be used to make almost any type of graphic.

The layered grammar of graphics centers around two concepts: *geoms* and *mappings*. A geom is a visual element in a graph whose appearance can vary in relation to an underlying data set. For example, the points in a scatterplot are a type of geom. Their locations (and sometimes their sizes and colors) reflect values in the underlying data set. Other types of geoms include the bars in a bar chart, the lines in a line chart, boxplots, et cetera. Each type of geom has its own visual characteristics (called *aesthetics*). These visual aesthetics can be altered in meaningful ways to display the values of an underlying data set. For example, the color of a point can be used to display the gender of an observation in the data set. Two of the most important aesthetics are a geom's position along the x axis and y axis. The grammar of graphics calls the rules used to map aesthetics to variables in a data set *mappings*. Geoms and mappings provide a useful framework for building generalized graphs.

Embedded graphics fit seamlessly with the grammar of graphics if we recognize that a plot can be a geom (and that every geom is a plot). Embedded subplots share

the useful characteristics of a geom. They can visually represent data within a graph, and they possess aesthetics that can be mapped to a data set's values. Subplots have two primary aesthetics: their position in the cartesian plane and their internal drawing of a graph. This second aesthetic makes subplots appear more complicated than other geoms, but they function in the same way.

Cleveland's subcycle plot demonstrates the equivalence between subplots and geoms, Figure 3.5. The plot visualizes atmospheric CO_2 concentrations as measured at the Mauna Loa Observatory in Hawaii from 1959 to 1990 (Cleveland, 1994). This data was some of the earliest to suggest the presence of man made global climate change. CO_2 readings are organized by month along the x axis. Within each month, CO_2 readings are arranged by year. This gives the cycle plot its embedded structure. Each group of readings from a particular month can be read as a stand alone plot once the appropriate axes are added back in, see Figure 3.5.b. In the subcycle plot, each subplot contains an x position, a y position, and a drawing of a line graph. If we remove the internal drawings of the line graphs, as in Figure 3.5.c, what remains is a scatterplot whose points are rectangular. This demonstrates that subplots are equivalent to a rectangle geom, but contain a specialized aesthetic: the internal drawing of a graph. This aesthetic can be mapped to the underlying data set with a graph specification.

It may seem exotic to equate a description of a graph with a mapping between a visual feature and data, but this follows a basic tenet of the grammar of graphics: a graph is an abstract mapping from data to visualization:

We can construct a graphic that can be applied to multiple datasets. Data are what turns an abstract graphic into a concrete graphic. (Wickham, 2010)

In summary, a subplot is a type of geom with its own set of aesthetics. One of these

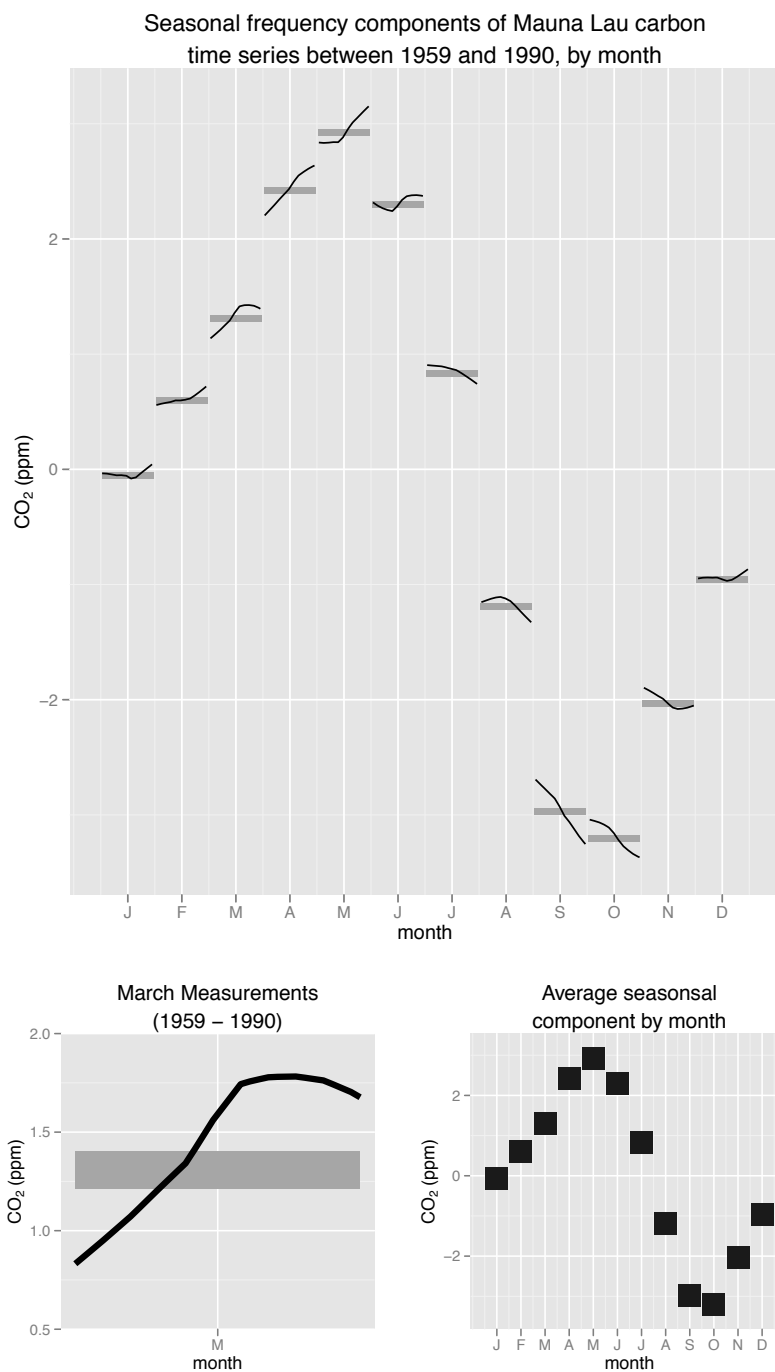


Figure 3.5: Cleveland's subcycle plot can be decomposed into twelve subplots arranged as a scatterplot. The subplots behave as a rectangle geom with an internal drawing aesthetic.

aesthetics is an internal drawing of a graph. The appearance of this aesthetic is controlled by a graph specification, which creates a mapping between the data and the aesthetic. Note that the internal drawing of a subplot may or may not contain axes, a grid, a legend, etc. just as a regular graph may or may not contain these elements.

Although it may seem trivial, the equivalence between subplots and geoms operates in the opposite direction as well. Each geometric object is itself a type of subplot when viewed in isolation. This is easy to see with boxplots and bar graphs, but for many geoms the resulting subplot is so uninteresting that it may go unrecognized, see Figure 3.6.

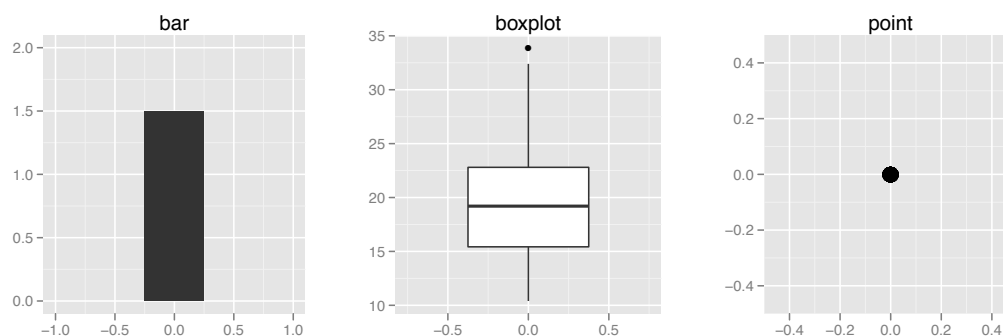


Figure 3.6: Every individual geom is a self contained plot when paired with a set of axes. Such plots may be not be very interesting, as is the case with point geoms.

3.4.1 Advanced implementation

The grammar of graphics does more than describe the components of a graph, it defines how these components can be combined to make useful images. Implementing subplots as a geom requires specific considerations when combining subplots with the technical details of the grammar of graphics. These details include stats, position adjustments, and reference objects, such as coordinate axes. In this section, we discuss these considerations and illustrate them with code from `ggsubplot`.

3.4.1.1 Geom

`ggsubplot` introduces two new geoms that draw embedded subplots. These geoms will serve as examples for the technical considerations in the remainder of this section. `geom_subplot` uses a *group* aesthetic to assign data to subplots and then positions each subplot based on a summary of its data points. `geom_subplot2d` bins the surface of a plot into a two dimensional grid and then represents each bin with a subplot. The use of these geoms is demonstrated in the example code below, which was used to create Figure 3.1.b and Figure 3.1.c.

```
## Figure 1.b, a glyphmap
ggplot(nasa) + map_americas +
  geom_subplot(aes(long, lat, group = id,
  subplot = geom_star(aes(r = surftemp, angle = date, fill =
mean(surftemp)),
  r.zero = FALSE, alpha = 0.75))) +
  coord_map()

## Figure 1.c, a binned plot
ggplot(ordinary.diamonds) +
  geom_subplot2d(aes(carat, price,
  subplot = geom_bar(aes(color, fill = color), position = "dodge")),
  bins = c(10, 14), y_scale = free, height.adjust = 0.8,
  width.adjust = 0.8, ref = ref_box(aes(color = length(color)))) +
  scale_color_gradient("Total\ncount", low = "grey70", high = "black")
```

3.4.1.2 Stats

A *stat* is any function that summarizes a group of data values into a smaller set of information. Stats and mappings form a two step processes whenever a single geom is used to display multiple data points. First, the stat summarizes the data points into summary level information. Then a mapping keys the aesthetics of the geom to the summary level information. For example, a boxplot geom uses a stat to calculate Tukey’s five number summary for a group of data points. Then the numbers are used to determine the location of each part of the boxplot. Specific geoms are usually associated with specific stats. Box plots always use a five number summary, histograms always use a bin and count procedure. These patterns allow users to largely ignore stats; software can automatically implement the correct stat once a geom is chosen. When a user does decide to specify a stat, they are usually constrained to choose from a prepackaged set of stat functions.

Embedded subplots also rely on stats, but subplots require more freedom in the choice of stat than is offered in current implementations of the grammar of graphics. Each subplot must map a group of data points to a single location on the x and y axes of the large plot. The way a user chooses to do this is likely to change from graph to graph. In Figure 3.1.a., the y position for each subplot is mapped to the mean value of CO_2 for the observations in the subplot. In Figure 3.1.b., both the x and y positions of each subplot are mapped to the common longitude and latitude of the observations within the subplot. In Figure 3.1.c., the x and y positions are mapped to the midpoint of the 2D bin that each group of observations has been assigned to. This variety prevents the subplot from relying on a set of prepackaged stats. Instead, users need the same freedom to create a stat as they have to create a mapping.

`geom_subplot` provides this freedom by having the mapping directly serve as a stat. If a mapping involves subsetting or a function that returns a single value (such

as a mean), it will perform its own summarizing. The user just needs to ensure that the mapping is applied separately to each group used in the graph. Otherwise, the mapping will be applied to the entire underlying data set at once and each geom will be keyed to the same information, for example, the mean of the entire data set. `ggsubplot` manages these requirements with the `ply_aes` function. `ply_aes` takes a `ggplot2` layer object and modifies it so that the layer's mappings are applied groupwise according to the layer's *group* aesthetic. `ply_aes` enforces summarization by subsetting the output of each mapping to just its first value. A warning message is given if the mapping would have otherwise returned multiple values. `geom_subplot` automatically uses `ply_aes`.

This arrangement provides a new insight into the relationship between mappings and stats. Mappings and stats perform the same function but are keyed to different levels in the hierarchy of information: individual level and group level. This parallelism is made clear in embedded plots. When we consider any single subplot in Cleveland's subcycle plot (Figure 3.1.b), the mean concentration of CO_2 is a groupwise statistic, (i.e., a stat) that summarizes an entire group of data. When our attention shifts to the higher level plot (Figure 3.1.c), the mean concentration of CO_2 becomes an aesthetic mapping of the subplots.

`ply_aes` can also be used with non-subplot layers. It behaves in the same way, turning individual mappings into groupwise mappings (i.e, stat + mapping). This technique replaces each group of geoms with a single geom that displays group level information. Figure 3.7 shows how this technique can remarkably reduce overplotting to reveal structure.

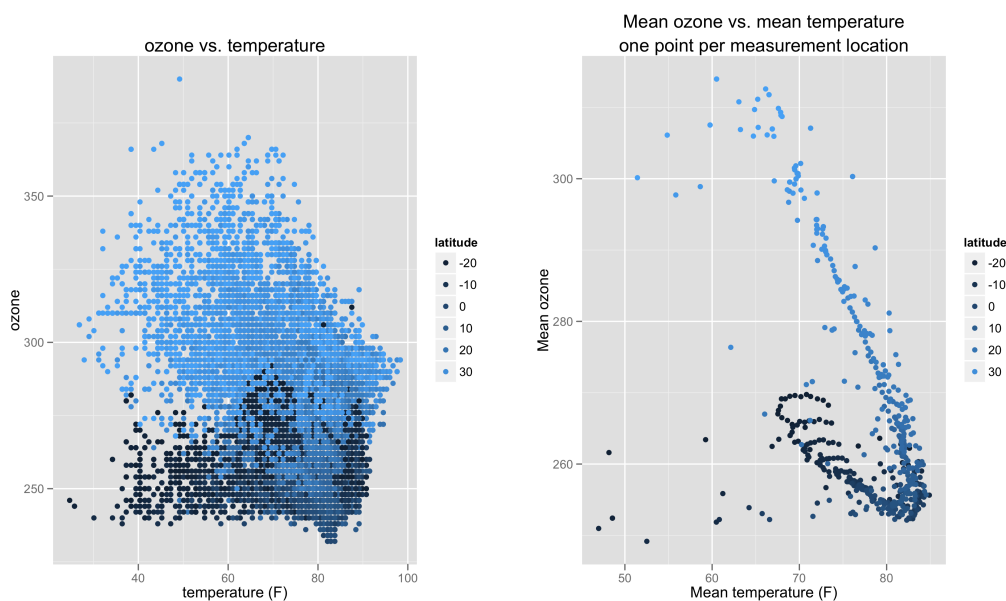


Figure 3.7: `ply_aes` offers a new strategy for overplotting. Groups of geoms are combined into single geoms that display summary information. This approach reveals that $mean(ozone)$ has a different linear relationship with temperature in the southern hemisphere than it does in the north.

3.4.1.3 Position adjustments

Many embedded plots will require nontraditional choices for a position adjustment. Each layer of a graphic contains a position adjustment that determines how to plot graphical elements that interfere with one another. Wilkinson and Wills (2005) refers to this concept as a collision modifier. Position adjustments are often implicitly set to identity, which means that elements will be plotted on top of one another if they overlap. Alternatively, overlapping elements can be adjusted to appear above each other (stacking), next to each other (dodging), in random nearby locations (jittering) or in other places. These solutions are inefficient for a large subset of embedded graphs.

Embedded graphics such as Figure 3.1.b and 3.1.c use the position of the subplot to signal which observations are included in the subplot. In Figure 3.1.b, every observation with a certain longitude and latitude is mapped to the subplot positioned

at that longitude and latitude. In Figure 3.1.c, every observation that falls within a 2D bin is mapped to the subplot positioned at that bin. Even Figure 3.1.a uses position along the x axis to signal which observations are included in which line graph. This arrangement does not appear in every embedded graphic, but it can be useful. Traditional position adjustments such as stacking, dodging, and jittering disrupt this relationship. We suggest a new position adjustment that preserves the relationship between position and group membership: when two subplots overlap one another they can be merged into a single subplot.

Programming a merge adjustment is more complicated than programming traditional position adjustments. The merge adjustment will affect stat values because it alters group membership. Therefore, it must be computed early in the building process for graphs. The merge adjustment also presents a second difficulty: how do we define which subplots should be merged? `ggsubplot` combines each clique of overlapping subplots into a single subplot positioned at the mean location of the clique. This works well when graphs are relatively sparse, but can remove an undesirable amount of visual real estate when graphs are dense, see Figure 3.8. Clustering methods may provide a more useful approach to identifying graphs to be merged. Future versions of `ggsubplot` will explore this approach, but the most useful ways of merging are likely to arise from observing the actual application of embedded plots in data analysis. As an alternative to merging, users who wish to avoid overlaps can grid the subplots within a graph with `geom_subplot2d`. This is not a position adjustment, but a way of assigning the group aesthetic. However, gridding guarantees that membership will be mapped to position without any overlaps.

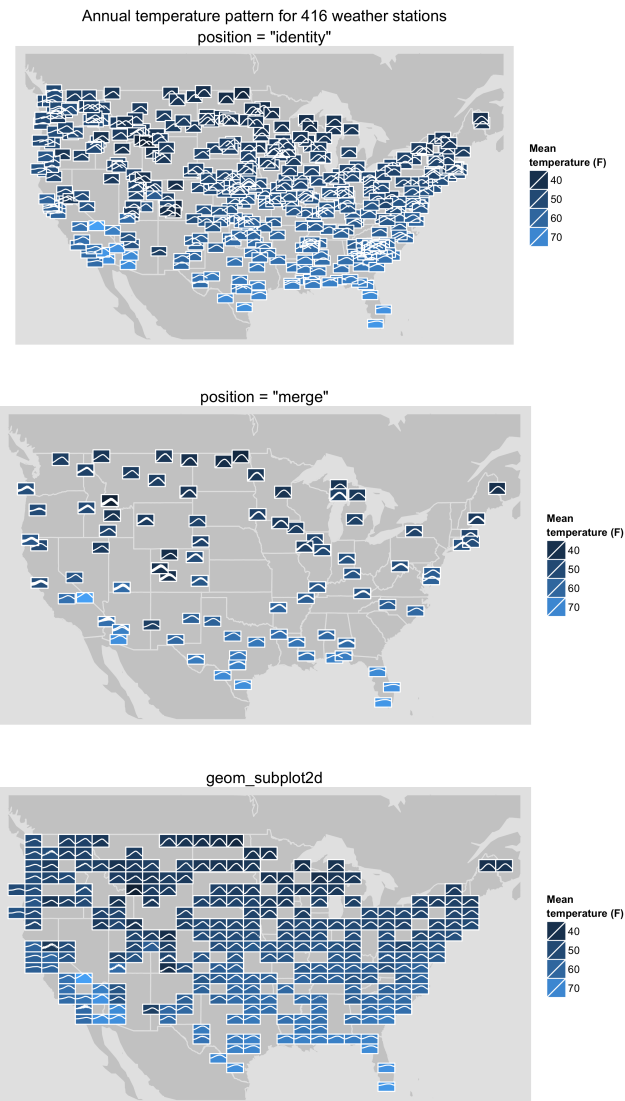


Figure 3.8: The position of a subplot is often related to which points the subplot shows. Position = merge and geom_subplot2d provide two ways to avoid overlapping subplots without disrupting this relationship.

3.4.1.4 Reference objects

Reference objects are any object that is added to each subplot to provide a standard of comparison across subplots. The axes of most graphs are one of the most commonly used type of reference object. However, axes are difficult to read at the small scales used in subplots. Boxes and lines can also allow comparison and scale better to the smaller sizes of subplots. `ggsubplot` creates these objects with a reference parameter in the subplot layer, see Figure 3.9.

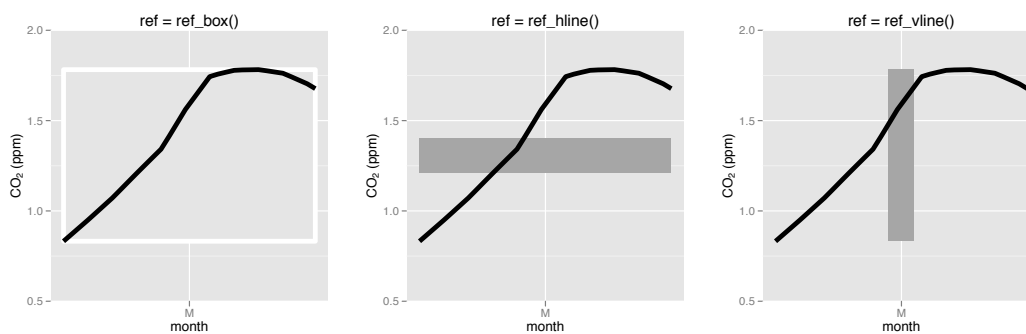


Figure 3.9: Reference objects allow comparison across subplots and can be more easily read at small scales than coordinate axes. In `ggsubplot`, users can add one of three types of reference objects to subplots by adding `ref = ref_box()`, `ref = ref_hline()`, or `ref = ref_vline()` to `geom_subplot` and `geom_subplot2d` calls.

These reference objects allow viewers to judge the position of geoms inside the subplot and to make comparisons against the position of geoms in other subplots. To allow accurate comparisons, the dimensions of reference objects do not vary across subplots. They are fixed to the dimensions of the subplot. However, other features of the reference object can vary to provide additional information about a subplot. For example, the fill, color, and transparency of a reference object can display group level information about the data in a subplot. The `ggsubplot` reference parameter allows users to set these aesthetics with the functions `ref_box`, `ref_vline` and `ref_hline`, see Figure 3.9. By default, `ref_box` displays with a grey background and white border. This matches the color scheme of `ggplot2`'s default background, while still

delineating the dimensions of the subplot. Reference objects provide a quick way to compare across subplots. However, if users require a precise judgement they should still plot the subplot in its own graph with a pair of axes.

3.4.2 Implications for the structure of graphics

Embedded plots demonstrate that graphics have a hierarchical, recursive structure. Graphs summarize information into an image, and images can themselves be summarized into a larger graph. This structure parallels a common pattern found in both human thought and data analysis, and suggests that graphs obey universal rules of data representation.

Many mental processes involve classifying, grouping, and aggregating. This is how we make sense of sensory information, and it is how we build data into information. At the cognitive level, the mind combines information in a number of ways. For example chunking, which extends our attentional resources, and building schemas, which assigns meaning to data. Even the sensory data that we collect is progressively aggregated and summarized as it travels through the neural network of our brain.

These mental processes guide the data collection process. As a result, they shape data that has been collected by, cleaned by, or manipulated by people. Measurements often do not directly collect information of interest. This information is built by grouping together similar observations and applying some aggregating function to the data. For example, a researcher may observe a subject's height, hair length, body shape and clothing choices and then aggregate them into a conclusion about the subject's gender. This pattern of grouping and summarizing is also an important strategy for dealing with variation, both random and systematic. The average of a groups of observations is less affected by random variation than a single observation.

Data built this way often has a hierarchical structure. A set of observations can

be summarized in a table of counts. A set of counts can be summarized by a single average. Averages from different populations can be compared against each other and so on. Moreover, a hierarchical structure can be built from any multivariable data set. A variable can be selected for sorting the data into groups. Groups can be made from all observations that share the same value of a discrete variable or all observations that fall into the same range of a categorical variable. Each group can be summarized in a variety of ways with a variety of functions to provide a higher level data set. Correlation between summary measures and grouping variables reveals structure in the data and hints at possible causal relationships. This process underlies the split-apply-combine strategy for data analysis that has been embedded into R as the `plyr` package (Wickham, 2011).

Mathematical functions are one way to summarize a group of data points. Graphics are another. Graphics organize data points into a meaningful summary that can be processed by the mind. This summary is the visual image of the graph. A graph resembles a cognitive schema because it puts various facts in relationship with one another and suggests meaning. A graph resembles a chunk because it can be more easily attended to, remembered, and analyzed than its constituent components. Graphs even have an efficiency advantage over mathematical functions at the cognitive level because they provide visual input. Embedded plots show that just as numerical summaries of data can themselves be summarized, graphical summaries of data points can be organized into higher level graphs.

Implementing embedded plots suggests that components of the graphics of grammar may originate from the universal, hierarchical structure of information. Aesthetic mappings create mappings between individual data points and visual aesthetics. Stats create mappings from group level information to visual aesthetics. They accomplish this by aggregating and summarizing the individual data points within a group of

data. In other words, stats and aesthetics perform the same function but are keyed to different levels in the hierarchy of a data set. This parallelism is made clear in embedded plots. When we consider any single subplot in Cleveland's subcycle plot, the mean concentration of CO_2 is a groupwise statistic, (i.e., a stat) that summarizes an entire group of data. When our attention shifts to the higher level plot, the mean concentration of CO_2 becomes an aesthetic mapping of the subplots.

Graphics, data, and human modes of information processing all utilize hierarchical structures of information. This universality suggests that a universal language of data representation may also exist. This idea is further attested to by the way each of these domains groups data points and summarizes these groups to obtain higher level data points. Identifying the rules of this universal language may make it easier to teach and practice data visualization and data analysis.

3.5 Conclusion

Embedded plots are a powerful visualization tool for many data analysis tasks. Because embedded plots organize multiple dimensions of data into a static two dimensional graph, they can provide insights not found in other types of graphics. Because embedded plots present complex data in a cognitive friendly way, they facilitate understanding that could not occur otherwise. Despite presenting more complex data, embedded plots are not much more complicated than other graphs. Embedded plots are a particularly useful data analysis tool when exploring spatio-temporal data and big data, which is subject to overplotting. Embedded plots also aid the exploration of interaction effects and second order relationships.

Subplots function the same way as geoms in the layered grammar of graphics. They provide a visual element whose appearance can be mapped to the values in an underlying data set. Because of this, embedded plots are easily accommodated by the

layered grammar of graphics. This paper demonstrates how methods for embedded plots can be programmed into software built on the layered grammar of graphics, such as `ggplot2`.

Extending the grammar of graphics to account for embedded plots also reveals a conceptual insight about graphics. Graphics have a hierarchical, or recursive, structure where plots can be organized into higher level plots. This ability to organize individual data points by group according to group-level summaries is a potentially useful feature of graphics that has not been well developed in statistical graphics. However, it does parallel principles of infoVis that recommend “Overview first, filter, zoom for details” (Shneiderman, 1996).

As useful as embedded plots can be, we do not recommend embedded plots for every situation. Subplots increase the complexity of a visual. They make it easy to create overwhelming, cluttered and uninterpretable graphs. We recommend the following guidelines for the effective use of embedded plots.

1. Do not use embedded plots when a simpler graph will suffice.
2. Give subplots just the elements necessary to convey the main idea of a graphic. Additional elements become distracting more quickly with embedded graphics than with simpler graphics.
3. Use subplots to highlight structure and pattern, not small details like individual values. Subplots are necessarily smaller than a full graph, which makes it harder to accurately perceive details (in accordance with Weber’s law). Subplots are fine for estimation and approximate arithmetic, which the mind seems to perform visually at the cognitive level anyways (Dehaene et al., 1999). But precise calculations require clear labels and numerical values. If detailed inspection is required, a subplot can and should be drawn by itself at full size.

These suggestions are meant to improve, and not prevent, the use of embedded plots. Embedded plots require good judgement in their use, but this is true of all graphs. Every graph should tell a clear story if it is to be useful, and embedded plots will often tell a more clear story than a simple graph plagued by overplotting or too few dimensions. As the examples in Section 5.1 illustrate, embedded plots can be powerfully useful in many contexts.

Dates and times made easy with lubridate

This chapter is reprinted from ‘Dates and times made easy with lubridate’ by Garrett Grolemund and Hadley Wickham, which appeared in The Journal of Statistical Software in 2011 (Grolemund and Wickham, 2011)

ABSTRACT — This paper presents the lubridate package for R (R Development Core Team, 2010), which facilitates working with dates and times. Date-times create various technical problems for the data analyst. The paper highlights these problems and offers practical advice on how to solve them using lubridate. The paper also introduces a conceptual framework for arithmetic with date-times in R.

4.1 Introduction

Date-time data can be frustrating to work with. Dates come in many different formats, which makes recognizing and parsing them a challenge. Will our program recognize the format that we have? If it does, we still face problems specific to date-times. How can we easily extract components of the date-times, such as years, months, or seconds? How can we switch between time zones, or compare times from

places that use daylight savings time (DST) with times from places that do not? Date-times create even more complications when we try to do arithmetic with them. Conventions such as leap years and DST make it unclear what we mean by “one day from now” or “exactly two years away.” Even leap seconds can disrupt a seemingly simple calculation. This complexity affects other tasks too, such as constructing sensible tick marks for plotting date-time data.

While Base R (R Development Core Team, 2010) handles some of these problems, the syntax it uses can be confusing and difficult to remember. Moreover, the correct R code often changes depending on the class of date-time object being used. `lubridate` acknowledges these problems and makes it easier to work with date-time data in R. It also provides tools for manipulating date-times in novel but useful ways. `lubridate` will enhance a user’s experience for any analysis that includes date-time data. Specifically, `lubridate` helps users:

- Identify and parse date-time data, see Section 4.3.
- Extract and modify components of a date-time, such as years, months, days, hours, minutes, and seconds, see Section 4.4.
- Perform accurate calculations with date-times and timespans, see Sections 4.5 and 4.6.
- Handle time zones and daylight savings time, see Sections 4.7 and 4.8.

`lubridate` uses an intuitive user interface inspired by the date libraries of object-oriented programming languages. `lubridate` methods are compatible with a wide-range of common date-time and time series objects. These include `character` strings, `POSIXct`, `POSIXlt`, `Date`, `chron` (James and Hornik, 2010), `fCalendar` (?), `zoo` (Zeileis and Grothendieck, 2005), `xts` (Ryan and Ulrich, 2010), `its` (Portfolio

and Risk Advisory Group, Commerzbank Securities, 2009), `tis` (Hallman, 2010), `timeSeries` (Wuertz and Chalabi, 2010), `fts` (Armstrong, 2009), and `tseries` (Trapletti and Hornik, 2009) objects.

Note that `lubridate` overrides the `+` and `-` methods for `POSIXt`, `Date`, and `difftime` objects in base R. This allows users to perform simple arithmetic on date-time objects with the new timespan classes introduced by `lubridate`, but it does not alter the way R implements addition and subtraction for non-`lubridate` objects.

`lubridate` introduces four new object classes based on the Java language Joda Time project (Colebourne and O’Neill, 2010). Joda Time introduces a conceptual model of the different ways to measure timespans. Section 4.5 describes this model and explains how `lubridate` uses it to perform easy and accurate arithmetic with dates in R.

This paper demonstrates the convenient tools provided in the `lubridate` package and ends with a case study, which uses `lubridate` in a real life example. This paper describes `lubridate` 0.2, which can be downloaded from the Comprehensive R Archive Network at <http://CRAN.R-project.org/package=lubridate>. Development versions can be found at <http://github.com/hadley/lubridate>.

4.2 Motivation

To see how `lubridate` simplifies things, consider a common scenario. Given a character string, we would like to read it in as a date-time, extract the month, and change it to February (i.e, 2). Table 4.1 shows two ways we could do this. On the left are the base R methods we would use for these three tasks. On the right are the `lubridate` methods.

Now we will go a step further. In Table 4.2, we move our date back in time by one day and display our new date in the Greenwich Meridian time zone (GMT). Again,

Base R method	lubridate method
<code>date <- as.POSIXct("01-01-2010", format = "%d-%m-%Y", tz = "UTC")</code>	<code>date <- dmy("01-01-2010")</code>
<code>as.numeric(format(date, "%m")) # or as.POSIXlt(date)\$month + 1</code>	<code>month(date)</code>
<code>date <- as.POSIXct(format(date, "%Y-2-%d"), tz = "UTC")</code>	<code>month(date) <- 2</code>

Table 4.1: `lubridate` provides a simple way to parse a date into R, extract the month value and change it to February.

Base R method	lubridate method
<code>date <- seq(date, length = 2, by = "-1 day")[2]</code>	<code>date <- date - days(1)</code>
<code>as.POSIXct(format(as.POSIXct(date), tz = "UTC"), tz = "GMT")</code>	<code>with_tz(date, "GMT")</code>

Table 4.2: `lubridate` easily displays a date one day earlier and in the GMT time zone.

base R methods are shown on the left, `lubridate` methods on the right.

`lubridate` makes basic date-time manipulations much more straightforward. Plus, the same `lubridate` methods work for most of the popular date-time object classes (`Date`, `POSIXt`, `chron`, etc.), which isn't always true for base R methods.

Table 4.3 provides a more complete comparison between `lubridate` methods and base R methods. It shows how `lubridate` can simplify each of the common date-time tasks presented in the article “Date and Time Classes in R” (Grothendieck and Petzoldt, 2004). It also provides a useful summary of `lubridate` methods.

Task	lubridate	Date	POSIXct
now (system time zone) now (GMT) origin x days since origin next day previous day	now() now("GMT") origin origin + days(x) date + days(1) date - days(1)	Sys.Date() structure(0, class = "Date") date + 1 date - 1	Sys.time() structure(0, class = c("POSIXt", "POSIXct")) structure(x*24*60*60, class=c("POSIXt", "POSIXct")) seq(date, length = 2, by = "day") [2] seq(date, length = 2, by = "-1 day") [2]
x days since date (day exactly 24 hours) (allowing for DST) display date in new time zone keep clock time, replace time zone	date + ddays(x) date + days(x) with_tz(date, "TZ") force_tz(date, tz = "TZ")	date + floor(x) DST and time zones	seq(date, length = 2, by = paste(x, "day")) [2] seq(date, length = 2, by = paste(x, "DSTday")) [2] as.POSIXct(format(as.POSIXct(date), tz = "TZ"), tz = "TZ") Exploring seq(date, length = 10, by = "day") seq(date, length = 3, by = "2 week") as.Date(format(date, "%Y-%m-01")) as.numeric(format(date, "%Y")) as.Date(format(date, "z-%m-%d")) as.numeric(format(date, "%w")) # Sun = 0 as.numeric(format(date, "%j"))
sequence every 2nd week first day of month round to nearest first of month extract year value change year value day of week day of year express as decimal of year	date + c(0:9) * days(1) date + c(0:2) * weeks(2) floor_date(date, "month") round_date(date, "month") year(date) year(date) < - z yday(date) # Sun = 1 yday(date) decimal_date(date)	seq(date, length = 10, by = "day") seq(date, length = 3, by = "2 week") as.Date(format(date, "%Y-%m-01")) as.numeric(format(date, "%Y")) as.Date(format(date, "z-%m-%d")) as.numeric(format(date, "%w")) # Sun = 0 as.numeric(format(date, "%j"))	seq(date, length = 10, by = "DSTday") seq(date, length = 3, by = "2 week") as.POSIXct(format(date, "%Y-%m-01")) as.numeric(format(date, "%Y")) as.POSIXct(format(date, "z-%m-%d")) as.numeric(format(date, "%w")) # Sun = 0 as.numeric(format(date, "%j"))
z = "1970-10-15" z = "10/15/1970" z = 15101970	ymd(z) mdy(z) dmy(z)	as.Date(z) as.Date(z, "%m/%d/%Y") as.Date(as.character(z), format = "%d/%m/%Y")	as.POSIXct(z) as.POSIXct(strptime(z, "%m/%d/%Y")) as.POSIXct(as.character(z), tz = "GMT", format = "%d/%m/%Y")
Duration	lubridate	base R	Durations Comparison
1 second 5 days, 3 hours and - 1 minute 1 month 1 year	seconds(1) new_duration(day = 5, hour = 3, minute = -1) months(1) years(1)	as.difftime(1, unit = "secs") as.difftime(60 * 24 * 5 + 60 * 3 - 1, unit = "mins") # Time difference of 7379 mins	

Table 4.3: lubridate provides a simple alternative for many date and time related operations. Table adapted from Grothendieck and Petzoldt (2004).

4.3 Parsing date-times

We can read dates into R using the `ymd()` series of functions provided by `lubridate`. These functions parse character strings into dates. The letters `y`, `m`, and `d` correspond to the year, month, and day elements of a date-time. To read in a date, choose the function name that matches the order of elements in your date-time object. For example, in the following date the month element comes first, followed by the day and then the year. So we would use the `mdy()` function:

```
R> mdy("12-01-2010")
```

```
[1] "2010-12-01 UTC"
```

The same character string can be parsed as January 12, 2001 by reversing the month and day element with `dmy()`.

```
R> dmy("12-01-2010")
```

```
[1] "2010-01-12 UTC"
```

The `ydm()` series of functions can also parse vectors of dates.

```
R> dmy(c("31.12.2010", "01.01.2011"))
```

```
[1] "2010-12-31 UTC" "2011-01-01 UTC"
```

These functions create a `POSIXct` date-time object that matches the date described by the character string. The functions automatically recognize the separators commonly used to record dates. These include: `"-"`, `"/"`, `"."`, and `""` (i.e., no separator). When a `ymd()` function is applied to a vector of dates, `lubridate` will assume that all of the dates have the same order and the same separators. `ymd()` type functions

Order of elements in date-time	Parse function
year, month, day	<code>ymd()</code>
year, day, month	<code>ydm()</code>
month, day, year	<code>mdy()</code>
day, month, year	<code>dmy()</code>
hour, minute	<code>hm()</code>
hour, minute, second	<code>hms()</code>
year, month, day, hour, minute, second	<code>ymd_hms()</code>

Table 4.4: Parse function names are based on the order that years, months, and days appear within the dates to be parsed.

also exist for times recorded with hours, minutes, and seconds. Hour, minute, and second measurements that are not accompanied by a date will be parsed as `period` objects, which are a type of timespan object, see Section 4.5.4. These functions make it simple to parse any date-time object that can be converted to a character string. See Table 4.4 for a complete list of `ymd()` type parsing functions.

4.4 Manipulating date-times

Every date-time is a combination of different elements, each with its own value. For example, most date-times include a year value, a month value, a day value and so on. Together these elements specify the exact moment that the date-time refers to. We can easily extract each element of a date-time with the accessor function that has its name, as shown in Table 4.5. For example, if we save the current system time

```
R> date <- now()
```

```
[1] "2010-02-25 09:51:48 CST"
```

we can extract each of its elements. Note that this was the system time when this example was written. `now()` will return a different date-time each time it is used.

```
R> year(date)
```

```
[1] 2010
```

```
R> minute(date)
```

```
[1] 51
```

For the month and weekday elements (`wday`), we can also specify whether we want to extract the numerical value of the element, an abbreviation of the name of the month or weekday, or the full name. For example,

```
R> month(date)
```

```
[1] 2
```

```
R> month(date, label = TRUE)
```

```
[1] Feb
```

```
R> month(date, label = TRUE, abbr = FALSE)
```

```
[1] February
```

```
R> wday(date, label = TRUE, abbr = FALSE)
```

```
[1] Thursday
```

We can also use any of the accessor functions to set the value of an element. This would also change the moment that the date-time refers to. For example,

```
R> day(date) <- 5
```

```
[1] "2010-02-05 09:51:48 CST"
```

Date component	Accessor
Year	<code>year()</code>
Month	<code>month()</code>
Week	<code>week()</code>
Day of year	<code>yday()</code>
Day of month	<code>mday()</code>
Day of week	<code>wday()</code>
Hour	<code>hour()</code>
Minute	<code>minute()</code>
Second	<code>second()</code>
Time zone	<code>tz()</code>

Table 4.5: Each date-time element can be extracted with its own accessor function.

changes our date to the fifth day of the month. We can also set the elements to more complicated values, e.g.

```
R> dates <- ymd_hms("2010-01-01 01:00:00", "2010-01-01 01:30:00")
R> minute(dates) <- mean(minute(dates))

[1] "2010-01-01 01:15:00 UTC" "2010-01-01 01:15:00 UTC"
```

Note that if we set an element to a larger value than it supports, the difference will roll over into the next higher element. For example,

```
R> day(date) <- 30

[1] "2010-03-02 09:51:48 CST"
```

Setting the date elements provides one easy way to find the last day of a month. An even easier method is described in Section 4.6.

```
R> day(date) <- 1
R> month(date) <- month(date) + 1
R> day(date) <- day(date) - 1
```

```
[1] "2010-03-31 09:51:48 CDT"
```

Lubridate also provides an update method for date-times. This is useful if you want to change multiple attributes at once or would like to create a modified copy instead of transforming in place.

```
R> update(date, year = 2010, month = 1, day = 1)
```

```
[1] "2010-01-01 09:51:48 CST"
```

Finally, we can also change dates by adding or subtracting units of time from them. For example, the methods below produce the same result.

```
R> hour(date) <- 12
```

```
[1] "2010-02-25 12:51:48 CST"
```

```
R> date <- date + hours(3)
```

```
[1] "2010-02-25 12:51:48 CST"
```

Notice that `hours()` (plural) is not the same function as `hour()` (singular). `hours()` creates a new object that can be added or subtracted to a date-time. These objects are discussed in the next section.

4.5 Arithmetic with date-times

Arithmetic with date-times is more complicated than arithmetic with numbers, but it can be done accurately and easily with `lubridate`. What complicates arithmetic with date-times? Clock times are periodically re-calibrated to reflect astronomical conditions, such as the hour of daylight or the Earth's tilt on its axis relative to the

sun. We know these re-calibrations as daylight savings time, leap years, and leap seconds. Consider how one of these conventions might complicate a simple addition task. If today were January 1st, 2010 and we wished to know what day it would be one year from now, we could simply add 1 to the years element of our date.

January 1st, 2010 + 1 year = January 1st, 2011

Alternatively, we could add 365 to the days element of our date because a year is equivalent to 365 days.

January 1st, 2010 + 365 days = January 1st, 2011

Troubles arise if we try the same for January 1st, 2012. 2012 is a leap year, which means it has an extra day. Our two approaches above now give us different answers because the length of a year has changed.

January 1st, 2012 + 1 year = January 1st, 2013

January 1st, 2012 + 365 days = December 31st, 2012

At different moments in time, the lengths of months, weeks, days, hours, and even minutes will also vary. We can consider these to be *relative* units of time; their length is relative to when they occur. In contrast, seconds always have a consistent length. Hence, seconds are *exact* units of time.

Researchers may be interested in exact lengths, relative lengths, or both. For example, the speed of a physical object is most precisely measured in exact lengths. The opening bell of the stock market is more easily modeled with relative lengths.

`lubridate` allows arithmetic with both relative and exact units by introducing four new time related objects. These are *instants*, *intervals*, *durations*, and *periods*. These concepts are borrowed from the `Joda Time` project (Colebourne and O’Neill, 2010). Similar concepts for instants, periods, and durations also appear in the C++ library `Boost.Date.Time` (Garland, 2011). `lubridate` provides helper functions, object classes and methods for using all four concepts in the R language.

4.5.1 Instants

An instant is a specific moment in time, such as January 1st, 2012. We create an instant each time we parse a date into R.

```
R> start_2012 <- ymd_hms("2012-01-01 12:00:00")
```

`lubridate` does not create a new class for instant objects. Instead, it recognizes any date-time object that refers to a moment of time as an instant. We can test if an object is an instant by using `is.instant()`. For example,

```
R> is.instant(364)
```

```
[1] FALSE
```

```
R> is.instant(start_2012)
```

```
[1] TRUE
```

We can also capture the current time as an instant with `now()`, and the current day with `today()`.

4.5.2 Intervals

Intervals, durations, and periods are all ways of recording timespans. Of these, intervals are the most simple. An interval is a span of time that occurs between two specific instants. The length of an interval is never ambiguous, because we know when it occurs. Moreover, we can calculate the exact length of any unit of time that occurs during it. `lubridate` introduces the `interval` object class for modelling intervals.

We can create `interval` objects by subtracting two instants or by using the command `new_interval()`.

```
R> start_2011 <- ymd_hms("2011-01-01 12:00:00")
R> start_2010 <- ymd_hms("2010-01-01 12:00:00")
R> span <- start_2011 - start_2010
```

```
[1] 2010-01-01 12:00:00 -- 2011-01-01 12:00:00
```

We can access the start and end dates of an `interval` object with `int_start()` and `int_end()`. Intervals always begin at the date-time that occurs first and end at the date-time that occurs last. Hence, intervals always have a positive length.

```
R> int_start(span)
```

```
[1] "2010-01-01 12:00:00 UTC"
```

```
R> int_end(span)
```

```
[1] "2011-01-01 12:00:00 UTC"
```

Unfortunately, since intervals are anchored to their start and end dates, they are not very useful for date-time calculations. It only makes sense to add an interval to its start date or to subtract it from its end date.

```
R> start_2010 + span
```

```
[1] "2011-01-01 12:00:00 UTC"
```

Adding intervals to other date-times won't produce an error message. Instead `lubridate` will coerce the interval to a `duration` object, which is like an interval but without the reference dates. See Section 4.5.3.

```
R> start_2011 + span
```

```
coercing interval to duration
```

```
[1] "2012-01-01 12:00:00 UTC"
```

In most cases this will return the intended result, but accuracy can be ensured by first explicitly converting the interval to either a `duration` or a `period`, as described in the next two sections.

We can convert any other type of timespan to an interval by pairing it to a start date with `as.interval()`. For example:

```
R> as.interval(difftime(start_2011, start_2010), ymd("2010-03-05"))
```

```
[1] 2010-03-05 -- 2011-03-05
```

4.5.3 Durations

If we remove the start and end dates from an interval, we will have a generic time span that we can add to any date. But how should we measure this length of time? If we record the time span in seconds, it will have an exact length since seconds always have the same length. We call such time spans *durations*. Alternatively, we can record the time span in larger units, such as minutes or years. Since the length of these units varies over time, the exact length of the time span will depend on when

it begins. These non-exact time spans are called *periods* and will be discussed in the next section.

The length of a duration is invariant to leap years, leap seconds, and daylight savings time because durations are measured in seconds. Hence, durations have consistent lengths and can be easily compared to other durations. Durations are the appropriate object to use when comparing time based attributes, such as speeds, rates, and lifetimes. `difftime` objects from base R are one type of duration object. `lubridate` provides a second type: `duration` class objects. These objects can be used with other date-time objects without worrying about what units they are displayed in. A `duration` object can be created with the function `new_duration()`:

```
R> new_duration(60)
```

```
[1] 60s
```

For large durations, it becomes inconvenient to describe the length in seconds. For example, not many people would recognize 31557600 seconds as the length of a standard year. For this reason, large `duration` objects are followed in parentheses by an estimated length. Estimated units are created using the following relationships. A minute is 60 seconds, an hour 3600 seconds, a day 86400, a week 604800, and a year 31557600 (365.25 days). Month units are not used because they are so variable. The estimates are only provided for convenience; the underlying object is always recorded as a fixed number of seconds.

`duration` objects can be easily created with the helper functions `dyears()`, `dweeks()`, `ddays()`, `dhours()`, `dminutes()`, and `dseconds()`. The `d` in the title stands for duration and distinguishes these objects from `period` objects, which are discussed in Section 4.5.4. Each object creates a duration in seconds using the estimated relationships given above. The argument of each function is the number of estimated units we wish to include in the duration. For example,

```
R> dminutes(1)
```

```
[1] 60s
```

```
R> dseconds(60)
```

```
[1] 60s
```

```
R> dminutes(2)
```

```
[1] 120s
```

```
R> c(1:3) * dhours(1)
```

```
[1] 3600s (1h) 7200s (2h) 10800s (3h)
```

Durations can be added and subtracted to any instant object. For example,

```
R> start_2011 + dyears(1)
```

```
[1] "2012-01-01 12:00:00 UTC"
```

```
R> start_2012 <- ymd_hms("2012-01-01 12:00:00")
```

```
R> start_2012 + dyears(1)
```

```
[1] "2012-12-31 12:00:00 UTC"
```

Durations can also be added to or subtracted from intervals and other durations. For example,

```
R> dweeks(1) + ddays(6) + dhours(2) + dminutes(1.5) + dseconds(3)
```

```
[1] 1130493s (13.08d)
```

We can also create durations from `interval` and `period` objects using `as.duration()`.

```
R> as.duration(span)
```

```
[1] 31536000s (365d)
```

4.5.4 Periods

Periods record a time span in units larger than seconds, such as years, months, weeks, days, hours, and minutes. For convenience, we can also create a period that only uses seconds, but such a period would have the same properties as a duration. `lubridate` introduces the `period` class to model periods. We construct `period` objects with the helper functions `years()`, `months()`, `weeks()`, `days()`, `hours()`, `minutes()`, and `seconds()`.

```
R> months(3)
```

```
[1] 3 months
```

```
R> months(3) + days(2)
```

```
[1] 3 months and 2 days
```

These functions do not contain a “d” in their name, because they do not create durations; they no longer have consistent lengths (as measured in seconds). For example, `months(2)` always has the length of two months even though the length of two months will change depending on when the period begins. For this reason, we can not compute exactly how long a period will be in seconds until we know when it occurs. However, we can still perform date-time calculations with periods. When we add or subtract a period to an instant, the period becomes anchored to the instant. The instant tells us when the period occurs, which allows us to calculate its exact length in seconds.

As a result, we can use periods to accurately model clock times without knowing when events such as leap seconds, leap days, and DST changes occur.

```
R> start_2012 + years(1)
```

```
[1] "2013-01-01 12:00:00 UTC"
```

vs.

```
R> start_2012 + dyears(1)
```

```
[1] "2012-12-31 12:00:00 UTC"
```

We can also convert other timespans to `period` objects with the function `as.period()`.

```
R> as.period(span)
```

```
[1] 1 year
```

Periods can be added to instants, intervals, and other periods, but not to durations.

4.5.5 Division with timespans

We often wish to answer questions that involve dividing one timespan by another. For example, “how many weeks are there between Halloween and Christmas?” or “how old is a person born on September 1, 1976?” Objects of each timespan class—`interval`, `duration`, and `period`—can be divided by objects of the others. The results of these divisions varies depending on the nature of the timespans involved. Modulo operators (i.e, `%%` and `%/%`) also work between timespan classes.

To illustrate this, we make an interval that lasts from Halloween until Christmas.

```
R> halloween <- ymd("2010-10-31")
```

```
R> christmas <- ymd("2010-12-25")
```

```
R> interval <- new_interval(halloween, christmas)
```

```
[1] 2010-10-31 -- 2010-12-25
```

Since durations are an exact measurement of a timespan, we can divide this interval by a duration to get an exact answer.

```
R> interval / dweeks(1)
```

```
[1] 7.857143
```

Intervals are also exact measures of timespans. Although it is more work, we could divide an interval by another interval to get an exact answer. This gives the same answer as above because `lubridate` automatically coerces `interval` objects in the denominator to `duration` objects.

```
R> interval / new_interval(halloween, halloween + weeks(1))
```

```
interval denominator coerced to duration
```

```
[1] 7.857143
```

Division is not possible with periods. Since periods have inconsistent lengths, we can not express the remainder as a decimal. For example, if we were to divide our interval by months, the remainder would be 24 days.

```
R> interval \% \% months(1)
```

```
[1] 24 days
```

Should we calculate 24 days as $24/30 = 0.8$ months since November has 30 days? Or should we calculate 24 days as $24/31 = 0.77$ months since December has 31 days? Both November and December are included in our numerator. If we want an exact calculation, we should use a duration instead of a period.

	Instant	Interval	Duration	Period
Instant	N/A	instant	instant	instant
Interval	instant	interval ¹	interval	interval
Duration	instant	interval	duration	period
Period	instant	interval	period	period

¹a duration if the intervals do not align

Table 4.6: Adding two date-time objects will create the above type of object.

If we attempt to divide by a `period` object, `lubridate` will return a warning message and automatically perform integer division: it returns the integer value equal to the number of whole periods that occur in the timespan. This is equivalent to the `%/%` operator. As shown above, we can retrieve the remainder using the `%%` operator.

```
R> interval / months(1)
```

```
estimate only: convert periods to intervals for accuracy
```

```
[1] 1
```

```
R> interval %%\ months(1)
```

```
[1] 1
```

In summary, arithmetic with date-times involves four types of objects: instants, intervals, durations, and periods. `lubridate` creates new object classes for intervals, durations, and periods. It recognizes that most common date-time classes, such as `POSIXt` and `Date`, refer to instants. Table 4.6 describes which objects can be added to each other and what type of object will result.

4.6 Rounding dates

Like numbers, date-times occur in order. This allows date-times to be rounded. `lubridate` provides three methods that help perform this rounding: `round_date()`,

`floor_date()`, and `ceiling_date()`. The first argument of each function is the date-time or date-times to be rounded. The second argument is the unit to round to. For example, we could round April 20, 2010 to the nearest day:

```
R> april20 <- ymd_hms("2010-04-20 11:33:29")
```

```
R> round_date(april20, "day")
```

```
[1] "2010-04-20 UTC"
```

or the nearest month.

```
R> round_date(april20, "month")
```

```
[1] "2010-05-01 UTC"
```

Notice that rounding a date-time to a unit sets the date to the start of that unit (e.g, `round_date(april20, "day")` sets the hours, minutes, and seconds information to 00).

`ceiling_date()` provides a second simple way to find the last day of a month. Ceiling the date to the next month and then subtract a day.

```
R> ceiling_date(april20, "month") - days(1)
```

```
[1] "2010-04-30 UTC"
```

4.7 Time zones

Time zones give multiple names to the same instant. For example, “2010-03-26 11:53:24 CDT” and “2010-03-26 12:53:24 EDT” both describe the same instant. The first shows how the instant is labeled in the United States’ central time zone (CDT). The second shows how the same instant is labelled in the United States’ eastern time

zone (EDT). Time zones complicate date-time data but are useful for mapping clock time to local daylight conditions. When working with instants, it is standard to give the clock time as it appears in the Coordinated Universal time zone (UTC). This saves calculations but can be annoying if your computer insists on translating times to your current time zone. It may also be inconvenient to discuss clock times that occur in a place unrelated to the data.

`lubridate` eases the frustration caused by time zones in two ways. We can change the time zone in which an instant is displayed by using the function `with_tz()`. This changes how the clock time is displayed, but not the specific instant of time that is referred to. For example,

```
R> date
```

```
[1] "2010-01-01 09:51:48 CST"
```

```
R> with_tz(date, "UTC")
```

```
[1] "2010-01-01 15:51:48 UTC"
```

`force_tz()` does the opposite of `with_tz()`: it changes the actual instant of time saved in the object, while keeping the displayed clock time the same. The new time zone value alerts us to this change. For example, the code below moves us to a new instant that occurs 6 hours earlier.

```
R> date
```

```
[1] "2010-01-01 09:51:48 CST"
```

```
R> force_tz(date, "UTC")
```

```
[1] "2010-01-01 09:51:48 UTC"
```

`with_tz()` and `force_tz()` only work with time zones recognized by your computer's operating system. This list of time zones will vary between computers. See the base R help page for `Sys.timezone()` for more information.

4.8 Daylight savings time

In many parts of the world, the official clock time springs forward by one hour in the spring and falls back one hour in the fall. For example, in Chicago, Illinois a change in daylight savings time occurred at 2:00 AM on March 14, 2010. The last instant to occur before this change was 2010-03-14 01:59:59 CST.

```
R> dst_time <- ymd_hms("2010-03-14 01:59:59")
R> dst_time <- force_tz(dst_time, "America/Chicago")
```

```
[1] "2010-03-14 01:59:59 CST"
```

One second later, Chicago clock times read

```
R> dst_time + dseconds(1)
```

```
[1] "2010-03-14 03:00:00 CDT"
```

It seems that we gained an extra hour during that second, which is how daylight savings time works. We can completely avoid the changes in clock time caused by daylight savings times by working with periods instead of durations. For example,

```
R> dst_time + hours(2)
```

```
[1] "2010-03-14 03:59:59 CDT"
```

displays the clock time that usually occurs two hours after 1:59:59 AM. When using periods, we do not have to track DST changes because they will not affect our calculations. This will prevent errors when we are trying to model events that depend on clock times, such as the opening and closing bells of the New York Stock Exchange. Adding a duration would give us the actual clock time that appeared exactly two hours later on March 14, 2010. This prevents changes in clock time from interfering with exact measurements, such as the life times of a light bulb, but will create surprises if clock times are important to our analysis.

```
R> dst_time + dhours(2)
```

```
[1] "2010-03-14 04:59:59 CDT"
```

If we ever inadvertently try to create an instant that occurs in the one hour “gap” between 2010-03-14 01:59:59 CST and 2010-03-14 03:00:00 CDT, `lubridate` will return `NA` since such times are not available.

We can also avoid the complications created by daylight savings time by keeping our date-times in a time zone such as “UTC”, which does not adopt daylight savings hours.

4.9 Case study 1

The next two sections will work through some techniques using `lubridate`. First, we will use `lubridate` to calculate the dates of holidays. Then we will use `lubridate` to explore an example data set (`lakers`).

4.9.1 Thanksgiving

Some holidays, such as Thanksgiving (U.S.) and Memorial Day (U.S.) do not occur on fixed dates. Instead, they are celebrated according to a common rule. For example,

Thanksgiving is celebrated on the fourth Thursday of November. To calculate when Thanksgiving will occur in 2010, we can start with the first day of 2010.

```
R> date <- ymd("2010-01-01")
```

```
[1] "2010-01-01 UTC"
```

We can then add 10 months to our date, or directly set the date to November.

```
R> month(date) <- 11
```

```
[1] "2010-11-01 UTC"
```

We check which day of the week November 1st is.

```
R> wday(date, label = TRUE, abbr = FALSE)
```

```
[1] Monday
```

This implies November 4th will be the first Thursday of November.

```
R> date <- date + days(3)
```

```
[1] "2010-11-04 UTC"
```

```
R> wday(date, label = TRUE, abbr = FALSE)
```

```
[1] Thursday
```

Next, we add three weeks to get to the fourth Thursday in November, which will be Thanksgiving.

```
R> date + weeks(3)
```

```
[1] "2010-11-25 UTC"
```

4.9.2 Memorial Day

Memorial day also occurs according to a rule; it falls on the last Monday of May. To calculate the date of Memorial day, we can again start with the first of the year.

```
R> date <- ymd("2010-01-01")
```

```
[1] "2010-01-01 UTC"
```

Next, we set the month to May.

```
R> month(date) <- 5
```

```
[1] "2010-05-01 UTC"
```

Now our holiday occurs in relation to the last day of the month instead of the first. We find the last day of the month by rounding up to the next month and then subtracting a day.

```
R> date <- ceiling_date(date, "month") - days(1)
```

```
[1] "2010-05-31 UTC"
```

We can then check which day of the week May 31st falls on. It happens to be a Monday, so we are done. If May 31st had been another day of the week, we could've subtracted an appropriate number of days to get to the last Monday of May.

```
R> wday(date, label = TRUE, abbr = FALSE)
```

```
[1] Monday
```

4.10 Case study 2

The `lakers` data set contains play by play statistics of every major league basketball game played by the Los Angeles Lakers during the 2008-2009 season. This data is from <http://www.basketballgeek.com/downloads/> (Parker, 2010) and comes with the `lubridate` package. We will explore the distribution of Lakers' games throughout the year as well as the distribution of plays within Lakers' games. We choose to use the `ggplot2` (Wickham, 2009) package to create our graphs.

The `lakers` data set comes with a `date` variable which records the date of each game. Using the `str()` command, we see that R recognizes the dates as integers.

```
R> str(lakers$date)
```

```
int [1:34624] 20081028 20081028 20081028 ...
```

Before we can work with them as dates, we must parse them into R as date-time objects. The dates appear to be arranged with their year element first, followed by the month element, and then the day element. Hence, we should use the `ymd()` parsing function.

```
R> lakers$date <- ymd(lakers$date)
```

```
R> str(lakers$date)
```

```
POSIXct [1:34624], format: "2008-10-28" "2008-10-28" ...
```

R now recognizes the dates as `POSIXct` date-time objects. It will now treat them as date-times in any functions that have `POSIXct` specific methods. For example, if we plot the occurrences of home and away games throughout the season, our x axis will display date-time information for the tick marks (Figure 4.1).

```
R> qplot(date, 0, data = lakers, colour = game_type)
```

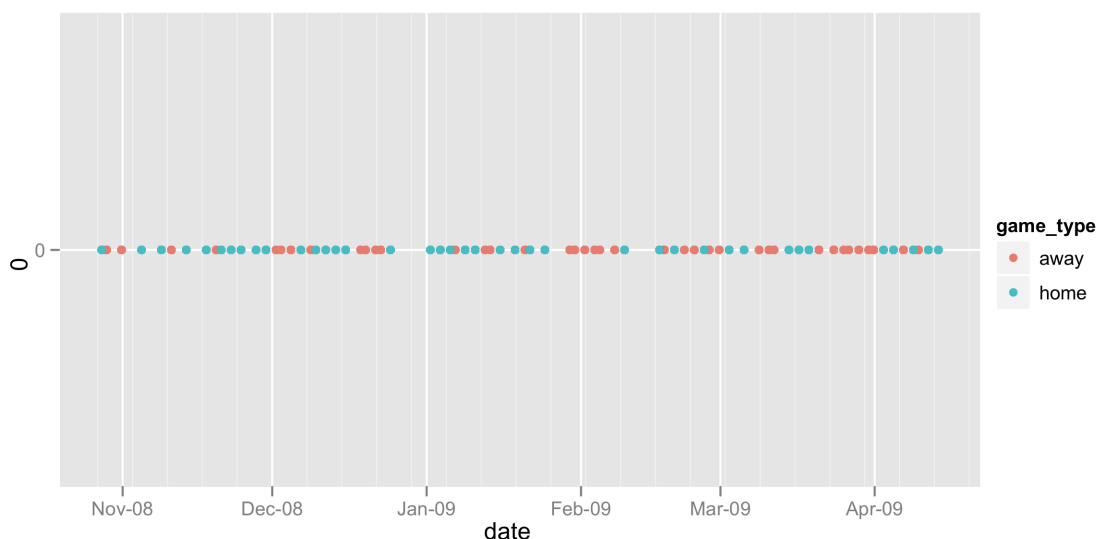


Figure 4.1: Home games and away games appear to occur in clusters.

Figure 4.1 shows that games are played continuously throughout the season with a few short breaks. The frequency of games seems lower at the start of the season and games appear to be grouped into clusters of home games and away games. The tick marks and breaks on the x axis are automatically generated by the `lubridate` method `pretty.dates()`.

Next we will examine how Lakers games are distributed throughout the week. We use the `wday()` command to extract the day of the week of each date.

```
R> qplot(wday(date, label = TRUE, abbr = FALSE), data = lakers,
        geom = "histogram")
```

The frequency of basketball games varies throughout the week (Figure 4.2). Surprisingly, the highest number of games are played on Tuesdays.

Now we look at the games themselves. In particular, we look at the distribution of plays throughout the game. The `lakers` data set lists the time that appeared on the game clock for each play. These times begin at 12:00 at the beginning of each period and then count down to 00:00, which marks the end of the period. The first

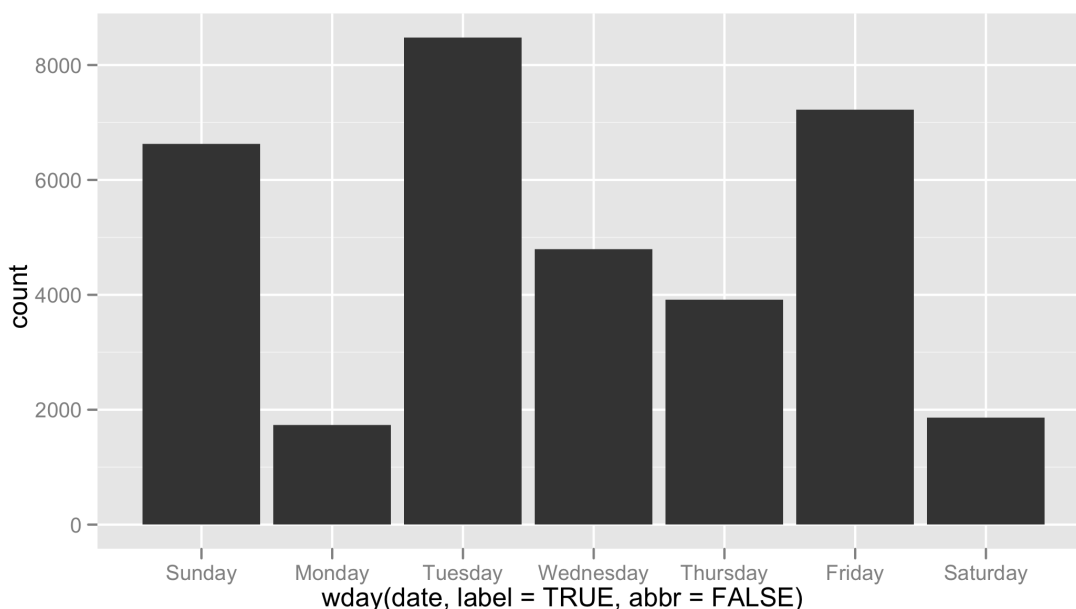


Figure 4.2: More games occur on Tuesdays than any other day of the week.

two digits refer to the number of minutes left in the period. The second two digits refer to the number of seconds.

The times have not been parsed as date-time data to R. It would be difficult to record the time data as a date-time object because the data is incomplete: a minutes and seconds element are not sufficient to identify a unique instant of time. However, we can store the minutes and seconds information as a *period* object, as defined in Section 4.5.4, using the `ms()` parse function.

```
R> lakers$time <- ms(lakers$time)
```

Since periods have relative lengths, it is dangerous to compare them to each other. So we should next convert our periods to *durations*, which have exact lengths.

```
R> lakers$time <- as.duration(lakers$time)
```

This allows us to directly compare different durations. It would also allow us to determine exactly when each play occurred by adding the duration to the *instant* the

game began. (Unfortunately, the starting time for each game is not available in the data set). However, we can still calculate when in each game each play occurred. Each period of play is 12 minutes long and overtime—the 5th period—is 5 minutes long. At the start of each period, the game clock begins counting down from 12:00. So to calculate how much play time elapses before each play, we subtract the time that appears on the game clock from a duration of 12, 24, 36, 48, or 53 minutes (depending on the period of play). We now have a new duration that shows exactly how far into the game each play occurred.

```
R> lakers$time <- dminutes(c(12, 24, 36, 48, 53)[lakers$period]) -  
  lakers$time
```

We can now plot the number of events over time within each game (Figure 4.3). We can plot the time of each event as a duration, which will display the number of seconds into the game each play occurred on the x axis,

```
R> qplot(time, data = lakers, geom = "histogram", binwidth = 60)
```

or we can take advantage of `pretty.date()` to make pretty tick marks by first transforming each duration into a date-time. This helper function recognizes the most intuitive binning and labeling of date-time data, which further enhances our graph. To change durations into date-times we can just add them all to the same date-time. It doesn't matter which date we chose. Since the range of our data occurs entirely within an hour, only the minutes information will display in the graph.

```
R> lakers$minutes <- ymd("2008-01-01") + lakers$time  
R> qplot(minutes, data = lakers, geom = "histogram", binwidth = 60)
```

We see that the number of plays peaks within each of the four periods and then plummets at the beginning of the next period, Figure 4.3. The most plays occur in

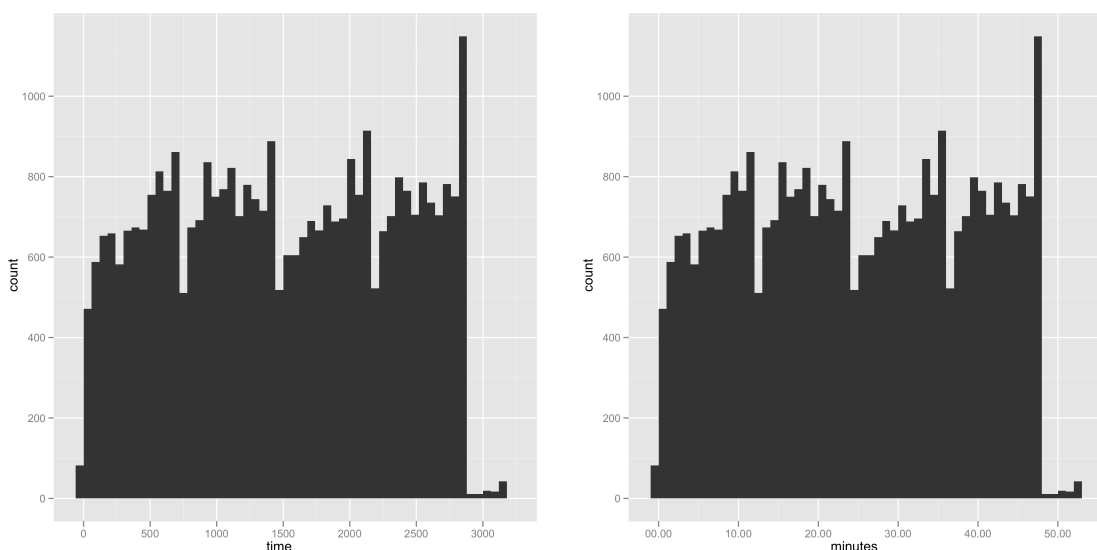


Figure 4.3: The graph on the left displays seconds on the x axis. The graph on the right uses a more intuitive division with minutes.

the last minute of the game. Perhaps any shot is worth taking at this point or there's less of an incentive not to foul other players. Fewer plays occur in overtime, since not all games go to overtime.

Now let's look more closely at just one basketball game: the game played against the Boston Celtics on Christmas of 2008. We can quickly model the amounts of time that occurred between each shot attempt.

```
R> game1 <- lakers[lakers$date == ymd("20081225"),]
R> attempts <- game1[game1$type == "shot",]
```

The waiting times between shots will be the timespan that occurs between each shot attempt. Since we have recorded the time of each shot attempt as a duration (above), we can record the differences by subtracting the two durations. This automatically creates a new duration whose length is equal to the difference between the first two durations.

```
R> attempts$wait <- c(attempts$time[1], diff(attempts$time))
```

```
R> qplot(as.integer(wait), data = attempts, geom = "histogram", binwidth = 2)
```

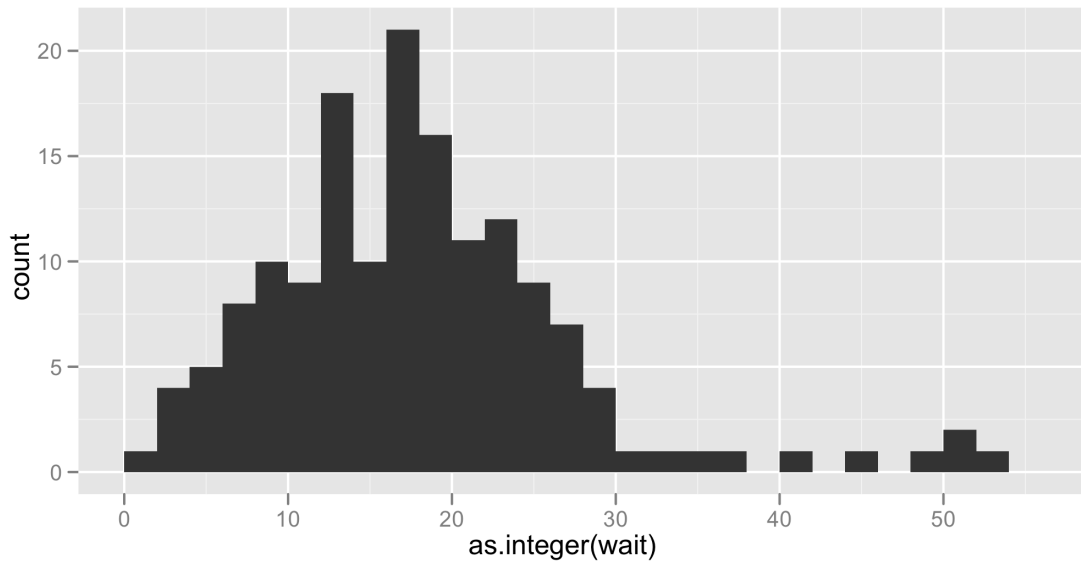


Figure 4.4: Wait times between shot attempts rarely lasted more than 30 seconds.

Rarely did 30 seconds go by without at least one shot attempt, but on occasion up to 50 seconds would pass without an attempt.

We can also examine changes in the score throughout the game. This reveals that though the game was eventful, the Lakers maintained a lead for the most of the game, Figure 4.5. Note: the necessary calculations are made simpler by the `ddply()` function from the `plyr` package, which `lubridate` automatically loads.

```
R> game1_scores <- ddply(game1, "team", transform, score = cumsum(points))
R> game1_scores <- game1_scores[game1_scores$team != "OFF",]
R> qplot(ymd("2008-01-01") + time, score, data = game1_scores,
        geom = "line", colour = team)
```

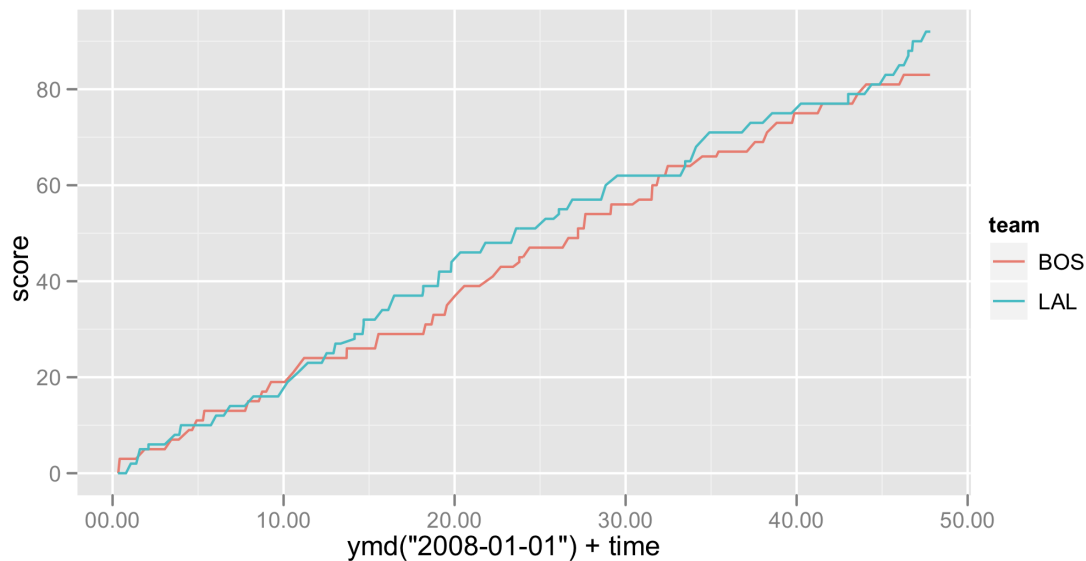


Figure 4.5: The lead changed between the Lakers and Celtics numerous times during the game.

4.11 Conclusion

Date-times create technical difficulties that other types of data do not. They must be specifically identified as date-time data, which can be difficult due to the overabundance of date-time classes. It can also be difficult to access and manipulate the individual pieces of data contained within a date-time. Arithmetic with date-times is often appropriate, but must follow different rules than arithmetic with ordinal numbers. Finally, date-related conventions such as daylight savings time and time zones make it difficult to compare and recognize different moments of time.

Base R handles many of these difficulties, but not all. Moreover, base R's date-time methods can be complicated and confusing. `lubridate` makes it to easier to work with date-time data in R. The package provides a set of standard methods for most common date-time classes. These methods make it simple to parse, manipulate, and perform calculations with date-time objects. By implementing the time concepts pioneered by projects such as `Joda Time` and `Boost.Date_Time`, `lubridate` helps

researchers perform precise calculations as well as model tricky time-related processes. `lubridate` also makes it simple to switch between time zones and to use or ignore daylight savings time.

Future work on the `lubridate` package will develop methods for handling partial dates and for modeling recurrent events, such as stock market opening times, business days, or street cleaning hours. In particular, we hope to create methods for R that work with reoccurring temporal date patterns, which were introduced by Fowler (1997) and have been implemented in Ruby by the `runr` project (Lipper, 2008).

Acknowledgements

We would like to thank the National Science Foundation. This work was supported by the NSF VIGRE Grant, number DMS-0739420.

How and why to teach statistical inference with simulations in R

ABSTRACT — This paper describes a software design that can be used to build a visual simulation tool in R and presents guidelines on how to use this tool in the classroom. Many researchers and educators have endorsed computer simulations as a tool to teach statistical concepts. Yet teachers who wish to use simulations face two hurdles: how to build a computer based simulation and how to use such a simulation in the classroom effectively. We demonstrate how the universal pattern of inference and the technical details of R can be used to create a highly efficient simulation tool, and provide a blueprint for coding this tool. The blueprint creates a program that can be customized to fit specific lesson plans and easily extended to visualize new inference methods, which we demonstrate with **Visual Inference Tools (VIT)**, a program based on the blueprint. We conclude with a review of Cognitive Load Theory, Multimedia Learning Theory and empirical studies on the use of simulations to teach statistical inference. This review suggests four principles to guide the presentation of visual simulations to teach statistical inference.

5.1 Introduction

Many researchers and educators have endorsed computer simulations as a tool to teach statistical concepts.¹ Yet teachers who wish to use simulations face two hurdles: how to build a computer based simulation and how to use such a simulation in the classroom effectively.

Empirical research shows that simulations yield mixed results. A number of studies find that students who learn with simulations performed better in a variety of ways (Weir et al., 1990; Lane and Tang, 2000; Lane and Scott, 2000; Ziemer and Lane, 2000; Wender and Muehlboeck, 2003). Yet some studies also observe disappointing results with simulations. Experiments described in delMas et al. (1999); Lane and Peres (2006) and Peres et al. (2010) all found that simulations did not provide the desired results until they were paired with some type of guided instruction. Simulations alone do not guarantee successful learning of statistical concepts; the success of a simulation is moderated by how it is used within a lesson plan. This paper provides guidance on how to create visual simulations for teaching statistics as well as guidance on how to incorporate these simulations into lesson plans. Our recommendations are based both on empirical studies of teaching statistics with simulations and on established theories of learning. We focus on building simulations with the R programming language (R Development Core Team, 2010). The R language has many useful benefits: it is freely available, it is popular, and it comes with many already written statistical functions. However, using R to build visual programs can be tricky, an issue we address.

This research takes place in the context of a larger research study sponsored by the government of New Zealand. We describe this background in Section 5.2.

The remainder of this paper proceeds as follows.

Section 5.2 describes the New Zealand teaching initiative which motivates this

¹See Mills (2002) for a list of 48 articles about teaching statistics with simulations.

research. As part of this initiative, the author developed Visual Inference Tools (VIT). VIT is a visual simulation program in R that can be easily expanded to simulate new statistical concepts. We will use VIT as an example throughout the rest of the paper to illustrate our ideas.

Section 5.3 presents the case for visual simulations as a teaching tool. The studies mentioned above, research on visualization, and two theories of learning all suggest that visual simulations can aid learning. We examine these theories and review the arguments that support using visual simulations to teach.

Section 5.4 presents the case for building statistical simulations in R. R is a freely available programming language with an open copyright. Unlike more popular languages, R comes with prewritten functions for many popular (and arcane) statistical techniques. This makes R ideal for creating a general statistical simulation tool. Users can rely on existing functionality to power their simulations. However, building visuals in R can be tricky. R is a functional programming language not directly suited to GUI programming.

Section 5.5 provides a blueprint for making visual simulations in R. This blueprint creates a visual GUI interface that teachers and students can use without knowing how to program in R. Moreover, the blueprint provides a universal base for hosting simulations of different statistical inference concepts. Users familiar with R programming can easily create new simulations by combining this base with customized methods.

Section 5.6 concludes with recommendations on how to incorporate visual simulations into lessons about statistical concepts. This section summarizes the relevant findings of both the empirical studies presented in Section 5.1 and the theories of learning presented in Section 5.3.

5.2 Background

The software and guidelines proposed in this paper have been created as part of a broader research project on the staged instruction of statistical inference ideas over a period of years. This project is conducted by New Zealand's Teaching and Learning Research Initiative (TLRI), a group of researchers partly based at the University of Auckland's Department of Statistics. Papers on this work that have already appeared include Wild et al. (2010, 2011) and Arnold et al. (2011). The TLRI program is part of an ongoing curriculum reform for K-13 mathematics for New Zealand schools. The software and guidelines discussed here form one plank of this program and will be incorporated into New Zealand's secondary education curriculum. However, the software and guidelines are also useful for any teacher who wishes to use simulation in a lesson plan.

The resources in this article are specifically intended to improve the instruction of statistical inference. Statistical inference is the process of drawing reliable conclusions from data in the presence of variation. It is an important skill for engineers and scientists, but students are failing to comprehend inference as it is traditionally taught. Shaughnessy (2007) documents how students fail to retain the many necessary concepts that motivate statistical inference, such as: sampling, variation, and distributions. In a study by Pfannkuch (2006), both Grade 10 students and their teachers misunderstood sampling variability and the differences between samples and populations. Pratt et al. (2008) confirmed that this confusion is common across high schools.

Chance et al. (2004) suggest that this lack of understanding is the result of cognitive overload. Formal statistical inference is typically introduced in the last year of high school, when it can be taught along with related mathematical skills. As a result, students are pressed to master multiple complex concepts all at once, something

few of them manage. An alternative method is to teach younger students informal inference. Informal inference circumvents cognitive overload by teaching inference over a longer period of time and without relying on mathematical techniques. The approach lays solid conceptual foundations upon which to build more formal inference in the longer term (Wild et al., 2011). The positive results of this method have been demonstrated by Makar and Rubin (2009), Arnold and Pfannkuch (2010), and Pfannkuch (2010).

New Zealand decided to incorporate informal inference into its current curriculum reform for K-13 mathematics. The challenge is now to devise specific lessons that teach inference concepts in an informal manner. One idea is to use interactive or animated visualizations to convey understanding about sampling variation. A preliminary study conducted by Pfannkuch (2010) supports this approach. Pfannkuch found that visualizations made with paper and pencil by the class could create understanding about sampling variation in an eleventh grade classroom. The staged curriculum will next build upon the paper and pencil plots and connect them to specific inference concepts, such as distribution, sampling variation, and regression, with visual simulation tools. These tools will need to be user friendly enough to be used by every secondary school teacher in New Zealand, but rich enough to be customized and extended to a variety of statistical concepts.

The author developed the Visual Inference Tools (VIT) computer program to meet this need. VIT is a software package written in R that provides a simple and effective way to teach statistical inference to beginning statistics students. VIT builds intuition about inference by demonstrating inference concepts visually with interactive animations. This approach allows teachers to build upon research that shows that visual instruction can improve student performance. VIT contains methods for presenting bootstrap, sampling, and confidence interval topics, but it is designed to be

extendable. A teacher proficient in R can easily add additional methods to the VIT software to demonstrate additional concepts.

VIT is the basis for the software blueprint described in Section 5.5. A lesson plan that uses VIT will also serve as an instructional example in Section 5.6. Readers who do not wish to recreate the software design proposed in this paper, or who wish to save time, can download VIT and customize it for their own needs as described in Section 5.5. VIT is available for download at <http://www.stat.auckland.ac.nz/~wild/VIT/>.

5.3 Why teach with visual simulations

Simulation programs like VIT can improve learning when they present information visually. Visualizations are a natural way to think about concepts, especially mathematical ones. In fact, cognitive and neurological research shows that the mind uses internal visualizations to think about mathematical concepts. Visualizations can also prepackage information in a way which is easy to understand, which can further accelerate learning. Finally, visual simulations can display the same results as mathematical calculations, but they do not require students to understand math as a prerequisite. In this section, we review the findings that support these assertions and introduce two theories of learning that support the use of visualizations in the classroom: Cognitive Load Theory and Multimedia Learning Theory. The review demonstrates that visualizations have the potential to significantly increase student learning.

5.3.1 Visual simulations and thinking

Visualization is a natural way for students to think about ideas. By teaching with visualizations, teachers can directly shape the way students think about new concepts. Cognitive scientists suggest that humans think by mentally reliving perceptual experiences (Davis, 1998; Barsalou, 1999; Reed, 2010). For example, when we think about the word *green*, we mentally relive the experience of seeing the color green. Vision is the primary way humans experience the world, and as a result, vision plays a large role in the way we think. The role of vision in thinking can be seen in the way people build associations between concepts. Paivio (1969) noticed that people build associations between words in two ways. They sometimes use the name of a word to think of related words. Other times, they use an image of the concept connected with a word to think of related concepts. Individual words vary in the number of verbal and visual associations that they support. Paivio demonstrated that people remember words better when the word has a large number of potential visual associations.

This insight led to the development of dual coding theory Paivio (1986, 2006). Dual coding theory states the humans encode information into memory in two formats, visually and verbally. Information that is encoded in both formats is more likely to be remembered than information that is only encoded in one format. Dual coding theory is supported by studies that show that a visual and a verbal task can sometimes be performed at the same time by the mind, but two visual or two verbal tasks will interfere with each other. For example, Brooks (1968) showed that it is easy to simultaneously recall visual information and describe it verbally, but difficult to recall verbal information and describe it verbally.

The visual/verbal duality of the mind has been well tested since Paivio proposed dual coding theory. It is now incorporated into the most influential models of the working memory (Baddeley and Hitch, 1974; Baddeley, 2000, 2001). The working

memory is the part of the mind that processes new information into knowledge that can be stored in the long term memory. As such, it plays a central role in learning. The most influential model of the working memory was proposed by Baddeley and Hitch (1974). Baddeley conceived of the working memory as two separate information processors: a visual-spatial sketchpad, which process visual information, and a phonological loop, which processes verbal and non-visual information. The working memory also contains a central executive, which coordinates the activity of the two processors. In 2000, Baddeley slightly expanded the model to include a new component which reflects the working memory's role in integrating visual and verbal information together to create knowledge.

The working memory may actually prefer to process information visually. Studies suggest that the mind converts verbal information into visual information when it is possible to do so. For example, Stanfield and Zwaan (2001) had students read a sentence that implicitly described an object in a certain orientation, such as a pencil laying flat or a pencil held up vertically. Students were then presented a picture of an object and asked whether the object in the picture appeared in the sentence. Students responded faster when the object appeared in the picture with the same orientation that it had in the sentence. The results suggested that students were mentally processing the sentence in a visual, not verbal format. Moreover, since the response times were so small this appeared to be happening at an innate level, not a conscious level. Zwaan and Yaxley (2003) used similar methods to determine that students relied on mental imagery to determine how pairs of words are related.

Visualization is also a natural way for students to think about mathematical concepts, such as statistics. The mind seems to use internal visualizations to reason about numbers and quantities. For example, Moyer and Landauer (1967) found evidence that the mind thinks about numbers with the help of a mental image of the num-

ber line. As a result, people compare numbers that appear farther apart on the line faster than numbers that appear closer together. Surprisingly, Dehaene (1997) were able to pinpoint the direction of the number line. Subjects consistently responded faster when asked to associate high numbers with the right and low numbers with the left. Neurologists have even identified a physical link between visualization and math. Walsh (2003) compared fMRI, PET, EEG, and brain lesion studies to demonstrate that the brain uses the same neural pathway to reason about space, time, and quantity. This pathway is in the right inferior parietal cortex of the human brain. Visualizing spatial information ignites this pathway, as does performing approximate arithmetic (Dehaene et al., 1999). These studies suggest that the brain is hard wired to process basic math in visual terms. Lakoff and Núñez (2000) argue that complex mathematical operations depend on visual experiences as well. In their book, *Where mathematics comes from*, they trace many elements of mathematical logic to metaphors built on visual experiences. For example, the concept of set membership is built upon a container metaphor that relies on a consistent visual experience. We never see something that is simultaneously inside a container and outside of a container.

In summary, visual simulations are a natural way to think about concepts, especially mathematical concepts. Using simulations may improve learning by engaging directly with the visual ideas that directly underpin student's conceptions of numerical phenomena. By providing students with a visual metaphor and a visual image, simulations may prevent students from developing incorrect internal imagery to associate with a new concept. Visual simulations can foster learning in other ways as well.

5.3.2 Visual simulations and cognitive load

Visualizations can promote learning by managing cognitive resources during instruction. Cognitive science has shown that the mind's ability to process new information is surprisingly limited. Studies estimate that the working memory can only hold from four (Cowan, 2000) to seven (Miller, 1956) pieces of novel information at once. Moreover, novel information seems to fade within the working memory after about 20 seconds (Peterson and Peterson, 1959). These limits create an obstacle to learning. Cognitive Load Theory (CLT) describes the effects of the limitation on learning and how they can be lessened.

Cognitive load refers to work performed by the working memory. Every learning task requires a certain amount of cognitive load. Students must identify information, hold it within their working memory and process it into knowledge that can be stored in the long term memory. Sweller (1988) demonstrated that learning fails to occur when the cognitive load of a learning task exceeds the capacity of the working memory. Sweller showed that learning can be facilitated by identifying and minimizing unnecessary sources of cognitive load. His Cognitive Load Theory divides load into two components; extrinsic cognitive load and intrinsic cognitive load (Sweller et al., 2011). *Extrinsic* load is created by the way information is presented. When information is poorly presented, a student must search for the information to be learned, which expends mental energy. Extrinsic cognitive load can also be caused by distractions during learning process or by pairing learning with unnecessary activities, such as problem solving. Extrinsic cognitive load can be reduced by improving the instructional design of a lesson. In contrast, *intrinsic* cognitive load is created by the complexity of the material to be learned. Intrinsic load cannot be reduced, the working memory will always have to work harder to understand complex material than simple material. However, intrinsic cognitive load can be managed with a variety of

strategies.

Mayer (2001)'s Multimedia Learning Theory (MLT) extends the idea of cognitive load even further. According to Mayer, a student must first expend extrinsic cognitive load acquiring information from the environment. Next, the student must expend intrinsic cognitive load to hold the information within the working memory. At this point, rote learning can occur; the student can memorize information, hold it in the working memory, and recall it later. However, to deeply understand material, a student must expend further cognitive load to build relationships between the new information and previously acquired knowledge. This activity will allow the student to apply the new information in new contexts, a phenomenon known as transfer. Mayer calls this third type of cognitive load *generative* cognitive load. Under Mayer's model the three types of cognitive load are sequential and additive. If extrinsic cognitive load surpasses the available capacity of the working memory, intrinsic processing and rote learning can not occur. If the combination of extrinsic and intrinsic load surpasses working memory resources, generative processing and deep understanding cannot occur. Moreover, the working memory does not automatically expend resources on generative load. A student must be interest and motivated in the material before they will engage in generative processing.

Cognitive Load Theory and Multimedia Learning Theory have been corroborated by many studies. Predictions based on these theories have been shown to improve leaning outcomes in a variety of settings. Together, CLT and MLT propose a clear guideline for instruction: information should be presented in an easily accessible manner with as few distractions as possible. Visual simulations can provide this manner. Simulations satisfy two principles tested by CLT and MLT research, the modality principle and the multimedia principle. As we discuss these principles below, we assume that visual simulations will be accompanied by an oral explanation given by

the instructor. Other practices can further strengthen the effectiveness of simulations, which we discuss in Section 5.6.

Visual simulations can foster learning by allowing the instructor to present visual and oral information at the same time. This creates a modality effect that has been shown to increase learning. The modality principle states that students learn better when material is presented as visual imagery accompanied by spoken words (Mayer, 2009). This arrangement both increases the capacity of the working memory and decreases the extrinsic cognitive load of the material to be learned. Working memory capacity is increased because the working memory can use both of its processors (visual and phonological) to process the information. When information is presented as just images or just words, the working memory is limited to a single processing channel and less information can be analyzed. By using spoken words instead of printed words, the lesson maximizes the gain. Spoken words can be analyzed in the phonological loop of the working memory, but printed words must first be passed through the visual-spatial sketchpad, which limits the amount of attention that can be given to other images. Dual mode lessons further decrease cognitive load because students can simultaneously attend to sights and sounds. As a result, the mind does not have to expend cognitive load to temporarily store some information in the working memory while other information is attended to.

The modality effect was first demonstrated by Mousavi et al. (1995). Mousavi et al. hypothesized that if the working memory contained separate processors for visual and phonological information, geometry students might learn better when diagrams are accompanied by oral rather than written explanations. Their results showed that students taught in the dual mode manner could solve subsequent problems more quickly and accurately than students taught in the visual only mode. Since then, multiple studies have recreated the modality effect in a variety of settings. The

modality principle has been shown to decrease cognitive load during instruction, improve post-test performance (Kalyuga et al., 1999, 2000; Tindall-Ford et al., 1997), improve student speed during problem solving (Jeung et al., 1997; Mousavi et al., 1995), and improve retention and transfer (Mayer and Moreno, 1998; Moreno and Mayer, 1999). While studying the modality effect, Mayer (2009) observed that dual mode lessons increased student learning in 17 of 17 tests.

The modality effect appears to improve learning the most when students are new to a subject and the material presented would otherwise be complex or difficult to understand. Moreover, the modality effect can not be achieved by adding spoken or visual information to material that can otherwise stand on its own. The visual and spoken material must both provide essential (and different) pieces of information related to the concept being taught. Adding new material to a lesson always increases cognitive load. If the added material is redundant, learning would occur better without it.

Visual simulations can also foster learning through the multimedia effect. The multimedia principle states that students gain a deeper understanding by learning through words and pictures than through either words or pictures alone (Mayer, 2009). The multimedia principle extends the modality principle to predict that multi-mode presentations will foster generative processing, which creates a deeper understanding than rote learning. Words and pictures allow a student to form both a visual and a verbal model of a concept. This encourages the student to integrate the two models in their working memory, which is a type of generative processing. Integrating visual and verbal models of a concept creates a deep understanding because visual and verbal models contain different types of information. As a result, multimedia presentations should improve a student's ability to transfer newly learned material to new situations. Mayer and his colleagues tested the multimedia effect in 11 tests spread across seven

studies (Mayer, 1989; Mayer and Gallini, 1990; Mayer and Anderson, 1991, 1992; Moreno and Mayer, 1999, 2002). Each test compared a multimedia presentation to a single media presentation. Studies examined computer presentations, paper lessons, and games. A strong multimedia effect was observed in each test. Visual simulations allow the multimedia effect to occur in the same way that they allow the modality effect to occur; simulations enable the teacher to present both visual and verbal material simultaneously.

5.3.3 Visual simulations and prior mathematical knowledge

Finally, visual simulations make it easier to learn inference because they are based on visual experience, not math. As a result, students can learn about inference without relying on prior mathematical knowledge, which may be limited. In traditional statistics classes, inference behavior is explained through the use of mathematical theorems, such as the Central Limit Theorem, or through the use of mathematical models, such as the normal curve. A student cannot learn inference with this method unless they first understand the math presented. While mathematical sophistry is an important goal of statistical education, studies have shown that students perform poorly with current methods of teaching inference through math Shaughnessy (2007). Pfannkuch (2010) show that student performance can be increased by teaching inference separate from math. Students are then able to develop an understanding of inference behavior that can later support learning mathematical inference concepts. This approach is the cornerstone of the New Zealand K-13 mathematics curriculum reform.

This approach may also have the side effect of increasing the number of underrepresented minorities in the STEM career fields. In 2009, the the National Assessment for Educational Progress (NAEP) measured the mathematics achievement gap be-

tween White and Black eighth graders to be 32 points and the gap between White and Hispanic students to be 26 points (Aud et al., 2010). This gap places minority students at a disadvantage when statistical inference is taught with traditional methods, which require mathematical mastery. Visual simulations create a level playing field where poor performing math students can still build understanding with physical and visual experiences.

In summary, visual simulations can provide many advantages to teaching statistics. These advantages are predicted by a variety of research and theory. The predictions have been verified by the empirical studies listed in Section 5.1, which show that simulations improve learning of inference concepts. However, simulations require advanced planning. Before a simulation can be used in a classroom, it must be built in a computer programming language. In the next section, we discuss the benefits of using R as this language.

5.4 Why program visual simulations in R

The features of the R programming language are uniquely suited to developing user friendly, effective, and customizable statistical simulations. R is an open source software language freely available at <http://cran.r-project.org>. R was developed at the University of Auckland by Ross Ihaka and Robert Gentleman and borrows heavily from the S programming language written by John Chambers. Since the introduction of R in 1997, the language has become a staple in statistical computing and is now widely used within universities and corporations such as Google, Pfizer, and the Bank of America (Vance, 2009). The number of R users is estimated at over two million (Revolution Analytics, 2012) and many online resources exist to help users program in R. R has two pragmatic qualities that recommend it for educational simulations: it is free and contains many pre-written statistical sub-routines. These features make R

user friendly for both the programmer and end user of the simulations.

To be user friendly in the classroom, a simulation program must be easy to access and easy to use. R can be downloaded to any computer connected to the internet. It runs on all common operating systems including Macintosh, Windows, and Linux. R programs do not need to be precompiled, and they are easy to disseminate because R is free and has an open copyright. Although R is a command line language, it can be used to create software that does not use a command line. This makes R simulations useful in the classroom. The end user of an educational simulation will be a teacher or student, who may not be familiar with command line programming. To accommodate these users, simulations should have a graphical user interface (GUI). Many computer languages can be used to build a GUI interface and R is no exception. R contains software libraries specifically designed to create GUI tools, such as `RGtk2`, `qtbase`, `tcltk2`, and `gWidgets`. The VIT program uses the `RGtk2` library to create a user interface like the one pictured in Figure 5.3.

R also provides visual tools that make effective simulations. The effectiveness of a simulation can depend on the software it is built with. Both Cognitive Load Theory and Multimedia Learning Theory suggest that understanding can be stymied by distractions that occur during learning. These distractions increase extrinsic load and reduce the amount of working memory that can be directed to processing information. A software that provides primitive looking graphics or jumpy animations calls attention to itself and may interfere with learning. R is known for its beautiful graphics tools, such as the `ggplot2` and `lattice` graphics packages. Moreover, R gives programmers low-level control of visual elements through the `grid` package. Animations in R are a source of continuing development. A number of software packages including `cranvas` and `qtpaint` are being developed to improve the animation features of R, which should lead to future enhancements. In the meantime, packages

such as `animation` demonstrate that R already has sufficient animation capacity to build seamless simulations.

R's most useful feature for simulations, however, is that it allows programmers to quickly modify simulations to demonstrate new concepts. R comes packaged with a sophisticated random number generator and many user contributed statistical methods. Hundreds of sub-routines that implement both common and uncommon statistical techniques have already been published for the R language. These subroutines set R apart from web based languages like `javascript` and `html`. With R, users can extend simulations to teach a variety of statistical concepts with a minimum amount of additional programming. Programmers do not need to worry about programming the statistical aspects of the new simulation because statistical methods have already been written. Programmers can instead focus on the visual aspects of the simulation, which can be recycled from simulation to simulation.

These benefits make R a useful platform for a general statistical simulation tool. However, programmers will still have to deal with many nuances when programming in R. R is functional programming language with a small set of object oriented programming (OOP) features. Ideal simulations will use interactive GUI interfaces, which must keep track of a large amount of information. Because R is a functional language, this information must be passed from function to function. Moreover, user input must be inserted into the information set at the appropriate places in the chain of functions that underly the program. These requirements can quickly create an overwhelming, chaotic, and bug prone simulation program. The next section presents a blueprint for building simulations in R that avoids this outcome. The blueprint creates a simple, elegant, and highly modular structure for a simulation program in R.

5.5 How to implement visual simulations in R

R is a logical choice for building a general simulation tool due to R's visual capabilities, wide availability, and pre-written statistical libraries. However, programming a general GUI simulation tool that can be used to study multiple inference topics is a complex task. This task is made more intricate by the nature of R. R is a functional programming language. It processes information by passing information into functions and returning output. This approach is not well suited to interfaces like GUIs that require large amounts of information to be stored together and manipulated in sophisticated, interdependent ways. The information store must be passed into a function, where it is copied, manipulated and returned as output that must be collected for the next stage of processing. This approach consumes large amounts of memory and creates intricately woven chains of functions that are difficult to understand, interact with, and reuse for similar but slightly different tasks.

In this section, we describe a software design devised by the authors that avoids these inefficiencies. The design relies on concepts of object oriented programming, as well as the common patterns of statistical inference, to create a general GUI simulation program that can be used to teach many different concepts related to statistical inference. The design creates a simple, modular, and memory efficient program that can easily be customized or extended to new teaching situations. This design blueprint is the basis of the VIT package. We use VIT to illustrate the principles behind this design; the figures in this section all come from screenshots of the VIT program. However, this software blueprint can also be applied outside of VIT to make entirely new simulation platforms. The blueprint is shaped by the interaction of two factors: the common patterns of statistical inference and the technical details of R.

5.5.1 Patterns of statistical inference

Statistical inference revolves around five concepts that are related to each other in fixed ways: a population of data, a sample of data drawn from that population, a measurement of the population known as a *parameter*, a measurement of the sample known as a *statistic*, and the distribution of that statistic over repeated sampling. To understand inference, a student must understand these concepts and how they are related to each other. This provides unity to simulations that teach statistical inference. Because these simulations are visual, each concept must be represented visually. Relationships between the concepts must be expressed with animation, with one concept originating in its parent concept, or left implicit. The teacher can explain implicit relationships, or they can be hinted at with shared visual aesthetics, such as color or shape. These common constraints have yielded a popular format for inference simulations, seen in Figure 5.1. Simulations visualize three distributions: a distribution of the population at the top of the screen, a distribution of the current sample in the center of the screen, and a distribution of the sample statistic at the bottom of the screen. In more sophisticated simulations, animations reinforce the relationships between each distribution. Sample points drop down from the population distribution and sample statistics drop down from the sampling distribution. As the simulation runs, repeated samples are taken, repeated statistics are calculated, and the statistics accumulate to reveal a distribution for the sample statistic. This format is often used to illustrate the Central Limit Theorem when the sample statistic is the mean. But it can also illustrate the behavior of a variety of statistical summaries, including box plots, confidence intervals, and regression lines, Figure 5.2.

The format described above creates a universal method of simulating inference concepts. This method can be applied to different types of data, different types of sampling method, and different types of statistical summary. To create such a display,

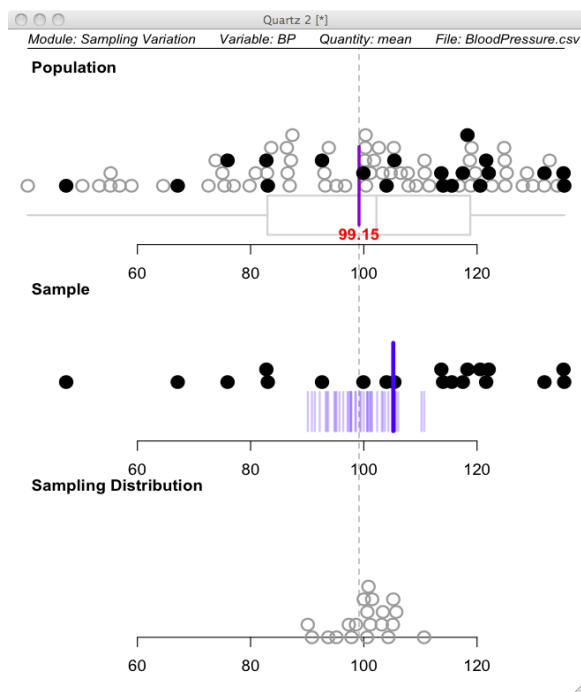


Figure 5.1: Statistical inference simulations are often organized around three fields: a field that displays the population data (top), a field that displays the sample data (middle), and a field that displays the sample statistic data (bottom).

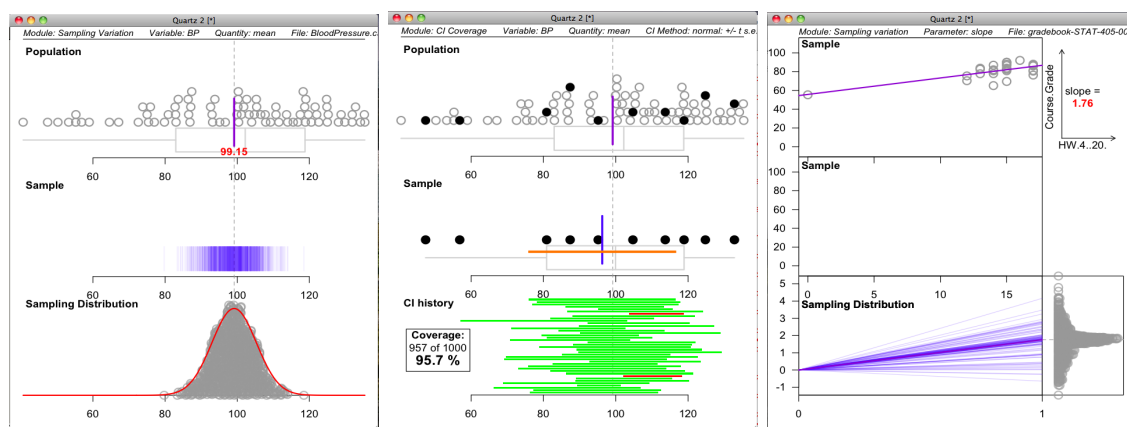


Figure 5.2: The three field format can be used to illustrate inference related concepts such as the Central Limit Theorem (left), confidence intervals (middle), and the variance of regression lines regression lines (right)

a simulation only has to perform five tasks:

1. draw a group of data
2. calculate a measurement
3. draw a measurement
4. animate sampling
5. animate collecting the sample statistic

The method used to draw a group of data can be reused to draw the population distribution, the sample distribution, and the distribution of the sample statistic. The method used to calculate a measurement can be reused to calculate the population parameter and the sample statistic. The same is true for the method used to draw a measurement. Some of these tasks can be subdivided to provide different behavior for different distributions (e.g., sample statistics can be drawn with a different shape than population parameters). However, these five tasks are sufficient to create a complete inference simulation. They form a “pattern” that characterizes inference simulations. The details of each task will change based on the concept that is being visualized. For example, simulating the standard error bands of a regression line requires different drawing methods than simulating the distribution of \bar{x} , Figure 5.2. However, the general sequence and goals of these tasks will remain constant across all inference simulations. These tasks provide a way to create a simple simulation program that can be extended to teach any inference related concept. The specifics of how to implement such a program depend on the technical details of R.

5.5.2 Technical details of R

An interactive, GUI based simulation program must manage two levels of information. First, the program must collect and store user input. To do this, the program must create and maintain a GUI interface that makes it easy for the user to interact with the program. The interface should respond visually to user actions and reflect the current state of the program. Second, the program must manipulate a visual canvas that displays simulation output. This canvas may be integrated into the GUI interface, but it performs a specialized role and must manipulate its own store of information to do so. A simulation program will also have to do some background processing of information to connect these two levels. A screenshot of VIT reflects these two levels of information, Figure 5.3. The panel on the left displays the current settings of the simulation and collects user input. The window on the right displays the output of the simulation on a canvas.

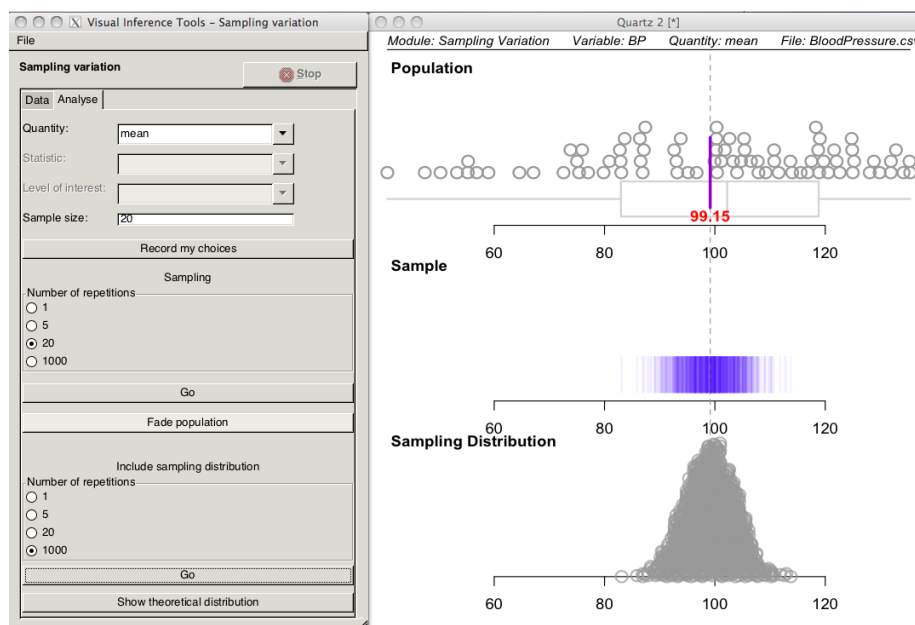


Figure 5.3: The user can interact with VIT through the GUI provided in the panel on the left. This panel is managed by the “gui environment” described in our software blueprint. The panel on the right provides a canvas to animate the simulation created by VIT. This panel is managed by the “canvas environment.”

The input level of the simulation can be efficiently managed by creating a new R environment dedicated to managing the simulation’s GUI interface and storing user input. In R, environments are elemental reference frames used to store data and functions. A dedicated “GUI environment” can become the basis of an object oriented programming (OOP) approach to the simulation program, which provides a number of advantages. As mentioned before, R is primarily a functional programming language. Information is passed into functions and output is returned. In object oriented programming, information structures (i.e, objects) replace functions as the focus of the program. Information is stored in an object and functions are assigned to specific objects. When a function is run, it causes a change in the object it is assigned to. This change happens automatically, no output from the function needs to be collected and saved. Object oriented programming allows simpler program designs and can improve memory management, which creates faster programs. Our design creates an object oriented program from a functional language by exploiting R’s scoping rules.

In R, functions as well as data objects are assigned to an environment. Environments are then assigned to other environments, which creates a hierarchy of information. When a function must use a data object, it first looks for the data object among the objects that have been passed into the function as arguments. If any object is not found there, the function next searches the environment to which the function is assigned for the object.¹ This “lexical scoping” rule lets us dispense with passing user input into every function and collecting the results on the other side. Instead we can create a new environment and save all GUI related information, including user input, to that environment. Next, we assign each function that must work with the GUI to

¹Note: this is not the same as the environment in which the function is called. We recommend that readers familiarize themselves with R Development Core Team (2012) to master the tricky scoping rules of R.

the same environment. This arrangement essentially creates a an OOP object, the environment, and a set of object specific methods, the functions. When a function is called, it has free access to the data objects stored in the environment. These objects do not need to be passed into the function as arguments. We can avoid the need to collect output from the functions by having the functions directly manipulate objects in the GUI environment. Functions can directly manipulate an object in a particular environment by calling the object within the function by the `<environment>$<object>` naming convention. Here `<environment>` is the name given to the GUI environment and `<object>` is the name of the object within the environment to be manipulated. This concept can be demonstrated with the following trivial piece of R code.

```
e <- new.env()
e$x <- 1
e$x # 1
change_x <- function(y) e$x <- y
change_x(2)
e$x # 2
```

`change_x` does not return any output, but directly manipulates the `x` object in the `e` environment. The only constraint here is that the function must be able to find `<environment>` using lexical scoping. Since `<environment>` will be the function's parent environment, this will never be a concern. `<environment>` will always be the first place the function looks. This approach to object oriented programming parallels that provided by the `proto` package, which develops simpler principles to create an OOP class within R. However, this approach is simpler and dispenses with all unnecessary complications. By managing the GUI interface with an environment, we greatly simplify the interactive aspects of the simulation program. Next we must address the simulation itself, including its visual output.

The output level of the simulation can be handled in a similar manner: it can be managed as a new environment that stores its own data and functions. As before, the functions should be written as methods that directly manipulate the objects in the new environment. If this second environment, or “canvas environment” is assigned to the GUI environment, the two objects will exist as a hierarchy, with the GUI environment as the parent environment and the canvas environment as the child. This arrangement will allow any function in the canvas environment to find and manipulate both objects in the canvas environment and objects in the GUI environment through lexical scoping.

The canvas environment has two tasks within the simulation program. It must process the data provided by the user and it must display the visual output of the simulation. Data processing includes such things as drawing random samples, calculating population parameters, and calculating sample statistics. These tasks should be performed before the program displays any visual output to avoid delays in animation. Displaying output will involve graphing the distributions of the population, sample, and sample statistic as well as animating the events of the simulation. In a single purpose simulation tool, the processing and display tasks will be fairly straightforward. Every simulation will process and display data in the same way. However, a general purpose simulation tool will require different types of processing and different types of display based on the inference method being simulated. Different processing and display methods will also be required whenever a teacher wishes to simulate the same concept in a different way, for example, with or without segmenting (see Section 5.6).

Writing a new processing and display engine for each simulation creates a large program that is difficult to extend. The user must supply large amounts of code for each new simulation. This arrangement also unnecessarily repeats large sections

of code. Creating visuals in R, such as the output of a simulation, requires a large amount of boilerplate code. The program must create a canvas, create graphical objects (grobs) to display in the canvas, create one or more reference systems within the canvas to arrange grobs on, and perform other tasks as well. R users usually rely on a graphics package such as `ggplot2`, `lattice` or the plot functionality of base R to do this work for them. Since we are creating our own custom graphics, we must do it ourselves and it is efficient to do as little as necessary and reuse code wherever applicable. The same is true to a lesser extent for the processing phase of the simulation.

Fortunately, the common patterns of simulating inference described in Section 5.5.1 provide a way to minimize boilerplate code while still allowing a large degree of customizability. The canvas should rely on a set of generic functions that parallel the common tasks of simulation (e.g., `DRAW.DISTRIBUTION()`, `CALC.MEASURE()`, `DRAW.MEASURE()`, `ANIMATE.SAMPLE()`, and `ANIMATE.STAT()`). When the program loads, these functions should be left empty, or preferably should return an error message to alert the user that something has gone wrong. A library of possible methods to replace each of these empty function names can be loaded with the program. The GUI interface should be constructed so that appropriate methods from the library are assigned to the empty function names above once a user selects the type of simulation they wish to run. Care must be taken to ensure that this assignment occurs in the environment where the empty functions are stored (i.e, the canvas environment). When the simulation code gets to the appropriate task, it will automatically perform the details specified by the appropriate method. Ideally, the methods library will be defined in the canvas environment so that method functions will share the same lexical scoping path as the task functions that they replace.

Due to R's lazy evaluation rules, new details can be assigned to the empty function

names every time the user requests a new type of simulation. The program will automatically use these new details the next time it performs one of the generic tasks. There's no need to close and reopen the canvas.

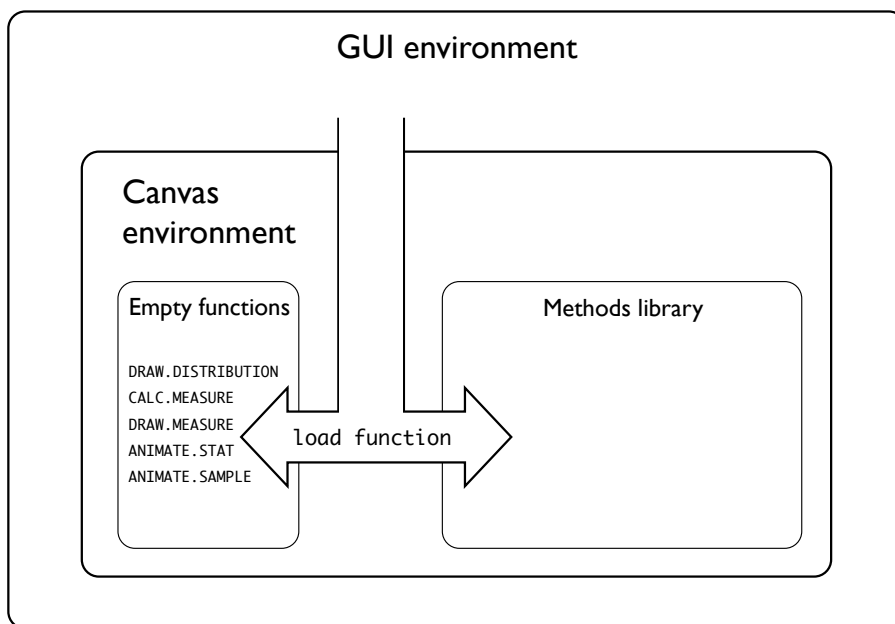


Figure 5.4: A general inference simulation tool can be built in R by organizing information into a GUI environment that manages the user interface and a canvas environment that displays the visual simulation. The canvas only needs to handle five actions, the details of these can be supplied at run time. To extend the program, a user only needs to supply additional methods for these five tasks.

The design described above, shown in Figure 5.4, allows a programmer to extend the simulation program to new applications in two steps. First, the programmer must add a small set of methods that will govern the new simulation to the methods library. Second, the programmer must amend the GUI's load function to include the option of loading the new type of simulation. We do not anticipate that a student or a teacher unfamiliar with R will be able to do this. The methods that must be added to the methods library need to manipulate visual objects and create animations, something that can only be done efficiently in R code. As a result, simulation tool will need a knowledgeable programmer to maintain it and extend it to new lessons. However, the

proposed design makes the programmer's job as simple as possible. The programmer must supply five methods. One of these, `CALC.MEASURE()`, can rely on pre-existing R functions. Many of the remaining methods can be recycled from already written simulations. Most statistical inference methods can be visualized in similar ways. For example, most simulations will draw either a 1D or 2D display for each distribution. Hence, only two separate methods for `DRAW.DISTRIBUTION()` are needed to cover most possible simulations: a method for drawing a distribution on one dimension, and a method for drawing a distribution on two dimensions. Programmers can recycle these methods if they already exist in the methods library.

The design described above creates a versatile “simulation engine” that can be combined with many outside features. For example, the VIT package also displays the raw data as a numerical list beside the popular tripartite simulation design. Users can also add extra features within the context of the simulation engine. Draw and animate methods can be written to highlight certain points, or vary the appearance or movement of points. These extra features can be chosen to support different ways of presenting the simulation in the classroom. This flexibility is important because the way a simulation is presented can directly affect its success as a learning tool. In the next section, we provide guidance that can be used to customize simulations and implement them within a lesson plan.

5.6 How to use simulations in the classroom

Simulations are a useful tool for helping students learn statistical inference. However, adding a simulation to a lesson plan does not guarantee that students will learn. A number of researchers have found that simulations occasionally produce disappointing results when used to teach statistical concepts. delMas et al. (1999) found that simulations of sampling distributions did not automatically improve student under-

standing of sampling concepts. Lane and Peres (2006) and Peres et al. (2010) both describe an undergraduate course at Rice University that introduced simulations of statistical inference. Although students rated the classes with simulations as “excellent”, they failed to retain the material taught by the simulations. To ensure that learning occurs, simulations should be carefully structured lesson plans. In this section, we discuss ways to maximize student learning with simulations. We examine the empirical findings listed above and review relevant implications of the theories of learning discussed in Section 5.3.

Unstructured simulations may make students passive, which can prevent learning. delMas et al. (1999) hypothesized that students could fail to learn from simulations because simulations foster passive learning. In passive learning, students may find the lesson too easy and do not actively think about the material in a simulation. As a result, they are less likely to remember it later. In the parlance of Multimedia Learning Theory, the student is not motivated to perform generative processing of the material, and hence only superficial learning occurs. The passive learning hypothesis is consistent with many studies that suggest that learning occurs when students are required to actively seek out and discover knowledge, an approach known as constructivist learning.¹ To test their hypothesis, delMas et al. repeated their study under active learning conditions. delMas et al. first gave students a pretest about the concepts to be learned. They then prevented passive learning by asking the students to use the simulations to evaluate their own answers on the pretest, a form of active learning. These students showed a significant improvement over those who just watched the simulation passively. Lane and Peres (2006) simplified this active learning approach into a technique that can be easily combined with any simulation: teachers can prevent passive learning by asking students to anticipate the results of

¹See for example Bruner (1961) and Hermann (1969)

a simulation before watching the simulation, an approach known as the “query-first” method. A study of the query-first method showed that instructors regard the method favorably and 86% of students found the method to be “somewhat helpful” or “very helpful” (Peres et al., 2010).

The theories of learning introduced in Section 5.3 suggest additional ways to structure the use of simulations. Both Cognitive Load Theory and Multimedia Learning Theory have implications that affect the way simulations should be presented in the classroom: simulations should be presented as a sequence of small, self-contained segments; pieces of related information should be visible (or audible) at the same time; and redundant information should be excluded from the simulation. Unlike the query-first method, these suggestions have not been specifically studied in the context of teaching statistical inference with simulations. However, they have been studied and verified in a range of similar educational settings.

Multimedia Learning Theory suggests that animations, like a simulation, are most effective when they are presented as a series of short animated segments. Inference simulations involve a series of sequential steps. When multiple steps are presented in a continuous animation, a student may fail to understand a step before the animation moves on. As a result the student will not be able to completely learn the relationships between the steps. In the statistical context, a student will not be able to understand how a sample statistic relates to a population parameter if he does not first perceive how samples are drawn from a population. Mayer (2009) suggests that segmenting the animation into short, self-contained steps can avoid this type of misunderstanding. When the animation pauses after each segment, a student has a chance to fully process the information within the segment before considering the next part of the animation. Segmented animations have been shown to improve learning over continuous animations in a variety of subjects including meteorology (Mayer and

Chandler, 2001), physics (Mayer et al., 2003), earth sciences (Mautone and Mayer, 2007), and chemistry (Lee et al., 2006). Gains in learning are increased even further when the student can control when the next animation begins after a segment ends.

The segmentation principle implies that when simulations are introduced in the classroom, they should be shown one step at a time. In the population-sample-statistic paradigm of statistical inference, two natural segments appear: drawing the sample from the population, and calculating a sample statistics from the sample. Each can be taught with its own segment. Segmenting can also be used to demonstrate the accumulation of sample statistics into a distribution. The distribution can be built as a sequence of segments that show drawing a single sample and calculating a single sample statistic to add to the distribution. As student understanding increases, the need to segment the simulations will decrease. Isolating a segment from the rest of the animation should no longer improve learning once a student understands what the segment shows.

Cognitive Load Theory suggests a second way to structure simulations. Sweller et al. (2011) proposes that animations can create a “transient information effect” that hinders learning: students are unlikely to understand relationships that require them to remember information from early in the simulation to make sense of information that occurs later in the simulation. First, students may not realize that they need to memorize early information and thus not be able to recall the early information at the appropriate time. Second, if students do memorize early information, the mental task of storing the information in the working memory decreases the amount of cognitive load that can be spent on learning and understanding. Sweller’s theory is supported by findings that animations with transient information increase the amount of work performed by the working memory (Ainsworth and VanLabeke, 2004). This “transient information effect” can be avoided by presenting all related information at the same

time. Students can then process information without having to memorize part of it. The transient information effect applies to both visual and oral information.

To avoid the transient information effect, teachers should provide any necessary explanations about a simulation while the simulation appears, not beforehand or afterwards. When important visual information appears in a simulation, it should remain visible until it is no longer needed. Many inference concepts rely on comparing the results of two or more samplings. The transient information effect implies that the visual results from each sample should be simultaneously visible to allow comparisons. VIT illustrates one way to accomplish this. Important information from previous samples remains visible as a “ghost” that can be compared to later samples, Figure 5.5.

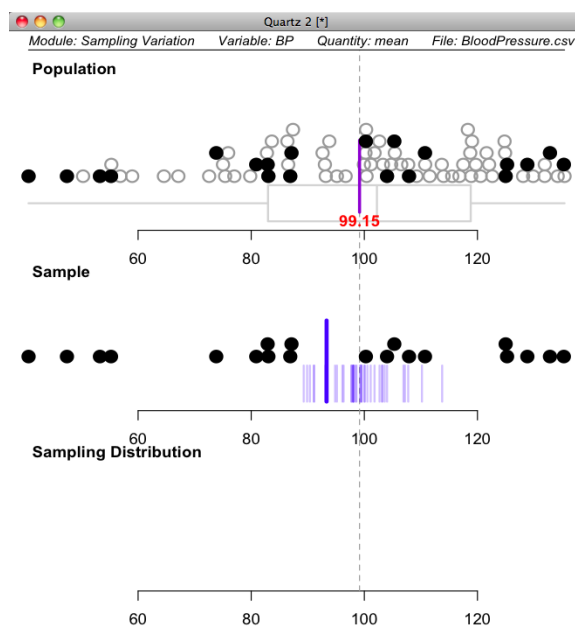


Figure 5.5: Previous sample means remain visible as “ghosts” in the middle field. These transparent blue bars are distinguishable from the current sample mean, which is larger and solid. Transparency also helps students see where multiple sample means have accumulated.

Cognitive Load Theory also suggests that redundant information can hinder learning. Students will spend unnecessary cognitive load attending to redundant informa-

tion, which decreases the load they can use to process necessary information Chandler and Sweller (1991). For example, when Chandler and Sweller (1991) added explanatory text to a self-explanatory diagram, student comprehension decreased. The redundancy effect can be used as a guide for designing the appearance of a simulation and for choosing narration to accompany a simulation. Teachers should only provide narration that explains components of the simulation that are not already self evident.

What material is redundant and what material is essential changes over time. Students need to hear information the first time they watch a simulation that they no longer need to hear once they become familiar with a simulation or the concepts it illustrates. This change in redundancy is sometimes called the expertise reversal effect. As students gain expertise, information that previously facilitated learning begins to become redundant, distracting, and an obstacle to learning. This can be avoided by providing less structure to learning as students build their abilities. In a simulation based lesson plan, a teacher can promote continuous learning by introducing new simulation formats and new statistical concepts with short segmented animations accompanied by ample explanations. As students begin to understand common patterns, the teacher can reduce explanations and can shift attention to the relationships between patterns. This method of instruction is called “guidance fading” in cognitive science research and has been shown to reduce the expertise reversal effect in many studies.¹

The VIT program demonstrates one way to create guidance fading with a simulation. It offers teachers the option of simulating just sampling or simulating the combination of sampling and calculating a sampling statistic. The program also allows teachers to simulate just one sample, five samples, or 1000 samples. When VIT

¹See for example Renkl and Atkinson (2003) for an overview.

simulates 1000 samples, it automatically ignores the animations that connects populations, samples, and statistics. VIT instead focuses on the growth of the statistic distribution over time. Instruction can begin by just simulating sampling. Once students understand sampling, the calculation of sample statistics can be simulated. At this point, teachers can skip over explaining the sampling step and instead focus on the sample statistic. Once students comprehend all of the details of a single iteration of the simulation process, the teacher can increase the number of iterations shown at once. The teacher can now show five samplings before pausing the simulation or explaining the sub-steps of each sampling. Instead, the teacher can help students see that repeated simulation creates variability in the sample statistic. Once students understand this, the teacher can further increase the number of simulations and demonstrate that the distribution of the statistic attains a stable shape over the long run. As the teacher moves on to each new step, he no longer needs to explain the information that was previously required to understand the last step. That information has become redundant, replaced by the student's own understanding. The redundant information can be phased out to avoid distracting the student from new learning.

To summarize, the success of a simulation can be enhanced by structuring the presentation of the simulation. Current research suggests that the following four guidelines will improve student learning of statistical inference with simulations. The first guideline has been specifically proven to improve student learning of statistical inference. The other guidelines are likely to improve learning of inference because they have improved learning in other domains with a simulation or animation.

1. Ask students to predict the outcome of a simulation before running the simulation.
2. Present the simulation in short, self contained segments until students under-

stand each piece.

3. Present all related information simultaneously. Provide oral explanation during the parts of the simulation that require it. Have visual information remain visible if it will be needed later.
4. As students begin to understand elements of the simulation, fade out explanations and shift students' focus to the relationships between the elements. This can be used both to teach the simulation format as well as the underlying inference material.

5.7 Conclusion

Simulations provide an opportunity to improve the way students learn statistical inference. They present information visually in a format that the human mind is well suited to consume. They align with insights from both Cognitive Load Theory and Multimedia Learning Theory. Empirical studies show that simulations specifically improve student learning of statistical inference. Moreover, other research suggests that current methods of teaching statistical inference disappoint. Simulations provide a timely way to improve the way we teach inference.

However, simulations are difficult to construct and can be misused. R provides a way to build a general simulation tool that can illustrate many different applications of statistical inference. It is free, it is open source, and most methods of statistical inference are already programmed into R. A general simulation tool that can be easily extended to new concepts can be written with a modicum of upfront programming. The program should be divided into two environments and organized around five generic functions. New simulations can be created by simply supplying new details to each of the generic functions. Details from previous simulations can be frequently

recycled to make new simulations. We introduce the VIT program as a generic simulation tool based on this blueprint.

A general simulation tool presents an educational advantage. Once students understand how the simulation program works in one context, they will more quickly understand how it works in a second context, and then a third, and so on. The tool will become less and less of a distraction from the material it illustrates. Studies suggest that the success of simulation tools can also be increased in four ways. Teachers should pair simulations with query-first methods of instruction. Teachers should initially divide simulations into short, self contained segments. Teachers should organize simulations and lessons so that related material is presented simultaneously. And, teachers should phase out explanations as student understanding increases and shift focus to relationships between concepts.

All of these principles have been tested in educational settings, but only the first principle has been tested specifically with statistical inference simulations. Simulations that illustrate statistical inference have a common structure and visualize reoccurring patterns of relationships between populations, samples, and statistics. Future work should examine empirically how this structure can best support the principles of segmenting, simultaneous presentation, and guidance fading. We believe that statistical simulations can further foster learning by developing ways of using ghost images to retain important information and presenting simulations as an orderly sequence of segments of growing lengths. In this manner, teachers can lead student attention from the nature of the simulation program, to the nature of sampling, to the idea of statistic and parameters, then to the concept of sampling variability, and finally to the realization that sample statistics attain a stable distribution over repeated sampling.

Acknowledgements

We would like to thank the National Science Foundation. This work was supported by the NSF EAPSI Grant, number 1107709.

Conclusion

The preceding chapters propose and describe a new way to look at data analysis. Data analysis is a cognitive task, known elsewhere as sensemaking. During data analysis, an analyst constructs knowledge about reality by comparing his or her schemas against quantitative data. When a schema does not agree with trusted data, the analyst either updates the schema or replaces it entirely. This is an innate task; humans perform sensemaking all the time by comparing schemas against everyday observations. However, quantitative data does not resemble everyday observations. It is precisely measured, which reveals variations that would otherwise go unnoticed. These variations undermine the sensemaking process, and statistical methods must be used to guarantee sensible results. This interpretation of data analysis explains how manipulating data into models can create understanding, and it aligns closely with prominent descriptions of data analysis in the statistical literature.

As a cognitive process, data analysis is vulnerable to the same weaknesses as other cognitive processes. One weakness is an over reliance on initial schemas. If an analyst begins with a faulty schema, he is less likely to notice that the schema is wrong and then replace it with a correct schema. Even scientists who are trained to be objective, such as NASA climatologists, are unlikely to notice when this bias occurs. A second

weakness is that data analysis can not logically verify its results. Like sensemaking, its generative methods are built around abductive, not deductive logic. Finally, as a cognitive process, data analysis relies heavily on the abilities of the mind, which are limited.

The actual process of data analysis is simple and familiar. It revolves around predicting, comparing, rejecting, and guessing. Readers may recognize the pattern of data analysis in the Scientific Method, in Karl Popper's theories of falsification, in the Hegelian Dialectic, or in many other places. This thesis even suggests a reason for these similarities: all of the above are knowledge building methods that ultimately must interact with a human's mental sensemaking process. In practice, however, the simple data analysis task is made complicated by two things the human mind cannot do. The human mind can not observe all of the information relevant to an analysis, and the human mind cannot simultaneously attend to all of the information that it can observe.

This first shortcoming has been studied by philosophers, scientists and statisticians. As a result many tools, such as probability theory and statistical inference, exist to address it. While an analyst cannot prophesize about unseen information, they can often make the best predictions possible and quantify exactly how good those predictions are.

The second shortcoming, the inability to attend to many pieces of information at once, has been studied by cognitive scientists, psychologists, and educators. The nature and boundaries of this limitation have been tested and explored. However, data analysts remain unaware that this limitation exists and affects data analyses. Unless our limited attentional resources are well managed, this limitation will reduce our ability to discover complex relationships in a data set. There are ways to manage attentional resources to promote understanding and discovery. These techniques of

attention management can be used to create data analysis tools. This dissertation demonstrates three tools that improve data analysis by invoking three strategies to manage attention.

Embedded plots are a class of graphs that preprocess large, complex data sets into a format that the human mind can readily attend to and explore. Humans are visual thinkers and our working memory can perform feats with visual information that it cannot perform with other types of information. Embedded plots bolster this power by also presenting information in a way that can be explored iteratively, with subparts presented in isolation as well as in a group. At the lowest level, this exploration is further facilitated by relying on the viewers automated expertise at reading graphs.

`lubridate` is a software program that helps analysts process information externally and, thus, avoid the attentional limitations of the working memory. Computers are a widely used tool that can expand an analysts' mental resources, but it is difficult to use computers with date-time data. Date-time data comes in a variety of forms and follows idiosyncratic, sometimes illogical, rules. As a result date time data requires a series of judgements that cannot be easily automated. `lubridate` makes it easier to automate these judgements, which restores the power of computers for processing date-time data outside of the human mind.

Visual Inference Tools (VIT) is a software package that helps develop expertise in statistical inference. Expertise allows analysts to focus their attention efficiently and productively. Introductory statistics classes attempt to build expertise in future generations of data analysts, but there is evidence that students fail to learn the material presented in these classes. Simulations offer a proven way to increase student retention and understanding of inference concepts. Well trained students can later devote more attention to the data they are analyzing and less attention to the methods they are using to analyze it.

6.1 Original contributions

The specific contributions of this thesis may be summarized as follows. This thesis:

1. Identifies and explains what data analysis is: a mental process adapted to the use of quantitative data and constrained by the analyst's inability to mentally process more than a small amount of data at once.
 - (a) The dissertation also provides three lines of support to corroborate this model: argumentation from first principles, testing the structure of data analysis predicted by this model against established descriptions of data analysis, and demonstrating that the model can be used to identify and explain previously unnoticed influences on data analysis.
2. Lends explanatory power to existing descriptions of the data analysis process by linking them to a well studied body of research and theory, cognitive science.
3. Identifies two obstacles to effective analysis that originate in the cognitive nature of data analysis. The first of these is the tendency of the mind to preserve existing schemas in the presence of contradicting information. The second is the inherently abductive nature of sensemaking. This basis in abduction is inherited by data analysis, but goes under-recognized. The thesis illustrates each of these obstacles with a prominent case study.
4. Identifies and defines a class of graphs well suited to exploring complex data, embedded plots. The dissertation demonstrates the usefulness of embedded plots with a case study.
5. Expands the current understanding of the grammar of graphics by demonstrating that graphs are hierarchical, or recursive, in nature. Graphs can be geoms,

and geoms can be graphs. The dissertation also provides methods for extending the concepts of stats and position adjustments.

6. The proposed thesis also applies the above general insights to provide a series of specific contributions to the scientific community. These include:
 - (a) `lubridate`, a software package that enables automated sensemaking methods to be applied to date-time data, which typically resists mathematical manipulation. `lubridate` also popularizes a model of time spans that requires users to develop more precise schemas for thinking about time information. This precision fosters more accurate data analysis results.
 - (b) `Visual Inference Tools (VIT)`, a software package that provides a method for creating interactive animations that visualize aspects of statistical sampling. VIT provides an alternative way to teach statistical inference, which is poorly understood by introductory statistics students as it is traditionally taught. VIT allows teachers to improve their instruction by applying findings from cognitive load theory and multimedia learning theory to their instruction. VIT is part of a larger research agenda which is examining whether programs like VIT can be used to teach statistical inference to younger students. This would have the secondary benefits of creating a more data literate society, attracting more students to the profession of statistics, and better advertising the usefulness of statistics to future employers of statisticians. This last goal is based on the hypothesis that many employers fail to recognize the value of a trained statistician because they did not adequately understand statistics as it was presented in the single introductory class they were required to attend. This model of teaching inference to younger students through visualization has been adopted by New Zealand for its K-13 curriculum. VIT is currently being used to pilot

test visualization based methods for use in this curriculum.

- (c) `ggsubplot`, a software package that facilitates the exploration of large and complex data. `ggsubplot` illustrates how cognitive mechanisms can be directly applied to make useful exploratory data analysis tools. It allows R users to explore complex data with the use of embedded plots, a visualization technique that relies on the visual-spatial sketchpad of the working memory, the isolated elements effect of cognitive load theory, and automated schemas.

6.2 Future Work

This dissertation is directed at a perceived weakness in the literature of statistics. Few studies have examined data analysis as a process. As a result, statisticians have descriptions of data analysis and many tools to use during an analysis, but little theory to guide the data analysis process. My dissertation proposes a foundation for such a theory: data analysis is a cognitive process based on sensemaking. I have corroborated this hypothesis with a number of lines of argument and demonstration. The next step is to test this theory in an experimental setting. Much has been learned about other cognitive tasks, such as learning, through empirical study. However, data analysis has largely escaped experimental scrutiny.

One reason for this may be that experimental study is not a common way for statisticians to examine the way they use statistics. Statisticians often work with scientists to collect and analyze experimental data. However, this research is usually directed by the scientists and applied to other fields than statistics. The type of studies likely to best test a theory of data analysis and to identify better practices of analysis are those commonly conducted by psychologists and educational researchers. As a result, the experimental study of data analysis would likely require, or at least

benefit from, an interdisciplinary effort.

This dissertation provides a theory of data analysis that can be tested immediately, as well as a means of testing it. My model of data analysis predicts that the three tools presented in this dissertation can improve the success of data analysis. An initial test of the model would be to prove whether this is in fact true. For the case of **Visual Inference Tools**, such studies are already underway.

The work presented in this dissertation also supports a second line of future work. The cognitive model of data analysis can be used to improve how statistics is taught to data analysts. Statistical methods are designed to analyze data, and different methods facilitate this process in different ways. The cognitive model of data analysis can scaffold student learning by organizing statistical techniques according to their purpose. A curriculum organized around this model may help students understand how different techniques relate to each other and may promote a “big picture” understanding of data analysis.

An organizing model of statistics emerges when we combine the sensemaking task of data analysis with a consideration of how we attempt to make sense of complex data, see Figure 6.1. We can not attend to every point of data in most data sets at once. As a result, we must attempt sensemaking with one of two strategies. First, we can attempt to mentally consider the entire data set by preprocessing it with visualizations or summaries, the left hand side of Figure 6.1. Although we cannot attend to the data itself, we can attend to the visualizations or summaries. This approach lets us perform sensemaking internally, which may provide benefits that foster discovery and understanding. Second, we can attempt to automate sensemaking and perform it outside of our mind, the right hand side of Figure 6.1. Many statistical algorithms do this by checking for discrepancies between data and a model. But before we can pursue this approach, we must choose a schema to test against the data and

transform the schema into some type of externally manipulatable, quantitative form. This approach allows us to use the data set as it is. The statistical power of this approach is high because we do not have to discard information by summarizing the data. A thorough data analysis should try a multitude of approaches, including techniques that combine methods from the right and left hand sides of Figure 6.1. Statistical methods can be organized based on the role they play in this process, and statistics curriculums can be built around this organization.

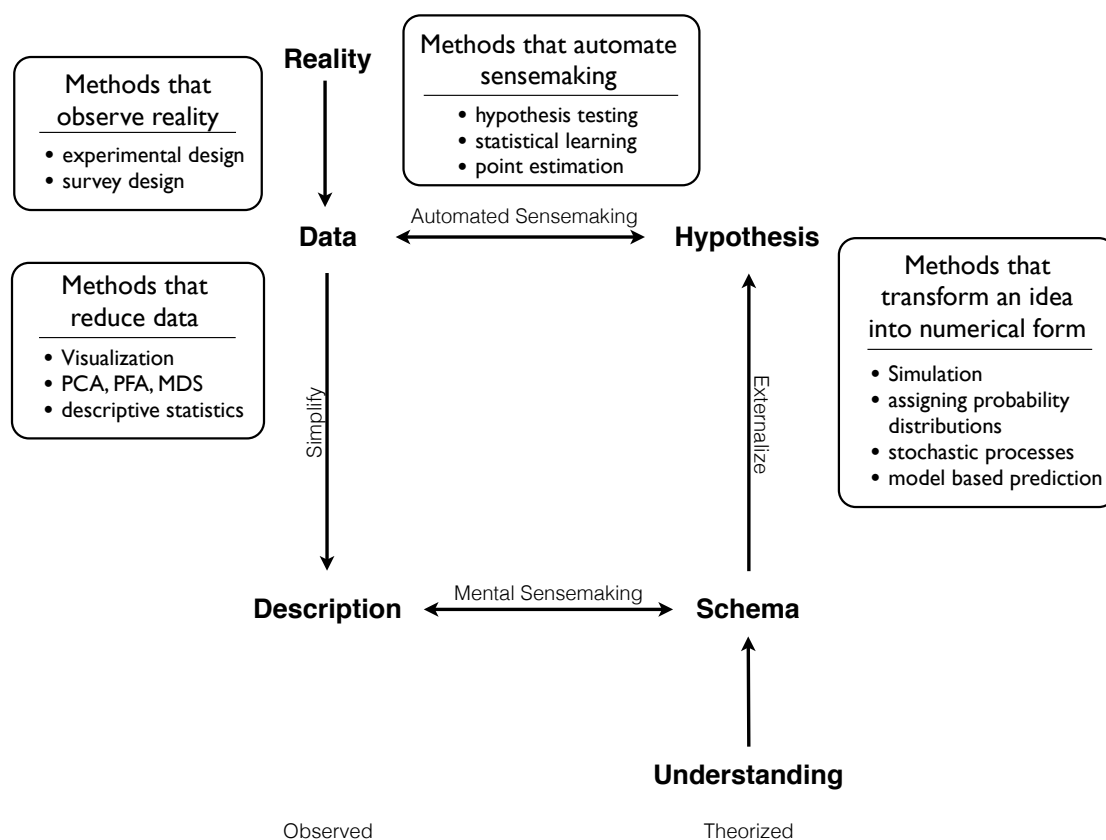


Figure 6.1: A cognitive view of data analysis can help organize a statistics curriculum. Data analysis uses sensemaking to develop mental models that accurately described observed data. Analysts can either simplify data in a way that allows mental sensemaking to occur (left hand side). Or they can transform schemas into hypotheses and attempt automated sensemaking, with a statistical or computerized technique. Statistical methods can be grouped by how they facilitate this process.

6.3 Final thoughts

This dissertation introduces the idea of purposefully managing cognitive resources during a data analysis. The importance of managing these resources will likely increase over the next few decades. Statistics has moved from examining small data problems to examining big data problems. In the small data context, attentional constraints are minimized and knowledge constraints are maximized. But in recent years, the situation has reversed. The large size of data sets gives stable averages and small p-values. But this size also strains our ability to attend to the data we possess. Sometimes even computers cannot manage to process and store the large datasets that are now available; even machines have a finite attention span. The connections that this thesis develops between data analysis and cognitive science can provide a foundation for developing statistical tools that purposefully manage cognitive limitations.

References

- Ainsworth, S. and VanLabeke, N. (2004), “Multiple forms of dynamic representation,” *Learning and Instruction*, 14, 241–255. 5.6
- Anderson, E. (1957), “A semigraphical method for the analysis of complex problems,” *Proceedings of the National Academy of Sciences of the United States of America*, 43, 923. 3.1
- Armstrong, W. (2009), *fts: R Interface to tslib (A Time Series Library in C++)*, r package version 0.7.6. 4.1
- Arnold, P., Education, C., Pfannkuch, M., Wild, C., Regan, M., and Budgett, S. (2011), “Enhancing Students Inferential Reasoning: From Hands-On To Movies,” *Journal of Statistics Education*, 19. 5.2
- Arnold, P. and Pfannkuch, M. (2010), “Enhancing students inferential reasoning: From hands on to movie snapshots,” in *Data and context in statistics education: Towards an evidence-based society. Proceedings of the Eighth International Conference on Teaching Statistics, Ljubljana, Slovenia, July*, Voorburg, The Netherlands: International Statistical Institute. 5.2
- Attfield, S. and Blandford, A. (2009), “Improving the Cost Structure of Sensemaking Tasks: Analysing User Concepts to Inform Information System Design,” *Human-Computer Interaction-INTERACT 2009*, 532–545. 2.3.1
- Aud, S., Hussar, W., Planty, M., Snyder, T., Bianco, K., Fox, M., Frohlich, L., Kemp, J., and Drake, L. (2010), *The condition of education 2010*, Washington, DC: National Center for Education Statistics, Institute of Educational Sciences, U.S. Department of Education, NCES 2010-028 ed. 5.3.3
- Baddeley, A. (2000), “The episodic buffer: a new component of working memory?” *Trends in cognitive sciences*, 4, 417–423. 5.3.1
- (2001), “Is working memory still working?” *American Psychologist*, 56, 851–864. 5.3.1

-
- Baddeley, A. and Hitch, G. (1974), “Working memory,” *The psychology of learning and motivation*, 8, 47–89. 3.3.2, 5.3.1
- Barsalou, L. (1999), “Perceptual symbol systems,” *Behavioral and brain sciences*, 22, 577–660. 5.3.1
- Bartlett, F. (1932), “Remembering: A study in experimental and social psychology.” . 2.3.1
- Bertin, J. (1983), *Semiology of graphics*, Madison, WI: University of Wisconsin Press. 3.4
- Box, G. (1976), “Science and statistics,” *Journal of the American Statistical Association*, 71, 791–799. 2.1, 2.3.1, 2.5
- Bransford, J., Brown, A., and Cocking, R. (2000), *How people learn: Brain, mind, experience, and school*, Washington, DC: National Academies Press. 2.6.1
- Breiman, L. (1985), “Nail finders, edifices, and oz,” in *Berkeley Conference in Honor of Jerzy Neyman and Jack Kiefer*, eds. Le Cam, L. and Olshen, R., Hayward, CA: Institute of Mathematical Sciences, pp. 201–214. 2.2
- (2001), “Statistical modeling: The two Cultures (with comments and a rejoinder by the author),” *Statistical Science*, 16, 199–231. 2.2
- Brooks, L. (1968), “Spatial and verbal components of the act of recall.” *Canadian Journal of Psychology/Revue canadienne de psychologie*, 22, 349. 5.3.1
- Bruner, J. (1961), “The act of discovery.” *Harvard educational review*. 1
- Carley, K. and Palmquist, M. (1992), “Extracting, representing, and analyzing mental models,” *Social Forces*, 70, 601–636. 2.3.1
- Chambers, J. (1983), *Graphical methods for data analysis*, New York, NY: Chapman and Hall. 3.1
- Chance, B., Mas, R., and Garfield, J. (2004), “Reasoning about Sampling Distributions,” *The challenge of developing statistical literacy, reasoning and thinking*, 295–323. 5.2
- Chandler, P. and Sweller, J. (1991), “Cognitive load theory and the format of instruction.” *Cognition and Instruction*, 8, 293–332. 5.6
- Chatfield, C. (1995), *Problem solving: a statistician’s guide*, Chapman and Hall. 2.1, 2.3.1, 2.5, 2.5.1, 2.5.3, 2.5.4
- Chernoff, H. (1973), “The use of faces to represent points in k-dimensional space graphically,” *Journal of the American Statistical Association*, 361–368. 3.1

-
- Cleveland, W. (1994), *Elements of graphing data*, Summit, New Jersey: Hobart Press. (document), 3.1, 3.4
- Cleveland, W. and Terpenning, I. (1982), “Graphical methods for seasonal adjustment,” *Journal of the American Statistical Association*, 52–62. 3.1
- Cobb, G. (2007), “The introductory statistics course: A ptolemaic curriculum?” *Technology Innovations in Statistics Education*, 1. 2.2
- Colebourne, S. and O’Neill, B. (2010), “Joda-Time – Java Date and Time API,” Release 1.6.2. 4.1, 4.5
- Cook, D. and Swayne, D. (2007), *Interactive and dynamic graphics for data analysis with R and GGobi*, Springer Publishing Company, Incorporated. 2.1
- Cowan, N. (2000), “The magical number 4 in short-term memory: A reconsideration of mental storage capacity,” *Behavioral and brain sciences*, 24, 87–114. 1, 1, 2.3.1, 2.3.2, 3.3.2, 5.3.2
- Cox, D. (2001), “Comment on Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author),” *Statistical Science*, 16, 199–231. 2.2
- (2007), “Applied statistics: A review,” *The Annals of Applied Statistics*, 1, 1–16. 2.5, 2.5.3
- Dalal, S., Fowlkes, E., and Hoadley, B. (1989), “Risk analysis of the space shuttle: Pre-Challenger prediction of failure,” *Journal of the American Statistical Association*, 945–957. 2.6.2
- Davis, R. (1998), “What is intelligence? Why?” *AI Magazine*, 19, 91–110. 5.3.1
- DeGroot, A. (1965), “Thought and mind in chess,” *The Hague: Mouton*. 2.6.1
- Dehaene, S. (1997), *The number sense: How the mind creates mathematics*, Oxford University Press. 3.3.2, 5.3.1
- Dehaene, S., Spelke, E., Pinel, P., Stanescu, R., and Tsivkin, S. (1999), “Sources of mathematical thinking: Behavioral and brain-imaging evidence,” *Science*, 284, 970–974. 3.3.2, 3, 5.3.1
- delMas, R., Garfield, J., and Chance, B. (1999), “A model of classroom research in action: Developing simulation activities to improve students statistical reasoning,” *Journal of Statistics Education*, 7. 5.1, 5.6
- Dervin, B. (1998), “Sense-making theory and practice: an overview of user interests in knowledge seeking and use,” *Journal of Knowledge Management*, 2, 36–46. 2.3.1
- Donald, M. (1991), *Origins of the modern mind: Three stages in the evolution of culture and cognition*, Harvard Univ Pr. 2.3.2

-
- Farman, J., Gardiner, B., and Shanklin, J. (1985), “Large losses of total ozone in Antarctica reveal ClOx/NOx interaction,” *Nature*, 315, 207–201. 2.6.1
- Fowler, M. (1997), “Recurring Events for Calendars,” Tech. rep. 4.11
- Freedman, D. (2009), *Statistical Models and Causal Inference: A Dialogue with the Social Sciences*, Cambridge University Press. 2.2
- Friedman, J. (1998), “Data mining and statistics: what’s the connection?” *Computing Science and Statistics: Proceedings of the 29th Symposium on the Interface*. 2.2
- Garland, J. (2011), “Boost.Date_Time – C++ library,” Release 1.46.1. 4.5
- Gelman, A. and Shalizi, C. (2010), “Philosophy and the practice of Bayesian statistics,” *Arxiv preprint arXiv:1006.3868*. 2.2
- Gilovich, T., Vallone, R., and Tversky, A. (1985), “The hot hand in basketball: On the misperception of random sequences,” *Cognitive Psychology*, 17, 295–314. 2.6.1
- Goffman, E. (1974), *Frame analysis: An essay on the organization of experience.*, Harvard University Press. 2.3.1
- Gribov, A., Unwin, A., and Hoffman, H. (2006), “About Glyphs and Small Multiples: Gauguin and the Expo,” *Statistical Computing and Graphics Newsletter*, 17, 14–17. 3.1, 3.4
- Grolemund, G. and Wickham, H. (2011), “Dates and Times Made Easy with lubridate,” *Journal of Statistical Software*, 40, 1–25. 1, 4
- Grothendieck, G. and Petzoldt, T. (2004), “Date and time classes in R,” *R News*, 4, 32. (document), 4.2, 4.3
- Hallman, J. (2010), *tis: Time Indexes and Time Indexed Series*, r package version 1.12. 4.1
- Helsdingen, A. and Van den Bosch, K. (2009), “Learning to make sense,” in *Cognitive Systems with Interactive Sensors conference*. 2.3.1
- Hermann, G. (1969), “Learning by discovery: A critical review of studies,” *The Journal of Experimental Educational*, 58–72. 1
- Hey, T. and Trefethen, A. (2003), “The Data Deluge: An e-Science Perspective,” , 809–824. 2.2
- Hobbs, J., Wickham, H., Hofmann, H., and Cook, D. (2010), “Glaciers melt as mountains warm: A graphical case study,” *Computational Statistics*, 25, 569–586. 3.1
- Huber, P. (1997), *Speculations on the Path of Statistics*, Princeton University Press. 2.2, 2.7

-
- (2011), *Data analysis what can be learned from the past 50 years*, Hoboken, New Jersey: John Wiley & Sons, Inc. 2.2, 2.5, 2.5.4
- James, D. and Hornik, K. (2010), *chron: Chronological Objects which Can Handle Dates and Times*, r package version 2.3-35. S original by David James, R port by Kurt Hornik. 4.1
- Jeung, H., Chandler, P., and Sweller, J. (1997), “The role of visual indicators in dual sensory mode instruction,” *Educational Psychology*, 17, 329–345. 5.3.2
- Jonassen, D. and Henning, P. (1996), “Mental models: Knowledge in the head and knowledge in the world,” in *Proceedings of the 1996 international conference on Learning sciences*, International Society of the Learning Sciences, pp. 433–438. 2.3.1
- Jones, A. (2008), “The Antarctic ozone hole,” *Physics Education*, 43, 358. 2.6.1
- Kalyuga, S., Chandler, P., and Sweller, J. (1999), “Managing split-attention and redundancy in multimedia instruction,” *Applied Cognitive Psychology*, 13, 351–371. 5.3.2
- (2000), “Incorporating learner experience into the design of multimedia instruction.” *Journal of Educational Psychology*, 92, 126. 5.3.2
- Klein, G. and Crandall, B. (1995), “The role of mental simulation in problem solving and decision making,” *Local applications of the ecological approach to human-machine systems*, 2, 324–358. 2.3.1, 2.6.1
- Klein, G., Phillips, J., Rall, E., and Peluso, D. (2003), “A Data Frame Theory of Sense Making,” in *Expertise out of context: proceedings of the sixth International Conference on Naturalistic Decision Making*, pp. 113–155. 2.3.1, 2.6.1
- Kleiner, B. and Hartigan, J. (1981), “Representing points in many dimensions by trees and castles,” *Journal of the American Statistical Association*, 260–269. 3.1
- Kotovsky, J. and Simon, H. (1985), “Why are some problems hard? Evidence from Tower of Hanoi,” *Cognitive Psychology*, 17, 248–294. 3.3.2
- Kuhn, T. S. (1962), *The Structure of Scientific Revolutions*, Chicago, IL: University of Chicago Press. 2.4.1
- Lakoff, G. and Núñez, R. (1997), “The metaphorical structure of mathematics: Sketching out cognitive foundations for a mind-based mathematics,” *Mathematical reasoning: Analogies, metaphors, and images*, 21–89. 2.1
- (2000), *Where mathematics comes from: How the embodied mind brings mathematics into being*, Basic Books. 2.1, 3.3.2, 5.3.1

-
- Lane, D. and Peres, S. (2006), “Interactive simulations in the teaching of statistics: Promise and pitfalls,” in *7th International Conference on the Teaching of Statistics*. 5.1, 5.6
- Lane, D. and Scott, D. (2000), “Simulations, case studies, and an online text: A web-based resource for teaching statistics,” *Metrika*, 51, 67–90. 5.1
- Lane, D. and Tang, Z. (2000), “Effectiveness of simulation training on transfer of statistical concepts,” *Journal of Educational Computing Research*, 22, 383–396. 5.1
- Lanzenberger, M., Miksch, S., and Pohl, M. (2003), “The Stardates-Visualizing highly structured data,” in *Proceedings of the Seventh International Conference on Information Visualization*, IEEE, pp. 47–52. 3.1
- Lavine, M. (1991), “Problems in extrapolation illustrated with space shuttle O-ring data,” *Journal of the American Statistical Association*. 2.6.2
- Lee, H., Plass, J., and Homer, B. (2006), “Optimizing cognitive load for learning from computer-based science simulations.” *Journal of Educational Psychology*, 98, 902. 5.6
- Lipper, M. (2008), “`runt` – Ruby Temporal Expressions,” Release 0.7.0. 4.11
- Luck, S. and Vogel, E. (1997), “The capacity of visual working memory for features and conjunctions,” *Nature*, 390, 279–280. 3.3.2
- Lundberg, C. (2000), “Made sense and remembered sense: Sensemaking through abduction,” *Journal of Economic Psychology*, 21, 691–709. 2.3.1
- Macabebe, E., Culaba, I., and Maquiling, J. (2010), “Pre-conceptions of Newton’s Laws of Motion of Students in Introductory Physics,” in *AIP Conference Proceedings*, vol. 1263, p. 106. 2.6.1
- MacKay, R. and Oldford, R. (2000), “Scientific method, statistical method and the speed of light,” *Statistical Science*, 15, 254–278. 2.5, 2.5.2
- Makar, K. and Rubin, A. (2009), “A framework for thinking about informal statistical inference,” *Statistics Education Research Journal*, 8, 82–105. 5.2
- Mallows, C. (1998), “The Zeroth Problem,” *The American Statistician*, 52, 1–9. 2.2
- (2006), “Tukey’s Paper after 40 years (with discussion),” *Technometrics*, 48, 319–325. 2.2
- Mallows, C. and Walley, P. (1980), “A theory of data analysis?” *Proceedings of the business and economics section, American Statistical Association*. 2.1, 2.2, 2.3
- Mautone, P. and Mayer, R. (2007), “Cognitive aids for guiding graph comprehension.” *Journal of Educational Psychology*, 99, 640. 5.6

-
- Mayer, R. (1989), “Systematic thinking fostered by illustrations in scientific text.” *Journal of Educational Psychology*, 81, 240. 5.3.2
- (2001), *Multimedia learning*, New York, NY: Cambridge University Press, 1st ed. 5.3.2
- (2009), *Multimedia learning*, New York, NY: Cambridge University Press, 2nd ed. 1, 3.1, 5.3.2, 5.6
- Mayer, R. and Anderson, R. (1991), “Animations need narrations: An experimental test of a dual-coding hypothesis.” *Journal of Educational Psychology*, 83, 484. 5.3.2
- (1992), “The instructive animation: Helping students build connections between words and pictures in multimedia learning.” *Journal of Educational Psychology*, 84, 444. 5.3.2
- Mayer, R. and Chandler, P. (2001), “When learning is just a click away: Does simple user interaction foster deeper understanding of multimedia messages?” *Journal of Educational Psychology*, 93, 390. 5.6
- Mayer, R., Dow, G., and Mayer, S. (2003), “Multimedia learning in an interactive self-explaining environment: What works in the design of agent-based microworlds?” *Journal of Educational Psychology*, 95, 806. 5.6
- Mayer, R. and Gallini, J. (1990), “When is an illustration worth ten thousand words?” *Journal of Educational Psychology*, 82, 715. 5.3.2
- Mayer, R. and Moreno, R. (1998), “A split-attention effect in multimedia learning: Evidence for dual processing systems in working memory.” *Journal of Educational Psychology*, 90, 312. 5.3.2
- Miller, G. (1956), “The magical number seven, plus or minus two: Some limits on our capacity for processing information.” *Psychological Review*, 63, 81. 3.3.2, 5.3.2
- (1962), “Some psychological studies of grammar.” *American Psychologist*, 17, 748. 2.6.1
- Mills, J. (2002), “Using computer simulation methods to teach statistics: A review of the literature,” *Journal of Statistics Education*, 10, 1–20. 1
- Minard, C. (1862), *Des Tableaux graphiques et des cartes figuratives, par M. Minard,...*, Paris, France: impr. de Thunot. 3.1
- Minsky, M. (1975), “A framework for the representation of knowledge,” *The psychology of computer vision*, 211–277. 2.3.1
- Molina, M. and Rowland, F. (1974), “Stratospheric sink for chlorofluoromethanes,” *Nature*, 249, 810–812. 2.6.1

-
- Moore, D. (1990), “Uncertainty,” *On the shoulders of giants: New approaches to numeracy*, 95–137. 2.3.1
- Moreno, R. and Mayer, R. (1999), “Cognitive principles of multimedia learning: The role of modality and contiguity.” *Journal of Educational Psychology*, 91, 358. 5.3.2
- (2002), “Learning science in virtual reality multimedia environments: Role of methods and media.” *Journal of Educational Psychology*, 94, 598. 5.3.2
- Mousavi, S., Low, R., and Sweller, J. (1995), “Reducing cognitive load by mixing auditory and visual presentation modes.” *Journal of Educational Psychology*, 87, 319. 5.3.2
- Moyer, R. and Landauer, T. (1967), “Time required for judgements of numerical inequality,” *Nature*. 3.3.2, 5.3.1
- Mulaik, S. (1985), “Exploratory statistics and empiricism,” *Philosophy of Science*, 52, 410–430. 2.3.1
- Neisser, U. (1967), *Cognitive psychology*, Appleton-Century-Crofts New York. 2.3.1
- (1976), *Cognition and reality: Principles and implications of cognitive psychology.*, WH Freeman/Times Books/Henry Holt & Co. 2.3.1, 1
- Norman, D. (1993), *Things that make us smart: Defending human attributes in the age of the machine*, no. 842, Basic Books. 2.3.2
- Paivio, A. (1969), “Mental imagery in associative learning and memory.” *Psychological Review*, 76, 241. 5.3.1
- (1986), *Mental representations: A dual coding approach*, Oxford, England: Oxford University Press, USA. 5.3.1
- (2006), *Mind and its evolution: A dual coding theoretical approach.*, Lawrence Erlbaum Associates Publishers. 5.3.1
- Parker, R. (2010), “Basketball Geek: Advancing our understanding of the game of basketball,” . 4.10
- Pea, R. (1987), “Cognitive technologies for mathematics education,” *Cognitive Science and Mathematics Education*, 89–122. 2.2
- Peirce, C. (1932), *Collected papers of Charles Sanders Peirce*, vol. 1, Belknap Press. 2.6.2
- Peres, S., Lane, D., and Griggs, K. (2010), “Using simulations for active learning: The query-first method in practice,” in *8th International Conference on the Teaching of Statistics*. 5.1, 5.6

-
- Peterson, L. and Peterson, M. (1959), “Short-term retention of individual verbal items.” *Journal of experimental psychology*, 58, 193. 5.3.2
- Pfannkuch, M. (2006), “Comparing box plot distributions: A teacher’s reasoning.” *Statistics Education Research Journal*, 5, 27–45. 5.2
- (2010), “Inferential reasoning: learning to “make a call” in practice,” in *Proc. 8th Int. Conf. Teaching Statistics*. 5.2, 5.3.3
- Piaget, J. and Cook, M. (1952), “The origins of intelligence in children.” . 2.3.1
- Pickett, R. and Grinstein, G. (1988), “Iconographic displays for visualizing multidimensional data,” in *Proc. IEEE Conf. on Systems, Man and Cybernetics, IEEE Press, Piscataway, NJ*, vol. 514, p. 519. 3.1
- Pirolli, P. and Card, S. (2005), “The Sensemaking Process and Leverage Points for Analyst Technology as Identified Through Cognitive Task Analysis,” . 2.3.1, 2.3.1
- Portfolio and Risk Advisory Group, Commerzbank Securities (2009), *its: Irregular Time Series*, r package version 1.1.8. 4.1
- Pratt, D., Johnston-Wilder, P., Ainley, J., and Mason, J. (2008), “Local and global thinking in statistical inference,” *Statistics Education Research Journal*, 7, 107–129. 5.2
- Presidential Commission on the Space Shuttle *Challenger* Accident (1986), *Report of the Presidential Commission on the Space Shuttle Challenger Accident (pbk).*, vol. 1, Washington, DC: Presidential Commission on the Space Shuttle *Challenger* Accident. 2.6.2, 2.6.2
- Qu, Y. and Furnas, G. (2008), “Model-driven formative evaluation of exploratory search: A study under a sensemaking framework,” *Information Processing & Management*, 44, 534–555. 2.3.1
- R Development Core Team (2010), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. 3.4, 4, 4.1, 5.1
- (2012), *Writing R extensions*, <http://cran.r-project.org/doc/manuals/R-exts.pdf>, version 2.15.1 ed. 1
- Reed, S. (2010), *Thinking visually*, New York, NY: Psychology Press. 5.3.1
- Renkl, A. and Atkinson, R. (2003), “Structuring the transition from example study to problem solving in cognitive skill acquisition: A cognitive load perspective,” *Educational Psychologist*, 38. 1

-
- Resnick, L. (1988), “Treating Mathematics as an Ill-Structured Discipline,” Educational Resources Information Center. 2.2
- Revolution Analytics (2012), “What is open source R?” . 5.4
- Rosch, E. (1977), “Classification of real-world objects: Origins and representations in cognition,” in *Thinking: Reading in cognitive science*, eds. Johnson-Laird, P. and Watson, P., Cambridge University Press, pp. 212–222. 2.3.1
- Rosch, E. and Mervis, C. (1975), “Family resemblances: Studies in the internal structure of categories,” *Cognitive Psychology*, 7, 573–605. 2.3.1, 2.4.2
- Rozeboom, W. (1997), “Good science is abductive, not hypothetico-deductive,” in *What if there were no significance tests?*, eds. Harlow, L., Mulaik, S., and Steiger, J., Lawrence Erlbaum Associates, pp. 335–392. 2.6.2
- Rudolph, J. (2003), “Into the big muddy and out again: Error persistence and crisis management in the operating room,” *Unpublished doctoral dissertation, Boston College, Chestnut Hill, Mass., at <http://escholarship.bc.edu/dissertations/AAI3103269>*. 2.3.1
- Rumelhart, D. (1980), “Schemata: The building blocks of cognition,” in *Theoretical issues in reading comprehension: Perspectives from cognitive psychology, linguistics, artificial intelligence, and education*, eds. Spiro, R. J., Bruce, B. C., and Brewer, W. F., Lawrence Erlbaum Associates. 1
- Rumelhart, D. and Ortony, A. (1976), *The representation of knowledge in memory*, Center for Human Information Processing, Dept. of Psychology, University of California, San Diego. 2.3.1
- Russell, D., Stefik, M., Pirolli, P., and Card, S. (1993), “The cost structure of sense-making,” in *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, New York, NY, USA: ACM, pp. 269–276. 2.3.1, 2.3.1
- Ryan, J. and Ulrich, J. (2010), *xts: Extensible Time Series*, r package version 0.7-1. 4.1
- Sarkar, D. (2008), *Lattice: multivariate data visualization with R*, Springer Verlag. 3.1, 3.4
- Schank, R. and Abelson, R. (1977), “Scripts, Plans, Goals, and Understanding:: An Inquiry into Human Knowledge Structures,” . 2.3.1
- Schloerke, B., Crowley, J., Cook, D., Hofmann, H., and Wickham, H. (2011), “GGally: Extension to ggplot2,” <http://cran.r-project.org>. 3.4

-
- Schneider, W. and Shiffrin, R. (1977), “Controlled and automatic human information processing: I. Detection, search, and attention.” *Psychological Review*, 84, 1. 3.3.2
- Shaughnessy, J. (2007), “Research on statistics and learning,” *Second handbook of research on mathematics teaching and learning: A project of the National Council of Teachers of Mathematics*, 957. 5.2, 5.3.3
- Shiffrin, R. and Schneider, W. (1977), “Controlled and automatic human information processing: II. Perceptual learning, automatic attending and a general theory,” *Psychological review*, 84, 127. 3.3.2
- Shneiderman, B. (1996), “The eyes have it: A task by data type taxonomy for information visualizations,” in *Visual Languages, 1996. Proceedings., IEEE Symposium on*, IEEE, pp. 336–343. 3.5
- Smith, E. (1978), “Theories of semantic memory,” *Handbook of learning and cognitive processes*, 6, 1–56. 2.3.1
- Smith, P., Giffin, W., Rockwell, T., and Thomas, M. (1986), “Modeling fault diagnosis as the activation and use of a frame system,” *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 28, 703–716. 2.3.1
- Stanfield, R. and Zwaan, R. (2001), “The effect of implied orientation derived from verbal context on picture recognition,” *Psychological Science*, 12, 153–156. 3.3.2, 5.3.1
- Sweller, J. (1988), “Cognitive load during problem solving: Effects on learning,” *Cognitive science*, 12, 257–285. 3.3.2, 5.3.2
- (1994), “Cognitive load theory, learning difficulty, and instructional design,” *Learning and instruction*, 4, 295–312. 3.1, 3.3.2
- (2003), “Evolution of human cognitive architecture,” *Psychology of Learning and Motivation*, 43, 215–266. 1, 3.3.2
- (2010), “Element interactivity and intrinsic, extraneous, and germane cognitive load,” *Educational Psychology Review*, 22, 123–138. 1
- Sweller, J., Ayers, P., and Kalyuga, S. (2011), *Cognitive load theory*, New York, NY: Springer. 1, 1, 3.3.2, 5.3.2, 5.6
- Sweller, J., van Merriënboer, J., and Paas, F. (1998), “Cognitive Architecture and Instructional Design,” *Educational Psychology Review*, 10, 251–296. 2.3.2
- Tierney, J. (1991), “Behind Monty Hall’s doors: Puzzle, debate and answer,” *New York Times*, 140, 20. 2.6.1

-
- Tindall-Ford, S., Chandler, P., and Sweller, J. (1997), “When two sensory modes are better than one.” *Journal of Experimental Psychology: Applied*, 3, 257. 5.3.2
- Trapletti, A. and Hornik, K. (2009), *tseries: Time Series Analysis and Computational Finance*, r package version 0.10-22. 4.1
- Tufte, E. (1997), *Visual explanations: Images and quantities, evidence and narrative*, Graphics Press, 1st ed. 2.6.2
- Tukey, J. (1960), “Conclusions vs decisions,” *Technometrics*, 2, 423–433. 2.6.1
- (1962), “The Future of Data Analysis,” *The Annals of Mathematical Statistics*, 33, 1–67. 2.1, 2.2, 2.3
- Tukey, J. and Wilk, M. (1966), “Data analysis and statistics: An expository overview,” in *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, ACM, pp. 695–709. 2.1, 2.3, 2.3.1, 2.5
- Tversky, A. and Kahneman, D. (1974), “Judgement under uncertainty,” *Science*, 185, 1124–1131. 2.3.1, 2.6.2
- (1981), “The framing of decisions and the psychology of choice,” *Science*, 211, 453–458. 2.3.1
- Unwin, A. (2001), “Patterns of Data Analysis?” *Journal of the Korean Statistical Society*, 30, 219–230. 2.2
- Vance, A. (2009), “Data analysts captivated by R’s power,” *New York Times*, 6. 5.4
- Varian, H. (2009), “Hal Varian on how the web challenges managers,” *McKinsey Quarterly*, 1. 2.2
- Velleman, P. (1997), *The philosophical past and the digital future of data analysis*, Princeton University Press. 2.2
- Viertl, R. (2002), “On the future of data analysis,” *Austrian Journal of Statistics*, 31, 241–244. 2.2, 2.7
- Vosniadou, S. and Brewer, W. (1989), *The concept of the earth’s shape: A study of conceptual change in childhood*, University of Illinois at Urbana-Champaign Center for the Study of Reading. 2.6.1
- Walsh, V. (2003), “A theory of magnitude: Common cortical metrics of time, space and quantity,” *Trends in Cognitive Sciences*, 7, 483–488. 3.3.2, 5.3.1
- Weick, K. (1995), *Sensemaking in organizations (Foundations for organizational science)*, Sage Publications, Inc. 2.3.1

-
- Weick, K., Sutcliffe, K., and Obstfeld, D. (2005), “Organizing and the Process of Sensemaking,” *Organization Science*, 16, 409–421. 2.3.1
- Weir, C., McManus, I., and Kiely, B. (1990), “Evaluation of the teaching of statistical concepts by interactive experience with Monte Carlo simulations,” *British Journal of Educational Psychology*, 61, 240–247. 5.1
- Wender, K. and Muehlboeck, J. (2003), “Animated diagrams in teaching statistics,” *Behavior Research Methods*, 35, 255–258. 5.1
- Wertheimer, M. (1938), “Laws of organization in perceptual forms,” *A source book of Gestalt psychology*, 71–88. 2.3.1
- Wickham, H. (2009), *ggplot2: Elegant graphics for data analysis*, Springer New York. 3.1, 3.4, 4.10
- (2010), “A layered grammar of graphics,” *Journal of Computational and Graphical Statistics*, 19, 3–28. 3.1, 3.4, 3.4
- (2011), “The split-apply-combine strategy for data analysis,” *Journal of Statistical Software*, 40, 1–29. 3.4.2
- Wickham, H., Hofmann, H., Wickham, C., and Cook, D. (Submitted), “Glyph-maps for Visually Exploring Temporal Patterns in Climate Data and Models,” *Environmetrics*. 3.1
- Wild, C. (1994), “Embracing the “Wider View” of Statistics,” *The American Statistician*, 48, 163–171. 2.1, 2.2
- Wild, C. and Pfannkuch, M. (1999), “Statistical thinking in empirical enquiry,” *International Statistical Review/Revue Internationale de Statistique*, 67, 223–248. 2.1, 2.2, 2.4.2, 2.5, 2.5.2, 2.5.3
- Wild, C., Pfannkuch, M., Regan, M., and Horton, N. (2011), “Towards more accessible conceptions of statistical inference,” *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 174, 247–295. 5.2
- Wild, C., Pfannkuch, M., and Regan, M. and Horton, N. (2010), “Inferential reasoning: Learning to “make a call” in theory,” in *8th International Conference on the Teaching of Statistics*. 5.2
- Wilkinson, L. and Wills, G. (2005), *The grammar of graphics*, Springer Verlag. 3.1, 3.4, 3.4.1.3
- Woodworth, R. (1971), *Experimental psychology.*, Holt, Rinehart and Winston. 2.6.1
- Wu, A., Zhang, X., and Cai, G. (2010), “An interactive sensemaking framework for mobile visual analytics,” in *Proceedings of the 3rd International Symposium on Visual Information Communication*, ACM, p. 22. 2.3.1

-
- Wuertz, D. and Chalabi, Y. (2010), *timeSeries: Rmetrics - Financial Time Series Objects*, r package version 2110.87. 4.1
- Yi, J., Kang, Y., Stasko, J., and Jacko, J. (2008), “Understanding and characterizing insights: how do people gain insights using information visualization?” in *BELIV '08: Proceedings of the 2008 conference on BEyond time and errors: novel evaluation methods for Information Visualization*, New York, NY, USA: ACM, pp. 1–6. 2.3.1
- Zeileis, A. and Grothendieck, G. (2005), “zoo: S3 Infrastructure for Regular and Irregular Time Series,” *Journal of Statistical Software*, 14, 1–27. 4.1
- Zhang, J. (1997), “The nature of external representations in problem solving,” *Cognitive science*, 21, 179–217. 2.3.2
- (2000), “External representations in complex information processing tasks,” *Encyclopedia of library and information science*, 68, 164–180. 2.3.2
- Zhang, P. (2010), “Sensemaking: Conceptual changes, cognitive mechanisms, and structural representations. A qualitative user study,” *Unpublished doctoral dissertation, University of Maryland, at <http://drum.lib.umd.edu/handle/1903/10371>*. 2.3.1
- Ziemer, H. and Lane, D. (2000), “Evaluating the Efficacy of the Rice University Virtual Statistics Lab,” *Poster presented at the 22nd Annual Meeting of the National Institute on the Teaching of Psychology, St. Petersburg Beach, FL*. 5.1
- Zwaan, R. and Yaxley, R. (2003), “Spatial iconicity affects semantic relatedness judgments,” *Psychonomic Bulletin & Review*, 10, 954–958. 3.3.2, 5.3.1